Implementation and evaluation of a 3D tracker

Examensarbete utfört i datorseende vid Tekniska högskolan vid Linköpings universitet
av
Andreas Robinson
LiTH-ISY-EX--14/4800--SE
Linköping 2014
Implementation and evaluation of a 3D tracker

Examensarbete utfört i datorseende vid Tekniska högskolan vid Linköpings universitet av

Andreas Robinson

LiTH-ISY-EX--14/4800--SE

Handledare: Johan Hedborg
ISY, Linköpings universitet

Examinator: Klas Nordberg
ISY, Linköpings universitet

Linköping, 30 september 2014
Implementation and evaluation of a 3D tracker

Many methods have been developed for visual tracking of generic objects. The vast majority of these assume the world is two-dimensional, either ignoring the third dimension or only dealing with it indirectly. This causes difficulties for the tracker when the target approaches or moves away from the camera, is occluded or moves out of the camera frame.

Unmanned aerial vehicles (UAVs) are increasingly used in civilian applications and some of these will undoubtedly carry tracking systems in the future. As they move around, these trackers will encounter both scale changes and occlusions. To improve the tracking performance in these cases, the third dimension should be taken into account.

This thesis extends the capabilities of a 2D tracker to three dimensions, with the assumption that the target moves on a ground plane.

The position of the tracker camera is established by matching the video it produces to a sparse point-cloud map built with off-the-shelf structure-from-motion software. A target is tracked with a generic 2D tracker and subsequently positioned on the ground. Should the target disappear from view, its motion on the ground is predicted. In combination, these simple techniques are shown to improve the robustness of a tracking system on a moving platform under target scale changes and occlusions.
Abstract

Many methods have been developed for visual tracking of generic objects. The vast majority of these assume the world is two-dimensional, either ignoring the third dimension or only dealing with it indirectly. This causes difficulties for the tracker when the target approaches or moves away from the camera, is occluded or moves out of the camera frame.

Unmanned aerial vehicles (UAVs) are increasingly used in civilian applications and some of these will undoubtedly carry tracking systems in the future. As they move around, these trackers will encounter both scale changes and occlusions. To improve the tracking performance in these cases, the third dimension should be taken into account.

This thesis extends the capabilities of a 2D tracker to three dimensions, with the assumption that the target moves on a ground plane.

The position of the tracker camera is established by matching the video it produces to a sparse point-cloud map built with off-the-shelf structure-from-motion software. A target is tracked with a generic 2D tracker and subsequently positioned on the ground. Should the target disappear from view, its motion on the ground is predicted. In combination, these simple techniques are shown to improve the robustness of a tracking system on a moving platform under target scale changes and occlusions.
# Contents

Definition of terms vii

1 Introduction 1
1.1 Background 2
1.2 Goals 2
1.3 Motivation 2
1.4 System overview 3
1.5 Evaluation 4
1.6 Limitations 4
1.7 Outline 4

2 Related work 5
2.1 Three-dimensional tracking 5
  2.1.1 Some approaches 5
  2.1.2 Analysis 6
2.2 Visual object tracking 6
  2.2.1 Introduction 6
  2.2.2 Brief overview 6
  2.2.3 Performance measures 7
  2.2.4 Selection 8
  2.2.5 The Adaptive Color Tracker 10
2.3 Mapping and localization 11
  2.3.1 Introduction 11
  2.3.2 Approaches 12
  2.3.3 Structure from motion 13
  2.3.4 Analysis 14

3 System description 15
3.1 Overview 15
3.2 The pinhole camera 17
  3.2.1 Extrinsic 17
  3.2.2 Intrinsic and distortion 18
3.3 Feature extraction and map building 19
| 3.4 | Tracker map localization | 20 |
| 3.4.1 | The camera localization algorithm | 20 |
| 3.4.2 | Perspective n-point estimation | 21 |
| 3.5 | Target tracking | 22 |
| 3.5.1 | Implementation notes | 23 |
| 3.6 | Target localization | 24 |
| 3.6.1 | Ground plane estimation | 24 |
| 3.6.2 | 3D bounding box estimation | 25 |
| 3.6.3 | Handling target scale changes | 28 |
| 3.7 | Motion modelling | 28 |
| 3.7.1 | The constant acceleration motion model | 28 |
| 3.7.2 | The Kalman filter | 30 |
| 3.7.3 | Controlling the Kalman filter | 30 |
| 3.8 | Visualization | 31 |
| 3.8.1 | Tracking | 31 |
| 3.8.2 | Image stabilization | 33 |

| 4 | Experiments | 35 |
| 4.1 | Setup | 35 |
| 4.2 | Positioning experiments | 36 |
| 4.2.1 | Test sequences | 36 |
| 4.2.2 | Camera pose experiment | 37 |
| 4.2.3 | Target position experiment | 39 |
| 4.3 | Qualitative experiments | 41 |
| 4.3.1 | Target out of view experiment | 42 |
| 4.3.2 | Target occluded experiment | 43 |
| 4.3.3 | Results | 45 |
| 4.4 | Runtime performance | 45 |
| 4.4.1 | Offline | 46 |
| 4.4.2 | Online | 46 |

| 5 | Analysis | 47 |
| 5.1 | Discussion | 47 |
| 5.1.1 | Map building and localization | 47 |
| 5.1.2 | Occlusions and out-of-view conditions | 48 |
| 5.1.3 | Speed | 48 |
| 5.2 | Conclusions and future work | 49 |

| Bibliography | 51 |
### Definition of terms

#### Definitions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2D tracker</td>
<td>A visual object tracker capable of following a target object moving in a video sequence over several seconds or minutes. This tracker does not use any information of the 3D structure of the scene in the video sequence.</td>
</tr>
<tr>
<td>3D tracker</td>
<td>A system like the 2D tracker, that also integrates some knowledge about the 3D structure of the scene.</td>
</tr>
<tr>
<td>Bounding box</td>
<td>In 2D, a rectangle drawn around a tracking target to mark its position in an image. In 3D, a rectangular cuboid marking the target in world space.</td>
</tr>
<tr>
<td>Camera image</td>
<td>A raster image captured by a camera, along with a set of features - keypoint-descriptor tuples generated from the image.</td>
</tr>
<tr>
<td>Camera pose</td>
<td>An affine 4x4 matrix on the form $C = \begin{bmatrix} R &amp; t \ 0 &amp; 1 \end{bmatrix}$ that places and orients a camera in the world space. In this thesis, the camera pose also refers to the position and orientation of the tracker system.</td>
</tr>
<tr>
<td>Camera projection matrix</td>
<td>Projects world space points seen by the camera onto the camera image plane. $P = \begin{bmatrix} I &amp; 0 \end{bmatrix} C^{-1} = \begin{bmatrix} R^T &amp; -R^T t \end{bmatrix}$ where $I \in \mathbb{R}^{3 \times 3}$ is the identity matrix.</td>
</tr>
<tr>
<td>Descriptor</td>
<td>A compact description of the appearance of the area around a keypoint in a camera image in the form of a n-dimensional vector. It is encoded so that matching features in different images with one another is as efficient and robust as possible.</td>
</tr>
<tr>
<td>Feature</td>
<td>A keypoint paired with a descriptor.</td>
</tr>
</tbody>
</table>
## Definitions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keypoint</td>
<td>A point in a camera image where some structure has been observed. It also specifies the size (scale) of the structure.</td>
</tr>
<tr>
<td>Landmark</td>
<td>A three-dimensional point in a point cloud, reconstructed from three or more features.</td>
</tr>
<tr>
<td>Plane</td>
<td>A two-dimensional Euclidean plane $P = (\hat{n}, d)$ embedded in $\mathbb{R}^3$ where $\hat{n} \in \mathbb{R}^3$ is the unit-length normal vector and the scalar $d$ is the distance from the plane to the origin. A point $x \in \mathbb{R}^3$ is on the plane if $\hat{n} \cdot x + d = 0$.</td>
</tr>
<tr>
<td>Pitch angle</td>
<td>The angle by which the camera &quot;looks down&quot;. At zero degrees pitch, the camera looks straight forward along a line parallel to the ground. At ninety degrees, it looks straight down.</td>
</tr>
<tr>
<td>Roll angle</td>
<td>The angle by which the camera is rotated around its optical axis. At zero degrees roll, the sky is up and ground is down in the image plane. At ninety degrees, the sky appears to the left and the ground to the right.</td>
</tr>
<tr>
<td>World coordinates</td>
<td>A three-dimensional Euclidean space ($\mathbb{E}^3$). The positions of the tracker, the target and landmarks in the map are all defined in coordinates in this space.</td>
</tr>
</tbody>
</table>
Introduction

Figure 1.1: Left: A six-rotor helicopter UAV about to take aerial photographs of the Atacama Large Millimeter Array radio telescope in Chile. Right: A fixed-wing reconnaissance UAV.
1.1 Background

The use of unmanned aerial vehicles (UAVs) in civilian applications is increasing. Low-cost, low-maintenance, small-scale remote-controlled airplanes and multirotor helicopters like those in figure 1.1, see diverse use in areas such as surveying, search-and-rescue, power-line inspection, in surveillance, traffic monitoring and in film photography. Much of this work can be time-consuming, monotonous or otherwise mentally fatiguing to the pilots operating these aircraft. Hence, it is of particular interest to build systems that automate the dull and dumb aspects of these jobs. But, this implies performing tasks only humans were capable of up until now.

One fundamental human ability is visual tracking of moving objects or individuals. Easy and natural to us, we rarely reflect on this capability. However, visual object tracking is a hard problem and a vast amount of work has gone into developing tracking systems. Typical implementations estimate the 2D image coordinates of targets moving across the video image plane, marked by some type of bounding box. This kind of two-dimensional tracking runs into problems as the actual motion is inherently three-dimensional. Some approaches adapt to the changing scale of an object as it approaches or retreats from the camera. Others also handle occlusions, when the target moves behind other objects. Yet, neither group explicitly involve depth information.

1.2 Goals

The aim of this work is to create and evaluate a new "3D tracker", exploiting knowledge of three-dimensional structure extracted from camera images.

The questions studied are

- What can be gained from combining 3D spatial information with a 2D tracker?
- Can such a system achieve real-time performance?
- What are its limitations?

1.3 Motivation

Although some very general uses of UAVs are outlined above, there are not really that many autonomous civilian unmanned aerial systems available yet. It is nevertheless easy to imagine some scenarios where a 3D tracking system could be useful.

---

1“The Hexacopter” by ALMA (ESO/NAOJ/NRAO) and the European Southern Observatory via Wikimedia Commons. Licensed under Creative Commons 3.0 Attribution Unported. This is a cropped version of the original.

2Public domain via Wikimedia Commons. Edited to emphasize the plane.
One such situation could be search and rescue. Imagine a rescue worker or vehicle being followed from above, providing situational awareness for staff at a base of operations. Another scenario is to have an autonomous system detect and track individuals moving around a restricted area. Similarly, it can perform wildlife monitoring, perhaps spotting and tracking wild boars destroying crops. A less serious application is filming of sports and other public events. The autonomous system can for example track a specific marathon runner, downhill skier or rally car from above.

1.4 System overview

With the applications in the previous section in mind, the system developed here is intended track a target moving on the ground with a small unmanned aircraft. A schematic view of the situation is shown in figure 1.2.

![Figure 1.2: Schematic scene with a UAV and a target.](image)

The actual tracking task is performed by an existing 2D visual object tracker. Since it does not model the three-dimensional nature of the problem, it is complemented by a number of supporting components, or subsystems. The first of these, is a map of the area in the form of a sparse 3D point cloud of visual landmarks. In this case the map is built in advance, while the rest of the system is intended to run in real-time (or online) aboard the aircraft.

In the online part, the map is used to position the tracker in world space. Once the position is known, it is combined with the output of the 2D tracker to place the target in world space. This position is in turn used to estimate the target motion and scale as seen by the tracker. With the motion model, it is possible to estimate the movement of the target if it should disappear from view. Knowing the scale reduces the likelihood of the tracker losing the target when the target size in the camera changes.
1.5 Evaluation

The 3D tracker will be evaluated in a set of experiments based on video sequences of an radio-controlled car captured with a hand-held camera. This setup is used in place of a UAV-mounted camera that is tracking life-size cars.

Experiments will quantitatively evaluate the precision and limitations of the tracker self- and target-localization. In qualitative tests, both the new tracker and a set of state-of-the-art 2D trackers are compared. Here, the video sequences are designed so that a 2D tracker may fail while the new system is expected to succeed, illustrating its added capabilities.

1.6 Limitations

The tracking problem can be made to be rather extensive, so it is necessary to restrict the scope of the system developed and described this thesis.

First, only one target at a time is tracked. Second, the target is a rigid object like a car, moving on a ground plane. These limitations simplify both the target tracking and motion prediction subsystems.

Third, other systems generate maps for localization purposes using partial or complete structure-from-motion pipelines. Here, standalone software is used to perform this task. Building a new or integrating an existing pipeline is a large undertaking and will be avoided.

Finally, the map-lookup or localization task uses a basic method based on a nearest-neighbor linear search. However, another strategy will be discussed in the related work section.

1.7 Outline

This thesis is organized as follows: Previous work related to the various subsystems is presented in chapter 2. This includes a brief discussion of both 3D object tracking, 2D object tracking, mapping and localization. Chapter 3 details how the new 3D tracker is implemented with experiments and results presented in chapter 4. Finally, discussion and conclusions follow in chapter 5.
This chapter reviews some of the previous work in the areas of three-dimensional tracking, visual object tracking, mapping and localization.

2.1 Three-dimensional tracking

In this thesis "3D tracking" refers to visual tracking of a rigid object moving on the ground but there is no set definition found in the literature. This section reviews some methods where either the tracking target, the surroundings or both are modelled as three-dimensional.

2.1.1 Some approaches

Object tracking in 3D space with monocular cameras has seen use in robotics and augmented reality applications. Lepetit and Fua [2005] surveyed existing work in this area and found two main approaches. The first tracks markers such as bar-codes affixed to the targets, rather than the target objects themselves. The second is model-based tracking. In this approach, targets are found and followed by comparing the camera view with a man-made model stored as a three-dimensional CAD-drawing.

A system designed for traffic applications was proposed by Manz et al. [2011]. Their method tracks the pose and distance to a target car driving in front of the tracking car. A simple CAD model encodes key boundaries between visual features on the target such as the outlines of the wheel wells, tail-lights and rear-view windows. Target position and orientation hypotheses are generated with a particle filter and tested against the input.
Del Bimbo et al. [2009] developed a surveillance-related application where a wall-mounted pan-tilt-zoom camera tracks multiple pedestrians. Motion trajectories are modeled and estimated with an extended Kalman filter and constant-velocity model. Targets are localized in a precomputed point cloud of sparse 3D landmarks.

Meuter et al. [2008] built a tracker capable of following multiple pedestrians from a car. People are detected in the input video by a standard pedestrian detector. Localization is done by assuming people's feet are on a ground plane shared with the tracker. Motions are predicted with an unscented Kalman filter implementing a constant-velocity model. The filter is extended with the motion state of the car. The extension places both the target and the tracker in a shared two-dimensional, ground-based coordinate system of relative motion.

Wojek et al. [2013] developed a system for tracking pedestrians and cars in complex traffic scenes. Like Meuter et al. [2008], the tracking camera is mounted on top of a car, with a known height and pitch angle relative to the ground plane that the targets are assumed to stand or sit on. Object detectors find and label the targets on a per-pixel basis and both detections and labels are processed in a Bayesian framework. It reasons both about the target's locations on the ground plane and whether a target occludes, or is occluded by, other targets.

2.1.2 Analysis

The approaches have aspects that will be used in the system developed in this thesis. In particular, the work of Del Bimbo et al. [2009] relates the tracking camera to, and places the targets in a map. Wojek et al. [2013] can track traffic and reason about its locations on the ground plane. Both Del Bimbo et al. [2009] and Meuter et al. [2008] employ Kalman filters to model target motion.

In the previous work outlined above, the targets are detected by either matching to CAD-models or using special-purpose (e.g. pedestrian) detectors. As a more general tracking mechanism is desired here, it is necessary to turn to a related topic, visual object tracking.

2.2 Visual object tracking

2.2.1 Introduction

This section presents some of the basic concepts of the 2D tracker component shown in figure 3.1. First, visual object tracking is introduced. Then a small number of trackers are selected and evaluated. Finally the best-performing tracker is picked as the 2D tracking component of the system.

2.2.2 Brief overview

The topic "Visual object tracking" attempts to solve the problem of visually following a moving object or target in an image sequence. It has received much
2.2 Visual object tracking

attention over the past 20 years so it is difficult to show a true and fair view that is also brief. Instead the following will focus on several trackers that can be considered state-of-the-art. All are based on a principle called tracking-by-detection, where a visual model of the tracking target is learned and then searched for in a video stream.

In a typical scenario, the target is first specified by either a user or another system placing a rectangular bounding box around it in the first video frame. The tracker generates its internal model of the target based on the appearance inside the bounding box.

In subsequent video frames, the tracker searches for image regions matching this model. This is done either in the entire frame or some region around the last known position, to save computational effort. It then moves the bounding box to the position where the target has been detected. Optionally, it updates its internal model, should the target have changed appearance.

2.2.3 Performance measures

Wu et al. [2013] benchmark several trackers and use two measures called precision and success to quantify the tracker performance on a number of test video sequences. Both measures require that the sequence is annotated with ground truth, a manually drawn bounding box tightly enclosing the target in every frame. Examples of these plots are shown in figure 2.2.

Precision

Precision is based on the center location error, illustrated in the left half of figure 2.1. In this figure, the ground truth bounding box and its center location are colored green. Similarly, the bounding box placed by the tracker and its center are colored blue. The center location error, labeled \( d \) in the figure, is the Euclidean distance in pixels from the center of one bounding box to the other.

The precision plot is a curve showing the fraction of frames in a test video sequence where the center localization error \( d \) is smaller than a location error threshold \( d_{thr} \), i.e. \( d < d_{thr} \). \( d_{thr} \). In the Wu et al benchmark, the fraction at \( d_{thr} = 20 \) pixels is also used to rank the different trackers. The median bounding box size across the sequences used in the benchmark is 56 by 81 pixels. In this context \( d_{thr} = 20 \) is equivalent to an error of less than 36\% of the height and 25\% of the width of a typical bounding box.

Success

Success is based on the overlap, illustrated in the right half of figure 2.1. This specifies the amount of overlap between the ground truth and tracker bounding boxes. If \( r_{gt} \) and \( r_t \) are the ground-truth and tracker bounding box regions, the overlap is given by

\[
S = \frac{|r_{gt} \cap r_t|}{|r_{gt} \cup r_t|},
\]

(2.1)
\(|r_{gt} \cap r_t|\) is the area of the intersection of the two regions, in number of pixels. Similarly, \(|r_{gt} \cup r_t|\) denotes the area of the union of the two regions. When both bounding boxes overlap perfectly, \(S = 1\). With no overlap, \(S = 0\).

\[ \]

**Figure 2.1:** Left: Tracker precision. Right: Tracker overlap.

The success plot is a curve showing the fraction of frames in a test video sequence where the area overlap \(S\) is greater than a threshold \(S_{thr}\), i.e. \(S > S_{thr}\). In the Wu et al benchmark, the trackers are ranked on the area under the curve of the success plots, so that larger area implies a better tracker.

### 2.2.4 Selection

**Method**

Many tracking methods have been developed over the years, in the search for an as accurate and robust method as possible. Fortunately, evaluation surveys appear regularly to define and compare the current state-of-the-art. The benchmark of Wu et al. [2013] presented above compares 25 trackers on 50 different test sequences. The Visual Object Tracking (VOT) challenge of 2013 (Kristan et al. [2013b]) evaluates 27 entries on 12 sequences. Seven of the trackers and most of the sequences overlap with the benchmark.

For practical reasons the choice is limited to the above mentioned surveys along with an additional tracker named ACT. This method is interesting to include, as it is developed at Linköping University by Danelljan et al. [2014]. It also closely reproduces the methodology of the Wu et al benchmark allowing direct comparison to the other approaches.

In total there are 46 methods to choose from and three criteria are set up to extract a manageable subset.

The first criterion is that a tracker should perform the actual tracking task well. To this end, a tracker must rank among the five best in either the Wu et al bench-
mark or the VOT challenge. The ACT must exhibit results that would have placed it in the top five if it had been part of either survey.

The second criterion is that the tracker source code must be available. This is both to simplify evaluation and allowing one of the 2D trackers to be integrated into the 3D tracking system later.

The third and final criterion is that a tracker must have real-time performance. This is an important requirement in any realistic application as it creates valuable computation headroom for other tasks. The lowest permissible frame rate is arbitrarily chosen as 20 frames per second for the Wu benchmark. The speed criterion is dropped for the VOT challenge however, as there is only one top-five contender with source code, and it runs at approximately 13 frames per second.

**Results**

Table 2.1 lists the trackers in Wu et al. [2013] ranked as overall top-five by the authors in any of their tests. Table 2.2 lists the top-five trackers of the VOT 2013 challenge, as ranked on the competition website (Kristan et al. [2013a]).

In both tables, "Ref" is the citation reference number in the benchmark paper, "Src" specifies whether source code is available for the tracker and "Mean FPS" is the average number of frames per second over the benchmark or challenge test sequences.

<table>
<thead>
<tr>
<th>Name</th>
<th>Ref</th>
<th>Src</th>
<th>Mean FPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Struck</td>
<td>[26]</td>
<td>✓</td>
<td>20.2</td>
</tr>
<tr>
<td>SCM</td>
<td>65</td>
<td>✓</td>
<td>0.51</td>
</tr>
<tr>
<td>ASLA</td>
<td>[18]</td>
<td>✓</td>
<td>8.5</td>
</tr>
<tr>
<td>TLD</td>
<td>31</td>
<td>✓</td>
<td>28.1</td>
</tr>
<tr>
<td>CXT</td>
<td>18</td>
<td>-</td>
<td>15.3</td>
</tr>
<tr>
<td>VTD</td>
<td>33</td>
<td>-</td>
<td>5.7</td>
</tr>
<tr>
<td>VTS</td>
<td>34</td>
<td>-</td>
<td>5.7</td>
</tr>
</tbody>
</table>

**Table 2.1:** The seven trackers ranked top-5 in at least one of the high-score success and precision plots in figure 3 of Wu et al. [2013]. See text for column legend.

<table>
<thead>
<tr>
<th>Name</th>
<th>Ref</th>
<th>Src</th>
<th>Mean FPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLT</td>
<td>[18]</td>
<td>-</td>
<td>169.59</td>
</tr>
<tr>
<td>FoT</td>
<td>46</td>
<td>-</td>
<td>156.07</td>
</tr>
<tr>
<td>EDFT</td>
<td>[13]</td>
<td>✓</td>
<td>12.82</td>
</tr>
<tr>
<td>LGTracker++</td>
<td>51</td>
<td>-</td>
<td>5.51</td>
</tr>
<tr>
<td>LT-FLO</td>
<td>30</td>
<td>-</td>
<td>4.10</td>
</tr>
</tbody>
</table>

**Table 2.2:** The top-5 average rank trackers of the Visual Object Tracking Challenge of 2013 (Kristan et al. [2013a]). See text for column legend.
Based on the selection criteria, the chosen trackers are

- Struck: Structured Output Tracking with Kernels (Hare et al. [2011])
- EDFT: Enhanced Distribution Field Tracker (Felsberg [2013])
- TLD: Tracking Learning Detection (Kalal et al. [2010])
- ACT: Adaptive Color Tracker (Danelljan et al. [2014])

![Performance plots of the ACT, EDFT, TLD and Struck trackers on 41 standard test sequences. Image courtesy of Martin Danelljan.](image)

The trackers are compared in figure 2.2 using the measures defined in section 2.2.3. It is apparent that ACT is the top performer, followed in order by Struck, EDFT and TLD. ACT is also the fastest method in this selection, capable of a median frame rate of 78.9 video frames per second. Finally, the source code is also available. Thus, both the second and third criteria are fulfilled. Beside the performance and speed advantages, the source code is compact and without external dependencies. Consequently, ACT is selected to be integrated with the system developed here.

### 2.2.5 The Adaptive Color Tracker

As was noted above, the ACT, or Adaptive Color Tracker by Danelljan et al. [2014] is very fast and performs well on many test sequences. The latter characteristic can be partly explained by its use of color information which sets it apart from most other trackers.

A naïve way of applying color is to operate on the red, green and blue components directly. However, a more involved method is used here. The usual RGB color of a pixel is transformed into an eleven-dimensional space by a highly non-linear function defined by Van De Weijer et al. [2009]. This encodes the likelihood that the color is one of eleven named colors as humans perceive them.

The speed is due to a cross correlation filter that performs the actual target tracking, efficiently implemented with the fast Fourier transform. When the coeffi-
cients of a correlation filter are convolved with an input signal, the tallest peak in the output indicates the point in space or time where the filter and input correlate maximally. In the ACT, this peak represents the most likely translation of the target image relative to the model. Both the model and target images are encoded with a representation described by Bolme et al. [2010], that was found to achieve good robustness against changes in illumination, scale and deformation.

A high-level description of the ACT algorithm follows below.

**The ACT algorithm**

**Input:** A sequence of images and a bounding box indicating the location of the target in the first frame.

**Output:** For each frame, a bounding box centered on the position where the target is most likely to be, and a confidence value.

**Repeat for all frames:**

1. Cut out an image tile centered on the bounding box. The size of the tile is larger than the bounding box and can be controlled by a padding parameter $p$. Pixels that fall outside the image are replicated from the edge.

2. Transform the RGB-colored tile to the color-name space and apply a Hann window. The windowing prevents spectral leakage in the Fourier transform in the next step. At the same time, the padding prevents the window from cutting away too much of the target appearance at the center of the tile.

3. Convert the tile to the representation of Bolme et al. [2010], in the frequency domain.

4. If this is the first frame, store the representation as the target model. Otherwise, convolve it with the existing model in the cross-correlation filter.

5. Transform the position of the correlation peak to a translation of the bounding box. The amplitude of this peak is the confidence in the tracking result where higher amplitude indicates greater confidence.

6. Update the model with the representation of the last image by linearly interpolating the two with a fixed learning rate $\lambda$.

7. Repeat from the beginning with the next frame in the video sequence.

Additional details concerning the implementation of this tracker are found in section 3.5 and the paper by Danelljan et al. [2014].

### 2.3 Mapping and localization

#### 2.3.1 Introduction

The previous section suggests possible methods for solving the 2D tracking problem. To extend this to 3D, the world must be mapped so that the tracker camera
can be placed in it. This section describes work related to both map building and camera localization, known as simultaneous localization and mapping, or SLAM. Specifically, when the sensor is a camera it is named visual SLAM.

Elsewhere in this thesis, the mapping and localization blocks are treated as separate processing stages. This division arises from using two different video sequences in the two tasks. In practice, localization is also performed as part of the mapping process. When a map is built, the camera poses of the mapping images are found at the same time.

2.3.2 Approaches

Map representations

A survey by Fuentes-Pacheco et al. [2012] outlines three kinds of representations; topological maps, occupancy grid maps and landmark maps.

Topological maps encode information about visited places and the paths between them as nodes and edges in a graph. This is suitable for navigation but cannot be used for precise positioning.

Occupancy grid maps quantizes the world into a grid of cells. Each cell can be either empty or occupied by an observed object like a tree or a building. Grid maps are suitable for path planning, but their voxel representations can require a lot of memory.

Landmark maps represent the world as sparse 3D clouds of landmarks. A landmark is a point in 3D space and two or more visual features that describe its appearance from different directions. This kind of map is not useful for navigation, but suitable for determining camera poses. It is also memory efficient.

Classes of SLAM methods

The survey of Fuentes-Pacheco et al. [2012] also finds three classes of solutions to the visual SLAM problem; biologically inspired, structure-from-motion and probabilistic filter systems.

Biologically inspired SLAM systems adopt strategies found in nature. One example is RatSLAM by Milford et al. [2004]. Based on a model of the rat hippocampus, it combines visual stimuli and wheel sensor data in a neuronal structure. The map representation is topological.

In structure-from-motion systems images are captured with a moving camera. Features are extracted and matched between images which associate them to the same landmark. The landmark positions are then triangulated while relative camera poses are estimated. Finally, the poses and landmarks are jointly fine-tuned or bundle adjusted. Bundle adjustment is a non-linear optimization that both finds optimal landmark positions and camera poses. The goal is to get the smallest possible deviation from the observed feature positions, when landmarks are projected back into the cameras. Examples of structure-from-motion systems are found in section 2.3.3.
Probabilistic filter SLAM replaces bundle adjustment with modelling of the uncertainty of landmarks and camera poses. One approach is MonoSLAM by Davison et al. [2007]. The initial feature matching and triangulation is similar to structure-from-motion, but landmarks and camera poses are encoded in the state of an extended Kalman filter. When either new landmarks are spotted or the camera moves, these are added to the filter as new state dimensions. When a new feature of an existing landmark is seen, the corresponding filter state dimension is updated.

**Analysis**

Localization of the tracker and target is an important feature for the 3D tracker. Thus, the best map representation appears to be based on sparse landmarks.

For the same reason, either the structure-from-motion or the probabilistic filter approach is feasible. The biologically inspired is not, at least not without accurate positioning.

The survey of Fuentes-Pacheco et al. [2012] notes that probabilistic filters are both sensitive to noise caused by incorrect matching and scales poorly. This leaves structure-from-motion as the best option.

**2.3.3 Structure from motion**

This section briefly describes some online structure-from-motion methods that may be suitable for deployment on a mobile platform such as an UAV.

One popular approach is Parallel Tracking and Mapping or PTAM by Klein and Murray [2007]. Originally developed for augmented-reality applications in small indoor areas. It uses a single moving camera to build a map of 3D landmarks as well as computing the associated camera poses. Each landmark is triangulated from sparse features seen by the camera in a number of carefully selected images or keyframes. As new camera images and features are captured and added to the map by one thread, another performs bundle adjustment on the map.

Strasdat et al. [2010] improves on PTAM with their OpenSLAM system by adding methods to control scale-drift that can cause problems with loop closure. Scale-drift is where the scale of new landmarks changes over time as new images are incorporated into the map. Loop closure is when the system recognizes that images from two points in time are of the same point in space and connects them in the map.

More recently, a system specifically for UAV navigation in outdoor terrain was described in papers by Weiss et al. [2011] and Wendel and Bischof [2013]. Unlike either PTAM or OpenSLAM, it creates an initial map offline and extends it in real-time. Offline building allows the distance between two points in the real world to be measured and added to the map, defining its scale.

A key feature is the creation of "virtual cameras". First a dense reconstruction of the mapped area is generated. Then a large grid of synthetic camera views are constructed, covering the mapped region. When the dense reconstruction is
projected into them, each view produces its own set of features from areas and angles where the mapping camera did not go.

This produces a large amount of additional data. The method clusters and quantizes the feature descriptors based on their similarity and place them in an index, called a vocabulary tree. This reduces the memory requirements and allows fast lookup of likely camera views near the view that must be localized. When used in the online localization, these additional images reduce the risk of the UAV getting lost.

### 2.3.4 Analysis

The method of Weiss et al. [2011] and Wendel and Bischof [2013] seems to be the best fit for this project as it permits precise real-time localization and mapping on a UAV. A major drawback is that no source code has been released. PTAM and OpenSLAM on the other hand, are both open source software.

However, it was determined early on that constructing or integrating a visual SLAM pipeline could be a large undertaking that should be avoided if other options exist. In this case, that option is VisualSFM (Wu [2011] and Wu [2013]), a GUI-based application for offline construction of a 3D map. It can process large photo sets and images do not necessarily have to be taken in sequence or even with the same camera. Given a set of images, it produces a point cloud of landmarks as well as feature sets (keypoints and descriptors) for each landmark. Critical to this project, the file formats are reasonably well documented.
This chapter outlines the actual system; the components and the methods used along with details such as parameter selection.

The general design philosophy is to create a straight-forward implementation. Many existing components are developed by others and by choosing simple methods for those that are written from scratch. At the same time, the system must obviously work properly and not be unusably slow.

### 3.1 Overview

A block diagram of the system is shown in figure 3.1. It consists of a number of different subsystems:

- The **map builder** shown in the top row of the diagram, constructs a representation of the area the 3D tracker operates in. This *map* is in the form of a 3D point cloud, with the points known as *landmarks*. The dashed line around this component indicates that it operates offline, i.e before the actual tracking starts. Related work and implementation details are found in sections 2.3 and 3.3.

- The **feature extractor** is shown on the left half of the second row. It detects structured areas (corners or blobs) in the video images, mark these as *key-points* and extracts *descriptors* from them. The combination of a keypoint and descriptor is known as *a feature*. This block works offline on tracking video sequences that were captured in advance. This choice is made for efficiency reasons while developing the system. In practical use, this block would necessarily have to operate online. A feature extractor is part of the
map builder and is therefore also described in sections 2.3 and 3.3.

- The localization block is found on the right half of the second row. It uses the features from the preceding step to localize the tracker in the map. The tracker position and orientation in the map is expressed as a camera pose matrix. Implementation details are found in section 3.4.

- The 2D tracker on the left half of the third row of the diagram, performs the task of finding and tracking the target as a two-dimensional object in the video sequence. It generates a bounding box around the target to indicate its position. Related work and implementation details are found in sections 2.2 and 3.5.

- Two space transformation blocks are shown on the right half of the third and fourth rows. The $\mathbb{R}^2 \rightarrow \mathbb{R}^3$ function converts the 2D bounding box into a 3D equivalent, locating the target in the map. The $\mathbb{R}^3 \rightarrow \mathbb{R}^2$ function is the inverse. Both blocks use the camera pose as well as an estimate of the ground plane for the transformations. Implementations of both transformations and ground plane estimates are described in section 3.6.

- The motion predictor on the bottom row of the block diagram takes over the tracking task from the 2D tracker if the target moves out of sight. It simultaneously directs the 2D tracker to the most likely area where the target might be found. Once the 2D tracker is confident that it sees the target again, it resumes control. A description of the motion predictor and details on how the system controls the 2D tracker are found in section 3.7.

To aid in the understanding of the system, the diagram will appear in later sections with the relevant subsystems highlighted.
3.2 The pinhole camera

The system uses the common pinhole camera model to represent the physical camera. It is defined as a point in space (the camera center) and an image plane. Rays of light are projected onto this plane from the world outside, in the direction of the camera center.

The coordinate systems used are depicted in figure 3.2. The basis vectors spanning the world frame are shown on the left. The positions of the tracker and camera, the target and the landmarks in the map are all defined using coordinates in this space.

![Figure 3.2: Coordinate systems. Left: World coordinate frame. Right: Camera coordinate frame](image)

The camera-local coordinate system is spanned by the basis vectors drawn in the camera frustum (pyramid) shown on the right in figure 3.2. The camera center is at the origin \((0,0,0)\) and the image plane is the plane \(z_c = 1\). The basis vectors \(x_c\) and \(y_c\) point right and down relative to the image plane, while \(z_c\) point toward it. The scale of the camera frame is the same as that of the world frame.

3.2.1 Extrinsic

Two concepts are used throughout this thesis; the camera pose matrix \(C\) and the camera projection matrix \(P\). Be careful to note that they are different from what is typically found in the computer-vision literature, like in Hartley and Zisserman [2003].

The camera pose \(C\) and its inverse \(C^{-1}\) are defined as

\[
C = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix}, \quad C^{-1} = \begin{bmatrix} -R^T & -R^T t \\ 0 & 1 \end{bmatrix}
\]  \( (3.1) \)

where \(R \in SO(3)\) is a rotation matrix and \(t \in \mathbb{R}^3\) a translation vector, together
known as the extrinsic camera parameters. Applying $C$ to a 3D point $X_c \in \mathbb{R}^4$ in normalized homogeneous coordinates in the camera’s local coordinate system with

$$X_w = CX_c,$$  

(3.2)

transforms the point to world coordinates. It can be thought of as a way to position and orient (or pose) the camera itself in the world frame.

Consequently

$$X_c = C^{-1}X_w$$  

(3.3)

transforms world points into the camera’s local coordinate system.

To project world points $X_w$ into the pinhole camera and onto the image plane, the points are are first transformed to the camera frame and then projected into the normalized camera $[I \ 0]$, where $I$ is a 3x3 unit matrix. The combination of these operations is the projection matrix

$$P = [I \ 0]C^{-1} = [R^T \ -R^Tt]$$  

(3.4)

### 3.2.2 Intrinsics and distortion

The intrinsic parameter matrix $K \in \mathbb{R}^{3 \times 3}$ deals with some imperfections found in any real camera; that the focal length is not 1.0, that the sensor is not centered on the optical axis and the image might be projected onto the sensor plane with a slight skew.

$$K = \begin{pmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{pmatrix}$$  

(3.5)

This system uses the simplified intrinsic matrix in equation 3.5. Here, the focal length $f$ is the only measured parameter. The center of the sensor is assumed to be aligned to the optical axis and consequently $c_x$ and $c_y$ are set to precisely half the image resolution. Also, skew is not considered.

The pixel coordinates $y_d \in \mathbb{R}^2$ in the camera image plane are related to normalized coordinates $x_d \in \mathbb{R}^2$ on the plane $z = 1$ as

$$x_{hd} = K^{-1}y_{hd}$$  

(3.6)

where $x_{hd}$ and $y_{hd}$ are the points on on homogeneous form. That is,

$$x_{hd} \sim \begin{pmatrix} x_d \\ 1 \end{pmatrix} \quad y_{hd} \sim \begin{pmatrix} y_d \\ 1 \end{pmatrix}.$$  

(3.7)

The subscript $d$ denotes that the points are distorted by the camera lens. This non-linear radial distortion is handled by the simple model

$$x_d = \left( 1 + k \|x\|^2 \right) x$$  

(3.8)
where \( x \in \mathbb{R}^2 \) is the undistorted and \( x_d \in \mathbb{R}^2 \) the distorted point on the image plane at \( z = 1 \). This equation is solved numerically to find \( x \).

In summary, the homogeneous coordinates \( y_{hd} \in \mathbb{R}^3 \) of a pixel in a video image is related to a point \( x \in \mathbb{R}^2 \) on the image plane \( z = 1 \) in the camera space with

\[
y_{hd} = \begin{pmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{pmatrix} \left(1 + k \|x\|^2\right) x
\]

(3.9)

This model precisely matches the design choices of VisualSFM. It also applies to both mapping and tracking, as features in the tracking videos are matched to features in the map.

### 3.3 Feature extraction and map building

The map helps the system place and orient the tracker. It consists of a set of photographs taken in the tracking area as well as salient features extracted from the photos.

VisualSFM extract feature keypoints and descriptors with the Scale-invariant feature transform, or SIFT (Lowe [2004]).

Mikolajczyk and Schmid [2005] showed that SIFT is robust against changes in viewing angle. This is a highly desirable trait in this case, as the tracking video sequence is likely to be captured from different angles from the images forming the input to the map.

The actual feature extraction and map building is a straightforward manual (off-line) process:

First the user captures either a mapping or a tracking video sequence and extracts them as a set of JPEG images.

These are then loaded into VisualSFM and the software is instructed to extract features from them. This is a relatively quick process if a GPU is used. With an Nvidia Geforce 760 GTX graphics card, VisualSFM generates 8,000 to 10,000 features from one 1280 x 720 pixel image in 220ms.

For tracking sequences, the process ends here. For mapping sequences, there are two more steps.

The first of these is to let VisualSFM match the features of every image against every other. This is a costly operation in terms of computing time. With \( n \) images to match, the complexity is \( O(n^2) \). For example, a map of \( n = 455 \) images, or \( 0.5n(n-1) = 103285 \) image pairs, need approximately seven hours of computing time on a i7 quad core workstation.
VisualSFM can be instructed exactly which images to match with one another to speed up the process, but this feature has not been used here.

In the final step, the point cloud and the mapping camera poses are reconstructed from the matched features. This typically takes a few minutes. To increase the quality of the output, the 3D points in the map are triangulated from features seen in at least three images.

### 3.4 Tracker map localization

The presentation up to this point has concerned the offline components of the system.

The next step in the processing pipeline is to place the tracker, or rather its camera in the map. A novel online method has been developed and is outlined below.

Briefly, the method has two main tasks: map lookup and pose estimation. The first task finds a map image that matches the camera image well. This establishes correspondences between features seen in the camera image and landmarks in the map. The second task uses the correspondences to estimate the camera pose that best aligns the features to the landmarks. This pose is then taken to be the location and orientation of the tracker camera.

A more detailed description of the method follows below:

### 3.4.1 The camera localization algorithm

**Inputs**

map: point cloud  

$I$: new image from the tracking sequence  
$C$: current tracker pose

**Output**

$C_{\text{new}}$: updated tracker pose or failure if none could not be found.

**The method**

1. Extract the tracker/camera world position $t$ from the camera pose $C$.
2. Sort the map camera views into a list $L$ according to their camera-center distances to $t$, shorter distances first.
3. Insert the last known map camera view at the top of the list $L$ so that it will be processed first.
4. For each map image $J$ in $L$, do the following:
(a) Match the features of the tracking camera image \( I \) with the features of the map camera view \( J \). This forms 2D-3D correspondences \( M \), linking the (2D)-features in the image \( I \) to (3D)-landmarks in the point cloud via the features in map image \( J \).

The matching is performed by OpenCV’s k-nearest-neighbor brute-force matcher on the GPU, with \( k = 2 \) and the query image from the tracker. \( k = 2 \) implies that for every feature in the tracker camera image, the two most similar features are picked in the map camera image. The similarity \( d \) is simply the Euclidean distance between two SIFT descriptors that in turn are encoded as vectors in \( \mathbb{R}^{128} \).

The two best matches have similarities \( d_1 \) and \( d_2 \), where \( d_1 < d_2 \). The ratio \( d_1/d_2 \) is computed to determine whether the best match with distance \( d_1 \) is unique. If \( d_1/d_2 > \alpha \) for \( \alpha = 0.8 \), the feature match is considered to be non-unique and is discarded. This is known as the nearest-neighbor distance ratio test, developed by Lowe [2004].

(b) Pass the 2D-3D correspondences \( M \) to perspective-n-point pose estimator, described in section 3.4.2. The estimator returns a camera pose \( C_{\text{new}} \) along with a count \( n \) of the correspondences that fit this pose.

(c) If \( n \geq 100 \) exit the method and return the new pose \( C_{\text{new}} \). Otherwise, treat the pose as bad and try the next map image in the list \( L \).

5. If the list is exhausted, the localization has failed. As a fallback it is then assumed that the tracking camera did not move.

This procedure is much more straightforward than the approach by Weiss et al. [2011] and Wendel and Bischof [2013] mentioned in section 2.3.3. A more elaborate method is not required as its main purpose is to serve as a testbed for the tracker. It still works well since the point cloud is much smaller and the tracker camera views are fairly close to the mapping camera views.

### 3.4.2 Perspective n-point estimation

Posing the camera in space is done by estimating the camera matrix \( \hat{\mathbf{P}} \) which aligns \( n \) 3D points in world space to the same number of 2D points in the camera image plane. In other words, when the projections \( \hat{x}_i \sim \hat{\mathbf{P}}X_i \) of the 3D points \( X_i \) line up with the observations \( x_i \), \( \hat{\mathbf{P}} \) will be the pose where the camera captured the image.

The minimal case, which is solved on closed form by finding the roots of a quartic equation, requires three 2D-3D point correspondences \( (x_i, X_i) \). This gives the estimator its name, the perspective-three-point method or P3P. However, only one of the four equation roots is correct. It is found by testing the roots against a fourth correspondence.

There are many P3P algorithms, the first was developed in 1841. The method used here is due to Kneip et al. [2011]. It is selected since it has available C++
source code and its authors claim a 15 times speedup and higher numerical stability over previous state-of-the-art methods. The P3P estimation is performed inside a RANSAC loop in order to find the camera pose that fit as many of all 2D-3D point correspondences as possible within the constraints specified by the given parameters.

RANSAC (Fischler and Bolles [1981]) is designed as a robust way of estimating a model that fit a set of data samples, even though some of portion of the samples are corrupted. There are many variations that improve on the RANSAC algorithm but this system implements the basic method, as it was found to work well on the test sequences.

The parameters that worked well in this system are

- RANSAC max iteration count, $n_{\text{max}} = 1000$
- RANSAC success probability, $p = 0.9999$
- Minimum reprojection error, $e_r = 2$ [pixels]

As a finishing step, the camera pose is adjusted with a nonlinear least-squares algorithm minimizing the total reprojection error

$$e_{\text{rtot}} = \sum_{(x,X) \in I} \|x - \hat{x}\|^2$$  \hspace{1cm} (3.10)

with $\hat{x} \sim \hat{P}X$. As this optimization considers all 2D-3D point correspondences at once, it is named the perspective n-point estimation or PnP.

### 3.5 Target tracking
Once the pose of the tracker camera is found, the next step is to track the target in 2D. Recall that the user provides the system with a bounding box outlining the target in the first frame of the tracking video. The 2D tracker begins by building a visual model of the target and proceeds to detect it in subsequent frames, moving the bounding box each time.

The system uses the ACT tracker described in section 2.2.5 extended with a small set of additional features. These were added to deal with scale changes and occlusions:

- The model update (i.e., learning) can be temporarily disabled. This is so that when an occlusion occurs, the tracker will not update the model with the appearance of the object covering the target.
- The bounding box size and position can be set externally at any time. This is to reflect the change in scale when the target-to-tracker distance varies, as well as allowing the motion predictor (see section 3.7) to direct the tracker when the target is occluded.
- The internal model size is fixed. Regardless of the size of the bounding box plus padding, the input tile is always resampled with cubic interpolation to 128 by 128 pixels. This allows the tracker to maintain its model even though the target bounding box changes in size over time.

### 3.5.1 Implementation notes

The tracker has been implemented to partially run on an Nvidia GPU with the help of Nvidia's CUDA language, CUDA components in OpenCV version 3.0 and the CUDA-based fast Fourier transform library CuFFT. The GPU implementation is partial, due to numeric problems that arise in the complete GPU implementation. Due to differing FFT data formats on the GPU and CPU sides, the root cause could not be determined within a reasonable time frame.

The tracker is configured with its default parameter values, with two exceptions. These are the padding and learning rate parameters:

The padding parameter $p \in \mathbb{R}$ controls the tile size by specifying the amount of space outside the bounding box that must considered by the tracker. If $w$ and $h$ are the width and height of the bounding box in pixels, then the padded size is $w(1 + p)$ by $h(1 + p)$ pixels. The default is $p = 1$, which effectively makes the tile twice as large as the bounding box. As is mentioned in section 2.2.5, the padding is needed since the tile is windowed to prevent spectral leakage when it is Fourier transformed. Without padding, too much of the target appearance would be weighted down by the window.

The learning rate $\lambda$ controls the speed at which the tracker adapts to a new target appearance, simultaneously forgetting the old. If the current model is $M_k$ and
the new appearance $A_k$, then $M_{k+1} = (1 - \lambda)M_k + \lambda A_k$. The default $\lambda = 0.075$ was found suitable for the test sequences used.

### 3.6 Target localization

In a two-dimensional tracker the target in the camera image is enclosed by a (typically) rectangular bounding box. It is only natural to translate this concept to 3D bounding box when moving to a three-dimensional space.

This section presents a method to estimate the ground plane from the point cloud map. Two heuristic methods for generating the 3D bounding box from its 2D counterpart and vice versa are also described.

The latter methods are somewhat limited in that they assume that the ground is flat and that the camera roll angle relative to it is small (or zero). However, it is sufficient for the test scenarios presented in chapter 4.

#### 3.6.1 Ground plane estimation

When the system starts, no ground plane is defined. Although the user can provide the plane coefficients in a configuration file, it is convenient to estimate it automatically. This estimation is performed in two parts.

The first part produces a candidate plane using a RANSAC-based procedure. Here, landmark triples are picked from the map point cloud and putative ground plane equations are generated from those.

In the second part, the ground plane with the largest inlier set is optimized to become the output of the estimator. This optimization is done by fitting the plane to the inlier landmarks using a procedure based on singular-value decomposition.

A more detailed description follows below:

1. Pick a landmark $X_0$ known to be on the ground from the point cloud set $Q$. This is done by specifying its index in the point-cloud in a configuration file.

2. Extract all points in $Q$ within a radius $r$ from $X_0$:
   \[ S = \{X_i : X_i, X_0 \in Q, \|X - X_0\| < r\} \quad (3.11) \]

3. In a RANSAC loop, repeat unconditionally 1000 times:
   (a) Randomly (with uniform distribution) draw three points $X_1, X_2, X_3$
from \( S \) and form a plane candidate \( \tilde{P} = (\hat{n}, d) \) with

\[
n = (X_2 - X_1) \times (X_3 - X_1) \tag{3.12}
\]

\[
\hat{n} = n / \|n\| \tag{3.13}
\]

\[
d = -X_1 \cdot \hat{n} \tag{3.14}
\]

(Less formally, the plane normal is the cross product of the two vectors \( X_2 - X_1 \) and \( X_3 - X_1 \) that lie on the plane.)

(b) Compute the perpendicular distances of the landmarks in \( S \) to the plane as \( d_i = \hat{n} \cdot X_i + d \). The landmarks closer to the plane than the mean distance, i.e., for which \( d_i < \frac{1}{|S|} \sum_{i=1}^{|S|} d_i \) form the inlier set.

When the loop completes it returns the largest inlier set \( T \subset S \). In the subsequent steps, fit a plane to the largest inlier set \( T \):

4. Set \( \mu \) as the mean coordinate in \( T \) and set up the matrix

\[
A = \begin{pmatrix} X_1 - \mu & X_2 - \mu & \ldots & X_{|T|} - \mu \end{pmatrix}
\] \tag{3.15}

where \( X_i \in T \) are column vectors in \( \mathbb{R}^3 \).

5. Perform singular value decomposition: \( U S V^T = A \)

The matrix \( U \in \mathbb{R}^{3 \times 3} \) can be interpreted as an orthonormal basis. Its first column is the direction of largest variance and the third the direction of least variance of the points in \( A \).

6. Consequently, select the direction of least variance as the normal vector, with

\[
\hat{n} = U_{:,3} \tag{3.16}
\]

This finally yields an estimation of the ground plane

\[
P = (\hat{n}, -\hat{n} \cdot \mu) \tag{3.17}
\]

### 3.6.2 3D bounding box estimation

The 3D bounding box has a shape of a rectangular cuboid. This is a 3D analog of the Cartesian rectangle that is the 2D bounding box. It both serves as a visualization tool and is used to define the size and location of the target in the world frame. Figure 3.4 illustrates the points on the bounding box, the camera and their relationships in space.

The 3D bounding box is directly generated from its 2D equivalent. Conceptually, the 2D bounding box can be thought of as being "pushed" away from the camera plane into the world, until its bottom corners touch the ground. At this position it becomes the front face of the 3D bounding box. This face is then duplicated to and moved away from the front to create the back face. It is positioned so that the footprint of the 3D bounding box is square.
Using more detail, the procedure to generate the 3D bounding box is as follows:

First, the bottom corners of the 2D bounding box are projected onto the ground. These projections mark the front face bottom corners of the 3D bounding box. The distance \( w \) between the corners is consequently the width of the bounding box.

The ground plane normal is then extended up a distance \( h \) from the bottom corners to generate the top corners. \( h \) is selected so that if the front face is reprojected into the camera, its height matches the 2D bounding box height.

The back face is subsequently extrapolated from the corners of the front face by extending the front face normal backward the distance \( w \).

Finally, the target center is defined as the centroid of the eight corners of the bounding box.

The procedure can also be expressed with mathematical notation:

1. With the ground plane

\[
P_g = (\hat{n}_g, d_g) \quad (3.18)
\]

project the bottom left and right corners of the 2D bounding box (\( x_1 \) and \( x_2 \) in figure 3.4) into \( P_g \) at \( F_1 \) and \( F_2 \), respectively.

The width of the 3D bounding box in the world frame is now

\[
w = \|F_2 - F_1\| \quad (3.19)
\]

2. Create the plane of the front face (i.e the face oriented toward the camera) using \( F_1 \), \( F_2 \) and the ground plane normal \( \hat{n}_g \). The plane is defined as

\[
P_{ff} = (\hat{n}_{ff}, d_{ff}) \quad (3.20)
\]
3.6 Target localization

with

\[ \hat{n}_{ff} = \frac{1}{w} (F_2 - F_1) \times \hat{n}_g \] (3.21a)

\[ d_{ff} = -\hat{n}_{ff} \cdot F_1 \] (3.21b)

3. Project the top left and right corners of the 2D bounding box into \( P_{ff} \) to yield \( F_3 \) and \( F_4 \).

The height of the 3D bounding box is then

\[ h = \|F_3 - F_1\| \] (3.22)

4. Finally, the corners of the back face \((B_1, B_2, B_3, B_4)\) and target center \( T \) are directly given by

\[ B_i = F_i - w\hat{n}_{ff} \] (3.23)

\[ T = \frac{1}{8} \sum_{i=1}^{4} (F_i + B_i) \] (3.24)

**2D bounding box mapping**

When the 2D tracker is guided by the motion predictor described in section 3.7, a reverse mapping from 3D back to a 2D representation is needed.

Reconstructing the 2D bounding box is a two-step procedure. The motion predictor only works with the target center positions and does not know about the 3D bounding box. Thus in the first step, the front face is placed on a line between the camera and target centers, parallel to the camera image plane. In the second step, the top-left and bottom-right corners of the face are projected onto the camera image plane. These projections then become the corresponding corners of the new 2D bounding box.

This mapping is described in more detail below, with the used entities shown in figure 3.4.

1. Project the camera pose \( C \) onto the ground and set \( c_f \) as the forward direction of the camera and \( c_r \) as the right direction. \( \|c_f\| = 1 \) and \( \|c_r\| = 1 \).

2. Project the target center \( T \) onto the ground at \( T_g \) and set

\[ t_f = -\frac{1}{2} wc_f \] (3.25)

\[ t_r = -\frac{1}{2} wc_r \] (3.26)

where \( w \) is the width of the 3D bounding box.

3. Compute the top-left and bottom-right corners of the front face as seen by
the camera, i.e \( F_3 \) and \( F_2 \) with

\[
F_3 = T_g + t_f + t_r + h\hat{n}_g \\
F_2 = T_g + t_f - t_r.
\]  

(3.27) \hspace{1cm} (3.28)

\( h \) is the height of the 3D bounding box and \( \hat{n}_g \) is the ground plane normal.

4. Finally, project \( F_3 \) and \( F_2 \) into the camera to create the top-left and bottom-right corners of the 2D bounding box.

Note that \( F_1 \) and \( F_4 \) are not used here, as the 2D tracker expects a Cartesian bounding box rectangle, i.e one that is aligned with the image coordinate axes. As long as the assumption that the camera roll angle relative to the ground is zero or small holds, the bounding box will be approximately correct.

### 3.6.3 Handling target scale changes

As an adaptation to the 3D tracking system, the 2D tracker must deal with that the target size changes with the distance from the camera, which the ACT cannot do in its original form.

If the tracking camera and target centers in frame \( k \), are located at world positions \( C_k \) and \( T_k \) respectively, the scale change is given by

\[
s = \frac{|T_k - C_k|}{|T_0 - C_0|}.
\]

\( s \) is used to scale the width and height of the bounding box determined by the user in frame \( k = 0 \), with the origin of the scaling at its center.

### 3.7 Motion modelling

When the target is occluded or moves out of the video frame, it is useful to model its movement so that its position can be predicted while not visible. While there are many choices of motion models and filters, this system implements the constant acceleration motion model in a standard Kalman filter, due to their simplicity.

#### 3.7.1 The constant acceleration motion model

When looking for ways to model motion, basic mechanics gives us the relationships between the position \( p(t) \), velocity \( v(t) \) and acceleration \( a(t) \) as

\[
p(t) = \int_0^t v(\tau)d\tau, \quad v(t) = \int_0^t a(\tau)d\tau
\]

(3.29)

The special case where the acceleration is assumed to be constant over the integration period is called the constant-acceleration (CA) motion model. Written on
differential form the CA motion model is
\[
\frac{dp(t)}{dt} = v(t), \quad \frac{dv(t)}{dt} = a(t), \quad \frac{da(t)}{dt} = 0 \quad (3.30)
\]

In the two-dimensional case, this equation system can be encoded into a six-dimensional vector \( x(t) \), with two dimensions each for its position, linear velocity and linear acceleration on a plane.

Following Li and Jilkov [2003], the differential motion model expressed in \( x(t) \) outlined below. Note that the matrix elements in these equations are 2x2 matrices, with \( I \) denoting the identity matrix. This permits two-dimensional quantities to be expressed more compactly. The continuous-time model is
\[
\frac{dx(t)}{dt} = \begin{bmatrix} 0 & I & 0 \\
0 & 0 & I \\
0 & 0 & 0 \end{bmatrix} x(t) \quad (3.31)
\]
and is discretized as
\[
x_{k+1} = Ax_k \quad (3.32)
\]
where
\[
A = \begin{bmatrix} I & TI & \frac{T^2}{2}I \\
0 & I & TI \\
0 & 0 & I \end{bmatrix} \quad (3.33)
\]
is the transition model from state (or time step) \( k \) to \( k + 1 \) and \( T \) is the sample rate. It is used to update the state as time moves forward, with increasing index \( k \).

The state is internal to the physical system and part or all of it might not be known, or observable. What is observed in this case, is the apparent position and velocity of the target on the ground plane, \( \hat{y} \).

The relation between observation and internal state, is
\[
\epsilon = \hat{y} - y = \hat{y} - Hx \quad (3.34)
\]
where
\[
H = \begin{bmatrix} I & 0 & 0 \\
0 & 0 & I \end{bmatrix} \quad (3.35)
\]
is the observation model, expressing that the target position and speed are what is observed.

In summary, the motion model is
\[
x_{k+1} = Ax_k \quad (3.36)
\]
\[
y_k = Hx_k \quad (3.37)
\]
3.7.2 The Kalman filter

The CA-model above assumes that both measurements and the internal state are perfectly known and disturbance free, which obviously is not true for any physical system. Adding two noise sources

\[ v_k \sim \mathcal{N}(0, Q) \in \mathbb{R}^6, \quad w_k \sim \mathcal{N}(0, R) \in \mathbb{R}^4 \]  

(3.38)

yields the model

\[
\begin{align*}
x_{k+1} &= Ax_k + v_k \quad (3.39) \\
y_k &= Hx_k + w_k \quad (3.40)
\end{align*}
\]

As time progresses, the state of this system is updated as follows, in two phases:

The first phase is the measurement update. When the 2D tracker has found the target in a new video frame, the 2D bounding box is converted to the world frame with the 2D-to-3D method of section 3.6. The center of the 3D bounding box is projected into the ground plane to become the observed target position \( \hat{y} \).

The difference \( \epsilon = \hat{y} - y \) between the observed target position \( \hat{y} \) and \( y \) derived from the state \( x \) is used to correct the filter’s state.

The state \( x \) and its covariance \( P \) are modified as

\[
\begin{align*}
\epsilon &= \hat{y}_{k+1} - Hx_k \quad (3.41) \\
S &= HPH^T + R_k \quad (3.42) \\
K &= PH^T S^{-1} \\
x_{k+1} &= x_k + K\epsilon \quad (3.44) \\
P_{k+1} &= P_k - KSK^T \quad (3.45)
\end{align*}
\]

The second phase is the time update which advances time one sample period. The filter state is also adjusted, to a prediction of where the target is most likely to be positioned if it had been observed. In this phase, the state and state covariance are modified as

\[
\begin{align*}
x_{k+1} &= Ax_k \quad (3.46) \\
P_{k+1} &= AP_kA^T + Q_k \quad (3.47)
\end{align*}
\]

Details on how Kalman filters are derived can be found in a reference text such as Statistical Sensor Fusion by Gustafsson [2010].

3.7.3 Controlling the Kalman filter

When the peak response of the detector function in the 2D tracker drops below a set threshold (0.5 was found to be good), this signals that the tracker cannot see the target in the current video frame.

At this point the tracker’s target model update is disabled. In practice, the model learning rate parameter defined in section 3.5.1, is set to zero, i.e \( \lambda = 0 \).
In addition, the measurement update is excluded in the Kalman filter as no new target observation was made. However, the time update is still performed and produces a prediction of the target position. This predicted position is converted to a 2D bounding box in the camera image frame with the 3D-to-2D procedure of section 3.6. The bounding box is passed back to the 2D tracker to indicate where it should look for the target in the next frame.

Due to the inevitable noise in the prior measurements, the predictions get more uncertain with every time update. If the target is out of view for too long, the system will likely lose track of it.

Finally, when the detector response rises above the set threshold, this signals that the target can be seen again. Now the control of the bounding box is handed back to the 2D tracker.

3.8 Visualization

3.8.1 Tracking

![Figure 3.5: 3D scene of the tracking area.](image)

To visualize the operation of the tracking system, an OpenGL based viewer application was developed where the user can navigate the tracking area with the mouse. It is also possible to see the current input image with the features matched to the map drawn on top.

Figure 3.5 is a screenshot of the 3D viewer, displaying the tracking area point cloud and the estimated ground plane as a white grid. The yellow-colored camera frustum of the current input frame is shown in the top left and the yellow 3D target bounding box can be seen to the right. Here, the front face of the bounding box is colored green. For reference, a photo of approximately the same view is shown in figure 3.6.
Finally, figure 3.7 depicts a current input image with currently matched features and the 2D bounding box. The insets are not part of the application, but added here for readability. The top inset shows a closeup of some keypoints matched to the map, drawn in purple. The bottom inset shows the 2D tracker bounding box. The green dot inside the box is the center of the 3D bounding box reprojected into this image. The blue bar at the bottom of the box, extending from left to right, shows the current tracker confidence which here is near 1.0.
3.8 Visualization

3.8.2 Image stabilization

The video captured with the hand-held camera always comes out somewhat shaky mostly due to walking around. As long as the shaking is limited in range, the tracking system is unaffected. Yet, even small amounts of instability can make it more difficult for an observer to see and reason about how the system performs.

However, since the tracker camera’s pose in world space is known, improving the situation is straightforward. This solution works on the premise that most camera shaking manifests as noisy rotations about its center. The video is stabilized by low-pass filtering the camera pose rotation and reprojecting the image into a new virtual camera. The output of this virtual camera is then what the user will see.

Pose filtering

Let $R_k$ be the "raw" 3x3 rotation matrix component of the camera pose $C_k$ and $\hat{R}_k$ the stabilized rotation, both in frame $k$. Also let $Q$ be the operator transforming a rotation matrix to a quaternion and $Q^{-1}$ its inverse. Then, $q_k = Q(R_k)$ and $\hat{q}_k = Q(\hat{R}_k)$.

The first step in the stabilizer is to smoothen a new rotation sample $q_{k+1}$ extracted from the camera pose, with an exponential moving average filter $\hat{q}_{k+1} = \text{slerp}(\hat{q}_k, q_{k+1}; \alpha)$. Slerp is the linear interpolation of the two quaternions with a blending factor $\alpha$, introduced by Shoemake [1985]. In this application, with a relatively slow-moving camera and a frame rate of 24 frames/second, a hard coded $\alpha = 0.2$ was found suitable.

Reprojection

The orientation change from the raw to the stabilized pose is given by the rotation $R_P = \hat{R}_k^{-1} R_k = Q^{-1}(\hat{q}^* q_k)$ where $\hat{q}^*$ is the complex conjugate of $\hat{q}$. Since no translation is involved, points on the raw image can be mapped to the stabilized image using the homography $H = KR_P K^{-1}$ with $\hat{x} = Hx$. Here, $K$ is the camera’s intrinsic parameters matrix, with $x \in \mathbb{R}^3$ and $\hat{x} \in \mathbb{R}^3$ the homogeneous coordinates in the raw and stabilized images, respectively. This reprojection is the second and last step of the stabilizer.

The net effect of these steps is a much smoother video sequence, with much less shaking.
This chapter details the experiments on the system; why they are performed, how they are set up, considerations that must be made and what the results are.

4.1 Setup

All video is captured with a Canon S95 compact camera with a global shutter, in 1280 by 720 pixels at 24 frames per second. The tracked target is a remote controlled car. Both are pictured in figure 4.1. Note that neither the camera nor the notebook computer mounted to the remote controlled car are used in this project. The red ball attached on the top of the vertical stick in the center of the car is used to estimate a reference position of the target.

Before tracking can begin, a map of the location must be constructed. Consequently, the first step is to record a video sequence of the area. One out of every
ten images, typically around 450 frames, are then extracted from the sequence and processed in VisualSFM with the procedure outlined in section 3.3. The application produces a point cloud of reconstructed landmarks and sets of camera positions and 2D image descriptors. Figure 4.2 shows the scene alongside the point cloud and reconstructed camera poses.

![Figure 4.2: Left: The testing area. Right: The point cloud map and mapping camera poses of the same scene in a birds-eye view.](image)

It was found to be important to have steady and smooth camera movements when capturing the mapping sequence. Panning or rotating the camera about its vertical axis too quickly will result in either motion blur or insufficient overlap between frames. When this happens, VisualSFM generates multiple, disjoint point clouds of the scene that cannot be used.

In conjunction with recording the mapping video, several tracking sequences are captured where the car is driven around the scene and followed by the hand-held camera. The first frame is hand-annotated with a bounding box drawn around the target. An example, with the bounding box drawn in yellow, is shown in figure 4.3.

![Figure 4.3: The first frame of a test sequence.](image)

### 4.2 Positioning experiments

One of the main features of this system is the ability to place both the tracker camera and the target in 3D space. It is therefore reasonable to test the accuracy of the camera poses and target locations.

#### 4.2.1 Test sequences

For these experiments there are two test sequences named "S-run" and "scale".

An overview of the sequences are shown in figures 4.4 and 4.5. Blue tracks and camera frustums represent positions and poses estimated with the methods of sections 3.4 and 3.6. Green tracks and camera frustums represent positions and
4.2 Positioning experiments

Figure 4.4: The "S-run" sequence, seen from above.

Figure 4.5: The "scale" sequence, seen from above.

poses estimated with the methods described in sections 4.2.2 and 4.2.3. Thin red connecting lines show the per-frame associations between both sets of estimates.

In "S-run", the target car is driven on an S-shaped path toward the tracker camera, which backs away to keep the target in view. This tests the accuracy while viewing the target up close and from different angles. The target has approximately the same two-dimensional size in the camera throughout the whole sequence. In figure 4.4, the S shape marks the target motion track on the ground, seen from above. The black and white trashcan in figure 4.2 can be seen in the bottom left corner. The inset shows a close-up of the start of the sequence. Both the target and the camera move left to right in this sequence.

In "scale", the car is driven away from the tracker camera which remains stationary. This tests the accuracy of the tracker as the target moves away. Consequently, the target shrinks in the camera view throughout the sequence. In figure 4.5, the target is shown moving from right to left, while the cameras are positioned in the small cluster to the right. The trashcan is on the bottom edge, slightly right of center.

4.2.2 Camera pose experiment

Generating reference poses

To determine the accuracy of the camera poses, a reference to compare the results to would be needed. However, no ground truth exists. In its place, the poses from the online system will be compared to camera poses generated by VisualSFM. This is reasonable given the lack of ground truth data.
In contrast to the online pose estimation method of section 3.4, VisualSFM matches features from more than one mapping image. Hence, it will possibly have more 2D-3D correspondences available for its pose estimation compared to the online localization system. In addition, all of its poses and landmarks are jointly optimized with bundle adjustment. These features indicate that VisualSFM is likely more accurate than the online method. At any rate, it cannot be worse.

In practice, a reference map generated by extending the original map with one out of every five images from the test sequence. This limitation is intended to prevent the test sequence from dominating the data going into building the map and possibly skewing the results.

**Results**

The results of the camera posing experiment show the absolute differences in orientation and position between the online and offline estimations. Figures 4.6 and 4.7 show a typical difference in camera orientations and positions in both sequences of less than 0.05 degrees and 3 cm, respectively.

![Graph showing orientation differences](image)

**Figure 4.6:** Orientation difference in "scale" (top) and "S-run" (bottom).

A deviation in the form of a spike can be seen in both figure 4.6 and 4.7 at frame 10 in the S-run sequence. It is likely due to the inevitable shaking caused by walking around holding the camera. It can also be observed in the inset in figure 4.4 as the pair of cameras with a relatively greater distance between them. Comparing the two camera trajectories in the inset, the deviation appears to be entirely on the trajectory of the online localization system, marked with blue cameras.
4.2 Positioning experiments

Analysis

These experiments reveal that both pose estimation methods produce very similar results. Although ground-truth is unavailable, it can be stated that the accuracy of the online camera pose estimator is on par with the map itself. Furthermore, the deviation observed in frame 10 of the "S-run" sequence hints at better noise immunity in VisualSFM. This is not surprising given that it estimates poses using more landmarks than the online localization method does, as was also noted above.

4.2.3 Target position experiment

This experiment aims to quantify how well the target position can be determined. To simplify the analysis, the measured entity is the distance $d$ from the camera center at $C$ to the target center projected into the ground at $T_g$.

It is assumed that the target’s proximity to the horizon is a large contributor to positioning errors. If the target is near the horizon, a small vertical movement in the image plane will be equivalent to a large target movement along the ground. This is made obvious in figure 4.8, showing that the camera to target distance $d = h/(\sin \beta)$. $h$ is the camera height above the ground plane and $\beta$ is the angle under the horizon line $HC$. When $\beta$ is small, the noise sensitivity is large since $\sin \beta$ is also small.

Generating reference positions

Similar to the camera positioning experiment, no ground truth target trajectory exists. Instead, reference positions are estimated by tracking the center of the red
ball attached to the car, shown in figure 4.1. This tracking is separate but uses the same system built to track the car itself. Nevertheless, the estimated position of the ball should be much more precise as the ball appears the same from every direction and is significantly smaller than the car itself.

**Results**

The graphs in figures 4.9 and 4.10 show the difference in distance between the two methods. In both figures, the leftmost graph shows the distance difference as a function of the frame number. The center graph plots the difference as a function of the angle $\beta$ under the horizon while the right graph plots it as a function of the camera-to-target distance $d$.

The estimates are assumed to be good as long as the difference between them is small. In this case "small" is arbitrarily chosen as half the width of the target, or 25 cm.

Figure 4.9 shows the absolute difference in camera-to-target distance between the two measure methods on the "scale" sequence. Recall that in this sequence, the car moves away from the camera, which remains stationary. The red markers indicate the boundary between "small" and "not small", where the difference grows larger than 25 cm. This occurs at the angle $\beta = 7.6$ degrees, equivalent to the distance $d = 11.2$ meters, as the target moves away.

Figure 4.10 similarly shows the distance difference on the "S-run" sequence. Here, the target moves towards the camera and turns left and right in an "S" pattern. Compared to the "scale" sequence, the distance $d$ between the camera and target is approximately the same throughout, between 3 and 4.5 meters as indicated by the rightmost graph. However, there is a large bias of around 20 cm between the two methods. In the leftmost graph of figure 4.10, this is shown to vary between 14 and 27 cm as the car turns.
4.3 Qualitative experiments

From the "scale" sequence it is clear the two measurement methods are in agreement with distances shorter than \( d = 11.2 \) meters, equivalent to angles greater than \( \beta = 7.6 \) degrees. However, the "S-run" sequence indicates that there is a systematic offset between the two methods. One possible cause is that the center of the 3D bounding box lacks a defined geometric relation to the red ball on the car.

**4.3 Qualitative experiments**

In this section, two qualitative experiments examine whether 3D information can improve the "spatial awareness" of a tracker. In both cases, the 3D system is compared to the Struck, EDFT, TLD and ACT trackers that were selected in section 2.2.4. As ACT is a component of the 3D tracker it is straightforward to compare 2D-only and 3D-enhanced tracking.
4.3.1 Target out of view experiment

The first experiment is depicted in figure 4.11 with selected frames from a video sequence named "out-of-view". It shows how both the 3D tracking system and the 2D trackers behave when the target moves out of the video frame. Here, the car is stationary while the camera pans away and then back to the target, from a different direction it left.

\[\text{Figure 4.11: Out-of-view sequence.}\]

The frames in figure 4.11 can be explained in more detail:

a) The first frame. All trackers have their bounding boxes on the target.

b) The camera is panning up. The bounding boxes of Struck and EDFT are clamped to the inside of the image, causing them to fail.

c,d,e) The camera has panned up, left and down.

f) The camera has panned right and the target enters the image. Only the 3D tracker is still locked onto the target.

g) The target is fully back in view.

h) The frame immediately after g. Here, the TLD tracker has snapped back onto the target, albeit with a smaller bounding box.

i) (Legend box)
4.3.2 Target occluded experiment

The second experiment is depicted in figure 4.12 with frames from a video sequence named "occlusion". This experiment is similar to the first, but aims to show how the 3D tracker handles occlusions. In this sequence, the car moves from left to right and is temporarily hidden behind a trashcan.

![Figure 4.12: The occlusion sequence. The frames are cropped and centered on the 3D tracker to improve readability.](image)

The frames in figure 4.11 are explained in more detail below.

a) The first frame. The tracker bounding boxes are identical.
b) All trackers are locked on target but only TLD and the 3D tracker have adapted to the change in scale.
c) The target is occluded. The 3D tracker maintains its bounding box (yellow, on the trashcan) where the target is most likely to be. The 2D trackers remain on the area where the target disappeared.
d) The target has reappeared. The 3D tracker is still locked on and TLD has reacquired its lock.
e) The target is turning right.
f) The TLD tracker is losing its lock.
g,h) The 3D tracker is locked on the target. TLD has started to "track the ground".
The other trackers are stuck to the trashcan.

In addition, the 3D bounding boxes corresponding to the target in frames c, d and e of figure 4.12 are seen from above in figure 4.13.

Figure 4.13: The 3D bounding box in the occlusion sequence. The top, middle and bottom bounding boxes on the right, correspond to the target in figure 4.12c, d and e.

A slightly more precise way of visualizing tracker behavior is shown in figure 4.14. This graph plots for each tracker how a distance measure varies over time. The distance is the number of pixels from the center of the tracker bounding box to the center of the video frame. When the trackers produce the same target position estimate, the corresponding plots overlap. When the trackers produce different results, the plots diverge. In other words, the relevant information is how well the tracker plots match each other, not the precise distances.

Figure 4.14: Occlusion sequence tracks.

The gray shaded section, covering frames 195 to 239, marks the part of the sequence where the target is considered occluded by the 3D tracker. From frame 1 to 195 the trackers agree. Beyond frame 195 the 3D tracker plot diverges from the others when the target is occluded. On frame 240 the TLD plot merges with the 3D tracker plot, as the target reappears.
4.3.3 Results

In the first experiment, Struck, EDFT and ACT fail once the target moves out of view while the 3D tracker and TLD succeed. Even though TLD is a 2D-only tracker, it can handle out-of-view situations and occlusions as it performs grid searches for the target across every input image.

The second experiment produces similar results; Struck, EDFT, ACT and TLD all fail once the car is occluded by the trashcan. The 3D tracker on the other hand keeps its bounding box moving behind the occluding object. Once the car reemerges, TLD snaps back onto the target and continues tracking.

The target prediction in the second experiment is not perfect however, since the Kalman filter drifts. This causes the bounding box jump from the predicted to the actual target position when the filter hands over control to the 2D tracker. In longer-lasting occlusions the tracker would likely fail since its effective image search area is limited.

This effect is too small to be apparent in figure 4.14, but is clearly visible in the two consecutive frames from the occlusion sequence shown in figure 4.15. In the left frame the target is occluded and the tracker is controlled by the Kalman filter. The bounding box is clearly misaligned on the target. In the right frame, the 2D tracker has assumed control and realigned the bounding box.

![Figure 4.15: Two cropped frames from the occlusion sequence. Left: Frame 239, target occluded. Right: Frame 240, target visible.](image)

4.4 Runtime performance

The computer used for these experiments has a 4-core Intel Xeon W3520 CPU running at 2.67 GHz with 8 GB RAM and an Nvidia GeForce 760 GTX GPU with 2 GB RAM.
### Table 4.1: Offline computation time breakdown.

<table>
<thead>
<tr>
<th></th>
<th>Items</th>
<th>Runtime / item</th>
<th>Percent</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Feature extraction</strong></td>
<td>101</td>
<td>(222 \text{ ms/image})</td>
<td>0.4</td>
</tr>
<tr>
<td><strong>Feature matching</strong></td>
<td>24839</td>
<td>((n(n - 1))/2) (240 \text{ ms/image pair})</td>
<td>98.5</td>
</tr>
<tr>
<td><strong>3D reconstruction</strong></td>
<td>279</td>
<td>(69604 \text{ ms/landmark})</td>
<td>1.1</td>
</tr>
</tbody>
</table>

#### 4.4.1 Offline

The offline computation detailed in section 3.3 is performed entirely by VisualSFM. Table 4.1 show the results of computing the map used with both the "scale" and "S-run" sequences. The image-to-image feature matching process clearly dominates, requiring approximately 7 hours to complete with VisualSFM using 3 out of 4 CPU cores. However, the feature extraction runs on the GPU and would likely take considerably longer had a CPU implementation been used.

#### 4.4.2 Online

The 3D tracker is implemented in C++ with OpenCV and CUDA and runs at a rate of 8 frames per second, or 124.7 ms per frame on the occlusion test sequence. Most of the system runs in a single thread on the CPU with some heavy computations, specifically the descriptor matching and the 2D tracker Fourier transforms, offloaded to the GPU.

![Figure 4.16](image)

Figure 4.16 outlines the amount of time spent on different tasks in the system. The map feature lookup and descriptor matching account for 56.2% of the work. As was mentioned in section 3.4, the feature lookup and descriptor matching is repeated until the system either finds a similar map image, or all map images have been searched. Therefore, the task complexity grows linearly in time with the number of images in the map.
This final chapter analyses the experimental results, the capabilities and limitations of the system. It concludes by returning to the questions posed in the introduction and suggesting possible improvements.

5.1 Discussion

5.1.1 Map building and localization

The tracker camera poses were estimated with both the online system and offline with VisualSFM. As was shown in figures 4.6 and 4.7 the methods are in good agreement. The orientation difference is less than 0.1 degrees while the positioning difference is typically 3 cm in the two test sequences used.

The difference between the methods that position the target is acceptable as long as the angle under the horizon is greater than approximately 8 degrees. The camera is held approximately 1.6 m above ground in the test sequences, meaning 8 degrees is equivalent to a tracker-to-target distance of 11.5 meters.

These numbers are easily scaled up if the system is to track a real car with the same apparent width and height in the camera view. The radio-controlled car is roughly 0.5 m wide seen from the rear and an ordinary four-door car is approximately 2 m. The difference in scale is thus 4. From this it follows that in real-world conditions the greatest usable distance is 45 meters with the camera mounted on a UAV hovering 6.4 m above the ground. Longer distances are workable if the camera is at a higher altitude. This would however require better optics and higher sensor resolution unless the target is proportionally larger.

The 3D bounding box model is a straightforward extension of the original 2D
bounding box, with only minimal additional data in the form of a ground plane and camera pose. It works well in the tests, where the assumption that the target is viewed from the side holds. However, two distinct limitations can be identified:

First, the assumption that the target is viewed from the side breaks down as the camera-to-target angle approaches 90 degrees. For example, this happens when the tracker is positioned right above the target and looks straight down. A simple workaround is to restrict the camera pitch angle. This will cause the target to disappear out of view well before this problem occurs.

Second, the rotation around the optical axis into the camera image frame, or roll angle, is assumed to be small. This is true in the test sequences where the ground is flat and the roll is typically less than 3 degrees. If the roll grows greater, the assumption that the bottom corners of the 2D bounding box are on the same image y-coordinate will be increasingly incorrect. Allowing the 2D bounding box to rotate in the image plane should help resolve the issue however.

5.1.2 Occlusions and out-of-view conditions

Unlike 2D trackers, the system exhibits well-functioning scale change, out-of-view and occlusion handling. This is entirely due to the modelling of the target position on the ground rather than the image plane. However, when the target is occluded, measurement noise will cause drift in the Kalman filter and possibly a loss of target lock.

This problem can be dealt with by searching a wider area in the image. One possibility is to extend the Kalman filter to search not just the mean position, but also points within the covariance ellipse of the ground position estimate. This is similar but not identical to using a particle filter, which is also an option. Another possibility is the approach found in the TLD tracker by Kalal et al. [2010]. Specifically, TLD uses a low-computational-cost target model and can search for the target in a grid across the whole image. This could generate candidate target positions for the 2D tracker.

5.1.3 Speed

The current system requires several hours to build a map using VisualSFM. Offline construction is acceptable in situations where the map can be built in advance and requires no subsequent changes.

The online part of the system is currently capable of a promising 8 frames per second, with slightly more than half the time spent on tracker localization. Realtime operation such as 25 frames per second might be attainable with certain changes to the design. Of primary concern is replacing the SIFT descriptors with a faster variant.

The simple linear search based on proximity in the tracker localization component can take a long time, but may be substituted by more intelligent methods such as the system of Wendel and Bischof [2013]. The Large-scale direct monocular SLAM system by Engel et al. [2014] is perhaps more interesting as it is avail-
able as open-source software from September 2014. Besides localization it builds maps online in real time, making VisualSFM redundant.

5.2 Conclusions and future work

Returning to the questions posed in section 1.2, it is apparent that a 3D extension does help the tracking problem since the target can be positioned with reasonable precision. Changes in scale, occlusions and out-of-view situations are handled very easily and computationally cheaply as a result.

The system might be capable of real-time processing with improvements to the localization and mapping components. These are the prime targets for optimization or re-engineering.

Most of the limitations have already been outlined above, but one additional restriction is that the ground is currently assumed to be flat. In reality, even the flattest terrain in nature has slopes, albeit very gentle ones. The most obvious approach is to generate separate ground plane estimates for different areas of the map.

Future work could primarily focus on addressing the limitations. For example

- replace the single ground plane with a terrain model, e.g. grid of locally flat patches,
- replace the current mapping and localization system with one of several faster and more robust implementations,
- improve the motion predictor using a particle-like or actual particle filter,
- possibly adding an image grid-search approach inspired by the TLD tracker as a fallback,
- generalize the 2D bounding box model so that the tracker can handle camera roll,
- improve the 3D bounding box model so that it can handle targets seen from above
- and finally a 2D tracker could be extended to maintain multiple models and choose the most likely based on the angle the target is seen from.
Bibliography


Michael Manz, Thorsten Luettel, Felix von Hundelshausen, and H-J Wuensche. Monocular model-based 3d vehicle tracking for autonomous vehicles in un-


**Upphovsrätt**

Detta dokument hålls tillgängligt på Internet — eller dess framtida ersättare — under 25 år från publiceringsdatum under förutsättning att inga extraordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för icke-kommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns det lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämnd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida http://www.ep.liu.se/

---

**Copyright**

The publishers will keep this document online on the Internet — or its possible replacement — for a period of 25 years from the date of publication barring exceptional circumstances.

The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for his/her own use and to use it unchanged for any non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its www home page: http://www.ep.liu.se/

© Andreas Robinson