Realtime auto tracking CCTV cameras for increased safety and productivity

Joakim Skjefstad

Master of Science in Cybernetics and Robotics

Submission date:	December 2015
Supervisor:	Tor Engebret Onshus, ITK
Co-supervisor:	Mads Hvilshøj, MHWirth AS
	Thor Eivind Brantzeg, MHWirth AS
	Johan Hansen, MHWirth AS

Norwegian University of Science and Technology
Department of Engineering Cybernetics

NTNU – Trondheim
Norwegian University of
Science and Technology

Realtime auto tracking CCTV cameras for increased safety and productivity

Joakim Skjefstad

17-12-2015

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

Abstract

This paper describes the development of machine vision implementations to be used on an offshore oil rig in order to track heavy machinery in motion with the CCTV camera system and verify tubulars stored in a fingerboard. A version of the machine tracking software has been implemented in C++11 which attempts to utilize OpenCL to increase its speed. A proof of concept implementation for detecting tubulars have been implemented in Python. Both cases utilize the OpenCV machine vision library. Based on the findings of the study, the possibility of both heavy machine tracking and tubular detection in fingerboard is verified, but the study is inconclusive on the effect of OpenCL for speedup of machine vision algorithms and further research has to be done. Improvement of the tubular detection implementation is also required.

Abstract in Norwegian

Denne oppgaven beskriver utviklingen av en maskinsyn-implementasjon som kan bli brukt på offshore oljeplatformer for å følge tungt maskineri med CCTV kamera systemet og bekrefte rør som er lagret i ett fingerbord. En versjon av systemet for å følge tungt maskineri har blitt utviklet i C++11, som forsøker å benytte OpenCL for å øke programhastigheten. Et forsøk på å implementere rør har blitt laget i Python. Begge programmene benytter OpenCV maskinsyn-biblioteket. Basert på funn i oppgaven, kan man konkludere med at det er mulig å både følge maskineri og finne rør i fingerbord, men oppgaven kan ikke konkludere med effekten av OpenCL for å øke hastigheten på maskinsyn-algoritmer, og mer forskning må gjøres på det området. Forbedringer på rør-deteksjons-implementasjonen gjenstår også.

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-supervisors Doctor Mads Hvilshøj and Thor Eivind C. Brantzeg from MHWirth AS for their guidance, domain knowledge and support throughout the project.

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

Summary

This thesis concerns the use of machine vision in order to increase safety and performance around the drillfloor on an oil rig. Two cases have been explored. The first case is an auto tracking CCTV system that allows the human supervisors to always see machines as they move around on the drillfloor. The second case is a proof of concept implementation of a system that can detect tubulars standing in fingerboards, in order to provide data verification for a control system.

In order to explore these two cases, software was developed in both C++11 and Python, and a dataset gathered from a test tower was used, together with a tabletop setup which involves a linear actuator that can translate glyph symbols. OpenCL was explored as a way to speed up machine vision for tracking the glyph. A high-performance CCTV dome camera was used to follow the glyph, connected to a machine running Linux and the software developed.

The conclusion of this research is that auto tracking CCTV cameras are preferably done in C++11 when compared to Python, but further research is needed to fully understand and utilize the OpenCV T-API interface for speeding up machine vision algorithms. Detection of tubulars in fingerboards is also working at a rudimentary level with the implementation, and further work has to be done on analysis of the features extracted by the implementation. Multiple camera sources have been tested for the tracking CCTV camera, but not for the tubular detection implementation.

Contents

List of Figures

Chapter 1

Introduction

1.1 Background

The Norwegian economy is cooling under a pessimistic oil price of close to 37 USD per barrel as of December 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, real-time decision support systems have been developed. One example of these was a product of work done at NTNU, by Verdande Technology, established 2004. The company is now closed down after becoming bankrupt in December 2014 as potential customers were cutting costs.

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 implementation 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?

How can we identify tubulars in a fingerboard using machine vision, so that the control system can verify its data?

The end goal is to reduce the possibility of an undesired event, be it either damage to humans or expensive equipment, and increase speed of operations.

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.

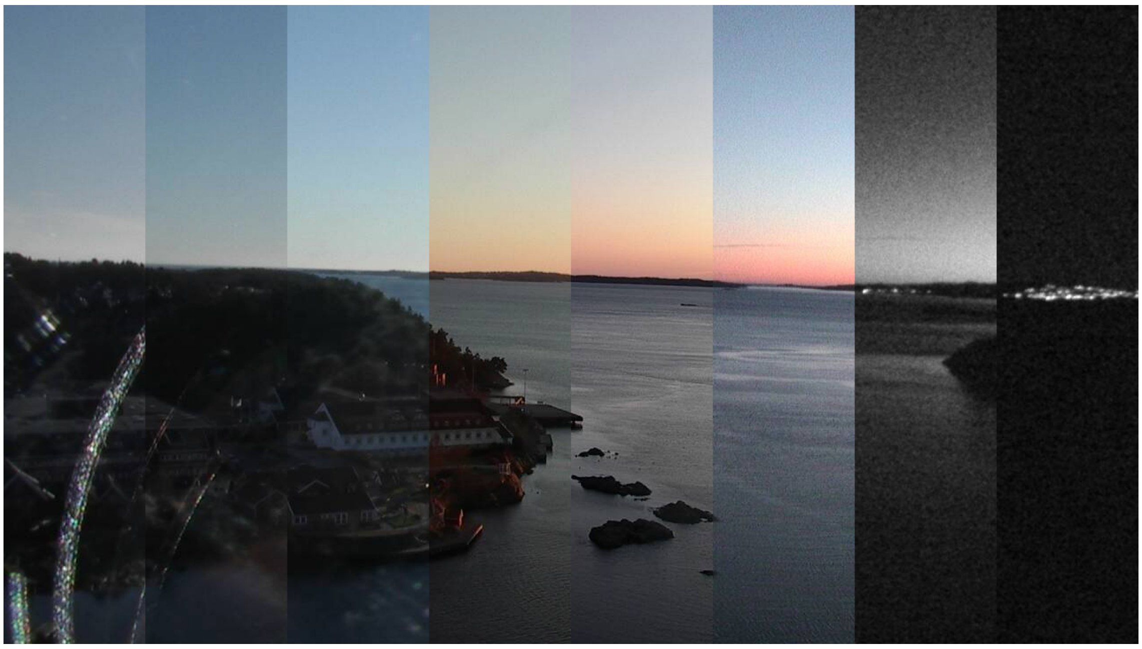

Figure 1.1: Timelapse through a day of a CCTV camera in Kristiansand, 4th of March 2015. Source: Own work

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, 2015)

The inherit differences between analog and digital IP[1] transmission of video have been examined by Hill et al. (2009), and the conclusion was that latency present in digital IP transmission is within acceptable values 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 latency was measured to be between 120 ms up to 1600 ms depending on the resolution of the image and its compression. The upsides of digital transmission include increased quality of image, flexibility of digital encoding and ability to use analytic software. The paper does not describe other forms of digital video transmissions that exist, including raw digital transmission using SDI[2], and it is also outdated in terms of the current state of the art cameras

[1]IP, the short form of Internet Protocol, a principal communications protocol for relaying datagrams across network boundaries. IPv4 was deployed in the ARPANET in 1983.

[2]Serial Digital Interface, high-speed digital video transmission defined by Society of Motion Picture & Television Engineers.

available, but it provides a reference.

Work done by Svensson and Söderlund (2013) involved methods to reduce the delays that are inherently found in digital IP 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 operating system 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 operating system 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. When images from several sources are used, the time-synchronization of 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.

The implementation and comparison of a robust and responsive automated CCTV tracking system remains to be done, as an earlier proof of concept implementation was done by the author in 2014.

Not much literature on fingerboard tubular detection have been found. It is believed that the reason for this is that the word fingerboard is a technical jargon used in the drilling industry, and that any research done is done in-house with no intention of publishing results.

The implementation of a simple fingerboard tubular detection program remains to be done, and the analysis of its performance.

We aim to combine as much as possible from the various fields, given constraints of time, into a proof of concept which can be the stepping stone to a commercial product in the case of CCTV tracking, and a prototype implementation for the fingerboard tubular detection.

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 glyph symbol, and comparison of its performance

2. Implementation of a proof of concept tubular-detection program for fingerboards

1.3 Limitations

The limitations that relates to this study includes both technical challenges, environmental and operational conditions, 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 resolutions 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, commonly using IP, 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. Packet loss is a real possibility in IP networks, 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 use of raw digital transmission such as SDI would minimize latency and give room for even higher resolution images with little to no chance of data loss, but this has not been explored as it requires specialized cameras, coaxial cables and frame grabbers.

Technology is highly guarded, and real offshore operations are not easy to get access to. Cooperation is not common in the industry, and transparency of systems and solutions may be less than ideal. This limits the available data set for the purpose of research and making robust systems.

The software world progress quickly, and new solutions can suddenly become obsolete. The technological debt increases quickly. 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.

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 become a successful implementation.

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 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 intends 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 implementation 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.

2.0.1 Camera

The camera specifications and performance directly affects the results. It is the source of input data for the machine vision software, and it may also be the output, which will be the case when the PTZ[1] is controlled by the software.

Some cameras support extra applications that run in their operating system, 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. Embedded systems which can be found in commercial CCTV cameras are commonly resource-constrained in terms of CPU and RAM, and this suggests that some complex software applications should be 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.

[1]PTZ, pan- tilt and zoom functionality to translate the image captured by a camera. Common in dome-type CCTV cameras.

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.

Aperture

The aperture is the opening in a camera objective which determines how much light enters the camera, and it is also a factor that can determine the focus range of a camera. An open aperture leads to larger amounts of light and thus works better in low-light conditions, but the depth of field gets smaller. See figure **??** for how depth of field relates to the aperture size. The amount of light that enters and hits the image sensor also determines how long the aperture 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 aperture control, that will try to adjust the aperture opening and shutter speed so that the picture does not get too bright or too dark. Axis Communications have introduced a more precise aperture control to some cameras, which they call "P-iris", claimed to provide improvements to contrast, clarity, resolution and depth of field beyond other methods of aperture control.

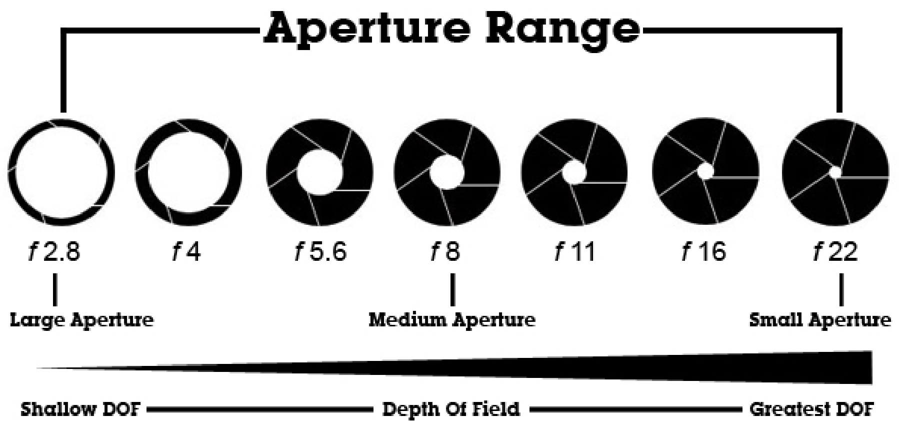

Figure 2.1: Aperture versus depth of field. Source: Online Tutsplus.com (2015)

Frame Rate

The frequency of consecutive image frames is expressed in frames per second. Video displays motion through the constant change of image frames, and the human visual system perceives this as motion. Capturing at a given frame rate requires the aperture, image sensor and the operating system of a digital camera to perform the job of capturing, processing and transmitting at the rate required. A large amount of frames per second requires a multiple of the image resolution of bandwidth to transfer the images to a viewer, which means that the most direct way of reducing data transmission requirements is by reducing frames per second. When it comes to CCTV cameras for surveillance, frame rate of stored movies are reduced to allow for longer storage of the video material. Higher frame rate gives perceived smoother motion in the video, and it also allows us to slow down actions and investigate them in detail. The frame rate of a normal CCTV camera may be 25 FPS, a high-definition movie may be 60 FPS while high-speed cameras capture several thousands of frames per second.

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 any video codec used to compress the image, 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, and the focus can be to reduce cost or increase performance. The optimal interface depends on the application. Some of the most used interfaces today includes USB, SDI 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.

2.0.2 Data transmission

A function of the camera hardware and operating system, 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. The way a video signal propagates from one place to another makes this an important parameter in the selection of a camera system. At each end of the transmission line, compression and decompression may take place.

Analog Analog video transmission relies on a continuous voltage range. A weak signal is susceptible to electronic interference. The analog signal is modulated on top of a carrier frequency using RF modulation, which is then transferred over a physical medium. The medium have major implications on the amount of interference the signal picks up. Coaxial cables are just one of the many mediums that can be used, including broadcasting over a VHF[2] or UHF[3] carrier.

Digital - SDI family A family of serial digital interface standards defined by the SMPTE in 1989 (Poynton, 2003) is another method to transfer video. This family of standard is not widely used in consumer electronics, as licensing agreements restricts its use. Usually used to transfer uncompressed and unencrypted digital video signals, they can also be used for packetized data. The physical medium can be of copper coaxial cables with BNC connectors and 75 ohms impedance for a max run length of typically 100 meters when used with HD video, or it can be fiber optics only limited by the maximum fiber length and repeaters. The quality of cabling and termination is important to ensure max range. Wikipedia (2015c)

SMPTE 292M is one of these standards, common name is HD-SDI and it was introduced in 1998. Its theoretical bitrate is 1.485 Gigabit per second, and it can

[2]VHF, Very high frequency is a designation for electromagnetic waves in the range from 30 MHz to 300 MHz.

[3]UHF, Ultra high frequency is a designation for electromagnetic waves in the range from 300 MHz to 3 GHz.

transfer up to 1080i images through coaxial cables.

Since SDI uses a dedicated coaxial cable, there is no network congestion.

Digital - IP Another digital method uses the Internet Protocol to transfer video as packets. This provides great flexibility and range of the video signal, at the cost of reliability and risk of packet loss. Its task is to deliver packets from a source host to a destination based on the IP addresses in its packet header. The physical medium is commonly twisted-pair cable for cheap and reliable end-node communication, fiber optical cable for long-range high-bandwidth connections, or wireless RF where the data is modulated on a carrier wave of 2.4 GHz or 5 GHz.

The complexity of IP communication is reduced to layers that provide a specific task, and a conceptual model known as the OSI model shows this. The OSI model can be seen illustrated in figure 2.2. Since the packets are moved through intermediate routers and switches, a packet can travel around the world almost instantly, but all packet handling and propagation of the signal will incur delay.

For local networks, the complexity of IP communication, wrapping of packets with destination address and other features of IP communication translates to delay between two host applications.

Video compression using a video codec is used to allow image data to be reduced in size to not congest the network, and the video codecs available for a given IP camera varies.

IP communication hardware is cheap and abundant, flexible and dynamic. Network congestion may lead to packet loss.

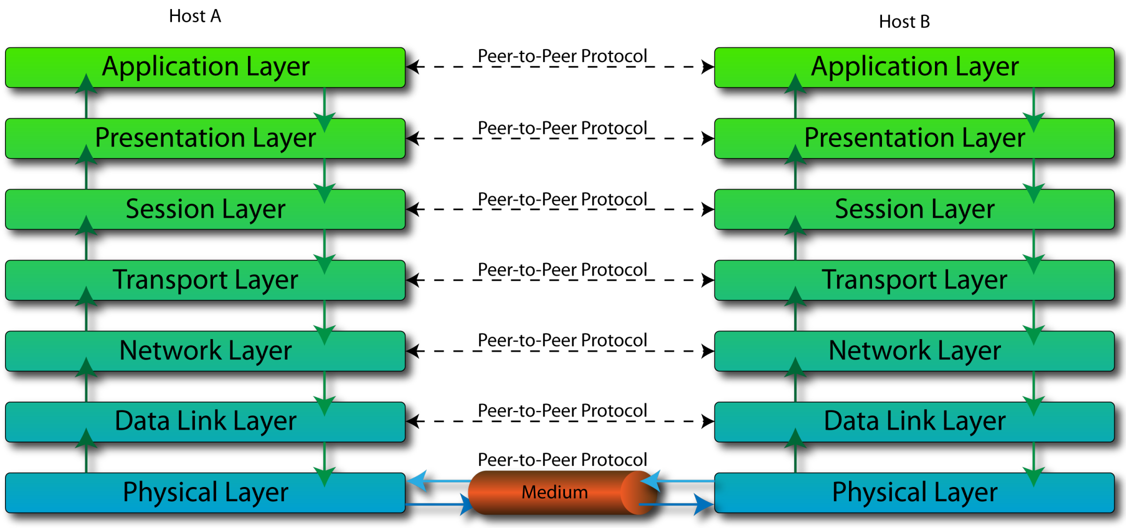

Figure 2.2: The OSI model and communication from host A to B. Source: Online (2015)

2.0.3 Video compression

A video codec is used to compress image data for transmission, and decompress it when it has arrived at its destination. It is also used for storing video on a computer. Compression is typically lossy, which means that data is removed on purpose to reduce the need for storage. We can divide video codecs into two groups, one group relying on intraframe- and the other on interframe compression.

Intraframe Every image frame is compressed on its own, with no relations to frames either before or after itself. This compression form is not as good in terms of size reduction, but it uses less CPU than the interframe compression. MJPEG is such a codec, that takes every single image frame and compresses it using the JPEG image compression algorithm. The bandwidth used by an intraframe compressed video stream is is more consistent than when interframe is used, but it is also on average higher.

Interframe At intervals, a full frame is stored, and a series of frames after this are reduced to only store the frame data that has changed from the full frame. This gives greater compression as it takes advantage of temporal redundancy between neighboring frames. This kind of compression is very CPU intensive, compared to intraframe compression. H.264 is such a codec, that creates I-frames at intervals, and fills in any changes in a frame by using B- and P-frames that are interpolated Each I-frame updates the full image.

2.0.4 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.

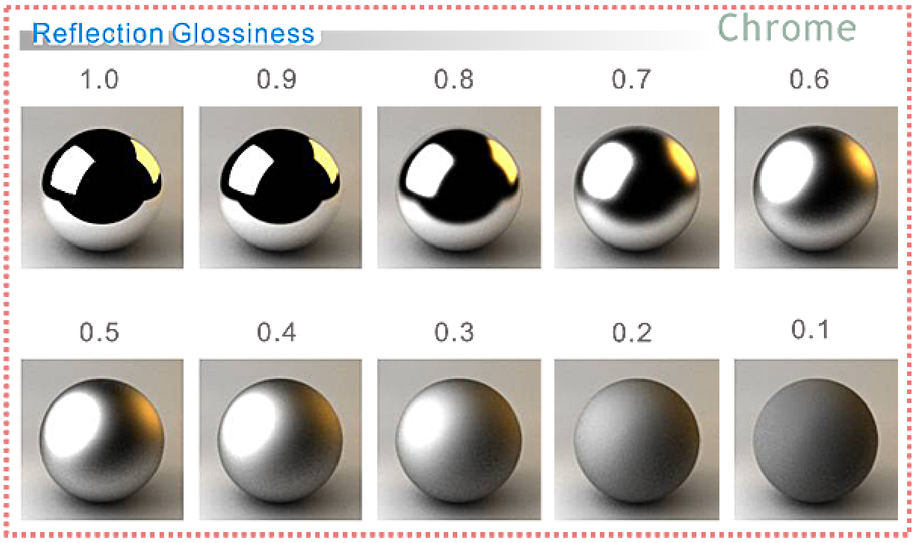

Figure 2.3: Glossiness and its change on reflection as shown in raytracing software. Source: Online VRay (2015)

2.0.5 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.

2.0.6 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.

In the case of video compression, some codecs are more suited for parallel computing, while others are not. If one is intending to compress video, this should be kept in mind so that a codec that supports parallel computing is selected. The codec may also use the GPU for speedups.

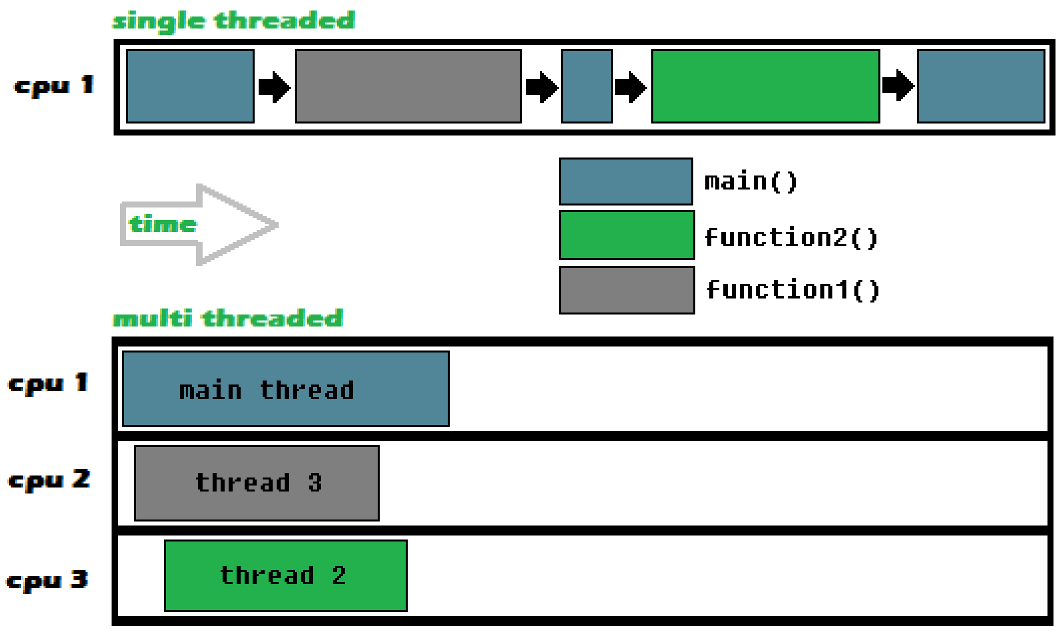

Figure 2.4: Singlethread versus multithread execution. Source: Online Codebase (2015)

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.

2.0.7 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.

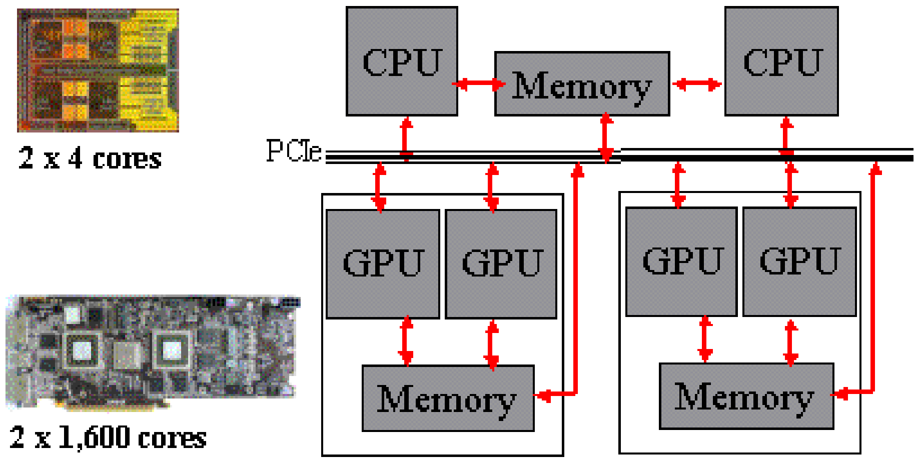

Figure 2.5: Conceptual overview of a heterogeneous system with CPU and GPU cores. Source: Online Technology (2013)

CUDA

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

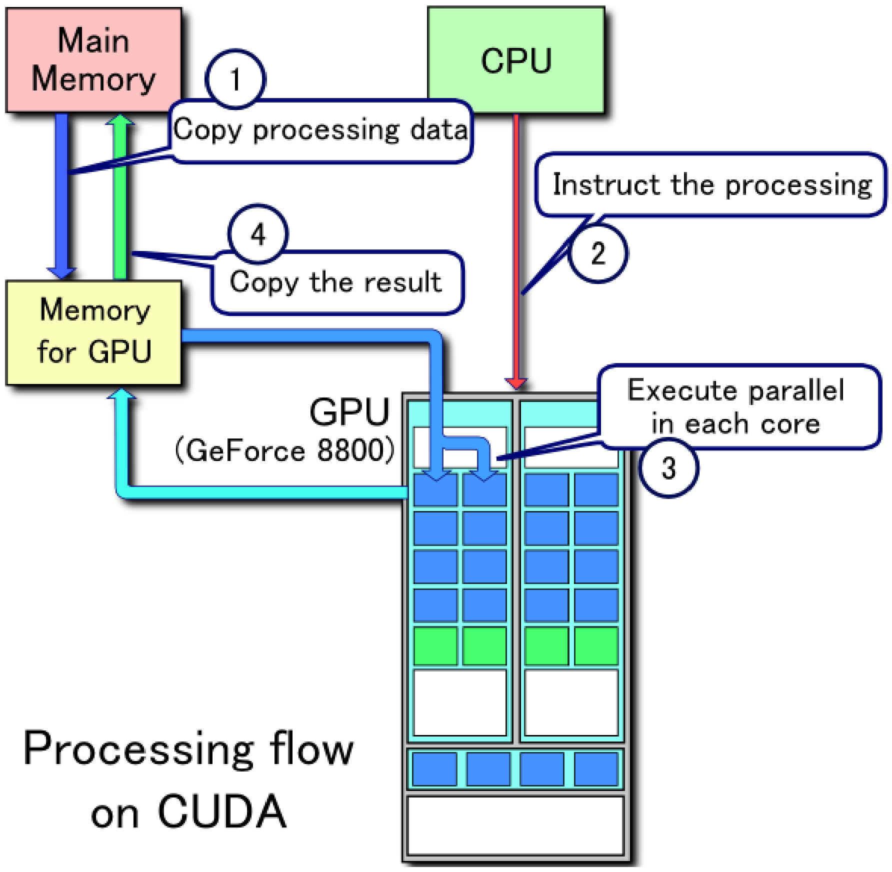

Figure 2.6: CUDA processing flow. Source: Online Wikipedia (2015a)

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.

2.0.8 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.

Performance speedups of groups of algorithms by using GPU acceleration with CUDA can be seen in figure 2.7, tests done by the OpenCV project.

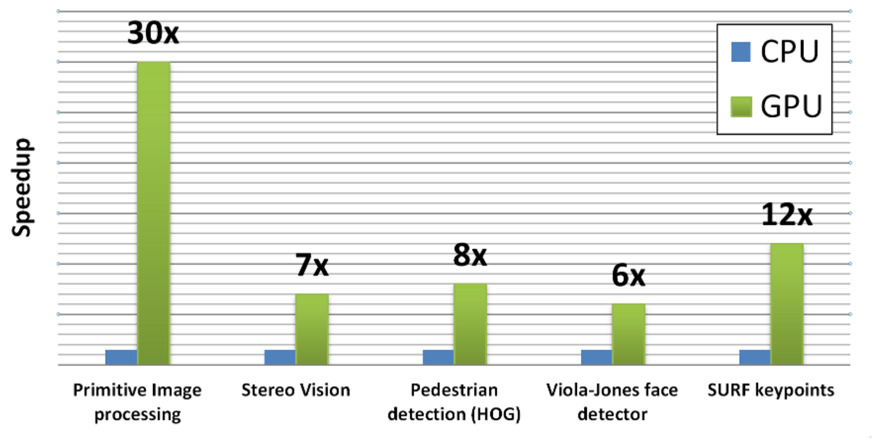

Figure 2.7: Performance speedups when using CUDA and a GPU. Source: Online OpenCV.org (2015)

2.0.9 Computer vision

Computer vision is a field with overlaps from machine vision. Computer vision concerns automatic extraction, analysis and understanding of useful information. Computer vision is also used for visual computer programs that displays data.

Author's comments The usage of ”computer vision” and ”machine vision” is often intermixed, and the exact definition of each is unknown. For the purpose of this thesis, the assumption is that when someone mentions one of these terms, the other may be the one they were meaning to use.

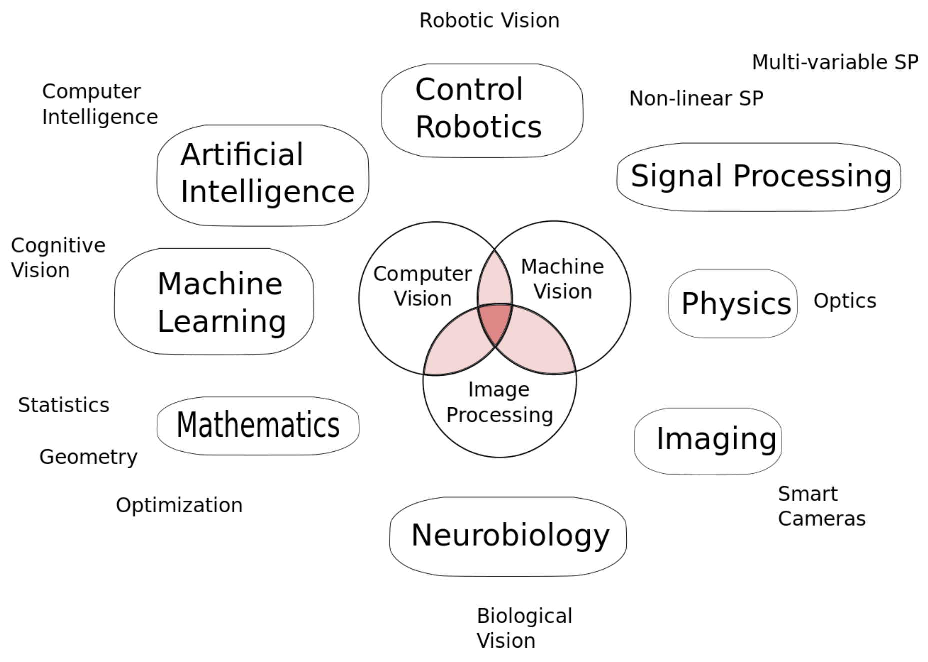

Figure 2.8: Relation between computer vision and various other fields, including machine vision. Source: Online Wikipedia (2015b)

2.0.10 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, 2015d) 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. Credit: Vegard Haugland, MHWirth AS, Haugland (2015)

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.

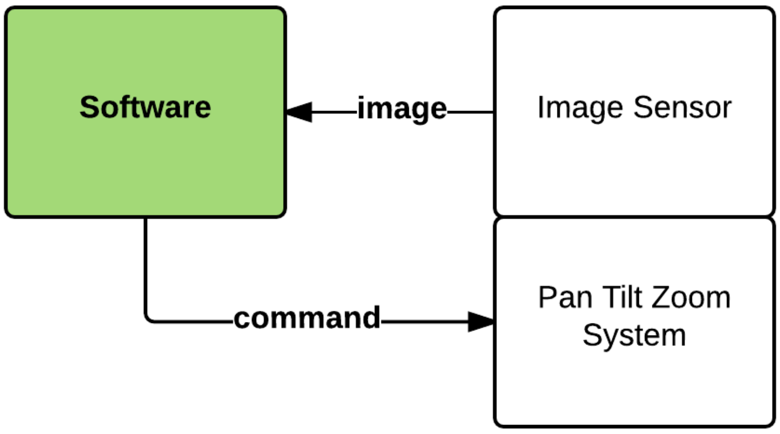

Figure 3.2: System overview using a single camera. Source:Own work, based on Skjefstad (2014)

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.

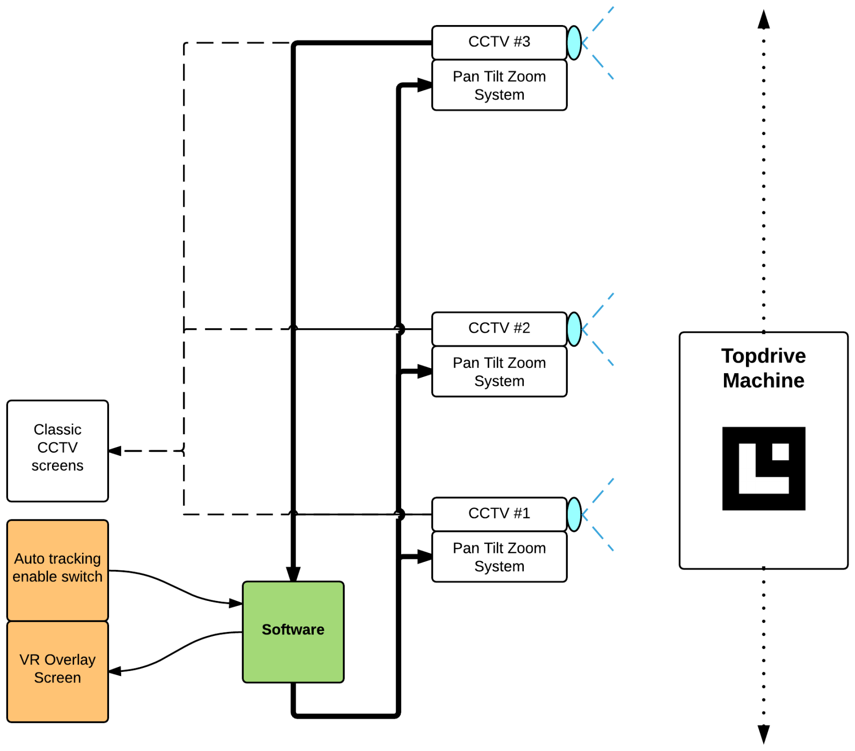

Figure 3.3: System overview when used on an oil rig. Source: Own work

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).

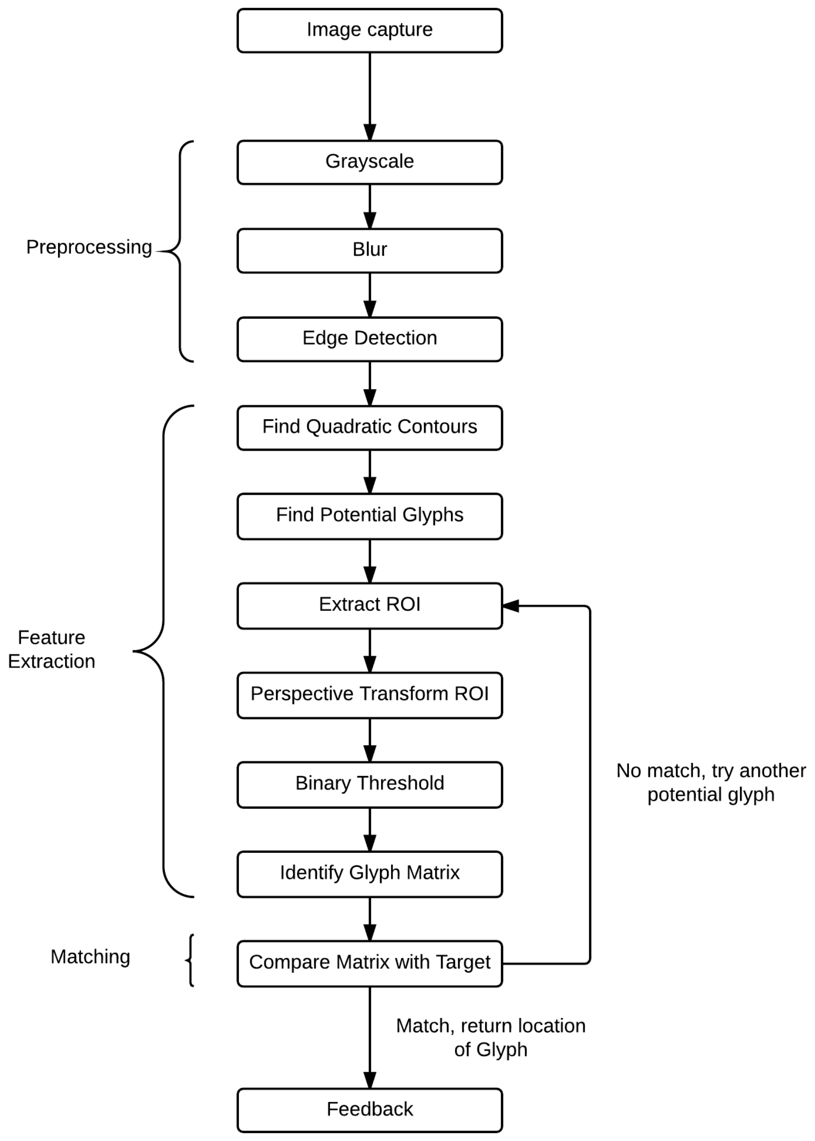

Figure 3.4: The vision algorithm as implemented. Source:Skjefstad (2014)

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().

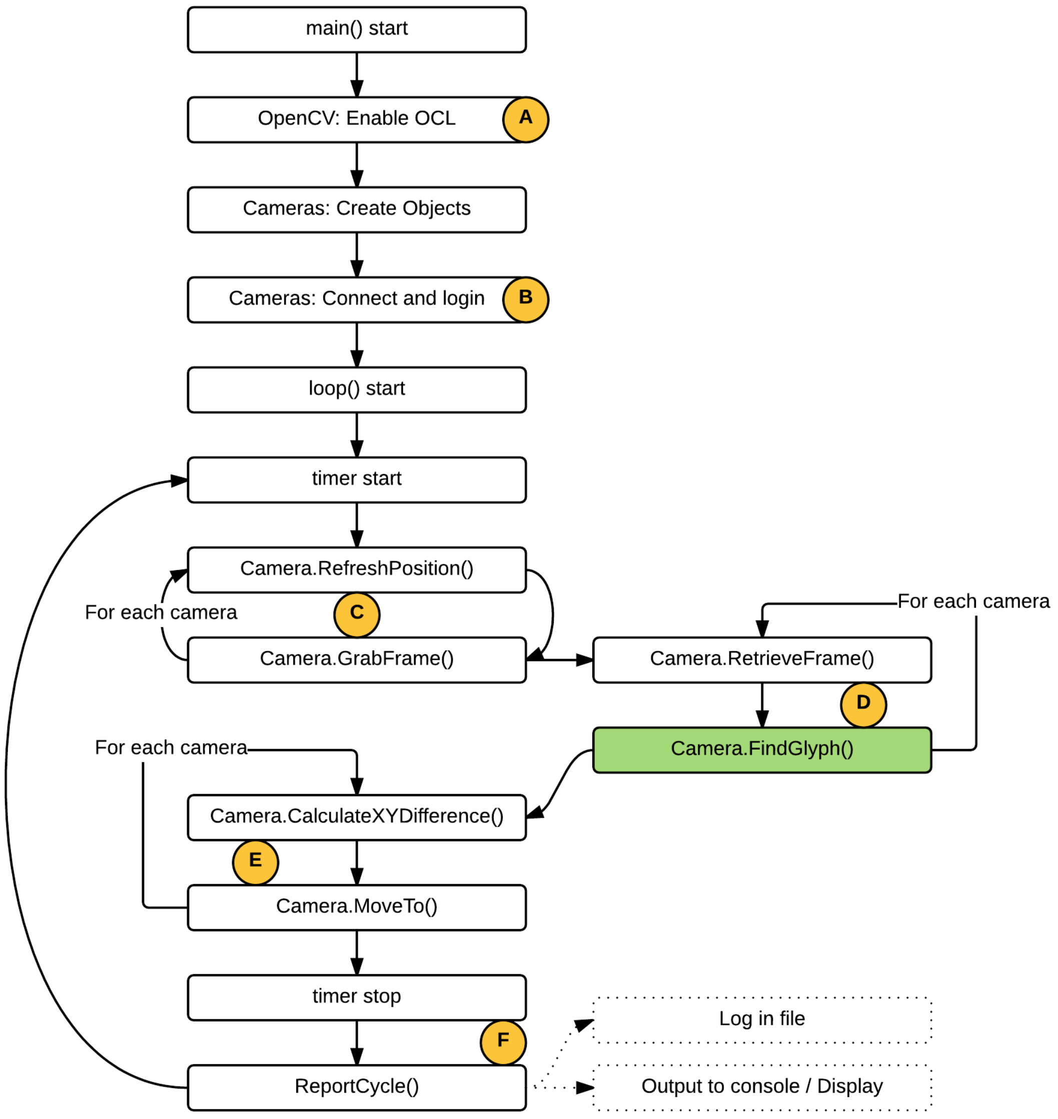

Figure 3.5: Software flowchart as implemented. Source: Own work

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.

Enable In order to enable OpenCL, we have to set an environment variable. At the same time, we can explicitly tell OpenCV which device should be used. ":GPU:0" means that we will use the first GPU for acceleration.

```
export OPENCV_OPENCL_DEVICE=":GPU:0"
```

In addition, we will have to run a function to enable OpenCL in the C++ code.

```
cv::ocl::setUseOpenCL(true);
```

We also have to create the context, and select device, which is done as follows:

```
cv::ocl::Context context;
context.create(cv::ocl::Device::TYPE_GPU);
cv::ocl::Device(context.device(0));
```

Disable If we rather want to disable OpenCL, we have to set the environment variable using "qqq" as value.

```
export OPENCV_OPENCL_DEVICE="qqq"
```

In addition, we will run the same function as when we enabled, but with another argument in the C++ code.

```
cv::ocl::setUseOpenCL(false);
```

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.

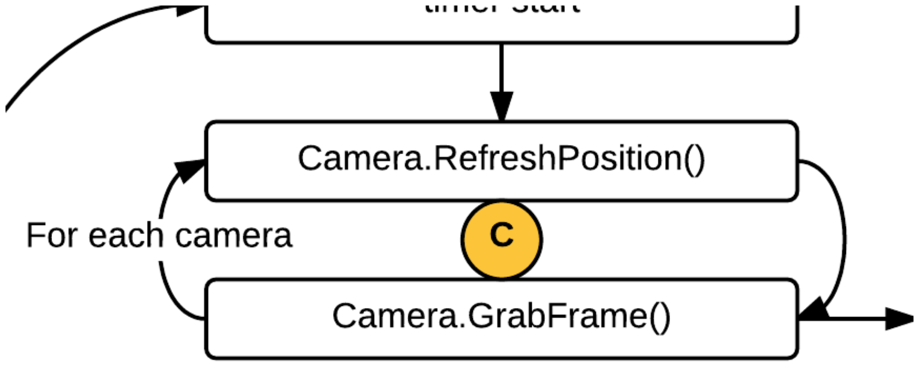

Figure 3.6: Part C of flowchart. Source: Own work

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.

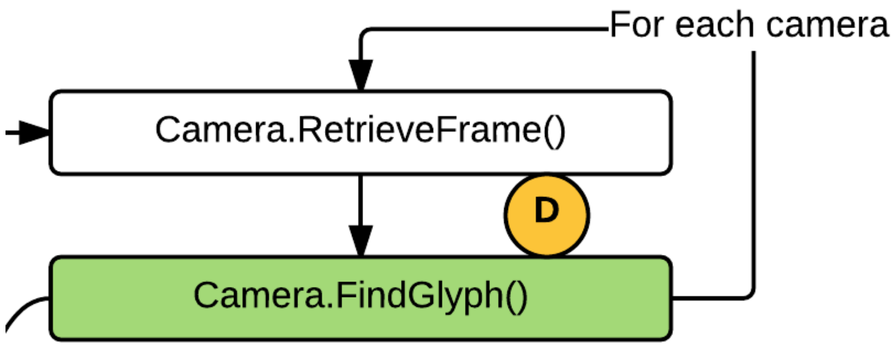

Figure 3.7: Part D of flowchart. Source: Own work

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.

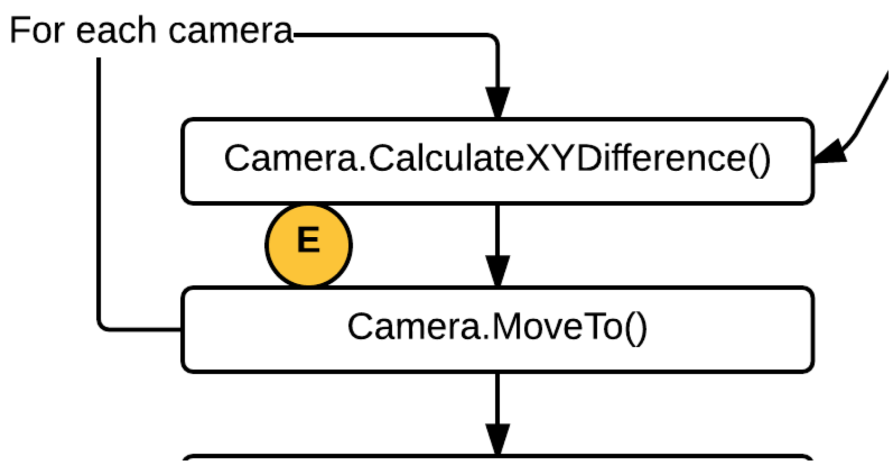

Figure 3.8: Part E of flowchart. Source: Own work

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.

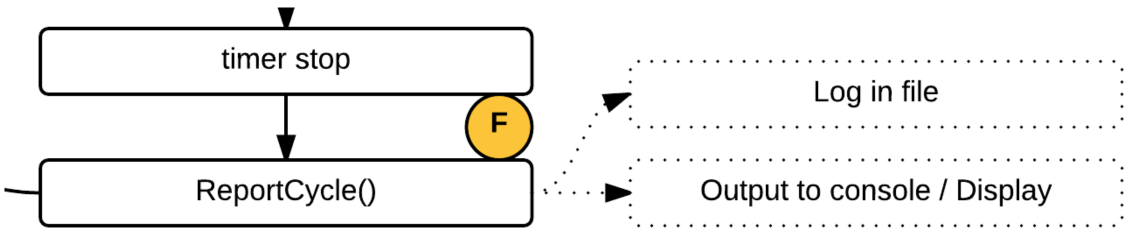

Figure 3.9: Part F of flowchart. Source: Own work

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.

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.

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 C page 79 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.

$$9.274741s - 9.033639s = 0.241102s \tag{3.1}$$

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

3.1.7 Camera settings

The AXIS Q6045 camera settings are adjusted in order to approach more favorable conditions in which the machine vision program operates.

- Resolution was set to 480x270 pixels as this seems sufficient for tracking a glyph. It also reduces the amount of data that has to be processed.
- Maximum frame rate was limited to 10 fps per viewer, where default is Unlimited. This also reduces the amount of images that pile up in the buffer on the computer.
- MJPEG will be used for encoding the video stream. MJPEG is an intraframe video compression format, and it is the option that requires the least amount of work by the operating system in the CCTV camera, leading to least amount of latency possible without using SDI or modifying the operating system.

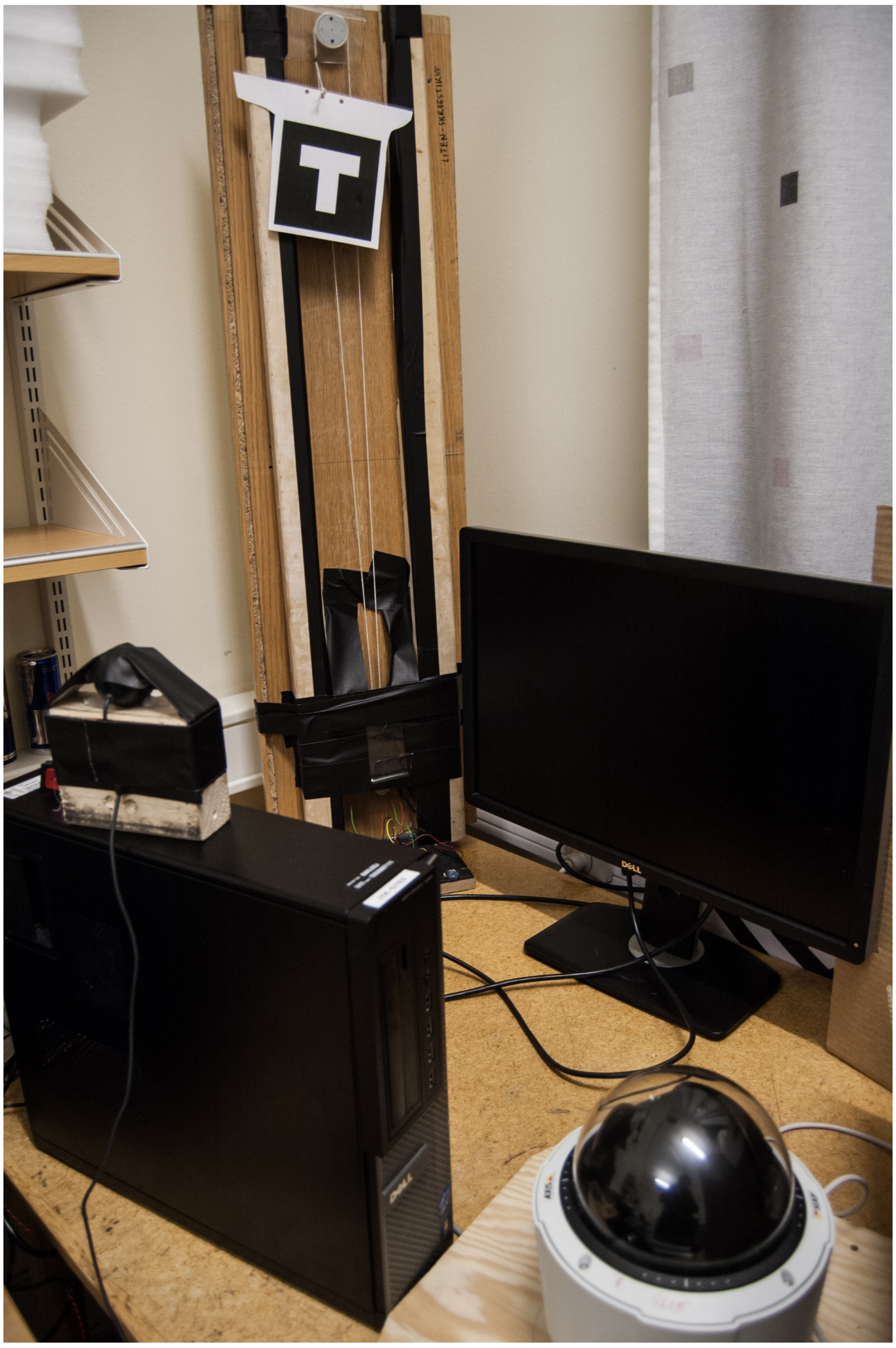

Figure 3.10: The tabletop setup. Source: Own work

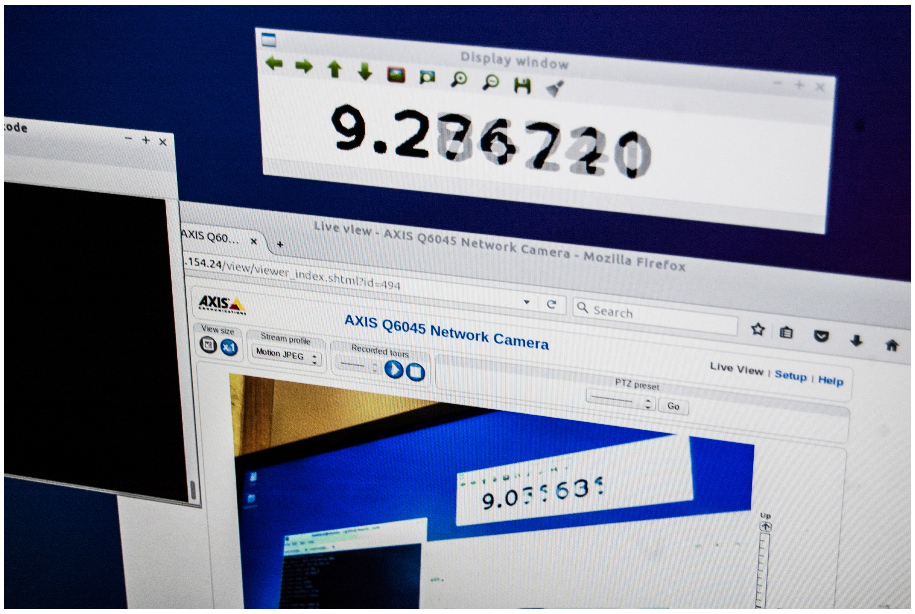

Figure 3.11: The timer software in use, showing 241 ms delay. Source: Own work

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.

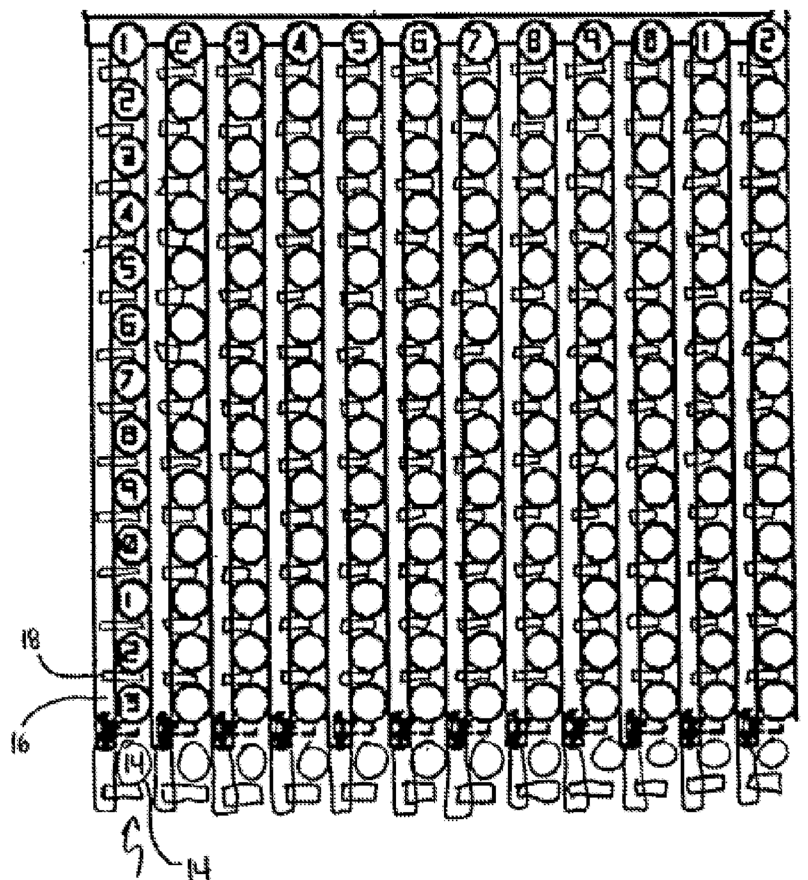

Figure 3.12: Diagram of a fingerboard with tubulars. Source:Braxton (2013a)

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.

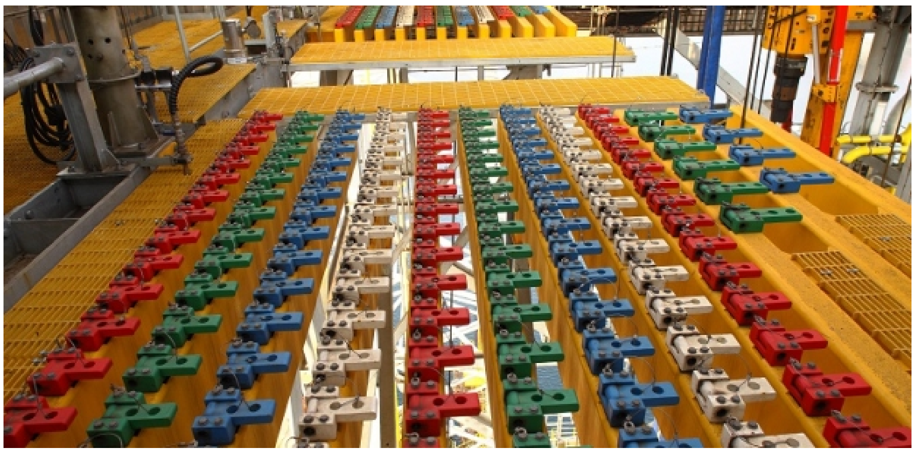

Figure 3.13: Fingerboard without tubulars. Source:TSC (2015)

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.

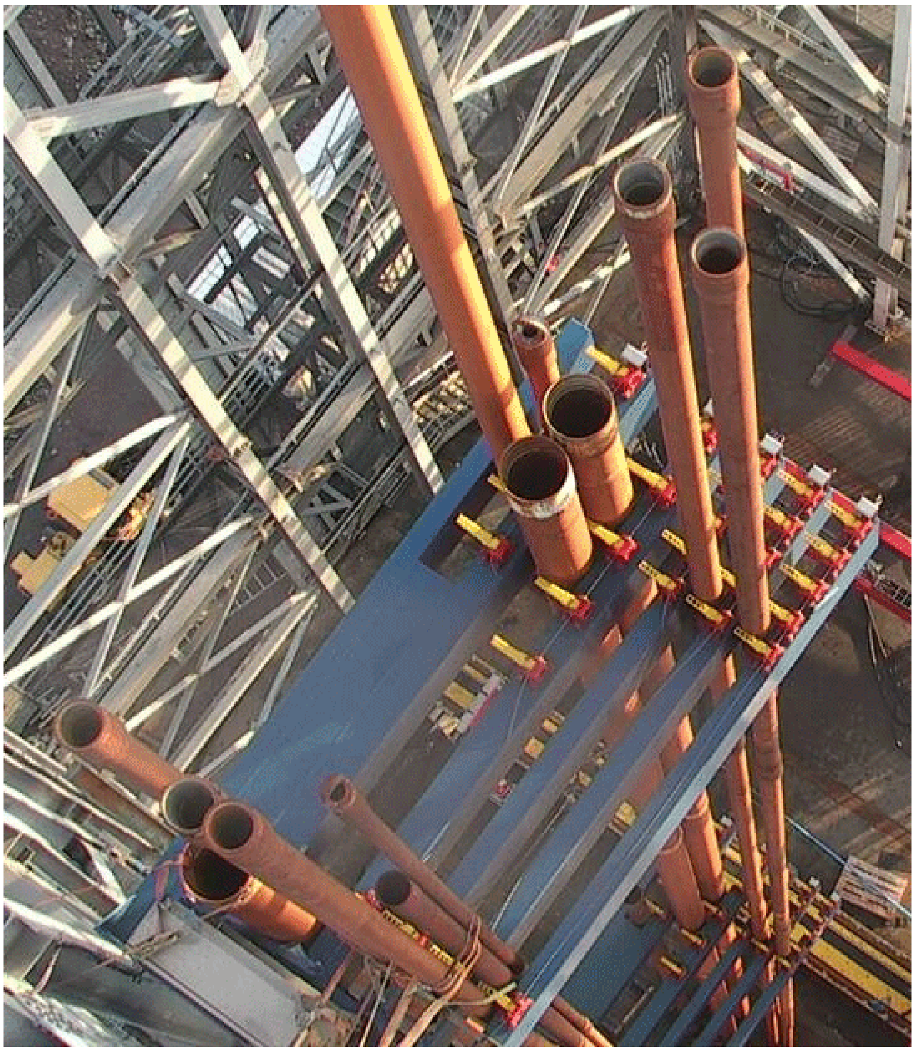

Figure 3.14: Fingerboard with tubulars. Source: Own work (MHWirth AS)

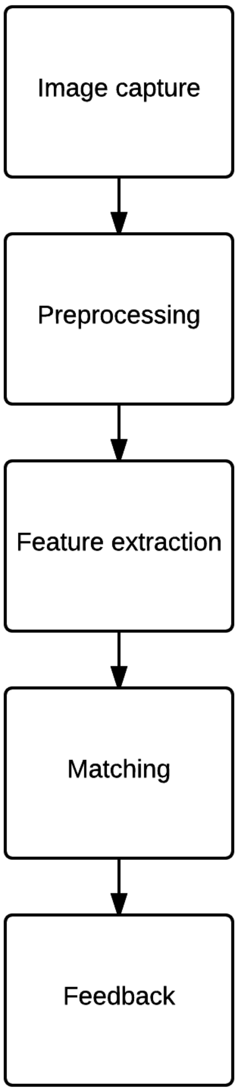

Figure 3.15: A generic machine vision implementation before detailed description. Source: Own work

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.

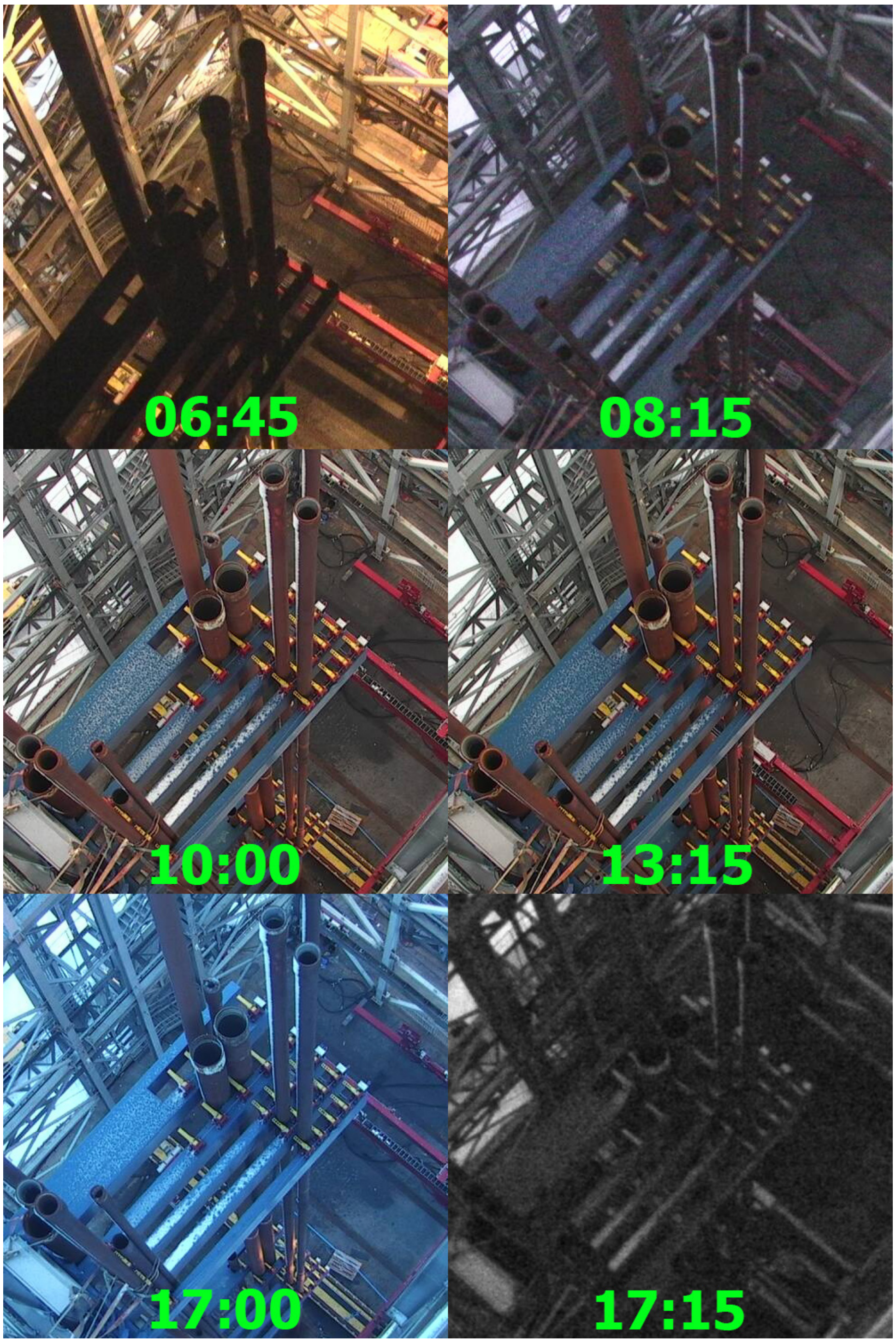

Figure 3.16: Matrix of CCTV images to show differences in lightning. All times in UTC+1, captured 25th of January 2015. Source: Own work

A thing to note is that the drilling floor on an offshore platform is not necessarily outdoors, but is partially sheltered from the environment, including natural light. This depends on the configuration. Machines may therefore move sheltered from natural light most of the time. Figure 3.17 shows a typical well center, partially in a partially sheltered area on an offshore oil platform. Figure 3.18 shows drillfloor as viewed from above, the fingerboards can be seen in the shade of the walls surrounding the tower.

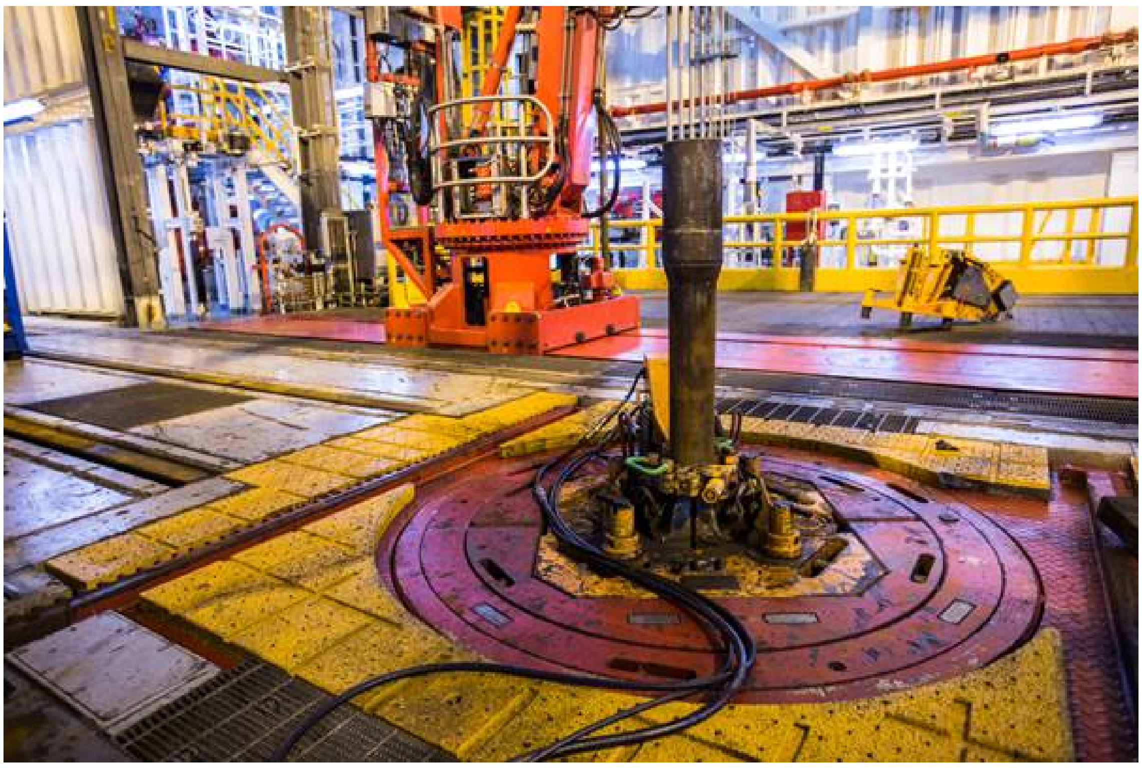

Figure 3.17: Well center, at the base of the drill floor. Credit: Vegard Haugland, MHWirth AS, Haugland (2015)

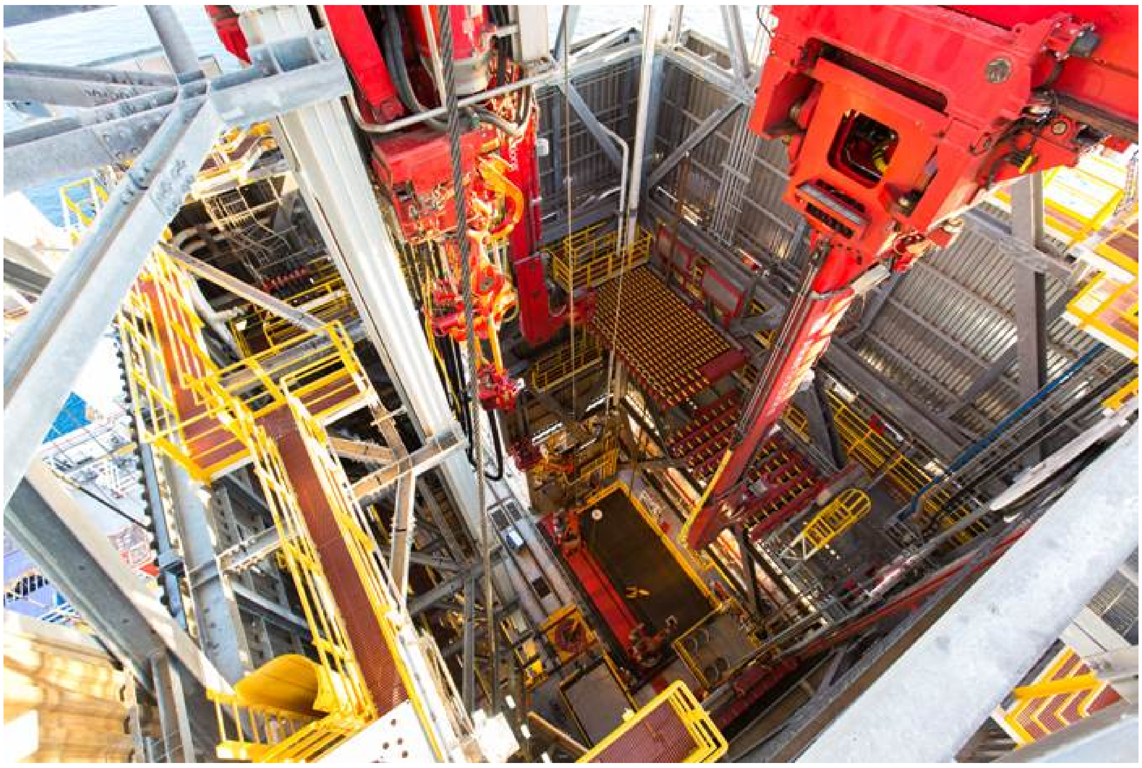

Figure 3.18: Looking down on the drill floor from outside. Credit: Vegard Haugland, MHWirth AS, Haugland (2015)

Since the drilling floor seems to be sheltered, we will avoid using a picture with snow on it, as was seen in the matrix figure 3.16, and instead we use figure 3.19, taken 18th of January 15:00 UTC+1 in the MHWirth test tower.

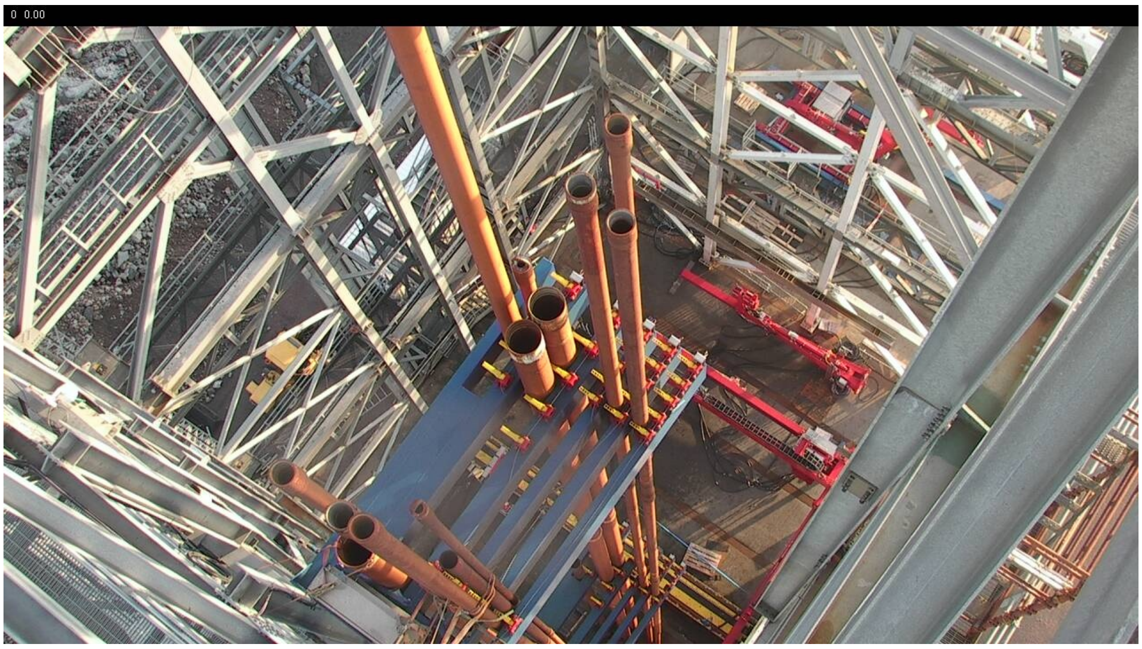

Figure 3.19: Fingerboard tubular detection sample image, taken 18th of January 15:00 UTC+1. Source: Own work

Preprocessing

After we have captured the image, we will need to preprocess it to maximize the performance of subsequent algorithms. The way we process the image depends on what we would like to do with it later on. One common step is to make a grayscale image and blurring it, reducing colors that may not be needed and filtering noise. In this case, we will also try to use what colors are found in the image, since the fingerboard, tubulars and fingers appear to have distinct colors. By using Adobe Photoshop or another image manipulation tool, we can quickly try out different blending methods, which may give us an idea of ways to improve the image to make it more suitable for known vision algorithms.

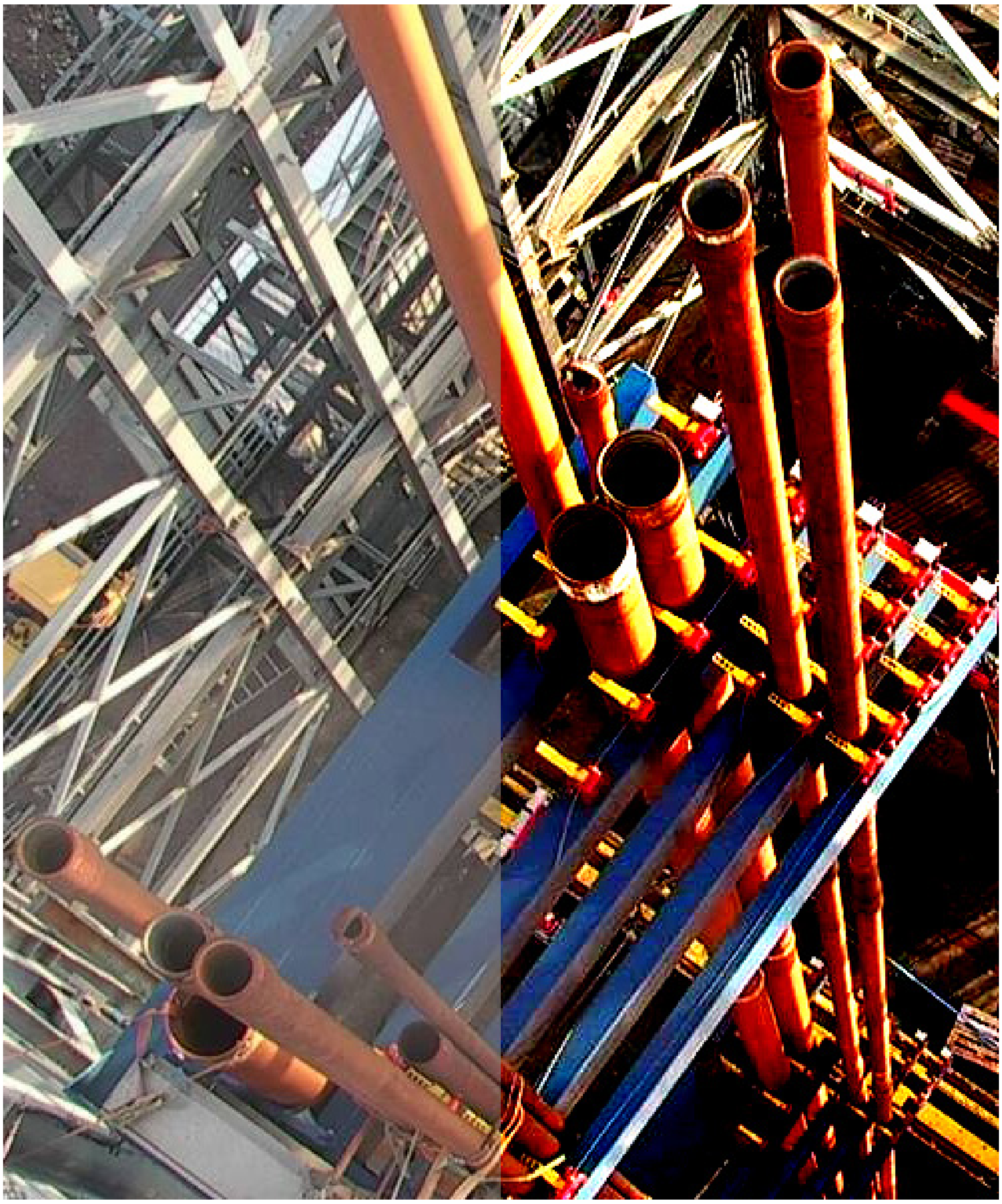

Figure 3.20: Linear light blending of color image using Adobe Photoshop. Source: Own work

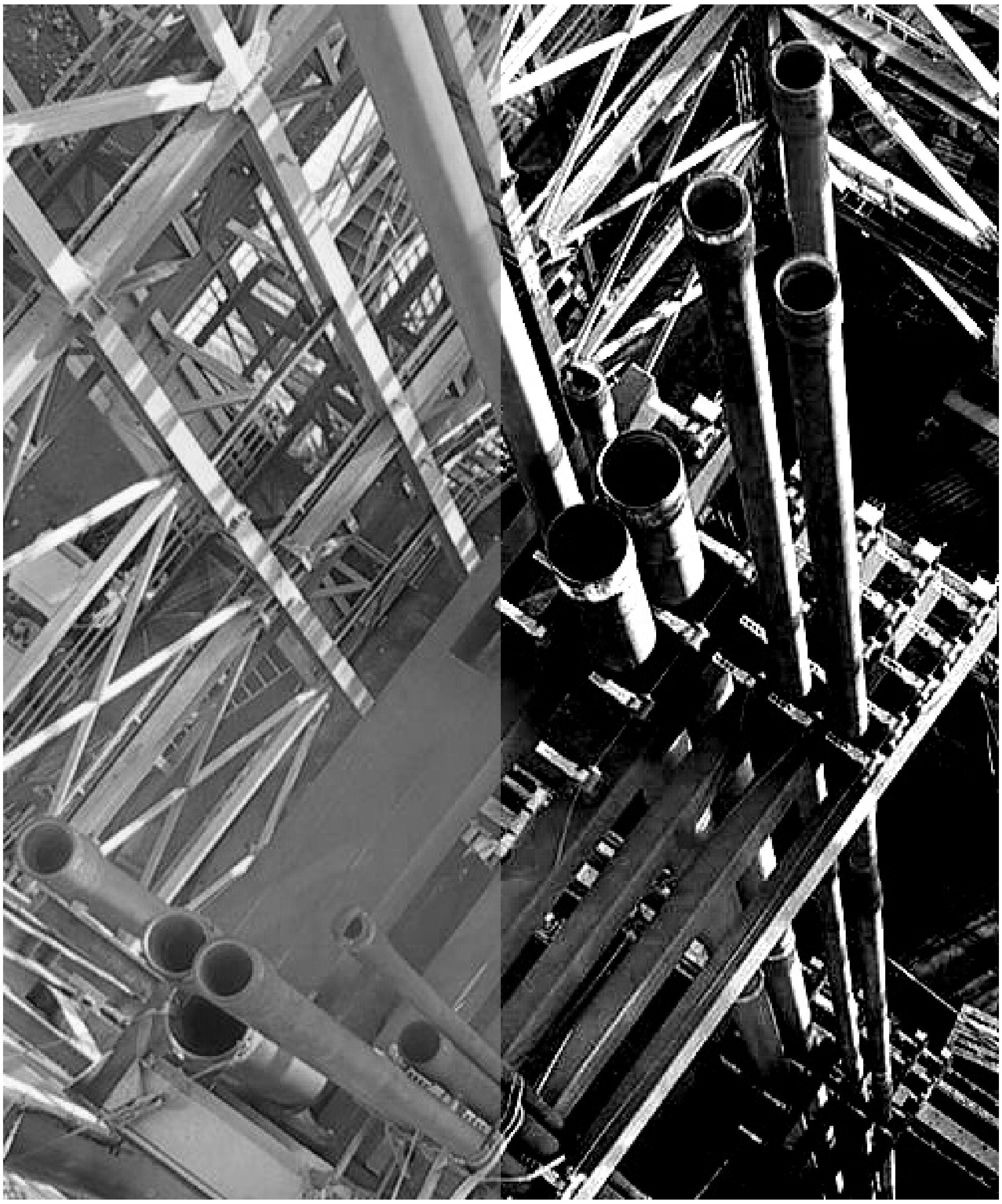

Figure 3.21: Linear light blending of grayscale image using Adobe Photoshop. Source: Own work

Feature extraction

We would like to identify the fingerboard, use its position and orientation to evaluate the position of the fingers, and use the extracted information to filter out any tubulars that may be detected which does not make sense.

A way to split our problem into several smaller problems is found in using the HSV color space, and masking out parts of the image depending on what HSV range a pixel falls inside. This is done for fingers, fingerboard and tubulars.

For the tubulars, we also use Canny edge detection on a grayscale, blurred image in order to find the dark circle in each tubular.

Matching

The SimpleBlobDetector as described in the OpenCV documentation **?** does most of the work. The algorithm does the following:

1 One source image is converted to several binary images using several thresholds.

2 Connected components are extracted from every binary image using findContours, and centers are found.

3 Centers from several binary images are grouped by their coordinates.

4 The groups estimate final centers and their radiuses, and is returned as locations and sizes of keypoints.

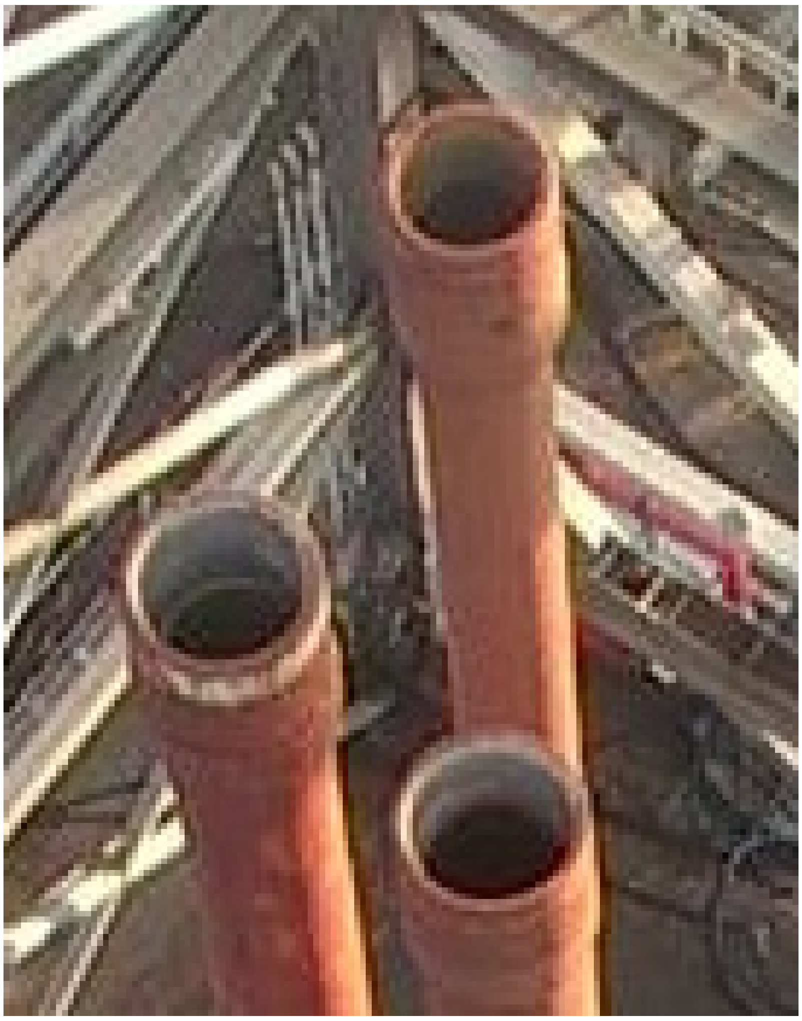

Figure 3.22: Scaled closeup of pipes from sample image. Source: Own work

It also is controlled by a set of parameters, experimentally found to work for the specific image and light-conditions. This is a weakness that does not provide robust detection throughout changing light conditions, but it provides a quick and easy way of finding shapes of a certain circularity and convexity.

Feedback

The contours from the HSV masking are colored depending on their assumed function, and the keypoints of all blobs found are marked with a circle.

Complete implementation

The program flow can be seen in 3.23. Its output can be seen in Chapter 4 Results.

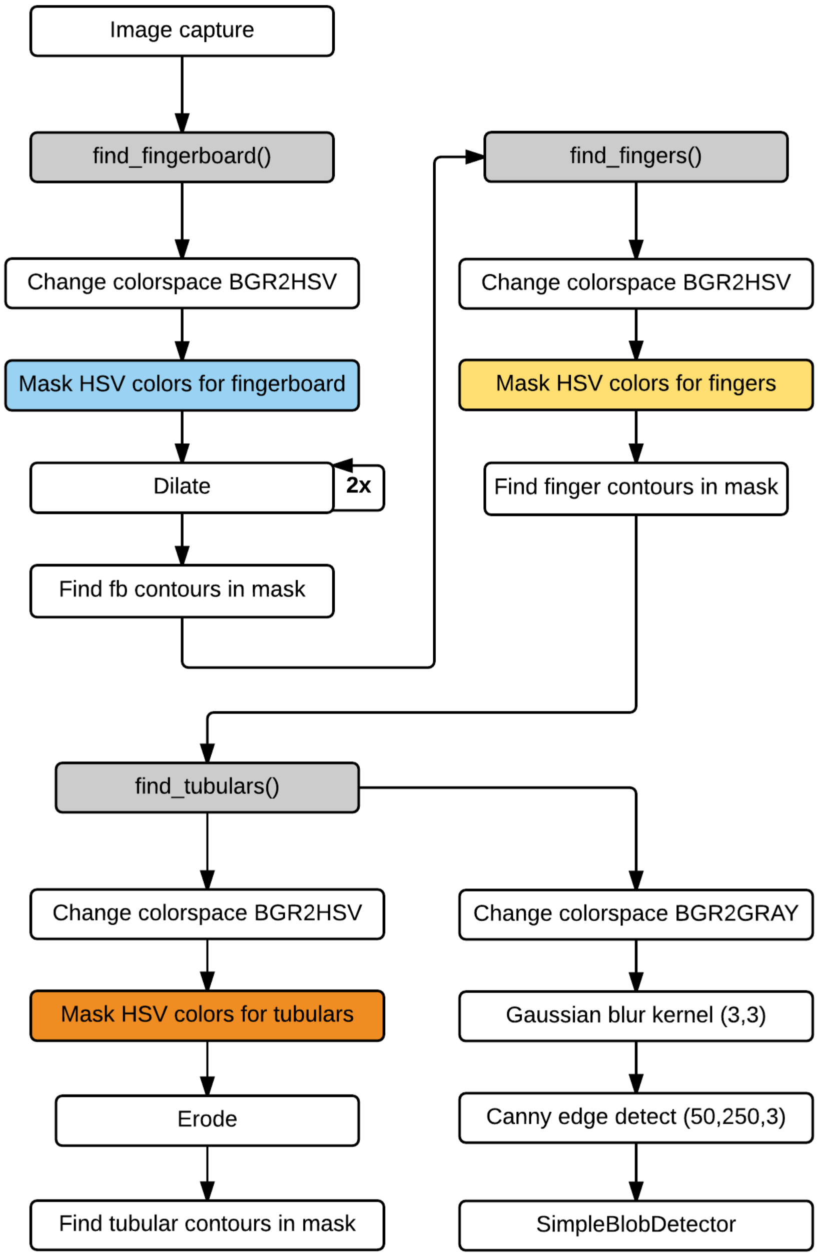
Image capture
find_fingerboard()
Change colorspace BGR2HSV
Mask HSV colors for fingerboard
Dilate
2x
Find fb contours in mask
find_fingers()
Change colorspace BGR2HSV
Mask HSV colors for fingers
Find finger contours in mask
find_tubulars()
Change colorspace BGR2HSV
Mask HSV colors for tubulars
Erode
Find tubular contours in mask
Change colorspace BGR2GRAY
Gaussian blur kernel (3,3)
Canny edge detect (50,250,3)
SimpleBlobDetector

Figure 3.23: The proof of concept implementation for finding tubulars. Source: Own work

Chapter 4

Discussion and results

4.1 Results

4.1.1 Auto CCTV tracking

All versions were allowed to run for 100 cycles, and the time for each full cycle was logged. One full cycle involves grabbing the image, processing it, finding any glyphs and sending PTZ-commands to the camera. It was also logged how much time was spent on grabbing the image. The mean full cycle time was plotted in the same plot.

C++ only The data can be found in appendix B.1 on 63.

- Mean: 43.6120 milliseconds
- Std Dev: 3.7799 milliseconds
- Minium: 38.0935 milliseconds
- Maximum: 176.378 milliseconds

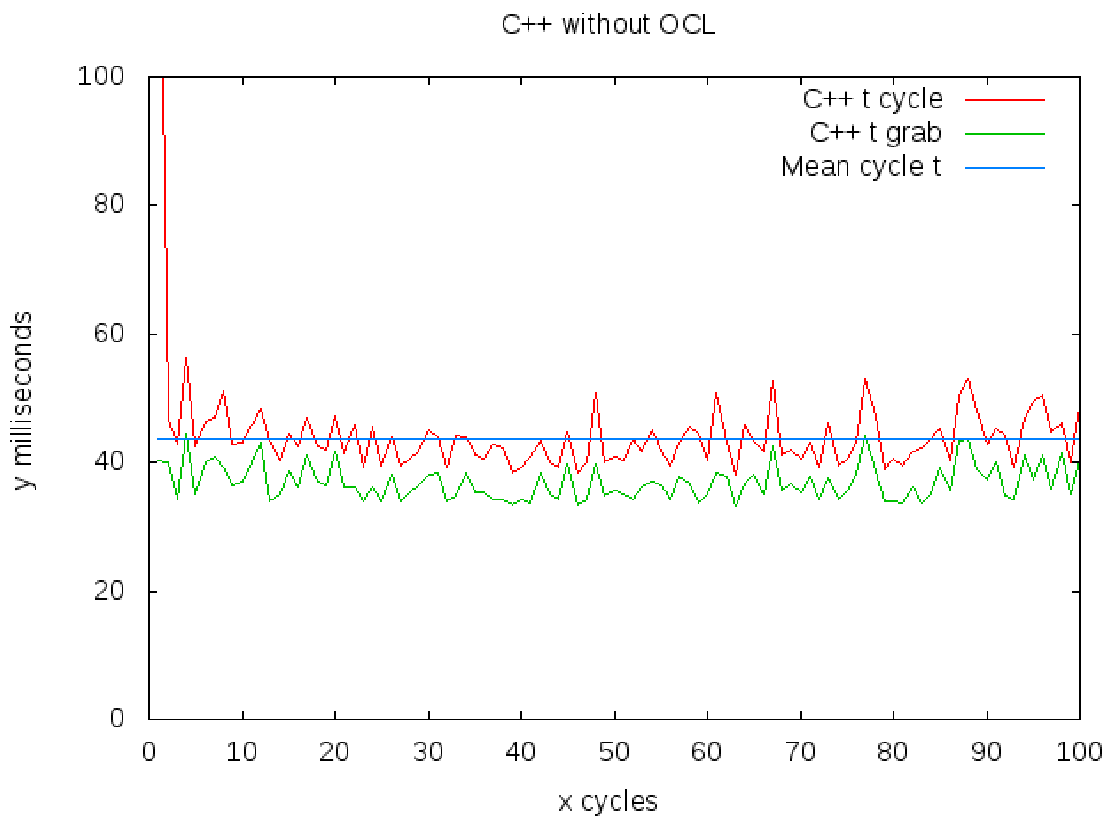

Figure 4.1: Cycle timing for pure C++ version. Source: Own work, data in B.1

C++ and OpenCL The data can be found in appendix B.1 on 66.

- Mean: 44.3577 milliseconds
- Std Dev: 4.7298 milliseconds
- Minium: 37.0841 milliseconds
- Maximum: 147.944 milliseconds

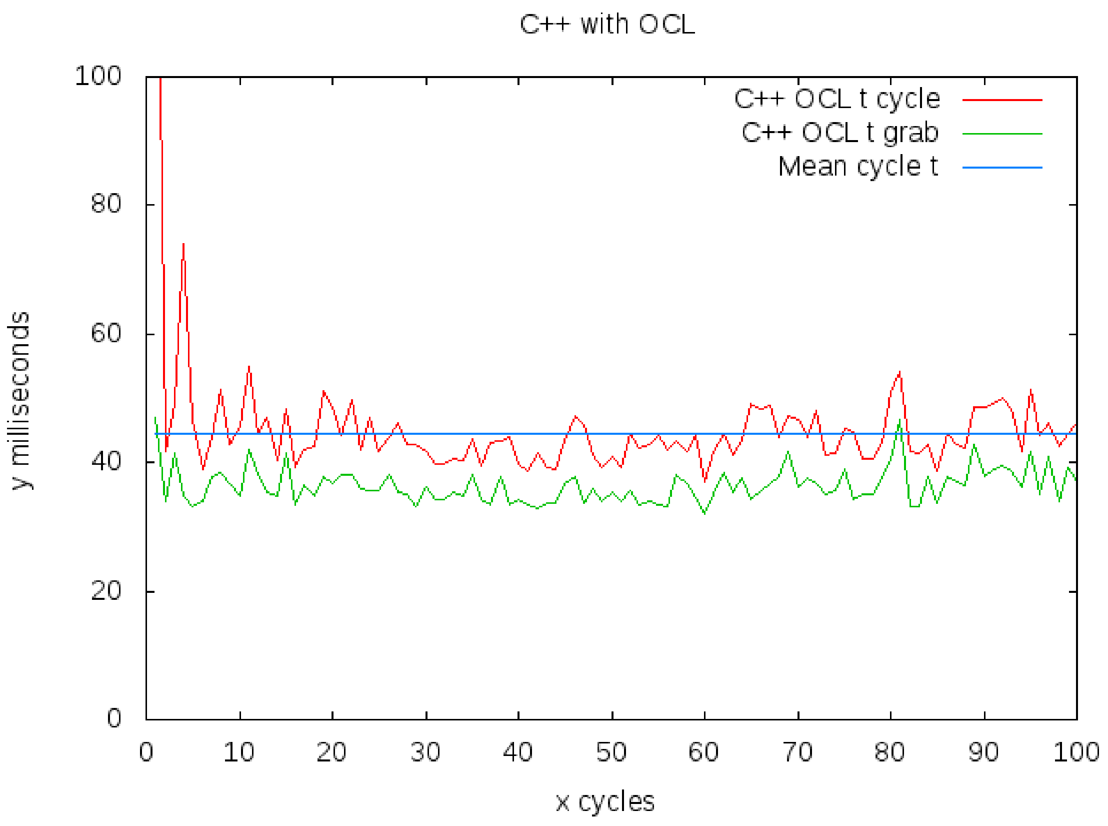

Figure 4.2: Cycle timing for C++ and OpenCL version. Source: Own work, data in B.1

Python The data can be found in appendix B.1 on 66.

- Mean: 8302.39 milliseconds
- Std Dev: 31.9534 milliseconds
- Minium: 8270.0 milliseconds
- Maximum: 8535.0 milliseconds

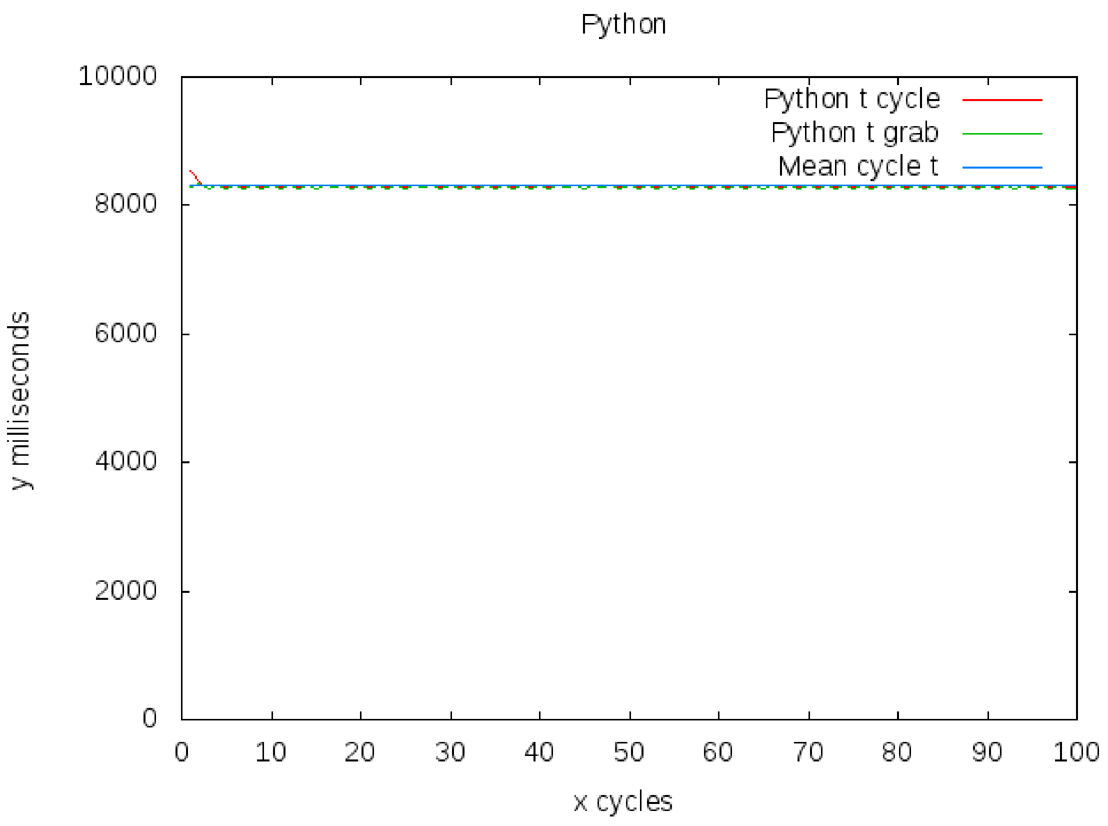

Figure 4.3: Cycle timing for Python version. Source: Own work, data in B.1

4.1.2 Tubular detection

The tubular detection proof of concept provided output with colored overlays. The pink circles represents a tubular that has been detected.

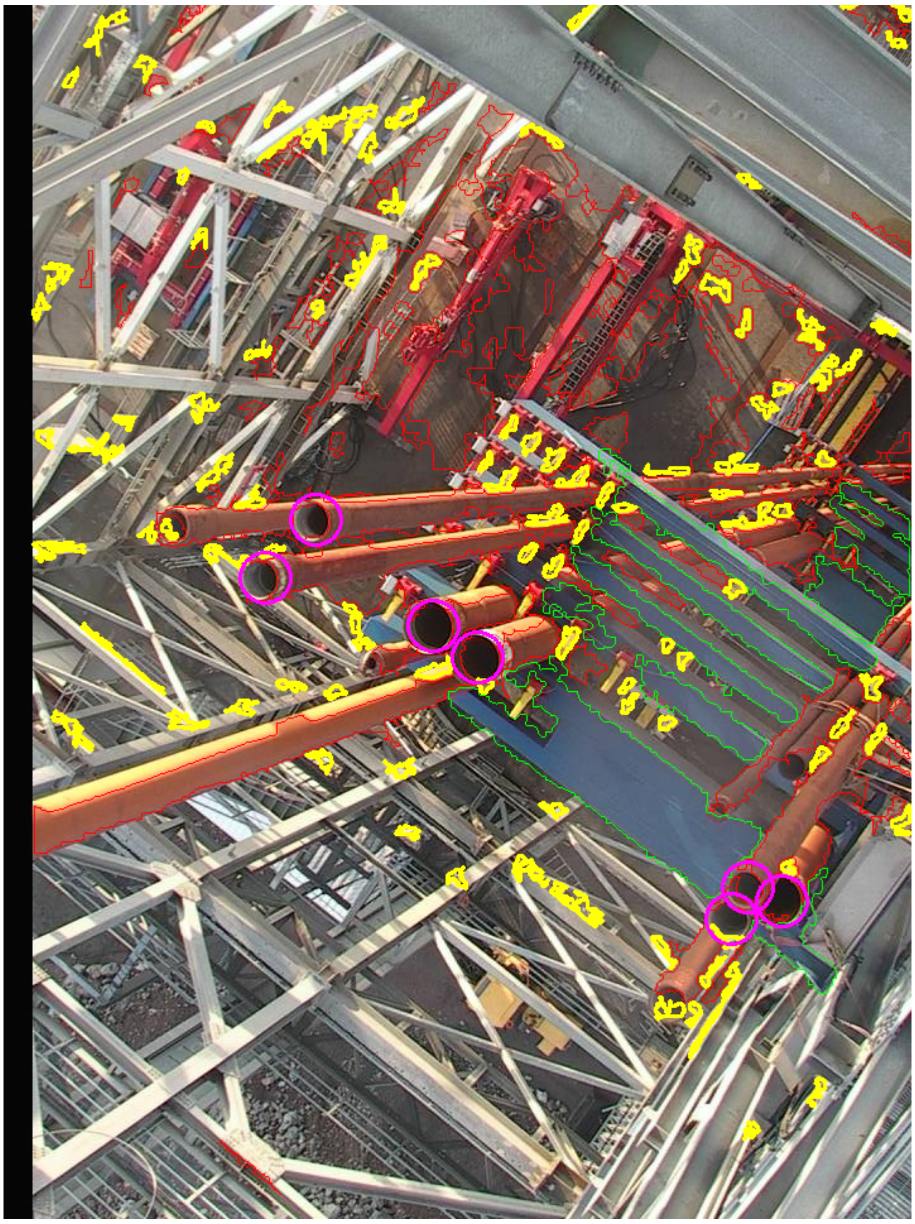
Figure 4.4: The output from the implementation. Source: Own work

4.2 Discussion

4.2.1 Auto CCTV tracking

The C++11-program ran quicker than the Python-program, but it did not seem that using OpenCL had much of an effect on the implementation. Contrary to what was expected, it seems that the OpenCL-accelerated run of the implementation actually increased cycle times and slowed down the program.

Some changes had to be made to the Python version to make it run using OpenCV 3.0.0, in order to have a common ground for testing. It is unknown whether these changes have made the Python implementation slower than when it was used by the author for the work on the unpublished project thesis. Specifically, the video capture was changed to use cv2.videoCapture, instead of a custom-made video capture method. This relies on external libraries.

The reason for the C++11-program to run with a cycle time mean of 43.6120 milliseconds, a lot less than the old Python version that had a cycle time mean of 8302.39 milliseconds, can also be contributed to a different implementation of the FindGlyph-method. The author tried to make the Python and C++-version equal, but in the end, the C++ version made assumptions about the glyph being the largest square in the image whereas the Python version inspects all squares in the image. However, by looking at the Python version grab cycle time, it appears to be mostly the video capture that consumes resources.

Looking at the C++11-program when used with OpenCL, difficulties with getting the T-API to work with all OpenCV-algorithms were encountered. This ended with most of the FindGlyph-method to work with the traditional cv::Mat instead of the new OpenCL-enabled cv::Umat. This explains why no real speedup was found. The process of enabling OpenCL and transferring image data into the GPU makes the program incur some overhead, which can be seen slightly in the cycle time plot.

The C++11-program did however track the glyph well, and were able to follow the glyph as it moved up the linear actuator.

4.2.2 Tubular detection

The program was implemented in Python and used OpenCV 3.0.0, and its output shows that seven out of eleven tubulars were identified properly, which is not great. The detection was a result of blob detection, a simple algorithm that does not use the extra information that is present in the image, including the position of the fingerboard, the tubular pipe walls and fingers found in the fingerboard.

The results show that it is possible to find tubular in a fingerboard using machine vision, however the reliability depends on how well the machine vision is implemented. By having images from several angles, it should be possible to achieve better results. The authors dataset was only from one camera, and the site was unsheltered, which means that snow and lightning effects make the valid dataset even smaller.

4.3 Recommendations for future work

4.3.1 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.

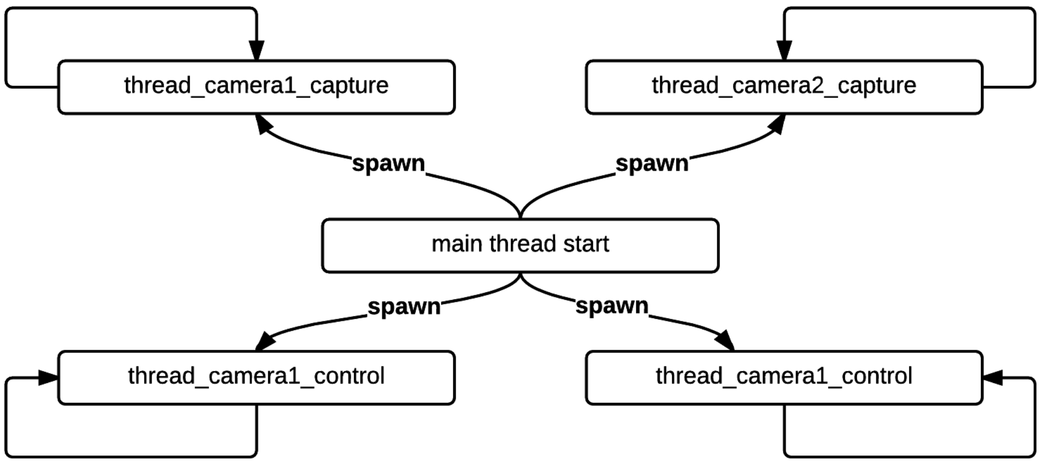

Figure 4.5: Multithreading concept runs camera capture independent of camera PTZ control. Source: Own work

4.3.2 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.

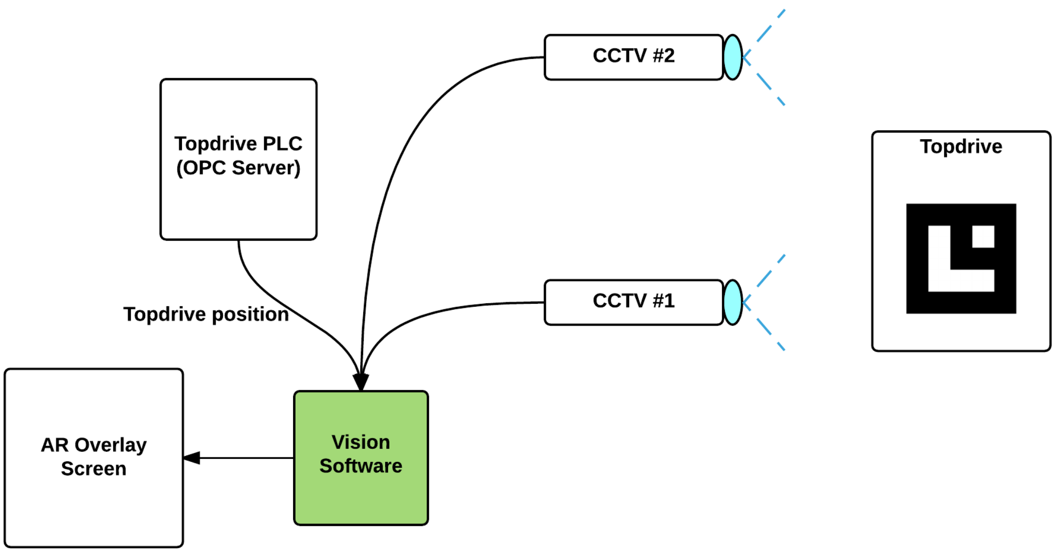

Figure 4.6: Data fusion using PLC and vision data. Source: Own work

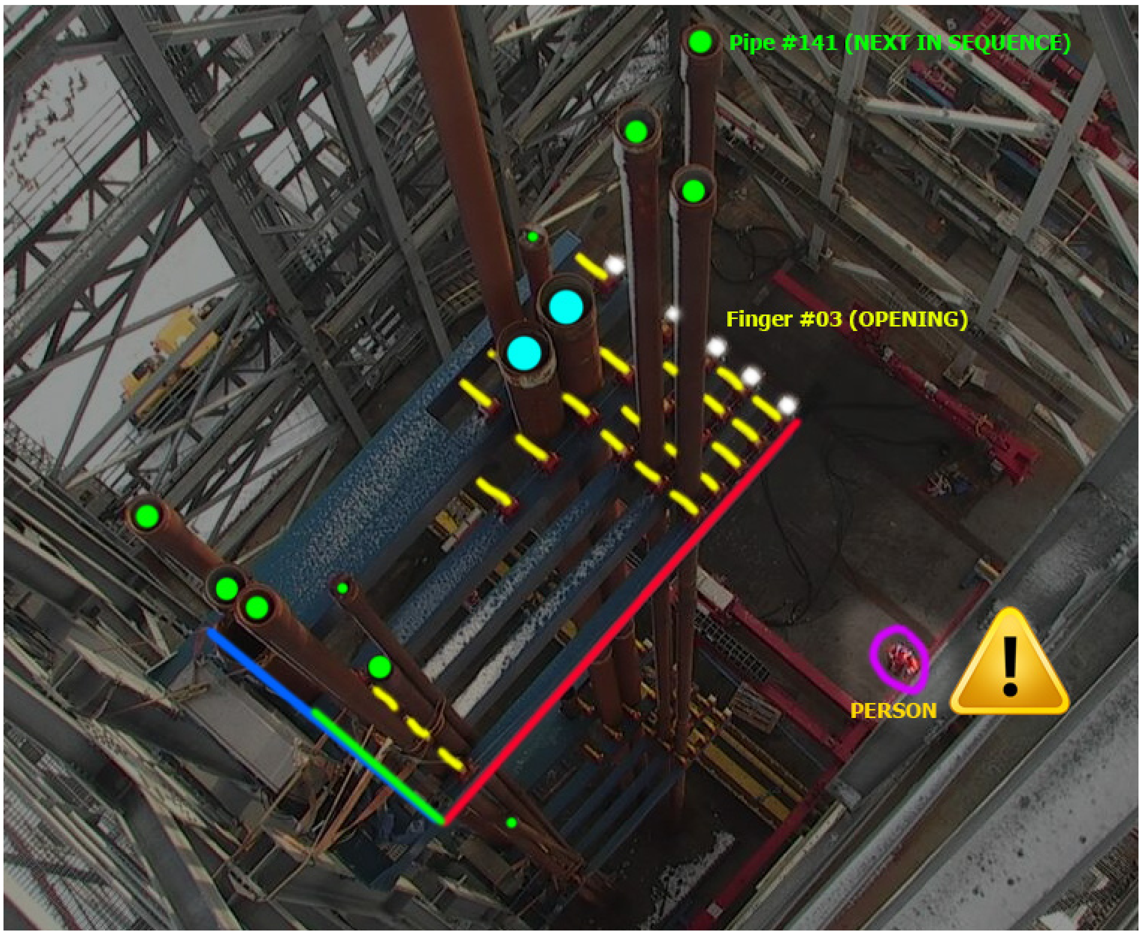

Figure 4.7: Fingerboard augmented reality mockup. Source: Own work

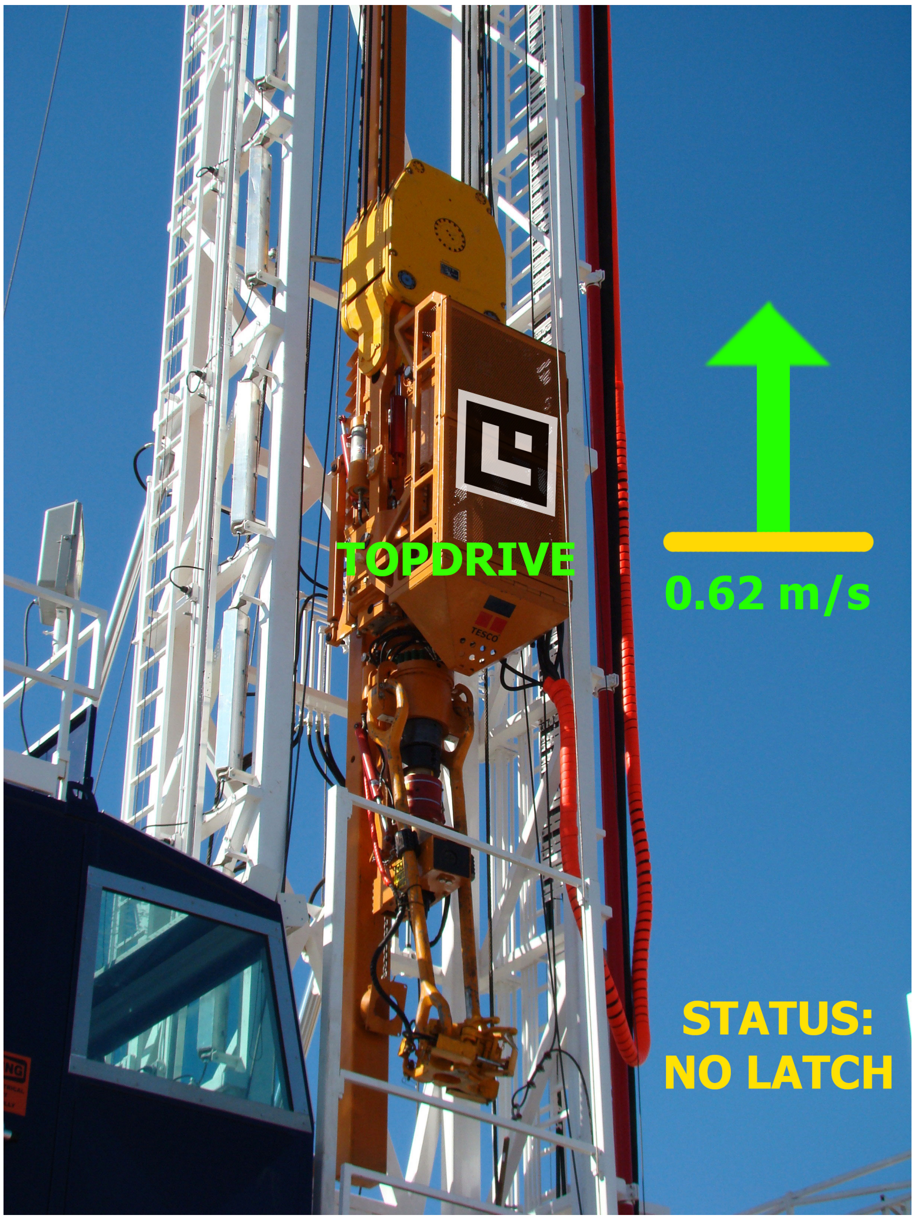

Figure 4.8: Topdrive augmented reality mockup. Source: Drillingcontractor (2015) modified by author

Appendix A

Acronyms

API Application Programming Interface

AR Augmented Reality

CD Compact Disc

CV Computer Vision

MJPEG Motion JPEG

PTZ Pan Tilt Zoom

SSL Secure Socket Layer

Appendix B

Data

All data is presented here. Please refer to Results chapter for interpretation.

B.1 Cycle times

C++ only full cycle

```
X i Y ms
1 176.378
2 46.7335
3 42.8079
4 56.2232
5 42.5046
6 46.4085
7 46.9647
8 51.2249
9 42.6807
10 43.074
11 45.7883
12 48.4487
13 43.4754
14 40.2384
15 44.5116
16 42.5299
17 46.8599
18 42.6178
19 42.0281
20 47.1095
21 41.5663
```

```
22 45.9818
23 39.272
24 45.6873
25 39.4183
26 44.0499
27 39.5539
28 40.6502
29 41.7309
30 44.9498
31 43.9632
32 39.127
33 44.1275
34 43.8152
35 41.2881
36 40.5132
37 42.7548
38 42.3039
39 38.295
40 39.2344
41 40.9386
42 43.2348
43 40.0477
44 39.2855
45 44.6541
46 38.3992
47 40.1653
48 50.7354
49 40.0467
50 41.0108
51 40.2891
52 43.6791
53 41.6941
54 45.0173
55 41.6503
56 39.5076
57 42.9829
58 45.7003
59 44.4445
60 40.3687
61 50.7763
```

```
62 42.9567
63 38.0935
64 45.7982
65 43.3269
66 41.7508
67 52.6519
68 41.081
69 41.9777
70 40.4794
71 42.9862
72 39.3377
73 46.1746
74 39.5005
75 40.7182
76 43.0511
77 52.9545
78 47.3266
79 38.8306
80 40.612
81 39.463
82 41.7259
83 42.3771
84 43.6022
85 45.2836
86 40.434
87 50.3894
88 53.1199
89 48.1457
90 42.7772
91 45.3578
92 44.0828
93 39.3102
94 47.0397
95 49.3594
96 50.4615
97 44.7283
98 46.2623
99 40.1131
100 48.8099
```

C++ and OCL full cycle

```
X i Y ms
1 147.944
2 41.7962
3 49.0975
4 74.1173
5 46.7381
6 38.9131
7 44.0426
8 51.3266
9 42.7418
10 45.5308
11 55.0263
12 44.4359
13 46.8387
14 40.4033
15 48.4533
16 39.1249
17 41.912
18 42.5968
19 51.0063
20 48.7218
21 44.0796
22 49.6526
23 42.0155
24 46.8542
25 41.5962
26 43.8959
27 46.2624
28 42.7899
29 42.8816
30 41.7049
31 39.7829
32 39.8501
33 40.6538
34 40.4389
35 43.5918
36 39.5095
37 43.1904
38 43.3349
```

```
39 43.9085
40 39.787
41 38.6203
42 41.4169
43 39.1201
44 38.9962
45 43.4387
46 47.202
47 45.9077
48 41.1482
49 39.1846
50 40.9689
51 39.2839
52 44.5515
53 42.2828
54 42.7026
55 44.2268
56 42.0034
57 43.3895
58 41.7159
59 44.2059
60 37.0841
61 41.7701
62 44.5976
63 41.2479
64 43.2646
65 49.1386
66 48.2914
67 48.987
68 43.996
69 47.3181
70 46.7929
71 43.99
72 47.9887
73 41.1715
74 41.4115
75 45.2948
76 44.7367
77 40.6896
78 40.7
```

```
79 42.9936
80 50.7188
81 54.2351
82 41.6682
83 41.4852
84 42.8574
85 38.7173
86 44.4089
87 42.8612
88 42.2858
89 48.5942
90 48.5784
91 49.2398
92 50.1071
93 48.3967
94 41.7731
95 51.3804
96 44.3058
97 46.0689
98 42.6652
99 44.4609
100 46.1737
```

C++ only grab cycle

```
X i Y ms
1 40.4479
2 40.171
3 34.1401
4 44.5604
5 35.0968
6 40.1398
7 40.8149
8 39.3088
9 36.5718
10 37.1214
11 40.1631
12 43.0393
13 34.1063
14 34.9761
```

```
15 38.7993
16 36.2434
17 41.0549
18 36.9862
19 36.4651
20 41.6133
21 36.1032
22 36.2356
23 33.9546
24 36.1023
25 34.1117
26 38.0405
27 34.09
28 35.2262
29 36.4719
30 38.2031
31 38.4599
32 33.8567
33 34.6792
34 38.3352
35 35.3878
36 35.3308
37 34.121
38 34.1908
39 33.3726
40 34.3166
41 33.6023
42 38.3679
43 35.2119
44 34.3403
45 39.6818
46 33.5562
47 34.1263
48 39.8495
49 34.678
50 35.5639
51 34.9845
52 34.2747
53 36.2712
54 36.9791
```

```
55 36.351
56 34.3481
57 37.8511
58 36.6444
59 33.8176
60 35.2038
61 38.2618
62 37.7549
63 33.0474
64 36.6988
65 38.1752
66 35.193
67 42.677
68 35.6331
69 36.6869
70 35.378
71 37.8307
72 34.1236
73 37.4406
74 34.3096
75 35.7669
76 38.075
77 44.1568
78 38.565
79 33.8788
80 34.0235
81 33.7066
82 36.0829
83 33.6856
84 34.9796
85 39.2255
86 35.5386
87 43.3328
88 43.6468
89 38.9814
90 37.4304
91 40.1763
92 34.8519
93 34.2843
94 41.2167
```

```
95 37.1548
96 41.2104
97 35.8812
98 41.3382
99 34.9546
100 40.4104
```

C++ and OCL grab cycle

```
X i Y ms
1 46.8777
2 33.9147
3 41.3123
4 34.6927
5 33.0837
6 34.0646
7 37.9428
8 38.5008
9 36.6312
10 34.8429
11 42.0151
12 38.0685
13 35.4915
14 34.6953
15 41.3995
16 33.4375
17 36.502
18 34.8345
19 37.8528
20 36.7164
21 38.0593
22 38.1241
23 35.9353
24 35.6687
25 35.5211
26 38.0137
27 35.3627
28 34.9916
29 33.2229
30 36.1855
```

```
31 34.207
32 34.3599
33 35.2636
34 34.7753
35 38.2164
36 34.1728
37 33.518
38 37.897
39 33.5148
40 34.3679
41 33.3197
42 33.0002
43 33.8256
44 33.6357
45 36.7895
46 37.7308
47 33.5817
48 35.7839
49 33.9321
50 35.2697
51 33.9255
52 35.6039
53 33.4517
54 33.9231
55 33.4059
56 33.1347
57 38.2364
58 36.717
59 34.6507
60 32.1222
61 35.0464
62 38.4109
63 35.3966
64 37.4838
65 34.1946
66 35.7341
67 36.6243
68 37.9721
69 41.5776
70 36.1288
```

```
71 37.5029
72 36.6084
73 35.2087
74 35.556
75 38.8714
76 34.3732
77 35.1184
78 35.185
79 37.4437
80 40.3325
81 46.7577
82 33.0452
83 33.1132
84 37.8172
85 33.6276
86 37.71
87 36.9494
88 36.45
89 42.9163
90 37.884
91 38.8178
92 39.6275
93 38.5493
94 36.1084
95 41.7693
96 34.9868
97 40.9042
98 34.0571
99 39.2282
100 36.8838
```

Python full cycle

```
# X i Y ms
1 8535
2 8363
3 8272
4 8317
5 8281
6 8315
```

```
7  8280
8  8325
9  8280
10  8316
11  8284
12  8318
13  8283
14  8328
15  8270
16  8316
17  8287
18  8317
19  8279
20  8322
21  8278
22  8318
23  8284
24  8322
25  8276
26  8317
27  8287
28  8315
29  8286
30  8318
31  8279
32  8320
33  8282
34  8319
35  8278
36  8321
37  8274
38  8320
39  8280
40  8323
41  8280
42  8317
43  8288
44  8324
45  8272
46  8317
```

47	8281
48	8323
49	8286
50	8314
51	8281
52	8316
53	8282
54	8328
55	8272
56	8323
57	8277
58	8324
59	8276
60	8321
61	8276
62	8325
63	8283
64	8327
65	8270
66	8317
67	8281
68	8325
69	8277
70	8317
71	8280
72	8322
73	8282
74	8320
75	8279
76	8318
77	8279
78	8324
79	8280
80	8320
81	8276
82	8322
83	8283
84	8317
85	8281
86	8317

```
87 8282
88 8322
89 8281
90 8319
91 8274
92 8327
93 8272
94 8327
95 8275
96 8322
97 8280
98 8321
99 8276
100 8276
```

Python grab cycle

```
# X i Y ms
1 8278
2 8342
3 8259
4 8306
5 8270
6 8307
7 8273
8 8314
9 8268
10 8306
11 8272
12 8307
13 8270
14 8316
15 8260
16 8308
17 8274
18 8307
19 8269
20 8311
21 8268
22 8310
```

```
23 8273
24 8310
25 8267
26 8309
27 8274
28 8306
29 8272
30 8306
31 8268
32 8309
33 8270
34 8306
35 8267
36 8310
37 8267
38 8313
39 8273
40 8314
41 8271
42 8310
43 8275
44 8314
45 8262
46 8310
47 8274
48 8313
49 8271
50 8303
51 8270
52 8309
53 8273
54 8316
55 8263
56 8311
57 8267
58 8312
59 8267
60 8311
61 8269
62 8314
```

```
63 8271
64 8314
65 8259
66 8308
67 8272
68 8312
69 8266
70 8309
71 8272
72 8311
73 8270
74 8309
75 8268
76 8307
77 8270
78 8311
79 8267
80 8308
81 8268
82 8312
83 8270
84 8307
85 8270
86 8308
87 8272
88 8311
89 8269
90 8309
91 8266
92 8314
93 8265
94 8316
95 8267
96 8312
97 8269
98 8309
99 8267
100 8267
```

Appendix C

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 C.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 C.2. The duct tape is used to reduce friction on the wood, so that the laminated glyph can translate without problems.

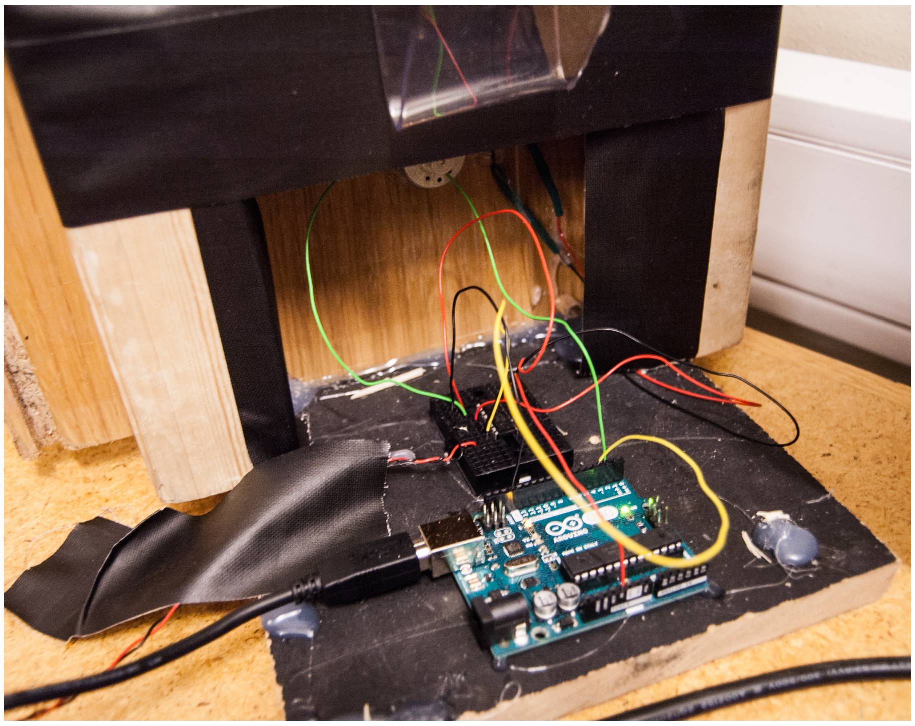
Figure C.1: Bottom assembly. Source: Own work

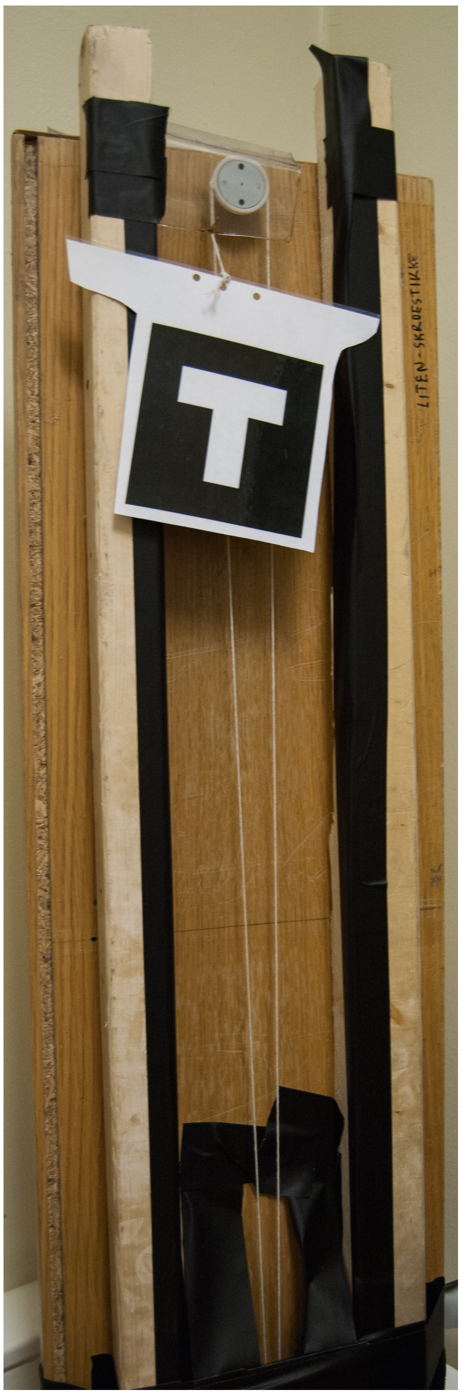

Figure C.2: Top assembly and main slider. Source: Own work

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 C.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.

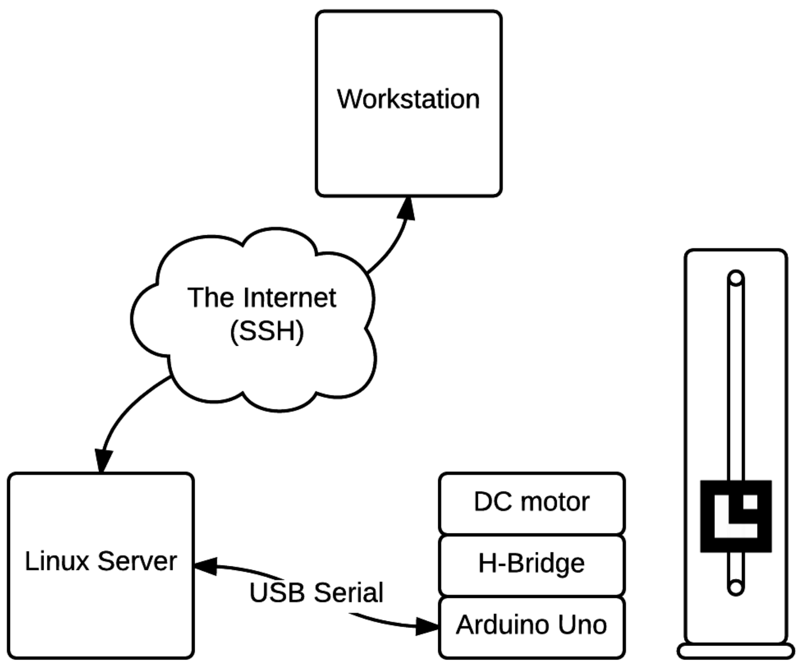

Figure C.3: Diagram over communication channel and system. Source: Own work

Appendix D

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

Tutsplus.com. (2015) Tutsplus.com. [Online]. Available: http://photography.tutsplus.com/articles/understanding-the-factors-that-affect-depth-of-field--photo-6844

S. Online. (2015) Osi model overview. [Online]. Available: http://student-online.co.za/images/OSI%20Model%20Overview.png

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

Wikipedia. (2015) Cv and mv overview. [Online]. Available: https://en.wikipedia.org/wiki/File:CVoverview2.svg

V. Haugland. (2015) Haugland foto. [Online]. Available: http://foto.haugland.at/

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," 2009.

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

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/

C. A. Poynton, *Digital Video and HDTV: Algorithms and Interfaces.* Morgan Kaufmann Publishers, 2003.

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

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.

OpenCV. (2015) Opencv 3.0.0 docs. [Online]. Available: http://docs.opencv.org/3.0.0/