Real-time Panorama Stitching using a Single PTZ-Camera without using Image Feature Matching

Rikard Lindahl
Hampus Linse

Master's thesis
2015:E17
Real-time Panorama Stitching using a Single PTZ-Camera without using Image Feature Matching

Rikard Lindahl, F09
Hampus Linse, F07
Mathematical Institution
Supervisor at Axis Communications: Björn Ardö
Supervisor at LTH: Karl Åström

June 8, 2015
Abstract

In surveillance applications one thing to consider is how much of a scene one can cover with a camera. One way to augment this is to take images with overlap and blend them, creating a new image with bigger field of view and thereby increase the scene coverage.

In this thesis work we have been looking at how one can create panorama images with a pan-tilt-camera and how fast it can be done. We chose a circular panorama representation for this. Our approach was that gathering enough metadata from the camera one can rectify the gathered images and blend them without matching feature-points or other computationally heavy operations. We show that this can be done.

The images gathered was corrected for lens distortions and rolling shutter effects arising from rotating the camera. Attempts where made to find an optimal path for the camera to follow while capturing images. An algorithm to do intensity corrections of the images was also implemented.

We find that one can rotate the camera at high speeds and still produce a good quality panorama image. The limiting factors are the precision of the metadata gathered, like motion data from the on-board gyro, and the lighting conditions, since a short shutter time is required to minimize motion blur.

The quality varies depending on the time taken to capture the images needed to create the spherical projection. The fastest run was done in 1.6 seconds with some distortions. A run in around 4 seconds generally produce a good quality panorama image.
## Contents

1 Introduction .............................................. 3  
1.1 Background ........................................... 3  
1.2 Aim of the thesis ...................................... 4  
1.3 Structure of the report ................................. 4  
1.4 Related work ........................................... 4  

2 Methods .................................................. 6  
2.1 Coordinate systems ................................. 6  
  2.1.1 World coordinates .............................. 6  
  2.1.2 Image coordinates .............................. 6  
  2.1.3 Spherical coordinates .......................... 7  
  2.1.4 Moving between coordinate systems ......... 8  
  2.1.5 Our approach .................................. 9  
2.2 Image distortions ................................. 10  
  2.2.1 Lens correction ................................ 11  
  2.2.2 Rolling shutter compensation ............. 11  
  2.2.3 Combining rolling shutter and lens correction ... 12  
  2.2.4 Intensity differences .......................... 13  
2.3 Image blending ...................................... 14  
  2.3.1 Laplacian pyramid blending ............... 14  
2.4 Camera moving pattern .......................... 15  
  2.4.1 Own made up patterns ....................... 15  
  2.4.2 Theoretically calculated pattern ............. 17  
2.5 Timestamps .................................... 20  
  2.5.1 Correcting the timestamps .................. 20  

3 Results .................................................. 24  
3.1 Image corrections .................................. 24  
  3.1.1 Lens correction .............................. 24  
  3.1.2 Intensity correction .......................... 24  
  3.1.3 Blending ....................................... 26  
  3.1.4 Combining rolling shutter and lens correction 26  
3.2 Camera path ..................................... 30  
  3.2.1 Calculated path ................................ 30  
  3.2.2 Path Analysis ................................ 33  
3.3 Time corrections ................................... 35
Chapter 1

Introduction

In many surveillance contexts it is of interest to monitor a larger area than that of the camera's wide angle vision. In this thesis one idea is presented to deal with the problem. In the project a PTZ camera (an Axis camera which can rotate 360 respectively 180 degrees around its two axles) was used. The purpose was to stitch images from a live-stream from the camera while rotating the same to thereby obtain a greater representation of the scene. The stitched image would finally be represented in a circular projected image (CPI). There are other possible projections, for example a regular panorama or stereographic projection, but we chose to concentrate our work on the circular panorama representation.

Both the position and a timestamp could be extracted from the camera. By using this the images could be stitched together without having to search for image point correspondences, an operation which speeds up the process a great deal. To get a greater accuracy the gyroscope data from the camera could be used as well. Some problems which occurred included: How can image quality be retained while increasing the speed of the rotation? What effects the accuracy of point correspondences and how can it be dealt with? In which patterns should the camera be rotated, taking into account the wear on the motor as well as the speed of sweeping the whole area of interest? Finally, how could different operations and algorithms be optimized to get a desired speed of the process?

1.1 Background

Axis Communications AB is one of the world leading surveillance company's [7]. Because of the large market and varieties of needed applications, the clients sometimes request a certain feature or the company needs to develop new features to present to customers, to continue being on top of the surveillance market. The application of a larger representation of the scene might be useful in many scenarios, for example if you want to know what's going on in more areas without having to orient the camera yourself, or if you want to use the overview as a map to easier orient the camera to positions of interest.

In the work of this thesis we used the Axis PTZ-camera P5635-E [8]. From the camera it is possible to retrieve information about both motor position and timestamps corresponding to positions and images. It also has an on board
1.2 Aim of the thesis

In the thesis we aim to get a large overview circular panorama image, as seamlessly stitched together as possible without loss of too much detailed information. We also aim to show that an update of the panorama image can be done in real-time to get a live-view of the scene.

1.3 Structure of the report

In chapter 2 we first explain different coordinate systems and how to move between them. Then we present our approach to build the circular panorama image from these coordinate system. We continue with methods to compensate for different image distortions and techniques for blending the images. We then discuss potential moving patterns for the camera. Finally, we end chapter 2 with a method for correction of timestamps corresponding to the pan and tilt positions.

In chapter 3 we present results from the methods used and comment on some of the results as well. We conclude the report in chapter 4 with discussions about methods used, conclusions from results, possible applications and proposals for future work.

Finally there is a small appendix with some images captured during a field trip.

1.4 Related work

In most panorama applications some type of information from the images is needed to select how to align them. Szeliski discuss such methods in [17], both direct (pixel-based) and feature-based. The difference from our approach is that we don’t use any information from the images in the alignment (only angles, timestamps and gyroscope data received from the camera). This way we save a lot of computation time.

Lens distortions, such as barrel distortion, is a known problem in imaging. They appear since it’s not possible to create perfect lenses. The distortions can be approximated by mathematical expressions as described in [13]. Then there are different ways to calculate the correction coefficients in those equations, for example there are open source libraries (e.g. OpenCV) which can calculate the coefficients from features, or the lens manufacturer might also provide information for the lenses. Another image distortion that might appear is the one from the rolling shutter cameras. These cameras saves information row by row in an image and this can lead to distortions if the camera is moving while the image is taken. In [1] they discuss a method of using information from gyroscopes to compensate for the movement. In our implementation we also use gyroscope data from the camera to do the correction. This is of importance since we move the camera in high speeds and then the rolling shutter effect becomes quite prominent.
When putting the images together, one problem that occurred was the different intensities between them, due to different lightning conditions and exposures amongst other things. In [3] they discuss methods for compensating for these intensity differences. They first propose a diagonal model which is independent of pixel correspondences and works fast. They then expand this to general linear models with affine transform which has more accuracy but with the disadvantage of heavier computation and the need of pixel correspondences. Finally they suggest a histogram based method, independent of both image scales and pixel correspondences, but also with the disadvantage of computation time.

[18] also discuss the simple diagonal model for the compensation. In the expansion of this model they propose working in a linearized RGB color space (by undoing intensity correction). Furthermore they suggest a global compensation, using all images, to adjust the intensity in each image as little as possible to avoid image saturation. In their method a first image must be chosen to correct all other images towards.

Blending images into the final image is not a straightforward task and a few different methods was considered. To keep a smooth transition between images a multi-band blending scheme based on [5] was used. This is to keep the high-frequency spatial information while blending the low frequency information seamlessly. The scheme described in [5] only uses one high frequency and one low frequency band. There is a possibility to use more bands as described by Burt et al. in [2]. This might help give a better blending, but is more expensive when it comes to computational time.
Chapter 2

Methods

Given an arbitrary image from a camera not much is known. But if also the horizontal field of view, aspect ratio and the orientation of the camera is given one can calculate the projection of the image on the unit sphere. This is the basis of our stitching method.

2.1 Coordinate systems

A large portion of the work concerns transformations of coordinates between different coordinate systems. To keep track of these they are explained below.

2.1.1 World coordinates

The standard cartesian coordinate system will be called world coordinates. Typically the camera is placed at origo and the image will be placed at unit distance from the camera. This can be seen as a model of our setup where the camera center is at origo and the image is projected unto the unit sphere. In world coordinates rotating the coordinates around origo is very easy and will be used to mimic the camera rotating while taking images.

2.1.2 Image coordinates

The standard image coordinate system will be called the image coordinate system. It is similar to the cartesian coordinate system but with the y-direction reversed. In image coordinates the coordinates of the top left pixel in any image is (1,1) and the bottom right is (width, height) in pixels.

There is also the term normalized image coordinates. The normalized image coordinates is here defined as having origo in the center of the image and the distance from origo to the image edges are 1. The x- and y-axis are oriented as the cartesian coordinate system. The physical interpretation is that the center of the image is where the optical axis intersects the image. To transform image coordinates to normalized image coordinates we use the equations

\begin{align}
    x_n &= \frac{2(x-1)}{w-1} - 1, \\
    y_n &= \frac{2(1-y)}{h-1} + 1,
\end{align}

\text{(2.1)}
where $x$ and $y$ are the image coordinates in pixels, $I_w$ and $I_h$ are width and height of the image in pixels and $x_n$ and $y_n$ are the normalized image coordinates. The result of the transformation is shown in figure 2.1. During our work we don’t use the normalized coordinates per say but the scaled normalized coordinates. They are scaled to represent the image in our world model and this will be explained below.

### 2.1.3 Spherical coordinates

To transform image coordinates into world coordinates we first transform them into spherical coordinates. This is done since every pixel in the image corresponds to a unique angle in relation to the orientation of the camera lens. If the $z$-axis is chosen as the optical axis transforming the pixel coordinates into spherical coordinates is straightforward. The spherical coordinates have the following components: radial distance $r$, polar angle $\theta$ and the azimuth angle $\varphi$, see figure 2.2.

This is done to correctly project the image coordinates into world coordinates while keeping the computations fairly simple. Imagine that the camera is placed at origo and oriented so it sees along the $z$-axis. The image is placed at the
world coordinate \((0,0,1)\), parallel to the \(xy\)-plane, and scaled to the correct size. Then the \(\theta\) value for a specific image coordinate depends on the distance from the image center and \(\varphi\) depends on the polar angle in the image. Since the coordinate should be on the unit sphere \(r\) can directly be set to 1.

The scaling is determined by calculating the width and height of the image given the field of view acquired from the camera and deciding the distance from the camera is 1. We used the equations

\[
\begin{align*}
    w &= 2 \cdot \tan \left( \frac{h \cdot \text{fov}}{2} \right), \\
    h &= 2 \cdot \tan \left( \frac{v \cdot \text{fov}}{2} \right),
\end{align*}
\]

(2.2)

to determine the scaling, where \(w\) and \(h\) is the image width and height in world space and \(h \cdot \text{fov}\) and \(v \cdot \text{fov}\) is the horizontal and vertical field of view given by the camera. Since the normalized coordinates range from \([-1,1]\) they are scaled by multiplying with half the width and height calculated above.

### 2.1.4 Moving between coordinate systems

To transform coordinates between the image and the world space coordinates we follow this pattern:

image \(\leftrightarrow\) spherical \(\leftrightarrow\) world

Combining equation 2.1 and 2.2 we can construct the matrix equation

\[
\begin{bmatrix}
    x_n \\
    y_n \\
    1
\end{bmatrix} =
\begin{bmatrix}
    w/2 & 0 & 0 \\
    0 & h/2 & 0 \\
    0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
    \frac{2}{I_w - 1} & 0 & \left(\frac{-2}{I_w - 1} - 1\right) \frac{w}{2} \\
    0 & \frac{-2}{I_w - 1} \left(\frac{1}{I_w - 1} + 1\right) & 0 \\
    0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
    x_I \\
    y_I \\
    1
\end{bmatrix},
\]

\[
\Leftrightarrow
\]

\[
\begin{bmatrix}
    x_n \\
    y_n \\
    1
\end{bmatrix} =
\begin{bmatrix}
    \frac{w}{I_w - 1} & 0 & \left(\frac{-2}{I_w - 1} - \frac{1}{2}\right) \frac{w}{2} \\
    0 & \frac{-h}{I_h - 1} & \left(\frac{1}{I_h - 1} + \frac{1}{2}\right) h \\
    0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
    x_I \\
    y_I \\
    1
\end{bmatrix},
\]

(2.3)

where \([x_I y_I 1]^T\) is the image coordinate and \([x_n y_n 1]^T\) is the scaled normalized coordinates for a specific pixel in the image. \(I_w\) and \(I_h\) is the image width and height in pixels. Returning from normalized scaled coordinates to image coordinates is then a simple operation of multiplying with the inverse of the transform matrix.

Moving from scaled normalized coordinates into spherical coordinates we use the following equations

\[
\begin{align*}
    \theta &= \arctan\left(\sqrt{x_n^2 + y_n^2}\right), \\
    \varphi &= \text{atan2}(y_n, x_n), \\
    r &= 1.
\end{align*}
\]

(2.4)

This gives the position of the pixel in spherical coordinates projected on the unit sphere.

Moving back from spherical coordinates to scaled normalized image coordinates is then done with following equation

\[
\]
\[ x_n = \tan \theta \cos \varphi, \]
\[ y_n = \tan \theta \sin \varphi. \]

Moving back and forth between spherical coordinates and world coordinates is straightforward since all scaling has been done before these steps. Spherical to world coordinates are transformed by

\[ x_w = r \sin \theta \cos \varphi, \]
\[ y_w = r \sin \theta \sin \varphi, \]
\[ z_w = r \cos \theta, \]

and back from world coordinates to spherical coordinates

\[ \theta = \arccos\left(\frac{z}{\sqrt{x^2 + y^2 + z^2}}\right), \]
\[ \varphi = \arctan\left(\frac{y}{x}\right), \]
\[ r = \sqrt{x^2 + y^2 + z^2}. \]

We will also need to move coordinates from the destination image (panorama image) to world coordinates. This depends on what transform one wants to use to create the panorama image. We use the so-called circular projection. In circular projections the image is square and the distance to the center of the image corresponds to the spherical coordinate \( \theta \) and the polar angle corresponds to the azimuthal angle \( \varphi \), defined as in figure 2.2. The radius in spherical coordinates will be defined as 1. So first the panorama image coordinates are normalized and then converted to spherical coordinates using the equation

\[ \theta = \arccos\left(\frac{z}{\sqrt{x_n^2 + y_n^2 + z^2}}\right), \]
\[ \varphi = \arctan\left(\frac{y_n}{x_n}\right), \]
\[ r = 1, \]

where \( x_n \) and \( y_n \) in this case are normalized image coordinates for the CPI. The spherical coordinates where then finally converted into world coordinates using equation 2.6.

Notice the use of the function \( \arctan2 \), this is because the standard \( \arctan \) function is discontinuous at \( \pm \pi \). The \( \arctan2 \) function is defined as

\[
\arctan2(y, x) = \begin{cases} 
\arctan\frac{y}{x} & x > 0 \\
\arctan\frac{y}{x} + \pi & y \geq 0, x < 0 \\
\arctan\frac{y}{x} - \pi & y < 0, x < 0 \\
\pi & y > 0, x = 0 \\
-\pi & y < 0, x = 0 \\
\text{undefined} & y = 0, x = 0.
\end{cases}
\]

### 2.1.5 Our approach

To avoid image distortion when mapping pixels from one image to another we use backward mapping. Backward mapping is when one calculates where the destination pixels are placed in the source image and forward mapping is the opposite.

If the source image has low resolution and every source image pixel is transformed into the destination image (forward mapping) we probably will miss...
pixels in the destination image. This will give black gaps in the destination image and that is not satisfactory. Therefore it is common practice to use backward mapping.

Our approach for handling the pixel transformations is then as follows:

1. Transform destination image (the CPI) pixel coordinates into world coordinates. This geometrically places the pixels from the destination image on the half sphere.

2. Rotate the coordinates with the inverse of the camera rotation. This can be seen as deciding that the camera view direction should be orientated along the z-axis. We are continuously checking the cameras motor positions to determine the cameras rotation when capturing the images.

3. Convert into source image coordinates. Now the destination image pixels have coordinates in the source image coordinates.

4. Grab the color from the nearest source image pixel and put into the destination image in the original pixel. If the destination pixel coordinate is out of the source image bounds then it is discarded.

Since we will have aberrations we don’t put the pixel information directly into the destination image, we use a blending algorithm described in section 2.3.

2.2 Image distortions

There are a lot of factors that distorts the image and therefore creates pixel mismatches. There are mainly three factors that we try to compensate:

- Lens distortion
- Rolling shutter effects
- Intensity differences between images

There are effects that we haven’t tried or been able to compensate, and these are vignetting and parallax effects.

Vignetting stems from light not hitting the sensor evenly and thus making parts of the image darker. There are several causes for this like mechanical obstruction, the optics and the angle of the light. We decided not to focus on this part because of the multiple causes and complexity of these effects.

Parallax effects appear since the camera rotates around its center of mass, not around the lens center. This means that the center is not perfectly at origo as described in section 2.1. Parts of the scene might be visible from one angle that is obstructed in another. This is most apparent for objects close to the camera. We have no good theory of how to correct for this. Assuming that the surveillance camera is placed sufficiently far from objects in the scene, depending on how far from the center of rotation the lens is, the errors are small and should be taken care of by the blending algorithm. An example of parallax effect can be seen in figure A.1, where the pole holding the camera is very close to the camera (about 10-15 cm).
2.2.1 Lens correction

Lenses are rarely perfect in the real world and therefore lens distortion must be taken into account if the images used are to be as accurate as possible. The model used in this report is the Brown-Conrady model, based on Conrady’s work[9]. In the model used we ignore the tangential distortions and only try to compensate for the radial distortions. This gives the equations

\[ x_d = x_u (1 + K_1 r^2 + K_2 r^4 + K_3 r^6 + ...), \]
\[ y_d = y_u (1 + K_1 r^2 + K_2 r^4 + K_3 r^6 + ...), \] (2.10)

or equivalently

\[ r_d = r_u (1 + K_1 r^2 + K_2 r^4 + K_3 r^6 + ...), \] (2.11)

where \( x_d, y_d \) are the distorted image points/pixels, \( x_u, y_u \) are the undistorted image points/pixels, both in scaled normalized coordinates, \( r_d \) and \( r_u \) is the same but along the radius and \( K_n \) are the radial distortion coefficients that transforms the undistorted image points into the distorted ones, given their distance \( r \) from the image center. The \( K_n \) coefficients need to be calculated in some way. We decided to calculate three coefficients \((K_1, K_2, K_3)\). The coefficients following these were so small for our lens that the error was considered negligible. We have data from the lens manufacturer and also a calculation using an open source program called OpenCV that calculates camera values. Which one of these to use is not obvious but we decided to use the measured values from OpenCV since these are calculated with consideration for the whole system, not only the lens. The lens manufacturer gave us data that described the amount of distortion given the radius of the lens. Using the least squares method we calculated the coefficients for the zoom and focus settings used throughout the thesis. New values would have to be calculated if these are changed.

The values calculated from the lens manufacturer data and the measured data is displayed in the table 2.1. The results supports our decision to use the OpenCV values but a more robust way of measuring this would be preferable.

<table>
<thead>
<tr>
<th>name</th>
<th>( K_1 )</th>
<th>( K_2 )</th>
<th>( K_3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lens Manufacturer</td>
<td>-0.03044461</td>
<td>-0.0138924</td>
<td>0.00990777</td>
</tr>
<tr>
<td>OpenCV</td>
<td>-0.0749898</td>
<td>0.0178027</td>
<td>0.0043686</td>
</tr>
</tbody>
</table>

Table 2.1: The two sets of lens-manufacturer coefficients

2.2.2 Rolling shutter compensation

Rolling shutter describes how the camera sensor captures the image. Intuitively we think that an image is taken simultaneously for all pixels, so called "global shutter", but the sensor in our camera uses the rolling shutter method. This means that only part of the sensor registers light and this part, usually a few rows of pixels, travels over the sensor. The gains of using this type of image capture is that it generally delivers images faster and thus this type of sensor is often used in video cameras. The disadvantage with using rolling shutter sensors is that distortions might occur if objects move during capture.
Since the camera will be moving a lot when gathering images for the panorama we will have quite prevalent rolling shutter effects in the images. To correct this we use position data from gyroscopes on the camera that registers the camera motion. The gyro data is converted through a camera API into pixel differences. We assume that the first row is correct, that is, it corresponds to the motor position data determined for the image. Then, for every row with registered gyro data, the gyro difference is added. The differences are represented as arrows in figure 2.3. The data from the gyro API, when writing this, give only the yaw and pitch of the camera, not the roll. This is one component that need to be added if we want to completely compensate for the rolling shutter effects.

![Figure 2.3](image)

Figure 2.3: How the image is changed due to rolling shutter capture. The left box represents the image captured and the right box is the image had all pixels been placed spatially correct. The arrows represent the gathered pixel differences for the rows where we have gyro data.

### 2.2.3 Combining rolling shutter and lens correction

We can't correct for rolling shutter and do lens correction separately as both depends on and modifies the pixel positions in the source image. Our approach tries to correct both at the same time by assuming that the distortions are separate of each other. The lens distortion does not depend on the camera motion and the rolling shutter effect does not depend on the lens. Therefore the individual contribution to the pixel differences, caused by the distortions, are stored separately. When both calculations are done they are applied simultaneously.

In our case the barrel distortion is calculated when the panorama image coordinates are transformed from world coordinates to image coordinates. This is between step 2 and 3 explained in section 2.1.5. A part of this step is going from spherical coordinates to image coordinates, and here we have the spherical coordinate \( r \) that is easily modified to compensate for the lens distortions. This change is stored as a separate coordinate and is also transformed into image coordinates. The difference between these coordinates are stored.

The rolling shutter effects depends on the position in the image and in our case it depends on the row of the image. When the coordinates are in image coordinates we use linear interpolation between the rows where we know the amount of distortion to get the individual pixels rolling shutter distortion. This is also stored for every pixel.

Finally these differences in pixel positions are added to the pixel coordinates and we have a simultaneous correction of both barrel correction and rolling shutter distortion.
2.2.4 Intensity differences

Because of different exposures and lightning conditions in a scene, the images received from different angles might be so diverse in intensity that it could be difficult to get an overview with smooth transitions (even after blending). Therefore we searched for a way to try to get a more similar intensity over the whole circular panorama. In the work of intensity corrections all calculations were made in YCbCr space, since this space represents one luminance channel (and the blue respectively red-difference chroma component). The luminance component ranged between 0 to 255 in value.

Intensity correction method 1 (not used)

As a first implementation, the linear least square method \[4\] was used to correct images. To increase speed of calculations a grid with equally spaced points was used. Points in overlapping pixels between a new image and the ones already put in the circular image were used to calculate the least square linear solution. The following four alternatives were tested to solve the system:

A new image was corrected towards the so far built up CPI by minimizing

\[
(i_{\text{circular}} - (a \cdot i_{\text{image}} + b))^2,
\]

where \(i_{\text{circular}}\) is a column vector containing the circular image intensity values at the grid points, \(i_{\text{image}}\) is the corresponding intensity vector for the new image, \(a\) and \(b\) are the constants to be found by the least squares method. The calculations were made both with and without the offset \(b\).

In the third and fourth alternative, both the new image and the circular image intensities were corrected towards the weighted mean value of the overlapping pixels in the grid. They were weighted with the number of images that had been sent into the circular image (with a maximum weight of 30 images), in order to give a larger weight to intensities in the circular image (to treat the problem if a new image would have abnormal intensities). The same linear formula was used with and without offset \(b\).

Unfortunately none of the results of the final images using these techniques were satisfactory. The algorithms were dependent of good accuracy in pixel correspondence and this could have led to outliers effecting the result to much. Also vignetting and noise in images could have effect the result. To handle some of these effects, the absolute difference could perhaps have been used instead (since it is more robust against outliers). A low pass filter might also have dealt with noise and some pixel mismatches. But instead we tried a different, more efficient model, which gave more satisfactory result. It is described below.

Intensity correction method 2

A grid was used as before to improve calculation speed. Then a modification of the diagonal model, described in both \[3\] and \[18\], was used. As said above, all work was done in YCbCr space, and therefore only the Y-channel (luminance channel) had to be corrected. The mean value of the Y-channel in the overlap in both images was calculated. Then a weighted average of these averages was calculated (as above) and the images were corrected by multiplying the Y-channel with the weighted average divided by their respectively averages. The
weighted average technique made sure that images with abnormal intensities, for example due to some bad lightning condition, would not effect the circular image too much, and if the lightning conditions in the whole scene would change, there would be a smooth adjustment to the new light. If pixels in an image had too high or too low intensity values in certain areas, they were not included in the calculations. This was done because saturated pixels might not provide correct information and bad lightning conditions among other things might have led too darker pixels than it should have been. As a lower bound pixel values less than 5 were not used and as an upper bound pixel values above 250 was not used. If a new image sent to the circular image was to dark in the entire overlap to retrieve usable information, it was simply not added to the circular image.

To avoid the problem of noise, low pass filtered images were passed through the intensity correction algorithm.

There might be other algorithms that could be used, resulting in a better image. But many of them are computationally more heavy and might require multiple images to calculate a correction for each image. Using, for example, the technique described in [18], would require images from the whole scene to receive a global correction factor used to correct all individual images. This would take more time and since the goal of our work is to be able to do a real-time update, fast calculations are of great importance.

2.3 Image blending

Assuming that there is neither intensity differences in overlapping images nor aberrations, from lenses or camera movement, the images would perfectly overlap and forming a pleasant panorama image would be trivial. However this is not the case in reality. Some blending will be required both for pixel mismatches and for intensity differences between the images. Our criterion for a blending algorithm is that it should be fast but try to keep as much detail as possible. If the intensity differences are big enough some intensity correction should be used.

2.3.1 Laplacian pyramid blending

We decided to use a multi-band blending scheme described by Brown and Lowe [5], who in turn made theirs on the work of Burt and Adelson [6]. The idea is to blend low frequency spatial information seamlessly while retaining only the high frequency spatial information of one image. The high frequency information is chosen in the same way as described in [5]. Every pixel in an image is assigned a weight given its position in the image. The weight linearly varies from 1 in the center and 0 at the edge of the image. When blending two images the high frequency spatial information is taken from the pixels with the higher weights. All this is done to get a smooth transition between images, which is more pleasant to look at, while retaining high frequency information, details in the image, that otherwise would have been blurred due to image mismatch, motion or distortions.

In our implementation we keep one circular projection image with the current panorama and create one circular projection image with only the input image...
that we want to add. We also store the weights from the input image in a weight map the same size as the circular panorama image. At this stage we therefore have two circular projected images and two weight maps, one pair for the current CPI and the other pair for the new input image. Using the theory of Laplacian pyramids one low frequency image is created by blurring the image with a Gaussian kernel and a high frequency image is created by subtracting the original image with the blurred image. The Gaussian images are then blended linearly using the weights at every pixel and the high frequency images are put together as described above. To get the final image the low frequency and the high frequency image are added on top of each other.

We decided to only do one high and one low frequency image. In theory we could do more pairs and blend these but decided to keep the approach simple to save time.

2.4 Camera moving pattern

To evaluate the pattern that the camera was moving in, we wrote a function returning registered motor position with timestamps registered in a camera data cache. Since discrete differentiation, for example with central differences, is susceptible to signal noise we instead used differentiators described by P. Holobarodilo[12]. With these we calculated both the speed and the acceleration. We analyzed both own made up patterns and a theoretically calculated pattern.

At first we tried to design the patterns intuitively, considering different operations to optimize at. There were several things to take into account, such as minimizing the wear on the motors by minimizing the acceleration, cover the whole view as fast as possible and update certain areas more often which might be of more interest, amongst other things. The patterns are described in 2.4.1. After designing these patterns we had the idea of trying to design algorithms to find a certain pattern theoretically, described in 2.4.2. This was harder than first anticipated and in the end was never used in our final implementations, so the reader can skip this part if not interested.

These patterns was then converted into control points and a simple C-program was made to make the camera follow this pattern. All motions where logged and used in the analysis algorithm.

2.4.1 Own made up patterns

We designed three different patterns which are described below.

1. The idea of our first pattern, "the roundabout" was to minimize the wear on the pan motor by running it in a constant speed. Starting at a tilt position reaching the outer limits in the circular panorama, the tilt would then move to a new position once every lap in pan. Since we needed an overlap between the images in the panorama in order to blend the images well, we chose tilt positions to get some overlap in an attempt to make the blending look good for the eye. After the second lap in pan direction, most of the view was covered with our camera, and the remaining area could be covered by simply tilting the motor to the corresponding position at the other side of the center tilt. Then the pattern could continue in the same manner throughout a run.
2. In our second pattern, "the flower", we still used a constant speed in pan direction. By moving the tilt motors from the lowest position on one side to the opposite position on the other side, forth and back, the whole view could be covered after 5 crossings over the center tilt position. This pattern would update the center more often, because of these crossings.

3. In the last pattern, "the frog", the pan speed was still constant. The tilt position moved up and down from the lowest position to a position making sure the top images covered the center position. The tilt had to move a total of twelve times up and down to cover the whole scene.

In figure 2.4, 2.5 and 2.6 the schematic path of the three different made up patterns are shown.

Figure 2.4: Pattern number one, "the roundabout", from own made up patterns.
2.4.2 Theoretically calculated pattern

The idea for calculating the moving pattern for the camera was the following:

We created a grid, with equally spaced points, covering the circular panorama image. The meaning of the spaced grid was only to speed up calculations. In figure 2.7 the grid is shown. For each of the points in the grid we determined which grid points an image would cover. In figure 2.8a and 2.8b two of these image views are shown.
The problem was now to find the least amount of images needed to cover the entire grid. We found a starting solution using the unweighted Greedy algorithm described in [16]. To solve it we built up a matrix where each row represented a viewing angle in the grid. The columns also represented the viewing angles in the grid. Looking in a row, the grid points covered by an image from this view were marked by a 1 in the columns, the others columns were marked as 0. Choosing rows iteratively, with each grid point having the same weight, the row
with the minimum overlapping grid points was chosen each time. When all grid points had been covered, the starting solution had been found. A schematic drawing of the problem is presented in figure 2.9. The starting solution could end up in a local minimum. To improve the solution we therefore checked if there was a possibility to remove an image and still have total coverage. To improve the solution even more, we removed one image at a time (removing the one with the most overlap). While removing images, some grid points might no longer be covered. Non-covered grid points were marked with a much larger weight (since full coverage was of importance). When an image was removed we checked if an image could be replaced by another to get a greater coverage (until the greatest coverage was found). When using the resulting viewing directions, a coverage of all grid points was ensured (or at least most of the grid points, if there wasn’t enough images to have full coverage). We then used these grid points as points we had to move through in our pattern.

![Figure 2.9: Schematic drawing of how to solve the set problem with the Greedy algorithm. Each row and column represents the views. Ones mark views that are covered from a certain center point in a view (a row). If the algorithm first chooses the first row, it will then choose the third one, since there are no overlapping ones between the columns. Then it would choose row 2 to fill the missing one in the last column. Here we would get some overlap at columns 1, 2 and 3, but it would still be necessary to use the row to get full coverage.](image)

To find the optimal path there were, as said in the first part of this section, several things which one can optimize towards. We decided to try to find the shortest way through the points. It might also have been possible to optimize towards for example a minimum acceleration, but we only used the shortest distance (after learning that the theoretical way of finding a path was harder than first anticipated and would be to time consuming). The problem of finding the shortest distance between a number of points is a famous problem called the "traveling salesman problem" [10]. One way to solve it is with the simulated annealing algorithm described in [15]. The algorithm starts with a random selected route between the points as a starting solution. Then it randomly switches
paths between two points if the total distance is decreased. Even though the solution is worse, it can still switch to this solution with a certain probability. To decide whether to change to a worse solution we used a probability function which returned a uniformly distributed number between 0 and 1. This number was compared with an exponential function which depended on both the difference in the current and previous total distance, and a variable temperature. If the random number was smaller than the value of the exponential function (which tended towards 0 over time) it would shift to the worse solution. The temperature variable dropped with constant factor each time a better solution was found. Using this, the probability of changing to much worse solutions would decrease over time. The idea of changing to worse paths was to try to avoid getting stuck in local minimaums. We ran the algorithm iteratively and saved the best of the solutions to be even more sure on that we hadn’t gotten stuck in a local minimum.

To get a smoother path, and thereby decrease accelerations, we used the technique of cardinal splining described in [11]. We only had to spline in the 2D-plane, between the pan and tilt angles, since the radius was constant.

2.5 Timestamps

Timestamps were logged while filming with the camera, both for positions of the camera and for the images in the film. Correct timestamps, corresponding to pan and tilt positions, was of great importance. We used the two timestamps for the positions closest to the image timestamp to linearly interpolate the correct position for the image. This position was used as the center position for the image, before correcting for image distortions. In reality the timestamp is only valid for the first captured row of pixels in the image. Since the camera is moving the subsequent rows are valid for other pan and tilt values, which we try to correct in the rolling shutter section. Moving the camera in higher and higher speeds could result in insecure timestamps. To deal with the problem we designed an algorithm using gyroscope data from the camera to find better timestamps for the positions. Since the gyroscope data was updated more frequently and since it had better precision, our idea was that it could be used to get better accuracy in the timestamps of the pan and tilt positions as well.

2.5.1 Correcting the timestamps

The gyroscope positions were updated approximately once every 0.5 ms and the pan and tilt positions at every 10 ms. The gyroscope data had some drift and therefore we could only rely on a difference in position between two measurements. The comparison between the data was therefore made in difference in degrees per timestamp. Since the gyroscope data, received from the camera API, was in pixel differences it had to be transformed to degrees. To derive the transformation we look in figure 2.10 which illustrates an image plane viewed from the camera center.

We have the following relationships in figure 2.10

\[
\Delta = \frac{\delta}{h/2} \cdot (p/2),
\]

(2.13)
\[ \delta = \tan(\alpha), \quad (2.14) \]
\[ h/2 = \tan(\text{FOV}/2), \quad (2.15) \]

where \( \Delta \) is the corresponding shift to \( \delta \), expressed in image coordinates (pixels), and \( p \) is the height respectively width of the image in pixels. Insertion of equations 2.14 and 2.15 in equation 2.13 leads to the following equation
\[ \Delta = \frac{\tan(\alpha)}{\tan(\text{FOV}/2)} \cdot (p/2), \quad (2.16) \]

and solving for \( \alpha \) leads to
\[ \alpha = \arctan\left(\frac{2 \cdot \Delta}{p} \cdot \tan(\text{FOV}/2)\right). \quad (2.17) \]

Since the difference in pixels was very small it would have been possible to approximate the equation with
\[ \alpha = \frac{2 \cdot \Delta}{p} \cdot \tan(\text{FOV}/2). \quad (2.18) \]

Then one could have calculated a constant \( \frac{2}{p} \cdot \tan(\text{FOV}/2) \), which when multiplied by \( \Delta \) would yield an approximation of \( \alpha \). But in our implementation we used the first of these equations anyway since it would have had little effect in computational speed compared to other operations.

Figure 2.10: Illustration to help understand how to move from difference in pixels to difference in angles. FOV is the horizontal respectively vertical field of view, \( \delta \) is a shift in world coordinates and \( \alpha \) is the corresponding shift in angles, \( h \) is the height respectively width of the image in world coordinates.

In figure 2.11 the difference in gyroscope data in horizontal direction, converted to difference in degrees per second, is plotted against the difference in pan...
data per second. The corresponding plot for gyroscope data in vertical direction and tilt data is shown in figure 2.12. As seen in these figures, the gyroscope data fluctuates. This was caused by a difference in sampling time between each and other sample. Looking in figure 2.12 the tilt data follows the gyroscope data almost perfectly. This might be explained by the fact that tilt data is fetched faster than pan data from the camera. Because of the good data from tilt positions we decided not to change timestamps for this data. Before trying to correct timestamps for the pan data we first filtered the gyroscope data with a Gaussian filter of length 5, to get a smoother curve to correct towards.

![Figure 2.11: Shows the raw unfiltered data. Red values are the difference in gyroscope data in horizontal direction per second, blue values are the difference in pan data per second.](image)

Below we explain our algorithm for correcting timestamps for the outliers. If a sample had a bigger offset from the filtered gyroscope data than a certain threshold, the timestamp for the second measurement of pan/tilt position was shifted in time by 0.5 ms (approximately the time between two samples of gyroscope data). The threshold was calculated from the greatest absolute difference in the fluctuation between two consecutive gyroscope samples (divided by 4). Depending whether on a sample was below or above the curve, positive or negative, the timestamp should shift either forward or backwards. Because of the shift of the second timestamp it would also effect the position of the sample after the first one with an offset. But as seen in figure 2.11, most outliers come in pairs, so moving the timestamp would probably lead to better values for both outliers. The fact that they come in pairs also points towards that the errors are in the timestamps and not in the positions.
Figure 2.12: Shows the raw unfiltered data. Red values are the difference in gyroscope data in vertical direction per second, blue values are the difference in tilt data per second.
Chapter 3

Results

3.1 Image corrections

3.1.1 Lens correction

We got lens distortion values from the lens manufacturer. However at the start of the master thesis work we were not satisfied with these values and were looking for a way to verify them or measure the distortion ourselves. We found an open source framework called OpenCV [14] that could measure these with images from the camera containing a calibration board. The results of using these values are shown in figure 3.1. The OpenCV values were considered better than the manufacturer's values due to reduced ghosting in details of the resulting images. We believe the lens correction is correct due to reduced ghosting, however the differences are so small that a more objective measurement would be preferable.

3.1.2 Intensity correction

As can be seen in 3.2 the correction algorithm suppresses the intensity changes and gives a more homogenous image to view.

There are some things that need to be improved. We noticed that the algorithm generally performed better when the images had large overlaps with surrounding images. In figure 3.2 we use the "roundabout" pattern and generally we have very good overlap with images on the same circle as the current image. Often better than the images on the other circles, and therefore we get some intensity changes between the circles.
Figure 3.1: Differences on the same scene using different lens correction values. a) is no correction, b) is corrected using values from the lens manufacturer and c) is measured values using the OpenCV framework. d) show a detail comparison with the same layout as the three other images. Notice the reduced ghosting with the OpenCV values. The scene is the auditorium in the Lund University building.
We noticed however that when moving the camera at high speeds we needed to set a fixed shutter length small enough that could give clear images with minimum motion blur. If we then set a fixed gain value for the images as well the images had basically the same intensity profile and gave good results without the intensity correction. In fact our intensity correction might even worsen the resulting panorama image with errors due to vignetting or image mismatching. These errors are more noticeable when the images have similar intensity values.

Fixing the shutter time and gain limits the camera in the sense that if the scene contains large intensity differences the camera cannot compensate (i.e. change gain). This is a problem for indoor scenes, where we get burnt out windows, but generally not so much for outdoor scenes, like figures A.3, A.4 and A.5.

### 3.1.3 Blending

The blending algorithm worked generally very well and was relatively effective using Octave’s built in filter routines. An example of how it blends seamlessly can be seen in A.2. We noticed however that the blending algorithm decides what is details depending on the size of the kernel used for image blurring. For large images we concluded that more bands would be required to keep consistent quality independent of image size, at the cost of computational time.

### 3.1.4 Combining rolling shutter and lens correction

In figure 3.3 we see the results of combining the rolling shutter correction. What we can see is that when capturing images at high speeds the rolling shutter distortion is the major distortion. The lens correction is barely noticeable in comparison. As can be seen in the complete image in figure 3.3b the images still do not line up perfectly which gives the uneven lines in the roof, although they should be straight. This could be due to imprecise position data or uncertainty in the gyro. But due to the choppy effect being more prevalent when the camera is pointing upwards we suspect it might be to the camera
Figure 3.3: Results from the combined image corrections. The images were taken in 2.32 seconds, a total of 62 images were used.
rolling. So far we have only considered rolling shutter effect when the camera is tilting and panning, but given the tilt angle panning will result in a rolling effect most prevalent when the camera is pointing upwards. This can be seen in figure 3.4, which shows the same scene captured at different speeds. Table 3.1 show the results of the run. The speed is in motor steps per second and there is 230400 motor steps per lap.

The resulting mismatch at high speeds is from saturating the gyro and not compensating for camera roll. An image showing the same scene at high speed

![Image](image_url)

(a) speed: 100k  
(b) speed: 200k  
(c) speed: 300k  
(d) speed: 450k

Figure 3.4: Using all the image corrections at different path speeds. Speed is in motor steps per second. Enhanced roundabout pattern used.

where the gyro sensitivity has been reduced is shown in 3.5. As can be seen this is an improvement but still the camera roll is not compensated.
Table 3.1: The time taken to capture the scene and the number of images used to assemble CPI’s in figure 3.4. Speed is in motor steps per second. Notice that when running at speed 100k we get such a surplus of images that only every other image is used.

<table>
<thead>
<tr>
<th>Speed</th>
<th>Time taken (s)</th>
<th>No. images used</th>
</tr>
</thead>
<tbody>
<tr>
<td>100k</td>
<td>5.28</td>
<td>66 (every other)</td>
</tr>
<tr>
<td>200k</td>
<td>2.72</td>
<td>69</td>
</tr>
<tr>
<td>300k</td>
<td>2.12</td>
<td>54</td>
</tr>
<tr>
<td>450k</td>
<td>1.68</td>
<td>43</td>
</tr>
</tbody>
</table>

Figure 3.5: Sequence captured at speed 450k with lowered gyro sensitivity. Compared with figure 3.4 d we have a lot less distortions.
3.2 Camera path

3.2.1 Calculated path

The starting solution of the least amount of images needed to cover the whole grid in the circular panorama, found by the Greedy algorithm, is shown in figure 3.6a. This solution contained 19 images and two of these were not necessary to still cover the whole grid. Therefore these two images were removed and the new solution is shown in figure 3.6b. The result after looking for images with the greatest overlap, and trying to replace these with images with less overlap, is shown in figure 3.6c. In figure 3.7a and 3.7b two respectively five more images were removed. The solution with twelve images was the one used when choosing which points we had to go through to cover the whole panorama.

The solution of the traveling salesman problem (shortest traveling distance between the points), is shown in figure 3.8. The problem was solved in 3D, by translating the points from the circular panorama to the corresponding points on the unit sphere. The result of the cardinal splining between the points in 2D is displayed in figure 3.9. To make more sense of the camera motion, the points are also shown in their corresponding 3D points in figure 3.10.

(a) Start solution.  
(b) 2 images removed.  
(c) Solution optimized.

Figure 3.6: Part of results from solving the least amount of images needed to cover the whole circular panorama. The bar in the right in the figures is a measurement of the overlaps in the panorama.
(a) A total of 4 images removed. (b) A total of 7 images removed.

Figure 3.7: Part result and final result from solving the least amount of images needed to cover the whole circular panorama. The final result contained 12 images. The bar in the right in the figures is a measurement of the overlaps in the panorama.

Figure 3.8: The solution to the traveling salesman problem solved by simulated annealing.
Figure 3.9: Cardinal splining between points in $\varphi$-$\theta$ plane.

Figure 3.10: Cardinal splining between points in 3D.
3.2.2 Path Analysis

We analyzed the different paths which resulted in the figures 3.11 and 3.12. From these we can tell that generally the roundabout pattern was the one who performed best. It had the lowest total acceleration, see table 3.2, which we assume is good when thinking of the wear on the camera motors. It also had the second shortest path length. The roller coaster pattern has an even shorter path length but a lot more acceleration.

The patterns were run at the same speed to be comparable. These values were all taken by specifying control points and letting the camera travel between them and logging its position. Then the speed, and later the acceleration, was derived using the smooth noise-robust differentiators described by [12]. We found that we had to smooth the original data as well to suppress noise as much as possible. Therefore these figures are not exact and should rather be seen as indicators of the length and acceleration required to run the patterns.

The symmetry of the frog, flower and the roundabout pattern indicates that rather than using control points a fixed custom program might be more efficient. For these three pattern we could instead define a constant pan speed and vary the tilt at defined time intervals. This was done with the roundabout pattern and the result is shown in figure 3.13. As can be seen the pattern is even more efficient.

![Figure 3.11: Analysis of the frog and the flower patterns. The top graphs are the tilt values and the lower graphs are the pan values. The blue line is the measured pan position, the red line is the calculated speed and the green line is the calculated acceleration, in degrees, degrees/s and degrees/s^2 respectively.](image-url)
Figure 3.12: Analysis of the roller coaster and the roundabout patterns. The top graphs are the tilt values and the lower graphs are the pan values. The blue line is the measured pan position, the red line is the calculated speed and the green line is the calculated acceleration, in degrees, degrees/s and degrees/s² respectively.

Figure 3.13: Analysis of the enhanced roundabout pattern. The pattern is run using a different algorithm than the 4 previous patterns but it is run at approximately the same speed and analyzed using the same analysis program. The blue line is the measured pan position, the red line is the calculated speed and the green line is the calculated acceleration, in degrees, degrees/s and degrees/s² respectively.
Table 3.2: The total amount of acceleration, calculated as the integral of the absolute value of the acceleration. The enhanced pattern is marked in light blue. This pattern is run by another program than the four others and therefore the length and time is not comparable.

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Tilt acc.</th>
<th>Pan acc.</th>
<th>Tilt length</th>
<th>Pan length</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>the Flower</td>
<td>2344.82</td>
<td>978.28</td>
<td>791.8</td>
<td>929.5</td>
<td>6.86</td>
</tr>
<tr>
<td>the Frog</td>
<td>4970.09</td>
<td>1005.20</td>
<td>434.4</td>
<td>1004.24</td>
<td>6.16</td>
</tr>
<tr>
<td>the Roundabout</td>
<td>1104.67</td>
<td>818.74</td>
<td>149.8</td>
<td>791.12</td>
<td>4.25</td>
</tr>
<tr>
<td>the Roller coaster</td>
<td>2859.79</td>
<td>2795.60</td>
<td>343.16</td>
<td>562.72</td>
<td>4.26</td>
</tr>
<tr>
<td>Enhanced Roundabout</td>
<td>695.41</td>
<td>217.73</td>
<td>93.75</td>
<td>1080.26</td>
<td>4.92</td>
</tr>
</tbody>
</table>

3.3 Time corrections

Figure 3.14 shows the same data as figure 2.11 but with the gyroscope data smoothed.

![Figure 3.14: Red values are the difference in gyroscope data in vertical direction per second, blue values are the difference in tilt data per second. The gyroscope data is filtered with a Gaussian filter.](image)

In figure 3.15, the new values for corrected timestamps are shown, and in figure 3.16b one can see the improvement of the corrected timestamps (especially it can be seen for one of the images in the far right) compared to figure 3.16a.
Figure 3.15: Red values are the difference in gyroscope data in vertical direction per second, black values are the difference in tilt data per second. The gyroscope data is filtered with a Gaussian filter and outliers in pan data has been shifted in time.

Figure 3.16: Images compensated by barrel and rolling shutter distortion. a) without correction for timestamps, b) with correction for timestamps. The improvement can be seen mostly at the glass door in the right.
Chapter 4

Conclusions

In the thesis we aimed towards updating a CPI of a half sphere view in real-time. In the sense of receiving a nice representation of the CPI we were quite successful (according to us while looking critically at images). When using a fixed shutter and gain and moving the camera in a reasonable speed, the images were stitched together seamlessly. The rolling shutter compensation helped a great deal as can be seen in the results. In the barrel correction on the other hand it was harder to prove which correction was better to use, and in some cases it was hard to see any improvement at all. This was mostly due to that the effects from the barrel distortion was quite small in the particular camera we used and we only used our eyes to determine what worked best. Also it was hard to determine if it was barrel distortion or other effects that led to mismatches. Some mismatches could for example come from inaccuracy of timestamps, parallax effects or inaccuracy in the rolling shutter compensation.

While the timestamp correction might lead to a better result, the difference won’t be particularly big. As seen in the results, only a few values are big outliers and these are only shifted milliseconds in time (which corresponds to moving images less than 1 degree even if the camera moves really fast). Since one goal was that computations should be done in real-time, one has to put the result of using the computations against not using them. In this sense, it might be a better idea not to use it. The same applies for the barrel correction.

The intensity correction worked good when using automatic settings in the camera. But if the camera moves continuously, as it should do if the CPI should be built as fast as possible, the shutter and gain needs to be fixed (otherwise the images will be blurred from the movement). Using fixed gain and shutter leads to more similar intensities between the images and therefore the intensity correction won’t serve as much purpose (in fact it could lead to, as mentioned in the results, even worse results). To save computation time, this part could probably be skipped in most cases too.

The pyramid blending was one of the most important operations to get a seamless CPI. We discovered that the resolution of the CPI mattered for the details in the image (i.e. an image with lower resolution might keep more detailed information). Suggestions for some improvement on the pyramid part is discussed in 4.2. One of the downsides of the pyramid blending is that it’s a quite heavy operation (filtering both the sent in image and the CPI with a low pass filter). But because of the result it produced, we considered it to useful.
not to be used.

Many of our operations works fast even though we have made all of our work in Octave. The operation that takes the longest time is the transformation of coordinates from the CPI. There is definitely room for some optimization in this and more operations (further discussions in 4.2). We are determined that the CPI can be built in real-time, or near real-time, after optimizations and coding in a different faster language.

Regarding the pattern to move the camera in, the roundabout pattern was the one we used in most test-runs. It ran fast and had the least wear on the motors. Since it could move really fast in the pan direction and since the tilt moved fast between the tilt steps, objects that moved during a run had to move fast to sabotage the CPI blending (distance from the camera also matters). If a movement of an object was seen between the different tilt levels it might lead to more ghosting, but in most cases this was not the case. In the flower pattern the risk of ghosting was larger since we had large overlaps. The frog pattern moved in such a way that we got great overlap for blending but did not revisit parts and risk ghosting in the same way as the flower pattern. But it could not be run at the same speed as the roundabout pattern. It had however significantly more acceleration. Our calculated pattern, the roller coaster, was not used. It had the shortest path but the most acceleration without any apparent benefits compensating for this.

4.1 Applications

One application is to make a small CPI and use as map of the entire scene. Then the operator can click in the map and the camera moves to this location. This map could be placed as an overlay over the camera image. The map could then be updated either automatically or by request from the operator.

Another application is to create a large CPI and monitor a really big area, like a stadium or a large airport terminal. Then the images can be used to see how the flow of people moves about the scene.

We used a spherical projection in this thesis work but other projections could be used. Creating a partial or full normal panorama image could be done quite easily and can be used in scenes where the whole half sphere area is not that interesting to monitor, say a field or a car park. Since the time required to take the images depend on the speed of the camera, reducing the area to cover should increase the potential fps from the camera.

Our algorithm relies on an input image and enough meta data to make the panorama image. This means that we could extend this and stitch images together from several different cameras. For example the Axis camera installation Q6000 have four cameras that could be used to give a full real time panorama image with potentially high fps.

Another application that might be outside the security scope would be to use the zoom-part of the camera. In practice when zooming in we get more information per image and can create more detailed panoramas. Potentially one could create gigapixel images with a single PTZ-camera.
4.2 Future work

Generally we have determined results with ocular evaluation. This is perhaps not always the most reliable way to work. Designing measurement algorithms to determine how good the results are might be a good idea.

The resolution of the CPI effected what things was interpreted as details in the image while blending. So if the resolution was increased it didn't insure a CPI rich of details, instead details could be blurred. One way to improve this would probably be to use more pyramid levels in the blending scheme. This would on the other hand be a downside for computational time.

There might be applications where the time of creating the CPI doesn't matter and all you want is a good quality image. Then more pyramids could be used without any problem. There could be a lot of other improvements if time wasn't an issue. For example, intensity corrections could be made better (using global information), vignetting problems could be partially removed with algorithms, point matching could be involved to improve the matching even further, ghosting effects from moving objects could be dealt with and so on.

Since we fix both shutter and gain while moving the camera it might be a good idea to try to find an automatic way to find what they should be set to. What the gain should be set to could perhaps be found by using global intensity properties from the whole scene. The shutter should only be dependent of which speed the camera moves in and therefore it shouldn't be to hard to find an automatic way to fix this. It should be set so that the images don't get blurred during the run.

As said with the time correction, there were only a few outliers. Instead of focusing on improving the timestamps mathematically, a better way would probably be to go into the core of the camera and find another way, with higher precision, to log the timestamps of both images and pan/tilt positions.

In the results we mentioned that we don't compensate for the rolling. Implementing this in the framework would most likely lead to better matches (especially when moving the camera in high speeds). Another way to get rid of some mismatches is to try to avoid the parallax effects. The most effective way to do this is to physically move the optical center to the rotation center of the PTZ-camera.

In general many of our algorithms can be optimized. For example all of the coordinates in the CPI are currently transformed during the part where we find what pixels to put in the CPI. Instead one could design it so that only necessary pixels around the area where the new image should be put in the CPI would be transformed. Also in the blending, regions of interest could be found to filter much smaller parts of the images. Doing all operations in a faster programming language would speed up all calculations. Some image operations could also be performed in parallel for example in a GPU, further enhancing the speed.

Regarding the patterns that we move the camera in, much more can be done. A lot more effort could be put in finding theoretical patterns. Also patterns not covering the whole half sphere might be used, one could scan only regions of interest. Other projections could also be used, for example a normal panorama or a stereographic projection.
4.3 Acknowledgments

We would first like to thank Axis Communications for the opportunity to do our master thesis work at their company. Their openness and hospitality has been great during this semester.

We would also like to thank Kalle Åström for help and good input during our work.

A big thanks to Björn Ardö who has been a constant support and a patient supervisor. Not much could have been done without him.

Also a thanks to the colleagues who have been helpful when Björn was not around.
API  Application Programming Interface. A predefined framework for communicating between different programs and routines. 12

CPI  Circular Projected Image. 3, 13

FPS  frames per second. 38

GPU  Graphical Processing Unit. Primarily used for creating computer graphics. 39

PTZ  Pan-Tilt-Zoom, indicates the camera movement options. 3
Bibliography


Appendix A

Field trip images

Figure A.1: The parking lot at Axis in Lund. Notice the Parallax effects from the pole resulting in ghosting.
Figure A.2: Inside the University building auditorium.
Figure A.3: Outside Plestra et Odeum in Lundagård.

Figure A.4: Beneath the magnolias in Lundagård.
Figure A.5: Outside the University building in Lundagård.