NTNU – Trondheim
Norwegian University of
Science and Technology

Realtime auto tracking CCTV cameras for increased safety and productivity

Joakim Skjefstad

2015-xx-xx

MASTER THESIS
Department of Engineering Cybernetics
Faculty of Information Technology, Mathematics and Electrical Engineering
Norwegian University of Science and Technology

Supervisor 1: Professor Tor Onshus
Supervisor 2: Doctor Mads Hvilshøj, MHWirth AS
Supervisor 3: Thor Brantzeg, MHWirth AS
Supervisor 4: Johan Hansen, MHWirth AS

Abstract

Here comes the abstract.

Abstract in Norwegian

Here comes the abstract.

Preface

This master thesis is submitted in partial fulfilment of the requirements for the degree MSc. in Engineering Cybernetics at the Norwegian University of Science and Technology.

The motivation has been to study the feasibility of using computer vision offshore, and follow this up with an implementation of a potential application, hopefully aiding in increased safety and reduced cost of operations.

The motivation for this work has been to implement an automated CCTV system for tracking machines that are in motion, improving upon work previously done in the field. By visually following machines, I hope this work will lead to improved safety and productivity in the industry. Part of this thesis will also look into increasing reliability of control systems that are managing drill pipes on an oil rig.

I would like to thank my supervisor Professor Tor Onshus from NTNU and co-supervisor Doctor Mads Hvilshøj from MHWirth AS for their guidance and support throughout the project.

I would also like to thank Omega Verksted for their support when I needed to bounce programmatic ideas and build parts of the test setup, and also for supplying an abundance of coffee and pictures of cute kittens when I needed it the most.

No work would be complete unless I also thanked NTNUI Dykkergruppa, a student organization that has been much like a family to me through my years at the university. Without their support, I would probably never have finished my degree.

Thank you.

Keywords: computer vision, offshore, extreme environment, decision support, motion tracking, cctv, drilling vessel, oil & gas, drilling applications, machine vision, opencv

Summary

Here comes the summary.

Contents

List of Figures

Chapter 1

Introduction

1.1 Background

The Norwegian economy is cooling under a pessimistic oil price of close to 45 USD per barrel as of autumn 2015. Under heavy pressure to perform, companies involved in the exploitation of hydrocarbons are cutting costs while trying to increase productivity.

As a means of improving profitability, decision support systems have been developed, which tries to provide real-time support when drilling oil wells. One example of these was a product of work done at NTNU.

A roadmap for automating drilling systems was established in June 2013 with the objective of providing a guideline to the future emergence of drilling systems automation.

Automation of pipe handling have been pursued by several companies in the oil rig equipment production market with moderate success. The industry have realized that more may be gained from supporting humans in the center of the operation, than to completely eliminate humans. As such, systems to support the human driller need to be developed.

Support systems that may improve safety and productivity includes improvements to the graphical user interface, presentation of relevant and aggrevated data, as well as integrated camera systems.

The main objective of this MSc thesis is to implement a system that provides a better situation overview through automated CCTV tracking of machines.

This MSc thesis should address the following:

- Implementation of an auto tracking CCTV system using glyphs as visual descriptor.
- Analysis of dataset with real-world weather over a long period.

- Prototyping of an algorithm to detect casings and pipes in a fingerboard.

1.1.1 Problem formulation

How can computer vision assisted camera tracking be exploited to increase safety and situational awareness in any process involving heavy machinery and remote operation?

How will GPU-acceleration of the computer vision algorithm affect the system?

How can we reduce the latency of the live camera feed to make feedback control of the camera better?

How can we track an object across multiple cameras?

We will develop a solution to provide benefits to a human driller operating heavy machinery on an oil rig, but the solution is not limited to this application. Our end goal is to reduce the possibility of an undesired event, be it either damage to humans or expensive equipment.

1.1.2 Literature survey

As discussed by Sklet (2005), the concept of a safety barrier is not clearly defined and its meaning is ambiguous. With this in mind, we are looking to implement what Rausand (2014) describes as a proactive safety barrier, also known as a frequency-reducing barrier, in other words a system to reduce the frequency of undesired events.

Semi-automated CCTV surveillance have been considered by Dadashi et al. (2012) as a method of increasing the capacity of a human operator in a traditional human surveillance situation. The reliability for fully automated systems were not considered good enough for an operator to trust. The findings recommend providing feedback about system confidence and accuracy to the operator, which makes the automated component of such a system more 'visible' to its user. For the case of this thesis, the act of displaying visual cues overlaid on the CCTV images are considered. This would hopefully increase trust, and expose the automated component of our fully automated tracking system.

The machine vision algorithm that was implemented in the authors earlier work, as mentioned in work done by Boyers (2013) and Kirillov (2010), will be used to recognize the distinct symbol, hereafter called a glyph.

The challenges of outdoor machine vision in uncontrollable weather and lightning conditions have been raised by the author in the same unpublished project thesis. Figure 1.1 shows a series of images captured by the author, which shows the differences in color, clarity and reflections.

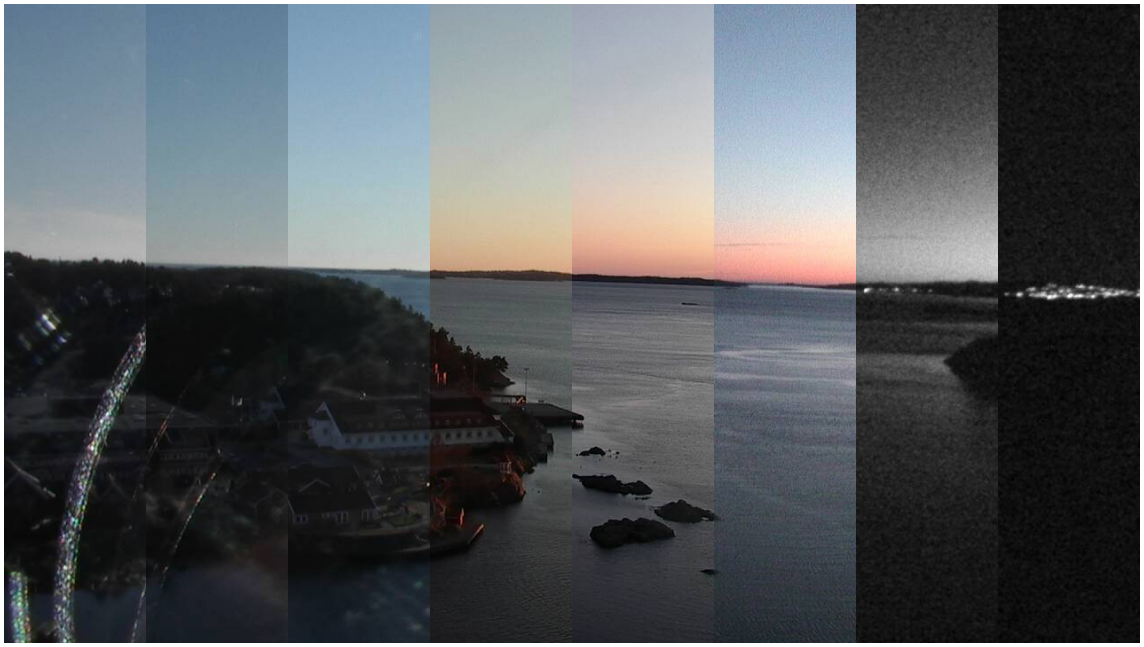

Figure 1.1: Timelapse through a day of a CCTV camera in Kristiansand, 4th of March 2015. Source: Own work

Our camera of choice, an AXIS Q6045 was selected both by its widespread use in the oil- and gas industry, and because this is what the author had at hand. It is a modern high-definition PTZ-camera produced by Axis Communications. Axis Communications have provided a white paper which provides a good overview of the various elements that increase latency in a live video stream. This document is available online. (Communications, 2015b)

The inherit differences between analog and digital video transmission have been examined by Hill et al. (2010), and the conclusion was that latency present in digital transmission is approaching negligible for normal usage. Still, the article presents findings that show latency of a digital transmission being more than 5x of the analog transmission latency. The upsides of digital transmission include increased quality of image, flexibility of digital encoding and ability to use analytic software.

Work done by Svensson and Söderlund (2013) involved methods to reduce the delays that are inherently found in digital video transmission systems and the control of these. Their research was done on AXIS Q6035 and AXIS Q6032 cameras. The author assumes these to be closely related to the AXIS Q6045, and their findings therefore useful for work done in this thesis. Findings include camera firmware being among the key factors for video delay, as the stock cameras have implemented an inefficient communication scheme, and there are other suggestions to reduce delay presented. For the scope of this thesis, we will have these delays in mind and build

around them, as any firmware upgrades have to come from Axis themselves if any company would consider using them.

Tracking of objects across multiple cameras have been explored with most focus on overlapping camera views. Some work has also been done on non-overlapping camera views by jav (2003), where camera topology and path probabilities are learnt without any inter-camera calibration. As soon as one uses images from several sources, the timestamp for the images becomes crucial as a point of reference. Through the use of Parzen windows, the inter-camera space-time probabilities can be mapped. The method mentioned does require a learning phase.

1.1.3 What remains to be done?

As a summary of the literature survey, we see that much work has been done in the different fields, but not much have been found on combining the results of these into a solution that can provide modern, automated CCTV tracking of industrial processes involving heavy machinery in a way that retains the human operator in the center of the process.

We aim to break down the information silo and combine as much as possible, given constraints of time, into a proof of concept which can be the stepping stone to a commercial product.

1.2 Objectives

The objectives of the work done through this thesis consists of:

1. Implementation of a multi-source GPU-accelerated machine vision program that can control a CCTV camera and follow a symbol

2. Implementation of a proof of concept pipe-detection algorithm for fingerboards

1.3 Limitations

The limitations that relates to this study includes both technical challenges, environmental and operational conditions as well as oil politics in a broad sense, but focus is on the technical challenges for the sake of brevity.

As CCTV cameras have evolved, the video transfer method has gone through some changes to cater for higher resolution and more true representation of the world as observed. By this, analog signals have been replaced by digital signals. Analog transmission is known for being both robust, simple and near-instant, however they are prone to signal deterioration which may affect machine vision algorithms, and

their flexibility of location is not as good as modern digital transmission. With digital transmission, usually over Ethernet, the cost of these systems have gone down, flexibility have gone up and resolution as well as control has improved, yet this have introduced new challenges. Data loss is a real possibility, increased latency through encoding and decoding of the video stream and a shared network highway puts more demand on the implementation. One serious limitation to the implementation of the system as presented, is therefore as a feedback system, the upper latency limit for which when the system becomes unstable.

The oil industry is considered a conservative one, as lost production time can lead to extreme expenses and questions regarding responsibility and passing on cost to service providers. Technology is also highly guarded, as the sheer amount of money that can be gained from saving a fraction of time, means that cooperation is not common in the industry, and transparency of systems and solutions may be less than ideal. The idea then, of making changes to what already works, is not easy to plant, and this also limits the available data set for the purpose of making robust systems.

Heterogeneous computing platforms are still considered to be in its infancy, and both CUDA as well as OpenCL is under active developement. Choosing one technology will lead to an exclusion of either benefits or available computing platforms. A limitation here is that the resulting speed and benefits of heterogeneous computing are not set in stone, and that there will always be improvements that can be done to make an otherwise unworkable system to a brilliant idea.

The software world progress quickly, and new solutions can suddenly become obsolete, requiring legacy maintenance and making in-house developed software seem much like building a tower of cards as the flux of developers who knows the system changes over time. This would make an externally maintained solution seem like a better idea, but sharing information to make the development work is not an easy task as each company protects its own interests.

Machine vision algorithms implemented will not be robust enough to handle all possible lightning conditions, and some of these may assume that a certain color can be identified. This means that artificial lightning is paramount for reliable machine vision outdoors, if we are to only rely on optical sensors. Alternatives exist, but these will not be explored further in this thesis. The author have made a summary of these that can be read in the unpublished project thesis Skjefstad (2014).

It is considered a hard challenge therefore, to make a truly reliable system that works in the real world. Making a tabletop solution is not nearly enough to allow a big company to test this offshore at a customers platform, and much work remains before a fully commercial solution is ready for sale.

Seeing past these limitations, however, is a world of possibilities, which this study tries to explore.

1.4 Approach

1.4.1 CCTV tracking system

The implementation of the CCTV tracking system will be based on the authors previous work and be written in C++ with heterogeneous support from a GPU to increase performance.

After this system has been built, it will be tested with and without GPU acceleration, to determine if the full solution becomes more stable and reliable.

A data-set will be analyzed to test the robustness of the machine vision algorithm and comment how snow, sun and other real-world factors affect the output.

It will also be tried to use multiple video sources, however due to the lack of several CCTV cameras, this will only partially be explored using a common webcamera, as a means of doing camera handover.

1.4.2 Pipe detection system

The construction of a proof of concept pipe detection system for fingerboards will be done in a rapid prototyping environment, to show that it is possible to increase control system awareness in existing infrastructure on the oil rig.

1.5 Structure of the report

All the software developed as a part of this project can be found at the authors personal repository at Github, Skjefstad (2015), feel free to use this for future non-commercial work. The Latex source is also available. Some information may have been omitted to hide confidential company information.

1.6 Structure of the DVD

The DVD contains a snapshot of the latest Github code repository as of the date of this report.

Chapter 2

Theory

Following is a brief introduction to some of the aspects that affects the solution, and should be among the things considered when developing a commercial application.

2.1 Hardware

2.1.1 Camera

The camera specifications and performance directly affects the results. It is the source of input data for the computer vision software, and it may also be the output, which will be the case when the PTZ is controlled by the software.

Some cameras support extra applications that run in their firmware, which can for example track moving objects and "fence" an area so that any objects who enter a region of interest, will raise an alarm. The CPU and memory of embedded systems is constraining, so any complex applications may be better off being executed on a stand-alone computer.

Resolution and Field of view

Resolution and Field of view are related such that an increased field of view leads to a lower amount of pixels available to distinguish objects. Modern image sensors come with capabilities of capturing 1920x1080 pixel images, whilst older image sensors may capture 320x240 pixel images. The field of view is a function of the optical objective as well as the size of the image sensor, and their distances in relation to each other.

Pan, Tilt and Zoom

Many modern CCTV cameras come with motors that can pan and tilt the camera, as well as optical and digital zoom functionality. Their ability to pan, tilt and zoom have uses for scanning a large area, as well as investigating small areas in detail. The speed and accuracy of which the pan, tilt and zoom can be manipulated may be of importance in some applications.

Focus

If light from an object converges as much as possible, it is considered in focus. Would the light rather diverge, it is considered out of focus. This leads to blurring of the object or scene in question. Many cameras come with automatic focus that will try to adjust the focus so that a target object becomes in focus. The focus is then controlled by either an ultrasonic motor or a stepper motor.

Iris

The iris determines how much light enters the camera, and it is also a factor that can determine the focus range of a camera. An open iris leads to larger amounts of light and thus works better in low-light conditions, but the depth of field gets smaller. The amount of light that enters and hits the image sensor also determines how long the iris should be kept open before a picture is fully captured, and less light means that it will need to stay open longer, which in turn leads to motion blur if an object is in motion. Many cameras come with automatic iris control, that will try to adjust the iris so that the picture does not get too bright or too dark. This is also called DC-iris. Axis Communications have introduced precise iris control to some cameras, known as P-iris, which provides improvements to contrast, clarity, resolution and depth of field beyond the DC-iris.

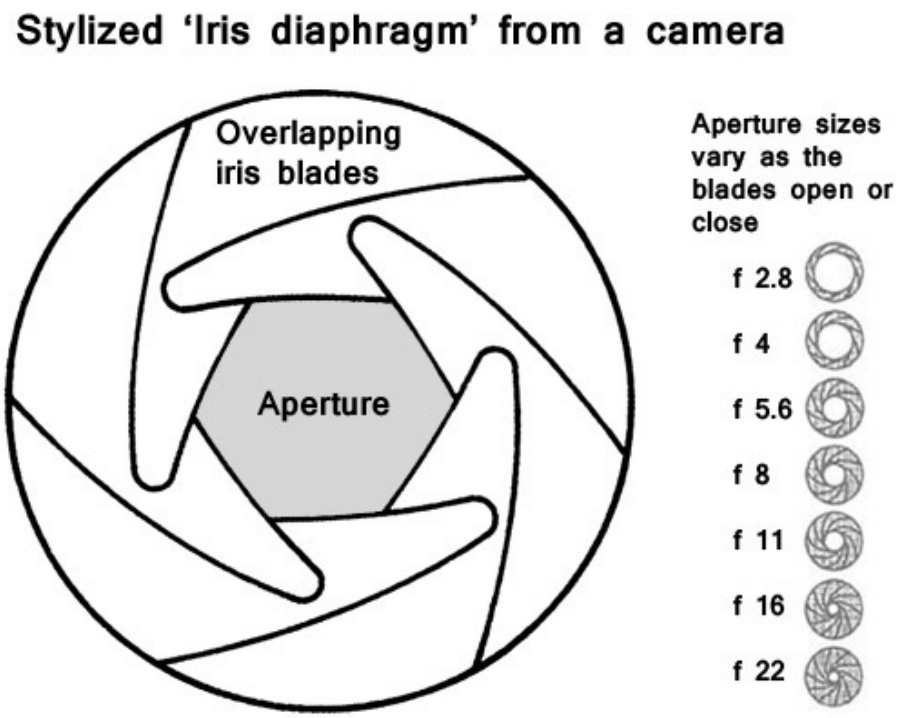

Figure 2.1: Iris. Source: Online Photokonnexion (2015)

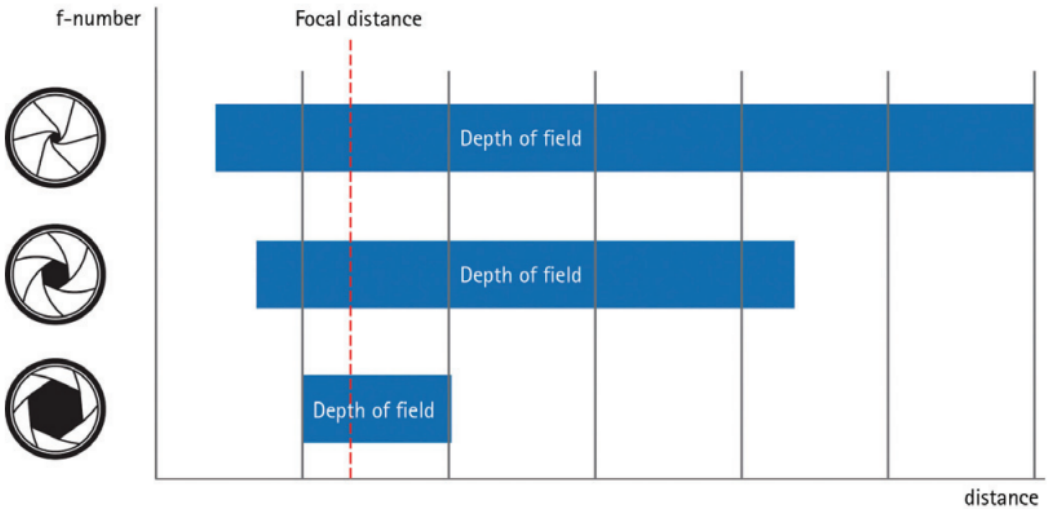

Figure 2.2: Correlation between iris opening and depth of field. Source: Online Communications (2015a)

Frame Rate

The number of frames that can be captured per second determines the frame rate, and is usually used to give the illusion of movement in a movie. The human eye usually sees 25 frames per second as a movie, any less will deteriorate the experience of watching a movie. A large amount of frames per second requires a multiple of the image resolution of bandwidth to transfer the images to a viewer. When it comes to CCTV cameras for surveillance, frame rate of stored movies are reduced to allow for longer storage of material.

Image Quality

The quality of an image is related to the quality of the optics, quality of the image sensor and amount of light available at a scene, as well as the firmware which translates the impression to data, in addition to other factors. Modern image sensors can also adjust the sensitivity of the sensor so that previously dark scenes look brighter, at the cost of increased pixel noise. A full overview over the factors that affect image quality can be found at (Imatest, 2015).

Camera Interface

Which interface that is supported for transferring images is an important factor to consider, both to reduce cost and increase performance. Some of the most used interfaces include USB, FireWire and Ethernet. It is also possible to find CCTV cameras that rely on Wi-Fi, but these are prone to intermittent frame drops if environmental conditions blocks the signal. The most cost-effective method of interfacing a camera is through Ethernet, as the cables are cheap and the Ethernet standard supports a large amount of data. Maximum theoretical capacity of the most common Ethernet link is 100 Mbit per second, but this is quickly shifting to 1000 Mbit per second for modern Ethernet.

2.1.2 Data transmission

A function of the camera hardware and firmware, the camera interface and path of transmission as well as any processing on the way, several aspects will affect data transmission from the image sensor to the screen. An unfortunate side-effect of data transmission is that an image lags after the actual event got captured, and we commonly call this for latency.

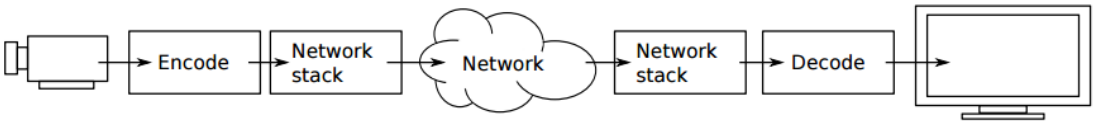

Figure 2.3: Diagram of data through an IP network. Source: Svensson and Söderlund (2013)

2.1.3 Glyph visibility

The glyph symbol, if it is going to be used outdoors or in difficult lightning situations, may have reduced visibility when light hits the glyph from some angles. The angle of light from a sun hitting a glyph can change with the time of the year, time of the day and any man-made sources of light. The method of drawing and displaying the symbol is important to consider. Using a thin translucent plastic sheet to laminate a paper print of the glyph may give specular reflections that appear white. The specular reflection is a function of the glossiness of a surface. Thus to increase reliability of glyph detection, the surface should have a low glossiness.

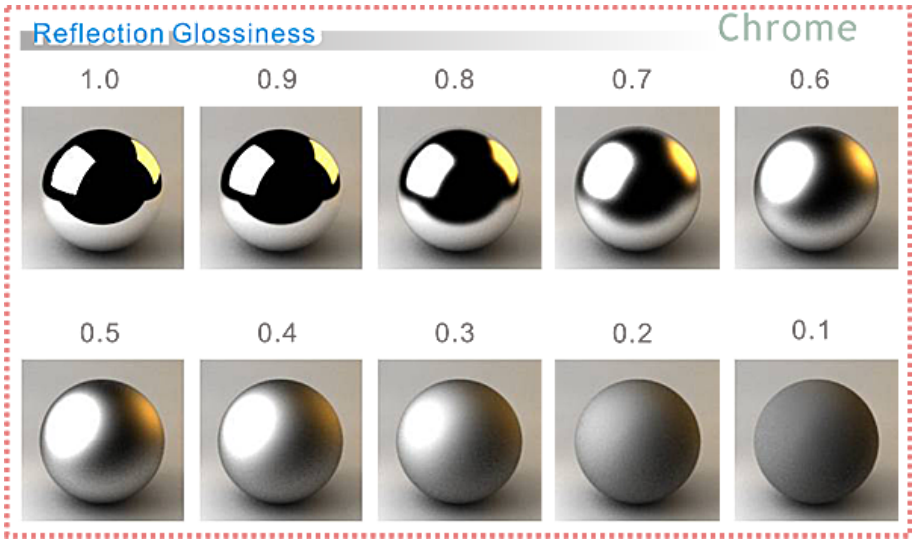

Figure 2.4: Glossiness and its change on reflection as shown in raytracing software. Source: Online VRay (2015)

2.1.4 Computing platform

The processing of images may be done by humans, in which the image is sent directly to a screen. In cases where we want to use machine vision algorithms, a general purpose computer is typically doing this job.

The computing platform is connected to the camera, and it gathers images which is then processed by a computer program.

Computer performance using a central processing unit is steadily increasing as new technologies evolve, but a relatively new method uses a graphical processing unit to accelerate application code. This allows for highly parallel execution of code, that may for some algorithms, speed up their execution and in turn speed up the program.

2.2 Software

2.2.1 Threading

In order to allow several different threads to run simultaneously, the operating system supports threads. It is also possible to delegate a thread to a specific computing core in a central processing unit, if there are more. This have both advantages and disadvantages.

Advantages include the ability for the computer to use idle processing time, and also allow for blocking functions to run in seperate threads to allow the main thread to continue running.

Disadvantages include possibilities of interfering with each other if they share memory, and it is also notoriously challenging to write good multi threaded applications, which in turn may lead to the program not functioning as expected.

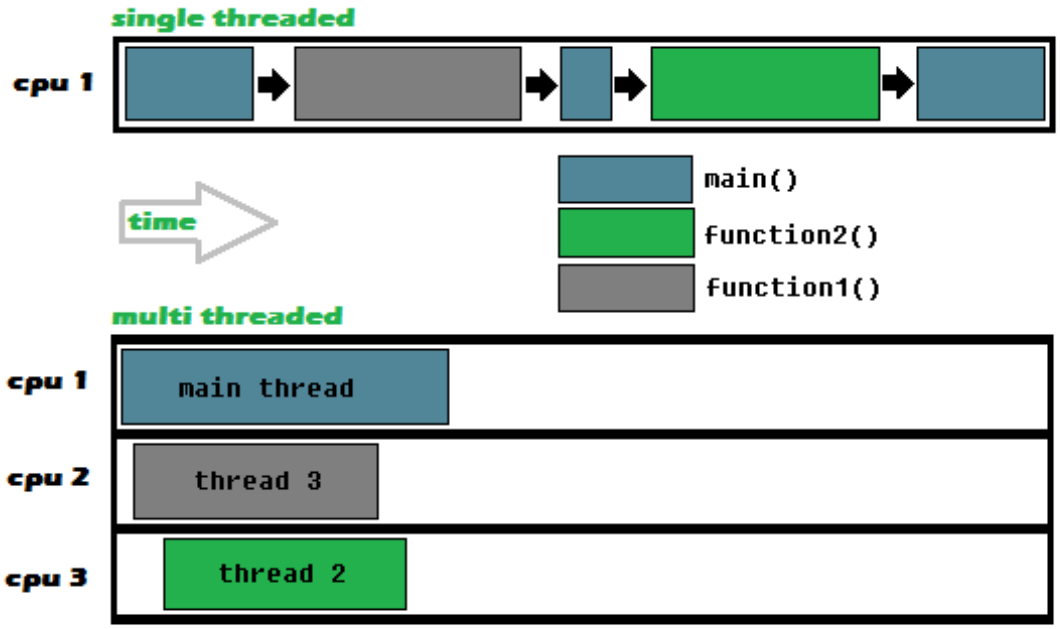

Figure 2.5: Singlethread versus multithread execution. Source: Online Codebase (2015)

Modern central processing units contains several computing cores, which can run their own threads if the programmer wishes. The number of computing cores usually range from one to eight in personal computers.

2.2.2 Heterogenous computing

Systems that utilize dissimilar processing cores are known as heterogenous computing systems. Not only do they have the benefit of several processing cores, they also bring the benefit of having dissimilar processing units that work differently and are better at handling specific tasks.

Heterogenous System Architecture can be used to integrate central processing units and graphical processing units, the alternative is to use a parallel computing platform like OpenCL or CUDA that relieves the programmer from having to move data between the processing units themselves.

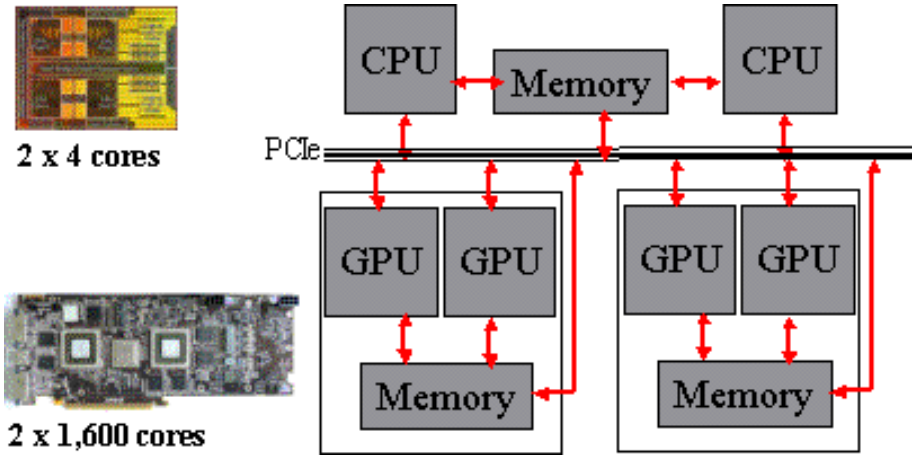

Figure 2.6: Conceptual overview of a heterogeneous system with CPU and GPU cores. Source: Online Technology (2013)

CUDA

The NVIDIA Corporation released the CUDA platform in June 2007, and is a parallel computing platform that only works with NVIDIA graphic cards. It can also utilize OpenCL. Language bindings exist for many programming languages, and it provides both low-level and high-level APIs.

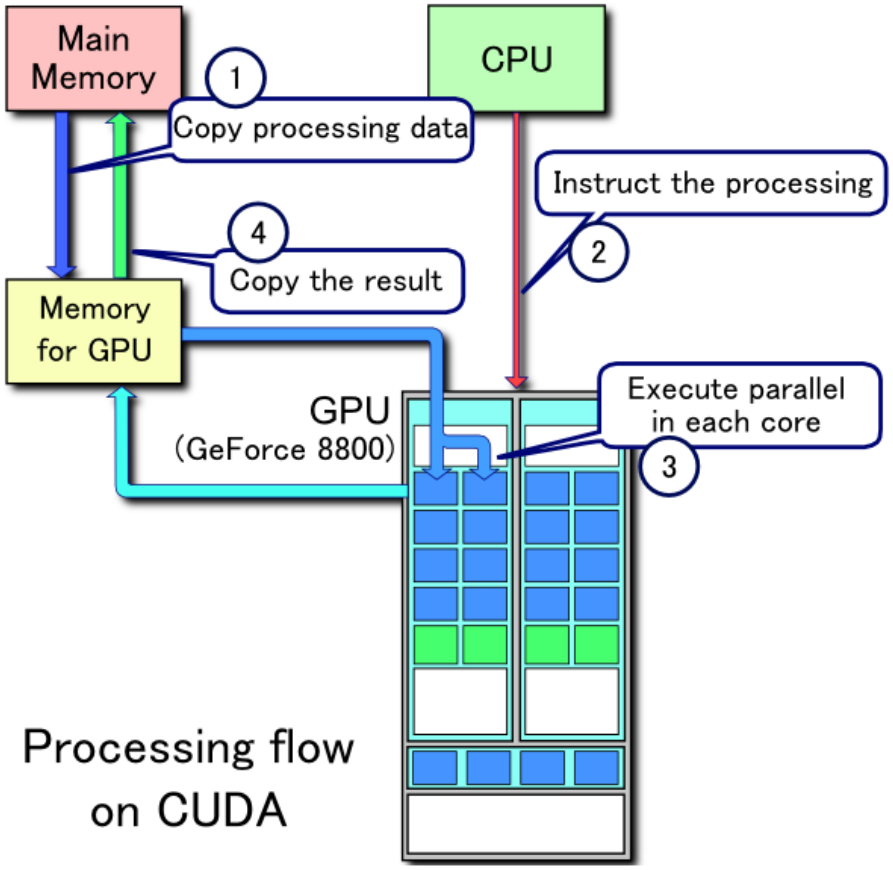

Figure 2.7: CUDA processing flow. Source: Online Wikipedia (2015a)

OpenCL

Apple Inc. authored a language and an API that executes across central processing units, graphical processing units and other processors in August 2009. This is now maintained by the Khronos Group, which is a non-profit consortium that develops open standards for graphics, media and parallel computation. They also maintain OpenGL, which sees great use in 3D applications.

OpenCL sees all computing devices, not only the graphical processing units, and a key feature of it is to be portable and run on any system that conforms to the standard.

A comprehensive study by (Fang et al., 2011) in 2011 compared CUDA to OpenCL, and the findings suggests that CUDA performs 30 percent faster than OpenCL when a program is directly translated between the two platforms. However, the conclusion is that there are no reason for OpenCL to perform worse than CUDA under a fair comparison, and OpenCL is considered a good alternative to CUDA.

2.2.3 Machine vision

Machine vision is a relatively new field that uses image capture and analysis for automating tasks. It sees wide use in industrial manufacturing for quality control and process automation, and also in more recent times, autonomous vehicles.

Implementing machine vision in software is often done by using vision libraries, and one well-known is the Open Source Computer Vision Library. The short-form is OpenCV, and it is currently on its third release, being maintained by a russian company named Itseez and developed by contributors all around the world.

OpenCV 3.0 gold release was made available in 4th of June 2015. It supports OpenCL using its transparent API, and support for CUDA was developed in 2010. Both the OpenCL and CUDA support is still under active development.

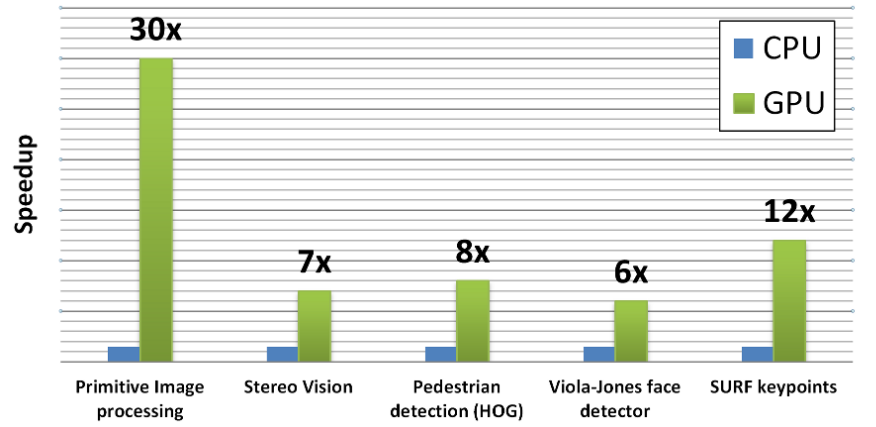

Figure 2.8: Performance speedups when using CUDA and a GPU. Source: Online OpenCV.org (2015)

2.2.4 Linux

An alternative to Microsoft Windows, the operating system that can run on the widest range of computer architectures is Linux. The operating system kernel was first released in 1991 by Linus Torvalds, and it is a clone of UNIX. It is a free and open-source collaboration, being developed by programmers all around the world. The great power of Linux comes through its flexibility, and it or a flavor of it is commonly found powering the servers that make up the world wide web. Every flavor of Linux is known as a distribution, and most of the larger distributions come with great package management tools. For Ubuntu, we can use the same tools as are used in the Debian distribution, which includes apt-get, to install the latest releases of open source software. It is possible to optimize software running on a

Linux system further than what is possible with a Microsoft Windows system, and there is by default less overhead on Linux. Some downsides of using Linux include a very fragmented software world, and many bleeding edge updates that can make the system stop working if carelessly updated. The learning curve is also steeper, and a lot of frustration is to be expected if one has not worked with Linux before. A great resource to learn more about the Linux operating system is Wikipedia (Wikipedia, 2015b) and Linus Torvalds GitHub-repository for Linux (Torvalds, 2015).

Chapter 3

Case study

3.1 Glyph tracking

3.1.1 Background

The norm in the industry is that CCTV cameras are manually selected depending on needs, usually through using touch screens that display a set of four pictures at once. They are controllable through PTZ, but they are usually only moved to preset locations, by navigating the user interface.

Figure 3.1: Modern driller cabin. Source:

3.1.2 Goal of case study

This case study is intended to further improve upon the work done in the unpublished project thesis covering work done by the author in the year 2014. Skjefstad (2014) describes the original idea, but it is briefly repeated here for brevity.

Through the detection and tracking of simple symbols attached to heavy machinery on an oil rig, it is possible to allow CCTV cameras to follow machines without any extra user input. These simple symbols are called "glyphs", and their design is chosen so to reduce processing requirements and increasing possibility of detection by the algorithm.

The case study described in Skjefstad (2014) was implemented in an interpreted programming language Python 2.7 using OpenCV 2.4, while this new implementation is being written in a compiled programming language C++11 using OpenCV 3.0, which was recently released.

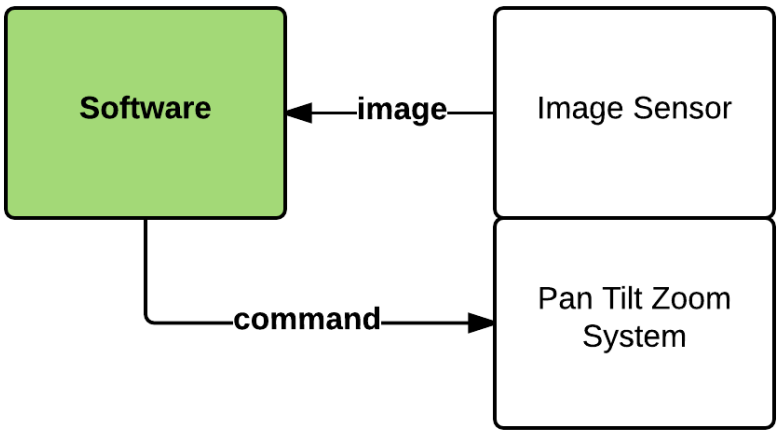

Figure 3.2: System overview using a single camera. Source:Own work, based on Skjefstad (2014)

The simple system as shown in figure 3.2 can be extended to use multiple sources. We extend the system to fulfill its role on an imaginary oil rig, where it allows the drillers to easily follow a moving top drive.

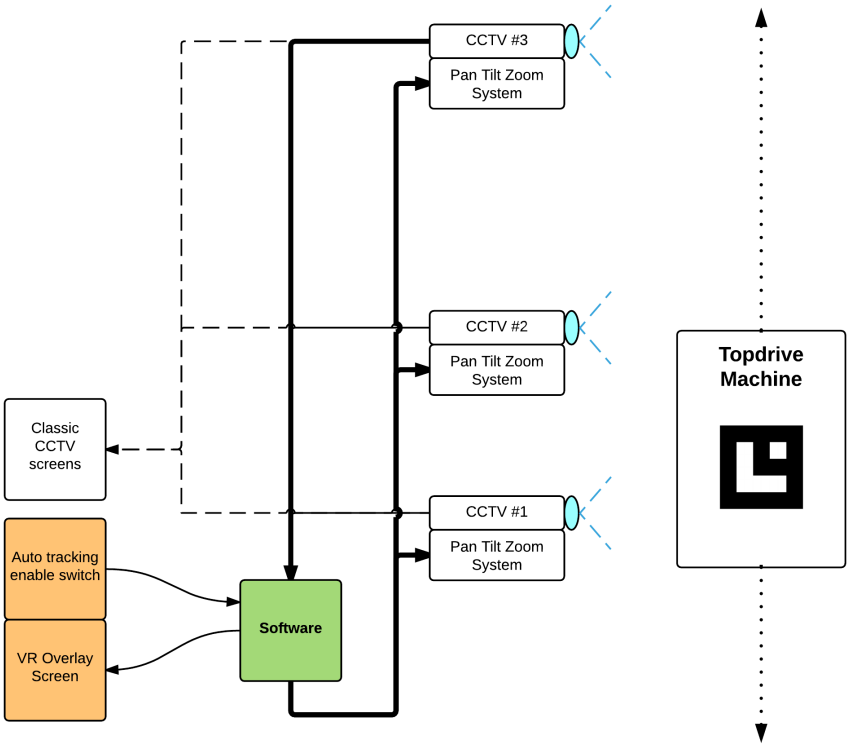

Figure 3.3: System overview when used on an oil rig. Source: Own work

With the software in place, controlling several CCTV cameras as shown in figure 3.3, we hope to allow the control loop to quickly and efficiently follow the machine as it moves about.

At the end of this case study, we will see a side-by-side comparison of the old and new implementation, and consider differences in performance.

3.1.3 Design of the machine vision algorithm

The machine vision algorithm remains the same as earlier developed, as to give us equal grounds for comparison. There are steps that can improve detection rate and reduce processing requirements which will be mentioned, but not necessarily implemented. For details about the algorithm described in figure 3.4, please refer to the unpublished project thesis by the author (Skjefstad, 2014).

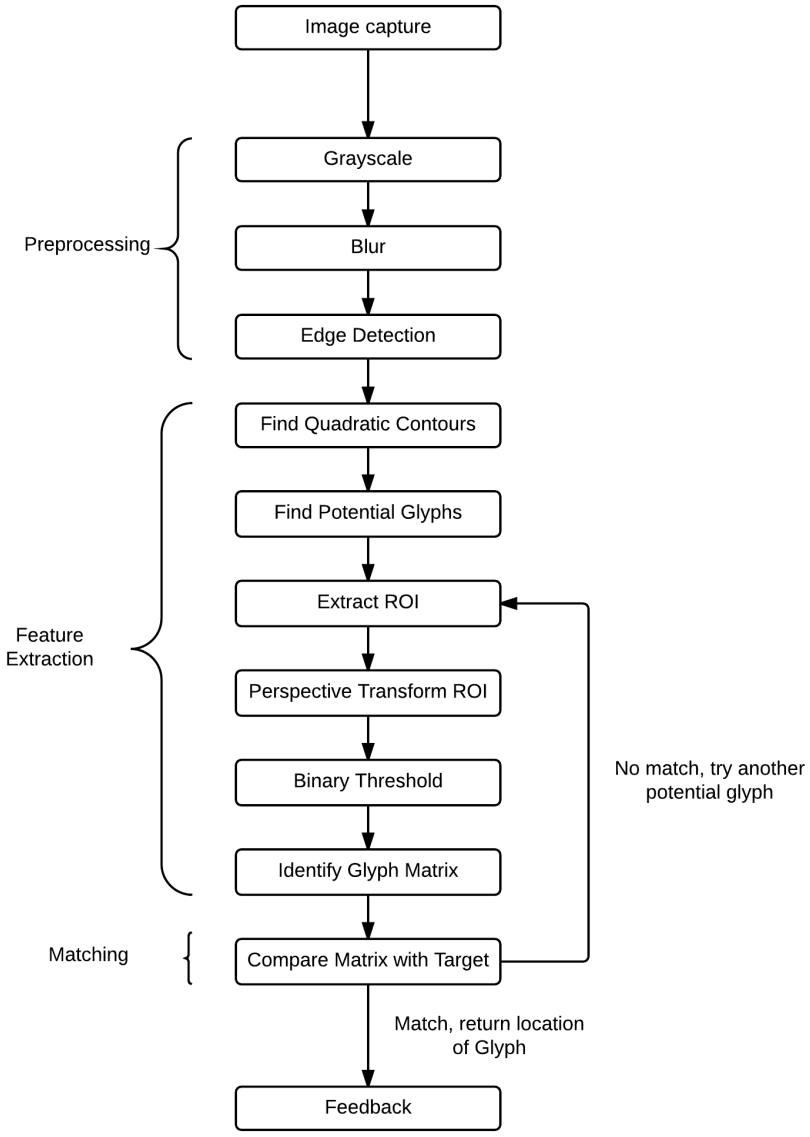

Figure 3.4: The vision algorithm as implemented. Source:Skjefstad (2014)

3.1.4 Design of the software

Using C++11, the software is expected to run faster than the similar Python 2.7-version implemented in Skjefstad (2014). When we have OpenCL-support in hardware, a greater speedup is expected to be seen for parts of the machine vision algorithm.

The software is designed to run in a single thread, despite of potential benefits from using multiple threads as described in Chapter 2, because of difficulties with making a multithreaded application behave as expected. This also means that the comparison between the Python 2.7-version and the C++11-version stands on equal ground. A flowchart of the software as a single thread can be seen in figure 3.5, where the machine vision algorithm is implemented inside Camera.FindGlyph().

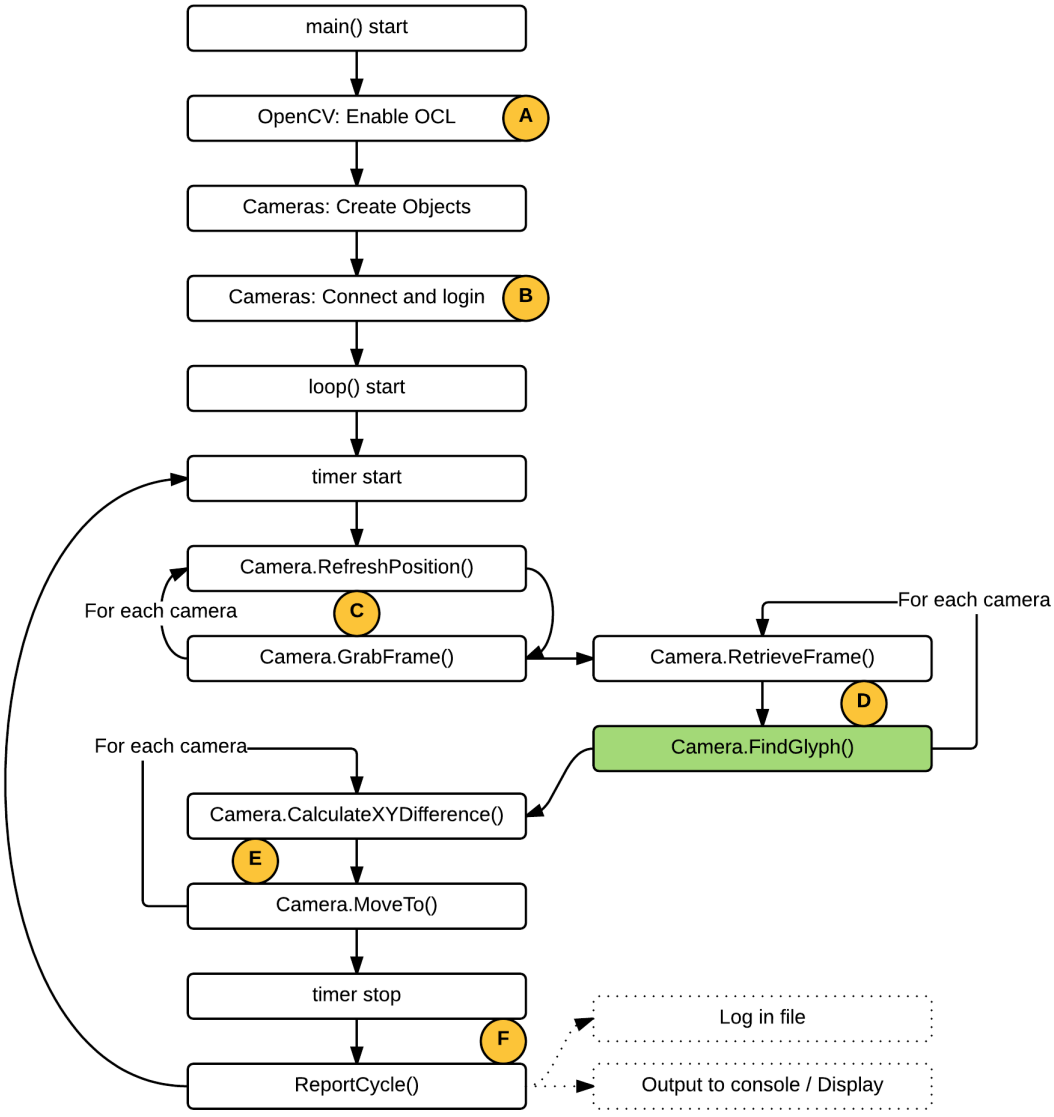

Figure 3.5: Software flowchart as implemented. Source: Own work

Step A

OpenCV 3.0 comes with a new transparent API, but we still have to enable the usage of OpenCL and verify that it is indeed working. This also gives us the opportunity to disable OpenCL to see how this affects the speed of our algorithm.

Step B

Each camera uses HTTP with Digest access authentication to prevent unauthorized access, which is an application of MD5 cryptographic hashing with nonce values to prevent replay attacks. The process of connecting to a camera and authenticating takes some time, so we do it once and increase the TCP parameters of the connection so that it is kept open. If we leave the TCP connection open, this should reduce the time needed for retrieving information in subsequent data transmissions. In order to create this connection, we query the camera for information while providing HTTP Digest information. The library we use is known as libcurl and more information about this open source project can be found at their webpage CURL (2015). Wireshark was used to find the authentication scheme.

Step C

Consider the loop in figure 3.6. Since we rely on receiving an image that is as up-to-date as possible, as well as the current camera parameters, we loop over each camera in our list. Camera.RefreshPosition() uses HTTP GET over the link previously established, in order to retrieve the following parameters, which were found using Wireshark:

- pan (float)
- tilt (float)
- zoom (integer)
- iris (integer)
- focus (integer)
- autofocus (on/off)
- autoiris (on/off)

Afterwards, we use OpenCV to grab the most recent frame from the MJPEG stream, and store this together with the most current timestamp. By using VideoCapture::grab(), we postpone decoding of the frame until a later stage. We do this to minimize the time-difference between two captured images from two different cameras.

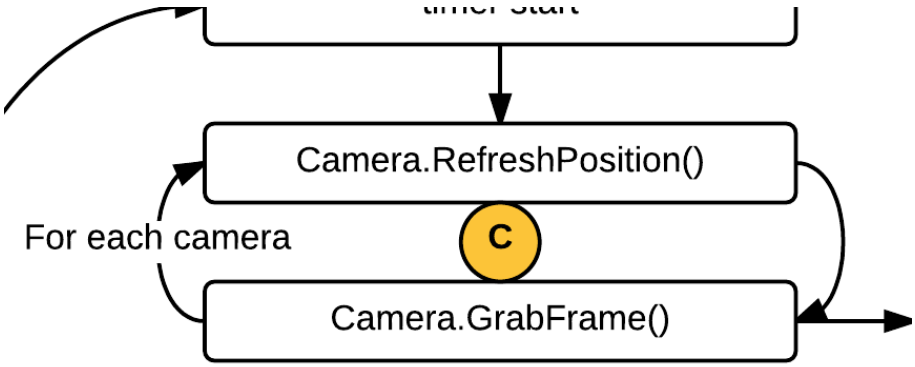

Figure 3.6: Part C of flowchart. Source: Own work

Step D

Consider the loop in figure 3.7. We have in the last step gathered both images, positions and timestamp from the cameras. Now, we let OpenCV decode the images and then we run the glyph finding algorithm on each of these.

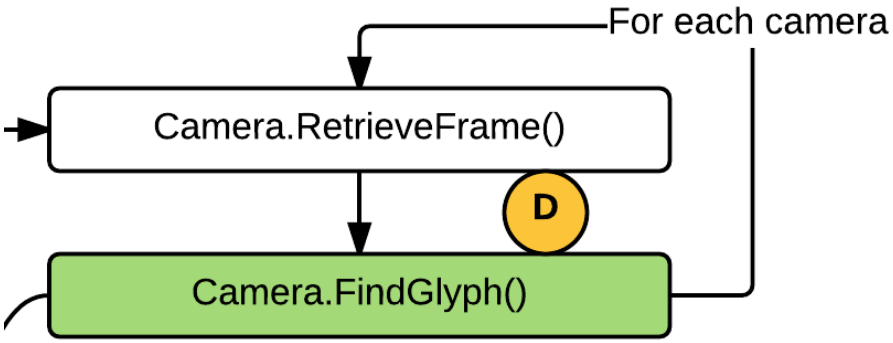

Figure 3.7: Part D of flowchart. Source: Own work

Step E

Consider the loop in figure 3.8. Using the center of the glyph distance from the target pixel on the image, we calculate the difference and move the camera towards this position. This step can be made more advanced by implementing various controllers, but for this thesis, it remains a Proportional controller. The Camera.MoveTo() method uses HTTP to engage the PTZ camera.

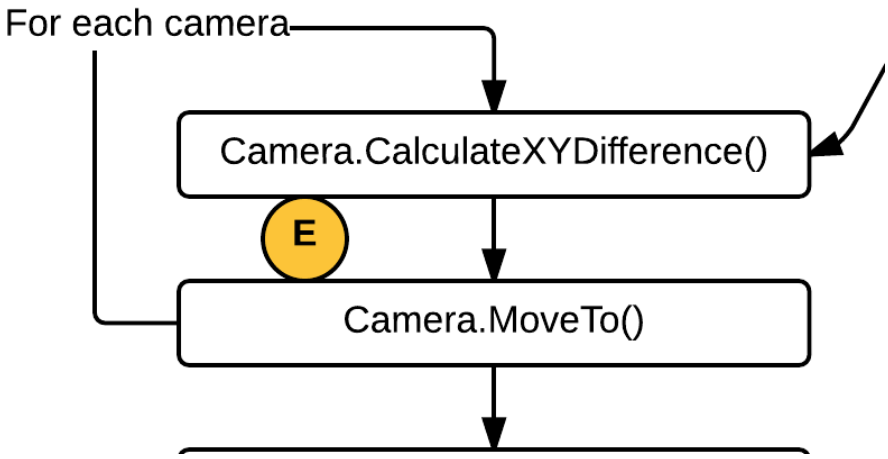

Figure 3.8: Part E of flowchart. Source: Own work

Step F

In 3.8, we are now at the end of a cycle. The time spent on this cycle as well as other useful data is logged to a file. The same information can be presented on an augmented reality screen. A new cycle begins.

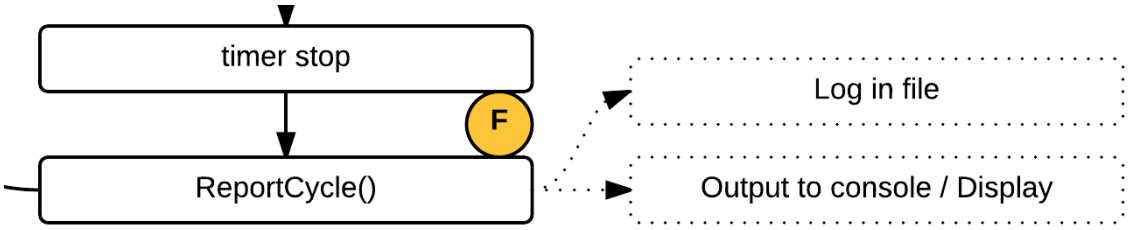

Figure 3.9: Part F of flowchart. Source: Own work

3.1.5 Analysis of dataset for robustness in weather conditions

Based on work done in the project thesis, a Python program using OpenCV 2 was created. Its purpose was to analyse pictures captured over the span of a year, from the MHWirth test tower located at Dvergsnes, Kristiansand.

$$58h08min18.3secN8h04min17.6secE \tag{3.1}$$

The dataset was gathered using another Python program written and deployed around christmas 2014, which at 15-minute intervals, captured images from a handful of CCTV cameras in the tower and stored them on a local server.

Located at various locations in the fifty meters tall test tower, the idea was that weather impacts would be strong.

As the CCTV cameras have built-in low-light mode, a period during the night is considered too dark for the algorithm to detect the glyph.

The program that gathered the pictures was running without supervision. A result of this was that some gaps of several days in the dataset exists.

Initially, all the pictures were stored as lossless 24-bit RGB Portable Network Graphics. However, after a few months, the amount of data being gathered was filling up the server storage. It was then decided to continue the data gathering using JPEG, which would cut the required storage space close to $\frac{1}{5}$ of the space required with PNG-24

$$\frac{pictures}{camera} = 365days \cdot 24\frac{h}{day} \cdot 4\frac{pics}{h} = 35040 \tag{3.2}$$

3.1.6 Tabletop setup for testing

In order to rapidly develop and test the software in a controlled environment, a tabletop setup was created. The main PTZ camera was connected to the internet, while the USB webcamera was connected to the Linux server.

A custom built linear actuator was used to provide repeated linear motion. See Appendix D page 49 for construction details.

A computer screen was used to roughly determine camera latency using a custom-built timer software. See figure 3.10 for the full setup. The timer in use can be seen on figure 3.11 where the camera looks at the screen which in turn displays the captured image. The difference between captured picture and displayed picture gives us a rough estimate of the latency we experience. In this case, we have close to 241 milliseconds of delay, nearly four frames per second.

$$9.274741s - 9.033639s = 0.241102s \tag{3.3}$$

In order to protect the privacy of other students in the room, the back of the PTZ camera was covered.

Figure 3.10: The tabletop setup. Source: Own work

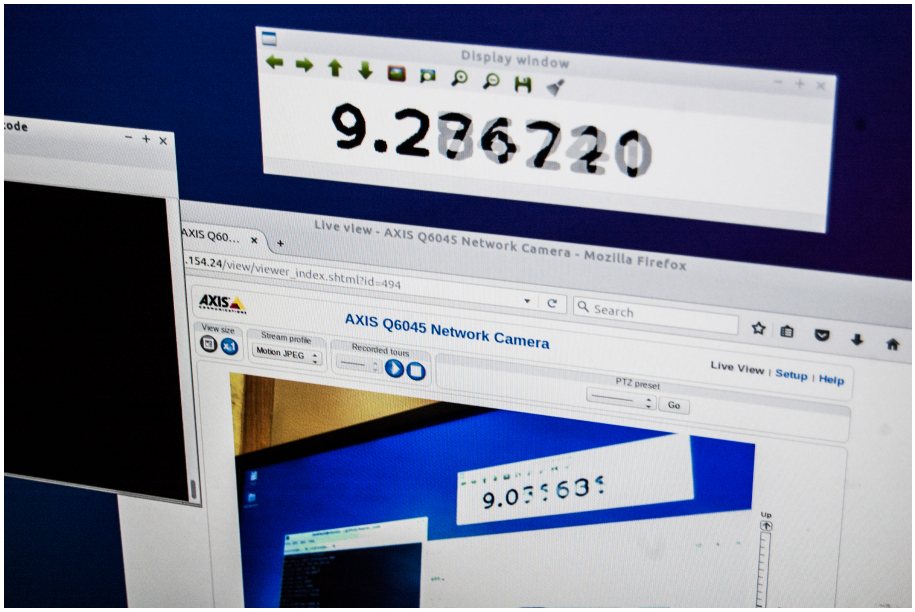

Figure 3.11: The timer software in use, showing 241 ms delay. Source: Own work

3.2 Tubular detection in fingerboard

3.2.1 Background

Drilling pipes and risers, commonly called tubulars, are used to drill deep into the earth. These tubulars need to be stored in between their use in the drilling process. The most common method for short-term to mid-term storage is in groups of a few units in a machine called fingerboard, a part of the pipe handling equipment on a drilling rig.

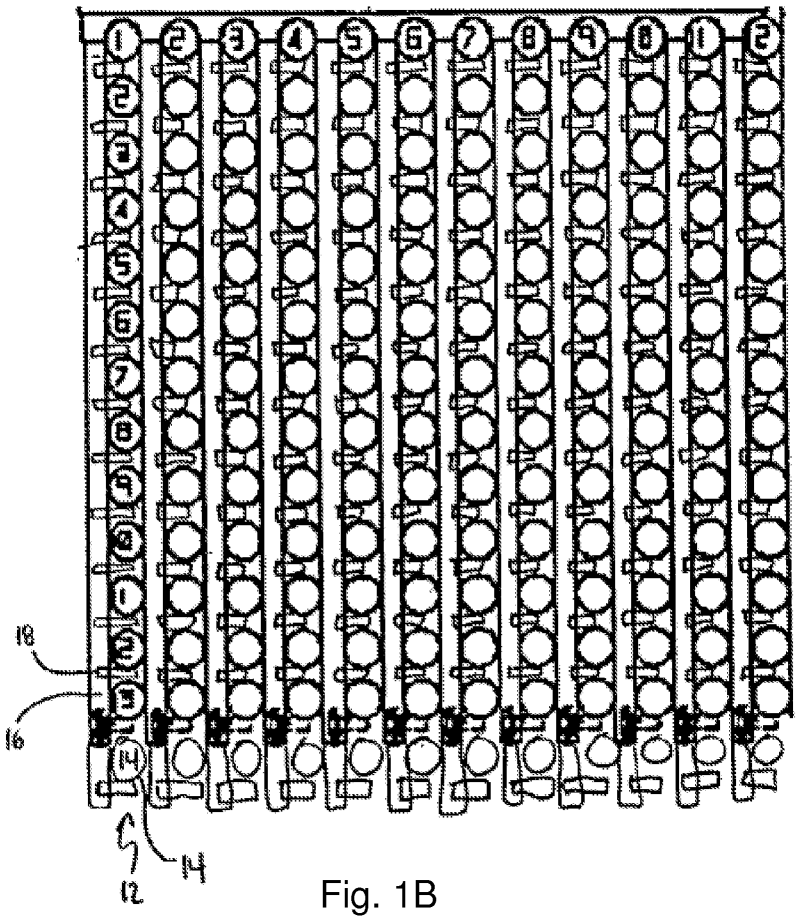

Figure 3.12: Diagram of a fingerboard with tubulars. Source:Braxton (2013a)

The first patent for a finger board was filed 1929 in the United States, and ever

since, new patents that increase their capabilities have been filed. The latest ones use pneumatic cylinders to latch and secure tubulars in place, and a control system managing and tracking the pipes that should be in the fingerboard.

For more information regarding the history of the fingerboard, the reference list of a patent publicized February 2013 by Braxton (2013b) is suggested. The patent in question discuss a method to report finger position data to the control system. With this feature, the control system becomes aware of its fingers, but knowing if a tubular exists in a given cell requires a fail-free logging of the actions done by other machines.

Figure 3.13: Fingerboard without tubulars. Source:TSC (2015)

3.2.2 Goal of case study

The goal of this case study is to draft an algorithm that can detect tubulars and associated parameters in a fingerboard. It is possible to develop the algorithm using more user-friendly click-and-drop vision packages, but the author settled on using OpenCV 3 because this does not require expensive and resource demanding software.

3.2.3 Design of the machine vision algorithm

The circularity of a circle can be described using the Heywood circularity factor. This factor can be used to narrow feature detection.

One possible solution involves using the Hough transform method to find ellipses in the picture, then analyze the region contained within to determine wheter the detected ellipse may outline a tubular. The algorithm implemented in OpenCV is based on a variant called the 2-1 Hough Transform by Yuen et al. (1990). Partial occlusion of tubulars are a challenge.

Another more recent method of ellipse detection is presented by Wang et al. (2014) based on sorted merging. This algorithm is not yet implemented in OpenCV, but may be better at handling partial occlusion of the ellipses, and faster than the Hough transform based methods.

Figure 3.14: Fingerboard with tubulars. Source: Own work (MHWirth AS)

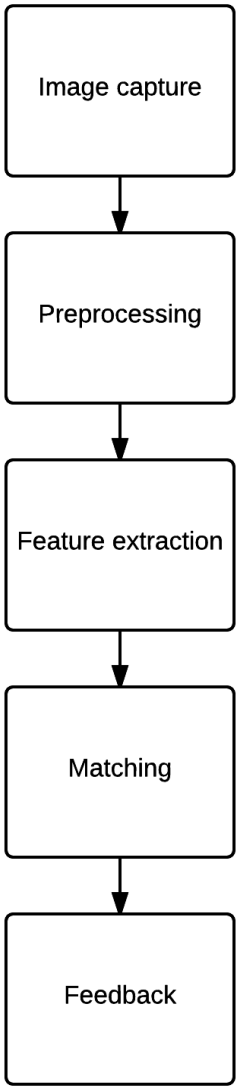

Figure 3.15: A generic machine vision algorithm before detailed description. Source: Own work

We will now build up an algorithm as a hypothetical solution to the challenge, and the design choices made will be explained.

Image capture

The image is captured and stored with colors from the camera. In our case, we use the high-resolution AXIS Q6045 footage captured outdoors at MHWirth AS. For an actual implementation, any camera at a certain angle and position above the fingerboard will do. We are depending upon good lightning and decent weather, and we assume that no water droplets are found on the camera housing.

For the sake of comparison, a matrix that shows some mid-winter lightning conditions and how it affects the image, which in turn may break our algorithm. See figure 3.16. The author's comments on this figure follows.

06:45, only artificial light can be seen. The tubulars and the fingerboard are black, no features besides their orientation and if lucky, width of the pipe, can be extracted.

08:15, we can see the fingerboard and the tubulars, but the picture is filled with artifacts that appear in low-light conditions. It should be possible to extract features from this image, but it is sub-optimal.

10:00, the natural light from the sun is flat and thus, the image does not give us too much contrasts. The picture in itself should suffice to detect the tubulars.

13:15, the sun has now risen close to its max elevation above the horizon for the current date. A good amount of contrasts in the image makes this a good candidate for implementing a proof-of-concept algorithm.

17:00, colors captured have shifted to a cold blue, the sun is almost gone. If we use colors to distinguish the tubular, this would provide a challenge, unless we adjust the white-balance.

17:15, the camera went into low-light mode, which removes the IR-filter. Any artificial light is also turned off at this time, leading to a dark muddy and noisy picture that is not good for our use.

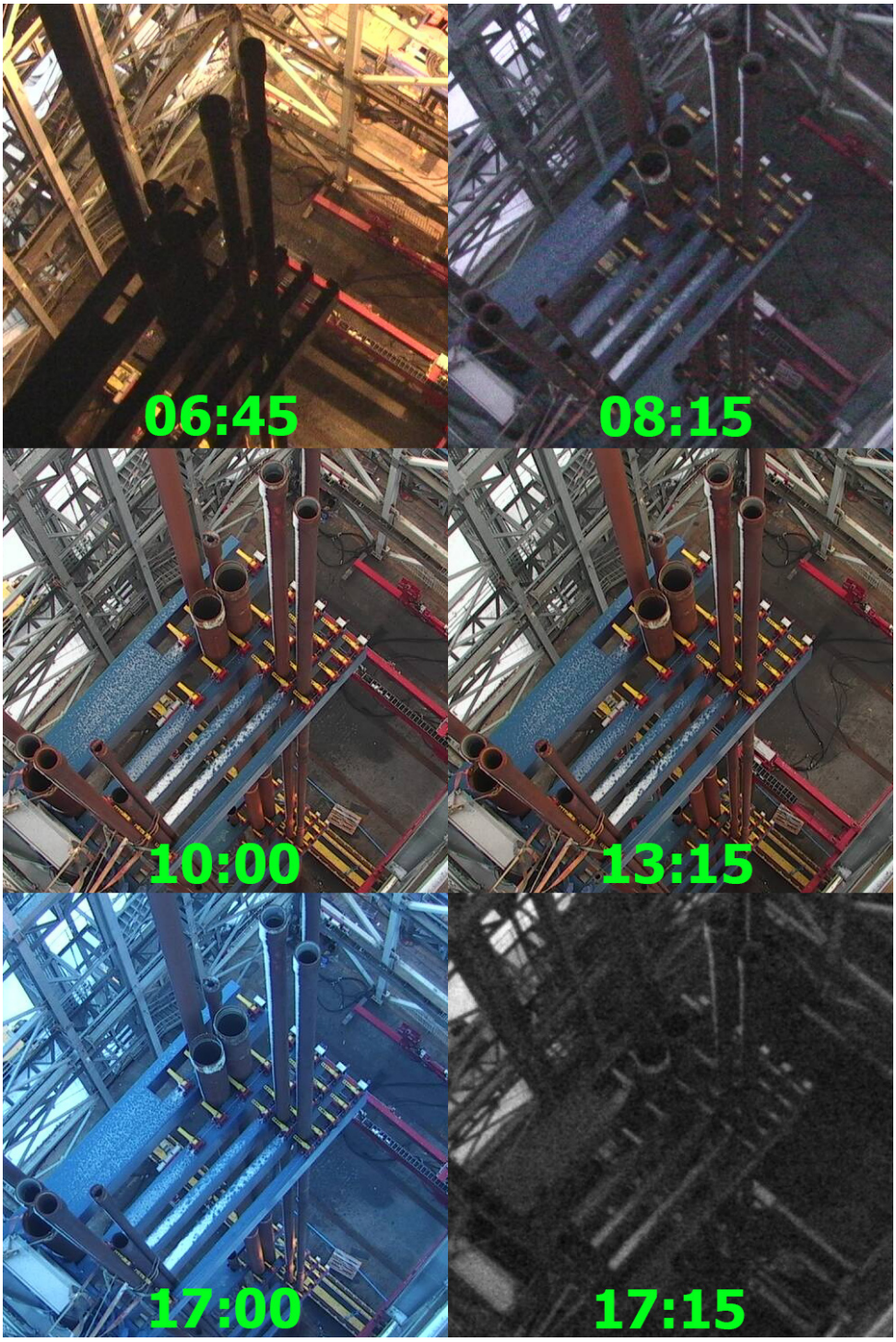

Figure 3.16: Matrix of CCTV images to show differences in lightning. All times in UTC+1, captured 25th of January 2015. Source: Own work

Chapter 4

Discussion and results

4.1 Discussion

4.2 Recommendations for future work

4.2.1 Multithreading

If a function blocks the execution of our single thread, the whole program runs slower, which in turn leads to an increased delay in the control system and may make the auto tracking CCTV system unstable.

By separating functionality into threads that run by themselves and communicate using ZeroMQ or similar software library, a temporary latency spike in the communication between a camera and the software will not slow down the rest of the system.

In order to implement this fully, great care needs to be taken to ensure that either we wait for all data to be present before we act upon it, or we discard it if it gets delayed too much to ensure a near-realtime behavior.

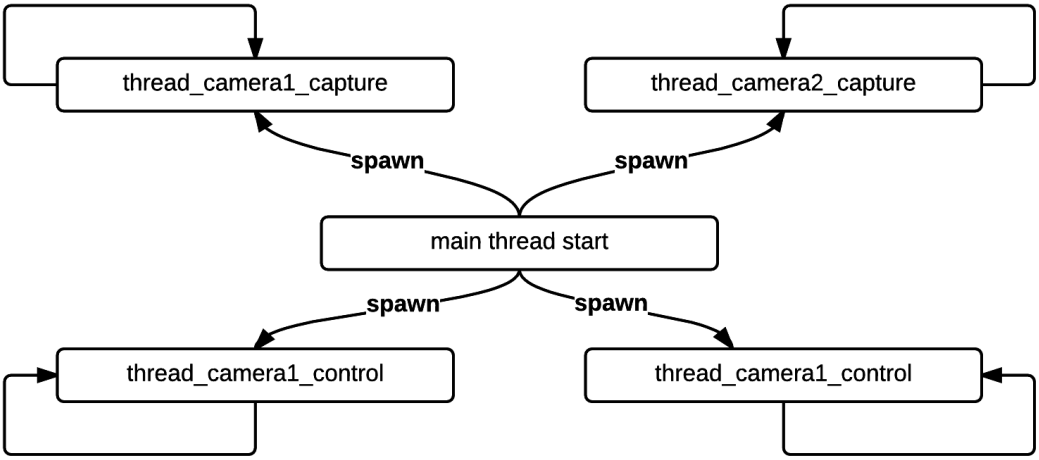

Figure 4.1: Multithreading concept runs camera capture independent of camera PTZ control. Source: Own work

4.2.2 Augmented Reality

In order to provide the driller and assistant driller with valuable information, a heads-up display could be used. This could use the video stream from the auto tracking CCTV cameras to overlay useful information depending on where in a sequence the machines are. By augmenting the image with data from both the control system as well as data interpreted from the machine vision system, we may also use data fusion to determine how accurate the displayed data is, and output the best result possible.

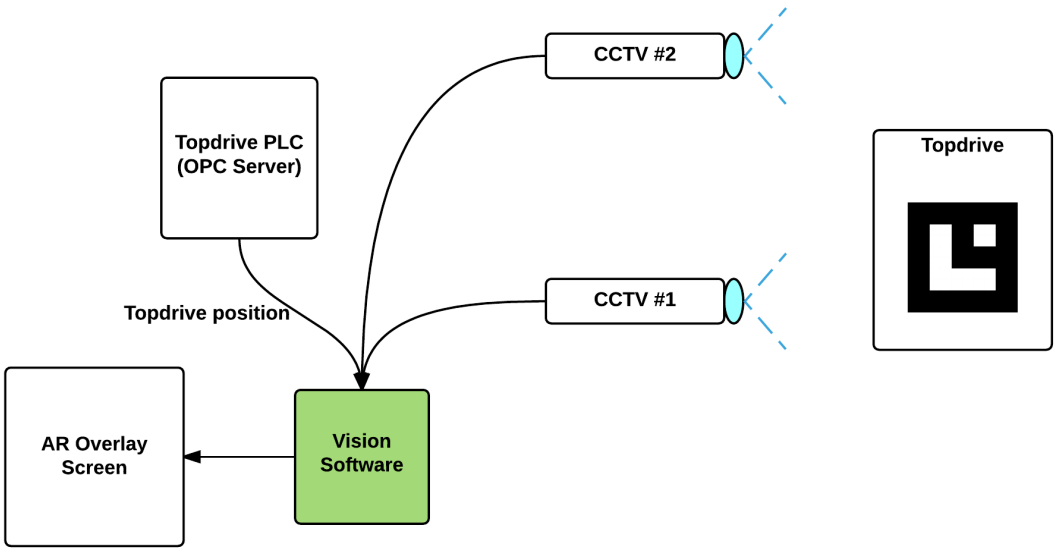

Figure 4.2: Data fusion using PLC and vision data. Source: Own work

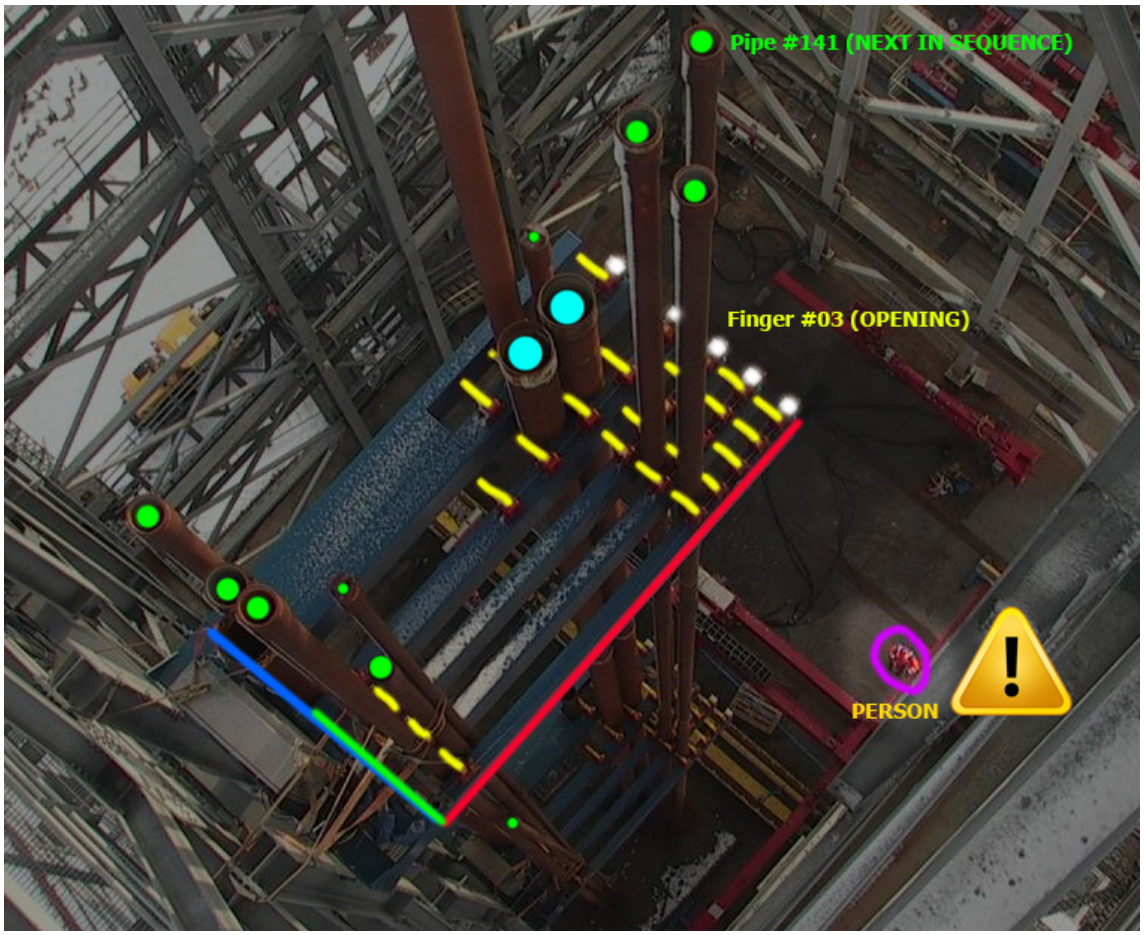

Figure 4.3: Fingerboard augmented reality mockup. Source: Own work

Figure 4.4: Topdrive augmented reality mockup. Source: Drillingcontractor (2015) modified by author

Appendix A

Acronyms

API Application Programming Interface

AR Augmented Reality

CD Compact Disc

CGI Common Gateway Interface

CV Computer Vision

IADC International Association Of Drilling Contractors

MJPEG Motion JPEG

OPC UA Open Platform Communications Unified Architecture

PTZ Pan Tilt Zoom

SOAP Simple Object Access protocol

SSL Secure Socket Layer

WITSML Wellsite Information Transfer Standard Markup Language

Appendix B

Axis Q6045 Camera

Introduction The Axis Q6045 PTZ Dome Network Camera is a high-performance IP-camera which can deliver its video stream over a computer network. It is delivered in both indoor and outdoor models, and have a 20x optical zoom. Endless 360 degree pan and a high resolution.

Outdoor edition The outdoor edition also supports electronic image stabilization to reduce effects of camera vibration. It also includes automatic defogging capabilities using digital filters.

Camera

Image Sensor	1/3” progressive scan CMOS
Lens	f=4.45 - 89 mm, F1.6 - 2.9, autofocus Horizontal angle of view: 62.98 deg - 3.49 deg
Min. illumination	Color: 0.6 lux at 30 IRE F1.6 B/W: 0.04 lux at 30 IRE F1.6
Shutter time	1/33000 s to 1/3 s (50 Hz), 1/33000 s to 1/4 s (60 Hz)
Pan/Tilt/Zoom	Pan: 360 degrees endless, 0.05 - 450 degree/s Tilt: 180 degree, 0.05 - 450 degree/s 20x optical zoom and 12x digital zoom, total 240x zoom

Video

Video compression	H.264 Main and Baseline Profiles Motion JPEG
Resolutions	1920x1080 to 320x180
Frame rate	H.264: Up to 25/30 fps (50/60 Hz) in all resolutions Motion JPEG: Up to 25/30 fps (50/60 Hz) in all resolutions

System integration

API	VAPIX AVHS ONVIF Profile S

General

Operating conditions	0 degree to 50 degree Celsius Humidity 10-85% RH (non-condensing)

Appendix C

Python Profiling

Introduction It is well known that Python as an interpreted language is considerably slower than both compiled C and C++ code. However, the Python module that provides our OpenCV-interface is not much more than a wrapper for the C++ core in OpenCV.

As such, we use Python for quickly developing and structuring the program, while the actual heavy lifting is done by compiled C++ code. Our design goal is then to write efficient Python and using quick algorithms where possible, leveraging the built-in functionality in the C++ core of OpenCV.

In order to investigate both time and memory used for our solution, a few good methods exists.

line_profiler A module for Python that provides line by line profiling capabilities is the line_profiler program. The actual time spent inside a function is outputted.

We can install the module through pip.

```
pip install line_profiler
```

After adding the keyword profile ahead of each function we would like to profile, we run the profiler as kernprof.py through Python.

```
python kernprof.py -l -v program.py
```

By running line_profiler on the initial proof of concept, we will get the time spent on each function we used @profile in front.

```
Total time: 0.475104 s
File: jsg.py
Function: find_potential_glyphs at line 90

Total time: 0.116176 s
File: main.py
```

```
Function: init_capture_device at line 18

Total time: 2.35552 s
File: main.py
Function: grab_frame at line 29

Total time: 5.41517 s
File: ptz.py
Function: execute_command at line 25
```

This tells us that through the running time of the program, most time was spent inside ptz.py.

We can for example, see the following output for code from ptz.py, the file that sends ptz commands to the camera.

We see that the function execute_command contains, at line 30, a http request which consumes 99.7% of the time used inside the function. Out of this, it is suggested that the overhead by using the CGI element for doing PTZ is a function of network latency.

memory_profiler Another important resource is the amount of memory being used by the program. Using this, we can discover memory leaks and further optimize our code. We can install the module through pip. We are also adviced to install psutil, as it improves performance of memory_profiler.

```
pip install memory_profiler
pip install psutil
```

Appendix D

Building a Linear Actuator

Introduction In order to test the glyph tracking system with repeatable motion, a linear motion system was needed. After trying a very powerful electronic linear actuator originally used for antenna movement, it was concluded to be too slow for any practical use in testing the glyph tracking system. In the matter of an evening, a simple, remotely operated and fully functional linear actuator was built, and its build procedure follows.

Parts required A handful of items are required to build the linear actuator, most of which can be found lying around a workshop or in your desk drawer.

- 1x Microcontroller (Arduino Uno)
- 1x H-bridge motor controller (SN754410)
- 1x DC motor with pulley
- 1x Free pulley on rod
- Some prototyping wires
- Some wood boards
- Some acrylic plates
- A piece of thread in a loop
- Hot glue gun
- Ducttape and screws

Build procedure

1. The wood board is cut to wished length and a base is built using screws.
2. Free pulley and DC motor with pulley are glued/drilled in place on the board.
3. Electronic circuit is put together with H-bridge getting power from a stand-alone supply. We use an extra USB-port which can deliver up to 500 mA 5 volt.
4. The Arduino Uno is programmed to trigger the H-bridge depending on input commands from the serial port.
5. The thread is tightened until friction makes it move with load.
6. A piece of thread is attached, that holds the payload. In our case, a laminated glyph symbol.
7. Acrylic plates are bent to protect the pulley and thread path. Use a propane burner or other gas burner to heat up the plastic in order to bend it.
8. Two thin pieces of wood can help in giving the glyph a smooth path to move along, and these are screwed onto the main board if needed.
9. Ducttape and hot glue gun. The ducttape is used for construction reinforcement as well as reduction of friction so that the glyph can be pulled by the weak motor. The glue gun is used to fix some wires and further reinforce joints.

The bottom assembly with electronics can be viewed in figure D.1. This includes an acrylic bend to hold the glyph as it rests on the bottom. The electronics are rapidly put together and glued in place where applicable. The DC motor is hiding behind the duct tape. Two sets of cables are exiting the frame on the left side, one is the USB UART cable that also powers the Arduino Uno, while the other one is a pair of red-black wire that carries power from another USB port, which is used by the DC motor.

The top assembly and slider can be viewed in figure D.2. The duct tape is used to reduce friction on the wood, so that the laminated glyph can translate without problems.

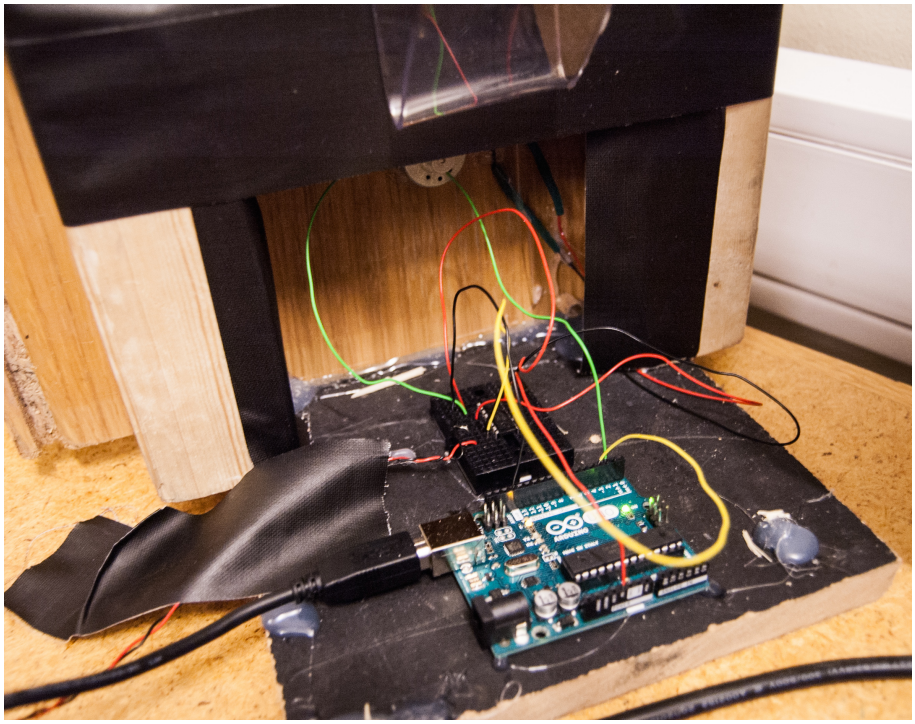

Figure D.1: Bottom assembly. Source: Own work

Figure D.2: Top assembly and main slider. Source: Own work

Operation Since we are using a micro-controller board with built-in support for UART over USB, we can send commands to the linear actuator. By remotely connecting to the Linux server using SSH, we are able to test the system from anywhere in the world. See figure D.3 for full communication channel.

The DC motor is too powerful to be run from the Arduino Uno 5 volt rail, it is advisable to use another power source, which feeds the H-bridge.

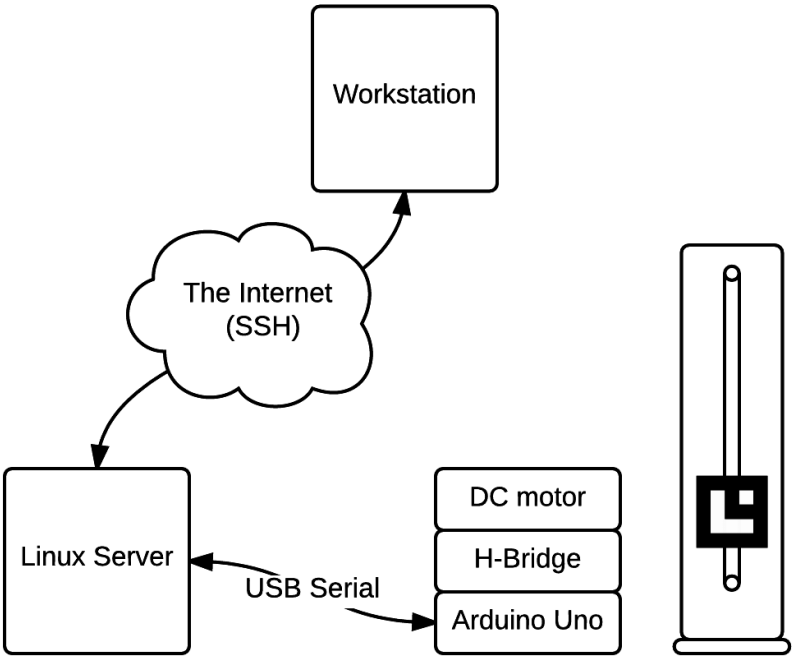

Figure D.3: Diagram over communication channel and system. Source: Own work

Appendix E

Compiling OpenCV 3.0 for OS X

OpenCV 3.0 gold release arrived in June 2015. Being this new, the resources to properly compile OpenCV 3.0 for OS X are not easy to come by. A few blogs have a partial solution for setting up the software, so this appendix will not contain a complete solution itself, but mention the most important points when building OpenCV 3.0 for OS X.

Homebrew is utilized to settle most build-dependencies, with good success.

When building opencv, we also would be interested in building opencv_contrib modules.

```
cd ~
git clone https://github.com/Itseez/opencv.git
git clone https://github.com/Itseez/opencv_contrib.git
```

We use the command-line interface for cmake, but it should also be possible to use the GUI version. Notice that we explicitly disable CUDA in this case, and enable OpenCL. We also build with support for Python 2.7, as the author considers this a good way to rapidly test functionality.

```
cmake -D CMAKE_BUILD_TYPE=RELEASE \
-D CMAKE_INSTALL_PREFIX=/usr/local \
-D PYTHON2_PACKAGES_PATH=~/.virtualenvs/cv/lib/python2.7/site-packages \
-D PYTHON2_LIBRARY=/usr/local/Cellar/python/2.7.10/Frameworks/Python.framework/Versions/2
-D PYTHON2_INCLUDE_DIR=/usr/local/Frameworks/Python.framework/Headers \
-D INSTALL_C_EXAMPLES=ON \
-D INSTALL_PYTHON_EXAMPLES=ON \
-D BUILD_EXAMPLES=ON \
-D WITH_OPENCL=ON \
-D WITH_CUDA=OFF \
-D WITH_FFMPEG=ON \
-D BUILD_DOCS=ON \
```

```
-D OPENCV_EXTRA_MODULES_PATH=~/opencv_contrib/modules ..
```

We have not installed clBLAS nor clFFT, two libraries that are considered advantageous when using OpenCL. These libraries will have to be compiled with their own specific dependencies.

Bibliography

Photokonnexion. (2015) Photokonnexion. [Online]. Available: http://farm8.staticflickr.com/7222/7228218072_042541190a.jpg

A. Communications. (2015) White paper: P-iris. [Online]. Available: http://www.axis.com/files/whitepaper/wp_p-iris_38023_en_1009_hi.pdf

L. Svensson and P. Söderlund, "Delays in axis ip surveillance cameras," 2013.

VRay. (2015) Vray. [Online]. Available: https://www.vray.com/vray_for_sketchup/manual/vray_for_sketchup_manual/other_parameters_within_the_reflection_layer/reflection_layer_in_vray_for_sketchup_4.png

Codebase. (2015) Creating multi-threaded c++ code. [Online]. Available: http://codebase.eu/tutorial/posix-threads-c/

B. D. Technology. (2013) Opencl tutorial: N-body simulation. [Online]. Available: http://www.browndeertechnology.com/docs/BDT_OpenCL_Tutorial_NBody-rev3.html

Wikipedia. (2015) Wikipedia: Cuda. [Online]. Available: https://en.wikipedia.org/wiki/CUDA

OpenCV.org. (2015) Cuda platform. [Online]. Available: http://opencv.org/platforms/cuda.html

J. Skjefstad, "Computer vision in drilling vessel applications," 2014.

J. R. Braxton. (2013) Offshore drilling rig fingerboard latch position indication. [Online]. Available: http://www.google.com/patents/US20130032405

TSC. (2015) Fingerboard without tubulars. [Online]. Available: http://www.t-s-c.com/upload/mediawindow/2014-12/m_p198p86l291a875cce2e6og7o55.JPG

Drillingcontractor. (2015) Drillingcontractor topdrive. [Online]. Available: http://www.drillingcontractor.org/wp-content/uploads/2012/01/webDSC03307.jpg

S. Sklet, “Safety barriers on oil and gas platforms,” Ph.D. dissertation, Norwegian University of Science and Technology, 2005.

M. Rausand, *Reliability of Safety-Critical Systems: Theory and Applications*. Hoboken, NJ: Wiley, 2014.

N. Dadashi, A. W. Stedmon, and T. P. Pridemore, “Semi-automated cctv surveillance: The effects of system confidence, system accuracy and task complexity on operator vigilance, reliance and workload,” 2012.

O. H. Boyers, “An evaluation of detection and recognition algorithms to implement autonomous target tracking with a quadrotor,” 2013.

A. Kirillov. (2010) Glyphs recognition. [Online]. Available: http://www.aforgenet.com/articles/glyph_recognition/

A. Communications. (2015) Latency in live network video surveillance. [Online]. Available: http://www.axis.com/files/whitepaper/wp_latency_live_netvid_63380_external_en_1504_lo.pdf

R. Hill, C. Madden, A. van den Hengel, H. Detmold, and A. Dick, “Measuring latency for video surveillance systems,” 2010.

Tracking Across Multiple Cameras With Disjoint Views, 2003.

J. Skjefstad. (2015) Github. [Online]. Available: www.google.com

Imatest. (2015) Imatest image quality. [Online]. Available: http://www.imatest.com/support/image-quality/

J. Fang, A. L. Varbanescu, and H. Sips, “A comprehensive performance comparison of cuda and opencl,” 2011.

Wikipedia. (2015) Wikipedia: Linux. [Online]. Available: https://en.wikipedia.org/wiki/Linux

L. Torvalds. (2015) Github: Linux kernel. [Online]. Available: https://github.com/torvalds/linux

CURL. (2015) Curl. [Online]. Available: http://curl.haxx.se

J. R. Braxton. (2013) Offshore drilling rig fingerboard latch position indication. [Online]. Available: http://www.google.com/patents/US20130032405

H. K. Yuen, J. Princen, J. Illingworth, and J. Kittler, “Comparative study of hough transform methods for circle finding,” 1990.

G. Wang, G. Ren, Z. Wu, Y. Zhao, and L. Jiang, "A fast and robust ellipse-detection method based on sorted merging," 2014.