Handling of Rolling Shutter Effects in Monocular Semi-Dense SLAM Algorithms

Lukas Tallund
Master of Science Thesis in Computer Vision

Handling of Rolling Shutter Effects in Monocular Semi-Dense SLAM Algorithms

Lukas Tallund
LiTH-ISY-EX--16/5016--SE

Supervisor: Mikael Persson
ITY, Linköpings universitet
Alper Aydemir
Volumental AB

Examiner: Per-Erik Forssén
ITY, Linköpings universitet

Computer Vision Laboratory (CVL)
Department of Electrical Engineering
Linköping University
SE-581 83 Linköping, Sweden

Copyright © 2016 Lukas Tallund
Abstract

Since most people now have a high-performing computing device with an attached camera in their pocket, in the form of a smartphone, robotics and computer vision researchers are thrilled about the possibility this creates. Such devices have previously been used in robotics to create 3D maps of environments and objects by feeding the camera data to a 3D reconstruction algorithm.

The big downside with smartphones is that their cameras use a different sensor than what is usually used in robotics, namely a rolling shutter camera. These cameras are cheaper to produce but are not as well suited for general 3D reconstruction algorithms as the global shutter cameras typically used in robotics research. One recent, accurate and performance effective 3D reconstruction method which could be used on a mobile device, if tweaked, is LSD-SLAM.

This thesis uses the LSD-SLAM method developed for global shutter cameras and incorporates additional methods developed allow the usage of rolling shutter data. The developed method is evaluated by calculating numbers of failed 3D reconstructions before a successful one is obtained when using rolling shutter data. The result is a method which improves this metric with about 70% compared to the unedited LSD-SLAM method.
Acknowledgments

A big thank you to my tutor at ISY, Mikael who have had the patience to see this thesis form although it materialized a bit slower than intended. You have always given me very honest feedback which have helped enormously!

Another big thank you to my tutor Alper at Volumental who have brought the algo brain game and C++ style guide perfection when needed! Stay true Souljah!

I’d also like to thank my examiner, Per-Erik, who also is the co-author of a lot of the methods used in the thesis. Thank you for the patience and all the great advice during the process.

Stockholm, November 2016
Lukas Tallund
Contents

Notation ix

1 Introduction 1
  1.1 Goal ......................................................... 3
  1.2 Approach .................................................. 3

2 Related Work 5
  2.1 Simultaneous localization and mapping [SLAM] ................. 6
  2.2 Introduction to VSLAM ....................................... 7
    2.2.1 Feature extraction and correspondences ................ 7
    2.2.2 Correspondence filtering ................................ 7
    2.2.3 Triangulation ........................................... 10
    2.2.4 Measurement certainty .................................. 11
  2.3 Introduction to rolling shutter sensors ........................ 13
    2.3.1 Rotation ................................................ 13
    2.3.2 Translation .............................................. 16
    2.3.3 Illumination changes .................................... 16
    2.3.4 Camera movement and distortions ....................... 16
    2.3.5 Rolling shutter sensor calibration ...................... 17
  2.4 Representations in image processing ............................ 18
  2.5 Feature based methods ....................................... 19
    2.5.1 Feature method example: PTAM .......................... 19
  2.6 Direct Methods .............................................. 22
    2.6.1 Direct method example: DTAM ........................... 22
  2.7 LSD-SLAM .................................................... 24
  2.8 Rolling shutter rectification ................................. 27

3 Method 29
  3.1 Suggested system ........................................... 30
  3.1.1 Implementation .......................................... 30
  3.2 Evaluation .................................................. 32
    3.2.1 Tracking stability ....................................... 32
    3.2.2 Trajectory comparison .................................. 32
## Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>RS</td>
<td>Rolling shutter</td>
</tr>
<tr>
<td>GS</td>
<td>Global shutter</td>
</tr>
<tr>
<td>SLAM</td>
<td>Simultaneous Localization And Mapping</td>
</tr>
<tr>
<td>VSLAM</td>
<td>Visual SLAM</td>
</tr>
<tr>
<td>MVSLAM</td>
<td>Monocular VSLAM</td>
</tr>
<tr>
<td>SfM</td>
<td>Structure from Motion</td>
</tr>
<tr>
<td>PTAM</td>
<td>Parallel Tracking And Mapping</td>
</tr>
<tr>
<td>DTAM</td>
<td>Dense Tracking And Mapping</td>
</tr>
<tr>
<td>LSD-SLAM</td>
<td>Large-Scale Direct SLAM</td>
</tr>
<tr>
<td>DSLR</td>
<td>Digital Single-Lens Reflex Camera</td>
</tr>
<tr>
<td>RGB</td>
<td>Red, Green and Blue (common camera color representation)</td>
</tr>
<tr>
<td>RANSAC</td>
<td>RANdom SAmple Consensus</td>
</tr>
<tr>
<td>ZSSD</td>
<td>Zero-mean Sum of Squared Differences</td>
</tr>
</tbody>
</table>
The visual sense was a very versatile sense for the early humans. It served us well in our quest to explore the world and investigate objects in it. Humans walked through the world looking at things and were later able to retell not only how things looked, but also where in the world they were. This proved to be a very important capability in our understanding of the world we live in.

In the last few decades we have been trying to teach machines to have the same capabilities. Having a robot that can move through an area and in the same time create a 3D map of it was a sought-after application. This is the basic idea of Simultaneous Localization And Mapping [SLAM]. SLAM is a general idea and data from a multitude of different sensors can be used as input. While specialized sensors like laser range sensors can be very effective, they are often very expensive and power consuming.

Therefore, the use of cameras as localization sensors has been a growing research topic in the last decade. Cameras are not only cheap but also give images that contain more information than range to different objects, which can be beneficial in computer vision applications such as 3D reconstruction. In these applications cameras with global shutter sensors are prominent. The global shutter sensor captures images by exposing the full sensor to the scenes light in the same instant. This creates an accurate image of the world in front of the camera. When using camera data together with the SLAM estimator the method is usually referred to as Visual SLAM [VSLAM]. A special case of VSLAM is monocular VSLAM [MVS-LAM] (also known as Structure from Motion [SfM]) which refers to when a single camera is used as input data, seen in Figure 1.1. A more technical introduction to VSLAM is presented in Section 2.2.
Figure 1.1: The general idea of the MVSLAM is to take images of an object from different views, and then estimate a 3D model of both the object and the camera positions that the images were taken from.

The drawback of global shutter camera sensors is that they are expensive compared to the rolling shutter camera sensor found in almost all consumer electronics that contain cameras, e.g. smart phones and DSLR cameras. The rolling shutter camera exposes each pixel row in the camera sensor in serial. This creates an equivalent image to the global shutter sensor if the camera is still and scene is not changed during the exposure. If there instead is changes or movement during the exposure, distortions will be created in the image which will create a non-accurate image of the world. An example can be seen in Figure 1.2 where the illumination has changed rapidly during exposure creating a striped pattern that is not present in the real world. A more technical introduction about rolling shutters are presented in Section 2.3.

These distortions are problematic in computer vision applications since they are not constant, but dependent on changes in the scene and camera movement. An interesting research topic would therefore be how to handle the distortions in a good way, without adding too much complexity to the algorithm. This thesis tries to handle these rolling shutter distortions when found in VSLAM pipelines.
1.1 Goal

Since the rolling shutter sensors are so prevalent in consumer products, it would be necessary to find a way to mitigate these movement dependent image distortions in some way before 3D scanning can become ubiquitous. The goal of the thesis is therefore to present a possible way of mitigating these image distortions in a MVSLAM algorithm. To do this, the following questions will create the base of the thesis:

- How much does rolling shutter data affect VSLAM results?
- What parts of the algorithm are sensitive to rolling shutter effects?
- Can these sensitivities be mitigated in some way?

1.2 Approach

The thesis analyses the algorithm LSD-SLAM as presented in Engel et al. [2014] and tries to find weaknesses in the algorithm that make it sensitive to rolling shutter data. This is done by using a dataset containing both rolling shutter and global shutter data. A suggestion of how to do alterations to the LSD-SLAM code is then given and implemented.
This chapter describes previously developed VSLAM and rolling shutter correction methods.

In Section 2.1, the idea of SLAM is introduced by stating general problems and some limitations.

In Section 2.2, the idea of VSLAM is discussed by a short introduction to the SfM pipeline.

In Section 2.3, the difference between global shutter and rolling shutter sensors is described. A few different distortion types from rolling shutter sensors are presented and the idea of estimating camera movements to handle these distortion is introduced.

In section 2.4, a comparison between feature based and direct methods of image processing is made.

In Section 2.5, 2.6 and 2.7, the SfM methods PTAM, DTAM and LSD-SLAM are described with focus on how they estimate camera movement.

In Section 2.8, rolling shutter rectification is introduced as a concept.
2.1 Simultaneous localization and mapping [SLAM]

SLAM is a computational problem where one tries to both estimate an agent’s location $x_t$ and at the same time estimate a map $m_t$ of its current surroundings. This is done based on sensor observations $o_t$ over discrete time steps $t$. Since the input and outputs are generally probabilistic the problem can be described as an optimization that finds the $x_t, m_t$ that maximizes the probability:

$$P(x_t, m_t | o_{1:t})$$  

(2.1)

If given some initial estimate of position, an initial map and transition function $P(x_t | x_{t-1})$, one could rewrite the probability for each of the two outputs as a sum of previous estimates using Bayes rule (Described in section A.2):

$$P(x_t | o_{1:t}, m_t) = \sum_{m_{t-1}} P(o_t | x_t, m_t) \sum_{x_{t-1}} P(x_t | x_{t-1}) P(x_{t-1} | m_t, o_{1:t-1}) / Z$$  

(2.2)

$$P(m_t | x_t, o_{1:t}) = \sum_{x_t} \sum_{m_t} P(m_t | x_t, m_{t-1}, o_t) P(m_{t-1}, x_t | o_{1:t-1}, m_{t-1})$$  

(2.3)

This thesis focuses on a special case of the SLAM estimation called VSLAM and some of the problems that this contains. No further introduction to the SLAM estimator will therefore be presented, but the resulting equations are simply indications of the possibility of SLAM itself. One can also note that previous observations of $P(m_t | x_t, o_{1:t})$ and $P(x_t | o_{1:t}, m_t)$ are considered static after estimation. This corresponds to the assumption of a static scene around the agent. This makes the SLAM estimation easier and is therefore considered true in this thesis. In the general case this is of course not a true statement and there are implementations of SLAM that takes this into consideration, e.g. Lee et al. [2010].
2.2 Introduction to VSLAM

As mentioned in the previous section VSLAM is a SLAM method that uses camera data as input to the SLAM estimator. Typically, any form of camera data could be used, e.g. IR-imaging data, stereo camera data, but in this thesis the focus will be on monocular RGB camera data which is captured by standard consumer cameras. If the scene in front of the camera can be considered static over time one could take multiple pictures of the object and from that estimate a 3D model of the scene. In MVSLAM, 3D reconstruction is done by comparing two frames in an image sequence and using the camera movement to create a stereo image. Monocular VSLAM is also called Structure from Motion [SfM]. To give an insight into how 3D reconstruction works this section will describe a general SfM pipeline. SfM is based around the notion that if a 3D point can be seen in multiple images and we know the transformations between the cameras positions, it is possible to estimate the depth of where the point is in 3D space in relation to the cameras.

2.2.1 Feature extraction and correspondences

To be able to estimate the camera position transformation between two images one needs to find a way to compare the two images. This can be done in multiple ways. Usually, this is done by extracting notable features from the two images and finding correspondences between the two images. To do this one could use a feature extractor, e.g. the Harris corner detector (Harris and Stephens [1988]). An example image can be seen in Figure 2.1 and the same image with extracted features in Figure 2.2. For each feature $\alpha_f$ found in the image, extract an image patch $P_f$ around the feature. A patch could for example be a square of pixels around the found feature. Using this image patch one can then search for a corresponding patch in a neighborhood $\Omega_i$ of the feature $\alpha_f$ in image $I_k$ in the other image $I_{k+1}$ by comparing some kind of similarity measurement. One example of a comparison method could be by a sum of the pixel-wise differences of intensity between the two patches. A patch with similar intensity values will be a probable correspondence. An example of matched features between two images can be seen in Figure 2.3.

2.2.2 Correspondence filtering

Not all of these corresponding features will be correct correspondences and will add errors if taken into consideration when doing the camera transformation estimate. Incorrect correspondences are called outliers. To handle outliers one typically uses RANdom SAmple Consensus [RANSAC] introduced in Fischler and Bolles [1981]. In RANSAC a number of $N$ correspondence pairs are selected randomly from the dataset, where $N << M$ and $M$ are the total number of data points. These hypothetical inliers, i.e. not outliers, are used to estimate a fitting model for the dataset. The fitting model is then used to evaluate the full dataset of $M$ points. The estimated model is then tested by applying it to the of the data in the set, e.g. all $M$ data points. Data points which are in line with the model
Related Work

Figure 2.1: An example of an input image of a package of fries to be used during the SfM algorithm.

Figure 2.2: An illustration of found Harris corners in the image in Figure 2.1 indicated by green rings.
are considered inliers and part of the consensus set. If the number of inliers in the consensus set for this model is compared to the consensus sets of previous estimated models and if a better result, e.g. more inliers than previously, has been achieved the new fitting model is saved. A new set of \( N \) random points is then selected and the process is iterated until a good enough estimate has been found. An example of a fitted model to data can be seen in Figure 2.4 In the SfM case, the estimated model usually is the essential matrix \( E_{kl} \), which represents the transformation between pose \( k \) and pose \( l \). The essential matrix is defined as:

\[
E = R[t]_x
\]  

(2.4)

where \( R \) is a 3 x 3 rotation matrix that describes the rotation between the two poses, \( t \) is a 3 x 1 translation vector between the two poses and \( [t]_x \) the matrix crossproduct operator of \( t \) (see Section A.1). As a note for further reference, this can be used to estimate a transformation in the Lie-group SE(3) (presented in A.5) which will be used in later methods.
2.2.3 Triangulation

Using the final estimated essential matrix $E_{kl}$ from RANSAC the transformation between the two cameras is estimated. Each feature position is then projected through the corresponding camera center creating what can be seen in figure 2.5. If an intersecting point can be found, it is added to the point cloud. An example of the final result of the point cloud is shown in Figure 2.6.
2.2 Introduction to VSLAM

Figure 2.5: An illustration of triangulation where the corresponding points \( y_i \) are projected along the line going through the corresponding camera center \( O_i \). The 3D point \( x \) that corresponds to the two 2D points are the intersection of the two lines. Image source: Public Domain

2.2.4 Measurement certainty

As often in image processing, there are multiple noise sources involved in each step of the 3D reconstruction. One could for example be that the measured features are not placed in the same place in multiple images due to limited resolution. To counter this, the 3D point estimates usually also contain a certainty measurement. This could be used to filter out points that have low certainty, or to merge points that seem to be very similar. To minimize the amount of uncertainty in the VSLAM algorithm, it is important to know as much as possible about the noise sources causing them. Some of them are constant and can therefore be calculated beforehand and be taken into consideration during the estimation. Therefore, one usually tries to find out as much as possible about the camera to make the result as accurate as possible.
Figure 2.6: The final result of the SfM algorithm run on the example dataset.
2.3 Introduction to rolling shutter sensors

Knowing the camera type and its parameters are a very important part in image processing. Camera parameters such as lens distortion, sensor format and focal length are usually obtained by doing a camera calibration, e.g. Zhang’s method introduced in Zhang [2000]. Zhang’s method results in a $3 \times 3$ upper triangular matrix, with 5 degrees of freedom, representing the internal camera parameters, notated by $K$. One camera parameter that Zhang’s method does not take into consideration is the shutter type. Today there are two major shutter types present in digital cameras: global shutter [GS] and rolling shutter [RS].

The most common type today in consumer electronics are the RS sensors, mostly due to being much cheaper than the GS sensors. The difference between these two is how the camera sensor is exposed. In the GS sensors the scene is exposed at one time for the full sensor. This captures a full image in the same instant for all sensor rows. In a RS camera sensor each row of the image sensor is given exposure to the world at different times. Because of this RS sensors are prone to image distortions due to environmental changes and camera movement since each row is an independent image. In VSLAM algorithms, each full image is generally considered to be a capture of a certain time $t$, which does not take into account the time difference between individually captured rows in the RS case.

Due to this sensitivity to changes, the RS sensors are undesirable for use in 3D reconstruction algorithms such as SfM. Since most consumer electronics uses RS cameras and the use cases for VSLAM and similar algorithms in our daily lives are increasing it would be interesting to add such RS awareness to VSLAM algorithms. There are multiple distortion types that originate from RS sensors. A few of these distortions are presented in Sections 2.3.1-2.3.3.

2.3.1 Rotation

Figure 2.7 is an illustration of RS distortion when rotation is present in the scene. The resulting image creates a pattern that smears the colors on the row that is present during the exposure. In Figure 2.8, one could identify a similar pattern due to the rotating rotary wings of the helicopter.
Figure 2.7: An illustration of a capture of a rotating colored object of a 24 row rolling shutter camera sensor. Each time step shows how the object is rotating to the left and the resulting image to the right. Image source: Cmglee (https://commons.wikimedia.org/wiki/File:Rolling_shutter_effect.svg), "Rolling shutter effect", https://creativecommons.org/licenses/by-sa/3.0/legalcode
Figure 2.8: An example of rolling shutter distortions where the rapidly moving rotors of the helicopter is distorted. Image source: Jonen (https://commons.wikimedia.org/wiki/File:Jamtlands_Flyg_EC120B_Colibri.JPG), "Jamtlands Flyg EC120B Colibri", https://creativecommons.org/licenses/by-sa/3.0/legalcode
2.3.2 Translation

Fast translation in the scene or of the RS camera creates a pattern where the resulting image is skewed since each captured row is subject to a small translation compared to the previous row.

Figure 2.9: An example of rolling shutter distortions where a rapidly moving car is present in the scene. Image source: Axel1963 (https://commons.wikimedia.org/wiki/File:CMOS_rolling_shutter_distortion.jpg), “CMOS rolling shutter distortion”, https://creativecommons.org/licenses/by-sa/3.0/legalcode

2.3.3 Illumination changes

If the illumination changes during the image capture, different parts of the image will have illumination distortions. An example is presented in Figure 1.2 in the introduction chapter.

2.3.4 Camera movement and distortions

In the general case a combination of these distortions is often present. One can note that all of the distortions are processes that are time dependent changes in the scene and/or camera movement. Due to the assumption that the scene is static made in section 2.1 one could therefore represent the changes by some transformation $\xi(t)$ which represents the camera movement during the image capture. To lessen the influx of these distortions it would therefore be interesting to estimate this $\xi(t)$ for each row in the image. In sections 2.5 and 2.6 a few SfM implementations are presented with focus on how this transformation is esti-
2.3 Introduction to rolling shutter sensors

In a RS sensor, each row is exposed to the environment in series. Typically, the row that is exposed first is the top-most and that will be the assumption in the thesis. In Figure 2.10, an illustration of a RS sensor with $N$ rows is shown. Each row $i$ starting from the top is exposed at time $t_i$, starting at $t_0 = 0$. Using the camera movement estimate $\xi(t)$ and $t_i$, one could therefore estimate the camera pose for when each row is captured. In Forssén and Ringaby [2010], the authors define these timings as below and derive them by using a method from Meingast et al. [2005].

**Rolling shutter sensor parameters**

Making the assumption that each row is exposed during a set time $t_r$ called the readout time, the $t_i$ can be calculated as:

$$t_i = i \cdot t_r$$  \hspace{1cm} (2.5)

The sensor sampling frequency $f$ is often stated from the hardware manufacturer and is considered known. The sensors sampling time $1/f$ should therefore be considered:

$$\frac{1}{f} = t_r + t_d$$  \hspace{1cm} (2.6)

In Hedborg et al. [2012], these are estimated for the camera sensors during their data collection which is used for evaluation of this thesis. $t_d$ in this expression is the rolling shutter sensors inter-frame delay which is the time between frames in the sequence when no row is exposed.

*Figure 2.10: An illustration of a rolling shutter sensor of $N$ rows where each row has a timing $t_i$. 

mated. In section 2.8, a method is presented for how the transformation estimate can be used to rectify the image and remove some of the distortions.


2.4 Representations in image processing

When doing digital image processing there are multiple approaches for how each image is represented. When comparing two images to each other, one would like to represent an image so that is possible to overlay one image on top of the other by using simple transformations (e.g. rotation, translation, skewing etc.) The most straightforward way would be using the pixel intensities for the full image to estimate these transformations.

Generally, the limiting factor in image processing is computing power. This motivates a representation that makes the computations as simple as possible. A typical approach to this is extracting features from the image that are easily distinguishable. This could for example be a corner of a table where the table has a different color than the background. It could also be a certain texture or derivative combination, all depending on the application.

Extracting features from an image could limit the image representation from millions of pixels to just a few hundred features or less. This approach is called sparse image processing or a feature based method. Although this adds feature extraction as another calculation step it generally is beneficial when one is trying to limit calculation complexity.

The problem with this approach is that while it limits the amount of calculations needed by creating a sparse calculation map, it also only gives a sparse result. Also, since only parts of the image is used for calculations, one could imagine that a better result could have been achieved if the full image was used instead. Imagine for example a VSLAM method based on corner features where the scene suddenly goes out of focus and the image becomes blurry. There are now fewer clear corners that are available for calculation, which results in lowered performance.
2.5 Feature based methods

As in the general SfM algorithm described in Section 2.2 most current VSLAM systems use features to do transformation estimations. This is largely done to, as mentioned in section 2.4, limit the amount of calculations needed. Another prominent advantage of using features would be that they are very specialized. One could therefore choose a feature that fits a specific application very well and by that lower the amount of noise in the reconstruction process.

2.5.1 Feature method example: PTAM

One of the most popular feature based methods is Parallel Tracking And Mapping [PTAM] introduced in Klein and Murray [2007]. The authors introduce a robust and fast VSLAM algorithm that differs from previous algorithms by separating the mapping and tracking of the SLAM algorithm in different threads. By separating the calculations in two threads, the vital tracking thread can handle the camera images in real time without lowering the mapping quality. This creates a system that both has very exact depth estimates from its camera frames, together with an optimized map that can be improved independently from the tracking thread. This section will describe PTAM in more detail.

Key frames

PTAM uses a key frame based approach where a few frames are selected as key frames to which other frames are compared. To represent the position and viewing angles of the camera for each frame, a pose $\xi_{a,b} \in SE(3)$ is introduced. $a$ and $b$ are camera coordinate systems and $\xi_{a,b}$ is the transformation from $b$ to $a$. The transformation group $SE(3)$ is described in Appendix A.5. Each frame consists of one such pose estimate and the camera image. The pose estimate of key frame $c$ is related to the world’s coordinate system origin $W$, described by $\xi_{c,W}$. Each normal frame $i$ will belong to a key frame $c$ and will be represented by a pose $\xi_{j,i}$.

Tracking thread

The tracking thread is responsible for the camera pose tracking. The frame image is warped by estimating an homography from an initial estimate of the pose $\xi_{j,i}$. The pose estimate is then refined in two stages, first by doing a coarse estimation on four different pyramid levels of the image and then a fine estimation on the full image. In each of the iterations a new pose is estimated from the found correspondences. From the new pose estimate, the original image is warped and used in the next iteration. The pose updates are done by minimizing a robust
objective function of the re-projection error, as follows:

$$\mu' = \arg \min_{\mu} \sum_{j \in S} \text{Obj} \left( \frac{|e_j|}{\sigma_j}, \sigma_T \right)$$

(2.7)

$$e_j = \left( \hat{u}_j \; \hat{v}_j \right) - \omega(\exp(\mu)\xi_{k,W}, p_j)$$

(2.8)

where $\xi_{k,W}$ is the camera to world estimate and $p_j$ is the original point. Also $\omega(\exp(\mu)\xi_{k,W}, p_j)$ is a warping function that warps the image point $p$ by the transformation $\xi_{k,W}$. The warping function is described in more detail in section appendix A.6. Finally the $\text{Obj}(\cdot, \sigma_T)$ is the maximum-likelihood estimator called Tukey’s biweight objective function introduced in Huber [1981].

Coarse correspondence estimation From the original image, an image pyramid is created by scaling the image by $(\frac{1}{2})^n$, $n \in [0,3]$. For each of the four pyramid levels the FAST corner detection (FAST is introduced in Rosten et al. [2010]) is used to extract features from the images and the 50 most important features are selected for future processing. Around each selected feature a matching template $W$ is created and used for a fixed radius search around the projected position in the key frame image by minimizing the zero-mean sum of squared differences error usually abbreviated ZSSD. In Skarbek [2003] it is defined as:

$$\sum_{(i,j) \in W} (I_1(x+i,y+j) - \bar{I}_1(x,y) - I_2(x+d_x+i, y+d_y+j) + \bar{I}_2(x+d_x, y+d_y))^2$$

(2.9)

Where $\bar{I}_k$ is the mean of $I_k$. The smallest error is selected as the location of the feature and is considered found if below a certain threshold.

Fine correspondence estimation After the coarse estimation step, a fine estimation is done. In this step the selected features is instead 1000 random features but the fixed radius search is done on a smaller area around the re-projection.

Mapping thread

The mapping thread works in parallel with the tracking thread and provides the map for the algorithm. When the tracking thread finds a new key frame candidate, the mapping thread decides if it should accept the candidate as a new key frame. Only key frame candidates that have a large enough camera center distance from other key frames in the map are added and key frames are only added as long as the tracking quality is good enough. If the candidate is accepted, the mapping thread extracts FAST corners in the key frame image and does an epipolar search in neighbouring key frames. All found correspondences in the key frame and neighbouring key frames are added to the 3D map by triangulation. When there are no new key frames from the tracking thread, the mapping thread can be used to do re-optimizations of the current map.
PTAM Results

The resulting maps from PTAM are good enough to use in simple augmented reality. As a use case example the authors fits a plane to the sparse points in the 3D map and use that to project 3D models onto the video stream.
2.6 Direct Methods

As mentioned in section 2.4, using a dense image map increases the amount of data that needs to be handled. Graphical processing units (GPU) are created to be able to do many parallel computations, something that usually is beneficial in image processing due to the independence of calculations between different features/pixels. This have paved the way for doing more dense image processing, as can be seen in Engel et al. [2013], Newcombe et al. [2011], Schöps et al. [2014].

In contrary to feature based methods this is based on comparing pixel intensities as described in 2.4. An obvious advantage is that the, sometimes, computationally heavy feature extraction step can be skipped, with the downside of having to do calculations for each pixel instead.

2.6.1 Direct method example: DTAM

The first algorithm that fully uses a dense map both for tracking and mapping in real time is presented by Newcombe et al. [2011] in "DTAM: Dense Tracking and Mapping in Real-Time". Newcombe et al. [2011] are later referenced as the authors in this section. DTAM continues PTAM’s idea of that the computations in MVSLAM are mostly independent of each other and can therefore be done in parallel. Just as PTAM, it is a key frame based method that uses multiple frames to improve a depth map that is estimated for each key frame. Adding GPU hardware to the do the direct calculations on the frames removes the need for feature extraction and therefore maintaining the tracking threads high speed constraints.

Direct alignment

To align each frame to its respective key frame the authors uses a transform $\xi_{k,c} \in SE(3)$ that is the transform from the current frame $c$’s camera coordinate system to the key frame $k$’s. But instead of extracting features and finding correspondences, the cost function $F(\Psi)$ is minimized:

$$F(\Psi) = \frac{1}{2} \sum_{p \in \Omega} \left( f_p(\Psi) \right)^2$$

$$f_p(\Psi) = I_l(\omega(\xi_{k,c}(\Psi)), \omega^{-1}(p, \rho_v(p))) - I_v(p)$$

$$\xi_{k,c}(\Psi) = \exp \left( \sum_{i=1}^{6} \Psi_i gen_{SE(3),i} \right)$$

Here $\Psi$ is a parametrization of the pose $\xi_{k,c}$, $\omega(\xi_{i,j}, I_k(p))$ is a warping function that warps the image point $I_k(p)$ by the transformation $\xi_{i,j}$. The warping function is described in more detail in section appendix A.6. Finally, $\rho_v(p)$ is the inverse depth estimate for point $p$ in the map and $gen_{SE(3),i}$ is the generating function for the Lie-group $SE(3)$. 
Map creation

To create the map from the previously mentioned inverse depth estimate $\rho_v(p)$, the authors calculate the average photometric error:

$$C_r(p, d) = \frac{1}{|I(r)|} \sum_{m \in I(r)} \| p_r(I_m, p, d) \|_1,$$  \hspace{1cm} (2.13)

and the photometric error $p_r$,

$$p_r(I_m, p, d) = I_r(p) - I_m(\omega(\xi_{m,r}, \omega^{-1}(p, d)))$$ \hspace{1cm} (2.14)

The inverse depth $d$ that minimizes $C_r(p, d)$ is chosen as the inverse depth estimate $\rho_v(p)$ for the pixel $p$.

**DTAM results**

The DTAM method creates a dense map that, similarly to PTAM’s, can be used for augmented reality. The authors also note that DTAM has a very high resilience to degraded images since it is not relying on any feature extractor. This means that no inherent weakness in those feature extractors have been taken into consideration. For example, PTAM’s FAST corner detection needs non-blurry images to extract corners, otherwise fewer features will be found and the performance is degraded. Since when using DTAM the entire image is used for pose estimation the condition of the image does not matter, as long as it is similarly degraded as the previous image.
2.7 LSD-SLAM

Large Scale Direct Simultaneous Localisation And Mapping [LSD-SLAM], introduced by Engel et al. [2014], is another direct method and is the chosen MVSLAM method for the thesis. LSD-SLAM is, similarly to the previously mentioned MVS-LAM methods, also based on key frames and uses a dense approach as in DTAM. Arguably, the largest difference from DTAM is the use of a semi-dense depth map where for each pixel in the dense map we decide if the pixel will be a good candidate for depth map triangulation and therefore only use the ones that are useful. An example of such a semi-dense map is shown in Figure 2.11 and the an example of a resulting 3D point cloud can be seen in Figure 2.12

Alignment

As opposed to DTAM and PTAM, LSD-SLAM uses the lie-group Sim(3) to represent it’s poses $\xi_{ji} \in Sim(3)$. The group is similar to SE(3) but also contains scale. Appendix A.5 describes Sim(3) in more detail. Similarly to DTAM, the cost function for alignment is a norm of photometric errors. In LSD-SLAM, the error is also variance-normalized to take illumination changes into account:

$$E_p(p, \xi_{ji}) = \sum_{p \in \Omega_{Dj}} \left\| \frac{r_p(p, \xi_{ji})}{\sigma^2_{r_p}(p, \xi_{ji})} \right\|_2$$

$$\sigma^2_{r_p}(p, \xi_{ji}) := 2\sigma^2_I + \left( \frac{\partial r_p(p, \xi_{ji})}{\partial D_i(p)} \right)^2 V_i(p)$$

where the norm $\| \cdot \|_\delta$ is the Huber norm which is described in the Appendix A.4. The photometric error is calculated between a warped version of the image $I_j$ and the key image $I_i$.

$$r_p(p, \xi_{ji}) := I_i(p) - I_j(\omega(p, D_i(p), \xi_{ji}))$$

Here, the warping function $\omega(p, D_i(p), \xi_{ji})$ warps each image point in image $j$ to image $i$. The warping function is described in more detail in Appendix A.6.

Depth map estimation

Just as in PTAM and DTAM, the camera motion estimate is used for refining the 3D depth map of the key frame. If the motion is too large to give a good baseline a new key frame is initiated at the current frames position. The depth map refinement is done by multiple per-pixel small-baseline stereo comparisons. To decide if a pixel is a good candidate for stereo update, the observation variance of the inverse depth is used:

$$\sigma^2_{d,obs} := \alpha^2(\sigma^2_{\lambda(\xi,\pi)} + \sigma^2_{\lambda(I)})$$

If the observation variance is below a certain threshold, the pixel is suitable for stereo triangulation and the pixel is added to the depth map. Each of the three parameters in the equation are presented and motivated below:
Figure 2.11: An example of LSD-SLAM’s debug window. Black and white areas are not used for depth estimation. In the current frame these areas correspond to shadows and a gray concrete edge which is too monochromatic and therefore contains low amounts of texture which can be used for triangulation. The map’s color indicates the estimated distance from the camera where objects close to the camera are red, the intermediate green and objects furthest away are blue.

Figure 2.12: An example of the resulting map in LSD-SLAM’s output viewer. The estimated key frame poses are shown in red and the resulting point cloud in gray scale intensities.
26        2 Related Work

Geometric disparity error $\sigma_{\lambda(\xi,\pi)}^2$: describes that a positioning error on the epipolar line causes a small disparity error if the epipolar line is parallel to the image gradient, and otherwise large, according to

$$\sigma_{\lambda(\xi,\pi)}^2 := \frac{\sigma_i^2}{\langle g, l \rangle^2}$$  \hspace{1cm} (2.19)

where $g$ is the normalized image gradient, $l$ the normalized epipolar line and $\sigma_i^2$ the variance of the disparity positioning error.

Photometric disparity error $\sigma_{\lambda(l)}^2$: describes that small image intensity errors have large effect on estimated disparity if the image gradient is small, and is otherwise small, according to

$$\sigma_{\lambda(l)}^2 = \frac{2\sigma_i^2}{g_p^2}$$  \hspace{1cm} (2.20)

where $g_p$ is the gradient of $I_p$ and $\sigma_i^2$ is the variance in image intensity noise.

Pixel to inverse depth ratio $\alpha$: Is inversely proportional to the length of camera translation, according to

$$\alpha := \frac{\delta_d}{\delta_\lambda}$$  \hspace{1cm} (2.21)

where $\delta_d$ is the length of the searched inverse depth interval and $\delta_\lambda$ is the length of searched epipolar line segment.
2.8 Rolling shutter rectification

As mentioned in section 2.3, using the estimated transformation $\xi(t)$ from the SfM methods could be used to un-distort the image. Forssén and Ringaby [2010], in this section referred to as the authors, use the pinhole camera model to derive a series of transformation matrices for rectification of the distortions when used in a feature based SfM method. The pinhole camera model can be written as follows:

$$x = K[R|d]X$$ (2.22)

where $R$ and $d$ are the rotational and translational parts, respectively of $\xi$. The matrix $K$ is the intrinsic camera parameter matrix, $X$ is a 3D point and $x$ is the projection of that 3D point onto the camera plane. Since the transformation $\xi(t)$ is dependent on time $t$ in the rolling shutter case the transformations dependent on $\xi(t)$ also depend on $t$ so the equation then becomes:

$$x = K[R(t)|d(t)]X$$ (2.23)

Since the projection can be written:

$$x = KR(t)D(t)X$$ (2.24)

where $D$ = \begin{pmatrix} 1 & 0 & d_1 \\ 0 & 1 & d_2 \\ 0 & 0 & d_3 \end{pmatrix} (2.25)

where all the $d_i$ are also dependent on $t$. As a rolling shutter image can be seen as multiple images taken at different time instances, rectification can be seen as projection points from an image taken in time $t_2$ onto an image taken in time $t_1$. Therefore the projection can be written:

$$x = KR(t_1)D(t_1)D^{-1}(t_2)R(t_2)^TK^{-1}y$$ (2.26)

where $y$ is a point in the rolling shutter image and the time instances $t_1$ is the time instance where the corresponding GS picture was taken and $t_2$ is the time instance when the row where the point $y$ is captured in the rolling shutter image. The authors then set $t_1 = 0$ and $t_2$ to the time the row where $y$ was exposed.
In this chapter, the methods that are implemented are described.

In Section 3.1, the implemented system is described.

In Section 3.2, the evaluation method is described.
3.1 Suggested system

The suggested solution is based on Hedborg et al. [2012] where the authors do rolling shutter rectification on image data for use in a VSLAM algorithm. Their VSLAM algorithm is a sparse, feature based method where they use the same features to calculate both the image rectification and the VSLAM algorithm. Since the chosen VSLAM algorithm in this thesis is a direct dense method, there are slight modifications to Hedborg et al. [2012] algorithm.

3.1.1 Implementation

As described in section 2.7, LSD-SLAM updates its pose estimate iteratively during the alignment step. Since LSD-SLAM assumes GS images, the estimate will be harder to do on RS images.

Pixel projection

To make LSD-SLAM handle RS data better, each pixel in the image should be warped according to the movement from the start of the RS image to the corresponding row in the image. To do this, the equation 2.26 is used. The equation states that:

\[
x = K R(t_1) D(t_1) D(t_2)^{-1} R(t_2)^T K^{-1} y
\]

(3.1)

If translation is not taken into account, i.e. \( D(t) = I \) \( \forall t \), the expression can be written:

\[
x = K R(t_1) R(t_2)^T K^{-1} y
\]

(3.2)

The camera parameter matrix \( K \) is known in this expression. In LSD-SLAM, the transformation \( \xi_{ji} \in \text{Sim}(3) \) between the key frame and the current frame is estimated.

Pose interpolation

Since the transformation \( \xi_{ji} \) is between the key frame and the current frame, a SLERP interpolation (introduced in Shoemake [1985]) is done between the start pose \( I \) and \( \xi_{ji} \) just as in Forssén and Ringaby [2010], resulting in \( \xi_{ji}(t) \). From \( \xi_{ji}(t) \), one can extract the \( R(t) \) and \( D(t) \) matrices for the whole transformation. The matrices \( R(t_1), R(t_2) \) and \( D(t_1), D(t_2) \) can then be sampled from these.

Estimating timings

The camera movement during the exposure of the frame is what affects the RS distortions in the frame. In a GS frame, all rows are exposed at the same time. To warp the RS frame into something that is close to a GS estimate, a single time is selected onto which all rows in the frame are projected, called \( t_f \). To keep time calculations as simple as possible in this implementation, \( t_f \) is selected as
the time when the first row of the frame is exposed compared to the current key frame $F_k$, denoted

$$t_f = N \cdot (r_{total} \cdot t_r + t_d),$$

(3.3)

where $N$ is the number of frames since $F_k$, $t_r$ is the read out time, $t_d$ is the inter-frame delay of the RS sensor and $r_{total}$ is the total number of rows in the image. A time estimate for each row $t_{row}$ is then calculated as

$$t_{row} = t_f + \left( \frac{r_{row}}{r_{total}} \right) \cdot t_r,$$

(3.4)

where $r_{row}$ is the respective row number. Note that $t_{row=0} = t_f$.

**Per-row projection**

Combining the results from the previous chapters, it is possible to write a per-row projection as follows:

$$x = KR(t_f)D(t_f)D(t_{row})^{-1}R(t_{row})^T K^{-1} y$$

(3.5)

Since the time dependent parts of this are constant per row, one only has to calculate the warp one time per row, but apply it per pixel. The new image is then fed to LSD-SLAM to use as a new frame.
3.2 Evaluation

To compare results in VSLAM algorithms, one could look at many different aspects of the results. The most obvious method from a mapping perspective would be to compare the generated map to some kind of ground truth map. In reality this proves to be difficult since the ground truth in this case also would be the 3D map of the object that has its own errors. A more practical solution would be comparing the estimated tracking trajectory generated by the algorithm to a ground truth trajectory.

To generate a ground truth trajectory, multiple solutions could be used. One could use some advanced tracking equipment as suggested in Weiss [2012] (e.g. Vicon tracking or similar). This equipment is expensive and sets limitations on where the datasets can be recorded since it might not be possible to setup such a system everywhere. Another problem with this approach would be that one also would measure the errors in the SLAM method. Since this study only focuses on the differences between the global shutter and rolling shutter data, this would not be a good approach.

In this thesis, two different methods of evaluation will be used. One that compares both LSD-SLAM and the modified version of LSD-SLAM tracking stability compared to the GS case when used with rolling shutter data, this is presented in section 3.2.1. The other compares the two case’s trajectory compared to the global shutter case and is presented in 3.2.2.

3.2.1 Tracking stability

Since LSD-SLAM is non-deterministic and often loses tracking on the RS sequences a statistical measurement on stability is done. To check how often this happens a set number of successful runs $N$ are set. A run where LSD-SLAM do lose tracking during the image sequence is considered successful. Loss of tracking occurs in LSD-SLAM when not enough data are available in the part of the 3D map for which the frame is to be aligned. For each required successful run $i$ the number of iterations $r_i$ is counted until the successful run is achieved. An average and standard deviation for the number of successful runs is calculated per method.

$$\mu_{\text{runs}} = \frac{1}{N} \sum_{i=1}^{N} r_i$$  \hspace{1cm} (3.6)

$$\sigma_{\text{runs}} = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (r_i - \mu_{\text{runs}})^2}$$  \hspace{1cm} (3.7)

3.2.2 Trajectory comparison

In Hedborg et al. [2012] a method is suggested that uses trajectory comparisons to evaluate their dataset. In the study, a trajectory estimated from GS data is used as ground truth. Since their dataset is also used for evaluation it seems suitable to
3.2 Evaluation

also use the same method for evaluation. In this chapter the method is described in more detail together with some implementation details.

Data extraction

The SLAM method returns key frame graphs where each key frame is represented as a 3D pose $\xi_k \in SE(3)$ where $k$ is the frame number of the image that the key frame represents. To compare trajectories, the translation part $\bar{t}_k$ of $\xi_k = (R_k, t_k)$ is extracted together with the index $k$ of the data point. This is done for both the global shutter data set, later referred to as the ground truth data set, $\{\bar{y}_k\}$ and the rolling shutter data set $\{\bar{x}_{k'}\}$. Note that the indices $k$ and $k'$ do not always have correspondences since key frame generation is non-deterministic.

Interpolation

Since the SLAM method is non-deterministic, the key frames are not always created in the same positions on both trajectories $\{\bar{y}_k\}$, $\{\bar{x}_{k'}\}$. This is represented by the $k'$ index of $\{\bar{x}_{k'}\}$. To handle this a Piecewise Cubic Hermite Interpolation [PCHIP] is done on $\{\bar{x}_{k'}\}$ and then sampled in each frame index $k$ that corresponds to an index existing in $\{\bar{y}_k\}$ to create $\{\bar{x}_k\}$ where each $\bar{y}_k$ corresponds to $\bar{x}_k$. An example of the interpolation method can be seen in Figure 3.1.

Figure 3.1: An example of the interpolation. The original data $\{\bar{x}_{k'}\}$ (blue circles) is used to create the orange trajectory. The trajectory is then sampled at the sample points corresponding to $k$ (red stars) to create the $\{\bar{x}_k\}$. 
Here the PCHIP is used since it creates somewhat natural camera motion patterns by not taking to much earlier movement into account when interpolating later parts, e.g. humans do not generally continue shaking the camera just because it shakes when they lift it from a table.

**Aligning the trajectories**

Since there is scale ambiguity in the trajectories, due to the limitations in MVS-LAM methods, one must also estimate how the trajectories relate to each other before calculating the error. To do this, one needs to estimate the similarity transformation ($\xi \in Sim(3)$) that minimizes

$$\min_{s,R,i} \sum_{k=1}^{N} ||\bar{y}_k - sR(\bar{x}_k - \bar{i})||^2,$$

(3.8)

where $R, i$ and $s$ are the rotation, translation and scaling of $\xi$. To estimate these parameters one could use Horn’s quaternion-based method from Horn [1987] which is summarized in the following sections.

**Calculating centroids** To estimate each part of the transformation it is beneficial if the two point sets are in approximately the same coordinate system. A good way to do this is removing the mean of each set, usually referred to as centering the data. This is done by calculating the centroids of the two point sets:

$$\bar{x}_0 = \frac{1}{n} \sum_{k=1}^{n} \bar{x}_k, \quad \bar{y}_0 = \frac{1}{n} \sum_{k=1}^{n} \bar{y}_k$$

(3.9)

After this is done two new sets are created by removing the mean from the datasets and thereby making sure the new sets have their centroids in the origin.

$$\bar{x}_k' = \bar{x}_k - \bar{x}_0, \quad \bar{y}_k' = \bar{y}_k - \bar{y}_0$$

(3.10)

**Estimating rotation $R$** These new datasets $\{\bar{x}_k'\}$ and $\{\bar{y}_k'\}$ are then used to estimate the rotation $R \in SO(3)$. Horn’s method estimates the rotation as a quaternion $q$ which is then converted to $R$ as follows

$$R = \begin{bmatrix}
1 - 2q_i^2 - 2q_j^2 & 2(q_jq_k - q_kq_i) & 2(q_iq_k + q_jq_r) \\
2(q_jq_k + q_kq_i) & 1 - 2q_i^2 - 2q_k^2 & 2(q_kq_i - q_jq_r) \\
2(q_iq_k - q_jq_r) & 2(q_jq_r + q_kq_i) & 1 - 2q_i^2 - 2q_j^2
\end{bmatrix}$$

(3.11)

**Estimating translation $i$ and $s$** After estimating the rotation $R$, the translation $\bar{i}$ can be calculated. Horn [1987] does this as follows

$$\bar{i} = \bar{b}_0 - R\bar{a}_0,$$

(3.12)
where \( \bar{a}_0 \) and \( \bar{b}_0 \) are the previously calculated centroids of the datasets. Lastly the scale \( s \) is calculated.

\[
s = \frac{\sum_{k=1}^{N} (\bar{x}'_k)^T R \bar{y}'_k}{\sum_{k=1}^{N} (\bar{y}'_k)^T \bar{y}'_k}
\]

(3.13)

All of these can now be combined to a similarity transformation:

\[
\xi = \begin{bmatrix} sR & \mathbf{i} \\ 0 & 1 \end{bmatrix},
\]

(3.14)

which can be used to calculate the aligned trajectory \( \{\hat{x}_k\} \) by transforming each \( \bar{x}_k \), according to

\[
\hat{x}_k = \xi \bar{x}_k
\]

(3.15)

**Calculating the error**

To give a good estimation of how much the two trajectories differ, and have a good comparable error between different runs of the SLAM method, a geometric error is used which is defined in Hedborg et al. [2012]. For each data point in the ground truth set, one calculates the area of the two triangles that is formed between itself and points with indexes \( k \) and \( k + 1 \), visualized in Figure 3.2.
Figure 3.2: An illustration of the two triangles that are calculated for the error estimation.

Each triangle’s area is calculated by Heron’s formula which is shown in Figure 3.3. The formula is presented to the left and the lengths of the triangles edges $a, b$ and $c$ are visualized the triangle to the right. To simplify notation a function $\text{tri}(\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3)$ is introduced. This function first calculates the edge lengths between the points ($\mathbf{p}_1, \mathbf{p}_2$ and $\mathbf{p}_3$) and then returns the total area of the triangle they form. All the areas of the triangles are then added and form the final error $\varepsilon$.

$$\varepsilon = \sum_{k=1}^{N-1} \text{tri}(\mathbf{y}_k, \mathbf{x}_k, \mathbf{x}_{k+1}) + \text{tri}(\mathbf{y}_k, \mathbf{y}_{k+1}, \mathbf{x}_{k+1})$$ (3.16)

$$A = \sqrt{s(s-a)(s-b)(s-c)}$$
$$s = \frac{a + b + c}{2}$$

Figure 3.3: Heron’s formula calculates the area of the triangle to the right according to the equations to the left. Image source: Public Domain
Single and mean trajectory comparison

Since LSD-SLAM is non-deterministic, a unique trajectory is generated each time it is run. To be able to compare trajectories one therefore needs to do a statistical analysis of the trajectories. This is done in two ways on the same data.

**Single trajectory comparison**  In the single trajectory comparison, a number $N$ trajectories are generated for each method that needs to be evaluated and the ground truth method. Each generated trajectory from the method under evaluation is then compared against each ground truth trajectory by applying the method in section 3.2.2. From these $\varepsilon_{i,\text{method}}$ one can then estimate a mean error and standard deviation of the error for each method.

$$\mu_{\varepsilon_{\text{method}}} = \frac{1}{N} \sum_{i=1}^{N} \varepsilon_{i,\text{method}} \quad \text{(3.17)}$$

$$\sigma_{\varepsilon_{\text{method}}} = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (\varepsilon_{i} - \mu_{\varepsilon_{\text{method}}})^2} \quad \text{(3.18)}$$

**Mean trajectory comparison**  In the single trajectory comparison a number $N$ trajectories are generated for each method that needs to be evaluated and the ground truth method. For each method an average trajectory is calculated. The averaged method trajectories are then compared to the ground truth trajectory as described in section 3.2.2 giving a resulting $\varepsilon_{\text{method}}$. 
To test the new implementation the datasets introduced in Hedborg et al. [2012] are used. The dataset contains 36 sequences where a rig containing a global shutter camera and a rolling shutter camera are moved along a path. Out of these 36 one is chosen for evaluation. Calibrations for the two cameras are given in the dataset including estimates for $t_r$ and $t_d$ for the RS camera which makes it possible to use the suggested method on these datasets. This chapter describes the experiment setup.
4.1 Selecting a sequence

Since LSD-SLAM is randomly initialized and non-deterministic its convergence relies on that enough frames are available in the sequence. To be able to measure how well the implementation actually works, a sequence where LSD-SLAM performs well on the GS case should be selected so it is easier to determine failed tracking as an effect of RS artifacts rather than a hard sequence for the SLAM method itself. In Hedborg et al. [2012] most sequences are around 13 seconds. These also contain a few seconds of frames which are used to align the RS and GS streams. Since these are to be cut away from the actual sequences the results are often too short to get LSD-SLAM to converge to a reliable result. There are three sequences above 20 seconds when cut, and out of these, sequence 29 is the most stable in the GS case. Alignment between the two streams in the sequence is done by the operator of the camera rig snapping finger in front of both cameras. This creates an audible cue to where the sequences are in sync. Any frame before this are cut from the sequences. The selected sequence, number 29 in Hedborg et al. [2012], is simply referred to as the sequence below.

4.2 Parameters

LSD-SLAM has an extensive configuration file which controls many aspects of the algorithms. If not specifically mentioned, the LSD-SLAM configuration is kept as out of the box. The used parameters are presented in Table 4.1. The parameters which were added during this thesis are described in the section below.

Rolling shutter correction (-R) Activates the rolling shutter correction in LSD-SLAM.

Rolling shutter readout time ($t_r$, kRsReadOutTime) Sets the RS readout time for the used camera sensor.

Inter-frame delay ($t_d$, kRsInterFrameDelay) Sets the RS inter-frame delay for the used camera sensor.

Pause on Key Frame (kRsPauseOnKeyFrame) Pauses the SLAM after each key frame change. Used for debug.

4.3 Sequence setups

The sequence is run with three different setups. These are described in the sections below. Each stream in the sequence is run with the corresponding camera calibration file.
4.4 Experiments

Table 4.1: Parameters available

<table>
<thead>
<tr>
<th>Configuration option</th>
<th>Flag/Variable name</th>
<th>Value set</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Calibration file</td>
<td>-c &lt;file&gt;</td>
<td>Depends on sequence</td>
<td>string</td>
<td>Specifies the file which contains the camera calibration of the camera used per sequence.</td>
</tr>
<tr>
<td>Input folder</td>
<td>-f &lt;folder&gt;</td>
<td>Depends on sequence</td>
<td>string</td>
<td>Set input folder containing the frames of the sequence.</td>
</tr>
<tr>
<td>Output file</td>
<td>-o &lt;file&gt;</td>
<td>Depends on experiment</td>
<td>string</td>
<td>Set output name of file containing estimated key frames.</td>
</tr>
<tr>
<td>Rolling shutter correction</td>
<td>-R</td>
<td>Depends on experiment</td>
<td>bool</td>
<td>If set, rolling shutter correction is active during the run.</td>
</tr>
<tr>
<td>Image frequency</td>
<td>-h</td>
<td>0</td>
<td>int</td>
<td>Sets unlimited number of iterations in frame pose optimization.</td>
</tr>
<tr>
<td>Rolling shutter readout time ((t_r))</td>
<td>kRsReadOutTime</td>
<td>0.0032</td>
<td>float</td>
<td>Specifies the rolling shutter read out time if rolling shutter correction is on.</td>
</tr>
<tr>
<td>Inter-frame delay ((t_d))</td>
<td>kRsInterFrameDelay</td>
<td>0.004</td>
<td>float</td>
<td>Specifies the rolling shutter inter-frame delay if rolling shutter correction is on.</td>
</tr>
<tr>
<td>Pause on Key Frame</td>
<td>kRsPauseOnKeyFrame</td>
<td>Used during debug</td>
<td>bool</td>
<td>Pauses on key frame selection.</td>
</tr>
</tbody>
</table>

4.3.1 Global shutter sequence [GT]

In this setup, the global shutter stream is used as input to LSD-SLAM without any options changed from the default values. Rolling shutter correction is off.

4.3.2 Rolling shutter sequence [LSD-SLAM]

In this setup, the rolling shutter stream is used as input to LSD-SLAM without any options changed from the default values. Rolling shutter correction is off.

4.3.3 Rolling shutter sequence with rectification [LSD-SLAM-RS]

In this setup, the rolling shutter stream is used as input to LSD-SLAM with rolling shutter correction activated. The rolling shutter correction needs the sensor readout time \(t_r\) and inter-frame delay \(t_d\).

4.4 Experiments

Two different experiments are done with the sequence. These are described below.

4.4.1 100 trajectories

In this experiment all three cases are run a total of 100 runs. The estimated trajectories are recorded as is without consideration of how many frames have been processed. This is done to evaluate how good the resulting trajectories of the two evaluation methods, LSD-SLAM and LSD-SLAM-RS, are compared to the GT case.
4.4.2 100 trajectories of successful runs

In this experiment all three cases are run until a total of 100 successful runs are done. This is done to evaluate how stable the tracking is for the two evaluation methods, LSD-SLAM and LSD-SLAM-RS, compared to the GT case.
In this chapter, the resulting numbers of the experiments described in Section 4.4, using the different measurements described in Section 3.2, are presented.

In Section 5.1, the results of the experiment in Section 4.4.1 are presented.

In Section 5.2, the results of the experiment in Section 4.4.2 are presented.
5.1 100 trajectories

This section covers the results of the experiment explained in Section 4.4.1.

5.1.1 Single trajectory comparison

When running the single trajectory comparison described in Section 3.2.2 on the 100 trajectories estimated from this experiment, the following results are obtained.

Table 5.1: Resulting numbers for single trajectory comparison on 100 iterations.

<table>
<thead>
<tr>
<th>Method</th>
<th>$\mu_\varepsilon$</th>
<th>$\sigma_\varepsilon$</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSD-SLAM</td>
<td>2.811497e-01</td>
<td>1.568001e-01</td>
</tr>
<tr>
<td>LSD-SLAM-RS</td>
<td>2.034002e-01</td>
<td>3.411518e-02</td>
</tr>
</tbody>
</table>

The values are visualized in Figure 5.1.

5.1.2 Mean trajectory comparison

When running the mean trajectory comparison described in Section 3.2.2 on the 100 trajectories estimated from this experiment, the following results are obtained.

Table 5.2: Resulting numbers for mean trajectory comparison on 100 iterations.

<table>
<thead>
<tr>
<th>Method</th>
<th>$\varepsilon$</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSD-SLAM</td>
<td>1.639921e-01</td>
</tr>
<tr>
<td>LSD-SLAM-RS</td>
<td>9.938141e-02</td>
</tr>
</tbody>
</table>

The values are visualized in Figure 5.2.
Figure 5.1: The average error for the two methods when doing single trajectory comparison to the GT trajectory. The mean is indicated with the bars and deviation with the fences.

Figure 5.2: The average error for the two methods when doing mean trajectory comparison to the GT trajectory.
**Trajectory comparison**

The following pictures visualize the estimated and calculated trajectories produced during this experiment. The trajectories start close to origin in the plot and end towards the lower right corner.

- Figure 5.3 displays the estimated ground truth trajectories which are used in this experiment.
- Figure 5.4 displays the estimated LSD-SLAM trajectories which are used in this experiment.
- Figure 5.5 displays the estimated LSD-SLAM-RS trajectories which are used in this experiment.
- Figure 5.6 displays the calculated mean trajectories which are used in the mean trajectory comparison.
Figure 5.3: The ground truth trajectories from which the mean is calculated. Total tracking failures are filtered to give a reasonable scale.

Figure 5.4: The LSD-SLAM method trajectories from which the mean is calculated. Total tracking failures are filtered to give a reasonable scale.
Figure 5.5: The LSD-SLAM-RS method trajectories from which the mean is calculated. Total tracking failures are filtered to give a reasonable scale.

Figure 5.6: A comparison of the mean trajectories.
5.2 100 trajectories of successful runs

This section covers the results of the experiment explained in Section 4.4.2.

5.2.1 Tracking stability

When running the tracking stability described in Section 3.2.1, the following results are obtained.

Table 5.3: Average and standard deviation of number of iterations per successful run per method.

<table>
<thead>
<tr>
<th>Method</th>
<th>( \mu_{\text{runs}} )</th>
<th>( \sigma_{\text{runs}} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>GT</td>
<td>1.3300</td>
<td>0.6039</td>
</tr>
<tr>
<td>LSD-SLAM</td>
<td>2.5900</td>
<td>2.0454</td>
</tr>
<tr>
<td>LSD-SLAM-RS</td>
<td>1.5400</td>
<td>0.9366</td>
</tr>
</tbody>
</table>

The values are visualized in Figure 5.7.

Figure 5.7: The average number of iterations needed for a successful run for the three methods. The mean is indicated with bars and deviation with fences.
5.2.2 Single trajectory evaluation

When running the single trajectory comparison described in Section 3.2.2 on the 100 trajectories estimated from this experiment, the following results are obtained.

Table 5.4: Resulting numbers for single trajectory comparison on 100 iterations of successful runs.

<table>
<thead>
<tr>
<th>Method</th>
<th>$\mu_\varepsilon$</th>
<th>$\sigma_\varepsilon$</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSD-SLAM</td>
<td>2.134927e-01</td>
<td>3.922384e-02</td>
</tr>
<tr>
<td>LSD-SLAM-RS</td>
<td>2.184509e-01</td>
<td>3.029994e-02</td>
</tr>
</tbody>
</table>

The values are visualized in Figure 5.8.

5.2.3 Mean trajectory evaluation

When running the mean trajectory comparison described in Section 3.2.2 on the 100 trajectories estimated from this experiment, the following results are obtained. The values are visualized in Figure 5.9.

Table 5.5: Resulting numbers for mean trajectory comparison on 100 iterations of successful runs.

<table>
<thead>
<tr>
<th>Method</th>
<th>$\mu_\varepsilon$</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSD-SLAM</td>
<td>9.241975e-02</td>
</tr>
<tr>
<td>LSD-SLAM-RS</td>
<td>1.275805e-01</td>
</tr>
</tbody>
</table>
Figure 5.8: The average error for the two methods when doing single trajectory comparison to the GT trajectory in the reiteration case. The mean is indicated with bars and deviation with fences.
<table>
<thead>
<tr>
<th>Method</th>
<th>LSD-SLAM</th>
<th>LSD-SLAM-RS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avarage error</td>
<td>0.1</td>
<td>0.15</td>
</tr>
</tbody>
</table>

**Figure 5.9:** The average error for the two methods when doing mean trajectory comparison to the GT trajectory for the sequence in the reiteration case.
Trajectory comparison

The following pictures visualize the estimated and calculated trajectories used in this experiment. The trajectories start close to origin in the plot and ends towards the lower right corner.

- Figure 5.10 displays the estimated ground truth trajectories which are used in this experiment.
- Figure 5.11 displays the estimated LSD-SLAM trajectories which are used in this experiment.
- Figure 5.12 displays the estimated LSD-SLAM-RS trajectories which are used in this experiment.
- Figure 5.13 displays the calculated mean trajectories which are used in the mean trajectory comparison.
Figure 5.10: The ground truth trajectories from which the mean is calculated in the reiteration case.

Figure 5.11: The LSD-SLAM method trajectories from which the mean is calculated in the reiteration case.
5.2 100 trajectories of successful runs

Figure 5.12: The LSD-SLAM-RS method trajectories from which the mean is calculated in the reiteration case.

Figure 5.13: A comparison of the averaged mean trajectories in reiteration case.
This chapter discusses the results.

In Section 6.1, the results of Section 5.1 are discussed.

In Section 6.2, the results of Section 5.2 are discussed.
6.1 100 trajectories

During the 100 trajectories experiment, the LSD-SLAM-RS method clearly improves the results of the rolling shutter sequence. As can be seen in Figure 5.1, both the mean error and standard deviation of the error have been reduced, which indicates that the method both lowers the error and also makes the result more predictable. Similarly in Figure 5.2, the resulting mean trajectory has lower error if compared to the ground truth.

6.1.1 Visual features of trajectories

Looking at the trajectory figures in this experiment, a similar pattern can be seen. In Figure 5.3, the ground truth trajectories can be seen. The trajectories align well, with a few exceptions. It is noticeable that a number of trajectories have outlier points in the end of the trajectory indicating failures in the end of the sequence.

In Figure 5.4, one can recognize the shape of the trajectory in the ground truth case but the trajectories seem to be very noisy and do not align as well as the ground truth. The obvious late failures that could be seen in the ground truth case are not as obvious here which could either indicate early failures that result in trajectories which do not span the entire path, or that the noise level makes them less visible.

In Figure 5.5, a similar shape as in the previous cases can be found. The noise level seems to be reduced compared to the rolling shutter case but not as well as the ground truth case. This further indicates increased stability and aligns with the lowered error, mean error and standard deviation in Figure 5.1 and Figure 5.2. In this Figure some outliers similar to the ground truth case can be seen indicating late failures.

Looking at the mean trajectories in Figure 5.6, one can see that when the rolling shutter case is averaged it is not similar to the shape in the previous figures. This can be the result of having too many failures or that the data is too noisy.

6.2 100 trajectories of successful runs

During the 100 trajectories of successful runs experiment, the outcome is not as obvious as in the 100 trajectories experiment. In Figure 5.8, the mean error and standard deviation of the error are very similar for the two methods. In Table 5.4, one can see that the difference is in the magnitude of $10^{-3}$ which in this case most likely is negligible. In the same table, the standard deviation has been lowered slightly for the LSD-SLAM-RS method. The conclusion here should likely be that the trajectory quality is similar for the two methods when only successful runs are taken into account. In Figure 5.9, the error is noticeably larger for the LSD-SLAM-RS method. The reason for this is not obvious since the quality of the single trajectory comparison seem to be very similar. An explanation would be
that since the LSD-SLAM-RS method gives more stable results, an actual difference between the ground truth sequence and the rolling shutter sequence might be more prominent in this method.

6.2.1 Tracking stability

In Figure 5.7, the comparison between the two method’s stabilities is shown. Here, one can see a significant improvement of LSD-SLAM-RS over LSD-SLAM without rolling shutter correction. The average number of runs $\mu_{\text{runs}}$ needed for a successful run has gone down from 2.59 to 1.54 which is an improvement of 70% compared to the ground truth. Similarly looking at the standard deviation of runs $\sigma_{\text{runs}}$ it has also been lowered from around 2.0 to 0.9.

6.2.2 Visual features of trajectories

Looking at the trajectory figures in this experiment, there is a similar pattern as in 100 trajectories.

Figure 5.10 shows a similar trajectory shape as the one in Figure 5.3 but without failed tracking in the end of the sequence. This is reasonable since they are not saved in this experiment.

In Figure 5.11 a slightly straighter shape than in Figure 5.4 takes form. This indicates that at least some of the noise present in the figure from the 100 trajectories experiment are due to mid-sequence failures. Still, there are more noise left in the figure compared to the GT case.

In Figure 5.12, a similar shape as the previous cases can be found. As in the 100 trajectory experiment the noise level seems to be lower than in the LSD-SLAM case, but not as low as in the GT case. One quite noticeable outlier is present in the end of the sequence.

In Figure 5.13 one can see that both in the LSD-SLAM and the LSD-SLAM-RS case a trajectory which resembles the GT has been calculated by averaging the trajectories. Both are noisy, especially in the beginning of the sequence.
To conclude the thesis, it seems one can improve the stability of semi-dense direct SLAM methods such as LSD-SLAM by taking into account the time difference from samples in rolling shutter data. The biggest problem in such a SLAM method used with rolling shutter data seems to be the alignment step when rolling shutter distortions are present, since trying to correct the distortions in this step improved the stability significantly.

Although the stability of the algorithm is improved it does not seem to affect the resulting trajectory that much. This is indicated by the results during the 100 trajectory of successful runs experiment where the rolling shutter aware model performed very similarly (for single trajectory comparison) or worse than the original method (for mean trajectory comparison).

Since all current consumer mobile cameras use rolling shutter cameras, improving stability and performance of 3D reconstruction on such sensors is a must if they are to be used as 3D scanners in a consumer setting. Since most mobile devices also have IMUs that can measure magnetic fields, acceleration and rotations, these could be used to do movement estimations that could help with both the pose estimation and the rolling shutter correction.
Bibliography


Appendix
A.1 Cross product operator

The cross product operator is defined as:

$$[a]_\times := \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix}.$$ (A.1)

A.2 Baye’s rule

Baye’s rule gives the conditioned probability that event $A$ happens given that event $B$ is observed. This can be calculated as:

$$P(A|B) = \frac{P(A)P(B|A)}{P(B)}$$ (A.2)

A.3 Intrinsic camera parameters

A camera’s intrinsic camera parameters are described as:

$$K = \begin{bmatrix} \alpha_x & \gamma & x_0 \\ 0 & \alpha_y & y_0 \\ 0 & 0 & 1 \end{bmatrix}$$ (A.3)

The parameters can be estimated by camera calibration.
A.4 Huber norm

The Huber norm is defined as:

\[ \|r\|_\delta := \begin{cases} \frac{r^2}{2\delta} & \text{if } |r| \leq \delta \\ |r| - \frac{\delta}{2} & \text{else.} \end{cases} \]  

(A.4)

A.5 Rigid and similarity transform in 3D

3D body transformations are in the report used to align and compare frames to each other. The following transformations are used:

3D rotation

A 3D rotation transformation \( R \) is contained in the Lie-group \( SO(3) \) which contains all rotations about the origin in \( R^3 \). These can be represented by real orthogonal matrices with determinant 1. A 3D rotation can be represented as a combination of 2D rotations around 1D subspaces of \( R^3 \) called the axis of rotation. Rotation of angle \( \phi \) around the z-axis in a Cartesian \( R^3 \) coordinate system would be represented:

\[ R_z(\phi) = \begin{pmatrix} \cos(\phi) & -\sin(\phi) & 0 \\ \sin(\phi) & \cos(\phi) & 0 \\ 0 & 0 & 1 \end{pmatrix} \]  

(A.5)

A full 3D rotation is then a combination of these:

\[ R = R_z(\phi)R_y(\alpha)R_x(\gamma) \]  

(A.6)

As a note the elements of \( SO(3) \) has three degrees of freedom, one for each rotation axis.

3D rigid body transform

A rigid body transformation \( G \) is contained in the Lie-group \( SE(3) \). It can contain a series of rotations and translations and can be represented as:

\[ y = \sum_{i=0}^{10} x_i G = \begin{pmatrix} R & t \\ 0 & 1 \end{pmatrix} \]  

where \( R \in SO(3) \) and \( t \in R^3 \).  

(A.7)

As a note the elements of the \( SE(3) \) group has six degrees of freedom. Three from the rotation and three from translation.
### 3D similarity transform

A similarity transformation $S$ is contained in $Sim(3)$. It can contain a series of rotations and translations and scaling.

$$S = \begin{pmatrix} sR & t \\ 0 & 1 \end{pmatrix}$$

where $R \in SO(3)$, $t \in \mathbb{R}^3$ and $s \in \mathbb{R}^+$. (A.8)

As a note the elements of the $Sim(3)$ group has seven degrees of freedom. Three from rotation, three from translation and one from the scale.

### A.6 Warping functions

In both LSD-SLAM and DTAM a warping function is used to project the entire image from one image plane to the other. In LSD-SLAM, the notation $\omega(p, D_i(p), \xi_{ji})$ is used. The warping function takes an image point in an image ($p$), projects it into 3D space by using the inverse depth map estimate ($D_i(p)$), applies the transformation ($\xi_{ji}$) and finally projects it back into the other image plane. Using the LSD-SLAM notation this can be written:

$$\omega(p, d, \xi) := \begin{pmatrix} x'/z' \\ y'/z' \\ 1/z' \end{pmatrix} \quad \text{with} \quad \begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} := \exp_{SE(3)}(\xi) \begin{pmatrix} p_x/d \\ p_y/d \\ 1/d \\ 1 \end{pmatrix}.$$  (A.9)