Image optimization algorithms on an FPGA

Kenneth Ericsson
School of Innovation, Design and Engineering. Mälardalen University

Robert H. Grann
School of Innovation, Design and Engineering. Mälardalen University

Supervisor: Jörgen Lidholm
Examiner: Lars Asplund
2009-04-02
Abstract

In this thesis a method to compensate camera distortion is developed for an FPGA platform as part of a complete vision system. Several methods and models is presented and described to give a good introduction to the complexity of the problems that is overcome with the developed method. The solution to the core problem is shown to have a good precision on a sub-pixel level.
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Start Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Introduction</td>
<td>4</td>
</tr>
<tr>
<td>1.1</td>
<td>Background</td>
<td>4</td>
</tr>
<tr>
<td>1.2</td>
<td>Purpose</td>
<td>5</td>
</tr>
<tr>
<td>1.3</td>
<td>Delimitations</td>
<td>5</td>
</tr>
<tr>
<td>1.4</td>
<td>Conventions, abbreviations and notations</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>Related work and related theory</td>
<td>7</td>
</tr>
<tr>
<td>2.1</td>
<td>Problem formulation</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>Analysis of problem</td>
<td>9</td>
</tr>
<tr>
<td>3.1</td>
<td>Theory of the camera</td>
<td>9</td>
</tr>
<tr>
<td>3.1.1</td>
<td>Lens Theory</td>
<td>10</td>
</tr>
<tr>
<td>3.1.2</td>
<td>Thin Lens Theory</td>
<td>10</td>
</tr>
<tr>
<td>3.1.3</td>
<td>Thick lens Theory</td>
<td>11</td>
</tr>
<tr>
<td>3.1.4</td>
<td>An introduction to camera modelling</td>
<td>12</td>
</tr>
<tr>
<td>3.1.5</td>
<td>Camera coordinate systems</td>
<td>13</td>
</tr>
<tr>
<td>3.1.6</td>
<td>The Pinhole Camera Model</td>
<td>13</td>
</tr>
<tr>
<td>3.1.7</td>
<td>An introduction to affine transformation</td>
<td>15</td>
</tr>
<tr>
<td>3.1.8</td>
<td>The new generic camera model</td>
<td>16</td>
</tr>
<tr>
<td>3.2</td>
<td>Model/Method</td>
<td>16</td>
</tr>
<tr>
<td>3.2.1</td>
<td>Introduction to the vision system</td>
<td>16</td>
</tr>
<tr>
<td>3.2.2</td>
<td>The calibration toolbox</td>
<td>17</td>
</tr>
<tr>
<td>3.2.3</td>
<td>Correction alternatives</td>
<td>19</td>
</tr>
<tr>
<td>4</td>
<td>Solution</td>
<td>24</td>
</tr>
<tr>
<td>4.1</td>
<td>The VHDL component</td>
<td>24</td>
</tr>
<tr>
<td>4.1.1</td>
<td>Fixed-point numbers representation</td>
<td>25</td>
</tr>
<tr>
<td>4.1.2</td>
<td>The LUT file maker</td>
<td>26</td>
</tr>
<tr>
<td>5</td>
<td>Results</td>
<td>27</td>
</tr>
<tr>
<td>6</td>
<td>Summary and conclusions</td>
<td>27</td>
</tr>
<tr>
<td>7</td>
<td>Future work</td>
<td>27</td>
</tr>
<tr>
<td>Appendix 1</td>
<td></td>
<td>31</td>
</tr>
<tr>
<td>Appendix 2</td>
<td></td>
<td>32</td>
</tr>
<tr>
<td>Appendix 3</td>
<td></td>
<td>35</td>
</tr>
<tr>
<td>Appendix 4</td>
<td></td>
<td>36</td>
</tr>
<tr>
<td>Appendix 5</td>
<td></td>
<td>37</td>
</tr>
<tr>
<td>Appendix 6</td>
<td></td>
<td>39</td>
</tr>
<tr>
<td>Appendix 7</td>
<td></td>
<td>40</td>
</tr>
<tr>
<td>Appendix 8</td>
<td></td>
<td>41</td>
</tr>
<tr>
<td>Appendix 9</td>
<td></td>
<td>44</td>
</tr>
</tbody>
</table>
1 Introduction

1.1 Background

Robot vision and machine vision are two major branches focusing towards basically the same goal, to give the machine properties that make them able to make own decisions which decrease the amount of work done by hand to automate some process. Whether the process is to recognize doors, tables or people, cogs, nails or raw-material some basic tools are needed, a sensor or device that gather information from the surrounding environment, a processing unit that translate the gathered information, software that can interpret the information and take an accurate action based on the information and finally some kind of manipulator that is fulfilling the physical action. A good description of machine vision is stated in [1], “Machine vision is concerned with the automatic interpretation of images of real scenes in order to control or monitor machines or industrial processes. The images may be visible light, but could also be of x-ray or infrared energy, and could even be derived from ultrasound information.” This shows the width of vision. Multiple different sensors could be used to derive the information needed for the specific task, as long as the information can be interpreted into something useful e.g. manipulation by grippers on industrial robots or obstacle avoidance by an autonomous robot. Today’s advanced vision systems used in industry is often built with expensive, high precision cameras. This type of camera has a low degree of distortion which mean that the camera give a good representation of reality, the so called object world. To reduce the cost of vision systems, less sophisticated cameras could be used, similar to inexpensive web cameras. The drawbacks of using these low quality cameras are that the system has to compensate for a big amount of distortion and different kinds of distortion. Compensate distortion, could be illustrated as putting glasses on a camera, to find the correct characteristics of the glasses a calibration has to be carried out. In this thesis some methods to compensate image distortion by software, that could be implemented on programmable hardware, FPGA, is presented together with a well-known calibration method [16]. Sources of distortion might be several, but the lens of a simple camera is one of the greatest sources of error, compare the radial and tangential error in Figure 1 and Figure 2 in Appendix 2, also see the plots of error in [1], the lens might incorporate a wide variety of errors e.g. the well known fish-eye lens which has very much so called radial distortion, the fish-eye lens has a very wide viewing angle and thus expose the camera sensor to a big amount of the object world, this might of course be a wanted phenomenon, but could also be a major obstacle when different post processing methods is to be used in the system. Pictures taken with this kind of lens look like the picture in Appendix 6, as can be seen in the example straight lines look bent, this kind of distortion is inherent in all kind of lens systems, the cameras used during experiments and testing in the thesis does not have as severe distortion as with a fish-eye lens.
1.2 Purpose
At the School of Innovations, Design and Engineering at Mälardalen University, a vision system is under development, the system is built according to the picture in Appendix 1. The system is primarily used for robot navigation but is also used in other areas of vision where distance measurement is needed e.g. pick and place [11]; the triangulation [10] capability with the stereo vision system is powerful and could basically be used in any area where a cheap alternative to expensive measurement systems is preferred. The system has some major issues due to the simplicity of the cameras and built in imperfections of the camera carrier circuit board, when image data is collected it contains some degree of distortion. Several different post processing algorithms is and will be implemented on the system such as corner detection, blob detection and object recognition, the different areas of use demands good resemblance according to correct information between the right and left picture data, in the present form the system can’t reach the precision that is desired due to the distortion.

1.3 Delimitations
The calibration method that is used to extract the camera parameters is performed offline which means before the system is put in use, the parameters is then known in advance.
### 1.4 Conventions, abbreviations and notations

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FPGA</td>
<td>Field Programmable Gate Array</td>
</tr>
<tr>
<td>FPD</td>
<td>Field Programmable Device</td>
</tr>
<tr>
<td>ASIC</td>
<td>Application Specific Integrated Circuit</td>
</tr>
<tr>
<td>CPLD</td>
<td>Complex Programmable Logic Device</td>
</tr>
<tr>
<td>OS</td>
<td>Operating System</td>
</tr>
<tr>
<td>SRAM</td>
<td>Static Random Access Memory</td>
</tr>
<tr>
<td>DLT</td>
<td>Direct Linear Transformation</td>
</tr>
<tr>
<td>PCM</td>
<td>Pinhole Camera Model</td>
</tr>
<tr>
<td>GCM</td>
<td>General Camera Calibration</td>
</tr>
<tr>
<td>CMOS</td>
<td>Complementary Metal Oxide Semiconductor</td>
</tr>
<tr>
<td>LIV</td>
<td>Light Intensity Value</td>
</tr>
<tr>
<td>LUT</td>
<td>Look-up Table</td>
</tr>
<tr>
<td>KK</td>
<td>Camera parameter matrix</td>
</tr>
<tr>
<td>kc</td>
<td>Distortion parameter matrix</td>
</tr>
<tr>
<td>f</td>
<td>Focal Length</td>
</tr>
<tr>
<td>CCx, CCy</td>
<td>Principal Point</td>
</tr>
<tr>
<td>(Xw, Yw, Zw)</td>
<td>World coordinate system</td>
</tr>
<tr>
<td>(Xo, Yo, Zo, αo, βo, γo)</td>
<td>Object coordinates and angles</td>
</tr>
<tr>
<td>(Xc, Yc, Zc, αc, βc, γc)</td>
<td>Camera coordinates and angles</td>
</tr>
<tr>
<td>X_i, Y_i</td>
<td>Incoming pixel coordinate</td>
</tr>
</tbody>
</table>
2 Related work and related theory

A short introduction is given to the computing hardware used in the thesis the so called Field Programmable Gate Array, FPGA; which is one type of field programmable device, FPD. Other types of programmable logic are presented in [1], where the FPGA is presented more thoroughly. The development within the area of FPDs is very fast, one improvement was the possibility to use a programming language, for example VHDL, Verilog or Handle-C. These decrease implementation time compared to traditional scheme construction. VHDL and Verilog have opened a door to more complex circuit constructions; it is possible to implement processors, operating systems, and connections between hardware and OS.

One of the great strengths of FPGAs is the possibility to build highly parallel systems, they are not as fast as conventional general processing units considering clock frequency, but the possibilities to perform a big amount of calculations every clock cycle make the FPGA a good replacement for micro controllers which are the most used technology in embedded and small real-time systems. The procedure, from code to a working program on FPGA is as follows, programs are created in a suitable tool, the code is then synthesized which is similar/equal to compilation, the next step is to connect in- and output pins of the FPGA to the external devices, final step is to load the synthesized code into the FPGA. A deeper description of how the code looks like and how programs are built is found in [4]. FPGAs are mostly used in products that are produced in small series and in prototype constructions. If big series are to be made the VHDL/Verilog construction can be statically put into an Application Specific Integrated Circuit, ASIC. The difference in market price is still large between FPGAs compared to ASICs, which is why large series are made in ASICs although they are impossible to reprogram. The difference in price is slowly decreasing, resulting in the possibility to more often use FPGAs in larger consumer product series but as stated still uncommon. FPGA systems are found in many different areas of technology, e.g. airplanes, robotics, military equipment, cars, hand held media devices, set-top boxes and cell phones.

Camera models, calibration and correction methods have been well studied in research, [5][6], [8], [16], and several introductions to the methods can be found through internet. Only a handful has a similar concentration as our thesis present, the following papers cover similar work, [12], [13], [14], [21], [23]. David Eadie et. al. [14] present several approaches to pixel correction; they investigate the possibility to use Field Programmable Gate Arrays to enhance the calculation speed of the commonly known heavy calculations needed to correct images. Further more they examine whether it is suitable to use fixed point, floating-point and logarithmic number representation for their image processing algorithms, performance results are presented in terms of execution time and resource requirements. The results look promising, see Table 1 and strengthen the theory that FPGAs is well suited for this type of image processing, pipelining is as they state the key to extra high performance, and this can easily be seen in the results. The conclusion of the work is pointing towards one common denominator; FPGAs can reach speeds that are multiple magnitudes greater than possible with general processing units. As seen in Table 1 the difference in results is quite astonishing, the lookup table implementation is preferred when resources is to be used for further calculations, one of the few major issues with the methods is the fact that they use a big amount of SRAM.
Table 1: Content taken from David Eadie et. al. [14], the FPGA implementations clearly have an advantage over the GPU in all aspects.

<table>
<thead>
<tr>
<th>Implementation</th>
<th>Area (%)</th>
<th>SRAM (MB)</th>
<th>Clock speed</th>
<th>Time (ms) 600x600 pixels</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pentium 4 real time calculation</td>
<td>n/a</td>
<td>n/a</td>
<td>1.7 GHz</td>
<td>780</td>
</tr>
<tr>
<td>Pentium 4 lookup table</td>
<td>n/a</td>
<td>n/a</td>
<td>1.7 GHz</td>
<td>760</td>
</tr>
<tr>
<td>Non-pipelined FPGA real time calculation</td>
<td>63</td>
<td>5</td>
<td>10 MHz</td>
<td>360</td>
</tr>
<tr>
<td>Non-pipelined FPGA lookup table</td>
<td>7</td>
<td>7</td>
<td>22 MHz</td>
<td>130</td>
</tr>
<tr>
<td>Pipelined FPGA real time calculation</td>
<td>64</td>
<td>5</td>
<td>25 MHz</td>
<td>15</td>
</tr>
<tr>
<td>Pipelined FPGA lookup table</td>
<td>6</td>
<td>6</td>
<td>34 MHz</td>
<td>11</td>
</tr>
</tbody>
</table>

Álvaro Hernández et. al. [13] proposes an architecture that is implemented in FPGA. The architecture has some core features from [14] and has a similar working flow as in [15], but the work present a closer to solution architecture that is well suited for an FPGA. The only concern is that the paper does not present an estimation of resource requirements, nor performance figures, the proposed implementation could on the other hand be implemented for evaluation thanks to the well-described architecture. The conclusion of the authors work is that this kind of architecture could play a great role in low cost, high-resolution photography, especially as the architecture is able to handle dynamic zoom and focus with an extension.

Janne Heikkilä and Olli Silvén [16], present a four-step calibration method. They give a good view of how to derive parameters and pinpoint which parameters are needed for a correct calibration, both extrinsic and intrinsic. They conclude that direct linear transformation, DLT, and similar direct methods are computationally fast but have a couple of disadvantages, distortion cannot be added to the calculation and therefore not calculated in most cases, still a couple of solutions to this problem have been proposed as extensions to the DLT, another issue is the accuracy which according to Heikkilä and Silvén is below acceptable levels, the accuracy problem come from the presence of noise which is always a component when dealing with measurements. The paper suggest a combination of DLT and nonlinear methods to calculate parameters, the DLT is used to get good initial values that is used as starting point for an iterative method, in this work the Levenberg-Marquardt optimization method is used, the DLT estimation decrease the risk of ending in a local minimum which would destroy the calibration. The experimental part is concentrated on correcting images, not on calibration that the writers show in a prior work [5], the result show that the error is below pixel level, sub pixel precision, in realistic conditions errors below 0,01 pixels is expected according to results in their prior work.

### 2.1 Problem formulation

The objective of this thesis is to get familiar with some of the common calibration techniques and methods used in vision- and camera systems of today, differences and suitability is compared, a method or several methods to compensate distortion will be developed and implemented in programmable hardware, FPGA, focusing on throughput and footprint. Results from the master thesis will be added to the research group at the School of Innovations, Design and Engineering at Mälardalen University, the vision system developed in the research group has no correction for camera distortion which results in problems to map the 2D images to 3D models. Another problem will also occur, if a point in a stereo/multi camera system is detected
and matched as the same point it will start to differ when it comes closer to the edges of the picture, see figure 3 in Appendix 1.

The goals of the master thesis were to:
- Get a general understanding of camera calibration
- Present several camera models and their differences
- Develop correction methods for FPGA
- Implement a suitable method

Calibration and compensation has been well covered by research, but few have worked with speed and footprint in mind, thus hard to use in inexpensive and relatively slow hardware. This is one of the greatest challenges of this project, to develop a correction method suitable for the hardware in hand, as any one can imagine it is a great task to be able to use a similar working flow of [15] or at least have a result similar or better than that program, especially as the hardware used in the thesis is about 20 times slower with regard to clock speed, than a common personal computer, on top of this the system is supposed to do different post processing of the picture data e.g. corner detection [17], spin-image [11].

3 Analysis of problem

3.1 Theory of the camera

This text is a short description of how a camera works, more complete information can be found in for example “Computer Vision -- A modern approach” [18]. The simplest camera is the pinhole camera (camera obscura), Figure 1. The pin-hole camera was first described in documents from Mozi (470 BC to 390 BC), a Chinese philosopher. The first realized version is known from the sixteenth century and uses a small hole instead of a lens. Light reflected from an object in this case a candle goes through a pinhole and projects on the image plane. Observe that the image appears upside down on the image sensor. In theory only one light ray from every point of the object passes through the pin hole. Today it is more common that a camera uses a positive lens instead of a simple pin hole. To use a lens in a camera gives advantages in form of higher resolution and higher sensitivity but also problems as short focus depth due to a large aperture. Most common lenses have anomalies that cause distortion. It is hard to manufacture a good lens and that’s why a good lens often is rather expensive. A good lens is often a combination of several lenses with different properties mounted into one unit.

![Figure 1: The camera obscura.](image)

There are several different kinds of distortion that can occur in this report we will show two of the most common distortions radial and tangential see Figure 2.
3.1.1 Lens Theory

To understand optics we will use a simplified model of how a lens works. The thin lens model is simplified in that way that you can see the lens as a layer that are able to deflect the rays only one time and only in the middle of the lens, could be seen as an atomic membrane. The opposite is the thick lens that deflects the rays two times both on the way into the lens body and on the way out of it. The thick lens model is more complex and has many parameters. One parameter is the aperture or light intake width. The aperture controls the focus zone, more about this later.

3.1.2 Thin Lens Theory

In a system with a thin lens, diverted rays from an object will focus on the image plane, Figure 3. The light rays’ way through the lens is depending on the angle of approach. Light rays from the object that are parallel to the optical axis will deflect through the focal point on the image plane side of the lens. Light rays that intersect the optical axis in the middle of the lens will not be deflected. Light rays that intersect the optical axis on distance $f$ in front of the lens will go in parallel with the optical axis after the ray has been deflected through the lens, see Figure 3. In this figure the aperture is big which means that a lot of light rays from the same object point end up onto the image plane; this will result in increased resolution but gives, on the other hand, a shorter focal length.
Figure 4: Light rays way through a thin lens with small aperture.

If the aperture is small the camera behaviour will be like a pin-hole camera, this is because rays that have their intersection of the optical axis different from the middle of the lens will be hindered by the small aperture, see Figure 4. The good thing with small aperture is an increased focal depth; the drawback is lower resolution and lower light sensitivity.

Figure 5: Parallel rays.

Light rays parallel to the optical axis will be deflected into the focal point, Figure 5. This means that \( \frac{1}{a} + \frac{1}{b} = \frac{1}{f} \Rightarrow a \to \infty \Rightarrow b = f \) see Figure 4.

3.1.3 Thick lens Theory

Thin lenses is only a simplified theoretical model and have very little to do with real lenses. Even though the model is simple it can be a seed to a complex camera model. Real lenses are more complicated and therefore they have a special model called thick lens model.
The thick lens model is more complicated to define. Only light rays that travels on the optical axis, the dashed line, remains undeflected see Figure 6. That means that the focal point is harder to find and the focal point cannot be seen as a single point, rather like a floating focal zone, see Figure 7. Here we instead can talk about a point of less confusion. The Point $P$ in Figure 7 will appear as a circle due to the picture plane position, the point of so called less confusion is found at the dashed line, this is where the most information co-occur, another phenomenon that is modelled is the so called vignetting. If a more concentrated point is expected you can use several lenses in a lens system with unique characteristics. The drawback of the thick lens is that some of the light rays that end up on the edge or corner of the lens will not pass the lens system correctly, the rays that pass the lens will end up beside the image sensor, and this effect is the vignetting, see Picture 2 in Appendix 6.

3.1.4 An introduction to camera modelling.

A camera model is the connection/interface between the real world that is represented in 3D and the image represented in 2D that will be reproduced on the image sensor. A camera model is a mathematical model of the characteristic associated with a camera. Included in the camera
model is the lens and sensors characteristic i.e. distortion that comes from inconsistency built in, both in the lens and the sensor. The distortion can also come from misplacement of both lens and sensor. A camera model can be simple or it can be complicated but one common thing is both can predict the light rays’ way from the object into the image sensor and the other way around. Some differences between a simple and a more complex model can inter alia be that the more complicated model manages zoom, focus and wide angle lenses like fish-eye lenses a lot better. Notable is that you lose one degree of freedom when you go from 3D into 2D which means that it is impossible to go back to 3D again from only one image. If the goal is to get the 3D information you need two cameras imaging the same thing from two directions.

Camera models contain a number of different parameters called intrinsic and extrinsic, the intrinsic parameters describe the light rays’ way through the lens onto the image sensor, whilst the extrinsic parameters describe the camera position and heading in the world coordinate system. The intrinsic camera parameters are different between different camera models while the extrinsic parameters often are the same.

### 3.1.5 Camera coordinate systems.

When a camera model is to be defined the first things that have to be done is to define all the involved coordinate systems, Figure 8. We need to define one global coordinate system to represent the world, \((X_w, Y_w, Z_w)\) another coordinate system including its rotations is needed for each object, \((X_o, Y_o, Z_o, \alpha_o, \beta_o, \gamma_o)\), the camera coordinate system and the angles of rotation in each axis, the so called heading, is defined as \((X_c, Y_c, Z_c, \alpha_c, \beta_c, \gamma_c)\), see Figure 8. The last two definitions are known as the extrinsic camera parameters and is seen in Figure 8 as the external coordinate system of the camera; inside the camera there is another coordinate system, the image coordinate system that is a 2D coordinate system and a part of the camera model and the so called intrinsic camera parameters.

![Figure 8: Several coordinate systems involved.](image)

### 3.1.6 The Pinhole Camera Model.

The most uncomplicated camera model is the Pinhole Camera Model, PCM. The PCM is a model of the first camera built out of a box and a little hole as a thin lens. The principal of a Pinhole Camera is shown in Figure 9. The light rays way from the object to the optical sensor can be seen as a straight line from the object through the pinhole onto the image sensor, see Figure 9. The Pinhole camera technique is modelled as two linear equations one for \(X\) and one
for $Y$. Included in the linear equations (0.1) and (0.2) is the focal length $f_{x,y}$ as a scale factor and the principal point $CC_{x,y}$ as an offset.

$$X_i = X_o \cdot f_x + CC_x \quad (0.1)$$

$$Y_i = Y_o \cdot f_y + CC_y \quad (0.2)$$

This mapping between the object space and the image space is also called affine transformation; affine transformation will be described later in this report. Often matrices are used for this kind of calculation, with an affine transformation matrix multiplication it is possible to both do the scaling, rotation, translation and skew in one matrix calculation. $X_i, Y_i$ is the normalized position on the image plane. $X_o, Y_o$ is the position on the image sensor in pixels. The equations are used to project the light rays from the object onto the image sensor, the camera model can then be added with more feature parameters as for example radial and tangential distortion. In the Matlab toolbox later used in this work polynomial equation of fourth order is used to model radial distortion, the model also include extrinsic parameters.

Figure 9: Principal of the Pinhole Camera Model, PCM.
3.1.7 An introduction to affine transformation.

Affine transformation is when a set of points formed as a figure or as in this case an image is manipulated so its form can be restored. Both 2D and 3D objects can be manipulated as for example a cube that can be scaled, rotated, moved and skewed and then restored to its original looks. As a calculation example the 3D point in Figure 10 is used to demonstrate how and what could be done with the help of affine transformation. A good approach is to use matrices when calculating affine transformations.

![Figure 10: A 3D object point is mapped onto the image plane.](image)

Let \( P \) be a 3D object point as a 1 by 4 matrix and \( T \), translation matrix, is the movement in each directions as a 1 by 4 matrix. \( P' \) is the result from this matrix addition in form of a 1 by 4 matrix, see (1.1).

\[
P' = P + T \quad (1.1)
\]

In this example \( P \) is the same 3D object point as a matrix 1 by 4 and \( R \) is the scaling, rotation and/or skew in each direction as a 4 by 4 matrix. \( P' \) is the result from this matrix multiplication in form of a 1 by 4 matrix, see (1.2).

\[
P' = R \cdot P \quad (1.2)
\]

It is possible to perform several transformations in one matrix calculation according to (1.3).

\[
\begin{bmatrix}
P'_x \\
P'_y \\
P'_z \\
1
\end{bmatrix} =
\begin{bmatrix}
\delta & s1 & s2 & T_x \\
\kappa & s3 & s4 & T_y \\
\lambda & s5 & s6 & T_z \\
0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
P_x \\
P_y \\
P_z \\
1
\end{bmatrix}
\]

The scaling, skew and rotation is applied on the 3 by 3 positions in the upper left-hand corner of the 4 by 4 transformation matrix, the translation is applied on the \( T_x, T_y \) and \( T_z \) parameters. Below is a description of the different transformation parameters that could be applied.

- \( \delta \) Scaling in X
- \( \kappa \) Scaling in Y
- \( \lambda \) Scaling in Z
- \( s1 \) X skew in respect to Y
- \( s2 \) X skew in respect to Z
- \( s3 \) Y skew in respect to X
- \( s4 \) Y skew in respect to Z
- \( s5 \) Z skew in respect to X
- \( s6 \) Z skew in respect to Y
The affine transformation is in our work used to transform a point in the object world to a point on the image plane.

### 3.1.8 The new generic camera model.

The Generic Camera Model, GCM, is an approach to do a camera model that can model properties like focus, zoom, big aperture and wide angle lenses. The concept of the model is seen in Figure 11. Anders Ryberg is this models inventor, more information about the GCM can be found in [19].

![Figure 11: The General Camera Model, GCM.](image)

### 3.2 Model/Method

The vision system that has been used during testing is presented below; in the rest of the chapter the methods and tools will be presented.

#### 3.2.1 Introduction to the vision system

Base of the vision system is the FPGA motherboard; the board includes, inter alia, a Xilinx Virtex-II XC2V8000 eight million gate equivalents FPGA, for further specifications see [22]. 512MiB 133MHz mobile-SDRAM memory and USB connectivity via an external FTDI FT245R chip. The cameras, up to four, is connected via a parallel interface, the camera carrier boards include the CMOS sensor and mounting holes for different lense holders, Appendix 1 and Appendix 4. The sensor chip, MT9P031, manufactured by Micron.
uses CMOS technology, interesting key performance features is found in Appendix 3. The absolute maximum frame rate that can be achieved with the sensor is 123 fps in VGA (640x480) mode, maximum rate at full resolution, (2592x1944), is 14 fps. In Figure 12 the block diagram of the sensor is shown.

The different signals represented in the figure as arrows pointing in to and out from the sensor boundary is used to communicate, set up and read the sensor. Most of the signals are physically connected to the FPGA board through I/O pins; the specific meaning of the different signals is presented in the data sheet of the sensor, most interesting is the D$_{out}$[11:0] signal that carry the pixel light intensity data. A picture of the complete system mounted for testing are found in Appendix 1.

### 3.2.2 The calibration toolbox

There are several methods of correction that will be presented with their benefits and drawbacks, and if the method is suitable for a certain hardware or area of use this will be presented as well. All the methods that is presented have a pre-calculation stage made off line, there for before a correction of a picture can be done it is necessary that the camera model parameters is known. There are several methods to obtain those parameters, in this thesis the Matlab toolbox from [15] is used for collecting data from pictures of a checker pattern plane. The toolbox is very useful and produces the camera parameters wanted and estimates the calculation errors. The input to the toolbox is a set of pictures from different angles see Figure 13; the output is the camera parameters presented as seen below in Table 2. The data can then be used to correct the pictures used to obtain the parameters or all other pictures taken with the same camera. The toolbox is built on methods presented in Zhengyou Zhang “Flexible Camera Calibration By Viewing a Plane From Unknown Orientations”, [20], and Janne Heikkilä and Olli Silvén’s paper “A Four-step Camera Calibration Procedure with Implicit Image Correction.”, [16].

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Focal Length, fc, pixels</td>
<td>[511.82267, 506.45123] ± [11.28509, 12.19115]</td>
</tr>
<tr>
<td>Principal point, cc, pixels</td>
<td>[323.15503, 208.70276] ± [13.28817, 12.38071]</td>
</tr>
<tr>
<td>Distortion, kc</td>
<td>[-0.40015, 0.16697, -0.00155, 0.00010, 0.00000]</td>
</tr>
<tr>
<td>Uncertainties</td>
<td>± [0.03214, 0.06170, 0.00529, 0.00366, 0.00000]</td>
</tr>
<tr>
<td>Pixel error, err</td>
<td>[0.71862, 0.28875]</td>
</tr>
</tbody>
</table>

Table 2: Example of camera parameters.
A short summary of how the toolbox works:
The algorithm needs a couple of thousand point-couples to calculate a good camera model. A point-couple is two points, one on the sensor plane and one on the object plane. To get one point-couple you have to make a connection between the two, the Matlab toolbox can manage to make those point-couples with help from the user. You have to load a set of calibration images into Matlab where the user needs to point out the corners, Figure 14, of the checker calibration pattern for each picture. It is important that the pictures have as many different angles as possible to make the camera model prediction better and more correct. After the corners are selected the toolbox tries to find every corner between the selected corners. The user has to control that everything is correct; when everything is fine the user starts the calibration, which is automatic.

Figure 13: 20 pictures taken from different angles to cover as many positions as possible.

Figure 14: An example picture of how to pinpoint the corners of the checker test pattern.
3.2.3 Correction alternatives

In Figure 15 the basic concept of image correction can be seen, this figure will be referenced to describe the distinctive features of the different methods of correction, amount of error and type of error due to the method will be presented as well.

![Figure 15: Conceptual figure of image correction.](image)

Online correction calculation

The camera and distortion parameters known in advance is used to do calculations on the incoming picture data, a similar process as with the Matlab toolbox is performed, some of the major issues with this method is that a big input buffer and a big output buffer is needed due to the bilinear interpolation of pixels and the serial data. The ellipse in Figure 15 represents the data that should be fitted to the whole outgoing correct image, as one can imagine the data in the incoming frame is not sufficient to fill the outgoing frame as the data is stretched out, with a interpolation technique, on more pixels then the amount that is moved i.e. the amount of moved pixels are less then the outgoing frame $n \times m$ pixels. Correction of an image is performed as follows, $x_i$ and $y_i$ coordinates of the outgoing frame is used to calculate which pixel data, light intensity, from the distorted picture that should be moved to the $x_m$ and $y_m$ coordinate. The first calculation that needs to be done is the affine transformation from correct pixel coordinate $x_i, y_i$ to the coordinate that the pixel data should be derived from. Matrix KK is the so called camera matrix, parameters in this matrix is derived from the Matlab toolbox, precise descriptions of the parameters is found in [9].

$$KK = \begin{bmatrix} fc(1) & \alpha & fc(1) & cc(1) \\ 0 & fc(2) & cc(2) \\ 0 & 0 & 1 \end{bmatrix}$$

The inverse camera model is applied to add the inverse mapping, (2.1), between the theoretically correct pixels in the outgoing correct frame, and the incoming distorted frame:

$$KK^{-1} \begin{bmatrix} x_i \\ y_i \end{bmatrix} = \begin{bmatrix} x_i \\ y_i \end{bmatrix} \quad (2.1)$$

$$r^2 = x_i^2 + y_i^2$$

$$a_1 = 2x_i \cdot y_i$$

$$a_2 = r^2 + 2x_i^2$$

$$a_3 = r^2 + 2y_i^2$$

The implicit inverse model include radial and tangential distortion according to the following calculations.

$$rad = 1 + kc(1)r^2 + kc(2)r^4$$

$$tad_x = kc(3)a_1 + kc(4)a_2$$

$$tad_y = kc(3)a_1 + kc(4)a_3$$

(2.2)
The last affine transformation into pixel coordinates, (2.3), include the distortions, the $x_{out}$ and $y_{out}$ coordinates are representing the coordinates where the data of the correct pixel is to be derived from.

$$KK \cdot \begin{bmatrix} x_{rad} + \text{tad}_x \\ y_{rad} + \text{tad}_y \\ 1 \end{bmatrix} = \begin{bmatrix} x_{out} \\ y_{out} \\ 1 \end{bmatrix} \quad (2.3)$$

The last step before correction is to perform bilinear interpolation \cite{8}; the reason is to fill the missing information due to the stretching and fitting of the correction procedure mentioned before. To begin with the $x_{out}$ and $y_{out}$ values is rounded off to get the coordinates of pixel A, see figure below. Pixel A, B, C and D consist of a position and a light intensity value i.e. the numerical representation of the light intensity.

$$LIV_{X_{in},Y_{in}} = LIV_A \cdot (1 - (y_{out} - y_A)) \cdot (1 - (x_{out} - x_A)) + LIV_B \cdot (1 - (y_{out} - y_A)) \cdot (x_{out} - x_A) + LIV_C \cdot (y_{out} - y_A) \cdot (1 - (x_{out} - x_A)) + LIV_D \cdot (y_{out} - y_A) \cdot (x_{out} - x_A)$$

The weighted Light Intensity Value, LIV, of the pixels surrounding $x_{out}$ and $y_{out}$ will be moved to position $x_{in}$, $y_{in}$, in the outgoing image, this procedure need to be done to each pixel. As can be seen in the bilinear interpolation calculation, \cite{7}, four pixels around the pixel under inspection $(x_{out}, y_{out})$ are needed to move the weighted data to the corrected position $x_{in}$, $y_{in}$. Another important notice is the fact that the incoming picture data is serial and read from the upper left corner, thus the data until the first pixel that should be moved in a row is discarded. This method needed a big input buffer the reason is the awkward process that begin with the theoretically correct coordinate $(1, 1)$, the data used for that pixel is read several rows later than the row of the first pixel that actually could be moved; an example is shown in Figure 16 below. To keep the order of the pixels intact the system needs to store several pixels in the buffer. In the example figure it seems like a small problem, but in a real world situation the distortion could be way worse, thus the difference produces a need of a big input buffer. To estimate input and output buffer some pre-calculations need to be done, maximum and minimum $y$ value of the first and last outgoing rows need to be calculated. The maximum value of $\Delta Y_1$ and $\Delta Y_2$ described in Figure 16 and Figure 17 below is used to describe the calculation of the buffer size.
Figure 16: Concept and example of how pixels are moved.

Figure 17: Concept and example of how output buffer could be calculated.

The good points of the online correction calculation method is the precision which keep a sub pixel level of approximately 0.01 pixels [16], another good thing is that even though it requires a large amount of memory, it consume less memory than the simple lookup table method, see [13], problems of the procedure is the relatively huge amount of FPGA resources needed, several multipliers is needed and the amount will increase with the amount of parallelism.

**Lookup table, Basic technique**

The second method that is analyzed and further developed is the idea of using lookup tables to perform direct correction of the incoming pixels. As previously mentioned in the related work the lookup table technique has already been exploited but only simulated without real life constraints such as limited amounts of memory. Several versions have been developed based on the most simple lookup table technique; the concept of the technique is shown below, Figure 18. The same pre-calculation as in the prior method have to be carried out to make it possible to calculate which pixels should be mapped and moved into the correct image. When all parameters is known every pixel in the correct image is pre-processed through the affine transformations and calculations.

As mentioned earlier the most basic look-up technique is to map the incoming serial stream of pixels to a certain movement into the new, correct, image. A couple of drawbacks are the bad scalability; if the resolution is changed the whole map has to be re-calculated, the second drawback is the amount of memory that easily can exceed 2MByte when high resolutions is used, the amount of reading from memory will also slow the system down as the readout will intrude on other post-calculation algorithms that utilize the same memory. With this in mind some other techniques had to be developed to minimize the memory usage.
Figure 18: Basic concept of how the simple look-up table technique work, table $t_1$ show the incoming pixel data, $t_2$ include the correction of the incoming pixel, the crossed boxes show an example of pixels that shouldn’t be moved.

**Compressed tables**

A good alternative to the simple table is to apply a compression algorithm on the table. As can be seen in Figure 16 and Figure 17 the only data that should be moved to the corrected image is the colored area, the other incoming pixels are as an example marked with cross in Figure 18, pixels that should not be moved. Big parts of the table will include the crossed pixels and can thus be chunked together to just one start and stop pixel decreasing the amount of needed memory considerably, the same procedure applies to the whole table. Gain in decreased memory usage is directly related to the amount of distortion, the more distortion the higher the memory reduction will be. One of the other big benefits of the compressed table is decreased amount of reading, it’s easily understood that by just reading two consecutive rows of pixels a lot of pixels could be moved or dispatched. Several other compression algorithms could probably be used but will not be covered in this work.

**Support tables**

Another possibility is to use look-up tables containing pre-calculated parts of the calculations in the online correction calculation chapter, 3. This is one way of decreasing the resource need of the correction algorithm. The idea of the table and how to access parts of the table is shown in Figure 19. The index is based on the squared radius of the incoming pixel, the pixel that should be moved; this calculation is easily performed with only two multiplications $x_n^2 + y_n^2$.

$$r^2 \text{- index}$$

| $r_1$ | $r_2$ | $r_3$ | $r_4$ | $r_5$ | $r_6$ | $r_7$ | $r_n$ |

Figure 19: Lookup table $r$ with $r^2$ as index, used instead of the square root function.

In Figure 20 another table is shown with the correcting movement, $\Delta r$, of a pixel with its given radius, the radius could either be calculated internally in the FPGA or extracted out of a table similar to Figure 19. As origin, incoming pixel coordinates $x$ and $y$ and the movement $\Delta r$ is known it is possible to calculate the new correct coordinates according to Figure 21.
The radial correction $\Delta r$ is calculated with an iterative process, this is called forward mapping [16], the opposite to the backward mapping used in the Matlab toolbox. To build up the table with $\Delta r$ the following process is performed; also see the pseudo code below. Calculate with the iterative method the new, correct position of the pixels; subtract the radius of the original pixel from the new radius to get all of the correction radiuses matched to original radius according to Figure 20.

FOR All pixels
    X = Xin_n * Scale_factor
    Y = Yin_n * Scale_factor
    FOR n_Times
        r_2 = X^2 + Y^2
        k_radial = 1 + k1 * r_2 + k2 * r_2^2
        X = Xin_n / k_radial
        Y = Yin_n / k_radial
    END
    X_store(n) = X * Scale_factor
    Y_store(n) = Y * Scale_factor
END

The iterative method is the only way of directly solving the movement of distorted pixels to new corrected positions, else way the Matlab toolbox backward mapping have to be done to calculate the new positions. The major advantage of the iterative method is that the close to exact position is given in a decimal format, as an example see Figure 22 where the crosses mark the discrete position and the circles mark the real pixel positions. The non discrete positions could in post processing stages be used as is to gain better precision compared to interpolated, discrete positions, or the non discrete pixels could be interpolated into discrete positions to get visible pictures out of the system. The major drawback and the reason to why the iterative method is not used instead of the look-up table is the iterations itself, it takes to much time to step through the iterative steps, especially if a lot of features are to be moved.
When $\Delta r$ and $r$ is known it is possible to move the light intensity data according to (3.1), the new $x$ and $y$ position of the pixel or feature:

$$X_{\text{new}} = x + \frac{x \cdot \Delta r}{r}$$

$$Y_{\text{new}} = y + \frac{y \cdot \Delta r}{r}$$

(3.1)

![Figure 22: Example of corrected pixels.](image)

4 Solution

The method that is chosen for testing and validation is based on the last method described in the Model/Method chapter, 3, what we call support tables. The reason to why this method is chosen is the combination of high precision, relatively low memory usage and decreased resource requirements compared to the online calculation correction method, [13]. Compared to the other LUT methods the chosen method utilizes less memory and due to the fact that no interpolation to pixels will be done as the correction of position only will be applied to features the precision is kept high. In our solution the new position of the features will be represented with fixed-point values instead of integer pixels. Our solution uses the Calib Toolbox [15] that supplies several parameters that describe the amount of distortion radial and tangential plus the principal point. In this solution only the radial distortion will be compensating for. The tangential distortion is harder to compensate for because it is depending on $X$ and $Y$ separated. A solution for compensating tangential distortion is more complex than the compensating for radial distortion. The amount of tangential distortion is within the range of 0.01 to 0.1 pixels in our example. See Appendix 2 figure 1. The tangential distortion is equal to the pixel error that can be seen in the result from the correction algorithm chosen, therefore it is not motivated to compensate for the tangential distortion.

4.1 The VHDL component

The component is rather simple; the correcting method is described later in this chapter. The components in data is pixel data of the feature position and a signal called “new_corner” tells the component when the pixel data is valid. Reset and the serial clock are also needed. Out from the component comes the new corrected feature position in fixed-point representation and a signal telling when the result is valid. A correction sequence starts with the new_corner flag and the $X$ and $Y$ coordinates supplied to the component. At that moment a counter starts and when the calculation is done the results comes out as new $X$ and $Y$ coordinates and the signal new_coordinat will be active.
4.1.1 Fixed-point numbers representation

If fractional values is needed it is preferred to use fixed point representation in FPGA. Compared to floating point representation fixed point is faster and easier to use. If floating point is used some kind of hardware support is preferred. In our case the extra amount of resources a Floating Point Unit, FPU, would consume in an FPGA cannot be motivated, the reason is shown in the results chapter, 5. In fixed-point representation the number is divided in two parts, integer part and fraction part. The name fixed-point means that the size of each part is fixed and must be chosen carefully according to the range and the resolution needed. The example showed here uses 8 bits for the integer part plus 1 bit for sign and 7 bits for the fraction part. The resolution in this example is $\frac{1}{128} = 0.0078125$. The highest value possible to represent is 255.9921875 and lowest -256.000000. Fixed-point numbers can use all ordinary arithmetic functions such as +, -, /, * similar to ordinary integers, without restrictions.

The values stored in the table for each position is $r_\Delta$ scaled up with 8192, because the initial value for $r_\Delta$ is within the range of 0.002 to 0.1 and it is more practical to use integers in the table, to solve this in the FPGA later on the fixed-point method will be used. One of the major reasons to why $\frac{\Delta r}{r}$ is stored instead of only $\Delta r$ is to save space and decrease the amount of read cycles in the lookup table. The table content now is between 2 and 809; the drawback is that the result needs to be scaled down after calculation with the same scale factor i.e. 8192, that means an extra down shift is needed, in this case 13 steps.

The index for the lookup-table is $r^2$ measured out from the principal point. Example, a point $P(10,20)$ and the principal point $Q(300,200)$ will generate an index according to
\[ I = (10 - 300)^2 + (20 - 200)^2 = 116500 \] this means that if the point is near the edge, \( r^2 \) is a large number and that will demand a very big lookup-table. The solution is to scale \( r^2 \) with some factor; our solution is to divide 116500 with 512, which give 227 after the fraction part is cut. With this method a table for an ordinary 640 x 480 picture will end up in around 340 values. The drawback here is that in the end of the table there will be some positions containing zero, starting from the middle of the LUT and increases towards the end, the reason is the quadratic index, this problem is solved with an interpolation between the value before and the value after the current empty “zero position”, see (4.1).

\[
S_n = \frac{S_{n-1} + S_{n+1}}{2} \quad (4.1)
\]

The reason to why this interpolation gives a satisfying result is because the function represented in the LUT is close to linear.

### 4.1.2 The LUT file maker.

The lookup table used for the undistortion module in the FPGA can be produced directly out from Matlab by running our script-file `Lookup.m`. The output file is a vhdl source file and can be included directly into the FPGA programming tool. It is possible to set some parameters in the script, from where the input data will come from Matlab’s workspace or directly from the script file. There are also possibilities to edit the two scale factors, `indexscalefactor` and `lookupscalefactor`, the `indexscalefactor` is used to scale the indexing number and the `lookupscalefactor` is used to scale the stored values in the lookup table.

To begin with the LUT technique was implemented in Matlab, pictures with the calibration pattern is extracted from the vision system for camera parameter estimation; see Figure 1, 2, 3 and 4 in Appendix 5. With the help of the calibration toolbox [15], intrinsic and extrinsic parameters are given after the calibration, the resulting values are used in the LUT generation procedure, see Figure 24.

![Flowchart showing the concept of correction table creation.](image)

After the LUT is generated the data is easily accessed according to Figure 25, \( F_{out_{x,y}} \) is the new feature position in \( x \) and \( y \), in this stage the coordinates are represented with fixed-point notation.
### 5 Results

The distance between the pixels moved by the iterative method and pixels moved by our lookup table method is shown in the three histograms in Appendix 7. The test results are derived from a random set of distorted pixels distributed over a 640 by 480 pixel surface according to Figure 1, in Appendix 8, it is easiest to see the movement of pixels far away from the center of the figure because the distortion is larger there, the stars in the figure symbolize the data that is derived from the sensor i.e. distorted pixels, the crosses and dark circles represent the position of the moved data, this is shown in detail in Figure 2, in Appendix 8. Due to the small distance difference between the movement methods used, an even more detailed figure has to be shown, Figure 3 in Appendix 8. The last figure show a high detail view of one moved feature, the ‘dot’ is where the feature is moved to when the LUT is used, the ‘cross’ show where the data is moved according to the Calib toolbox [15] and the ‘circle’ represent the feature moved directly by the iterative method. As can be seen the ‘dot’ doesn’t coincide with the ‘circle’, this has its reasons in several parameters such as truncation to the lookup table and interpolation in the lookup table. Looking at the histograms in Appendix 7 showing the delta distance, it is now easy to see and understand how big the impact of the scaling of the LUT index are. All figures in Appendix 7 and Appendix 8 are based on the assumption that we are able to handle real numbers, which in the FPGA will consume a lot of resources, so the solution was as earlier described to use the fixed-point method. It is possible to measure a difference in precision due to the fixed point method, comparing the second histogram in Appendix 7, Figure 2, with the histograms in Appendix 9 give a good view of the impact that the scaling of the data in the LUT has on the final precision. When the LUT data is scaled by 2048 and 4096 a significant degradation of precision is recognized, it is only when the fixed-point scale is equal or greater than 8192 that the precision approach the real number representation shown in Figure 2, Appendix 7.

### 6 Summary and conclusions

In this thesis a method to compensate camera distortion is developed for an FPGA platform as part of a complete vision system. Several methods and models is presented and described to give a good introduction to the complexity of the problems that is overcome with the developed method. The solution is shown to have a good precision on a sub-pixel level. A script file to extend the Calib Toolbox has been developed for auto generating the LUT as a vhd source file.

### 7 Future work

- What needs to be further investigated is whether the solution increases the precision of the complete system or not.
- Further testing with other camera models could be done especially when fish-eye lenses is used.
Testing with different lenses and different kind of cameras can also be a future task. Some of the improvements that could be looked into are to use an advanced interpolation to fill the zero value positions in the LUT generation procedure; another part that possibly could improve precision is to interpolate the extracted data from the LUT i.e. if the calculated index of a pixel isn’t discrete, then an interpolation of the data in the closest positions could be done.

Also the tangential distortion can be compensated for in some future version.

In [21] Niklas Pettersson and Lars Pettersson research the possibility to automatically extract features with hardware, they conclude that with the SIFT algorithm used it’s possible to extract features for calibration with good results. This technique or a similar method could possibly be used as a base for a totally automated camera calibration procedure that could be performed during the start-up phase of the stereo vision system.
References


Appendix 1

Fig 1: Dual camera board and FPGA mounted on carrier board

Fig 2: Cameras on the board in relation to an object.

Fig 3: Example of what could happen if uncalibrated cameras are used. In this example it is difficult to do proper feature detection.
Appendix 2

Fig 1: An example of the amount of tangential distortion.
Fig 2: An example of the amount of radial distortion.
Fig 3: An example of the amount of combined distortion, radial + tangential.
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optical format</td>
<td>1/2.5-inch (4:3)</td>
</tr>
<tr>
<td>Active imager size</td>
<td>5.70mm(H) x 4.28mm(V)</td>
</tr>
<tr>
<td></td>
<td>7.13mm diagonal</td>
</tr>
<tr>
<td>Active pixels</td>
<td>2,592H x 1,944V</td>
</tr>
<tr>
<td>Pixel size</td>
<td>2.2µm x 2.2µm</td>
</tr>
<tr>
<td>Color filter array</td>
<td>RGB Bayer pattern</td>
</tr>
<tr>
<td>Shutter type</td>
<td>Global reset release (GRR),</td>
</tr>
<tr>
<td></td>
<td>Snapshot only</td>
</tr>
<tr>
<td></td>
<td>Electronic rolling shutter (ERS)</td>
</tr>
<tr>
<td>Maximum data rate/</td>
<td>96 Mp/s at 96MHz (2.8V I/O)</td>
</tr>
<tr>
<td>master clock</td>
<td>48 Mp/s at 48MHz (1.8V I/O)</td>
</tr>
<tr>
<td>Frame rate</td>
<td>Full resolution</td>
</tr>
<tr>
<td></td>
<td>Programmable up to 14 fps</td>
</tr>
<tr>
<td></td>
<td>VGA (640 x 480, with binning)</td>
</tr>
<tr>
<td></td>
<td>Programmable up to 53 fps</td>
</tr>
<tr>
<td>ADC resolution</td>
<td>12-bit, on-chip</td>
</tr>
<tr>
<td>Responsivity</td>
<td>1.4V/lux-sec (550nm)</td>
</tr>
<tr>
<td>Pixel dynamic range</td>
<td>70.1dB</td>
</tr>
<tr>
<td>SNR&lt;sub&gt;max&lt;/sub&gt;</td>
<td>38.1dB</td>
</tr>
<tr>
<td>Supply Voltage</td>
<td>I/O 1.7V-3.1V</td>
</tr>
<tr>
<td></td>
<td>Digital 1.7V-1.9V (1.8V nominal)</td>
</tr>
<tr>
<td></td>
<td>Analog 2.6V-3.1V (2.8V nominal)</td>
</tr>
<tr>
<td>Power consumption</td>
<td>381mW at 14 fps full resolution</td>
</tr>
<tr>
<td>Operating temperature</td>
<td>-30˚C to +70˚C</td>
</tr>
<tr>
<td>Packaging</td>
<td>48-pin iLCC, Die</td>
</tr>
</tbody>
</table>

Key Performance Parameters of the Micron MT9P031 CMOS Digital Image Sensor, original table found in the data sheet.
Appendix 4

Picture 1: Some of the lenses that could be used with the camera system, the lens to the left is of fish-eye type, lens 3 and 5 from the left is concave lenses and the rest is of common convex type.
Appendix 5

Click on the four extreme corners of the rectangular pattern (first corner = origin)... Image 1

Figure 1

The red crosses should be close to the image corners

Figure 2:
Figure 3

Extracted corners

Figure 4

The red crosses should be on the grid corners...
Appendix 6

Picture 1: A picture showing the fish-eye phenomenon.

Picture 2: A picture showing the vignetting phenomenon.
Appendix 7

Figure 1: Histogram of pixels difference according to iterative movement relative to LUT movement, LUT index scaled down with 256, 2000 samples.

Figure 2: Histogram of pixels difference according to iterative movement relative to LUT movement, LUT index scaled down with 512, 2000 samples.

Figure 3: Histogram of pixels difference according to iterative movement relative to LUT movement, LUT index scaled down with 1024, 2000 samples.
Figure 1: Random distributed data over a 640 by 480 pixel surface, LUT index scaled down with 256.
Figure 2: Some pixels from the random distributed set in detail, it is easy to see the movement from the star markers (original position) to the new position marked with dark circles (iterative method) and our LUT method the small dot. LUT index scaled down with 256.

Example of original pixel and the moved position
Figure 3: A detailed view of the corrected data, the distance between the "dot" (LUT) and the "circle" (iterative method) and the "cross" (Calib toolbox). This distance is shown in the histograms in Appendix 7, LUT index scaled down with 256.
Appendix 9

Figure 1: Histogram of pixels difference according to iterative movement relative to LUT movement, LUT index scaled down with 512, 2000 samples, Fixed-Point scale 2048.

Figure 2: Histogram of pixels difference according to iterative movement relative to LUT movement, LUT index scaled down with 512, 2000 samples, Fixed-Point scale 4096.

Figure 3: Histogram of pixels difference according to iterative movement relative to LUT movement, LUT index scaled down with 512, 2000 samples, Fixed-Point scale 8192.