Synchronization of a multi camera system

Examensarbete utfört i Datorseende
vid Tekniska högskolan vid Linköpings universitet
av

Alexander Vibeck

LiTH-ISY-EX--15/0438--SE

Linköping 2015
Synchronization of a multi camera system

Examensarbete utfört i Datorseende 
vid Tekniska högskolan vid Linköpings universitet 
av

Alexander Vibeck

LiTH-ISY-EX--15/0438--SE

Handledare: phd student Mikael Persson
ISY, Linköpings universitet

Examinator: Klas Nordberg
ISY, Linköpings universitet

Linköping, 30 maj 2015
Synkronisering av ett multikamerasystem

Alexander Vibeck

In a synchronized multi camera system it is imperative that the synchronization error between the different cameras is as close to zero as possible and the jitter of the presumed frame rate is as small as possible. It is even more important when these systems are used in an autonomous vehicle trying to sense its surroundings. We would never hand over the control to a autonomous vehicle if we couldn't trust the data it is using for moving around.

The purpose of this thesis was to build a synchronization setup for a multi camera system using state of the art RayTrix digital cameras that will be used in the iQMatic project involving autonomous heavy duty vehicles. The iQMatic project is a collaboration between several Swedish industrial partners and universities. There was also software development for the multi camera system involved. Different synchronization techniques were implemented and then analysed against the system requirements. The two techniques were hardware trigger i.e. external trigger using a microcontroller, and software trigger using the API from the digital cameras.

Experiments were conducted by testing the different trigger modes with the developed multi camera software. The conclusions show that the hardware trigger is preferable in this particular system by showing more stability and better statistics against the system requirements than the software trigger. But the thesis also show that additional experiments are needed for a more accurate analysis.
Abstract

In a synchronized multi camera system it is imperative that the synchronization error between the different cameras is as close to zero as possible and the jitter of the presumed frame rate is as small as possible. It is even more important when these systems are used in an autonomous vehicle trying to sense its surroundings. We would never hand over the control to a autonomous vehicle if we couldn't trust the data it is using for moving around.

The purpose of this thesis was to build a synchronization setup for a multi camera system using state of the art RayTrix digital cameras that will be used in the iQMatic project involving autonomous heavy duty vehicles. The iQMatic project is a collaboration between several Swedish industrial partners and universities. There was also software development for the multi camera system involved. Different synchronization techniques were implemented and then analysed against the system requirements. The two techniques were hardware trigger i.e. external trigger using a microcontroller, and software trigger using the API from the digital cameras.

Experiments were conducted by testing the different trigger modes with the developed multi camera software. The conclusions show that the hardware trigger is preferable in this particular system by showing more stability and better statistics against the system requirements than the software trigger. But the thesis also show that additional experiments are needed for a more accurate analysis.
Acknowledgments

First I like to thank my supervisor Mikael Persson at CVL for acting as the "bad cop" in this project and a special thanks to the "good cop" Tommaso Piccini, CVL for all your input and cooperation in this thesis work. I would like to thank my office room mate Felix for our helpful discussions and also a thanks to the rest of the people at CVL, Linköping University. I would also like to thank my friends and colleagues at my present employer Sensus Studieförbund for letting me take a leave of absence to finish my degree. Finally a big hug to my wife and my two boys for having faith in me.

Linköping, May 2015
Alexander Vibeck
Contents

Notation ix

1 Introduction 1
  1.1 Background ................................................................. 1
  1.2 iQMatic project ............................................................ 1
  1.3 Purpose and scope ......................................................... 2
  1.4 System requirements ..................................................... 3
  1.5 Problem description ...................................................... 4
  1.6 Report outline ............................................................ 4

2 Related work 5
  2.1 Autonomous vehicles ..................................................... 5
  2.2 Synchronized multi camera systems ................................... 6
  2.3 Theory of camera synchronization ..................................... 6
  2.4 Microcontrollers .......................................................... 8

3 System setup 9
  3.1 Hardware ................................................................. 9
    3.1.1 Raytrix C42 digital camera ........................................ 9
    3.1.2 Microcontroller platform Digilent Chipkit UNO32 ............ 10
    3.1.3 NI Elvis II Oscilloscope ......................................... 10
    3.1.4 The lab cart ....................................................... 10
  3.2 Software ................................................................. 11
    3.2.1 Visual Studio 2013 .................................................. 11
    3.2.2 RayCamView 4.1 ..................................................... 11
    3.2.3 MPIDE ............................................................... 11

4 Method 13
  4.1 Microcontroller platform Digilent Chipkit UNO32 ................ 14
    4.1.1 Technical specifications .......................................... 14
    4.1.2 Programming the microcontroller ............................... 15
    4.1.3 External trigger program for the UNO32 ...................... 15
    4.1.4 Connections used on the UNO32 ................................. 16
| 4.1.5 | Verifying the trigger signal using NI ELVIS Oscilloscope | 18 |
| 4.2  | RayCamView 4.1 | 19 |
| 4.3  | Camera software | 19 |
| 4.3.1 | C++ software to connect to the UNO32 | 19 |
| 4.3.2 | Camera trigger modes | 20 |
| 4.3.3 | Single camera test software | 21 |
| 4.3.4 | Multi camera software | 22 |
| 4.4  | Experiments | 24 |
| 4.4.1 | Synchronization error test using flash clock | 24 |
| 4.4.2 | Long duration test | 24 |

| 5 | Results | 25 |
| 5.1 | Synchronization error test using flash clock | 25 |
| 5.1.1 | HardwareFreerun (External trigger) | 26 |
| 5.1.2 | SoftwareContinuous (Software trigger) | 29 |
| 5.1.3 | SoftwareSingle (Software trigger) | 32 |
| 5.2 | Long duration test | 35 |

| 6 | Conclusions and future work | 37 |
| 6.1 | Conclusions | 37 |
| 6.2 | Future work | 39 |
| 6.2.1 | External trigger | 39 |
| 6.2.2 | Experiments | 39 |

Bibliography | 41 |
## Notation

### Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>CMOS</td>
<td>Complementary Metal Oxide Semiconductor</td>
</tr>
<tr>
<td>CVL</td>
<td>Computer Vision Laboratory (at Linköping University)</td>
</tr>
<tr>
<td>FPS</td>
<td>Frames Per Second</td>
</tr>
<tr>
<td>GPS</td>
<td>Global Positioning System</td>
</tr>
<tr>
<td>GPU</td>
<td>Graphic Processing Unit</td>
</tr>
<tr>
<td>IDE</td>
<td>Integrated Development Environment</td>
</tr>
<tr>
<td>LIDAR</td>
<td>LIght Detection And Ranging</td>
</tr>
<tr>
<td>MCU</td>
<td>Micro Controller Unit</td>
</tr>
<tr>
<td>RADAR</td>
<td>RAdio Detection And Ranging</td>
</tr>
<tr>
<td>SDK</td>
<td>Software Development Kit</td>
</tr>
<tr>
<td>SLAM</td>
<td>Simultaneous Localization And Mapping</td>
</tr>
<tr>
<td>USB</td>
<td>Universal Serial Bus</td>
</tr>
</tbody>
</table>
1

Introduction

1.1 Background

In systems that use multiple cameras to produce images of the same scene it is imperative that the cameras are as synchronized as possible where synchronization means that the different cameras acquire the images at the same time, or with smallest delay possible. If you for example have a multi camera setup in an autonomous vehicle that is used for a collision avoidance system and you use the cameras images to triangulate the distance of an oncoming object the data will be corrupt if the synchronization between the cameras are off. Lets say the vehicle moves at 100 km/h and if we for example have a synchronization error of 1 s, the vehicle has moved approximately 28 meters between the images. This can have devastating effects when trying to avoid obstacles. If the error is instead 0.1 ms the vehicle has only moved 2.8 millimeters. Of course you would like the synchronization error to be exactly zero but that may not be possible due to limitations in the software or hardware. You can also compare it to if the human eyes aren’t synchronized you would certainly feel quite dizzy and it would be hard moving around efficiently.

1.2 iQMatic project

In 2013, a large project called iQMatic was started as a collaboration between industrial partners such as Scania, Saab and Autoliv, and with several research groups from the Swedish universities KTH and LiU. The goal is to develop a research platform for autonomous heavy-duty vehicles which will mainly be used in the mining industry. Mining facilities are a good start to introduce autonomous vehicles because they are usually a somewhat controlled environment. Because of the scale of this project it is divided in different subtasks and each partner gets
the responsibility for their subsystem. Examples of subsystems can be mission planning, navigation, obstacle avoidance and various decision making [Mårtenson et al., 2014], [Oliveira, 2014], [Evestedt et al., 2014].

This thesis work at CVL is a part of the design and implementation of the 360 vision component. The component consists of a set of cameras placed around the vehicle in such a way to produce a complete visual coverage of the surrounding environment. The 360 component will provide the system with an estimate of the vehicle’s egomotion [Olson et al., 2003] i.e. the vehicle’s position calculated by using the cameras images compared to fixed objects in the surrounding environment, by using state of the art SLAM algorithms. SLAM stands for Simultaneous Localization And Mapping [Dissanayake et al., 2001] and is used to calculate and map the vehicle’s position in an unknown environment compared to a car’s GPS system where there are pre-loaded maps used together with a GPS signal to calculate both speed and location of the vehicle. The 360 component can also be used to provide information about surrounding obstacles thus giving an estimate of the drivable area around the vehicle which is of utmost importance for the motion planning algorithms. Fig. 1.1 shows the two cameras mounted in the Scania truck during a test session in this thesis work.

![Figure 1.1: interior of Scania truck](image)

### 1.3 Purpose and scope

The purpose of this thesis work is to implement the synchronization part of a multi-camera system that will be used in the iQMatic project. The synchronization can be done by using different techniques to trigger the cameras i.e. send a signal to the cameras to take a picture. The two techniques that are implemented in this system are hardware trigger where the cameras are externally triggered by using the hardware in an external microcontroller to produce the trigger signal and software trigger where the cameras are triggered internally by the multi-camera software.

The different trigger modes are then invoked in the developed software for the multi-camera system. There are a number of people involved in this project.
where PhD student at CVL (Computer Vision Laboratory), Tommaso Piccini is the main programmer of the camera software and responsible of the computer vision part of the finalized multi camera system. The scope of this thesis is the synchronization between the cameras but I will also discuss and show the main features of the multi camera software.

### 1.4 System requirements

The system requirements comes from the main developer of the multi camera system, Tommaso Piccini at CVL.

| Req. 1 | The final multi camera system must provide the following functionalities: The camera stream must be initialized and stopped by the computer software (i.e. the camera software must be able to control both the hardware and software trigger) (req. 1.1). The 2 cameras must be triggered at the same time. A maximum error of 0.1 ms is acceptable (req. 1.2). The framerate, or the number of frames recorded every second must be constant and jitter must be as small as possible given the platform. Where jitter can be described as a variation or drift in the actual framerate compared to the estimated or expected framerate. Specifically: the average framerate must be constant at 30.3 FPS (req. 1.3) for at least 30 minutes (trigger interval of 33 ms). In the same operation time, the standard deviation in the measured inter-frame interval cannot exceed 1 ms (req. 1.4). |
| Req. 2 | The software must receive the incoming streams and measure the framerate. |
| Req. 3 | An analysis of the system performance is expected. This includes an analysis of the error in synchronization between the two cameras and a measurement of the drift/jitter in the framerate, also compare the performance of the system when using the different trigger techniques. |
| Req. 4 | A sample C++ program that starts, receives and stops the stream must be provided. Documentation must be provided for both the C++ program and microcontroller code. The documentation must be complete and allow a programmer with no previous knowledge of the system to operate it. |

**Optional:**

| Req. 5 | If the performances granted by the provided microcontroller (chipkit UNO32) are unsatisfactory, switch to a better microcontroller. |
| Req. 6 | The software on the computer side should also be capable of saving the uncompressed data stream from the cameras on disk, in real time and for extended periods of time. |
1.5 Problem description

After the implementation of the two synchronization techniques in the multi camera system an analysis is made against the system requirements and the following questions are answered:

- 1. Which of the two chosen synchronization techniques is preferred when compared to the system requirements?

- 2. Which of the two chosen synchronization techniques is preferred in the particular environment (in the Scania truck) of the multi camera system?

- 3. If none of the chosen techniques meet the demands, is there a more suitable way of synchronizing the cameras?

1.6 Report outline

- **Chapter 1 - Introduction.**
  Here are the background, purpose and scope, problem description of this thesis and also the system requirements.

- **Chapter 2 - Related Work.**
  This chapter describes some related work to this thesis and also a simplified theory behind camera synchronization.

- **Chapter 3 - System setup.**
  Contains a short description of the different hardware and software that are used in this thesis work.

- **Chapter 4 - Method.**
  Here are the information about the method used in this thesis work, such as the system design and experiments. The first section handles the microcontroller that are used to produce the external trigger signal. The Camera software section describes the single camera test software and the developed multi camera software. At the end of this chapter there are also descriptions of the experiments that were conducted.

- **Chapter 5 - Results.**
  This chapter contains the results of the experiments.

- **Chapter 7 - Conclusions and future work.**
  Here are the conclusions of this thesis work and also some proposals for future work.
Related work

2.1 Autonomous vehicles

Today we are searching for more and more ways to let our technology be more autonomous and thus leaving some decision making to the machines, for example in autonomous vehicles. The first truly autonomous car dates back to the 80s with the Navlab[Thorpe et al., 1988] vehicles from the Carnegie Mellon University in the U.S. Today there are a range of autonomous (partially or totally) vehicles for land, sea and air. One of the more familiar military implementation is of course the U.S. Military drones that can fly operations all over the world with only some supervision from the control room back in the states. One of

![Figure 2.1: A vehicle being developed for the 2007 Darpa Urban Challenge. (wikimedia commons)](image)

the more famous civilian project is The Darpa Grand Challenge (fig. 2.1) that invite teams from the entire world to compete in making autonomous cars that race on a pre-destined course. Different car makers are also implementing more and more autonomous features in their cars such as collision avoiding and active
parking assistance. There are different technologies that can be used in letting
the autonomous vehicle explore its surroundings. Some of them are RADAR, LI-
DAR, GPS and cameras in combination with computer vision. The latter are used
in our project.

2.2 Synchronized multi camera systems

As described in [Litos et al., 2006] there are several ways of synchronizing a multi
camera system. These are (a) hardware i.e. external trigger, (b) post process-
ing synchronization using computer vision, (c) network synchronization, (d) soft-
ware trigger where (a) is the most commonly used. But sometimes it is not pos-
sible to use a hardware trigger for example in wireless camera systems or when
there is a long distance between the cameras. In those cases it is more feasible to
use the (b) or (c) approach. In [Litos et al., 2006] they use a distributed PC system
with network synchronization to achieve this. You can also use audio and visual
data together to synchronize a multi camera system [Michel and Stanford, 2006].

Another factor when choosing synchronization method, can be cost efficiency
where cameras with external trigger possibilities are more expensive and there
are also the need of an external micro controller producing the trigger signal. In
[Svoboda et al., 2002] a description is presented of a low-cost software synchro-
nized multi camera system. In the iQMatic project we will focus on synchronizing
using hardware or software trigger because the system is in a relative confined
space inside of a truck.

2.3 Theory of camera synchronization

This section presents the theory of camera synchronization between two cameras.
If the cameras are not synchronized and the cameras or the objects captured are
not fixed, there will be an unwanted offset when combining the two cameras
images to produce a single view of the environment. This offset, or error corre-
lates to the pixels in the acquired digital images. This pixel error is calculated
depending on the synchronization error in seconds and the movement of the ob-
ject/camera. Let’s say we have two cameras cam0 and cam1 where cam0 is the
"master" camera i.e. takes the picture at the correct time and cam1 takes the pic-
ture with a delay thus not synchronized. The cameras are taking a picture of a
moving point \( P_0 \) (see fig. 2.2).

We can simplify this by having one camera take a picture of a point \( P_0 \) but
with the synchronization error \( \Delta t \) we get a pixel error \( e \). Instead of having a
moving point or object, we can let the camera move with the speed \( v_c \). Hence \( e \)
is proportional to \( \Delta t, v_c \). The 3D point \( P_0 \) is projected on the 2D image plane as a
point \( Y_0 \) (see fig. 2.3).

By having a synchronization error, the "original" point \( P_0 \) that we thought we
recorded will actually be the point \( P_1 \). Thus

\[
P_1 = P_0 + v_c \cdot \Delta t
\]
2.3 Theory of camera synchronization

Figure 2.2: Two cameras looking at the same point

Figure 2.3: Camera with synchronization error

The two 3D-points in the world plane can be written as: \( P_0 = \begin{pmatrix} x_0 \\ y_0 \\ z_0 \end{pmatrix} \) and \( P_1 = \begin{pmatrix} x_1 \\ y_1 \\ z_1 \end{pmatrix} \).

These two points are projected on the image plane as

\[
Y_0 = \left( \frac{k_1 x_0}{k_2 y_0} \right) \frac{1}{z_0} + \left( \frac{P_x}{P_y} \right) \text{ and } Y_1 = \left( \frac{k_1 x_1}{k_2 y_1} \right) \frac{1}{z_1} + \left( \frac{P_x}{P_y} \right)
\]

where \( k_1 \) and \( k_2 \) are part of the cameras calibration matrix \( K \). We can also call them the *intrinsic parameters* and they differ from camera to camera.

The point \( \left( \frac{P_x}{P_y} \right) \) is the *principal point* which is at the intersection of the optical axis and the image plane. The pixel error \( e \) is then calculated as the difference between \( Y_0 \) and \( Y_1 \).

\[
e = |Y_0 - Y_1| = \left| \left( \frac{k_1 x_0}{k_2 y_0} \right) \frac{1}{z_0} + \left( \frac{P_x}{P_y} \right) - \left( \frac{k_1 x_1}{k_2 y_1} \right) \frac{1}{z_1} + \left( \frac{P_x}{P_y} \right) \right| \tag{2.2}
\]

Simplify.

\[
e = \left| \left( \frac{k_1 x_0}{k_2 y_0} \right) \frac{1}{z_0} - \left( \frac{k_1 x_1}{k_2 y_1} \right) \frac{1}{z_1} \right| \tag{2.3}
\]
Insert the equation (2.1):

\[
e = \left| \left( \begin{array}{c} k_1 x_0 \\
\frac{1}{z_0} 
\end{array} \right) - \left( \begin{array}{c}
\frac{k_1 (x_0 + v_{cx_0} \Delta t)}{z_0 + v_{cz_0} \Delta t} \\
\frac{1}{z_0 + v_{cz_0} \Delta t}
\end{array} \right) \right| \tag{2.4}
\]

Assume that \( k_1 = k_2 = 1000 \) and the camera only moves in the x-axis \( v_c = \begin{pmatrix} v \\ 0 \end{pmatrix} \)

\[
e = \left| \left( \begin{array}{c} 1000 x_0 \\
0
\end{array} \right) - \left( \begin{array}{c}
1000 (x_0 + v \Delta t) \\
0
\end{array} \right) \right| \tag{2.5}
\]

Simplify

\[
e = \left| \left( \begin{array}{c} 0 \\
\frac{1}{z_0}
\end{array} \right) - \left( \begin{array}{c}
1000 v \Delta t \\
0 \end{array} \right) \right| \tag{2.6}
\]

Gives

\[
e = \frac{1000 \cdot v \cdot \Delta t}{z_0} \tag{2.7}
\]

Assume that the camera moves at 1 m/s and the distance on the z-axis between the camera and the point/object is 1 m. If we can tolerate a pixel error of 0.1 how large can the synchronization error be? We solve \( \Delta t \) from:

\[
0.1 = \frac{1000 \cdot 1 \cdot \Delta t}{1} \tag{2.8}
\]

Which gives a maximum synchronization error \( \Delta t \) of 0.1 ms. (like the system req. 1.2.) This is the maximum error since any motion not parallel to the image plane reduces the error.

### 2.4 Microcontrollers

To produce the external trigger signal for the cameras we need a microcontroller. A microcontroller or MCU is often described as a small computer on a single chip. This chip, or integrated circuit, often contains a processor, memory and programmable inputs and outputs. MCUs are mainly used to automatically control different products and devices such as cars, appliances and other electrical systems. MCUs has been commercially available since the mid 1970s. There are a lot of different manufacturers of MCUs. Some of them are Texas Instruments, Atmel, Microchip Technology and Motorola. In the beginning it was more common to use low level assembler language to program the MCU but now it’s more common to use a high-level programming language like C or others. The program memory can either be Read-Only where the program for the MCU are programmed once at the factory and the function of the MCU is static, or you can have a more versatile approach which lets the user write and upload new programs for every new implementation of the MCU.
This chapter describes the different hardware and software that are used in this thesis work.

3.1 Hardware

3.1.1 Raytrix C42 digital camera

The Raytrix C42 digital camera is a state of the art ultra high resolution camera with a CMOS image sensor. The total sensor size is 7728 x 5368 pixels corresponding to 41.5 Mega Pixels. It is capable of producing images at 7 FPS at full resolution (7708 x 5352 pixels), 30 FPS video at 4k UHD resolution (3856 x 2168 pixels) and 60 FPS video at 2k Full-HD. It uses a USB 3.0 super speed interface and has an external trigger input. It also has a C-mount for adapting any industrial or microscopic optics. In this project, a fish-eye lens from Fuji is used (see fig. 3.1). The specifications for the external trigger signal is a 3.3 V pulse of at least 1 ms duration.

Figure 3.1: The RayTrix C42 camera with a Fuji fish-eye lens.
3.1.2 Microcontroller platform Digilent Chipkit UNO32

A microcontroller is needed to produce an external trigger signal for the cameras. For this project we have chosen the Digilent Chipkit UNO32 Board (see fig. 3.2).

![The Digilent Chipkit UNO32 micro controller](image)

The chipKIT Uno32 is based on the Arduino™ open-source hardware prototyping platform. The Uno32 is the same form factor as the Arduino Uno board and is compatible with Arduino shields. It features a USB serial port interface for connection to the IDE and can be powered via USB or an external power supply. The Uno32 board takes advantage of the powerful PIC32MX320F128 microcontroller, which features a 32-bit MIPS processor core running at 80 MHz, 128K of flash program memory, and 16K of SRAM data memory.

The chipKIT UNO32 is more suitable for our project compared to its counterpart the Arduino UNO. The digital outputs on the Arduino operates at 5 V whereas the chipKIT operates at 3.3 V. The Arduinos processor only runs at 16 MHz so the chipKIT is much faster. I will describe more how to program and use the micro controller in the system design chapter.

3.1.3 NI Elvis II Oscilloscope

The NI Elvis II Oscilloscope stands for National Instruments Educational Laboratory Virtual Instrumentation Suite. It is a design and prototyping platform that integrates some of the most commonly used instruments such as oscilloscope, digital multimeter, function generator, and more as a Windows application. It is used together with the LabVIEW software to show the measurements on the PC screen.

3.1.4 The lab cart

The lab cart in fig. 3.3 is used to try to emulate the cameras position and to test the system before installing the cameras inside of the Scania truck. The cart contains a PC with the software, a 12 V car battery, a wireless keyboard, a USB 3.0 screen, the chipKIT UNO32 micro controller for external triggering and two
3.2 Software

3.2.1 Visual Studio 2013

Microsoft Visual Studio is an integrated development environment (IDE) from Microsoft. It is used to develop computer programs for Microsoft Windows, as well as web sites, web applications and web services. It is compatible with programming languages such as C, Visual C++, Visual C#, Visual Basic, F# and more. For this project we have chosen C++ (via Visual C++) and this version of the software will be operated from the prompt window (instead of a windows application).

3.2.2 RayCamView 4.1

This is the Windows based bundled software that comes from the manufacturer of the Raytrix cameras. It works only for a single camera at a time. This software is used in parallel with our own to see if the cameras are functioning properly and also to test the external trigger. When developing our own software, this program makes it easy to check if there is a problem with the cameras or if there are some errors in our own code which make the cameras not work properly. In the beginning of the development there were some problems with the USB 3.0 hub in the PC and the RayCamView software made it easier to see if the error was in the USB 3.0 hub, in the cameras, or in our own software.

3.2.3 MPIDE

The MPIDE is a Multi platform Arduino compatible IDE that is a modified version of the Arduino IDE. This software works on different microcontroller platforms such as the Chipkit UNO32 and on Arduino boards. It is used to program

mounted RayTrix C42 cameras.

Figure 3.3: The lab cart
the microcontroller in what is called a "sketch" in the Arduino language. The pro-
gram is written in the MPIDE software and then compiled and uploaded to the
microcontroller where it is executed. The Arduino language is based on C/C++
but has its own library [Arduino, 2015] which is used and explained in the sys-
tem design chapter. Both the hardware and the software are open-source which
means that there are a lot of information and different forums online where you

can gather knowledge about how to use and program your microcontroller.
This chapter is about the method used in this thesis work. It involves system design and experiments. Section 4.1 handles the programming of the MCU and implementation of the external trigger i.e. hardware trigger. Section 4.2 is about the usage of the bundled camera software RayCamView. Section 4.3 describes the different camera software that are developed. We have a single camera test software for testing the trigger and functionality of the camera and then the multi camera software that is a part of the 360 vision component in the iQMatic project. The multi camera software is tested in section 4.4 Experiments.

Figure 4.1 shows a block diagram over the multi camera system. The developed multi camera software is executed from the PC and control the cameras and the external trigger. The program for the external trigger is uploaded on the MCU beforehand but waits for a start signal from the multi camera software.

Figure 4.1: Block diagram of the multi camera system
4.1 Microcontroller platform Digilent Chipkit UNO32

In this section are the essential technical specifications of the MCU. There is also a description of the external trigger program for the MCU and a verification of the trigger signal using an oscilloscope.

4.1.1 Technical specifications

The key features of the UNO32 [DigilentChipkit, 2015] that are used in this implementation are:

- **43 available I/O pins (analog or digital).**
  The trigger signal to the cameras need one digital output (or more) to produce the pulse (3.3 V for at least 1 ms). The cameras have separate trigger cables but it is optimal to use the same digital output on the MCU for both cameras so they trigger exactly at the same time. When connecting multiple cables to an output we have to make sure that we do not exceed the I/O pin current restrictions but because it is a digital signal this will not be an issue.

- **2 user LED:s**
  One smart feature is that Pin 13 also connects to one of these LED:s named User LED LD4. This means that if the Pin 13 is used as the digital output for our trigger it is possible to see (when the LED flashes) if the trigger is active, without connecting the output to a voltage meter or oscilloscope.

- **3.3 V operating voltage.**
  This means that if the digital output is set as "high" you get 3.3 V and that matches the specifications of the trigger signal. If we would have used the Arduino UNO board instead (which operates at 5 V) we would have had to use a voltage divider on the output.

- **PC connection uses a USB A > mini B cable.**
  The USB connection is used for both serial communications and power supply to the board. One drawback is that when you power up your UNO 32 and open the serial communications to the PC it automatically waits for a new program to be uploaded to the microcontroller (even if you already programmed it). This will take a couple of seconds so a delay is introduced in our camera software to cope with this. This feature on the UNO32 is called "automatic reset" and can be disabled by cutting a trace at the bottom of the board but for our implementation this is not necessary.

- **The single I/O pin current must be restricted to +7/- 12mA. (max total +/-200 mA.)**
4.1 Microcontroller platform Digilent Chipkit UNO32

4.1.2 Programming the microcontroller

As mentioned in section 3.2.3 the microcontroller is programmed in the Arduino language using the MPIDE software. After the code is written and compiled the program is uploaded to the MCU where it performs whatever it was designed to do. The language is C/C++ based but has its own library with special functions like `digitalWrite()`, `pinMode()`, `delay()` which are used in our code. The Arduino library can be found at [Arduino, 2015]. The code is called a "sketch" and it contains (in simplest form) from top down: declarations of variables/constants, a `setup()` function and a `loop()` function. The first two will only run once but the `loop()` function is always active (compare to `while(true)` in C++) and here is where you put the main code. Here is an example of a sketch for the microcontroller and it starts running as soon as the program is uploaded to the microcontroller’s memory.

```c
/*
 * Blink
 * Turns on an LED on for one second, then off for one second, repeatedly.
 */

int PIN = 13; //declare a pin number

void setup() {
 // initialize the digital pin as an output.
 // Pin 13 has an LED connected on most Arduino and compatible boards:
 pinMode(PIN, OUTPUT);
}

void loop() {
 digitalWrite(PIN, HIGH); // set the LED on
 delay(1000); // wait for a second
 digitalWrite(PIN, LOW); // set the LED off
 delay(1000); // wait for a second
}
```

4.1.3 External trigger program for the UNO32

In this projects requirement 1.1 in section 1.4 it states that the camera software must be able to control both the software and hardware trigger so there has to be some way for the computer to communicate with the UNO32 otherwise the external trigger signal will always be active as soon as the trigger program is uploaded to the MCU. The functions `Serial.begin()`, `Serial.available()` and `Serial.read()` are used to achieve this. The `Serial.begin()` function is used to open the serial connection to the PC and takes the Baud-rate (connection speed) as a parameter. It is important to use the same Baud-rate in both the sketch and in the camera software, otherwise the data can be corrupted. The `Serial.available()` function is used to check if there is anything available to read from the serial port. The `Serial.read()` function is used to read the data from the serial port. Before the camera software is finished, it is possible to use the `Tools>Serial Monitor` in the MPIDE to test sending/receiving data between the PC and the microcontroller. The trigger program use the character ’1’ as a start for the trigger pulse and ’0’ as stop. The requirement 1.3 in section 1.4
states that the trigger interval should be 33 ms and the cameras trigger specifications states that the pulse duration should be 1 ms minimum. Here is the code for the external trigger sketch which is uploaded to the MCU and then waits for a start signal from the camera software:

```c
/*
 * External trigger
 * -----------------
 * Turns a trigger signal On/Off depending on the information
 * on the serial port (given by the software)
 * Created 09 Feb 2015
 * Alexander Vibeck
 */

int Trigger = 13; // select the pin for the Trigger (Digital Out)
char c; // variable to store the data from the serial port

void setup() {
  pinMode(Trigger, OUTPUT); // declare the Trigger pin as output
  Serial.begin(9600); // connect to the serial port
}

void loop () {
  c = Serial.read(); // read the serial port

  if (c == '1') {
    while (Serial.available() == 0){ // trigger as long as there is no data on the serial port
      digitalWrite(Trigger, HIGH); // pulse duration in ms ( >= 1 ms)
      delay(31);
      digitalWrite(Trigger, LOW);
    }
  }
  else {
    digitalWrite(Trigger, LOW);
  }
}
```

The accuracy of the `delay()` function is not perfect because every line in the code takes some microseconds to execute but it should be enough for this implementation. Its more important that the cameras get the trigger signal at the same time. Another drawback with the `delay()` function is that it suspends all of the processing in the UNO32. But because this is a simple program and there are no parallel processing e.g. other inputs/outputs that generate data, this wont be a problem. Another way of implementing a delay of the pulse without "locking" the processor is to use the `millis()` function to calculate the elapsed time between the pulses.

### 4.1.4 Connections used on the UNO32

Figure 4.2 shows an image of the connectors that we use on the UNO32 board. The colours correspond to the cable colours of the trigger cables from the camera manufacturer.
Figure 4.2: Connectors used on the UNO32

USB (power and serial communication)

G: gnd (black)

13: trig (white)

cable shield

LD4: LED connected to output 13
4.1.5 Verifying the trigger signal using NI ELVIS Oscilloscope

To verify the trigger signal before connecting to the cameras, the NI ELVIS Oscilloscope is used together with the LabVIEW software. The output pulse $V_{pp}$ should be 3.3 V and the frequency 30.3 Hz. As the figure 4.3 shows, both measurements meet the specifications.

![Oscilloscope Measurements](image)

*Figure 4.3: Oscilloscope measurements on trigger signal*
4.2 RayCamView 4.1

As mentioned in section 3.2.2 the RayCamView software is used to test the cameras functionality and the external trigger in parallel when developing our own camera software. Figure 4.4 shows the testing of the external trigger option in the RayCamView together with the external trigger program in the MCU and a single camera. The output frame rate is 30.3 (bottom left in figure 4.4) which corresponds to the pulse interval of 33 ms that is derived from the `delay()`-functions in the microcontroller sketch.

![RayCamView together with the external trigger](image)

Figure 4.4: RayCamView together with the external trigger

4.3 Camera software

This section handles the development of the multi camera software. The different parts are: test software for the computer to communicate with the UNO32, single camera test software to test the functionality (e.g. frame rate and jitter) of the camera together with the different trigger modes and when the single camera software works satisfactory we continue to develop the multi camera software. All of the above is C++ software that is executed from the command prompt in Windows.

4.3.1 C++ software to connect to the UNO32

As mentioned in section 4.1.3 the PC needs to connect to the UNO32 and then send a start and stop signal for the external trigger. The first test software for the serial communication was written in Visual C++ (see fig. 4.5) and in managed C++ code which is a special type of C++ language made for Microsoft’s .NET framework. This test software was developed before we decided that the multi
camera software should be written in native C++ ("standard" C++ usable on any operating system) so before the implementation of the external trigger control the serial communication code was rewritten from managed to native C++.

4.3.2 Camera trigger modes

To implement the different synchronization techniques (hardware or software trigger) we need to be able to set the cameras trigger mode. In the cameras API there are three different trigger modes that can be use in our camera software:

For the hardware synchronization we use the **HardwareFreerun** mode. This is used when you have an external trigger like the UNO32 connected to the camera and the frame rate is set by the microcontroller. For the software synchronization we have two options:

**SoftwareContinuous** is used when you want the software to set the frame rate internally by using the `setProperty(Rx::RayCam::ECamProperty::Frame rate, FRAMERATE)` function. But the drawback is that we can not exactly know when the camera starts taking the pictures. It starts taking the pictures when the cameras are initialized and ready but this time can differ between multiple cameras.

Figure 4.5: Visual C++ trigger control with RayCamView
**SoftwareSingle** is used to take a single frame by using the `Rx::RayCam::IDevice::Trigger()` function. But by having the `Trigger()` function in a different thread and inside of a delayed loop (like the external trigger sketch in section 4.1.3), this may be used as a software trigger. When used with multiple cameras we should have more control when the cameras start taking pictures. In SoftwareSingle mode the camera has an internal delay of approximately 10 ms between it is triggered and when it takes the picture. So if you want a frame interval of 33 ms the loop delay should be (33-10-exposure time).

### 4.3.3 Single camera test software

The test software for the single camera was mainly written by Tommaso but I was responsible for the triggering and measurements so it was necessary to understand and be able to do changes in all of the code to get the different tasks working together properly. If the software doesn't work properly it is also impossible to do measurements regarding the frame rate etc. Figure 4.6 shows a flowchart of the main functions in the test software:

![Flowchart of single camera test software](image)

**Figure 4.6: Simplified flowchart of single cam software**

The callback function which is used to acquire images from the camera is shown in a separate box than the main program. This is because it runs in a different thread than the main program i.e. in parallel with the main program. The full source code is shown in the separate document [Vibeck, 2015].
Timing and measurements

To check that the system meets the system requirements the time between the frames needs to be measured. To do so a timer is introduced inside the code to measure the time between the callback functions and when the program is running, printing the time of each frame interval to the screen. This is not exactly accurate because there is a small delay between when the actual image is taken and when the software calls the callback function. But the measured time between the callbacks still gives you a good view of the system performance.

4.3.4 Multi camera software

When running the single camera test software it was hard to get a constant frame rate because of limitations with the laptop's graphical processing speed and because there was image processing like color adjustments and resizing of the images inside the callback function which made the software work slowly. For the multi camera software to work we needed to switch to a more capable PC with a better GPU and more storage for the large amount of images that are produced and keep the callback function as "fast" as possible (i.e. no image processing or printing to screen inside of the callback function) and instead use multi threads for the image processing and storage. Fig. 4.7 shows a simplified flowchart of the multi camera software.

Timing and measurements of the cameras callback functions

For the multi camera software a "local" timer in each callback is used to calculate the interval and saving the time in a vector which is then used to calculate the average frame rate, standard deviation and MAX/MIN (requirements 1.3, 1.4 in section 1.4). To calculate the standard deviation of the inter frame timing i.e. the trigger interval we use the equation:

\[ \sigma = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (x_i - \mu)^2} \]  

(4.1)

, where \( \sigma \) is the standard deviation of the frame interval, \( N \) is the number of values, \( x_i \) is each frame interval value and \( \mu \) is the average of all frame intervals.

To calculate the max triggering error (requirement 1.2, section 1.4), i.e. synchronization error, a master timer is used and the time stamps of each cameras callback are saved to separate vectors. The time stamps are then compared to get the synchronization error. As mentioned before the callback measurements is not the same as the actual time when the images are taken but gives you a hint of the system’s performance. You can say that its the "worst case" scenario. The source code for the calculations can be seen in the separate document [Vibeck, 2015].
Header files:
RayCamAPI.h, RxImageTypes.h, RaytrixCamHandler.h, RaytrixMultiCamHandler.h
ExtTrigger.h, Timer.h, Measure.h.
The first two are from the cameras SDK and contains information how to interface with the cameras such as drivers, cam properties, trigger mode etc. ExtTrigger.h contains the Serial class for connecting to the UNO32. Timer.h declares the timer. Measure.h declares the functions to calculate standard deviation and avg framerate.

Figure 4.7: Simplified flowchart of multi cam software
4.4 Experiments

4.4.1 Synchronization error test using flash clock

To get a more accurate measurement of the synchronization error between the cameras and between the different trigger modes, the two cameras were used together with the multi camera software and a series of images were taken on a PC screen showing a flash animated clock (see fig. 4.8) with millisecond resolution but because the refresh rate of the computer screen is 60 Hz it cannot display images that changes faster than 16.7 ms (1/60). This means that we can’t measure differences in the synchronization error below that number. Three test runs were made on each trigger mode and the first 10 images were saved together with the callback statistics of each run. The exposure time was set to 1 ms so the images wouldn’t be blurry.

4.4.2 Long duration test

From the system requirements we also see that we have to measure the system’s performance for a duration of 30 minutes. This test only used the callback measurements because of the staggering amount of images being taken. One 30 minute test run on each trigger mode was made.
This chapter presents the results of the experiments of the multi-camera software that are described in sections 4.4.1 and 4.4.2.

5.1 Synchronization error test using flash clock

Three test runs of each trigger mode were made. The results are presented with each trigger mode divided in three test runs. The first figure in test run 1 of each trigger mode shows the images taken on the flash clock on the PC screen and the time stamps are printed out next to the images for better viewing. The "diff" i.e. synchronization error and frame interval derived from these time stamps are also printed out. On the right side of the same figure are the correlating time stamps derived from the measurements of the callback functions. The images are not printed out in the results of test runs 2 and 3 of each trigger mode but they contain the same numerical information.

The second figure in each trigger mode and test run is a screen shot of the text file of the callback measurements that the software saves when exiting the program. Here you can read the following information: which trigger mode is selected, total run time, number of callbacks i.e. frames that are captured on each camera, the average frame interval, the standard deviation of the frame interval, the average frame rate (FPS), the MAX/MIN frame interval and the maximum synchronization error derived from comparing the callback time stamps using the software master timer.
5.1.1 HardwareFreerun (External trigger)

Test run 1

Figure 5.1 shows the results from test run 1 using the HardwareFreerun mode (external trigger). Figure 5.2 shows the callback measurements from the same test run.

![Figure 5.1: Image statistics HF 1](image)
5.1 Synchronization error test using flash clock

Trigger mode: External trigger
Total time in millsec: 1111.6

<table>
<thead>
<tr>
<th>Number of Callbacks Cam0: 51</th>
<th>Number of Callbacks Cam1: 51</th>
</tr>
</thead>
<tbody>
<tr>
<td>AVG Cam0: 33.0179 ms</td>
<td>AVG Cam1: 32.9853 ms</td>
</tr>
<tr>
<td>Std_DEv Cam0: 0.735225 ms</td>
<td>Std_DEV Cam1: 0.6124 ms</td>
</tr>
<tr>
<td>Average fps Cam0: 30.2866</td>
<td>Average fps Cam1: 30.3165</td>
</tr>
<tr>
<td>MAX Cam0: 34.1077 ms</td>
<td>MAX Cam1: 34.0543 ms</td>
</tr>
<tr>
<td>MIN Cam0: 31.914 ms</td>
<td>MIN Cam1: 32.122 ms</td>
</tr>
</tbody>
</table>

Max synch error is: 1.11669 millsec.

**Figure 5.2:** Callback statistics HF 1

**Test run 2**

Figure 5.3 shows the results from test run 2 using the HardwareFreerun mode (external trigger). Figure 5.4 shows the callback measurements from the same test run.

<table>
<thead>
<tr>
<th>HF run 2</th>
<th>image timestamps</th>
<th>callback statistics</th>
</tr>
</thead>
<tbody>
<tr>
<td>cam 0 (s)</td>
<td>cam 1 (s)</td>
<td>diff (ms)</td>
</tr>
<tr>
<td>12.882</td>
<td>12.882</td>
<td>0</td>
</tr>
<tr>
<td>12.915</td>
<td>12.915</td>
<td>0</td>
</tr>
<tr>
<td>12.948</td>
<td>12.948</td>
<td>0</td>
</tr>
<tr>
<td>12.997</td>
<td>12.997</td>
<td>0</td>
</tr>
<tr>
<td>13.031</td>
<td>13.031</td>
<td>0</td>
</tr>
<tr>
<td>13.031</td>
<td>13.031</td>
<td>0</td>
</tr>
<tr>
<td>13.081</td>
<td>13.081</td>
<td>0</td>
</tr>
<tr>
<td>13.115</td>
<td>13.115</td>
<td>0</td>
</tr>
<tr>
<td>13.147</td>
<td>13.147</td>
<td>0</td>
</tr>
<tr>
<td>13.198</td>
<td>13.198</td>
<td>0</td>
</tr>
</tbody>
</table>

**Figure 5.3:** Image statistics HF 2

<table>
<thead>
<tr>
<th>HF run 2</th>
<th>image timestamps</th>
<th>callback statistics</th>
</tr>
</thead>
<tbody>
<tr>
<td>cam 0 (s)</td>
<td>cam 1 (s)</td>
<td>diff (ms)</td>
</tr>
<tr>
<td>12.997</td>
<td>12.997</td>
<td>0</td>
</tr>
<tr>
<td>13.031</td>
<td>13.031</td>
<td>0</td>
</tr>
<tr>
<td>13.081</td>
<td>13.081</td>
<td>0</td>
</tr>
<tr>
<td>13.115</td>
<td>13.115</td>
<td>0</td>
</tr>
<tr>
<td>13.147</td>
<td>13.147</td>
<td>0</td>
</tr>
<tr>
<td>13.198</td>
<td>13.198</td>
<td>0</td>
</tr>
</tbody>
</table>

**Figure 5.4:** Callback statistics HF 2
Test run 3

Figure 5.5 shows the results from test run 3 using the HardwareFreerun mode (external trigger). Figure 5.6 shows the callback measurements from the same test run.

<table>
<thead>
<tr>
<th>HF run 3</th>
<th>image timestamps</th>
<th>callback statistics</th>
</tr>
</thead>
<tbody>
<tr>
<td>cam 0 (s)</td>
<td>cam 1 (s)</td>
<td>diff (ms)</td>
</tr>
<tr>
<td>13.831</td>
<td>13.831</td>
<td>0</td>
</tr>
<tr>
<td>13.831</td>
<td>13.831</td>
<td>0</td>
</tr>
<tr>
<td>13.915</td>
<td>13.915</td>
<td>0</td>
</tr>
<tr>
<td>13.948</td>
<td>13.948</td>
<td>0</td>
</tr>
<tr>
<td>13.948</td>
<td>13.948</td>
<td>0</td>
</tr>
<tr>
<td>13.996</td>
<td>13.996</td>
<td>0</td>
</tr>
<tr>
<td>14.031</td>
<td>14.031</td>
<td>0</td>
</tr>
<tr>
<td>14.031</td>
<td>14.031</td>
<td>0</td>
</tr>
<tr>
<td>14.103</td>
<td>14.081</td>
<td>0</td>
</tr>
<tr>
<td>14.114</td>
<td>14.114</td>
<td>0</td>
</tr>
</tbody>
</table>

**Figure 5.5: Image statistics HF 3**

**Figure 5.6: Callback statistics HF 3**
5.1 Synchronization error test using flash clock

5.1.2 SoftwareContinuous (Software trigger)

Test run 1

Figure 5.7 shows the results from test run 1 using the SoftwareContinuous mode (software trigger). Figure 5.8 shows the callback measurements from the same test run.
Figure 5.8: Callback statistics SC 1

Test run 2

Figure 5.9 shows the results from test run 2 using the SoftwareContinuous mode (software trigger). Figure 5.10 shows the callback measurements from the same test run.

<table>
<thead>
<tr>
<th>SC run 2</th>
<th>image timestamps</th>
<th>callback statistics</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>cam 0 (s)</td>
<td>cam 1 (s)</td>
</tr>
<tr>
<td>05.948</td>
<td>05.948</td>
<td>0</td>
</tr>
<tr>
<td>05.998</td>
<td>05.998</td>
<td>0</td>
</tr>
<tr>
<td>06.031</td>
<td>06.031</td>
<td>0</td>
</tr>
<tr>
<td>06.081</td>
<td>06.081</td>
<td>0</td>
</tr>
<tr>
<td>06.115</td>
<td>06.115</td>
<td>0</td>
</tr>
<tr>
<td>06.147</td>
<td>06.147</td>
<td>0</td>
</tr>
<tr>
<td>06.147</td>
<td>06.147</td>
<td>0</td>
</tr>
<tr>
<td>06.198</td>
<td>06.198</td>
<td>0</td>
</tr>
<tr>
<td>06.232</td>
<td>06.232</td>
<td>0</td>
</tr>
<tr>
<td>06.282</td>
<td>06.282</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 5.9: Image statistics SC 2
Figure 5.10: Callback statistics SC 2

Test run 3

Figure 5.11 shows the results from test run 3 using the SoftwareContinuous mode (software trigger). Figure 5.12 shows the callback measurements from the same test run.

<table>
<thead>
<tr>
<th>SC run 3</th>
<th>Image timestamps</th>
<th>Callback statistics</th>
</tr>
</thead>
<tbody>
<tr>
<td>cam 0 (s)</td>
<td>cam 1 (s)</td>
<td>diff (ms)</td>
</tr>
<tr>
<td>58.915</td>
<td>58.915</td>
<td>0</td>
</tr>
<tr>
<td>58.948</td>
<td>58.948</td>
<td>0</td>
</tr>
<tr>
<td>58.998</td>
<td>58.998</td>
<td>0</td>
</tr>
<tr>
<td>59.031</td>
<td>59.031</td>
<td>0</td>
</tr>
<tr>
<td>59.031</td>
<td>59.031</td>
<td>0</td>
</tr>
<tr>
<td>59.081</td>
<td>59.081</td>
<td>0</td>
</tr>
<tr>
<td>59.115</td>
<td>59.115</td>
<td>0</td>
</tr>
<tr>
<td>59.150</td>
<td>59.150</td>
<td>0</td>
</tr>
<tr>
<td>59.198</td>
<td>59.198</td>
<td>0</td>
</tr>
<tr>
<td>59.232</td>
<td>59.232</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 5.11: Image statistics SC 3

Figure 5.12: Callback statistics SC 3
5.1.3 SoftwareSingle (Software trigger)

Figure 5.13 shows the results from test run 1 using the SoftwareSingle mode (software trigger). Figure 5.14 shows the callback measurements from the same test run.

<table>
<thead>
<tr>
<th>cam 0 timestamp (us)</th>
<th>cam 0 timestamp (s)</th>
<th>cam 1 timestamp (us)</th>
<th>cam 1 timestamp (s)</th>
<th>diff interval (ms)</th>
<th>cam 0 callback timestamps (us)</th>
<th>cam 1 callback timestamps (us)</th>
<th>diff intervals (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>44.116</td>
<td>44.116</td>
<td>0</td>
<td></td>
<td>32</td>
<td>351206526</td>
<td>351166778</td>
<td>0.9</td>
</tr>
<tr>
<td>44.148</td>
<td>44.148</td>
<td>0</td>
<td></td>
<td>0</td>
<td>354270517</td>
<td>354206664</td>
<td>0.9</td>
</tr>
<tr>
<td>44.148</td>
<td>44.148</td>
<td>0</td>
<td></td>
<td>51</td>
<td>357469253</td>
<td>357400172</td>
<td>0.9</td>
</tr>
<tr>
<td>44.201</td>
<td>44.201</td>
<td>0</td>
<td></td>
<td>31</td>
<td>360526332</td>
<td>360060844</td>
<td>0.8</td>
</tr>
<tr>
<td>44.232</td>
<td>44.232</td>
<td>0</td>
<td></td>
<td>0</td>
<td>369041963</td>
<td>369026695</td>
<td>0.8</td>
</tr>
<tr>
<td>44.232</td>
<td>44.232</td>
<td>0</td>
<td></td>
<td>50</td>
<td>365746518</td>
<td>365647919</td>
<td>0.8</td>
</tr>
<tr>
<td>44.282</td>
<td>44.282</td>
<td>0</td>
<td></td>
<td>33</td>
<td>369088132</td>
<td>369070723</td>
<td>0.8</td>
</tr>
<tr>
<td>44.315</td>
<td>44.315</td>
<td>0</td>
<td></td>
<td>34</td>
<td>370300108</td>
<td>370318116</td>
<td>0.8</td>
</tr>
<tr>
<td>44.349</td>
<td>44.349</td>
<td>0</td>
<td></td>
<td>49</td>
<td>376216604</td>
<td>376254334</td>
<td>0.9</td>
</tr>
<tr>
<td>44.398</td>
<td>44.398</td>
<td>0</td>
<td></td>
<td>42</td>
<td>379542265</td>
<td>379325416</td>
<td>0.8</td>
</tr>
</tbody>
</table>

Figure 5.13: Image statistics SS 1
5.1 Synchronization error test using flash clock

Test run 2

Figure 5.15 shows the results from test run 2 using the SoftwareSingle mode (software trigger). Figure 5.16 shows the callback measurements from the same test run.

<table>
<thead>
<tr>
<th>SS run 2</th>
<th>image timestamps</th>
<th>callback statistics</th>
</tr>
</thead>
<tbody>
<tr>
<td>cam 0 (s)</td>
<td>cam 1 (s)</td>
<td>diff (ms)</td>
</tr>
<tr>
<td>08.348</td>
<td>08.348</td>
<td>0</td>
</tr>
<tr>
<td>08.397</td>
<td>08.397</td>
<td>0</td>
</tr>
<tr>
<td>08.431</td>
<td>08.431</td>
<td>0</td>
</tr>
<tr>
<td>08.431</td>
<td>08.431</td>
<td>0</td>
</tr>
<tr>
<td>08.481</td>
<td>08.481</td>
<td>0</td>
</tr>
<tr>
<td>08.514</td>
<td>08.514</td>
<td>0</td>
</tr>
<tr>
<td>08.551</td>
<td>08.551</td>
<td>0</td>
</tr>
<tr>
<td>08.551</td>
<td>08.551</td>
<td>0</td>
</tr>
<tr>
<td>08.598</td>
<td>08.598</td>
<td>0</td>
</tr>
<tr>
<td>08.631</td>
<td>08.631</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 5.15: Image statistics SS 2

Figure 5.16: Callback statistics SS 2
Test run 3

Figure 5.17 shows the results from test run 3 using the SoftwareSingle mode (software trigger). Figure 5.18 shows the callback measurements from the same test run.

<table>
<thead>
<tr>
<th>cam 0 (s)</th>
<th>cam 1 (s)</th>
<th>diff (ms)</th>
<th>interval (ms)</th>
<th>cam 0 (us)</th>
<th>cam 1 (us)</th>
<th>diff (ms)</th>
<th>intervals (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>34.631</td>
<td>34.631</td>
<td>0</td>
<td>3012526.46</td>
<td>3013356.47</td>
<td>0.83</td>
<td></td>
<td></td>
</tr>
<tr>
<td>34.631</td>
<td>34.631</td>
<td>0</td>
<td>3043713.07</td>
<td>3044516.97</td>
<td>0.80</td>
<td>31.2</td>
<td>31.2</td>
</tr>
<tr>
<td>34.714</td>
<td>34.714</td>
<td>33</td>
<td>3105083.5</td>
<td>3106883.89</td>
<td>0.80</td>
<td>31.1</td>
<td>31.1</td>
</tr>
<tr>
<td>34.746</td>
<td>34.746</td>
<td>32</td>
<td>3138299.51</td>
<td>3137363.87</td>
<td>0.94</td>
<td>30.5</td>
<td>30.5</td>
</tr>
<tr>
<td>34.797</td>
<td>34.797</td>
<td>49</td>
<td>3168595.31</td>
<td>3169417.12</td>
<td>0.82</td>
<td>32.1</td>
<td>32.1</td>
</tr>
<tr>
<td>34.830</td>
<td>34.830</td>
<td>33</td>
<td>3200726.81</td>
<td>3199743.4</td>
<td>0.98</td>
<td>30.3</td>
<td>30.3</td>
</tr>
<tr>
<td>34.830</td>
<td>34.830</td>
<td>0</td>
<td>3231027.28</td>
<td>3231849.78</td>
<td>0.82</td>
<td>32.1</td>
<td>32.1</td>
</tr>
<tr>
<td>34.881</td>
<td>34.881</td>
<td>51</td>
<td>3263020.19</td>
<td>3262066.5</td>
<td>0.95</td>
<td>30.2</td>
<td>30.2</td>
</tr>
<tr>
<td>34.914</td>
<td>34.914</td>
<td>33</td>
<td>3294306.76</td>
<td>3293479.15</td>
<td>0.83</td>
<td>31.4</td>
<td>31.4</td>
</tr>
</tbody>
</table>

**Figure 5.17:** Image statistics SS 3

**Figure 5.18:** Callback statistics SS 3
5.2 Long duration test

Figure 5.19 shows the results of the long duration tests of the different trigger modes. The long duration test use only the callback measurements.

<table>
<thead>
<tr>
<th>Trigger mode: Hardware Freerun</th>
<th>Total time in millsec: 1.67784e+006 (ca 28 mins)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Callbacks Cam0: 50228</td>
<td>Number of Callbacks Cam1: 50228</td>
</tr>
<tr>
<td>AVG Cam0: 33.0001 ms</td>
<td>AVG Cam1: 33.0001 ms</td>
</tr>
<tr>
<td>Std_Dev Cam0: 0.520856 ms</td>
<td>Std_Dev Cam1: 0.484697 ms</td>
</tr>
<tr>
<td>Average fps Cam0: 30.3029</td>
<td>Average fps Cam1: 30.303</td>
</tr>
<tr>
<td>MAX Cam0: 35.0579 ms</td>
<td>MAX Cam1: 35.2263 ms</td>
</tr>
<tr>
<td>MIN Cam0: 30.7034 ms</td>
<td>MIN Cam1: 30.7757 ms</td>
</tr>
<tr>
<td>Max synch error is: 1.69037 mill sec.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Trigger mode: Software Continuous with framerate: 30.3</th>
<th>Total time in millsec: 1.74177e+006 (ca 29 mins)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Callbacks Cam0: 52197</td>
<td>Number of Callbacks Cam1: 52197</td>
</tr>
<tr>
<td>AVG Cam0: 33.2406 ms</td>
<td>AVG Cam1: 33.2406 ms</td>
</tr>
<tr>
<td>Std_Dev Cam0: 0.0456576 ms</td>
<td>Std_Dev Cam1: 0.108701 ms</td>
</tr>
<tr>
<td>Average fps Cam0: 30.0837</td>
<td>Average fps Cam1: 30.0837</td>
</tr>
<tr>
<td>MAX Cam0: 34.2299 ms</td>
<td>MAX Cam1: 34.3755 ms</td>
</tr>
<tr>
<td>MIN Cam0: 32.433 ms</td>
<td>MIN Cam1: 32.2886 ms</td>
</tr>
<tr>
<td>Max synch error is: 3.14775 mill sec.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Trigger mode: Software Single</th>
<th>Total time in millsec: 1.81703e+006 (ca 30 mins)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Callbacks Cam0: 57813</td>
<td>Number of Callbacks Cam1: 57813</td>
</tr>
<tr>
<td>AVG Cam0: 31.1992 ms</td>
<td>AVG Cam1: 31.1992 ms</td>
</tr>
<tr>
<td>Std_Dev Cam0: 0.593931 ms</td>
<td>Std_Dev Cam1: 0.580388 ms</td>
</tr>
<tr>
<td>Average fps Cam0: 32.0521</td>
<td>Average fps Cam1: 32.0521</td>
</tr>
<tr>
<td>MAX Cam0: 33.3717 ms</td>
<td>MAX Cam1: 33.4599 ms</td>
</tr>
<tr>
<td>MIN Cam0: 29.1523 ms</td>
<td>MIN Cam1: 29.0082 ms</td>
</tr>
<tr>
<td>Max synch error is: 1.85385 mill sec.</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 5.19**: Long duration test with three different trigger modes
6.1 Conclusions

Here I discuss the pros and cons of the three different trigger modes by analysing the test results presented in chapter 5 compared to the system requirements described in section 1.4 and by analysing the overall system design. Here are also the answers for the problem description in section 1.5.

**HardwareFreerun (external trigger):**
When analysing the results of the flash clock experiment described in section 5.1.1 it seems that there is no synchronization error by examining the images and correlating time stamps in figure 5.1 but the flash clock has only millisecond resolution and the screen cannot show images that changes faster than 16.7 ms so we cant measure differences below 16.7 ms and the accuracy of the flash clock is not known. We also see very deviating numbers in the frame interval measurements when analysing the images in figure 5.1 compared to the callback intervals in the same figure, but that also comes from the refresh rate of the computer screen. If we instead look at the callback statistics in figure 5.19 of the long duration test in section 5.2 the external trigger shows good results. The average interframe interval, standard deviation and average fps meet the system requirements. Remember that this is callback statistics and not the exact measurements when the images are being taken, but we can use this to compare the three trigger modes. One thing we can say for certain is that the cameras get the trigger pulse at the same time, because we are using the same output from the MCU for both trigger signals, .

**SoftwareContinuous (software trigger):**
In the first test run in the flash clock experiment it seems that we loose the first
image from one of the cameras when examining the time stamps in figure 5.7. The image time stamps in figure 5.7 runs diagonally from left to right meaning that cam 1 is one frame a head all the time. This is also confirmed in the callback statistics in figure 5.8 where cam 1 has one more callback than cam 0. This is not happening in the second and third test run and not in the long duration test. So this trigger mode can be unstable, at least with the multi camera software we created. But like the external trigger mode we don’t measure any synchronization error in test run two and three. If we look at the figure 5.19 in the long duration test result in section 5.2, this trigger mode produce good numbers in standard deviation but poorer in the rest.

**SoftwareSingle (software trigger):**

Also here we don’t see any synchronization error in the flash clock experiment but in the long duration test the numbers are not complying with the system requirements. This is mainly because it is hard to trigger the two cameras at an exact interval in this mode. We use two different threads with delayed loops and the accuracy of the delay its not as good as we want it. We also have an internal delay in the cameras of about 10 ms in this trigger mode. Information from the manufacturer tells us that this trigger mode should only be used taking single images hence the name.

The conclusion of the above is that it is preferable to use the external trigger because it shows good statistics and we have more control over the system. If we choose one of the two software trigger options we hand over this control to the software and the operating system. Because the Windows OS is not a "real-time" operating system this can be risky. In a real time operating system like for example LinuxRT, the programmer has more control over the operating system and how it prioritize the different tasks that are executed. A real-time OS also runs the applications with more precise and determined timing and with more reliability. One drawback of having an external trigger is that you add extra components to the system such as extra trigger cables and an external microcontroller.

Answers to the questions in section 1.5:

- 1. As derived from the results of the experiments, the hardware (external) trigger show the best results compared to the system requirements but further analysis may be needed as described in section 6.2.2.

- 2. Because the multi camera system is in a relative confined space inside of the truck, and by examining the test results the preferred synchronization technique for this system is the hardware trigger.

- 3. Since the hardware trigger meet the demands of this particular system there is no need to use another synchronization technique, but if the amount of cameras are increased to cover the entire 360 degrees field of view in the truck, there may be interesting looking into the network synchronization technique instead. If the number of cameras are increased the
amount of image data can be too large for a single PC so the use of network camera synchronization using multiple PC:s can be preferable.

6.2 Future work

6.2.1 External trigger

We could measure the accuracy of the external trigger pulse by recording it with an digital oscilloscope for 30 minutes and then calculate the standard deviation of the frequency. If the accuracy of the external trigger signal produced by the MCU is too low it could be possible to use the millis() function to calculate the expired time between pulses rather than the delay() function. We could also implement a user input of the external trigger frequency in the multi camera software's config file instead of re-programming the MCU each time you would like to change the trigger frequency i.e. FPS. This feature was implemented in a test program but not in the finalized system.

6.2.2 Experiments

In the system requirements we see that the synchronization error should be max 0.1 ms but our experiment that we conducted with the flash clock wasn't that accurate so we could set up another experiment using an array of flashing LEDs with known frequency. [Litos et al., 2006] describe a usable synchronization test of this type. Another way of measuring the synchronization error is to take pictures of an electronic clock with micro second accuracy instead of the flash clock. In that case we do not need to bother with the refresh rate of the PC screen.
Conclusions and future work


41
Clark F Olson, Larry H Matthies, Marcel Schoppers, and Mark W Maimone. 
Rover navigation using stereo ego-motion. *Robotics and Autonomous Systems*, 

Tomáš Svoboda, Hanspeter Hug, and Luc Van Gool. Viroom — low cost syn-
7. doi: 10.1007/3-540-45783-6_62. URL http://dx.doi.org/10.1007/ 

Charles Thorpe, Martial H Hebert, Takeo Kanade, and Steven A Shafer. Vision 
and navigation for the carnegie-mellon navlab. *Pattern Analysis and Machine 

Alexander Vibeck. User manual and source code for bachelor thesis: Synchro-
nization of a multi camera system. Contact author for a copy at 
familjenvibeck@gmail.com, May 2015. Cited on pages 21 and 22.
Upphovsrätt
Detta dokument hålls tillgängligt på Internet — eller dess framtida ersättare — under 25 år från publiceringsdatum under förutsättning att inga extraordinära omständigheter uppstår.

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

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

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

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

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

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

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

© Alexander Vibeck