Augmented Reality in Road Navigation

Author:
Doron Halevy

Supervisor:
Gil Shamai

May 8, 2016
Table of Contents

1 Introduction ................................................................................................................. 1

2 Related Work .................................................................................................................. 3
   2.1 Model-Based Localization ......................................................................................... 3
   2.2 Mapping and Tracking .............................................................................................. 4
   2.3 Mapping and Tracking and Model-Based Localization Combination ....................... 5

3 Theoretic Background ..................................................................................................... 7
   3.1 Cameras .................................................................................................................... 7
      3.1.1 Camera Pose ........................................................................................................ 7
      3.1.2 Minimal Pose Representation ............................................................................. 8
      3.1.3 Camera Projection Model .................................................................................... 8
      3.1.4 Projection Function ............................................................................................ 9
      3.1.5 Reprojection Error ............................................................................................. 10
      3.1.6 Back-Projection ............................................................................................... 10
      3.1.7 Camera Calibration ............................................................................................ 11
      3.1.8 Epipolar Geometry ............................................................................................ 11
   3.2 Monocular SLAM ...................................................................................................... 13
      3.2.1 Problem Background ......................................................................................... 13
      3.2.2 Gauge Freedoms and Scale Drift ....................................................................... 13
      3.2.3 Solutions to the monocular SLAM problem ...................................................... 14
   3.3 Monocular VO .......................................................................................................... 15
      3.3.1 Feature-Based Monocular VO ........................................................................... 16
      3.3.2 Feature Extraction and Matching ...................................................................... 18
      3.3.3 Pose Estimation ............................................................................................... 19
      3.3.4 Robust Estimation and Outlier Removal ............................................................ 21
      3.3.5 Triangulation ..................................................................................................... 23
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.3.1</td>
<td>Comparison to Standard VO</td>
<td>44</td>
</tr>
<tr>
<td>8.3.2</td>
<td>Evaluation of Path Correction Procedure</td>
<td>47</td>
</tr>
<tr>
<td>8.3.3</td>
<td>Evaluation of Different Intervals</td>
<td>50</td>
</tr>
<tr>
<td>8.3.4</td>
<td>Video Demo of System’s Operation</td>
<td>56</td>
</tr>
<tr>
<td>9</td>
<td>Usage in Augmented Reality</td>
<td>58</td>
</tr>
<tr>
<td>10</td>
<td>Conclusions</td>
<td>60</td>
</tr>
<tr>
<td>11</td>
<td>Future Work</td>
<td>61</td>
</tr>
<tr>
<td>References</td>
<td>63</td>
<td></td>
</tr>
</tbody>
</table>
## List of Figures

Figure 3.1: Pinhole Projection Model........................................................................................................... 9
Figure 3.2: Epipolar Geometry ......................................................................................................................... 12
Figure 3.3: Monocular Scale Ambiguity........................................................................................................... 16
Figure 3.4: Basic concept of a geometric single-camera VO implementation with 3D-2D correspondences ........................................................................................................................................... 17
Figure 3.5: RANSAC illustration for estimating a line......................................................................................... 22
Figure 3.6: Bundle Adjustment visualization .................................................................................................. 24
Figure 4.1: System Pipeline ............................................................................................................................. 30
Figure 6.1: VO Pipeline (Main Loop) ............................................................................................................... 33
Figure 6.2: VO Pipeline (Initialization) ............................................................................................................. 35
Figure 7.1: Image-based Global Map Registration Pipeline .................................................................................. 36
Figure 7.2: Drift Correction Process ................................................................................................................. 38
Figure 8.1: KITTI Sensor Setup ....................................................................................................................... 42
Figure 8.2: Qualitative comparison against standard VO ............................................................................... 45
Figure 8.3: Quantitative comparison against standard VO .............................................................................. 46
Figure 8.4: Comparison of position error over time with and without path correction .............................. 47
Figure 8.5: Qualitative comparison of estimated trajectories with and without path correction ............ 48
Figure 8.6: Qualitative comparison of estimated trajectories with and without path correction using 100 meter intervals between anchor images ........................................................................................................ 49
Figure 8.7: Qualitative comparison of the trajectories estimated using different intervals between anchor images ........................................................................................................................................ 51
Figure 8.8: Comparison of position error over time for different intervals between anchor images ....... 52
Figure 8.9: Qualitative evaluation of trajectories estimated for different sequences using different intervals – Part 1 ...................................................................................................................................................... 53
Figure 8.10: Qualitative evaluation of trajectories estimated for different sequences using different intervals – Part 2 ...................................................................................................................................................... 54
Figure 8.11: Qualitative evaluation of trajectories estimated for different sequences using different intervals – Part 3 ...................................................................................................................................................... 55
Figure 8.12: Screenshots from the video demo .................................................................................................. 57
Figure 9.1: Screenshots from the video demo of the simple AR application ..................................................... 59
List of Tables

Table 8.1: Average position error per sequence for different intervals between anchor images .................. 56
Abstract

Driving assistance systems for tasks such as aiding in navigation and improving safety are an increasingly important part of modern vehicles. These systems rely more and more on knowledge of the vehicle’s precise localization. Localization estimates for current vehicles come predominantly from automotive GPS, but its coarse positioning and issues such as signal unavailability mean that it cannot be relied on for the accuracy needed for this type of function.

In this work, we propose a vision-based solution for globally localizing vehicles on the road using a single on-board camera and exploiting the availability of priorly geo-tagged street view images from the surrounding environment together with their associated local point clouds. Our approach is focused on the integration of image-based localization into a tracking and mapping system in order to provide accurate and globally-registered 6DoF tracking of the vehicle’s position at all times. The method incrementally tracks the position of the vehicle using mapping and tracking techniques, which inevitably drift over time, and combines the tagged images as a source of accurate global positioning in order to correct the accumulated drift, whenever a good match is detected between the camera image and the tagged street view images. The proposed approach is tested on the public KITTI dataset, which covers realistic driving situations, and show that we are able to achieve lane-level localization in global coordinates. As our results indicate, the solution provides a reliable alternative to GPS systems, which is purely based on vision. We also show how the localization produced by our method can be utilized to provide accurate Augmented Reality overlays for a driver assistance application.
1 Introduction

Precise localization is a pre-requisite in modern vehicles for tasks ranging from driver assistance to autonomous driving. For example, augmented reality apps for road navigation today rely on high precision pose estimation for pixel-accurate graphical overlays. In this respect, lane-level localization with an error of a few meters is the desired goal in most cases. Current localization methods rely mostly on GPS information. However, GPS signals are not always available (e.g. inside tunnels or in urban canyons) and are usually accurate up to a range of several meters. The most reliable solution to the vehicle localization problem thus far has been through the use of 3D Lidar sensors, such as the Velodyne [1], which in addition to being expensive, are active sensors with their own set of challenges. Cameras provide an attractive sensing alternative to 3D Lidars. They have become ubiquitous (found in almost every mobile device) and cheap and their effectiveness to the problem at hand has already been shown in literature.

In this work, we propose a vision-based solution for globally localizing vehicles on the road. Our central idea is to leverage large image databases, such as Google Street View, as abundant sources of accurate geo-tagged street view imagery. We assume each database image also contains a sparse 3D point cloud of its observed environment. To make our approach as general as possible, we only make use of a single on-board camera.

Our work is focused on the integration of image-based localization into a real-time 6DOF tracking and mapping system. Our approach incrementally tracks the position of the vehicle from the camera images using computer vision techniques. These incremental methods inevitably drift over time and are only given in a local reference frame. We utilize the database images as anchors to the ground truth and as a source of global positioning, in order to correct the drift accumulated and to register localization to the global reference frame, whenever a good match is detected between the camera image and the street view images. This integration allows us to obtain an accurate continuous pose estimate of the vehicle at all times, even when the street view images are wide apart and sparsely distributed along roads.

Our approach can be considered as a reliable alternative to GPS systems, which computes accurate positioning from street view images. For this reason, we tested our method on the KITTI visual odometry benchmark [2]. In the experiments, we show that with our technique we are able to obtain a localization error of only a few meters, usually achieving even sub-meter accuracy.

The rest of this work is organized as follows: the following section (2) discusses related work. In section 3 we review the required theoretic background. Section 4 gives a brief overview of the proposed approach. In section 5 we lay out our assumptions about the geo-tagged street view database and sections 6-7 detail the main building blocks of the approach. Section 8 describes the experimental
procedure and demonstrates the experimental results. Section 9 demonstrates the use of our proposed system in a simple AR driver assistance application. We conclude the work in section 10 and in section 11 we suggest possible directions for future work.
2 Related Work

Vision-based methods to localiz
ation can be classified into three main categories based on the amount of prior knowledge we have on the surrounding environment. Assuming that a model of the target scene is known, model-based localization techniques can be used. If no information about the target scene is available, we can use mapping and tracking techniques that simultaneously build the model of the environment and localize within it. If only partial information about the scene is provided, a combination of both techniques can be used.

2.1 Model-Based Localization

One line of research has emerged towards model-based tracking, i.e., tracking directly from a pre-made environment model that is acquired offline. In this approach localization is performed by matching the acquired camera image directly to the scene model and estimating the position of the camera relative to the model. Many scene model representations exist in literature. Authors have looked into localizing images in large scale metrical 3D point cloud maps built from structure-from-motion ([3], [4], [5], [6]). Although localization results are promising, this approach still presents disadvantages. Firstly, it is time consuming and expensive to reconstruct an accurate map. Secondly, a robot (or another vehicle) has to visit the environment beforehand to build the map. Thirdly, matching each query image to the entire point cloud is computationally expensive and does not scale well to large-scale maps. Other authors have looked into scene models that are already available and more light-weight in the form of large geo-tagged image corpora acquired from specially equipped platforms (e.g., Google Street View data [7]). Zamir and Shah [8] build a dense map from 100,000 Google street view images and then localize query images by a GPS-tag-based pruning method. Majdik et al. [9] localize a Micro Aerial Vehicle by matching images acquired from air to Street View images. Both methods only solve a place recognition problem (that is, topological localization). We, on the other hand, compute a full 6DOF metrical localization. Zhang and Kosecka [10] triangulate the position of the query image by matching features with two or more geotagged images from a large database. Their method relies on the density of the tagged database images and will not work for tagged images that are wide apart (as no visual overlap will exist between the query image and at least two tagged images).

All methods mentioned above have the additional limitation that the online localization cannot extend beyond what was captured by the offline model. In our work we apply techniques from image-based localization by utilizing already available geo-tagged street view images from the surrounding environment along with their associated local point clouds. The resulting “map” of the environment is relatively light-weight (not a full reconstruction of the target
environment) and can scale up to large areas easily. The local point clouds are necessary to compute a 6DOF metrical localization. In order to fill-in the trajectory of the vehicle in places were the tagged images do not exist we use mapping and tracking methods, which are outlined next.

2.2 Mapping and Tracking

Another line of research is that of tracking without any prior environment knowledge. A popular method in this category is that of Visual SLAM (Simultaneous Localization and Mapping). Visual SLAM systems use the camera itself to determine device position, by tracking and mapping detectable features in the surrounding environment. They aim at obtaining a globally consistent estimates of the vehicle’s path and the map. Visual SLAM has a long history in computer vision and robotics and we refer the reader to [11] for a relatively recent survey. Davidson et al. [12] were the first to propose monocular SLAM using a filtering approach. Klein and Murray proposed keyframe-based SLAM [13]. In their approach, selected frames (keyframes) are sampled from the camera and processed using optimizations in a background thread to produce a point cloud reconstruction (the map). In parallel, the current camera image is tracked using the map. It was later shown by Strasdat et al. [14] that the keyframes + optimization approach of Klein and Murray [13] – based ultimately on well-known bundle adjustment methods – is strongly advantageous compared to the filtering methods like those of Davidson et al. [12]. For this reason current approaches to monocular SLAM use the former. The most accurate solutions (Lim et al. [15], Engel et al. [16], Mur-Artal et al. [17]) provide high accuracy camera tracking in real-time, and are based on a Visual Odometry (VO) component. Visual Odometry consists in determining simultaneously the camera pose for each video frame and the position of features in 3D world, using only images in an incremental way and in real time. Nister et al. demonstrated the seminal real-time monocular VO system [18] in 2004. While current implementations in monocular VO demonstrate impressive performance [19], their incremental characteristics inevitably lead to large drift at long distances, making the localization result unusable. This is especially true in the monocular scheme, where the absolute scale of the world is not observable, thus leading to rapid drift in scale [20]. We refer the reader to [21] for an in depth tutorial on VO.

The main difference between VO and a complete Visual SLAM system is the “loop closing” capability. By closing loops, localization error caused by drift, especially in a large environment, can be reduced considerably. In the context of SLAM, Loop Closure is the process of detecting an overlap between the current map and a pre-existing map, and then estimating the registration between the two maps. Typically, loop closure is used to detect overlaps within a single SLAM map, for example when the path of the camera crosses over itself. This forces the vehicle to drive in loops, so that the algorithm can recognize the same landmarks and thus compensate for errors in localization. However, driving in loops severely limits the vehicle’s freedom of movement and restricts exploration. In our case, we are interested in
detecting the overlap of the current map with some part of the pre-existing global “map” mentioned in the previous section, so the vehicle is not forced to drive in loops.

Another limitation of pure monocular SLAM is that the camera pose is only given in a local reference system, defined with respect to the first camera frame. Since our global map is given in global coordinates we can anchor tracking to the global reference frame when detection occurs.

In comparison to the pure model-based approaches mentioned in the previous section, the use of a tracking and mapping system allows for continuous 6DOF tracking even in areas not covered by the offline map.

### 2.3 Mapping and Tracking and Model-Based Localization Combination

A few previous works also use some combination of mapping and tracking with global registration using model-based localization approaches. In [22], the authors of [9] advanced their previous topological localization by computing and tracking the position of the flying vehicle in 3D space using cadastral 3D city models. In their work they describe an algorithm to track the position of the flying vehicle using VO and to correct the accumulated drift, whenever a match is detected between the airborne MAV and the street-level images. They do this by back-projecting the geo-referenced images onto the 3D cadastral model of the city to obtain the depth of the scene. Their tracking system only resets the drift when a good match is detected. It does not correct the drifted path between detections to achieve an overall accurate localization, as we do in our method. In addition, their method only works in an urban setting where a 3D cadastral model of the city is available. The authors of [23] use VO to track the position of the camera from a short monocular camera trajectory. They then estimate the 3D positions of the points in the environment based on the camera poses obtained from the odometry estimates using optimizations. Finally, they find Google Street View panoramas that match the images and compute their 6DOF transformation with respect to the camera trajectory and the estimated 3D points. As the GPS coordinates of the panoramic images are known, they obtain estimates of the camera positions relative to the global GPS coordinates. Their method only works offline, their localization results rely on an IMU aided VO which is much more accurate than pure monocular VO (as in our case), and the method only works for short trajectories. Other works that are worth mentioning are [24] that combines monocular SLAM with a pre-made globally registered point cloud reconstruction of the target environment, which was created offline. They achieve a 6DoF tracking and mapping system that provides globally registered tracking in real-time on a mobile device. [25] and [26] combine publicly available road maps (such as OpenStreetMaps) with VO, and are able to significantly reduce the drift compared to standard VO. Finally, [27] uses corners detected on road markings, which were previously tagged with an accurate
GPS, in combination with VO to obtain localization accuracy comparable to SLAM systems, in global coordinates.
3 Theoretic Background

In this section, the theory on which the implemented system relies is presented. Starting with the theory on how the camera maps the environment to an image, continuing with how to incrementally estimate the motion from the images, and ending with how SLAM algorithms can be used to obtain globally consistent estimates of the camera’s trajectory and map. While our work is not focused on building a standard SLAM system, much of the techniques we use in our approach are based or motivated by the components of a complete SLAM pipeline.

3.1 Cameras

A camera maps reflected light from 3D objects in space onto a 2D image. To use the images to build a map of the environment, the way the camera projects the environment to the image must be known. The maps considered here are locations of points on the surface of objects in the world. A mathematical camera model is used to represent the camera projection. In addition, the representation of the camera pose and the geometric relations between two camera views are presented. These are necessary for later sections.

3.1.1 Camera Pose

The pose (or frame) of the camera relative to the world frame is represented as a 3D rigid body transformation, \( T_{c,w} = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} \in SE(3) \). It consists of a rotation \( R \) and a translation \( t \) and allows mapping points from the world reference frame to the camera frame. It is also referred to as a 6DoF (Degrees of Freedom) transformation (since rotation in 3D has 3 degrees of freedom and translation another 3). The position of the camera center is not explicitly represented, but can be recovered as \( c = -R^T t \).

The inverse transformation that maps points from the camera frame to the world frame is satisfied by

\[
T_{w,c} = T_{c,w}^{-1} = \begin{bmatrix} R^T & c \\ 0 & 1 \end{bmatrix}
\]

and the relative pose between two camera frames \( T_{c_1,w} \) and \( T_{c_2,w} \) can be computed with \( T_{c_1,c_2} = T_{c_1,w} \cdot T_{c_2,w}^{-1} = T_{c_1,w} \cdot T_{w,c_2} \), which allows mapping from \( c_2 \) frame to \( c_1 \) frame of reference.

The pose of the camera can also be represented as a matrix \( M = [R \mid t] \), also called the extrinsic camera matrix.
3.1.2 Minimal Pose Representation

Optimizations that include the camera pose parameters, require that the camera parameters be provided in a minimal representation.

The camera pose has 6 degrees of freedom, 3 for translation and 3 for rotation, and so could be represented by a 6-dimensional vector.

There are many ways to represent the rigid body transformation \( T = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} \in SE(3) \) as a 6-dimensional vector.

One way is to use the Lie algebra \( se(3) \) corresponding to the tangent space of \( SE(3) \) at the identity.

Another way is to represent the rotation part as a 3-dimensional vector using either the Euler Vector, by applying the well-known Rodrigues’ formula on the rotation matrix, or using quaternions. In these cases the translation part remains as it is.

3.1.3 Camera Projection Model

Transforming 3D scene points into 2D image plane locations requires a camera model capturing the intrinsic parameters of the imaging device. The most basic model is given with the pinhole camera model: the image is formed by intersection of the light rays from the objects through the center of the lens (projection center), with the focal plane (see Figure 3.1). Let \( \mathbf{X} = (x, y, z)^T \) be a scene point in the camera reference frame and \( \mathbf{p} = (u, v)^T \) its projection on the image plane measured in pixels. The mapping from the 3D world to the 2D image is given by:

\[
\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} \tilde{u} \\ \tilde{v} \end{pmatrix}, \quad \begin{pmatrix} \tilde{u} \\ \tilde{v} \end{pmatrix} = \mathbf{K} \cdot \mathbf{X} = \begin{bmatrix} f_x & s & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix}
\]

Where \( f_x \) and \( f_y \) the focal lengths (in pixels), \( u_0, v_0 \) the image coordinates of the projection center, also called the principal point, and \( s \) is the skew parameter. The skew parameter will be zero for most normal cameras. These parameters are called the intrinsic parameters and \( \mathbf{K} \) is called the camera intrinsic matrix or calibration matrix.
Figure 3.1: Pinhole Projection Model

World point $X$ in the camera reference frame is projected onto the image plane at pixel coordinates

$$p = (u, v)^T.$$ 

$C$ is the camera center and $(u_0, v_0)$ are the coordinates of the principal point.

The pinhole projection is an ideal projection. However, real—especially wide-angle—lenses introduce distortion. This is most visible at the border of the images. In our work, we use a distortion model that accounts for radial and tangential distortion. The distorted image coordinates $p_d = (x_d, y_d)^T$ are expressed as a function of the undistorted image coordinates $p = (x, y)$, the distance to the principal point $r = \|p\|$ and the polynomial coefficients $k_1, \ldots, k_5$:

$$
\begin{pmatrix}
    x_d \\
    y_d
\end{pmatrix} = \begin{pmatrix}
    1 + k_1 r^2 + k_2 r^4 + k_3 r^6 \\
    k_4 (r^2 + 2y^2) + 2k_5 xy
\end{pmatrix}
\begin{pmatrix}
    x \\
    y
\end{pmatrix} + \begin{pmatrix}
    2k_4 xy + k_5 (r^2 + 2x^2) \\
    k_4 (r^2 + 2y^2) + 2k_5 xy
\end{pmatrix}.
$$

Typical distortions are dominated by the coefficients $k_1$ and $k_2$. The above equation can be used to undistort a whole image by sampling in the ideal image domain, computing the distorted coordinates and interpolating the distorted image domain (resampling).

3.1.4 Projection Function

The function $\pi(T, x_w)$ that maps points in the world to the camera image takes as input the camera pose

$$T = \begin{bmatrix} R & t^T \\ 0 & 1 \end{bmatrix}$$

and a point in the world frame $x_w \in \mathbb{R}^3$ and projects the point to the image coordinates $$(u, v)^T$$ through the pinhole camera model. The projection $\pi$ is determined by the intrinsic camera parameters which are known from calibration.
The function first applies the Projection Matrix, which is composed of the camera extrinsic and intrinsic matrices, followed by perspective division:

\[
\begin{bmatrix}
    u \\
    v
\end{bmatrix}
= \begin{bmatrix}
    \tilde{u} \\
    \tilde{v} \\
    \tilde{w}
\end{bmatrix}^T = \begin{bmatrix}
    \tilde{u} \\
    \tilde{v}
\end{bmatrix} = \begin{bmatrix}
    K \cdot [R | t] \\
    1
\end{bmatrix} \cdot \begin{bmatrix}
    x_w \\
    1
\end{bmatrix}
\]

Note that we assume the image is first undistorted before applying the projection function.

### 3.1.5 Reprojection Error

In an ideal situation the projection of a scene point \( \mathbf{x} \in \mathbb{R}^3 \) on the camera image and the corresponding image observation \( \mathbf{u} = (u, v)^T \) would satisfy:

\[
(u, v)^T = \pi(T, \mathbf{x})
\]

But in practice, due to noisy image measurements and due to estimation errors of camera pose and 3D point, the image measurement and the re-projected scene point will generally not exactly coincide.

The reprojection error is defined as the difference between the coordinates of the image measurement and the coordinates of the scene point reprojected onto the image:

\[
\mathbf{r} = \mathbf{u} - \pi(T, \mathbf{x})
\]

### 3.1.6 Back-Projection

The direction \( \mathbf{d} \) of an observed 3D point \( \mathbf{x} \in \mathbb{R}^3 \) with respect to the camera center, corresponding to an image observation \( \mathbf{u} \), can be recovered given the camera calibration:

\[
\mathbf{d} = \begin{bmatrix}
    \hat{u} \\
    1
\end{bmatrix} = \begin{bmatrix}
    K^{-1} \\
    1
\end{bmatrix} \cdot \begin{bmatrix}
    \mathbf{u} \\
    1
\end{bmatrix}
\]

\( \mathbf{d} \) is called the back-projection of \( \mathbf{u} \), or a bearing vector in the direction of \( \mathbf{x} \).

It defines a ray originating from the camera center and intersecting the image plane at the 3D coordinate \( (\mathbf{u}, f) \), where \( f \) is the focal length in metric scale. The scene point (in the ideal situation) lies somewhere along this ray, but the depth cannot be recovered. This is why a calibrated camera is often called a bearing sensor.

\( \hat{\mathbf{u}} \) is referred to as the normalized image coordinates of the image observation, since the effect of the known calibration matrix has been removed (by applying the inverse of \( \mathbf{K} \)), and the ray intersects \( \hat{\mathbf{u}} \) at the imaginary image plane with focal length 1 (or at depth 1 along the ray).
3.1.7 Camera Calibration

The aim of the camera calibration procedure, which is done offline, is to find the distortion coefficients and intrinsic parameters of the camera. The calibration typically requires images of a calibration object from different distances and angles. The most popular method uses a planar checkerboard-like pattern. The positions of the squares on the board are known and it is simple to detect the corresponding corner points in the images and thus construct sets of 2D/3D point correspondences for each image. The calibration requires the extrinsic camera parameters to be estimated, too. The procedure consists of several steps. The intrinsic and all the extrinsic camera parameters are initialized using the DLT algorithm ([28] pp. 88-91, 178-179) by ignoring distortions. This initial solution is then refined by solving a nonlinear least squares problem that minimizes the reprojection error between correspondences.

3.1.8 Epipolar Geometry

The epipolar geometry is the intrinsic projective geometry between two views (or cameras), viewing the same 3D scene. It is independent of the scene structure, and only depends on the internal and external parameters of the two cameras. A world point, \( \mathbf{X} \), is projected onto the image plane of two views at \( \mathbf{x} \) and \( \mathbf{x'} \). The two camera centers, \( \mathbf{C} \) and \( \mathbf{C'} \), the world point and the projected points will be coplanar, as can be seen in Figure 3.2. Let us call this plane, \( \pi \), the epipolar plane. Knowing that the projected points have to be on the epipolar plane, the search for a matching point for \( \mathbf{x} \) in the second view is limited to the line where the image plane intersects the epipolar plane, the epipolar line \( l' \), as is shown in Figure 3.2. This geometric constraint is called the epipolar constraint. The epipoles, \( \mathbf{e} \) and \( \mathbf{e'} \), are the points where the baseline, the line between the two camera centers, intersects each image plane.
Figure 3.2: Epipolar Geometry

The left figure is showing the two camera view centers, \( C \) and \( C' \), the world point, \( X \), the projected image points, \( x \) and \( x' \), and the epipolar plane \( \pi \). The right figure shows the epipoles, \( e \) and \( e' \), the epipolar line of the second view, \( l' \), and how all world points in the direction of \( X \) must be projected onto this line.

The Fundamental Matrix, \( F \), is a central part of epipolar geometry. The Fundamental Matrix projects image points in one view to their corresponding epipolar lines in the second view and relates the two views as \( x'Fx = 0 \), where \( x \) and \( x' \) is the projection (in homogeneous image coordinates) of a world point, \( X \), in the first and second view, respectively. The Fundamental Matrix is independent of scene structure. However, it can be computed from correspondences of imaged scene points alone, without requiring knowledge of the cameras’ internal parameters or relative pose. This can be done using the normalized 8-point algorithm ([28] pp. 279-282). In our work we will use the epipolar constraint through the Fundamental Matrix, to constrain matching image points in two views.

The Essential Matrix, \( E \), is the specialization of the fundamental matrix to the case of normalized image coordinates, that is, to the case of calibrated cameras. It fulfills the epipolar constraints \( x'Ex = 0 \) in terms of the normalized image coordinates of the corresponding image points \( x \) and \( x' \). The relation between the Fundamental Matrix and the corresponding Essential Matrix is: \( E = K'^T FK \) where \( K \) and \( K' \) are the calibration matrices of the corresponding cameras. \( E \) contains the camera motion parameters, up to an unknown scale factor for the translation, between the two views. It can be computed from correspondences of imaged scene points in normalized image coordinates and thus calibration of both cameras is required. In section 3.3.3 we will explain how to compute \( E \) and decompose it into the rotation and translation components. More information about epipolar geometry and multi-view geometry can be found in [28].
3.2 Monocular SLAM

Although this work is not focused on another approach to classical monocular SLAM, it is worth mentioning the research undertaken by monocular simultaneous localization and mapping, as much of our work is based or inspired by it.

3.2.1 Problem Background

SLAM (simultaneous localization and mapping) is the problem of estimating the motion of a moving robot in real-time as it continuously observes and maps its unknown environment using different sensors. Visual SLAM systems use cameras as the only sensor to determine the robot’s motion, by tracking and mapping detectable features in the surrounding environment.

The visual SLAM methods are classified into two main categories by the number of cameras employed: monocular and stereo. The monocular systems have several advantages over stereo systems in terms of cost, flexibility, and computational efficiency. A single camera always costs less than stereo camera systems, and also provides flexibility in installation of the camera to robots. For example, a stereo camera should have more than a half meter baseline for enough disparity when it is operated in a car for outdoor navigation. However, robots like micro aerial vehicles (MAVs) may not have the space for a wide baseline stereo camera at all, and when the distance to the scene is much larger than the stereo baseline (i.e., the distance between the two cameras), the stereo scheme degenerates to the monocular case.

Despite its advantages, it has proven more difficult to achieve real-time large-scale mapping with a monocular camera, due to its nature as a purely projective sensor (a bearing sensor). Geometry does not just ‘pop out’ of the data from a moving camera, but must be inferred over time from multiple images. In addition, due to the purely projective nature and without the known inter-camera distance of a stereo rig to serve as an anchor, the motion estimates and map structure can only be recovered up to scale. The fact that a single camera does not measure metric scale means that the scale of locally constructed map portions and the corresponding motion estimates is therefore liable to drift over time.

3.2.2 Gauge Freedoms and Scale Drift

Metric SLAM systems aim to build coherent maps, in a single coordinate frame, of the areas that a robot moves through. But they must normally do this based on purely relative measurements of the locations of scene entities observable by their on-board sensors. There will therefore always be certain degrees of gauge freedom in the maps that they create, even when the best possible job is done of estimation. These gauge freedoms are degrees of transformation freedom through which the whole map, consisting of feature and robot position estimates taken together, can be transformed without affecting the values of the sensor measurements. In SLAM by a robot moving in 3D and equipped with a sensor like calibrated
stereo vision or a 3D laser range-finder, there are six degrees of gauge freedom, since the whole map could experience a rigid body transformation in 3D space. In monocular SLAM, however, there are fundamentally seven degrees of gauge freedom, since the overall scale of the map, as well as a 6DoF rigid transformation, is undetermined (scale and a rigid transformation taken together are often known as a similarity transformation).

It is the number of gauge degrees of freedom in a particular type of SLAM which therefore determines the ways in which drift will inevitably occur between different fragments of a map. So maps built by a monocular camera with no additional information drift in seven degrees of freedom (rotation, translation and scale).

3.2.3 Solutions to the monocular SLAM problem

Monocular SLAM was initially solved by filtering ([12], [29], [30]). In that approach every frame is processed by the filter to jointly estimate the map feature locations and the camera pose. It has the drawbacks of wasting computation in processing consecutive frames with little new information and the accumulation of linearization errors. On the other hand keyframe-based approaches ([13], [20], [17], [15], [16]) estimate the map using only selected frames (keyframes) allowing to perform more costly but accurate bundle adjustment optimizations, as mapping is not tied to frame-rate. Strasdat et al. [14] demonstrated that keyframe-based techniques are more accurate than filtering for the same computational cost. In our work, we have therefore resolved to take a keyframe optimization approach.

Almost all recent approaches to keyframe-based SLAM consist in two main modules:
1) A Visual Odometry (VO) approach which consists in determining simultaneously the camera pose for each video frame and the position of features in 3D world, using only images in an incremental way and in real-time.
2) A loop closure module which prevents drift to achieve global consistency of the map and path.
Both modules are described in the next sections.
3.3 Monocular VO

Monocular Visual Odometry (VO) is the process of estimating the ego-motion of an agent (e.g., vehicle, human, and robot) using only the input of a single camera. VO operates by incrementally estimating the pose of the vehicle through examination of the changes that motion induces on the images of its onboard cameras.

Approaches to monocular VO can be divided into three categories: feature-based methods, direct methods, and hybrid methods. Feature-based methods are based on salient and repeatable features that are tracked over the frames. In this method, a set of feature observations is extracted from the image and the camera position and scene geometry is computed as a function of these feature observations only. Direct methods estimate pose directly on the intensities of all the pixels in the image or sub-regions of it, and enable the possibility of using all information in the image. By this, direct methods circumvent the limitation of feature based methods, that only information that conforms to the feature type can be used (for example, information contained in straight or curved edges will be ignored by corner or blob detectors). Direct methods have higher accuracy and robustness, compared to feature based methods, in particular in areas with few feature points, and in addition provide more information about the geometry of the environment. Their main limitation is that that they are computationally demanding. Hybrid methods use a combination of the previous two.

In the first category are the VO solutions described in the recent SLAM works of [15] and [17]. The first real-time, large-scale VO with a single camera, presented by Nister et al. [18], was also feature-based. Among the direct methods is the VO solution in the SLAM work of [31] that uses all pixels in the image to estimate pose. This solution is computationally demanding and requires a high-end GPU to run in real-time. In [32], the authors proposed a semi-dense solution for direct VO, which significantly reduces the computational complexity, compared to previous direct VO works. The approach of the method is to spend computations where the information gain is maximized. This is done by calculating a semi-dense inverse depth map only for the regions of the image with non-negligible gradient. This solution for VO was later incorporated in a full solution for the SLAM problem by [16]. Their results are very impressive as the system is able to operate in real-time, without GPU acceleration, building a semi-dense reconstruction of the surrounding environment, with more potential applications for robotics than the sparse output generated by feature-based SLAM. Nevertheless, their localization accuracy is lower than some feature-based SLAM works, such as [17]. In a halfway between direct and feature-based methods is the semi-direct VO work of [19], which is able to operate at high frame-rates obtaining impressive results in quadcopters with a downward looking camera.
In our work, we use the feature-based approach which is most common in literature, and is described next.

### 3.3.1 Feature-Based Monocular VO

As already mentioned in 3.2.1 (in the context of SLAM), one important fact about single camera VO algorithms is that they cannot recover metric scale. As an intuitive example, consider the observation of an object as indicated in Figure 3.3. Only the ratio between the distance to the object and its size is known – the true metric size of the object is undetermined. The ratio between the scale in the VO algorithm and the metric scale is denoted by the term \textit{visual scale factor}.

![Figure 3.3: Monocular Scale Ambiguity](image)

In a monocular VO context, only the ratio between the distance to the structure and its size is known. The true scale remains unknown.

Feature-based (or geometric) monocular VO frameworks can be grouped into two fundamentally different classes:

- **Following** the paradigm of Nister et al. [18], the first class covers solutions that use 2D-2D image correspondences in order to incrementally estimate the relative transformation between successive camera frames. Similar to the visual scale factor issue mentioned beforehand, these algorithms are not able to recover the magnitude of the translation. Therefore, two-view point triangulations and distance ratios over two pairs of frames still have to be recovered in order to ensure at least correct visual scale factor propagation.

- **The second class** groups solutions that are also deriving local 3D structure. They use 3D-2D correspondences in order to derive incremental relative displacements from consecutive absolute poses with respect to local structure. In this way, the visual scale factor is implicitly propagated. It has the advantage of continuously delivering camera displacement information, even if the relative displacements become too small for a safe direct derivation of relative frame-to-frame transformations. However, relative-pose computation is still needed for initialization.
The initialization step involves the determination of the relative pose between the first two keyframes and the triangulation of an initial point cloud. Subsequent steps then require only 3D-2D correspondences in order to derive absolute pose w.r.t. the current point cloud. New points are added each time the relative parallax surpasses a given threshold.

In this work, we use the latter approach only. The basic sequence of processing steps is indicated in Figure 3.4. The initialization procedure consists of waiting until the current camera frame shows enough disparity with respect to the initial frame, and then deriving the relative position and orientation between the current and initial frame in order to triangulate a first point cloud. The subsequent steps then involve the computation of the absolute camera pose, meaning the relative pose with respect to the current point cloud. New points are added each time the parallax (a measure of the camera displacement relative to the distance to the scene) surpasses a given threshold, and the frames from which we triangulate new features are called keyframes.

From this basic concept, we can derive the basic modules for the VO pipeline presented in this work.
• Extraction of salient points from an image.
• Matching of salient points between two images in order to establish correspondences.
• Robust computation of relative or absolute pose.
• Triangulation of 3D points.
• Joint nonlinear optimization of multiple camera poses and 3D points.

The remainder of this section focuses on a brief introduction to these modules.

3.3.2 Feature Extraction and Matching

In our context, 3D points originate from triangulating sparse correspondences of salient points in image space called features. Thus for subsequent steps, features in the image planes of successive images, that are projections of the same point on a physical object, must be identified.

There are two different, generally used approaches to find image features corresponding to the same world point. The first one is to find features in one image and track them in the following images using local search techniques, such as correlation. Often the Kanade-Lucas-Tomasi (KLT) [33] feature tracker is used. The second one is to independently detect features in all the images and match them based on some similarity metric between their descriptors. The former approach is more suitable when the images are taken from nearby viewpoints, whereas the latter more suitable when a large motion or viewpoint change is expected. Early research in VO is opted for the former approach while the works in the last decade concentrated on the latter approach. The reason is that early works were conceived for small-scale environments, where images were taken from nearby viewpoints, while in the last few decades, the focus has shifted to large-scale environments, and so the images are taken as far apart as possible from each to limit the motion-drift-related issues. For this reason, in our work we use the latter approach.

Feature Detection

During the feature detection step, the image is searched for salient keypoints that are likely to match well in other images. A local feature is an image pattern that differs from its immediate neighborhood in terms of intensity, color, and texture. Common feature detectors include corners or blob detectors, because their position in the image can be measured accurately. The appealing properties that good feature detector should have are: localization accuracy, repeatability, computational efficiency, robustness to noise, distinctiveness (so that features can be accurately matched across different images), and invariance to both photometric (e.g., illumination) and geometric changes (rotation, scale, viewpoint).

A huge variety of feature detectors has been proposed over the years. Commonly used corner detectors are Harris [34], FAST [35], Shi-Tomasi [33] and blob detectors are SIFT [36] and SURF [37].

Feature detectors usually work in two steps. The first step is to apply a feature response function on the entire image. For instance, Harris uses the corner response function and SIFT uses the Difference-Of-
Gaussians (DoG) operator. The second step is to localize all local extrema points on the output of the first step, by applying non-maxima suppression. To achieve invariance to scale, the feature detector is often applied to images of different scale.

**Feature Description**
For each detected feature point a compact descriptor based on the region around each point is calculated. The simplest feature descriptor is a descriptor of the appearance of the point, i.e. the intensity of each pixel in a region around the feature point. Then the sum of squared differences or the normalized cross correlation can be used to compare descriptors. However, these descriptors are not very robust to changes in orientation, scale and viewpoint. More elaborate descriptors are the SIFT descriptor [36], the SURF descriptor [37], BRIEF [38], ORB [39] and BRISK [40]. In common for these descriptors is that they generate a vector, usually 64 or 128 elements long. SIFT and SURF produce vectors of real numbers, while the others produce binary vectors.

**Feature Matching**
The set of features from the two images can be exhaustively matched, using a similarity measure on the feature descriptors. For SIFT and SURF the Euclidean distance can be used, and for binary descriptors the Hamming distance can be used. The complexity of exhaustive matching is quadratic in the number of features, and becomes impractical when the number of features becomes large. Binary descriptors outperform floating point descriptors in speed of matching, as the Hamming distance of a binary vector can be calculated extremely fast on a modern CPU in the form of a bitwise XOR operation. The matching process can also be speeded up by using an indexing structure, such as a multidimensional search tree or a hash table, to rapidly search for features near a given feature.

### 3.3.3 Pose Estimation
Motion estimation is the core computation step performed for every image in a VO system. More precisely, in the motion estimation step, the camera motion between the current image and the previous image is computed. By concatenation of all these single movements, the full trajectory of the camera and the agent (assuming that the camera is rigidly mounted) can be recovered.

This section explains how the transformation $T$ between two images $I_1$ and $I_2$ can be computed from two sets of corresponding features $f_1, f_2$. Depending on whether the feature correspondences are specified in two or three dimensions, there are three different methods.

- **2D to 2D**: In this case, both $f_1$ and $f_2$ are specified in 2D image coordinates and correspond to projections of the same 3D scene point observed in both images. Also known as relative pose estimation.
• 3D to 3D: In this case, both \( f_1 \) and \( f_2 \) are specified in 3D. This case is not used in our work and will not be discussed further (more suitable for stereo VO).

• 3D to 2D: In this case, \( f_1 \) are specified in 3D and correspond to 3D scene points observed by image \( I_1 \), and \( f_2 \) are their corresponding 2D observations in image \( I_2 \). Also known as absolute pose estimation.

2D-2D Relative Pose Estimation

The relative camera pose computation aims at recovering the relative translation and rotation between two image frames observing a common set of unknown 3D world points – only the 2D measurements in both frames are given.

The geometric relations between two images \( I_1 \) and \( I_2 \) of a calibrated camera are described by the so-called Essential Matrix, \( E \), presented in section 3.1.8. \( E \) contains the camera motion parameters up to an unknown scale factor for the translation. The Essential Matrix can be computed from 2D to 2D feature correspondences using the epipolar constraints, and rotation and translation can directly be extracted from \( E \). The minimal case solution involves five 2D to 2D correspondences and an efficient implementation was proposed by Nister in [41]. Nister’s five-point algorithm has become the standard for 2D to 2D motion estimation in the presence of outliers. A simple and straightforward solution for \( n \geq 8 \) non-coplanar points is the Longuet-Higgins’ eight-point algorithm [42]. The solution of the eight-point algorithm is degenerate when the 3D points corresponding to the 2D matches are coplanar. Conversely, the five-point algorithm works also for coplanar points. The eight-point algorithm works for both calibrated and uncalibrated cameras, whereas the five-point algorithm assumes the camera is calibrated.

From the estimate of \( E \), the rotation and translation parts can be extracted. In general, there are four different solutions for \( R \) and \( t \) for one Essential Matrix. However, by triangulation of a single point, the correct \( R, t \) pair can be identified by choosing the solution where the point is in front of both cameras.

3D to 2D Pose Estimation

The absolute camera pose problem consists of retrieving the absolute position and orientation of a camera from known correspondences between 3D world points and their 2D image observations.

The transformation \( T \) is computed from the 3D to 2D correspondences \( X_i \) and \( p_i \). The general formulation in this case is to find \( T \) that minimizes the image reprojection error:

\[
\arg\min_T \sum_i \|p_i - \pi(T, X_i)\|^2.
\]

This problem is known as perspective from \( n \) points (PnP) (or resection), and there are many different solutions to it in the literature. The minimal case involves three 3D to 2D correspondences. This is called
**perspective from three points** (P3P) and returns four solutions that can be disambiguated using one or more additional points. In the 3D to 2D case, P3P is the standard method for robust motion estimation in the presence of outliers.

A simple and straightforward solution to the PnP problem for \( n \geq 6 \) points is the DLT (direct linear transformation) algorithm ([28] pp. 88-91, 178-179). The DLT algorithm, however, is over-parameterized for the calibrated case (it also computes the intrinsic camera parameters, which is redundant in this case).

To use this method in the monocular case, it is necessary to triangulate 3D points and estimate the pose from 3D-2D matches in an alternating fashion. This alternating scheme is often referred to as SFM (structure from motion). Starting from two views, the initial set of 3D points and the first transformation are computed from 2D-2D feature matches. Subsequent transformations are then computed from 3D-2D feature matches. To do this, features need to be matched (or tracked) over multiple frames (at least three). New 3D features are again triangulated when a new transformation is computed and added to the set of 3D features. The main challenge of this method is to maintain a consistent and accurate set of triangulated 3D features and to create 3D-2D feature matches for at least three adjacent frames.

### 3.3.4 Robust Estimation and Outlier Removal

Point correspondences are usually contaminated by outliers, that is, wrong data associations. Possible causes of outliers are image noise, occlusions, blur, and changes in viewpoint and illumination. Outliers will bias the result if included in our model estimations (such as motion estimation) and cause irrecoverable errors for camera pose and map estimation. For estimating accurately models that rely on point correspondences, it is important that outliers be removed.

**RANSAC**

Robust estimators are commonly used to estimate model parameters from data containing atypical values. RANSAC (Random Sample Consensus) [43] estimates a global relationship adapting data, and at the same time classifies data under inliers (data which is consistent with the relationship) and outliers (not consistent with the relationship). RANSAC has been established as the standard method for model estimation in the presence of outliers. The idea behind RANSAC is to estimate a number of hypothesis models by repeatedly sampling a randomly selected minimum set of data points and count the total number of other data points in consensus with the estimated hypothesis. The hypothesis generating the highest number of data points in consensus with it will be selected as a solution and the corresponding data points (in consensus with the hypothesis) will be selected as inliers. As an example, for two-view motion estimation as used in VO, the estimated model is the relative motion \((\mathbf{R}, \mathbf{t})\) between the two camera positions, and the data points are the candidate (or putative) feature correspondences. An example
of how the RANSAC sampling process, to estimate a line in a plane, could look like is illustrated in Figure 3.5.

![RANSAC Illustration](image)

**Figure 3.5: RANSAC illustration for estimating a line**

The figures show two randomly selected points in blue, rejected outliers in red and the inliers in green for four different hypotheses. Here the hypothesis in the bottom right image will be selected, as it is supported by more inliers than the other hypotheses.

As observed, RANSAC is a probabilistic method and is nondeterministic in that it exhibits a different solution on different runs. However, the solution tends to be stable when the number of iterations grows.
The number of subsets (iterations) $N$ that is necessary to guarantee that a correct solution is found can be computed by

$$N = \frac{\log (1 - P)}{\log (1 - (1 - \varepsilon)^s)} ,$$

where $s$ is the number of correspondences from which the model can be instantiated, $\varepsilon$ is the percentage of outliers in the correspondences, and $P$ is the requested probability of success. As can be seen from the above equation, $N$ is exponential in the number of correspondences $s$ necessary to estimate the model. Therefore, there is a high interest in using a minimal parametrization of the model and so we would prefer using the 5-point algorithm ($s = 5$) over the 8-point algorithm ($s = 8$) for relative pose estimation. Similarly, we would prefer using the 3-point algorithm ($s = 3$) over the DLT algorithm ($s = 6$) for absolute pose estimation. This can also be viewed as another advantage of the 3D-2D method over the 2D-2D method (mentioned in section 3.3.1) for estimating motion. As mentioned previously, the 2D-2D case requires a minimum of five-point correspondences (using the 5-point algorithm). However, only three correspondences are necessary in the 3D-2D motion case (using 3-point algorithm). This lower number of points results in a much faster motion estimation.

### 3.3.5 Triangulation

Some of the steps mentioned earlier require triangulation of 3D points (structure) from their 2D image observations in two (or more) images. Structure computation is also needed by bundle adjustment, which will be discussed later, to compute a more accurate estimate of the trajectory. Triangulated 3D points are determined by intersecting back-projected rays from their 2D image correspondences of at least two image frames. In order to back-project rays in a common coordinate system, the cameras must be calibrated and their poses known. In perfect conditions, these rays would intersect in a single 3D point. However, because of image noise, camera model and calibration errors, and feature matching uncertainty, they never intersect. Therefore, the point at a minimal distance, in the least-squares sense, from all intersecting rays can be taken as an estimate of the 3D point position.

Triangulation is usually carried out using a linear least squares method followed by a non-linear refinement step. Linear triangulation methods are described in detail in ([28] pp. 312-313). Notice that the standard deviation of the distances of the triangulated 3D point from all rays gives an idea of the quality of the 3D point. Three-dimensional points with large uncertainty will be thrown out. This happens especially when frames are taken at very nearby intervals compared with the distance to the scene points. When this occurs, 3D points exhibit very large uncertainty. One way to avoid this consists of skipping frames until the average uncertainty of the 3D points decreases below a certain threshold. Frame selection is a very important step in VO and even more so in SLAM and will be described in section 3.5.
3.3.6 Bundle Adjustment

Assume a scene represented by 3D points and a set of cameras, each viewing some part of the scene. Bundle Adjustment (BA) ([28] pp. 434-435) aims to jointly optimize motion (camera poses) and structure (3D points) given image observations of the scene points onto the cameras (see figure ?).

![Bundle Adjustment visualization](image)

**Figure 3.6: Bundle Adjustment visualization**
Cameras and 3D scene points are explicitly linked through the respective image observations. Cameras are implicitly linked to other cameras through observations of the same scene points.

In more detail:

Consider a set of camera poses, \( T_i \), and a set of 3D scene points, \( X_j \), and the corresponding image observations, \( x_{i,j} \) (of point \( X_j \) observed in the image of camera \( T_i \)). Due to noisy image measurements, \( x_{i,j} \), the image coordinates of the 3D point reprojected onto the image, \( \pi_i(T_i, X_j) \), generally will not exactly coincide with the measurement. Therefore, the objective is to find estimates of the camera poses and 3D point positions, \( \hat{T}_i \) and \( \hat{X}_j \), respectively, that minimize the total reprojection error. Assuming Gaussian image noise on the measurements, the MAP (maximum a posteriori) estimates of the parameters can be found by minimizing the weighted sum of the squared reprojection errors:

\[
\left\{ \hat{T}_2, \ldots, \hat{T}_M \right\}, \left\{ \hat{X}_1, \ldots, \hat{X}_N \right\} = \arg \min_{\{T_i\}, \{X\}} \sum_{i,j \in C} r_{i,j}^T \Lambda_{i,j} r_{i,j}
\]

Where \( r_{i,j} = x_{i,j} - \pi_i(T_i, X_j) \) is the error term, \( \Lambda_{i,j} = \Sigma_{i,j}^{-1} \) is the information matrix for measurement \( x_{i,j} \) which indicates how accurate the measurement is, and \( C \) is the set of pairs \( i, j \), for which point \( X_j \)
is visible from pose $T_j$, i.e., $x_{i,j}$ exists. $T_j$ is usually held fixed and not optimized in order to anchor the optimization to the reference frame of the first camera (that usually also represents the world frame). This minimization problem can be carried out using non-linear iterative optimization techniques such as Gauss-Newton ([28] pp. 597-600) or Levenberg-Marquardt ([28] pp. 600-613).

The sparseness pattern in BA, that there are only links between points and cameras, but no point-point or camera-camera constraints and that not every point is visible in every camera, can be exploited in the optimization to significantly reduce complexity. The optimization requires a good initial estimate of the parameters in order to converge to the global minimum. It also requires that a minimal representation for the parameters is provided.

**Local BA**

In our context, our employed VO concept depends on the feasibility of robustly computing absolute pose w.r.t local 3D structure. This in turn, requires the availability of accurate 3D point coordinates. 3D points are – in the easiest case – triangulated and optimized from two views only. In practice, however, a point is typically observed by more than two frames, and tracked over a sequence of multiple camera frames. An optimal result for the structure is thus achieved by considering all feature measurements for each point, which finally constitutes a full BA optimization. Full bundle adjustment as described above adjusts the pose for all frames (apart from the first, which is a fixed datum) and all map point positions. It exploits the sparseness inherent in the structure-from-motion problem to reduce the complexity of cubic-cost matrix factorization from $O((N+M)^3)$ to $O(N^3)$ with $N$ and $M$ being the number of frames and points, respectively, and so the system ultimately scales with the cube of frames. One way or the other, it becomes an increasingly expensive computation as map size increases: For example, tens of seconds are required for a map with more than 150 frames to converge.

A way of reaping the benefits of the optimality of BA, while maintaining constant complexity, making it possible to use BA for VO, is to only consider a so-called window of the $n$ last image frames and then perform a parameter optimization of camera poses and 3D landmarks for this set of image frames only. This is called local BA (first proposed by the VO work of [44]). The idea is to reduce the number of calculated parameters in optimizing only the parameters of the $n$ last cameras and taking account of the 2D projections of the points they observe in the $N$ (with $N > n$) last frames. Thus, it optimizes only the last $n$ frames and all the map points seen by those frames. The earlier $N - n$ frames that see those points are included in the optimization but remain fixed. As a sliding window BA is applied, the optimization window has to be anchored to the previous map (BA is invariant to reference frame and scale if unconstrained). Thus, at least 7DoF (rotation, translation and scale) should be fixed and we must have $N \geq n + 2$ to fix the reconstruction frame and the scale factor at the sequence end.

25
Local BA reduces the drift compared to two-view VO because it uses feature measurements over more than two image frames. The current camera pose is linked via the 3D landmarks, and the image features track not only the previous camera pose but also the camera poses further back. The current and \( n - 1 \) previous camera poses need to be consistent with the measurements over \( N \) image frames.

**Structure Only BA**

Some steps require the refinement of 3D structure only, for example, when optimizing a newly triangulated point cloud before adding it to the map. This refinement step can be achieved by optimizing over the 3D landmarks and keeping the camera parameters fixed in the BA optimization. This effectively minimizes the reprojection error of all points involved with respect to the observing cameras in the optimization window.
3.4 Loop Closure

As mentioned in sections 3.2.1 and 3.2.2, since VO works by computing the camera path and map incrementally (pose after pose), the errors introduced by each new frame-to-frame motion accumulate over time. This generates a drift of the estimated trajectory from the real path. In the monocular scheme, as well as stereo and RGB-D, drift occurs in rotation and translation. In addition, and in contrast to RGB-D or Stereo, the monocular scheme is inherently scale-ambivalent, i.e., the absolute scale of the world is not observable. Over long trajectories this also leads to scale-drift, which is one of the major sources of error.

Because of the drift inherent in successively building a map by incremental, imperfect odometry, the map and path will never be perfectly aligned when returning to a previously visited location. To reduce the drift of the VO and to make sure the map is globally consistent, the SLAM algorithm needs to explicitly connect the new location with the previously visited location to which it corresponds, and adjust the path in between to take the new connection into consideration in order to make the parts of the map involved consistent. This process thus has two basic steps: loop detection and loop correction.

3.4.1 Loop Detection

Loop detection detects re-observations of previously mapped areas (loops). This is typically done by evaluating visual similarity between the current image and past images using place recognition algorithms. Traditionally, these methods often relied on visual bags of words based on SIFT or SURF features, but their computational overhead degraded the performance of visual SLAM systems. More recently, fast scene recognition methods using a vocabulary tree of binary descriptors have been proposed ([45], [15], [17]). With every new frame, the place recognizer will search among all previous frames for a loop candidate frame. If a candidate is found, we can match the already mapped scene points associated with the loop frame to their image observations in the current frame to establish 3D-2D correspondences that will attach the loop. Further geometric verification can be performed to accept the loop candidate and to reject outlier correspondences using RANSAC with absolute pose estimation.

3.4.2 Loop Correction

Loop correction is in charge resolving the accumulated error along the loop to achieve global consistency. Using the 3D-2D correspondences from the previous step we can link the loop frame (and its periphery) with the current frame (and its periphery) through the mutually observed map points and their corresponding observations in both images (and their peripheries).

Then we can apply BA using the new global constraints to distribute the loop closing error (due to drift accumulation) along the loop and align both sides of the loop.
However, optimizing with BA over a large number of frames and points is computationally demanding. More seriously, since BA is not a convex problem, and we could be far away from the global minimum due to the drift accumulated, the current state of the map could be far from the solution and it is likely that BA will get stuck in a local minimum.

Therefore, most recent SLAM works ([20], [15], [17]) compute an initial solution optimizing the pose graph formed from the current to the loop frame pose. In a pose graph each node is a camera pose and the constraints are the relative transformations between adjacent nodes. In our case this forms a loop of constraints. The relative transformation between the loop frame and the current frame is computed by utilizing the 3D-2D correspondences. This transformation connects both sides of the loop and informs about the drift accumulated along the loop. Then an optimization is performed over relative constraints between poses along the loop using pose-graph optimization [46], that distributes the loop closing error along the graph. Pose-graph optimization requires much less parameters to optimize than BA, which leads to fast convergence to a solution that is close to optimal (pose-graph optimization is only a rough approximation of BA). The optimization is usually performed over similarity transformations (7DoF), as opposed to rigid body transformations (6DoF), to account for scale-drift [20], [17] (to compute a similarity transformation between the loop and current frame we need 3D-3D correspondences, which we can establish by associating duplicated reconstructed map points that correspond to the same physical scene point observed in both frames). While an optimization over the 6DoF constraints would efficiently correct translational and rotational drift, it would not deal with scale drift, and would lead to an unsatisfactory overall result [20].

Finally, the whole map can be further optimized using the estimated solution as an initial solution to structure-only or full BA. Since the initial solution in this case is already very close to the optimal solution, BA will converge much faster.
3.5 Keyframe Selection

Frame selection is a very important step in VO and more so in a complete SLAM system. Many video frames contain redundant information, particularly when the camera is not moving. As complexity grows with the number of frames (due to optimizations), their selection should avoid unnecessary redundancy. In addition, as explained in section 3.3.5, when frames are taken at nearby positions compared to the scene distance (small parallax), triangulated 3D points will exhibit large uncertainty that will corrupt the whole map and trajectory. Thus, we would like to achieve a well spread set of frames observing points with significant parallax to achieve accurate results and reduce complexity. This will also allow operation with a larger numerical map size. These selected frames are called keyframes.

The ideal keyframe selection policy would be to select a frame as keyframe only when the parallax with respect to the closest keyframe exceeds a certain threshold that allows for accurate triangulation of points. The parallax is a measure that depends on the distance of observed scene points relative to the distance between the observing cameras. Since both, the scene points’ locations and the translation between the cameras are only estimations and may contain a lot of drift, especially in scale, a heuristic for predicting if enough parallax is gained needs to be used.

Heuristics for selecting a new frame as keyframe could be based on visual change, such as, when the median pixel disparity between matched keypoints with respect to the previous keyframe exceeds a threshold, or when the number of tracked features from last keyframe drops below a given threshold. It could also be based on a geometric change, such as, when the estimated motion of the camera from last keyframe exceeds a threshold, or when the depth of the observed scene relative to the baseline with respect to the last keyframe exceeds a threshold.
4 System Overview

As mentioned earlier our approach applies Mapping and Tracking techniques in combination with Image-based localization with respect to a light-weight map that consists of geo-tagged and mapped street view images from the surrounding environment. Figure 4.1 shows the general pipeline we employ. We assume the camera is calibrated and is rigidly mounted to the vehicle.

The proposed pipeline receives as input monocular images from the vehicle’s on-board camera. A monocular Visual Odometry module is in charge of incrementally tracking the 6DoF pose of the vehicle with every frame. Finally an image-based global map registration module is in charge of registering tracking with our pre-existing global map in order to achieve globally-referenced tracking and to correct the accumulated drift of the VO, whenever a good match is detected between the camera image and the tagged street view images.

![Figure 4.1: System Pipeline](image)

The pipeline is very similar to that of monocular keyframe-based SLAM, consisting of VO and a module to correct the accumulated drift and obtain global consistency. However, we do not detect loops with areas previously visited by the algorithm nor do we use the map generated by the algorithm to correct the drift. Instead we detect overlaps with the surrounding street view images (image-based) and utilize our pre-existing “map” to correct the drift. That being said, the techniques used to achieve these steps are very similar to the loop closure procedure of SLAM frameworks as outlined in section 3.4.

The next sections describe in detail the main components of the system as well as our assumptions about the geo-tagged street view database.
5 Geotagged Street View Database

This database represents our light-weight global scene model that is used for registering tracking to the global coordinate frame and correcting the accumulated drift.

Our motivation is to leverage the availability of large geotagged image databases, such as Google Street View image dataset [7], as abundant sources of accurate geotagged imagery.

Our assumptions about the database are that:

- The database covers the area travelled by the vehicle.
- The images are collected along roads. We do not assume the images are visually overlapping or that they are evenly distributed.
- Each image in the dataset is geotagged with an accurate GPS position and a full 6DoF pose can be extracted.
- Every image contains a local mapping of its observed environment in the form of a local point cloud.

From now on we will refer to the tagged street view images as anchor images, since they contain the true 3D pose of their respective camera and the true 3D positions of points they observe, in a global reference frame.
6 Monocular VO Framework

In our work we use a feature-based VO pipeline as outlined in section 3.3.1. This module maintains two main data structures:

1) A sparse point cloud which represents the map that is built by the algorithm. It holds all reconstructed 3D map points so far. These points are initialized from image measurements detected by a sparse feature detector. Each map point contains a list of references to the keyframes from where it is observed.

2) A set of Keyframes which represent the camera trajectory built so far. Each keyframe stores:
   - The corresponding 6DoF camera pose $T$ (with respect to the world frame), that allows us to map points from the world coordinate frame to the camera frame of reference.
   - A list of extracted 2D features with references to the corresponding map points (if existing).

It consists of an initialization stage to bootstrap the system, and the main loop. These will be described in detail in the next sections.

6.1 VO Main Loop

Figure 6.1 shows the pipeline of the main loop. For each new camera frame, the main loop starts by extracting local invariant keypoints from the image. The algorithm then goes on with extracting descriptors for each extracted feature, and matches them against those of the last keyframe.

After the establishment of proper feature correspondences to the previous keyframe, the algorithm proceeds to tracking the current map in order to establish 3D-2D correspondences between the current map points and the features of the current frame.

Using these correspondences, the pose of the new camera frame is computed.

We then apply a heuristic to either accept the frame as keyframe or discard it and continue with the next frame. If the frame is chosen as keyframe, we triangulate new points using the new keyframe and the previous one and add them to the map. Finally a local optimization is performed in order to refine the last keyframe poses and the local map. It is important to note, that the map mentioned in this section is the map built by the algorithm and not the pre-existing map.

In the next subsections the implementation details of the different building blocks are described.
6.1.1 Feature Extraction

The feature extraction process is the first process in our pipeline. It receives an undistorted image (using the distortion model introduced in section 3.1.3) and uses the SIFT feature detector and descriptor to extract keypoints and their corresponding descriptors from the image.

SIFT is a feature detector and descriptor devised for object and place recognition and found to give outstanding results for VO. SIFT features have proved to be stable against changes in illumination, rotation, and scale, and even up to 60° changes in viewpoint.

6.1.2 Feature Matching

Once keypoints and their descriptors are extracted from the current image we can match keypoints in the current frame with those of the previous keyframe by matching their descriptors to find 2D-2D correspondences, that is, corresponding points in the image planes that are projections of the same point in the scene.

We first find a putative feature match in the current frame for each feature in the last keyframe by applying fast ANN search in descriptor space, and avoid double referencing of features (multiple features matched to the same feature in the last keyframe). We then apply Lowe’s ratio test [36] to eliminate ambiguous matches. Lowe’s ratio test accepts the closest match (the one with the minimum Euclidean distance) only if the ratio between the closest and the second closest match is smaller than a certain threshold. The idea behind this test is to remove matches that might be ambiguous, e.g., due to repetitive structure.

We then compute a Fundamental Matrix in a robust RANSAC scheme, using the normalized 8-point algorithm, and eliminate those matches that do not adhere to the epipolar constraints, that is, the Sampson distance ([28] pp. 313-315) from the epipolar line exceeds a threshold.

In this way we are left with geometrically consistent matches for subsequent steps.

6.1.3 Map Tracking

Map tracking associates mapped scene points to their corresponding observations in the current frame. In our implementation this is done by utilizing the 3D-2D correspondences (map points associated to their respective keypoints) of the most recent keyframe as well as the 2D-2D correspondences found in the
matching process in order to establish 3D-2D correspondences between the map points (observed by the last keyframe) and the features of the current frame.

6.1.4 Absolute Pose Estimation

Given the set of 3D-2D correspondences from the previous step, we can compute the absolute pose of the vehicle (with respect to the map) by applying the P3P algorithm of [47] together with a RANSAC scheme to discard outliers. Finally, using the inlier points we recompute the camera pose by applying the EPnP algorithm [48]. We refine the resulting camera pose estimate using the Levenberg-Marquardt optimization ([28] pp. 600-613), which minimizes the reprojection error given by the sum of the squared distances between the observed image points and the corresponding reprojected 3D points. Finally we dissociate map points from their matched keypoints in the current frame if the matches were outliers to the estimated pose.

6.1.5 Keyframe Selection

Pose estimation is followed by an estimation of the overall parallax between the previous keyframe and the new frame. The availability of enough parallax is usually identified by simply thresholding the outlier-robust median pixel disparity (Euclidean distance) between the previous keyframe and the current frame. This is done by computing the median pixel disparity between the corresponding 2D-2D matches. In case this value exceeds a certain threshold, the parallax is considered to be high enough and the new frame is selected as keyframe. One problem with this approach is that the nature of the motion is simply ignored, meaning that this parallax identification process fails if the disparity in the image plane mainly results from rotation instead of translation (and thus not induced by parallax).

6.1.6 New Points Creation

2D-2D correspondences between the new and previous keyframe, that were not associated to a map point during map tracking, are triangulated using the DLT algorithm ([28] pp. 312-313) to form a new point cloud. Triangulated points whose reprojection error with respect to their corresponding keypoints is too high, or those that are behind one of the cameras (fail the chirality test), are rejected. The remaining point cloud is further refined using structure-only BA considering only these two keyframes. Finally, new surviving points are associated with their corresponding observations in both keyframes and added to the map.
6.1.7 Local Optimization

In the final stage, local BA is applied to refine the local trajectory and map. It optimizes the currently processed keyframe along with a fixed number of the last keyframes, and all the map points seen by those keyframes. A fixed number of earlier keyframes that see those points are included in the optimization but remain fixed. The minimal representation for the camera pose parameters is provided by representing rotation as an Euler Vector and leaving the translation part as it is (see section 3.1.2 for more details on minimal pose representation).

6.2 VO initialization

When the system is initialized from scratch, there are no 3D points available as we consider a monocular camera (due to its purely projective nature, it cannot observe scene structure using one image). The goal of the initialization stage is thus to compute the relative pose between the first two keyframes to triangulate an initial set of map points, for the subsequent absolute pose estimations. Figure 6.2 shows the initialization pipeline.

In this case the first frame is selected as keyframe and features are extracted as usual (section 6.1.1). Features are extracted from subsequent frames and matched to the first keyframe as in 6.1.2. The second keyframe is chosen as in 6.1.5. We now have 2D-2D keypoint matches and can use the 5-point algorithm of [41] with RANSAC to estimate the relative pose (up to a scale factor in translation). We can now initialize the initial point cloud using the first two keyframes as in 6.1.6.

Remarks

The translation in the initialization stage is estimated only up to a scale factor. In our current implementation, we assume the real translational scale between the first two keyframes is known, and we correct the scale of the reconstruction with this ground truth scale.

This scale is implicitly propagated to the rest of the map and poses through the use of absolute pose estimations in the main loop.

In addition, we assume the pose of the first keyframe in the global coordinate system is known.
7 Image-based Global Map Registration

This module is inspired by the loop closure procedure of SLAM frameworks, outlined in section 3.4, to correct the drift accumulated by the VO when an overlap with a previously visited and mapped location is detected. In SLAM the previously visited location was visited earlier in the same SLAM session and the mapped area is part of the same SLAM map that is constructed in parallel to the camera trajectory. In our case, the previously visited location is the location of an anchor image and the mapped area is its local point cloud.

Technically, our “loop detection” does not define a loop since this location was not visited by the same vehicle using the same path, and loop correction is not well-defined in this case.

In our case, instead of connecting the current frame with the loop frame, we connect the path in between two successively detected anchor images to the respective anchors at each end of the path. These connections are taken into consideration during optimization in order to correct the drift along the path and make the path consistent with our global pre-existing map.

The main data structure contained in this module is a set of Anchor frames which represent our pre-existing global map of anchor images. The structure of each anchor frame is similar to a keyframe but the extracted features are referencing the local point cloud associated with the anchor image and not the map points generated by the algorithm.

![Image-based Global Map Registration Pipeline](image.png)

**Figure 7.1: Image-based Global Map Registration Pipeline**

The pipeline is shown in Figure 7.1 and proceeds as follows:

The system receives as input keyframes from the VO system and searches for the visually closest anchor image to the new keyframe (the candidate). This is similar to using place recognition for loop detection. In case a match is found, knowing the anchor’s pose allows us to correct the new keyframe’s pose accordingly. This in turn allows us to immediately reset the drift accumulated.
The previous path since the last correction still contains drift. We correct the drift accumulated along the path by connecting the currently detected anchor to the end of the path. The previously detected anchor is already connected to the start of the path. Finally, we run an optimization to resolve the accumulated error along the path (similar to loop correction). The whole process is illustrated in Figure 7.2. We now turn to describe in detail the main components of the pipeline.
Figure 7.2: Drift Correction Process

The figure shows the first time detection occurs and drift corrected since the vehicle started driving. The green circle marks the starting position. The estimated path is shown in blue and the true path in red. The black circles mark the position of anchor images (only one is visible in this case). We can see that significant drift has been accumulated.

The top figure shows the detection phase. The position of the current frame estimated by the VO system, where an overlap with an anchor image is detected, is marked with a cyan unfilled circle and the corresponding detected anchor frame is marked with a red unfilled circle. Also shown is a close-up view of the area where detection occurred. The magnified view shows a cyan line that connects the estimated position with the corrected position according to the anchor frame (marked with a small cyan circle) and indicates the drift accumulated along the path.

The middle figure shows the instantaneous correction of the current frame’s pose and the immediate drift reset.

The bottom figure shows the correction of the drift accumulated along the previous path to obtain an accurate localization.
7.1 Candidate Detection

The candidate detection process receives as input keyframes from the VO system and searches for the visually closest anchor image in the database. We only consider anchor images that are geographically closest to the estimated (drifted) position of the current keyframe, and not the entire dataset, in order to reduce complexity and avoid selecting a wrong candidate. We do this by utilizing a kd-tree for fast nearest neighbor search in the plane (we do not consider the vertical axis coordinate). We compare the current image with the neighboring anchor images using feature matching as in 6.1.2 and the neighbor with the highest overlap (most matches) to the current keyframe is selected as the candidate image. We consider up to 4 neighbors and the comparison is done in parallel on multiple cores. Detection is considered successful only if enough matches are found and only if we “just passed” the closest anchor image. Therefore, we utilize the 3D-2D correspondences of the anchor image (points from the local point cloud associated to their respective keypoints) as well as the 2D-2D correspondences found in the matching process in order to establish 3D-2D correspondences between the anchor’s point cloud and the features of the current frame. We then compute the pose of the current keyframe relative to the anchor’s reference frame using the same pose estimation procedure described in section 6.1.4. Finally, using this relative pose we can easily determine whether the anchor image is behind the current keyframe. If it is, and if we haven’t already passed it earlier, it means we just passed the closest anchor image and detection is successful.

7.2 Instantaneous Pose Correction

If a match with an anchor image is found in the candidate detection step, we can immediately correct the pose of the current keyframe. As the global ground-truth pose of the anchor image is known, and since we have already computed the relative pose between the anchor image and the current image in the previous step, we can obtain an estimate of the current camera pose relative to the global ground-truth coordinates. This allows us to immediately reset the drift accumulated and to globally register the current pose. After pose is corrected, the current path starting from the current keyframe should be independent from the previous path that still contains drift. In addition we want to connect the anchor frame to this new path so it could be included in later optimizations and serve as an anchor to the global coordinate system and the ground-truth. Therefore, we disconnect the current keyframe from the previous path and map by dissociating map points from their observations in the current image, and instead we connect the anchor frame to the current keyframe by utilizing the 2D-2D correspondences between them found in the previous step and initializing a new point cloud. These new points are associated with their corresponding observations in both frames and a new map is initialized. Finally, the local BA window mentioned in
section 6.1.7 is reset and is reinitialized with the anchor frame and current keyframe as fixed frames for later local optimizations. This effectively creates a new path that is independent of the previous one.

7.3 Path Correction

The final step is to resolve the accumulated error (drift) along the previous path between two consecutive detections of anchor images. This is done by applying BA optimization to distribute the error along the path. Both anchor images at each end of the path are included in the optimization, as well as all intermediate keyframes and the map points they observe. Both anchor images are remained fixed in the optimization (not optimized), as they contain ground truth information in global coordinates that anchors the optimization window to the global coordinate system and scale, and force the correction of drift along the path. Before applying BA we need to first connect the anchor frames to the path. The previous anchor frame was already connected to the start of the path by the previous invocation of step 7.2. The current anchor frame is connected to the end of the path by associating mapped scene points observed by the keyframe at the end of the path to their observations in the current anchor. This is done by matching the keyframe to the current anchor by using the matching procedure in section 6.1.2 to find 2D-2D correspondences between the frames and then utilizing the map tracking procedure in section 6.1.3 to establish 3D-2D correspondences between the map points and the features in the current anchor frame. This is followed by applying the absolute pose estimation step from section 6.1.4 as a geometric verification step to discard outlying correspondences. These data association steps are very important because it is the only way in which the path is attached to the ground truth and the global reference frame, when bundle adjustment is later performed. The minimal representation for the camera pose parameters required by the optimization is provided as in section 6.1.7.

Remarks

As described, whenever we detect an overlap with an anchor image we detach the previous path and map from the current keyframe and start a new path. This effectively makes the current path independent from the previous one. The reason we do this is twofold: In this way the path correction procedure does not affect the current path and thus can run in the background in a separate thread and is detached from real-time constraints. A second reason is that since path correction does not immediately correct the previous path and map, they still contain accumulated drift and we do not want this drift to “contaminate” the current path, whose drift was reset.

The path correction procedure can be seen as an offline refinement step that aims for accurate localization of the entire vehicle’s trajectory, which can be used as an input for other applications, such as the AR application that will be described later in the report. It is not intended for real-time tracking of the
vehicle’s pose. While driving, the vehicle is only guaranteed to “see” its drifted position with occasional drift resets whenever overlaps with the anchor images are detected.

We would also like to note that in the current implementation the first keyframe acts as an anchor frame. Its pose is assumed to be known in global coordinates and it is used in the path correction procedure just like any other anchor frame. The only difference is that it is not associated with a local point cloud (and so, the initial point cloud needs to be created).

Finally, in the previous section we have remarked that the translation in the initialization stage of the system is estimated only up to a scale factor, and therefore, we assume the real metric translational scale between the first two keyframes is known in order to deduce the visual scale factor (section 3.3.1) and correct the scale of the initial reconstruction (this corrected scale will be propagated onward). The visual scale factor can also be estimated by using the anchor images. This can be achieved by comparing the metric distance traveled computed between the first keyframe (whose global position is assumed to be known) and the first globally localized keyframe (by instantaneous pose correction), with the unscaled motion estimate between the two keyframes returned by the VO. This requires a longer initialization stage where tracking is performed in an arbitrary scale until the first invocation of instantaneous pose correction occurs. We also assume that the pose of the first keyframe in the global coordinate system is known. Without this assumption, at system startup tracking is performed in a local reference frame and so the current image can’t be compared to its neighboring anchors in the candidate detection step (since their positions are not measured in the same reference frame). Thus, we would have needed to search the entire street view database for a matching anchor image in the candidate detection step. This requires an initialization stage where tracking is performed in an arbitrary local reference frame until the first detection and global localization occurs. In addition, comparing the current image to the entire database is very computationally intensive and may result in wrong candidates (due to repetitive structures).
8 Experimental Procedure and Results

This section provides an experimental evaluation of our system. We first describe the experimental dataset and setup used for evaluating our method, and then present both qualitative and quantitative results and analyze them in detail.

8.1 KITTI Dataset

We evaluate our method on the KITTI vision benchmark suite [2]. The odometry benchmark from the KITTI dataset contains 11 stereo sequences captures by a car driven around the mid-size city of Karlsruhe, Germany, in rural areas and on highways. Each sequence is provided with accurate ground truth of the travelled path from GPS and a Velodyne laser scanner mounted on the car (see Figure 8.1). The images are undistorted 1241x376 monochromatic. This is a very challenging dataset for monocular vision due to fast rotations, areas with lot of foliage, which make more difficult data association, the vehicle speed varies from 0 to 90 kph, the frame capture rate is low as 10Hz, and there exist many moving objects in the scene (such as other vehicles and pedestrians). All these make the pose estimation procedure very hard.

The sequences have different characteristics in terms of environment, traffic and speed to assess performance in realistic driving situations.

Figure 8.1: KITTI Sensor Setup
The vehicle is equipped with four video cameras (two color and two grayscale cameras), a rotation 3D laser scanner and a combined GPS/IMU inertial navigation system.
8.2 Experimental Setup

In order to simulate the use of anchor images, we extract from each sequence a number of images along with their ground truth information to serve as anchor images. These anchor images are taken at regular intervals along the path. We test 20, 50 and 100 meter intervals between one anchor image to the next. Since we are dealing with a monocular camera, we only use a single channel from the stereo sequences in KITTI.

For each anchor image, the local point cloud it observes is generated using the corresponding stereo pair using feature extraction, feature matching and new points creation (from the resulting 2D-2D matches) as outlined in sections 6.1.1, 6.1.2 and 6.1.6 respectively. The end result is that each anchor image contains 3D-2D correspondences between its locally mapped environment and corresponding image observations.
8.3 Results and Analysis

In this section we present the experimental results on the KITTI dataset. The qualitative results are shown using a bird-eye-view perspective, as is usually done in literature when evaluating a road vehicle’s trajectory (the motion is usually very planar). In the quantitative results, time is measured in seconds and was extracted from the timestamps (provided by the ground truth information in KITTI) of the keyframes along the path. The positional error is measured as the Euclidean distance between the estimated 3D position of the vehicle and the corresponding ground truth position.

Unless stated otherwise, evaluation is performed on a subsequence of sequence 07 from the KITTI dataset and anchor images are distributed approximately every 50 meters along the path.

Important Note
The main goal of this work is to present a proof-of-concept of the system, rather than a real-time, efficient implementation. Therefore, we do not present here an analysis of the processing times of the algorithm and its components.

8.3.1 Comparison to Standard VO

In a first round of experiments, we evaluate the performance improvement that our proposed system yields in comparison to a standard VO algorithm. In particular, we compare our own VO pipeline as described in section 6 to the full method proposed, where anchor images are used to compensate for possible drifts of the VO algorithm.

Figure 8.2 shows a qualitative comparison of the trajectories estimated by both methods against the ground truth. It can be seen that when the VO is not updated by the image-based global map registration system, a very big error is accumulated in the trajectory of the vehicle. It can also be seen that the path generated by the VO is similar in shape, at least locally, to the real path but the scale changes with time. That is, it suffers from significant scale-drift. The large amount of drift can be partially explained by the fact that the visible scene is always very local and thus only a very local map is tracked at each frame, leading to only local consistency of the trajectory and not global consistency.

Conversely, when the proposed image-based global map registration system is used, the trajectory of the vehicle is corrected, achieving an almost perfect alignment with the true path (at least compared to the VO result). The path visualizations show that our system is capable of compensating for the drift that the visual odometry locally accumulates.
Figure 8.2: Qualitative comparison against standard VO
We display the trajectory measured by the ground truth in red, the path estimated by the algorithm described in this work with blue, and the path estimated by the standard VO with green. The black circles mark the positions of the anchor images, and the cyan unfilled circles mark the positions of keyframes where an overlap with an anchor image was detected and drift corrected.

Figure 8.3 shows a quantitative comparison of the position and scale error over time, relative to the ground truth. The blue squares mark the moments when detection occurs and drift corrected in the proposed method. The scale error depicts the drift in scale and is computed by comparing the Euclidean norm of the relative translation between adjacent poses against the ground truth (it is measured in percentage):

\[
\text{scale error} = 100 \cdot (\text{scale ratio} - 1), \quad \text{scale ratio} = \frac{\|\text{EST } t_{k}^{k+1}\|}{\|\text{GT } t_{k}^{k+1}\|}
\]

Where \text{EST } t_{k}^{k+1} is the estimated relative translation between adjacent keyframe poses and \text{GT } t_{k}^{k+1} is the ground truth relative translation between them.
Figure 8.3: Quantitative comparison against standard VO
The top figure shows the position error (or drift) over time. The bottom figure shows the scale drift over time. The blue squares mark the moments when detection occurs and drift corrected in the proposed method.

Here we can clearly see that standard VO rapidly accumulates significant scale-drift, which leads to a significant accumulation of drift in position. Analyzing both the visualization of the trajectory and the scale-drift graph of the VO reveals that most scale-drift accumulation occurs around turns. This can be explained by the fact that at turns we transition from one local scene to another and many observations are lost in the process, leading to a very local visible scene. The sharper the turn is, the more local the visible scene becomes, and as explained earlier this leads to a more rapid drift in scale. Another factor is that at turns the vehicle slows down. The sharper the turn is the more it slows. This leads to small relative translations between poses and more rotational motion, which in turn leads to a small parallax of the visible scene relative to the camera motion and to a large uncertainty in the triangulation of new points. This uncertainty is propagated to the subsequent camera pose through the absolute pose estimation process, which in turn propagates the uncertainty to new triangulated points and so on. This propagated uncertainty is manifested as drift. As already mentioned in section 6.1.5, our keyframe policy fails to select frames with enough parallax in this case since the disparity in the image plane mainly results from rotation instead of translation and is not induced by parallax.

Future improvements in our visual odometry implementation could lead to a reduction in drift during exploration. Nevertheless, a certain amount of drift during exploration is unavoidable and our main focus is how to deal with drift when it occurs.

As can be seen, our global map registration system resets and corrects the drift in scale and position whenever a match is detected. This leads to a localization error that is orders of magnitude smaller and does not accumulate. In particular, the standard VO system yields an average error of 26 m in comparison to 26 cm of the proposed method (two orders of magnitude). In addition, as can be seen the VO error
would keep rising with the length of the path (if it were any longer) while the proposed approach would not be much affected by this.

This demonstrates that localization accuracy can increase significantly with the use of geotagged imagery from the vehicle’s surroundings.

### 8.3.2 Evaluation of Path Correction Procedure

Here we evaluate the performance of the path correction procedure by comparing it to a system without path correction. In particular, we compare the method described up-to section 7.2 that resets the drift when detection occurs but does not correct the previous path, against the full system that also includes path correction. This comparison is also important for assessing how accurate localization is during live tracking of the vehicle’s position, since, as mentioned earlier, path correction is applied in an offline manner and the refined localization is not immediately “seen” by the vehicle.

Figure 8.4 shows a quantitative comparison of the position error over time for both methods. Without path correction we can clearly see how the localization error is accumulated until detection occurs and drift is immediately reset. This is characterized by a sawtooth graph. We can also see how path correction, using optimizations, distributes the error more evenly along the path between two successive detections and avoids large error buildup in one location. The resulting shape of the distributed error along the sub-paths is of an upside down parabola, which is characterized by small error near the edges of the path and large error close to the center. This shape characterizes least squares optimizations such as BA.

In this case without path correction, the system yields an average error of 80 cm, but with a maximum of 8.3 m, in comparison to 26 cm and with a maximum of 1.2 m when applying path correction.

![Drift vs. Time](image)

**Figure 8.4: Comparison of position error over time with and without path correction**

Black is with path correction and light blue is without path correction. The squares mark the moments when detection occurs for the respective method (the squares overlap since the methods differ only in the path correction component).
Figure 8.5 shows a qualitative comparison of the trajectories estimated by both methods against the ground truth.

Using path visualizations, it is hard to notice a difference in the trajectories (which both align quite nicely with the ground truth) since the error after drift reset is already relatively small, in the order of a couple of meters, compared to the size of the entire trajectory.

Therefore, we provide a close-up view on some part of the trajectory that shows a difference between the paths. In the magnified view it can be seen that without path correction there is a deviance from the true path that accumulates, but does not build up to a significant error by the time the next drift reset occurs. Still we can see that with path correction, the path achieves a better alignment with the true path.

**Figure 8.5: Qualitative comparison of estimated trajectories with and without path correction**

We display the trajectory measured by the ground truth in red, the path estimated using path correction with black, and the path estimated without path correction with light blue. The black circles mark the positions of the anchor images. A close-up view on some part of the path (indicated by a black rectangle) is also provided in order to show fine-scale differences between the estimated trajectories.
Figure 8.6 shows another trajectory comparison, this time with anchor images distributed about every 100 meters along the path. In these path visualizations we can clearly see significant accumulation of drift between two successive detections when not using path correction, and how path correction corrects the accumulated drift along the path to achieve a much more accurate localization.

![Vehicle Trajectory](image)

**Figure 8.6:** Qualitative comparison of estimated trajectories with and without path correction using 100 meter intervals between anchor images

We can also see that without path correction, deviations from the true path are mainly realized around turns. The rapid scale-drift around turns analyzed in the previous section might contribute to this phenomenon. However, in this case this is mainly because during the straight path before the turn, drift accumulates in scale, making the straight path longer (or shorter) than the true path, but the direction is preserved. If there is no turn this deviation from the true path will not be noticed, because the drift reset will shift the estimated path back (or forth) to align with the true path, and the deviation that remains (in the form of a straight line) will be hidden by the next straight path. This can also explain why in the
previous figure analyzed, there is a significant quantitative difference between the methods, but when we
turn to the visualizations of the trajectories the error is not as apparent. On the other hand, if there is a
turn, the deviation goes in one direction and the true path turns to another direction. This will eventually
reveal the lack of alignment.
Finally, this evaluation has demonstrated that standard VO can produce rather accurate results for short-
range distances (in this case 50 meters). Over longer distances (in this case 100 meters), however, even
though standard VO is able to reconstruct the topology of the traversed path, it often gives significant
localization errors due to the accumulated drift. Our algorithm (with path correction), on the other hand, is
able to keep the localization error consistently in a low range, making the whole system more robust and
stable.

8.3.3 Evaluation of Different Intervals

Here we compare and evaluate our proposed method applied with different intervals between anchor
images. This is done in order to assess the performance of our method as the density of database images
along roads is decreased. If our method can extend to sparser distributions of anchor images, it will
become much more general. We evaluate performance using 20, 50 and 100 meter intervals between one
anchor image to the next.

Figure 8.7 shows a qualitative comparison of the trajectories estimated using the different intervals. Using
path visualizations, it is hard to notice a difference in the trajectories (which all align quite nicely with the
ground truth) since the localization error is relatively small compared to the size of the entire trajectory.
Therefore, we provide close-up views on some parts of the trajectory that show a difference between the
different paths. In the bottom magnified view it can be seen that using 100 meter intervals causes a
deviation of about 1 or 2 meters from the true path. It is hard to notice any difference in this magnification
scale when using 20 or 50 meter intervals, which both align with the true path. The second magnified
view shows that, as expected, using 20 meter intervals results in the most aligned path with the real one,
this is followed by the result of using 50 meter intervals, and finally using 100 meter intervals results in
the least aligned path. It is important to note that here we are dealing with a very fine magnification scale,
and that the error differences between the paths is in the order of a couple tenths of centimeters.
Figure 8.7: Qualitative comparison of the trajectories estimated using different intervals between anchor images

We display the trajectory measured by the ground truth in red, the path estimated using 20 meter intervals with black, the path estimated using 50 meter intervals with green, and the path estimated using 100 meter intervals with light blue. The circles mark the positions of the anchor images in the different interval configuration and their color mark the respective interval configuration they belong to. Two close-up views on two parts of the path (indicated by black rectangles) are also provided in order to show fine-scale differences between the estimated trajectories.

Figure 8.8 shows a quantitative comparison of the position error along the path for the different interval configurations. Here we can clearly see that using 20 meter intervals results in the most accurate localization yielding an average error of 12 cm (with a maximum of 54 cm), this is followed by 50 meter intervals that yield an average error of 26 cm (with a maximum of 1.14 m), and finally 100 meter intervals that yield an average error of 64 cm (with a maximum of 1.48 m).
Figure 8.8: Comparison of position error over time for different intervals between anchor images
The squares mark the moments when detection occurs and drift corrected in each of the configurations, and their color mark the respective interval configuration they belong to.

Qualitative evaluations of the estimated trajectories against the ground truth for different sequences from the KITTI dataset when using 20, 50 and 100 meter intervals are shown in Figure 8.9, Figure 8.10 and Figure 8.11. It is hard to notice any difference between the different interval schemes since the localization error, even when using 100 meter intervals, is relatively small compared to the size of the entire trajectory and there is a need to zoom in, in order to see fine scale differences. The difference is most apparent in sequence 07, where using 100 meter intervals results in a considerable lack of alignment at the bottom left part of the trajectory, compared to the other interval configurations.

Overall we can see that our proposed method generates results that are very close to the true trajectories, even when anchor images are far apart.
Figure 8.9: Qualitative evaluation of trajectories estimated for different sequences using different intervals – Part 1
Each column corresponds to a different sequence from the odometry benchmark of the KITTI dataset. The left column corresponds to sequence 03 and the right to sequence 06. Each row shows the trajectories estimated when using a different interval configuration. The top row shows results for 20 meter intervals, the middle row for 50 meter intervals, and the bottom row for 100 meter intervals.
Figure 8.10: Qualitative evaluation of trajectories estimated for different sequences using different intervals – Part 2
Each column corresponds to a different sequence from the odometry benchmark of the KITTI dataset. The left column corresponds to sequence 07 and the right to sequence 09. Each row shows the trajectories estimated when using a different interval configuration. The top row shows results for 20 meter intervals, the middle row for 50 meter intervals, and the bottom row for 100 meter intervals.
Figure 8.11: Qualitative evaluation of trajectories estimated for different sequences using different intervals – Part 3
The figure shows results for sequence 10. Each row shows the trajectories estimated when using a different interval configuration. The top row shows results for 20 meter intervals, the middle row for 50 meter intervals, and the bottom row for 100 meter intervals.
Table 8.1 shows the average localization error for the different sequences when using the different interval configurations. We also provide the length of the trajectories to put in context the errors.

As can be seen, the average error when using 20 meter intervals does not surpass 20 cm, when using 50 meter intervals we achieve sub-meter error, and when using 100 meter intervals still the error is around a single meter.

The results demonstrate that, even when using sparsely distributed anchor images, our system is very accurate and is able to localize the vehicle with an error of only a few meters, usually achieving sub-meter accuracy.

Table 8.1: Average position error per sequence for different intervals between anchor images
Blue shows results for 20 meter intervals, orange for 50 meter intervals, and gray for 100 meter intervals. The length of each sequence appears next to the sequence number and is shown to put in context the errors.

8.3.4 Video Demo of System’s Operation

An example of the system’s operation is provided in the accompanying video file and illustrated in Figure 8.12. It demonstrates the localization procedure using the camera video for sequence 07 from the KITTI dataset. The camera video (using the KITTI recording platform) was captured at 10FPS (frames per second) and the video is shown at 20FPS. Note that our current implementation does not run in real time and the video provided was first captured by our system and then played back much faster for demonstration purposes.
Figure 8.12: Screenshots from the video demo

The top image shows the observed environment of the current camera view, overlaid with green tracks. Each track tracks the observations of a single scene point, observed by the current camera, over several consecutive frames. These tracks show the vehicle’s ego-motion (similar to optical flow). The bottom two figures show the trajectory that is generated by our proposed method (blue), against the true path (red). The black circles mark the positions of anchor images. The right image shows the whole trajectory generated so far from the world coordinate system’s point of view. The left image shows the local trajectory generated from the vehicle’s point of view (current camera reference frame).
9 Usage in Augmented Reality

To investigate the suitability of the resulting localization of our proposed method for AR driver assistance tasks, we have developed a simple AR application. The proposed application receives as input the resulting localization of a vehicle, acquired by our method, and draws an overlay on the road of the 6DOF path traversed by the vehicle for a second vehicle driving on the same path. This can be used, for example, when the second vehicle wishes to track the location of the first vehicle, even when the first vehicle is visually lost or has already reached its destination long ago. In order to generate the overlay for the current camera view of the second vehicle, we transform all the camera poses along the first vehicle’s trajectory, that are visible by the current camera, to the coordinate system of the current camera. The poses are represented as 2D rectangles in 3D space (which represent the cameras’ coordinate frames) and their vertices can be combined together using interpolations to form one smooth 3D representation of the visible path seen by the second vehicle. Finally, this 3D path is projected onto the image of the current camera and drawn as an overlay on top of the image.

In order to simulate the use of this application using the KITTI dataset, the path travelled by the second vehicle is taken to be the same path of the first vehicle. In this case, the application actually degenerates to a vehicle viewing its own future path.

An example of the application’s operation is provided in the accompanying video file and illustrated in Figure 9.1. We use sequence 09 from the KITTI dataset and the localization results are obtained using 20 meter intervals between anchor images. The camera video was captured at 10FPS (using the KITTI recording platform) and the video is shown at 30FPS. As can be seen, the lane-level localization accuracy obtained by our method allows for pixel-accurate graphical AR overlays.
Figure 9.1: Screenshots from the video demo of the simple AR application

The future path of the vehicle (colored in transparent cyan) is overlaid on top of the camera images.
10 Conclusions

In this work, we presented a novel approach for globally localizing a vehicle on the road using a single on-board camera and utilizing the availability of priorly geo-tagged street view images from the surrounding environment together with their associated local point clouds. Our approach combines SLAM techniques with image-based localization in order to provide accurate and globally-registered 6DoF tracking of the vehicle’s position at all times. The method incrementally tracks the position of the vehicle using a classical visual odometry pipeline, and combines the tagged images as a source of global positioning along with techniques inspired by loop closure in order to correct the accumulated drift, whenever a good match is detected between the camera image and the tagged street view images.

In comparison to SLAM systems, by integrating a light-weight model-based approach, the vehicle is not forced to drive in loops and its freedom of movement is not limited, which permits wide area exploration. Furthermore, the pose of the camera is given in the global reference frame and the true scale of the world. In comparison to Model-based approaches, the integration of SLAM techniques allows for continuous 6DoF tracking that is not limited only to areas covered by the offline model (in our case, the tagged images). The infrastructural cost of our system is low and our prior assumptions about the environment are rather weak, as we only assume a very sparse and light-weight “map” of the environment, where street view images exist, and not a full reconstruction of the environment. Additionally, the sensor requirements of our technique is only a monocular camera, which makes the algorithm easy to deploy and affordable to use.

We have evaluated our approach using data from a demanding benchmark and we have shown that the proposed system is able to stabilize odometry against drift over long travel distances, even when the tagged images are wide apart and sparsely distributed along roads, and we are able to localize our vehicle with an error of only a few meters, usually achieving even sub-meter accuracy. The results show that our solution provides a reliable alternative to GPS systems, which is purely based on vision.

Finally, we have shown how the accurate lane-level localization produced by our method can be utilized to provide pixel-accurate AR overlays for a driver assistance application.

We believe that this technique paves the way toward a new cheap and widely useable outdoor localization approach.
11 Future Work

In the future we plan to incorporate GPS information into the system, when available, in order to achieve further accuracy during tracking. Visual odometry is complementary to GPS in offering localization which is smooth and accurate locally whilst GPS is coarse but offers absolute measurements. [49] shows a method for combining visual odometry estimates obtained from a vehicle’s rear-facing parking camera with standard automotive GPS to provide accurate global localization.

In addition, currently we only simulate the use of geotagged imagery by our system. One of our future goals is to leverage Google Street View as an abundant source of accurate geotagged imagery for our image-based global map registration system. Google Street View image database, consists of billions of street-level panoramic images acquired (and continuously updated) all around the world and collected each 5-10 meters along roads. In order to integrate its use in the system, our candidate detection system should be robustified to allow matching images that observe the same scene from totally different viewing angles, taken at different seasons and times of the day, acquired by different cameras, etc.

Another direction for future work would be to improve our VO implementation to reduce the accumulation of drift. The current implementation is rather naïve with regard to the map tracking procedure. Each frame tracks only a very local map, specifically, the map observed by the previous keyframe. The points in the map act as the “glue” that implicitly connects between cameras along the path through mutual observations of the same map points in the cameras’ images, when later BA optimizations are performed. Tracking a very local point cloud thus leads to generating very weak and local connections between the cameras along the path. In addition, occlusions and disocclusions of a point in this case will lead to point duplications (multiple reconstructed points correspond to the same physical point in space).

We need global and strong camera connections so that lots of constraints between the cameras are present later when optimizations are performed. This will enable the optimization to successfully correct the drift accumulated along the path and to obtain global consistency. Additionally, since map tracking produces 3D-2D associations used for pose estimation, tracking only a very local map will lead to a very local consistency of the trajectory, thus leading to a much more rapid accumulation of drift in scale during tracking. A better approach would be to track the entire map, or at least a more global map, in order to produce global and strong camera connections, to avoid duplication of map points, and to estimate more robustly the camera pose that would lead to a reduction of drift during tracking. To this end, we need to utilize sophisticated data structures that will allow tracking the entire map at each frame in reasonable time.

Finally, we could improve our optimization framework to allow more accurate and faster path correction. Currently path correction is performed by BA optimization using all frames and observed map points.
along the path. As mentioned in section 3.4.2, this is computationally demanding and the optimization most likely will get stuck in a local minimum, and a better approach would be to compute a solution using pose-graph optimization along the path. This initial solution could be further optimized using structure-only or full BA. This will result in a much faster convergence to a more accurate solution.
References


