UiA University of Agder

Master

Tapping beam recognition and measurement with infrared camera

Optimisation of an industrial process using an thermal imaging camera and computer vision algorithms

JØRGEN NILSEN

SUPERVISOR
Sondre Sanden Tørdal

University of Agder, 2021
Faculty of Engineering and Science
Department of Engineering Sciences

Abstract

This thesis presents the research and development of an application for recognition and measuring of a ferrosilicon tapping beam using an Infrared (IR) camera. Ferrosilicon production is a highly endothermic process, requiring large amounts of energy to operate; thus, the need for increased efficiency is constant. Using an IR camera for non-intrusive measurements of the tapping beam and developing computer vision algorithms that can identify distinctive process parameters are investigated in this project. Today a video stream from an IR camera overlooking the tapping area at Elkem Thamshavn is directly displayed in the tapping control room with no form of warning labels or data logging. Thus, the motivation for this research is derived from the need for more direct feedback on the tapping beam to the on-site human operators and data logging for the metallurgists. Developing a system that can detect temperature, the shape of the tapping beam and alert operators of unwanted situations is set as the main goals for this project.

An application that connects the IR camera to a series of algorithms written in Python and publishes the resulting data to a web-based Human Machine Interface (HMI) is developed and tested on an NVIDIA Jetson Nano single-board computer with an Optris IR camera. It incorporates OpenCV, NumPy, Numba, SQLite database, and GStreamer as computing and data communication tools. With testing on a tapping beam test stand, where the molten ferrosilicon is replaced with warm water; the application has proven to give feedback on the beam shape, temperature and status of the tapping beam through the web-based HMI.

Preface

The work presented in this master's thesis has been performed as the final project in a two-year master's degree in Mechatronics at the University of Agder, Department of Engineering Sciences. It covers multiple elements within Mechatronics; programming of single-board computers, computer vision, camera technology and application development. The project has demanded dedication, resilience and structure, which have been challenging and highly rewarding.

Elkem has contributed to the project with the necessary equipment and expertise on the tapping process and production of ferrosilicon. Throughout the conduction of this master's project, multiple meetings have been held with the representatives at Elkem and arising questions have been quickly dealt with, and the correct information evoked. This has resulted in a project that complies with the challenges that are of interest for Elkem to solve and the requirements from the university regarding content and field of research.

And finally, a special thank you to the supervisor on this project, Sondre Sanden Tørdal, for valuable assistance and guidance.

Grimstad, Norway 2019-05-27

Jørgen Nilsen

Jørgen Nilsen

Table of Contents

List of Figures

List of Tables

List of Abbreviations

AC Alternating Current. 8, 10, 12
AI Artificial Intelligence. 3, 61
API Application Programming Interface. 36

BGR blue, green and red. 39
BOF Basic Oxygen Furnace. 6

DLL Dynamic-link Library. 37, 38
dpkg Debian package. 37

HMI Human Machine Interface. 4, 5, 21, 22, 43, 46, 48–50, 58, 61, 63, 64
HTML Hypertext Markup Language. 50, 58

IR Infrared. 4–7, 15–17, 21, 35–39, 43, 46–48, 50, 51, 53, 54, 58, 60, 61, 63

JIT Just-In-Time. 19, 20, 53, 57, 62

NumPy Numerical Python. 18, 19, 36, 40, 43, 57, 62

OOP Object-oriented programming. 18
OpenCV Open Source Computer Vision Library. 21, 24, 36, 39, 53, 60–63

PER Plan, Execute and Reflect. 33
PIF Industrial Process Interface. 35, 36
PLC Programmable Logic Circuit. 35, 36

R&D Research and Development. 2, 6
RANSAC Random Sample Consensus. IV, 28–30, 42, 43, 56, 57, 60, 61, 63, 64
ROI Region of Interest. 7, 44, 54, 60, 63

SAF Submerged Arc Furnace. 3, 8, 9
SDK Software Development Kit. 36, 37

UN United Nations. 2
USB Universal Serial Bus. 35, 36, 53, 61

VPN Virtual Private Network. 60

List of Nomenclature

Constants

π	Pi	3.141592

Units

V	Volts
Hz	Hertz
s	Second
m	Meter
ϵ	Emissivity
η	Fill factor
C	Celsius
K	Kelvin

Chapter 1

Introduction

1.1 Brief history of Elkem and silicon production

Elkem was founded in 1904 by the industrial entrepreneur Sam Eyde, with the intention of developing technology and industry based on the natural resources in Norway. Over 110 years, Elkem has played a vital role in the development of the Norwegian process industry and technology. Covering aluminium production, the invention of the Söderberg electrode and the position as a world-leading producer of silicon and other special alloys [1]. Today, Elkem consists of three business divisions. Silicones, a fully integrated silicones producer. Silicon Products, a provider of silicon, ferrosilicon, foundry alloys, micro-silica and related speciality products. And finally, Carbon Solutions, a supplier of electrode paste and speciality products to the ferroalloy, silicon and aluminium industries [2].

Sustainable development is central to Elkem's production strategy and a defining factor in conducting their business and production development. New technology is implemented for developing products of higher quality and increasing production efficiency. Using less energy to create the best possible products. Elkem has developed its sustainability goals in line with the principles of the United Nations (UN) Global Compact, supporting the ambitions of the Paris climate agreement and the UN Sustainable Development Goals [3].

Silicone-based materials and products are particularly common and essential in embedded electronics, the healthcare industry, industrial processes and energy production. In 2013, 2.1 million tonnes of silicone products were sold on a global basis, yielding a turnover of $11 billion and around $40 billion in downstream added value [4]. The silicone production industry maintains a high focus on innovation and implementation of new technology, investing approximately 4% of the global turnover into Research and Development (R&D) [4].

1.2 Motivation

In the process industry, the implementation and development of new technology have always been a defining feature and a necessity for increasing production efficiency, quality and plant safety. Elkem has since its origin placed a focus on technology, with the development of the Söderberg electrode as an example. With the advancements in computer science, Artificial Intelligence (AI), and sensor technology; information and process data availability allows for a higher level of understanding and real-time status on previously immeasurable operations. Many parameters in the production of Elkem's products are not directly measurable due to high temperatures in and around the furnace. Traditional sensors that need to be in direct contact with the measuring medium will not survive over extended periods. Non-intrusive measurements, measure process parameters without disturbing the process and are the appropriate method for extracting information from a ferrosilicon furnace.

Ferrosilicon is produced in a Submerged Arc Furnace (SAF), where raw material is added from the top, and the three electrodes reduce the mass down to the resulting molten ferrosilicon. The resulting alloy is retrieved through a spout on the side of the furnace, in a process called tapping. When tapping is initiated, the hole above the spout into the furnace, is manually opened by drilling. The molten alloy is then poured into a ladle; a vessel that transports the molten alloy. The process of tapping molten metal into the ladle is a fairly non-complex task but provide valuable information about the furnace operation and the produced ferrosilicon.

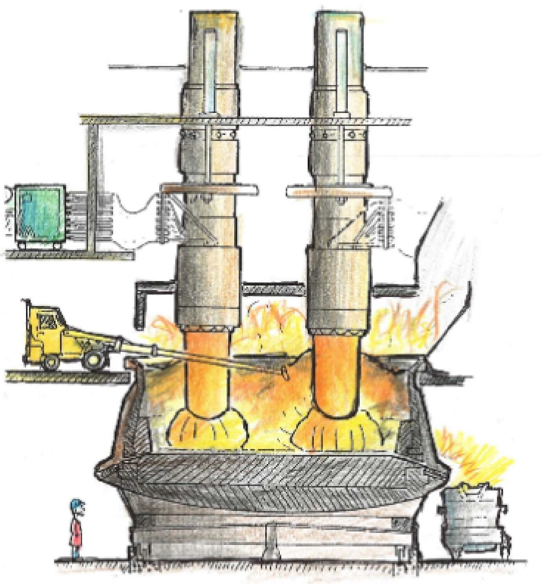


Figure 1.1: Drawing of a silicon furnace [5].

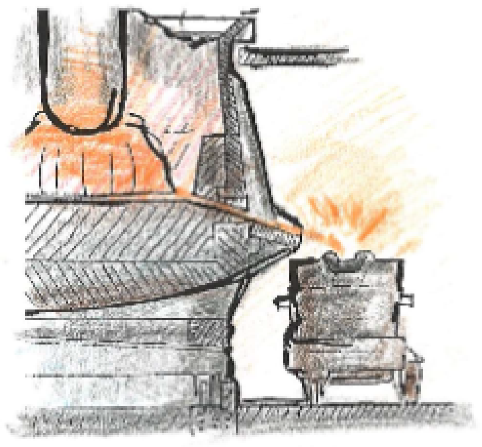


Figure 1.2: Drawing of the tapping process [5].

Sensors that can provide a means of non-intrusive measurement in this process is thermal imaging cameras and pyrometers. They both offer temperature measurement, but the thermal camera can be used in combinations with computer vision algorithms to extract information about tapping beam shape, slag detection and temperature at specific regions. Providing the process operators and metallurgists with as much information as possible about the process and the post tapping conditions, increases the final product quality.

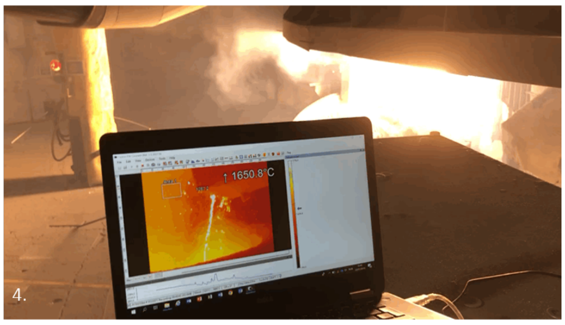


Figure 1.3: IR camera installed at Elkem Thamshavn oven nr. 2 [6].

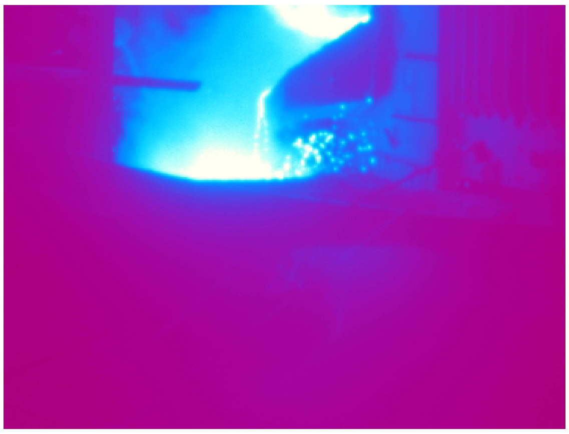


Figure 1.4: Thermal image from the tapping beam at Elkem Thamshavn oven nr. 2 [6].

As of the time of this project, an Infrared (IR) camera is installed at Elkem's plant in Thamshavn, Orkanger, providing a display of the tapping beam, ladle and surrounding area. The video feed is displayed on a separate monitor in the tapping control room, not directly connected to the operator's control and information system. Therefore, alerting the operators of unwanted and potentially dangerous conditions and logging information about the tapping beam temperature is highly sought-after.

Dangerous and unwanted conditions in the tapping process are listed as:

- Clogging of the tapping hole: Tapping beam flow ceases due to clogging of the furnace tapping hole.
- Inverted tapping beam: Tapping flow does not separate from the furnace spout; instead, it flows down and along with the spout, potentially hitting the ladle lining.
- Detect separation of tapping beam: The tapping beam is split into multiple smaller beams, increasing the surface area and lowering the overall temperature.

1.3 Project scope and objectives

This project aims to solve the challenge of using an IR thermal imaging camera for sensing parameters from an industrial process, specifically the tapping beam of molten ferrosilicon from the furnace into the ladle. Existing software for image processing, application development, and database communication is implemented as tools to help develop detection algorithms and display the parameters of interest. An IR camera is implemented as the sensor for non-intrusive measurement om the tapping beam.

Elkem has stated their expectations for this project, summarised as follows:

- Measure metal temperature in tapping beam.
- Measure the tapping beam profile.
- Give feedback on the shape of the tapping beam; is it a solid continuous beam or sparse and disrupted.
- Give feedback if the tappingbeam is no longer present, i.e. the tap-hole is clogged.

To help facilitate the above mentioned goals, further tasks are implemented:

- Build an intuitive cross-platform web-based Human Machine Interface (HMI) for interfacing the technology and presenting the operator with important values and video stream.

1.4 Limitations

Limitations for the work in this master's project. The main limiting factor in this project is the timeline, with a starting date of January 5^{th} and a submission date set to the 28^{th} of May, affecting the overall scope and possible workload. Throughout the project, simplifications of the process had to be made to develop a system that solves the objectives within the proposed timeline. For example, building a test stand of the tapping process that uses hot water as the medium was necessary as implementation throughout the project's development phase was impossible. Planning for a project of this magnitude and complexity also proved to be a challenge, as the author of this project had little experience with computer vision systems, process industry, IR camera and Python programming.

1.5 Source code repository

The application source code developed in this project is available as a GitLab repository. The Python code for the computer vision algorithms and the HMI is also included in the appendix C.1 and C.2.

https://gitlab.com/gurgle96/elkemvision/

1.6 State-of-the-art

Using an IR camera to sense parameters from an industrial environment is not a novel implementation. The British army was the first to develop a stand-alone camera system for detecting objects based on their heat (infrared radiation) [7]. This camera system was developed in 1929, and its sole purpose was to detect enemy aircraft. Since then, the IR camera development was mainly conducted through military R&D, resulting in systems for portable night vision and naval firefighting equipment. Today the IR technology is used in various industries, including; law enforcement, search and rescue, process industry and building inspection [7]. Regarding this thesis, implementations and research on IR thermal camera for detecting equipment-wear levels and process parameters are evaluated and put into context with this thesis scope and objectives.

Following the de-classification of IR camera technology in 1992 and the concurrent development of new detector technology, IR sensors and cameras became much more affordable and efficient. The IR detector's previous design and construction required the detector to be actively cooled, adding complexity and cost. At the end of the 1990s, these detectors were designed around Barium Strontium technology, which does not require cooling. Thus, significantly lowering the production complexity, cost and physical size [7]. Moving into the 21st century, where a higher focus has been placed on industrial efficiency, predictive maintenance and safety, the usage of IR cameras accelerated.

An industrial application of IR camera technology for predictive maintenance and fault detection is demonstrated in article [8]. In this article, a 3-phase electric motor, commonly used in industrial applications, is monitored with an IR camera to develop fault detection algorithms and techniques. The authors of this article evaluated several different signals for estimating the electric motor behaviour, such as mechanical vibration, stator current, acoustic signals, and thermal images. Closing in on the thermal image due to the availability of information and noise-free behaviour. To diagnose the electric motor, a series of neural network algorithms were developed and trained. The results of the research in this article was a system for detecting faults and give diagnostic feedback on a 3-phase electrical motor, categorising the motors into three separate categories; (1) healthy, (2) two broken rotor bars and (3) faulty squirrel-cage ring. This article also emphasises the use of neural networks for algorithm development, given that there is, or it is possible to create, data sets of reasonable size for training.

In the process industry, especially the metal production industry, thermal power is a common energy source and contains vital parameters about the raw materials, equipment and final product. In article [9], detecting slag in the tapping stream of steel from a Basic Oxygen Furnace (BOF) is investigated. Slag is a byproduct of the reduction process in the furnace; it floats on top of the molten metal and protects it from further oxidisation. When poured from the furnace into the ladle, slag is present in the stream and increases from 0 - 100 percent as the tapping process continues. Therefore, having an understanding of the amount of slag in the ladle is essential for further processing and alloying. In this article, the authors developed a system for monitoring the tapping beam and detecting slag based upon the difference in infrared radiation between slag and the desired metal product. A metal temperature probe mounted inside the BOF is also incorporated to calibrate the IR camera. Furthermore, the authors developed methods for calibrating the threshold temperature and tracking the tapping beam; these methods are described in the points below.

- Temperature threshold calibration: To separate the slag from the tapping beam, the temperature matrix from the IR camera and data from the internal temperature probe in the furnace is used. When tapping is initiated, and the flow of metal is slag-free, the IR camera temperature of the tapping beam is compared to the molten steel temperature from the probe inside the furnace, and the emissivity of steel is calculated with equation 1.1. The calibrated emissivity value for steel is used to give a precise representation of the tapping beam temperature, and while the beam is slag free the maximum IR camera temperature is recorded. This value is the threshold temperature between slag and molten metal in the tapping beam.

$$\epsilon = \left(\frac{T_{infrared}}{T_{probe}}\right)^4 \tag{1.1}$$

- Tracking of the tapping beam: The Region of Interest (ROI) where the tapping beam is located is found by horizontally iterating through the temperature columns, from left to right and from right to left. Finding the vertical edges of the tapping beam based upon the threshold temperature value. A square bounding box can then be defined around the tapping beam, with top and bottom height values manually defined.

With the research in the articles mentioned above, it is clear that using an IR camera for non-intrusive measurement of process parameters and extracting data not visible to the human eye is not a novel development. Thus, the existing research motivates this thesis, despite clear challenges and differences in this specific implementation.

In this project, the focus is not on closed-loop control of the ferrosilicon furnace, with feedback from the tapping beam. This is primarily due to the high degree of complexity in the ferrosilicon process. Despite this, the project is leaned towards extracting information about the tapping process that is of value to both on-site operators and the metallurgists for a safe and economic tapping operation and correct post-processing of the tapped metal. An IR camera developed by Optris GmbH is mounted at one of Elkem's ferrosilicon plants and monitors the tapping process. Today this video feed is only viewed by the on-site operators and not implemented in any algorithm.

Chapter 2

Theory

2.1 Production of ferrosilicon

In this section, an overview of the metallurgical processes and equipment used in ferrosilicon production is described. The process of turning raw materials into ferrosilicon, basic construction of a Submerged Arc Furnace (SAF) and the arrangement of the Söderberg electrodes, are explained to better understand the complexity of the process.

2.1.1 Producing ferrosilicon

Ferrosilicon and other ferrous alloys are produced by carbothermic reduction of the added raw materials in a SAF. Mineral ores and carbon are added as charge material into the furnace, converted to the desired alloy through carbothermic reduction. The reduction process of charge material is highly endothermic, thus requiring a large amount of thermal energy. The thermal energy is converted from electrical energy by three self-baking Söderberg electrodes through Alternating Current (AC) arcing. Where heat is generated in the AC arcs that span between the electrode tip and the molten alloy bath. The electrodes are electrically arranged in a star connection, with the transformers supplying the stepped-down voltage arranged in a delta connection. Each electrode is connected to one phase.

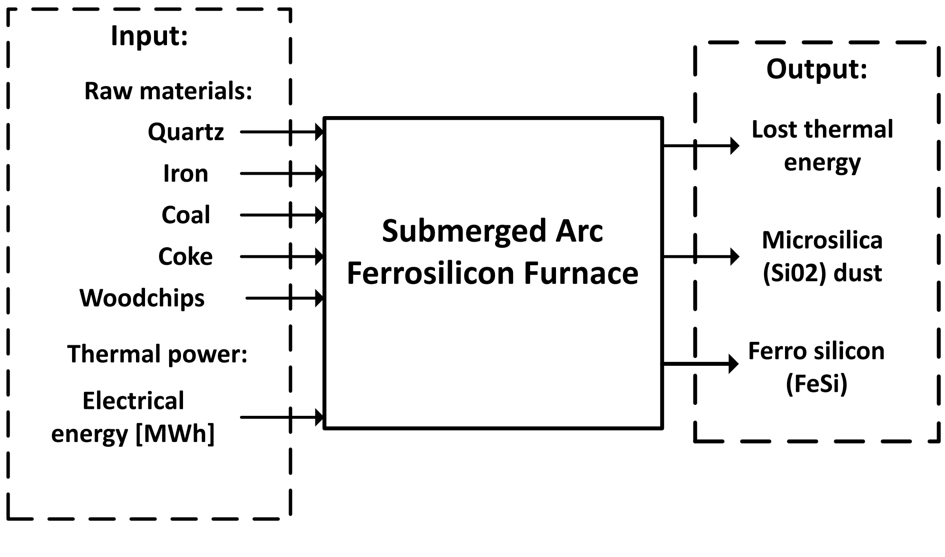


Figure 2.1: Process block of the Ferrosilicon process

Figure 2.2 describes the process of creating ferrosilicon from raw material to the tapping process. Starting with piles of raw material; quartz, iron, coal, coke and wood chips, loaded onto conveyor belts, refined and then stored in silos, ready to be discharged into the furnace. Inside the furnace, the three Söderberg electrodes creates electric arcs that generate the necessary thermal energy for the raw materials to reduce and combine into ferrosilicon.

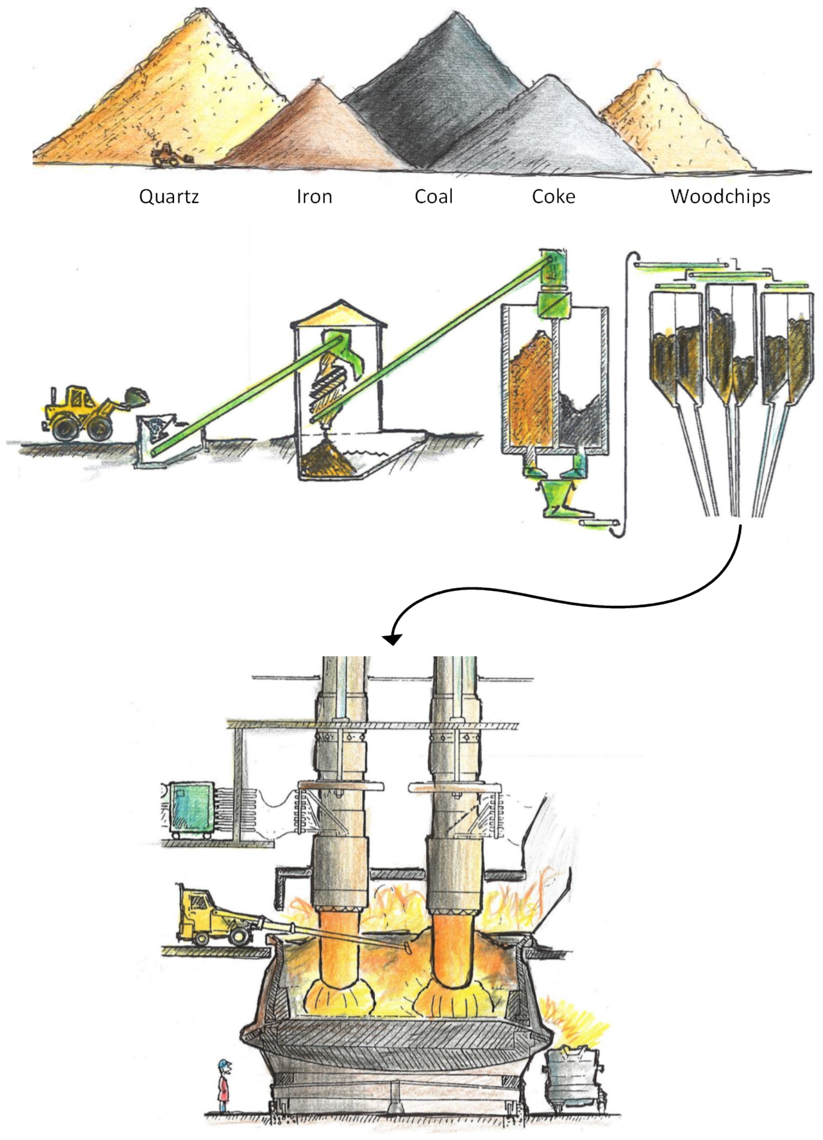


Figure 2.2: System overview of the ferrosilicon process [5].

2.1.2 Furnace construction

At the Elkem plant in Thamshavn, oven nr. 2 is designed and constructed as a typical SAF. The furnace is constructed as a cylindrical body, with an outer steel casing providing the structural rigidity and carbon blocks insulating the furnace to withstand the internal process temperatures. Tapping spouts that transfer molten silicon out from the furnace are positioned on the outside circumference of the furnace, towards the bottom. Around the furnace, there are tracks that move the ladle around with the furnace as it rotates. The furnace rotates around its centre axis to reduce the formation of cavities in the charge material, using the electrodes as stirring sticks. It takes several hours for the furnace to complete one full rotation, and the tapping process is done within approximately 2 hours. The tapping beam is still within the same angular quadrant from start to finish, continuously in line of sight for the tapping operators.

Transporting the molten metal away from the inside of the furnace is done by tapping into a ladle. A ladle is the containment vessel for the tapped ferrosilicon, constructed from many of the same materials as the inside of the furnace. The tasks of the ladle are to be positioned under the tapping spout, be filled up with molten ferrosilicon and contain this molten metal as it is transported over to the casting moulds. After casting, the molten metal cools down and hardens. When it reaches a specific temperature, it is removed from the moulds and crushed into finer particles. The tapping process is further explained in section 2.2.2.

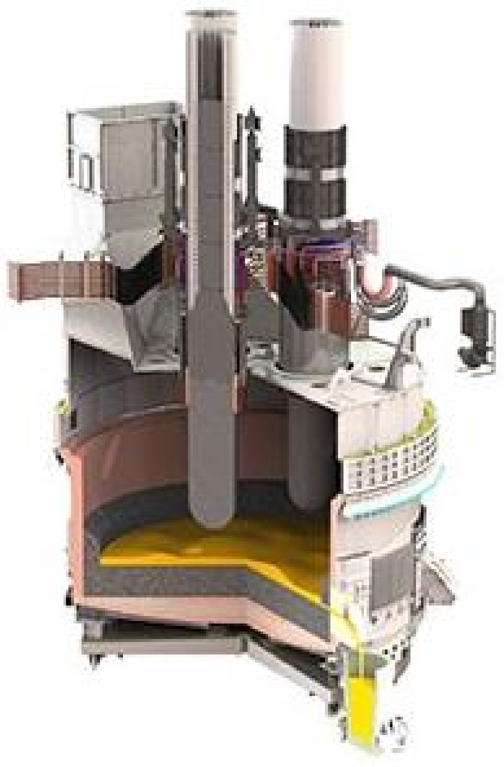


Figure 2.3: Furnace construction[10].

2.1.3 Electrical arrangement

Thermal energy is necessary for the carbothermic reduction process to take place; the most efficient way to produce this is to convert it from electrical energy. The electrical energy is converted to heat through AC arcing. Three Söderberg electrodes are positioned in a triangle, each connected to one phase of the system in a delta-star arrangement. The three electric arc spans from the tip of each electrode into the molten pool of ferrosilicon, directly heating the metal and surrounding charge. Some thermal energy is also generated in the upper parts of the electrode due to the electrical resistance, and the surrounding charge material absorbs this thermal energy. A simplified model of a typical electrical arrangement of the connected electrodes can be seen in figure 2.4 along with a detailed electrode cross-section in figure 2.5.

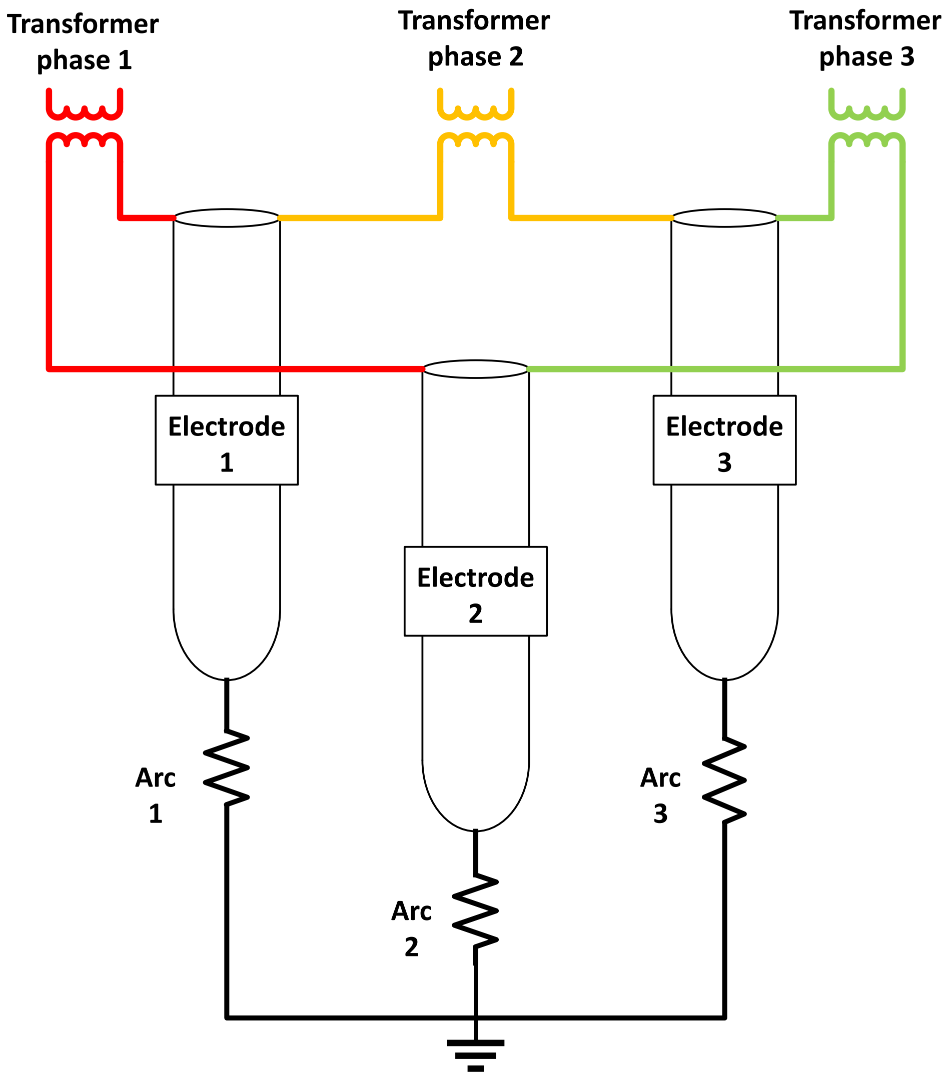


Figure 2.4: Electrical arrangement of electrodes.

The Söderberg electrode is composed of an outer steel casing, electrode paste, electrical slip-rings and height adjusting cylinders. The steel casing contains the paste and provides structural rigidity to the electrode. The electrode paste is a substance consisting of calcined anthracite, petroleum coke and pitch as a binder. Initially the paste is loaded into the electrode as hard briquettes, it is then liquefied as the temperature increases. Once the temperature reaches about $500[^{\circ}C]$, the paste is baked; transitioning from a soft viscous substance into the hard electrode tip that generates the AC arc [11]. During nominal operation, the phase voltage of the electrode is kept constant and the current flow in the electrode is regulated by the height between the electrode tip and ferrosilicon bath, explaining the need for the height adjusting cylinders [12].

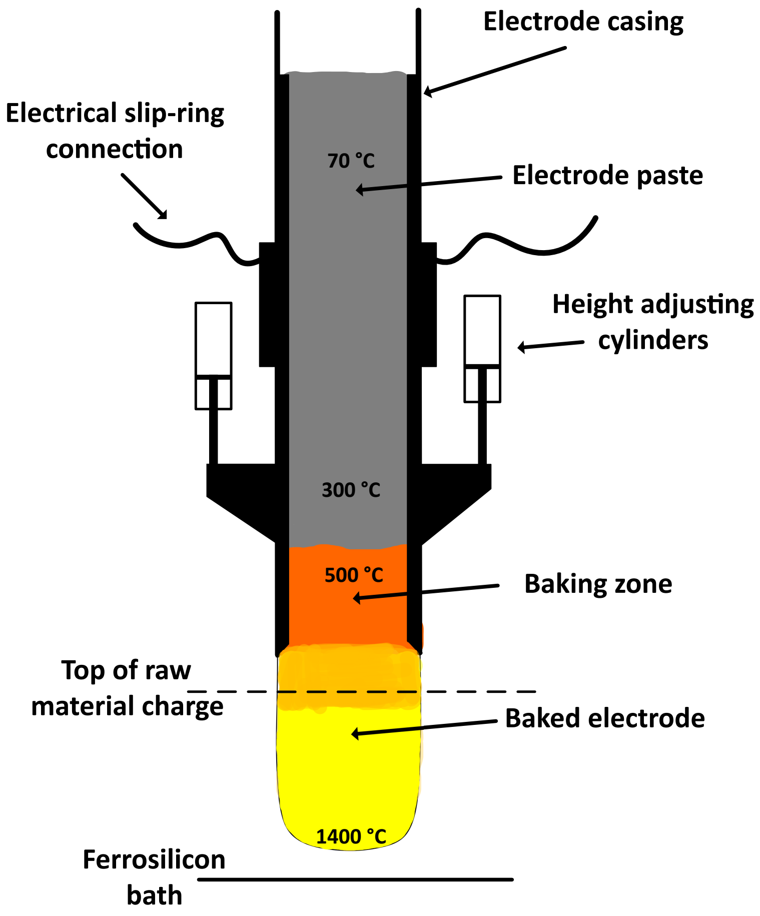


Figure 2.5: Section view of a Söderberg electrode

2.2 Tapping process

In this section, the theory and knowledge surrounding the process of tapping molten ferrosilicon from the furnace to ladle are explained. The working environment and challenges that the operators face in and around the tapping area and the discrete tasks that they complete, are presented.

2.2.1 Human operators

The process of creating ferrosilicon is a hybrid of automated systems and manually executed tasks. The integration and synchronisation of these systems are vital to sustain optimal performance of the furnace and to ensure the best possible product. The post-tap-hole operations that are done manually, can be summarised as follows:

- Temperature measurements of the tapped ferrosilicon.
- Material sampling.
- Monitoring the level of ferrosilicon in the ladle.
- Opening and closing of tap-hole.
- Cleaning and maintaining the tapping spout.
- Adding material to alloy the raw ferrosilicon.

The environment in the tapping area is described as a hazardous environment, due to high temperature radiation generated by the furnace and the risk of molten metal splashing during the tapping process. Figure 2.6 shows how close the operators are to the process. The equipment used to perform sampling and testing of the tapping beam is designed in such a way that the operators can stay within reasonable distance, an example are long reach lances and temperature probes used to condition and monitor the tapping beam during operation. The workers also wear protective clothing that reflects most of the radiant heat and can withstand minor splashing of slag/liquid metal.

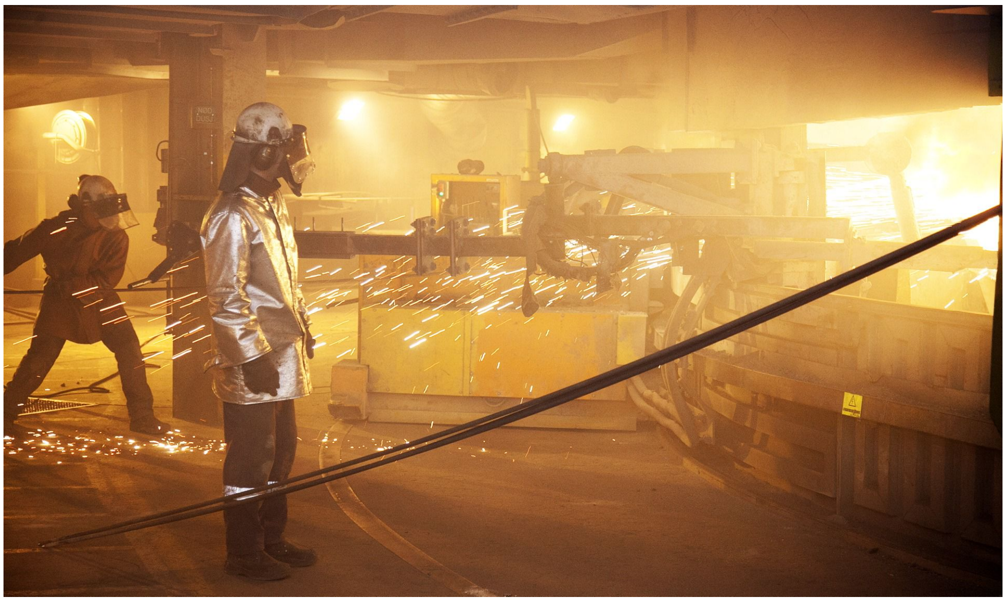


Figure 2.6: Operators working in the tapping area at Elkem Salten [13].

2.2.2 Tapping timeline

The tapping process is initiated by manually drilling open a plug in the furnace, creating a direct flow-path for the molten metal into the tapping spout. Operators must continuously monitor the tapping beam and treat the spout and hole to prevent clogging and improper flow. Tapping of ferrosilicon into the ladle is done with certain interval's, as the internal level of ferrosilicon in the furnace must be brought back. One tapping process elapses over approximately 120 minutes and throughout this time a series of tasks are executed, as mentioned in section 2.2.1. Terminating the tapping process is done by inserting a plug made of clay, the plug rapidly solidifies due to the high temperature and blocks any more ferrosilicon to exit the tap-hole. Furthermore, the flow-chart in appendix B.1 describes the timeline for the tapping-process and the details surrounding each task.

2.3 Infrared thermal camera

In this section, the theory regarding the IR camera is presented. Explaining the basics of infrared radiation, where it is placed in the electromagnetic spectrum and how the wavelengths are classified as temperature regions. The detector element in an IR camera is also presented, explaining the design and working principle.

2.3.1 Infrared radiation

A body with a temperature above absolute zero ($-273[°C]$) emits radiation in electromagnetic waves, scientifically called IR radiation. Figure 2.7 shows the spectrum of electromagnetic radiation and classifies the known sources of radiation. In the electromagnetic spectrum, the IR region is located to the right of the visible region, covering wavelengths from $0.7[\mu m]$ to $1[mm]$. Therefore, increasing the temperature of the emitting body yields a higher intensity of radiation, and IR waves with high energy are denoted by short wavelengths (close to $0.7[\mu m]$). However, IR is not visible to the human eye, requiring special sensors and cameras to be detected[14].

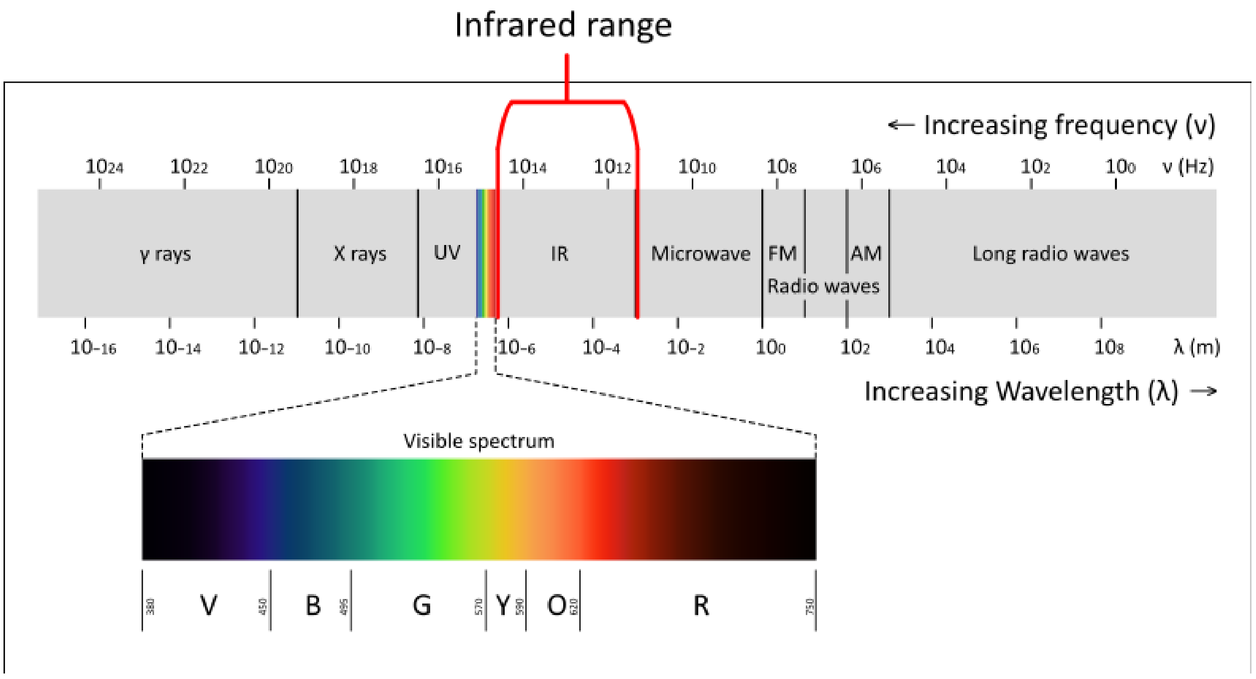


Figure 2.7: The electromagnetic spectrum [15].

The IR range can be divided into 5 categories [14]:

1. Near infrared: Wavelengths of $0.7[\mu m]$ to $1.4[\mu m]$, emitted from bodies with temperature between $3600[°C]$ and $1800[°C]$.
2. Short-wave infrared: Wavelengths of $1.4[\mu m]$ to $3[\mu m]$, emitted from bodies with temperature between $1800[°C]$ and $700[°C]$.
3. Median infrared: Wavelengths of $3[\mu m]$ to $8[\mu m]$, emitted from bodies with temperature between $700[°C]$ and $90[°C]$.
4. Long-waved infrared: Wavelengths of $8[\mu m]$ to $15[\mu m]$, emitted from bodies with temperature between $90[°C]$ and $-80[°C]$.
5. Far infrared: Wavelengths of $15[\mu m]$ to $1[mm]$, emitted from bodies with temperature between $-80[°C]$ and $-270[°C]$.

2.3.2 Infrared camera

To measure the IR radiation and temperature of a body and display this information as an image, a camera with a sensor that can detect the IR radiation is necessary. IR camera functions similarly to digital cameras. They have an optical lens, field-of-view angles, focal length, and a detector array that converts the focused beam of radiation into electrical signals. An IR camera's detector array consists of pixels, ranging in count from 20 000 to 1 million. A microbolometer is located in each pixel, and this is the detector element that changes its electrical properties when exposed to infrared radiation. Each microbolometer element is approximately $17[\mu m]$ x $17[\mu m]$ to $35[\mu m]$ x $35[\mu m]$ in size [16].

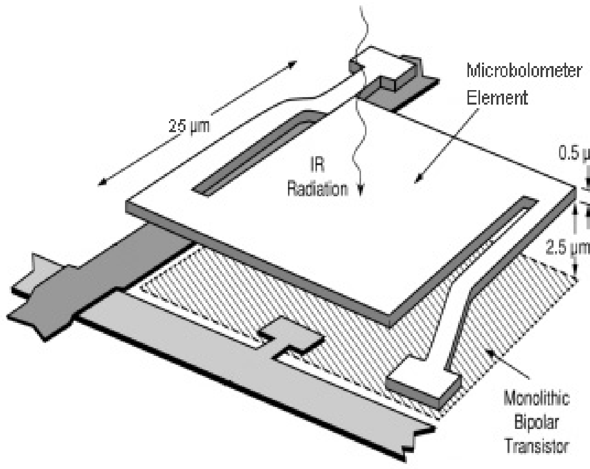


Figure 2.8: A typical microbolomter [17].

Figure 2.8 shows the composition of a typical microbolometer. When hit with IR radiation, the microbolometer element increases in temperature and thus, the electrical resistance of the element changes. Figure 2.9 further explains the key elements and their working principle. When the IR element is subjected to IR radiation, the absorbed radiation is converted to thermal energy as it impacts the absorbing element. The thermal energy is then transferred into the heat-sink through the support link, and a change in temperature over the support link is measured. From equation 2.1, the absorbed power from the radiation is calculated, given that the thermal conductance between the absorber and the heat-sink is G_t.

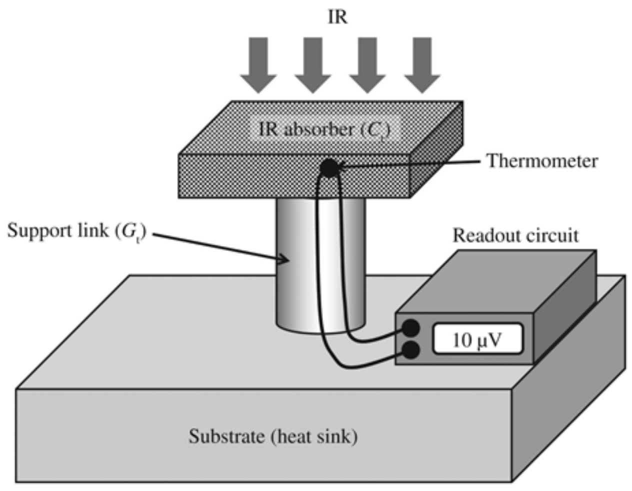


Figure 2.9: Working principle of a bolometer [18].

Equation 2.1 calculates the absorbed power in the bolometer circuit, based upon the change in temperature between the absorber element and the heat-sink. Where equation 2.2 uses the emissivity and fill-factor of the bolometer to calculate the actual amount of IR radiation on the absorber element, since some of the radiation is reflected of the pixel due to the physical properties of the microbolometer. Where the fill-factor (η) indicates the relationship between IR sensitive area to the total pixel area, and emissivity (ϵ) is the ability of the pixel to emit IR radiation. Figure 2.10, describes the readout circuit for an IR camera detector array.

$$P_{abs} = \Delta T G_t \tag{2.1}$$

$$P_{abs} = \eta \cdot \epsilon \cdot P_{in} \tag{2.2}$$

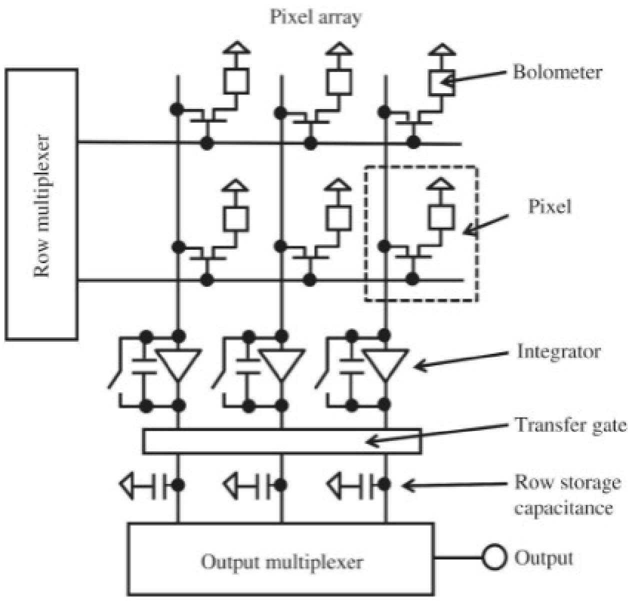


Figure 2.10: Readout circuit for a uncooled microbolometer array [18].

2.4 Software tools and packages

In this section, the theory regarding the programming language, software and other packages used to develop the source code and application are explained. Some examples of usage and implementation are also shown.

2.4.1 Python

Python is an efficient, easy to learn, high-level programming language constructed around the principles of Object-oriented programming (OOP). Compatible with an abundance of modules and packages from the open-source community. Python is suitable for rapid development of applications and combining other pieces of working software in either C, C++ or Python. These features make it an attractive programming language for developers and novices in programming, hence making it suitable as the code language in this project. Also, Python omits the compilation step, making the edit-test-debug process remarkably efficient [19].

Figure 2.11: Python logo [19].

2.4.2 Numerical Python

Numerical Python (NumPy) is a library package developed specifically for scientific computing in Python. It provides data types for multi-dimensional array objects and a wast assortment of routines for array operations. For example; linear algebra, discrete Fourier transforms, sorting and selecting, to mention a few. At the core of the NumPy library is the `ndarray` object; this object encapsulates n-dimensional arrays of homogeneous data types and performs many operations in compiled code for increased performance [20].

```python
# Python NumPy example

import numpy as np

a = np.array([1, 2, 3],[4, 5, 6],[7, 8, 9])

b = np.array([1, 1, 1],[1, 1, 1],[1, 1, 1])

c = a*b

d = a+b

e = np.append(a,b)
```

2.4.3 Numba compiler

In cases where Python and NumPy code does not achieve the appropriate computational frequency or uses too much processing power, optimizing the code by compiling it to C/C++ will give the code better performance. For Python and NumPy, Numba is an open-source Just-In-Time (JIT)-compiler, which translates Python code into faster machine code using the LLVM compiler infrastructure [21]. Figure 2.12 shows the general structure that Numba follows to JIT compile Python code, only a simple `@jit` decorator is required to compile a function. The code below shows two different implementations of Numba and JIT, one for a single function and one for an entire class, using the `@jitclass` decorator from `numba.experimental`. When compiling a function or class, it needs to be initiated with a set of data similar to the data it will be given when used after compilation.

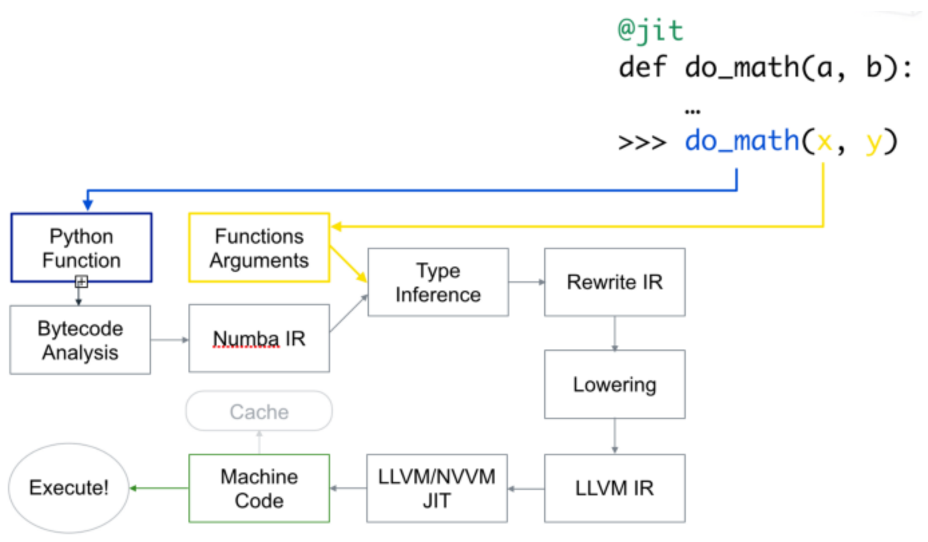


Figure 2.12: Process for compiling Python code with the JIT decorator [22].

```python
from numba import jit
from numba.experimental import jitclass
import numba as nb

# JIT compiled function

@jit
def randomfunction(x):
    # your loop or numerically intensive computations
    return x

# JIT compiled class

# Specify the shared parameters within the class and their datatype
spec = [('self_parameter'), nb.self_parameter_datatype]

@jitclass(spec)
class randomclass(object):
    # functions to be JIT-compiled
```

A simple example shows the effect of JIT compilation on other wise computational heavy code.

```python
from numba import jit
import numpy as np
import time

# Define dataset
x = np.arange(100).reshape(10, 10)

# This function will be compiled with JIT.
@jit(nopython=True) # Set "nopython" mode for best performance,
    equivalent to @njit
def go_fastJIT(a): # Function is compiled to machine code when called
    the first time
    trace = 0.0
    for i in range(a.shape[0]):   # Numba likes loops
        trace += np.tanh(a[i, i]) # Numba likes NumPy functions
    return a + trace              # Numba likes NumPy broadcasting

# This function will not be compiled, it is only used for reference.
def go_fast(a):
    for i in range(a.shape[0]):
        trace += np.tanh(a[i, i])
    return a + trace


# This step compiles the function to machine code
go_fastJIT(x)


# Measure the execution time of the JIT compiled function
start_time = time.time()
a = go_fastJIT(x)
JIT_time = time.time() - start_time


# Measure the execution time of the regular function
start_time = time.time()
b = go_fast(x)
time = time.time() - start_time

# Print the execution times to terminal
print('JIT time = ',JIT_time)
print('Regular time = ', time)
```

After running this Python script, the execution times for the JIT-compiled function and regular function is printed to terminal. For this example the times where; `JIT time = 1.1682510375976562e-05` and `Regular time = 0.0002472400665283203`. This yields that the JIT compiled function is approximately 21 times faster.

2.4.4 OpenCV

Open Source Computer Vision Library (OpenCV) is a computer vision library that provides a broad range of image analysis and manipulation tools [23]. In this project, the Python implementation of OpenCV is used. An image is interpreted as a matrix, and each pixel in a grayscale image is represented by a value between 0 and 255. Grayscale images improve calculation speed and simplify the process since a it only uses one value per pixel instead of three in RGB. In addition, this allows for mathematical manipulation of the image, for example, in a video stream, where each video frame is handled as a separate image and processed in real-time.

Figure 2.13: OpenCV logo[23].

2.4.5 GStreamer

GStreamer is a library for constructing video-feed and multimedia pipelines that can link together complex processes and create efficient workflows for media and other data types. In a specific case where two applications each run in their own separate thread, where one application is a source, and the other is a sink, GStreamer can develop a local streaming server that handles the communication between the two. For this project, GStreamer is used to manage the video-streaming between a computational application that runs the IR camera and computer vision tasks, and a second application that hosts the HMI [24].

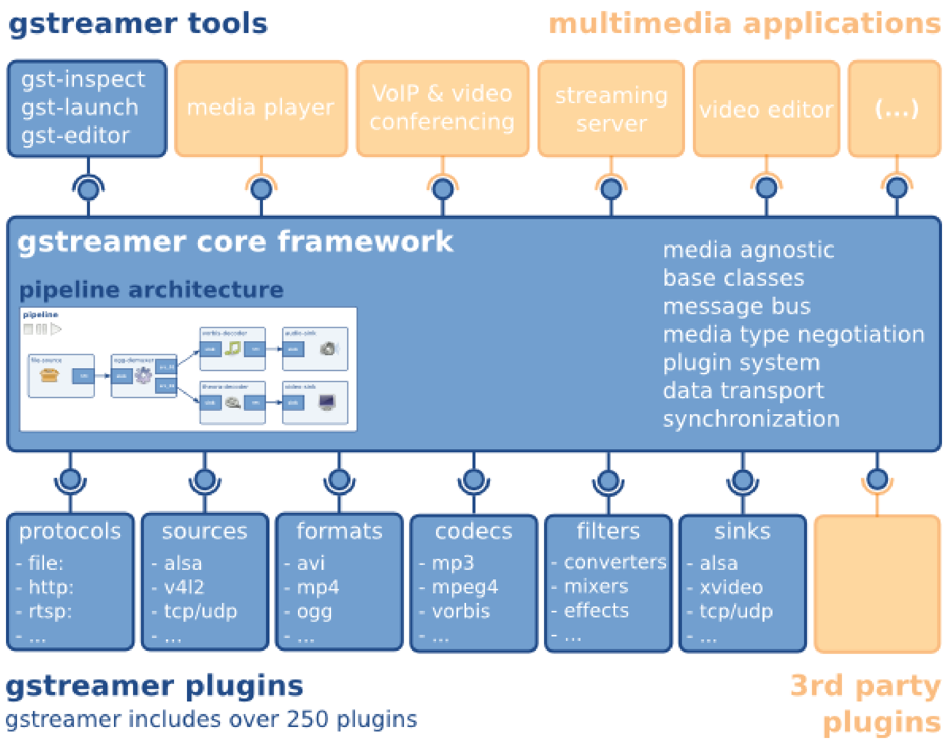


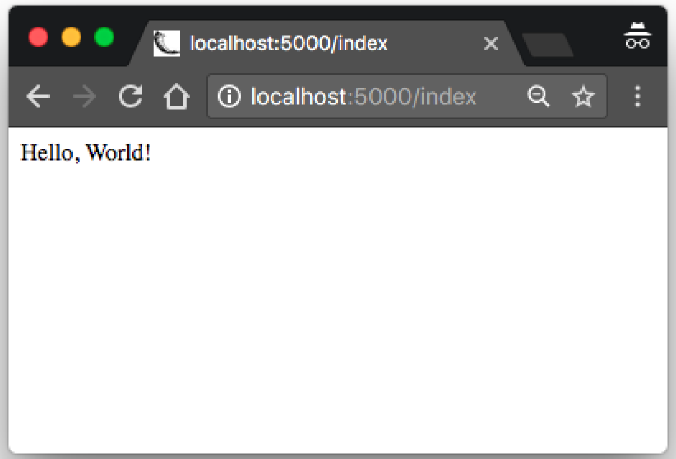
Figure 2.14: Gstreamer overview [24].

2.4.6 Flask application

Flask is a small and compact framework for developing web-hosted applications, varying in size and complexity. It is possible to create interfaces for hardware and display real-time information, host a blog or wiki. Flask is written in Python and makes it easy to give quite advanced Python applications a good looking interface. It is also possible to combine the framework with a database (SQLite), and front-end tool kit's such as *Bootstrap* [25]. For this project, it provides the framework for the HMI and creates the local server.

```python
from flask import Flask
app = Flask(__name__)

@app.route('/index')
def hello_world():
    return 'Hello, World!'
```



Hello, World!

Figure 2.15: Flask "Hello world!" example.

2.4.7 GitLab

GitLab is a web-based hosting service generally used for version control of code. Version control allows for saving and logging changes made to a project. With version control, it is possible to track all the specific changes made, when they were made, and who made them. This makes GitLab widely used for various projects, as it provides the ability to track potential errors that might occur and find out when they were made. For projects involving multiple people, GitLab offers everyone in a group to synchronize their code, making sure all participants are updated with the latest version. Moreover, if a repository is made public, it enables users to view code that has been created and take inspiration [26].

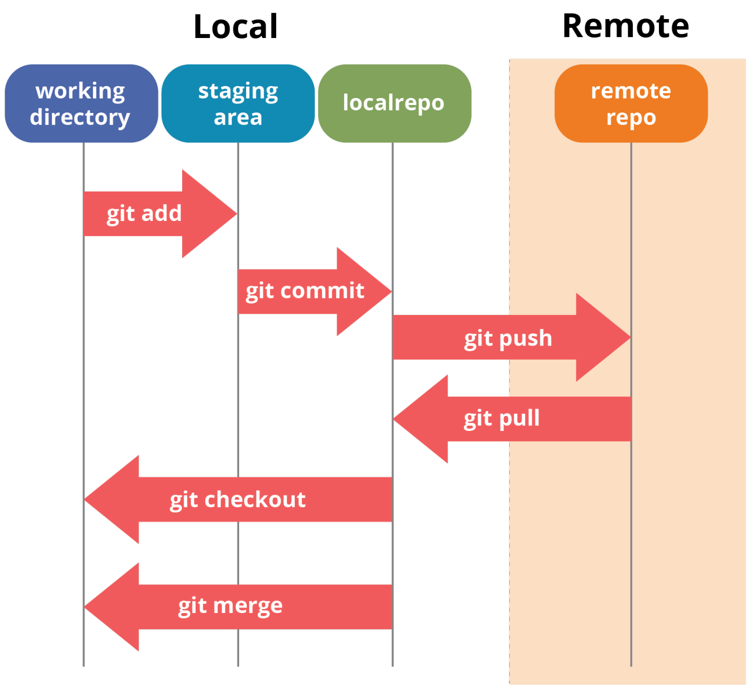


Figure 2.16: Git workflow [26].

2.5 Computer vision

In this section, the theory on computer vision and the algorithms developed in this project are explained. The mathematical principles behind the various functions and algorithms are explained, along with examples and figures to place the theory into context with research questions.

2.5.1 Locating features

To locate special features in an image, template matching is a popular option. It is quite simple to implement and requires minimal computational power when compared to regression with a neural network. For this thesis, it is chosen as the method for finding the position of the tapping beam. The theory behind this function is described in detail in the paragraphs below.

Template matching is the computational process where a template, typically a small image of a detail in a larger image, is used to find the position in the larger image that is most similar to the template. This function to locate specific regions is best implemented when the shape or form of the desired feature is kept more er less the same in the image stream supplied to the function. Within OpenCV there exists a function called: `cv.matchTemplate()`, that applies the common functions within template matching. Two-dimensional convolution is used to slide the template image across the larger image and compute the correlation between the template and the image patch. Within `cv.matchTemplate()`, it is also possible to choose between several different types of template matching operations. For this thesis, the function `cv.TM_CCOEFF_NORMED` is used, the mathematical expression for this function is described by equation 2.3.

Equation 2.3, normalised correlation-coefficient technique:

$$R(x,y) = \frac{\sum_{x',y'} (T'(x',y') \cdot I'(x+x',y+y'))}{\sqrt{\sum_{x',y'} T'(x',y')^2 \cdot \sum_{x',y'} I'(x+x',y+y')^2}} \tag{2.3}$$

Equation 2.4, iterate through every pixel in the template and subtract the mean template value from it:

$$T'(x',y') = T(x',y') - 1/(w \cdot h) \cdot \sum_{x'',y''} T(x'',y'') \tag{2.4}$$

Equation 2.5, iterate through every pixel in the image patch and subtract the mean image patch value from it:

$$I'(x+x',y+y') = I(x+x',y+y') - 1/(w \cdot h) \cdot \sum_{x'',y''} I(x+x'',y+y'') \tag{2.5}$$

Where:

$T(x',y')$: Grayscale value of template pixels.
$I(x+x',y+y')$: Grayscale value of pixels in the image patch.
x, y: Local coordinates of pixels in the larger image.
x', y': Local coordinates of pixels in the template.
x'', y'': Are coordinates for iterating across the template $x'' \in [0,\ w-1]$ and $y'' \in [0,\ h-1]$.
w, h: The width and height of the template.

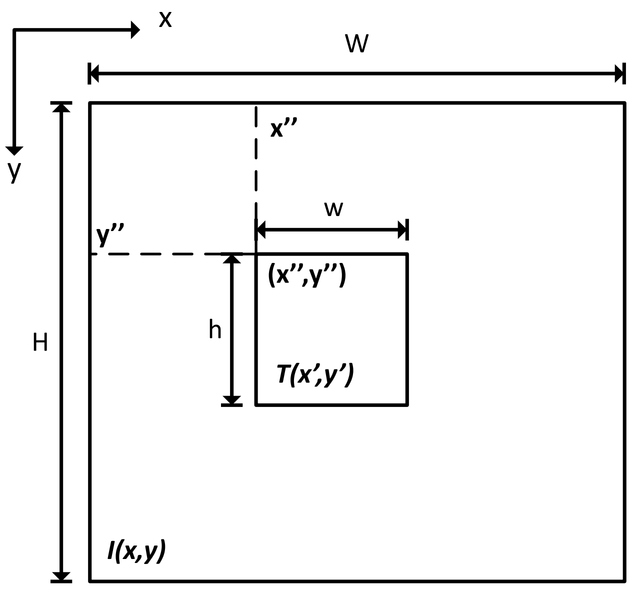


Figure 2.17: Coordinates used in template matching. $T(x', y')$ is the coordinates locally to the template and $I(x, y)$ is the coordinates locally to the larger image.

Equation 2.4 and 2.5, calculates the correlation between the template and the image as the filter is convoluted across the image. Only multiplying $T'(x'.y')$ with $I'(x', y')$ would yield a value between $-\infty$, 0 and ∞. Where $-\infty$ equals no correlation, 0 equals pixel values of zero, and ∞ equals a perfect match between the template and image. Since the function implemented is called `cv.TM_CCOEFF_NORMED`, the output value is normalised. Therefore, if $R(x, y)$ equals 1 we have a perfect match between image and template, 0 equals pixel values of zero and -1 equals no match. To select the image coordinates defined where the image and template match, a simple threshold is applied, and pixels with a higher value than the threshold are selected as the coordinates for the matching area.

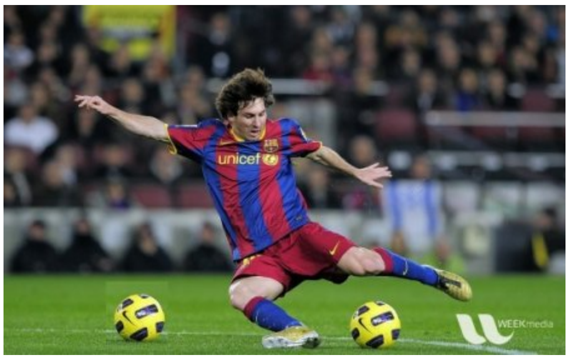


Figure 2.18: Example picture for template matching.

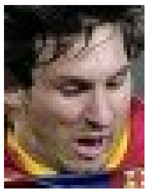


Figure 2.19: Template image extracted from figure 2.18.

An example of template matching is shown with figure 2.18 as the large picture and figure 2.19 as the extracted template. Using *cv.matchTemplate()* and *cv.TM_CCOEFF_NORMED*, the template correspondence to the image is calculated and shown in figure 2.20. The picture to the left is a grayscale image that describes the correlation between the image and template after using the *cv.TM_CCOEFF_NORMED*-method. Picture to the right is the detected face marked with a white boundary box. The bright spot visible in the upper part of the middle region in the left picture of figure 2.20, is where the correspondence is the highest and thus where the face is.

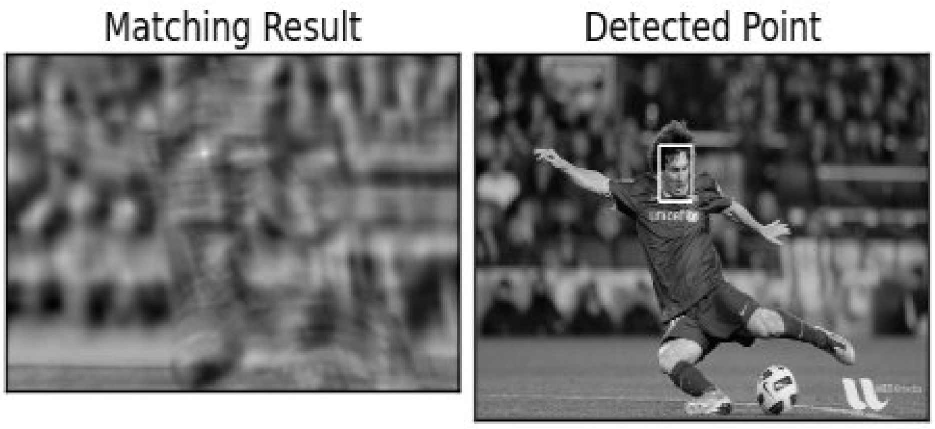


Figure 2.20: Applied template matching.

2.5.2 Detail enhancement with histogram equalisation

Enhancing the level of detail within an image can be done by increasing the contrast between bright and dark parts in the image, especially if the image is made up of a high concentration of similar pixels. The concentration of pixel values in an image is most commonly described with a histogram, a graph that shows the number of pixels of a specific value. Figure 2.21 shows the effects of equalisation on a histogram with such a narrow unimodal distribution, where the resulting histogram is transformed into a wide and flat distribution.

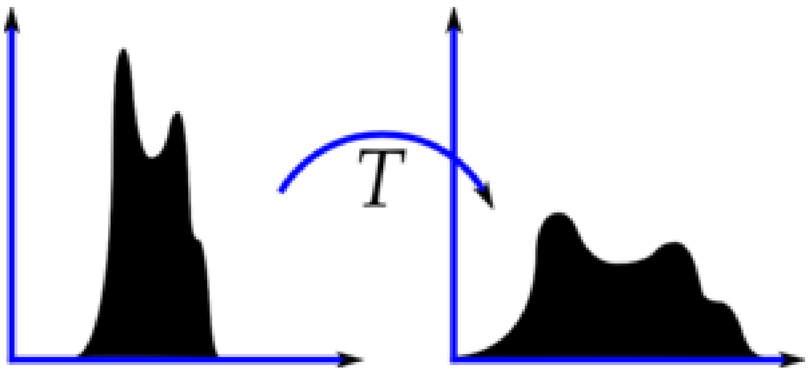


Figure 2.21: Effects of histogram equalisation [27].

$$H(j) = n_j,\ 0 \leq j < 255 \tag{2.6}$$

Where:

n_j: Number of pixels with graylevel j.

Calculate the cumulative distribution and find the max and minimum values of the histogram. Note that the minimum value for the histogram is not to be 0:

$$cdf_x(i) = \sum_{i=0}^{255} H(j = i) \tag{2.7}$$

$$cdf_{max} = max\ cdf_x \tag{2.8}$$

$$cdf_{min} = min\ cdf_x \tag{2.9}$$

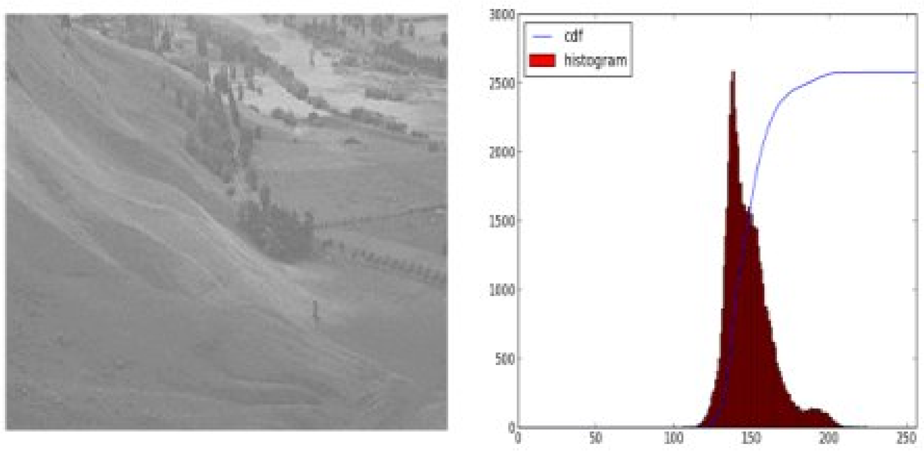


Figure 2.22: Normal histogram and the cumulative histogram [27].

To create an equalised image, it is necessary to transform the original image so that the cumulative distribution histogram of the new image would be linearised across the value range. See equation 2.10, where the cumulative distribution of the histogram is i, that is in the range of $[0, 255]$, multiplied with the constant K.

$$cdf_y(i) = iK \tag{2.10}$$

Finally, to create the new cumulative distribution histogram, equation 2.11 is applied.

$$cdf_y = round\left(\frac{cdf_x - cdf_{min}}{cdf_{max} - cdf_{min}} \cdot 255\right) \tag{2.11}$$

The resulting transformation from the histogram equalisation can be seen in figure 2.23. The histogram distribution is now spread over the grayscale range, and the cumulative distribution follows closely to a linear profile. By comparing the image in figure 2.22 to the image in 2.23, the results from histogram equalisation is clearly visible by the enhanced level of detail.

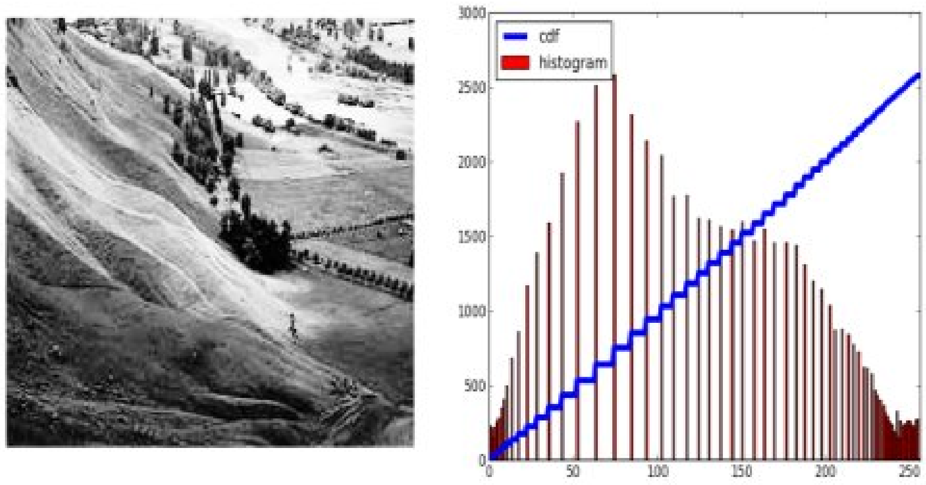


Figure 2.23: Histogram equalised image [28].

2.5.3 Model fitting and data filtering

Separating noisy data from robust signals is a common problem when working with all types of signals, and many approaches exist to separate noise from usable data. In this project, the signals are in the form of binary image arrays, basically a x-y plot of points. These points are either inliers or outliers, depending on their formation and what mathematical model that is to be fitted to the data. If the mathematical model is a linear system, $y(x) = ax + b$, a simple approach would be to use the least-squares algorithm. However, Least-squares uses all data points to establish the mathematical model, thus the noisy invalid data is biasing the result, see figure 2.25. A better and more robust approach is to use a iterative method called Random Sample Consensus (RANSAC).

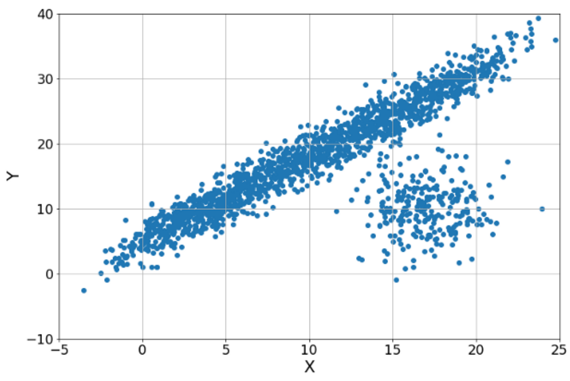


Figure 2.24: Data set containing both inliers and outliers [29].

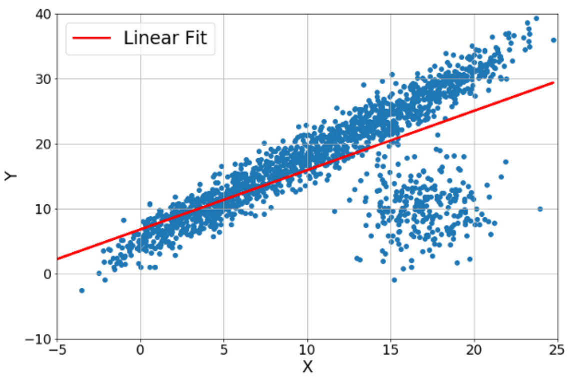


Figure 2.25: Least-squares linear fitting on the dataset [29].

RANSAC is an iterative method for fitting a mathematical model to a set of data and identify potential outliers. This method is classified as non-deterministic since the validity of the result is based upon a certain probability, thus increasing this probability by increasing the number of iterations of the algorithm. For every iteration, data points are randomly chosen and fitted into the mathematical model, a threshold is then applied, and every point in the dataset that is within this threshold from the experimental line is marked as a *inlier*. The distance from the point to the experimental line is calculated according to equation 2.12. The relationship between inliers and outliers determine if the iteration gave a valid result, RANSAC can work with up to 50 percent inliers.

$$distance = \frac{|ax_0 + by_0 + c|}{\sqrt{a^2 + b^2}} \tag{2.12}$$

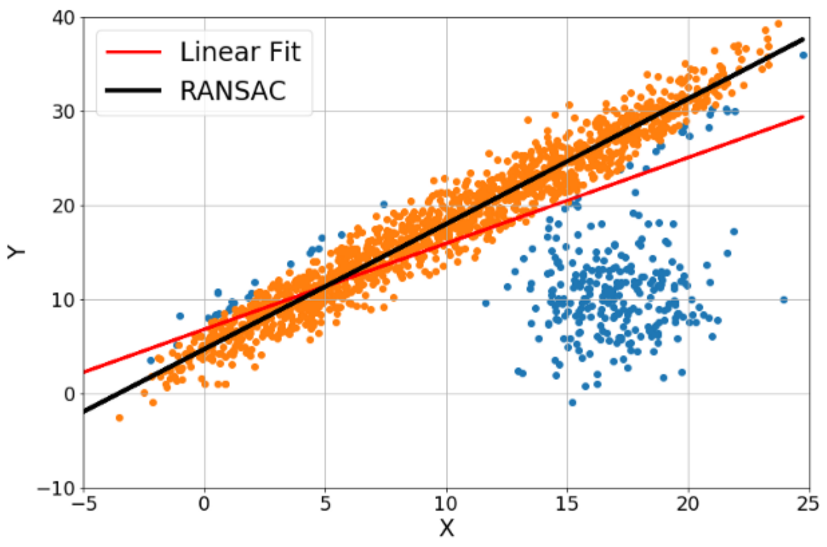


Figure 2.26: RANSAC compared to least-squares [29]. Note that the least-squares linear fitted line is biased by the blob of outliers.

RANSAC basics

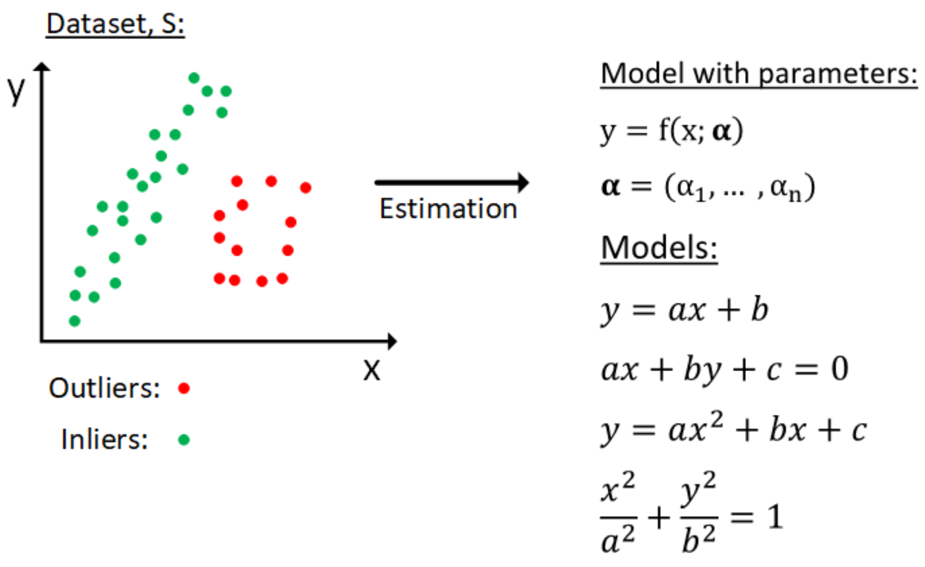


Figure 2.27: Illustration of the RANSAC principle. Where the data is presented in the $\mathbb{R}^2$-domain.

Figure 2.27 illustrates the process of fitting a mathematical model to a set of data embedded with outlier noise. Note that the data described here is designated for a linear $y = ax + b$ model. The iteration process and selection of inliers is the same for every mathematical model; they are only separated by the complexity of establishing model parameters from the subset of random points.

Algorithm 1 describes how a RANSAC algorithm is implemented with pseudo-code.

Algorithm 1: RANSAC psuedocode

```
numIteration = 0;
bestNoInliers = 0;
bestModelParams = None;
while numIteration < maxIteration do
    Select random points in dataset;
    Calculate model parameters from random points;
    Find the inliers based upon the model parameters and the inlier threshold;
    Count the detected inliers;
    if number of detected inliers > bestNoInliers then
        Update bestModelParams;
        Update bestNoInliers;
    else
        Continue sampling random points from dataset;
    end
end
```

2.5.4 Edge detection

An edge is defined as a sudden change in image brightness, i.e. going from black to white in a binary image or the opposite. These changes in brightness are classified as discontinuities. The ability to detect edges in an image is an important feature in many computer vision algorithms, and in this project used to count the number of edges in a horizontal vector of the detected tapping beam. Usually, edge detection is computed across an entire image, finding edges in both the vertical and horizontal axis of the image. However, it is also possible and highly common to only detect edges in one direction. Thus only finding vertical or horizontal edges.

By extracting a horizontal 1D - vector from a binary image, it is possible to see if there are such discontinuities in the data when moving from left to right. Figure 2.28 shows an example of this data vector in both image format and signal format. In this example, the edges are clearly defined, and the discontinuities are strong. Convolution is applied to generate a signal that describes the position of the edges.

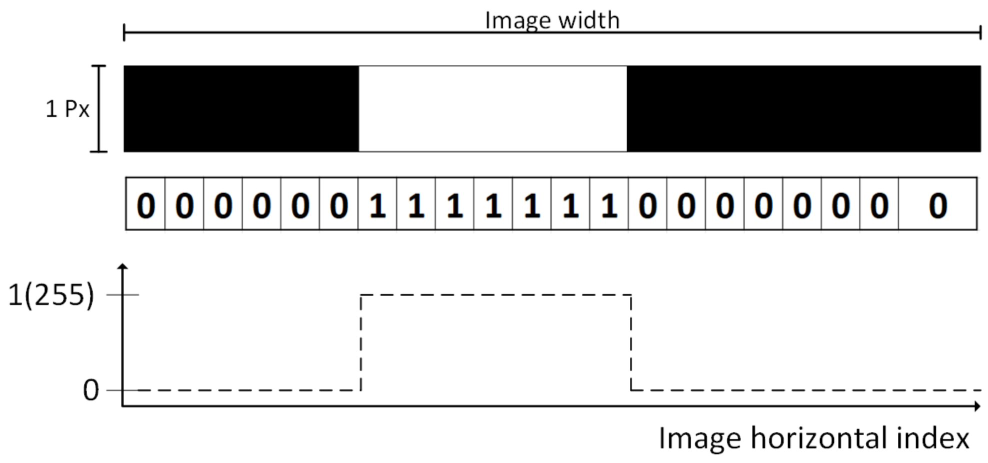


Figure 2.28: Sampled horizontal 1D-vector of an image

Convolution is the principle of combining two signals to form a third. This third signal is a representation of how the two influences each other. In edge detection, convolution is used on data similar to figure 2.28 in combination with a specified filter. The design of this filter controls the output from the convolution and must be chosen in regards to what type of data that is wanted [30]. An example of a filter for one-dimensional data is the Gaussian filter, which has a Gaussian normal distribution shape and can be used to detect discontinuities in data. The convolution is applied by sliding the filter across the data vector, computing the convoluted value for every step. Figure 2.29 is an example of using a Gaussian-filter, the edge can be located by looking for the peak in $\frac{\delta}{\delta x}(h * f)$.

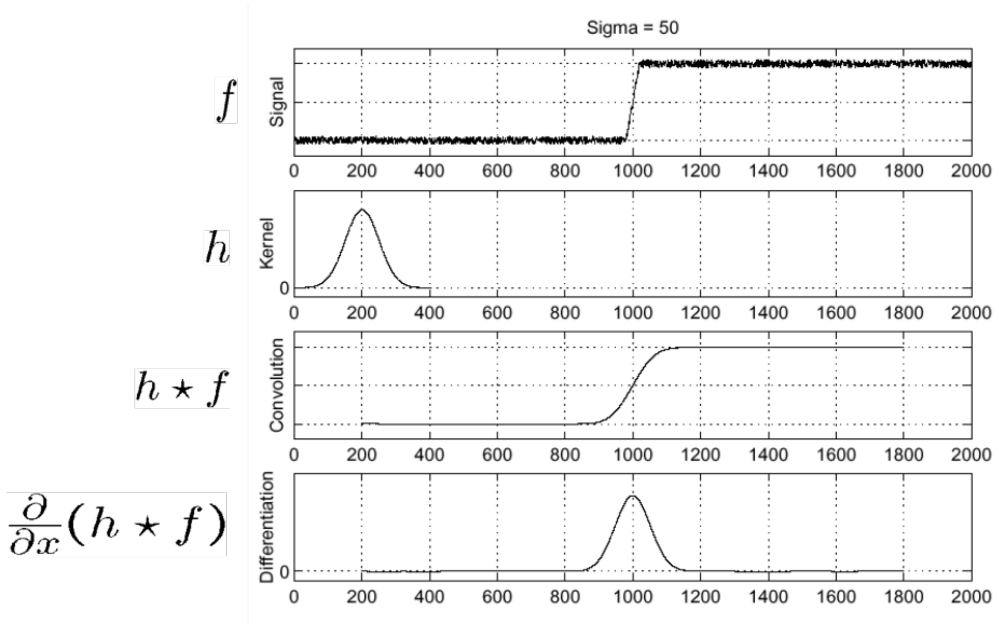


Figure 2.29: One dimensional convolution of a Gaussian filter and it's partial derivative.

A simplification of the convolution process in figure 2.29 is to take the partial derivative of the filter before the convolution takes place, see equation 2.13. This the equivalent to taking the partial derivative after convolution with a Gaussian filter.

$$\frac{\delta}{\delta x}(h * f) = (\frac{\delta}{\delta x}h) * f \tag{2.13}$$

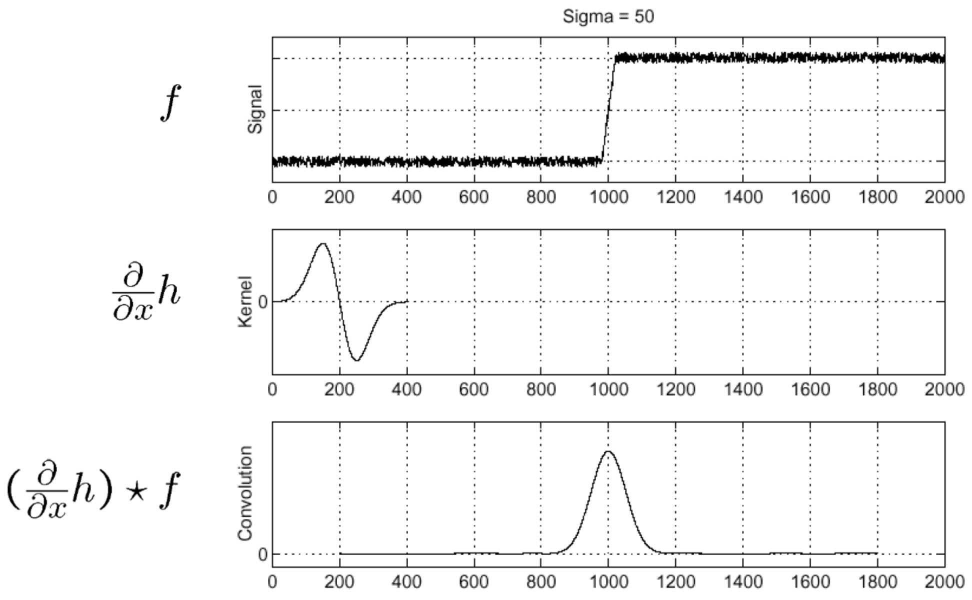


Figure 2.30: Convolution with a Gaussian partial derivative filter [31].

2.6 Project management and development

Planning for, and keeping track of a project of this size and complexity, implies that project management tools must be incorporated and used. Such tools are used to keep track of milestones, expose internal dependencies and manage weekly tasks. This section describes the different methods and approaches used in managing and developing the timeline and tasks for this project. Common for both timeline and task planning is the focus on using a strategy of Plan, Execute and Reflect (PER) and agile methods.

2.6.1 Gantt chart

Development of the project timeline is done using a Gantt chart. Gantt displays the different tasks and milestones in the project visually as horizontal bars, with start and finish dates indicating the length of these bars and with clearly marked dependencies for each task. One of the main benefits of a Gantt chart is the holistic overview it applies, making it easy to monitor the status of a project and quickly develop a overview of the tasks at hand. For this project, TeamGantt is used to keep track of milestones and task timelines. TeamGantt is an online project management tool that is based upon the Gantt structure, offering a clean and straightforward user interface and all the tools needed to structure milestones, tasks and sub-tasks [32].

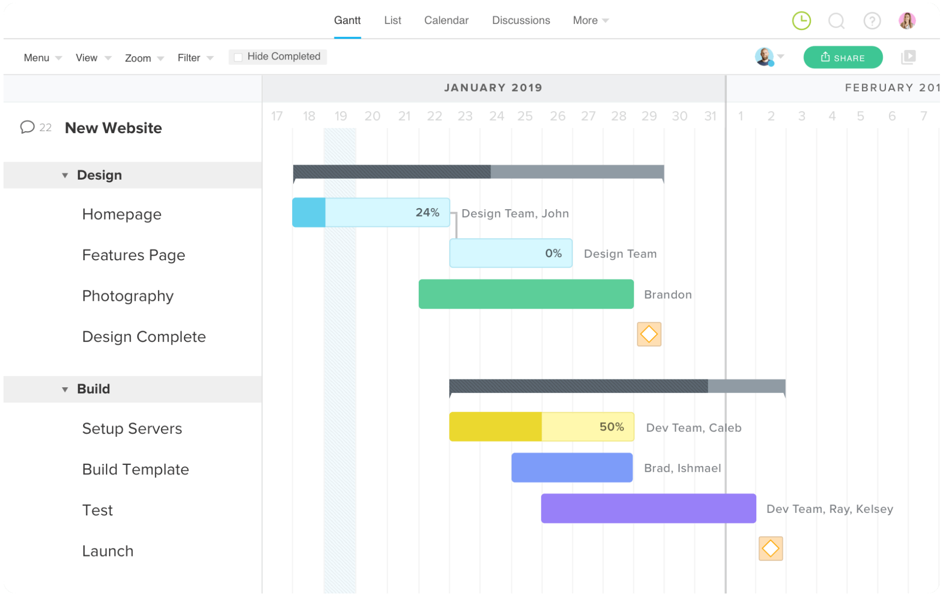
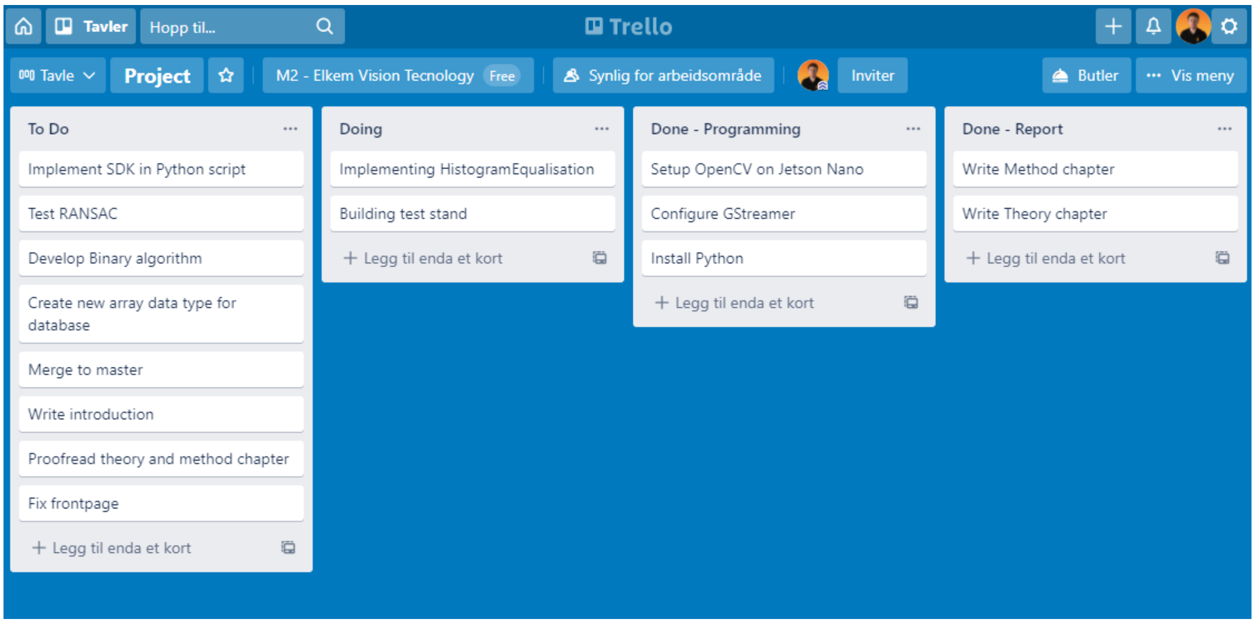


Figure 2.31: TeamGantt example [32].

2.6.2 Task management

As TeamGantt applies a holistic overview of the project and puts it into perspective with a timeline, it does not keep a detailed description of the tasks that must be executed to reach the specific milestones. Therefore, a task scheduler is necessary. Trello is another web-based project management tool, though different from TeamGantt in that it does not incorporate a timeline; instead using a Kanban-style board for keeping track of tasks [33]. For this project, Trello is used as the framework for detailed task scheduling and revised daily with both new and updates on ongoing tasks [34]. An agile approach is held towards the structuring and development of different Trello-boards, where it is adapted to fit new developments on the scope of the project [35]. PER strategy also helps new tasks and future planning learn from old tasks, increasing the effectiveness of the project management.



To Do
- Implement SDK in Python script
- Test RANSAC
- Develop Binary algorithm
- Create new array data type for database
- Merge to master
- Write introduction
- Proofread theory and method chapter
- Fix frontpage
- + Legg til enda et kort

Doing
- Implementing HistogramEqualisation
- Building test stand
- + Legg til enda et kort

Done - Programming
- Setup OpenCV on Jetson Nano
- Configure GStreamer
- Install Python
- + Legg til enda et kort

Done - Report
- Write Method chapter
- Write Theory chapter
- + Legg til enda et kort

Figure 2.32: Trello example

Chapter 3

Method

This chapter presents the method's used throughout this project to develop the final system design, hardware integration and software. An effort has been made to explain the application structure, computer vision processes and data communication as clearly as possible with either code examples or describing figures.

3.1 Implementation and use of thermal infrared camera

3.1.1 Optris PI400i thermal camera

For the development and testing in this project, an Optris Pi 400i thermal camera was borrowed from Elkem. The Optris PI 400i is a compact high-performance thermographic camera developed by Optris GmbH, with both Universal Serial Bus (USB) 2.0 and Industrial Process Interface (PIF) connectivity. Making it a popular choice for integration into Programmable Logic Circuit (PLC) systems and other types of industrial monitoring. Also worth noting that the temperature range specified in table 3.1 for this camera makes it ideal for local testing since the temperature of the test specimen can be below any dangerous levels for humans to interact with.

For the duration of this project, this camera will be implemented into the software loop, supplying the computer vision algorithms with raw data from an experimental test setup. Also worth noting that this camera has the same interface, both software and hardware, as the IR camera installed at Elkem's plant in Thamshavn, Orkanger, overlooking the tapping process at oven nr. 2. See section 1.2 and figure 1.3 for more details on the camera installation.

Figure 3.1: Optris PI 400i thermal camera with standard O29 lens [36].

Optris PI 400i specifications	
Temperature range	-20°C to 900°C
Spectral range	8 to 14 [μm]
Resolution	382x288 @ 27/80 [Hz]
Connectivity	USB 2.0 & PIF
Software	PIX Connect & IRImager SDK
Detector	Uncooled focal panel array (UFPA)
Optics	29° x 22°, F = 0.9
Focal length	13 [mm]
Thermal sensitivity	40 [mK]

Table 3.1: Optris PI 400i general specifications [36].

3.1.2 Hardware integration

Connectivity between the IR camera and the processing computer is simply done with the USB connection supplied with the camera. The provided PIF cable is not used in this project, as the system architecture does not include any form of PLC systems. Figure 3.2 shows the input terminals on the camera body and figure 3.3 shows how the camera is connected to a computer, running either Ubuntu or Windows 10.

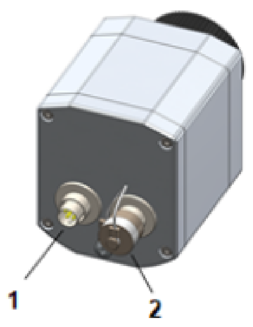


Figure 3.2: Optris PI 400i thermal camera, USB and PIF input terminals [36].

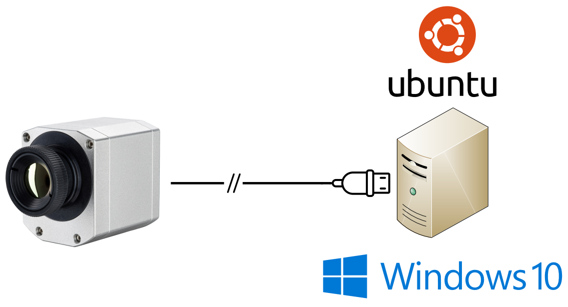


Figure 3.3: Optris PI 400i thermal camera connected to a computer running either Linux Ubuntu or Windows 10.

The applications developed in this project is implemented on a NVIDIA Jetson Nano single-board computer running Ubuntu 18.04. It is chosen due to its affordability, good GPU and CPU performance and the potential to train neural networks. The specification on the Jetson Nano is listed in table 3.2.

Jetson Nano specs:	
GPU	128-core Maxwell
CPU	Quad-core ARM A57 @ 1.43 GHz
Memory	4 GB 64-bit LPDDR4 25.6 GB/s
Storage	64 GB microSD
Connectivity	Gigabit Ethernet, M.2 Key E
USB	4x USB 3.0, USB 2.0 Micro-B

Table 3.2: NVIDIA Jetson Nano specifications [37].

3.1.3 Software integration

Supplied with the IR camera are two different software packages, dependent on the type of implementation. There is the *Optris PIX Connect* software, a real-time IR analysis program with several built-in functions to monitor and log temperature data. The software is fully capable of monitoring an industrial process, but prototyping detection algorithms and implementation on single-board computers such as the NVIDIA Jetson Nano is not possible. *PIX Connect* is only compatible with Windows 7, 8 and 10. As cross-platform compatibility between Windows and Linux-based operating systems is desired, it is not used as the interface for the camera data.

Integrating the IR camera data-feed in real-time with algorithms from open-source libraries such as OpenCV and NumPy, requires the use of the Software Development Kit (SDK) developed by Evocortex for the Optris PI-series cameras [38]. The SDK supplied is called *IRImagerDirectSDK* and can output thermal image and temperature data for both C/C++ and Python programming languages. Implementation of the SDK is possible with two different Application Programming Interface (API)'s, *Expert API* and *Easy API*, dependent on the desired level of control over the camera functions, see table 3.3 for comparison. Since the *Easy API* has support for the Python language, it is used in developing the computer vision application.

Comparing Easy API and Expert API	Easy API	Expert API
Programming style	Simple function calls	Object oriented
Programming language	C/C++, Matlab, Python & Labview	C++
Thermal image	Yes	Yes
Palette image	Yes	Yes
Change palette color	Yes	Yes
Change Palette Scale	Yes	Yes
Change Palette Temperature Range	Yes	Yes
Set Focus Motor Position	Yes	Yes
High Precision Mode (PI450)	Yes	Yes
Extended Temperatur Range	Yes	Yes
RAW Data Recording for PIX-Connect	No	Yes
Access to visible frame (PI230)	No	Yes
Multiple Cameras	Yes	Yes
PIF I/O Support	No	Yes
Detect Connection Loss	No	Yes
Flagcontrol during Runtime	Yes	Yes
Create Optris Tiff Files	No	Yes
Temperature referencing with external probe (BR 20AR)	No	Yes

Table 3.3: Optris SDK API comparison [38].

IR Imager Direct-SDK on Windows OS

For the SDK to be implemented into the program, it is saved as a Dynamic-link Library (DLL) file inside of the application folder structure. Figure 3.4 shows the general structure of the application. The SDK DLL file is then called into the application file `elkemvision/src/IRCamera.py` by the function `ctypes.CDLL("PATH_TO_DLL")` from the `ctypes` foreign function library in Python [39]. This method is specific to Windows operating system, section 3.1.3 describes the process for systems running Ubuntu.

IR Imager Direct-SDK on Ubuntu OS

The SDK is also compatible with the Linux-based operating software Ubuntu and supports the *amd64*, *I386*, *arm64*, *armhf*, *armel* and *atom* architectures. For this project, the IR camera is integrated with a NVIDIA Jetson Nano single-board computer with *arm64* and Ubuntu 18.04 operating system. Installation of the SDK is done using the Debian package (dpkg) manager and is dependent on *cmake*. A basic installation guide is available at the Evocortex *IRImagerDirectSDK* documentation website [40]. Implementing the SDK in Python for a Linux-based operating system is mostly similar to the method described in section 3.1.3, but does not require the path to the DLL-files. The path is found using `ctypes` and the function `ctypes.util.find_library("name")`, where `"name"` must be the library name without any prefix, suffix or version nr.. Then the library is loaded with `ct.cdll.LoadLibrary("path")`, where `"path"` is the path to the library.

```
/
└── elkemvision
    ├── bin
    │   ├── Win32
    │   │   └── libirimager.dll
    │   └── x64
    │       └── libirimager.dll
    ├── config
    │   ├── <"serial-number">.xml
    │   └── Formats.def
    ├── src
    │   └── IRCamera.py
    └── mainRealTime.py
```

Figure 3.4: General application structure for cross-platform compatibility.

Receiving data from the IR camera

Initialising the connected IR camera and reading thermal data is done in the file `IRCamera.py`, which contains the class `IRCamera()`. This class contains all elements that are connected to the IR Imager Direct-SDK. During initialisation, the class will detect if the host's operating system is either Windows or Linux and then load the appropriate .DLL-file. Shared variables across the class are initiated with `self`-parameters. All functions within this class are described in the paragraphs below. The complete code for this class is available in the appendix C.1.2.

`IRCamera()` Class constructor; takes in the global path where the program is executed. And prints the camera serial number, thermal image size and temperature array size to the terminal.

`IRCamera.status()` Status function; when called it prints the status of the camera and library to the terminal.

`IRCamera.getThermal()` Get thermal data; when called it returns the thermal image and temperature array as `np.array` datatype.

`IRCamera.calibrate()` Calibrate the thermal image. Takes in a distorted thermal image, using the camera matrix and distortion coefficients it applies `cv.initUndistortRectifyMap()` and returns a undistorted image. Given that the correct camera matrix and distortion coefficients are used.

3.2 Computer vision algorithms

This section will cover the development, structure and implementation of the computer vision algorithms developed in this project. Code generated in this project is written using Microsoft Visual Studio Code, a well-supported and straightforward source code editor that makes developing code more efficient. The general structure of the application code is described in figure 3.5. Regarding re-usability, the code is developed to be as versatile as possible, meaning that all the necessary parameters are initialised during instantiating of the classes.

```
/
└── elkemvision
    ├── bin
    ├── config
    ├── src
    │   ├── BeamDetector.py
    │   ├── Database.py
    │   ├── FeatureTracking.py
    │   ├── IRCamera.py
    │   └── RansacFit.py
    ├── template
    │   └── template.png
    ├── webapp
    ├── mainRealTime.py
    ├── mainFlask.py
    └── runProgram.bash
```

Figure 3.5: Application layout.

3.2.1 Tappingbeam detection

Finding the position of the tapping beam in the IR camera image is done using the theory described in section 2.5.1. The source code for detecting the tapping beam is located in the file `elkemvision/src/FeatureTracking.py`, where the program is structured as a class, with initialised `self`-variables and functions. Each function is built as a wrapper for their respective OpenCV functions. This is done, so integration with other algorithms is as simple as possible. The usage of these functions is described in the paragraphs below and the code is available in appendix C.1.3.

`FeatureTracking("path_to_template")` Class constructor; takes in the path to the template and initialises it as a `self`-variable.

`TemplateMatching(thermal_image)` Template matching function; takes in a blue, green and red (BGR) image from `IRCamera.getThermal()` and calculates the correlation between the image and template. This function uses the OpenCV-function `cv.matchTemplate(image,template,method)` where; `image` is the input grayscale image, `template` is the image template and `method` is the chosen correlation method. The matching function returns an array containing the correlation parameters, this array contains values in the range of -1 to 1, section 2.5.1 describes these values. The best matched position of the template is found with a threshold value of 0.8, using `np.where(condition[, x, y])` to find indexes for values higher than the threshold. If the match is successful, the function returns a tuple containing the template-position, width and height of the template and a black image with the tappingbeam region. An example input image, template and output image are shown in figure 3.6, 3.7 and 3.8.

`BoundingBox(image, location, angle)` Draw bounding-box function; takes in an image, location tuple and angle tuple. The purpose of this function is to overlay a bounding box on a preferred image of the tapping beam, only meant as a tool for visual representation. The `location` data field contains the top left corner coordinate as well as the height and width of the template, and the angle data includes the absolute angle of the trapping beam measured from the image x-axis.

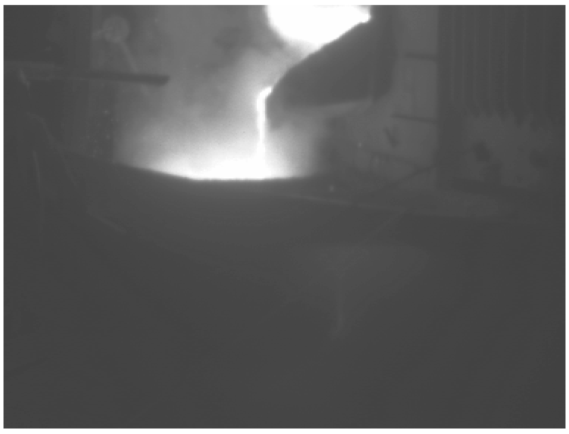


Figure 3.6: Graysacle input image for the template matching.

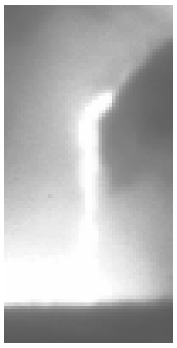


Figure 3.7: Template for the input image.

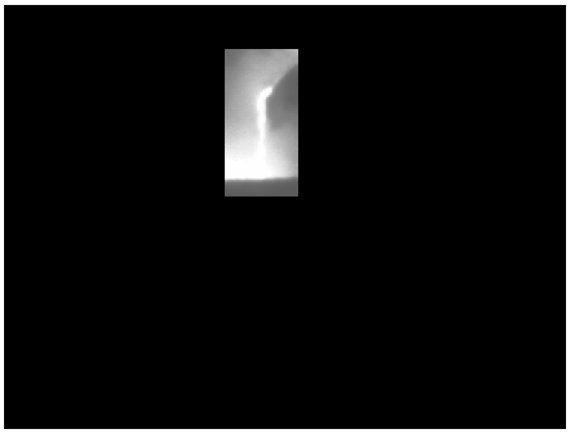


Figure 3.8: Resulting example image from `TemplateMatching()`.

3.2.2 Detail enhancement and binarization

Template matching and segmentation, as described in section 3.2.1 eliminates most of the disturbances in the image. However, some are still present, and as visible in figure 3.8 where most of the picture is now black, i.e. a grayscale value equal to zero. The tapping beam is visible in the middle of the image, surrounded by smoke and other bright disturbing objects. To increase the contrast in the image and better define a separation between the desired features in the foreground and the background, a histogram equalisation as described in section 2.5.2 is performed. The code for these functions are available in appendix C.1.4.

Equalising the histogram

`HistogramEqual(image)` The function for histogram equalisation, it is located in the class `BeamDetector()`, in the file `BeamDetector.py`. Input for this function is a template-matched and segmented image, as represented by figure 3.8. First, the histogram for the image is created using NumPy function `np.histogram(image, bins, range)` where; `image` is the input, `bins` is the number of different values the histogram should represent and `range` defines the upper and lower range for the `bins`. Then, a cumulative histogram is created with the NumPy method `numpy.ndarray.cumsum()`. Following equation 2.11, the minimum value of the histogram can not be 0, this is solved by applying a masked array. The NumPy masked array module is used to filter out and delete certain values within an array. In this instance the function `np.ma.masked_equal(x, value)` is used, where `x` is the input array to be filtered and `value` is the value to filter out.

After filtering of the cumulative array, the minimum and maximum value is calculated and the histogram is iterated through equation 2.11. The returned equalised histogram array has some indices where the value is equal to `NaN`, function `np.ma.filled(x,value)` solves this by filling in the empty array positions with the value 0. Using the histogram as a look-up table, the input image is transformed accordingly with `image_equalised = cdf_m[image]`. An example output image is available in figure 3.9.

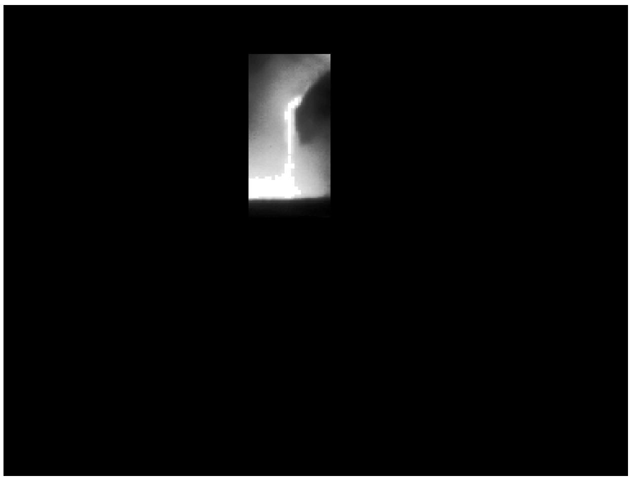


Figure 3.9: Histogram equalised image.

Creating a binary image

After equalising the image histogram, the desired features in the image is now more prominent and thus easier to separate from the background. This algorithm is based upon the idea that the tapping beam always exists in the brightest parts of the image and within some margin below the maximum intensity value. A gain is applied to adjust the margin for which pixel intensities should be considered a part of the tapping beam. This is only adjusted if the binary representation is not valid, where the binary image displays an improper tapping beam. The algorithm is located in the class `BeamDetector()`.

$$i_{thresh} = np.round\left(i_{max} - \frac{i_{mean}}{i_{max}} * thresh_{gain}\right) \tag{3.1}$$

`BinaryImage(image)` The function for creating a binary image. Input for this function is the histogram equalised image, represented in figure 3.9. The max and mean values, i_{max} and i_{mean} are calculated using the `np.amax` and `np.mean` functions. They are then applied into equation 3.1 and the threshold for binarization of the image is calculated. Indices for the foreground objects in the binary image are found using `np.where(condition[, x, y])`, this function returns the indices of the array where the condition is satisfied. With `indices = np.where(image >= i_thresh)`, the tapping beam is separated out from the background objects. `indices` is a tuple containing the indexes in x- and y-direction, a black image (`image_binary`) is then indexed and the respective positions have their value set to 255. Figure 3.10 displays the binary image that is returned from function `BinaryImage(image)`.

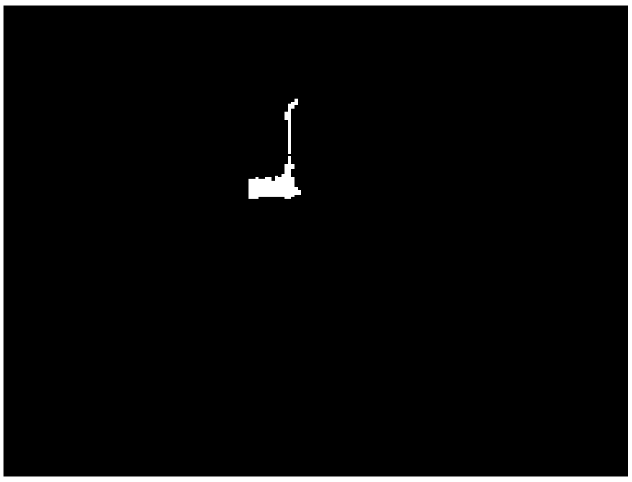


Figure 3.10: Binary image.

3.2.3 Data-filtering and model-fitting

The algorithms in section 3.2.1 and 3.2.2 removes much of the disturbances and unwanted features in the image. However, the image displayed in figure 3.10 still contains unwanted features that are not a part of the actual liquid metal tapping beam. And by looking at the raw thermal image in figure 3.6, it is fairly easy for a human to visually separate the tapping beam from the high-temperature smoke that the previous algorithms also detected. Given that the beam of liquid metal is mostly following a 2D linear model, pouring downwards into the ladle after leaving the tapping spout. The task of isolating the tapping beam is then adapted into finding the parameters for a 2D linear model ($y = ax + b$), that gives the desired number of inliers. To solve this, a RANSAC algorithm as described in section 2.5.3, is implemented into the Python application. If successful, this algorithm will find the best parameters of the model it is assigned, based upon the dataset it is given. The algorithm is structured in the class `RansacFit(threshold)` and located in file `elkemvision/src/RansacFit.py`. During the initialisation of the class, the inlier threshold is also defined. The source code for this class is available in appendix C.1.5.

`FindInliers()` The main function of the class. Designed as a wrapper for the `fit_ransac` function. Takes in a binary image of the tapping beam with outliers and returns the best inliers, model parameters of the fitted line and an image with the inliers showing. First, the input image is analysed, and the indexes that have a value of 255 are chosen as data points with `np.where(image == 255)`. These data points are then given to the function `fit_ransac`, that finds the best-fitted inliers and model. The resulting model parameters and inliers are then used to calculate the absolute angle of the tapping beam using the image coordinate system. Using model parameters in variable `bestModel` and the lowest and highest index value for the inliers (`np.min(bestInliers[:,0])` and `np.max(bestInliers[:,0])`) , the data points for the two-dimensional line is calculated. The angle is calculated by taking the coordinates for the two end-points of the line data points, calculating the delta distance in x and y and using the function `np.arctan2(deltaY,deltaX)`.

`fit_ransac()` The function that contains the core RANSAC algorithm. Takes in a data set of inliers and outliers, the maximum number of iterations, number of samples for the model parameters, a threshold to classify data points as inliers and a minimum number of inliers to find. First, the RANSAC parameters are initialised and then iterated through the number of iterations.

`evaluate_model()` Function for evaluating the chosen model parameters. Input for this function is the dataset of inliers and outliers, model parameters from randomly selected points and the inlier distance threshold. The distance from the experimental line to every point in the data set is calculated according to equation 2.12. The inlier indexes are found by filtering out the distances that are larger than the desired threshold.

`fit_lsq()` The function that fits a two-dimensional line model to the randomly selected data points. Takes in coordinates of the randomly sampled data points and returns the model parameters. Calculating the model parameters for a two-dimensional line using least-squares method, `np.linalg.lstsq()`.

3.2.4 Temperature calculation

Along with a thermal image, the IR camera function also outputs a temperature matrix. Combining the temperature matrix with the filtered image of the tapping beam from the RANSAC algorithm, it is possible to find the maximum, average and minimum temperature of the liquid metal tapping beam and display this as a parameter in the HMI. Function `TempCalculation()` in the `BeamDetector()` class, incorporates NumPy function `np.where()` to find the pixel coordinates of the tappingbeam and transfers these points over to the temperature array. Then, `np.max()`, `np.mean()` and `np.min()` calculates the max, mean and minimum temperature in the beam. The code for this function is listed in appendix C.1.4.

3.2.5 Tapping beam algorithms

To give the process operators the desired feedback on the status of the tapping beam, an algorithm that can retrieve the desired data must be developed. Typically the beam is pouring directly downwards as a consistent stream of liquid metal, but with the introduction of slag and potentially degrading flow properties of the tapping spout, the tapping beam will at times divide itself into two to three smaller, more sparse and disrupted beams. The effect from this is an increased surface area of the liquid metal exposed to cool air and a lower temperature of the tapped metal in the ladle. If a split tapping beam occurs, it is possible to intervene and normalise the flow by treating the tap-hole or cleaning out the tapping spout.

The continuity and uniformity of a tapping beam can be represented by the average number of edges in a series of horizontal vectors over the height of the tapping beam. If the average of detected edges in the tapping beam is 2 per horizontal vector analysed, then the beam is classified as highly uniform and continuous. Figure 3.11 displays how to the tapping beam is divided into horizontal vectors of
$1\ by\ (width-1)$-dimension. The value 0 indicates the black colour, and value 1 indicates white, note that this value is 255 in a real image. An edge is denoted by the image values going from low to high or from high to low. Detecting the edges in each horizontal vector is done according to the theory described in 2.5.4.

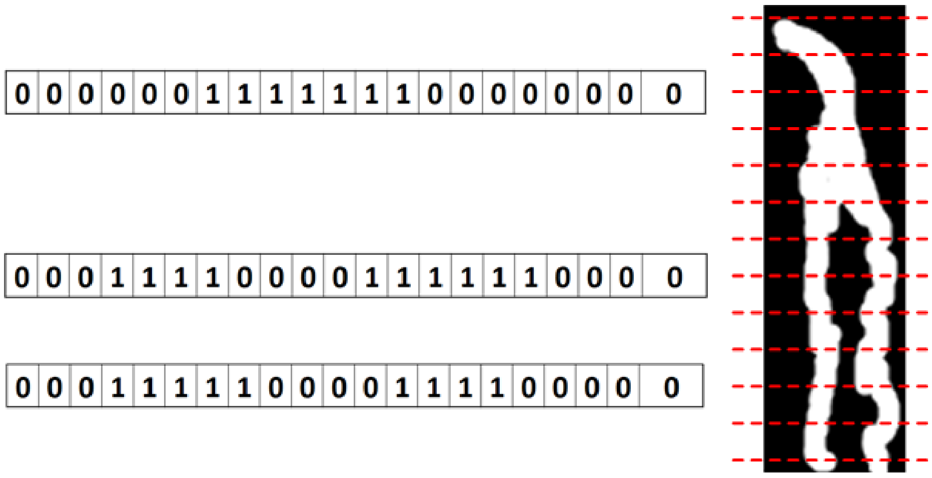


Figure 3.11: Horizontal sampling of the tappingbeam to detect shape.

The algorithm for counting the edges in the tappingbeam is located under `elkemvision/src/BeamDetector.py`, structured in function `BeamShape()` in the class `BeamDetector()` and available in appendix C.1.4

`BeamShape()` Function for classifying the shape of the beam. Takes in a binary image of the tapping beam and a tuple containing coordinates for where the template matched, and the width and height of the template. With this information, the function can crop the binary image down to the size of the template and only focus on the tapping beam ROI. It is also possible to adjust the vertical start and stop position of the horizontal sampling by adjusting the values of `height_top` and `height_bottom`, note that `height_bottom` must be a negative value. After adjusting the vertical span for the sampling, a vector containing the specific increments are created with `np.arange(heigth_top, (heigth + heigth_bottom), 10)`, this vector is used to index the sampling of horizontal vectors for every 10 pixels. Iterating through the index vector, a row of pixel values is sampled from the image and convoluted with a derivative-Gaussian filter. The immediate results from the convolution is a data vector as shown by the *Convoluted vector* in figure 3.12. For a rising edge, the convolution generates a normal-distribution-like curve with a positive maximum point, and for a falling edge, it generates a curve with a negative maximum point. To solve this, and only have to look for peak-values in the detected edge-curves, the absolute value of the signal is computed.

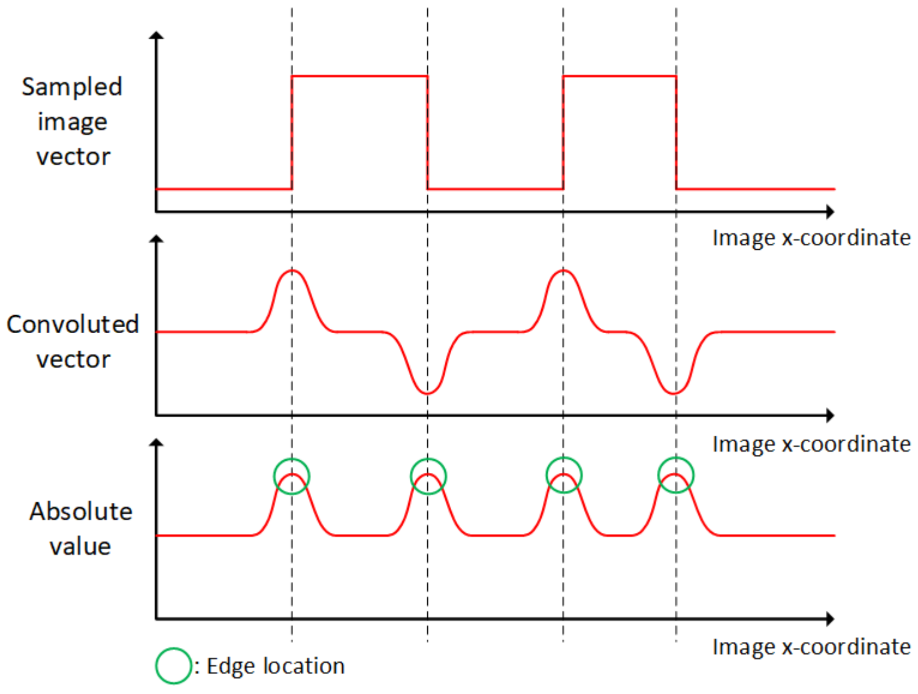


Figure 3.12: Convolution with a derivative-Gaussian filter and absolute value on the result.

Peak values in the data vector, *Absolute value*, are found by differentiating the dataset with respect to x, computing the sign of the differentiated data and then differentiating the sign data. It can be viewed in figure 3.13, that the edge is located at a transient point equal to -2. In a real situation, these transient values will possibly go towards negative infinity. The edges are therefore found by looking for where the graph; *Derivative of sign-data*, is of value smaller than 0. Note that this process of differentiating the sign-data will shift all the indexes one pixel to the left; therefore, the peak positions are added with a value of 1.

```python
# Isolate row of image data
row = image_isolated[i,:]

# Convolve the extracted image data with a Gaussian derivative filter
row_convolve = np.convolve([1,0,-1], row)

# Find peaks in dataset
row_convolve_peak = (np.diff(np.sign(np.diff(np.abs(row_convolve)))) <
    0).nonzero()[0] + 1

# Count number of edges in row
edges = len(row_convolve_peak)
```

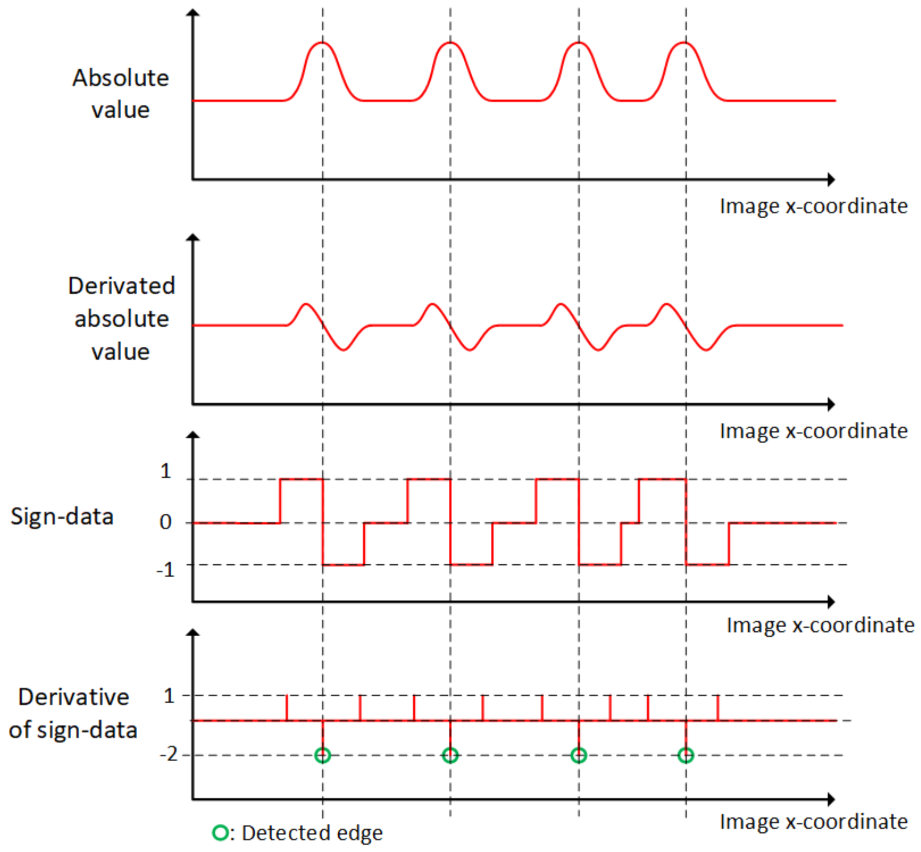


Figure 3.13: Edge detection algorithm example.

After the iteration process mentioned above, the average number of detected edges in each row is computed and used as a parameter for the shape of the beam. If the average lies in between 1 and 3, the shape is classified as optimal; if the average value is larger than 3 it is classified as sparse and discontinuous and finally if it is less than 1 a warning label is presented. An average value of detected edges less than 1 indicates that something is wrong with the input image or the algorithm itself.

3.2.6 Application structure and data flow

Figure 3.5 provides an overview on the application structure, in regards to where the files are located and the hierarchy that exists and figure 3.15 explains how the application is incorporated with the tapping operators. Every function is dependent on certain data and variables generated by the previously executed function. The Python script `mainRealTime.py` serves as the main file for launching the IR camera application, importing the necessary packages, creating instances of the classes and executing the functions inside of a while-loop. And the Python script `mainFlask.py`, launches the web-based HMI. A flowchart that presents the dependency and execution order for each function in `mainRealTime.py` is available in appendix B.4.

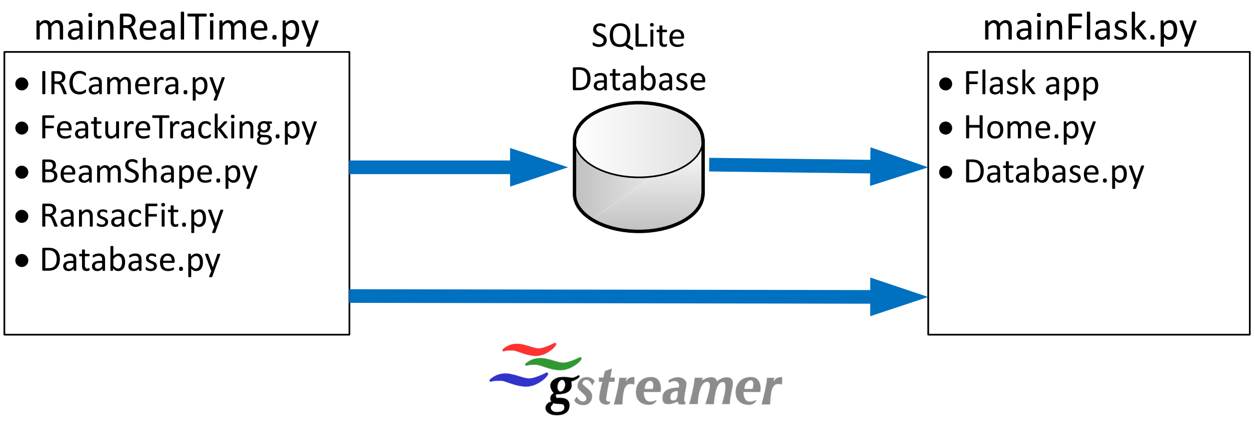


Figure 3.14: Application structure and data flow.

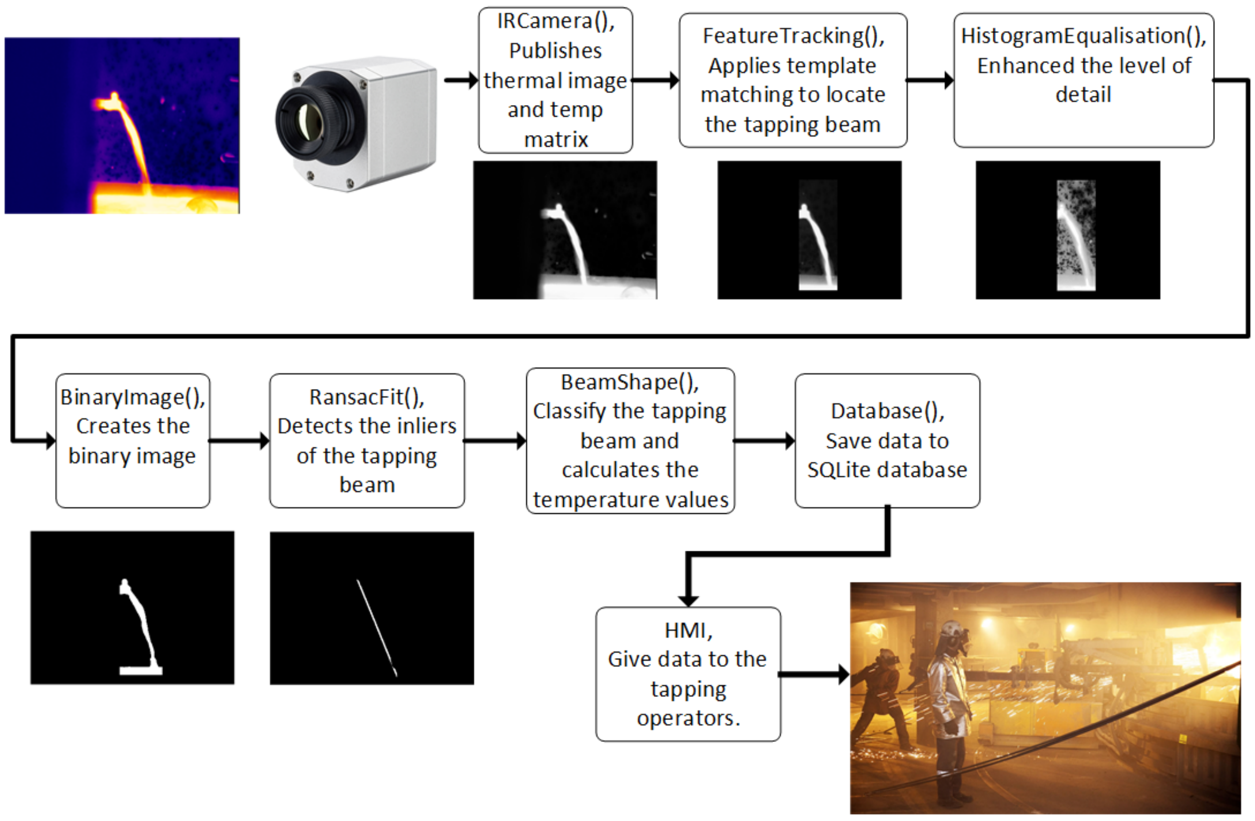

IRCamera(), Publishes thermal image and temp matrix
FeatureTracking(), Applies template matching to locate the tapping beam
HistogramEqualisation(), Enhanced the level of detail
BinaryImage(), Creates the binary image
RansacFit(), Detects the inliers of the tapping beam
BeamShape(), Classify the tapping beam and calculates the temperature values
Database(), Save data to SQLite database
HMI, Give data to the tapping operators.

Figure 3.15: Application sequence and data flow.

3.3 Database and Human Machine Interface

As mentioned in this projects scope and objectives in section 1.3, an interface between the human operators and the IR camera application is to be developed. This application aims to provide the operators with detailed real-time information about the ongoing tapping process and data-logging for metallurgists. The source of information is provided by the applications developed in section 3.2, where specific data are stored in a database at a given frequency and a video stream is transported with a pipeline-based multimedia framework.

3.3.1 SQLite database

SQLite is chosen as the framework for data storage since it easily incorporates a Python application and allows for the customisation of tables and data types. File `Database.py` contains the functionality of the database and is implemented into both the `mainRealTime.py` and `mainFlask.py`, launch file for the IR camera application and web-based HMI. The functionality is structured in the class `Database()` with functions for connecting the database file, writing and reading of data. Functionality is described in the paragraphs below and the code is available in appendix C.1.6.

`Database("path_to_database")` The initialisation of the class. Takes in the global path for the `.db`-file, and stores it as a `.self`-variable. After execution, a connection to the database is established, and if the file does not exist, a new one is created with the specified tables. After executing, the function prints to the terminal; `"===== Created connection to database! ====="` if successful, and the opposite if the connection fails.

`create_connection()` Establishes a connection to the the database. Takes input from the class `.self`-variables. `conn = sqlite3.connect()` is then used to establish connection to the database, given input parameters; `self.dbPath, check_same_thread=False, detect_types=sqlite3.PARSE_DECLTYPES` . Where `self.dbPath` is the path to the database file, `check_same_thread=False` enables multi-thread operation and `detect_types=sqlite3.PARSE_DECLTYPES` makes the SQlite module parse the declared type for each column of data it returns [41].

`create_table()` Creates the desired tables. Takes input from `.self`-variables and the variable for the declared table structure. First the function establishes a cursors to the active database connection with `c = self.conn.cursor()`, and then executes the `create_table` command. This function is only used during the initialisation of the class. The structuring and creation of data tables are shown below.

```
# Create temperature data table
sql_temperature_table = """ CREATE TABLE IF NOT EXISTS temperature (
        id              integer PRIMARY KEY,
        tempMax         float                   NOT NULL,
        tempMean        float                   NOT NULL,
        tempMin         float                   NOT NULL,
        timestamp       text                    NOT NULL); """

# create a database connection
self.conn = self.create_connection()

# create tables
if self.conn is not None:
    # create temperature table
    self.create_table(sql_temperature_table)
```

`update_temperature()` **and** `update_imagedata()` Functions for updating the database with new data. Takes in input from `.self`-variables and the data it is to update the database with. A cursor to the database is established and the specific data is written to the database with `cur.execute(sql, data)` and `self.conn.commit()`. Where `data` denotes the variables that is to be stored in the database and `sql` specifies the data table. This function is integrated into `mainRealTime.py` and is executed for every 5th iteration in the `while`-loop.

```
# Define data to be updated
sql = ''' INSERT INTO temperature(tempMax, tempMean, tempMin, timestamp)
        VALUES(?,?,?,?) '''
# Create connection cursor
cur = self.conn.cursor()

# Updata data
cur.execute(sql, data)

# Commit the data into the database
self.conn.commit()
```

`select_data()` The function that reads from the database and returns the values as a list of tuples. Takes input from the class `.self`-variables and returns a specified number of rows from the tables, only publishing the newest data. A cursor to the active database is established and the specified number of rows are extracted from the table, the command `cur.fetchall()` returns the row as a list of tuples [41]. This function is integrated into `home.py`, the Python-view for the home page of the web-based HMI.

3.3.2 Flask web interface

A HMI between the tapping operators and the IR camera application is developed using the Flask framework, providing the operators with a live feed of the detected tapping beam, status indicators and a temperature trend graph. Where the SQLite database and GStreamer pipeline are used to connect the HMI application to the IR camera application, sharing temperature data, image data and video feed asynchronously between the two. Figure 3.16 displays the application structure, divided into sub-folders containing either CSS, JavaScript, Python or HTML files.

Position, size and configuration of the different elements in the HMI are defined in the HTML templates, with the help of Bootstrap CSS and JavaScript templates. Bootstrap makes it possible to structure the website into columns and rows that dynamically adapts and changes its size, making it compatible with all types of browsers and units (PC, tablet, smartphone).

```
/
└── elkemvision
    ├── bin
    ├── config
    ├── src
    ├── template
    ├── webapp
    │   ├── database
    │   │   └── data.db
    │   ├── static
    │   ├── templates
    │   │   ├── base.html
    │   │   └── home.html
    │   ├── views
    │   │   ├── __init__.py
    │   │   └── home.py
    │   └── __init__.py
    ├── mainRealTime.py
    ├── mainFlask.py
    └── runProgram.bash
```

Figure 3.16: Application layout.

`mainFlask.py` The main python script for the HMI, imports the application instance from `webapp/__init__.py`. Thus, launching this python program initialises the HMI and starts the Flask web-server, on the condition that the file `mainRealTime.py` have been launched and is running. This is because the GStreamer pipeline must exist for the HMI to successfully launch. The code is available in appendix C.2.1.

`webapp/__init__.py` Initialises the Flask application instance. Imports `webapp/views/__init__.py`, which in return imports `views/home.py`.

`home.py` The main python function for the HMI application. This file serves as the connection layer between the database, GStreamer and the HTML templates, where data is extracted from the database and video feed is read from the GStreamer pipeline using a series of functions. Every function returns data to the HTML-templates via separate URL's that are routed into the application with `@application.route("URL")`. The routing function either returns a rendered HTML-template or a response to any variables in the template, for instance; displaying a video stream, updating temperature chart data and status labels. Which status label that is presented on the HMI is decided by if-else statements in `home.py`. For the beam angle; two angle intervals are used to categorise the tapping beam, and which interval the tapping beam angle lies within decides the label. The same principle applies for the beam angle; where the label is decided by an if-else statement using the beam existence and beam status parameter. The various labels used to signal the state of the tapping beam are listed below in figures 3.17 to 3.21. The source code for this Python script is available in appendix C.2.2 and the HTML-templates are listed in appendix C.2.3 and C.2.4.

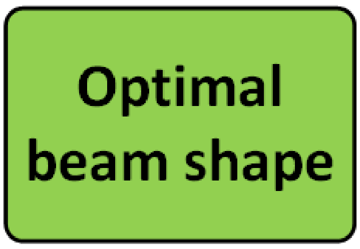

Optimal beam shape

Figure 3.17: HMI label for optimal tapping beam profile.

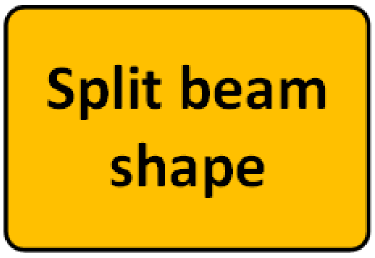

Split beam shape

Figure 3.18: HMI label for split tapping beam.

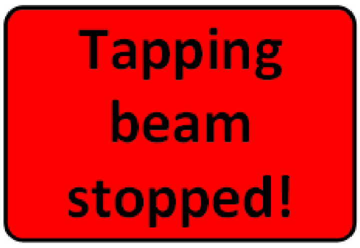

Tapping beam stopped!

Figure 3.19: HMI label for stopped tapping beam.

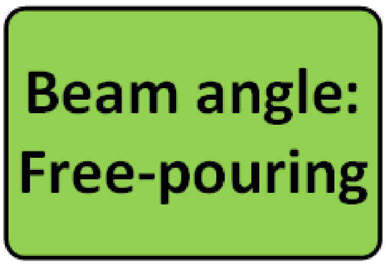

Beam angle: Free-pouring

Figure 3.20: HMI label for free pouring tapping beam.

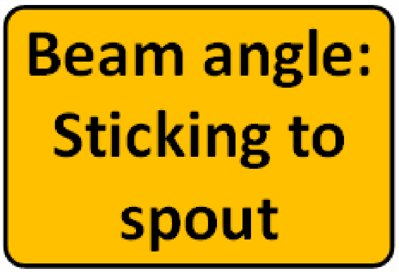

Beam angle: Sticking to spout

Figure 3.21: HMI label for inverted tapping beam.

3.4 Tapping beam test stand

Initial development of the algorithms presented in the previous sections was done using images from the IR camera installed at Elkem's plant in Thamshavn. These images provided a good base for development and local testing of the `HistogramEqual()`, `BinaryImage()` and `FindInliers()` functions. But for development and real-time testing of the IR application a simple test-stand that could to some extent replicate the actual tapping beam was necessary.

A simple test-stand was built using a $12[V]$ pump and warm water. The benefits of this setup were the simplicity and low-risk usage; there were no temperatures that could be of potential harm, and testing only required warm water from the sink. Figures 3.22, 3.23 and 3.24, show the configuration of the test stand as well as the resulting thermal image.

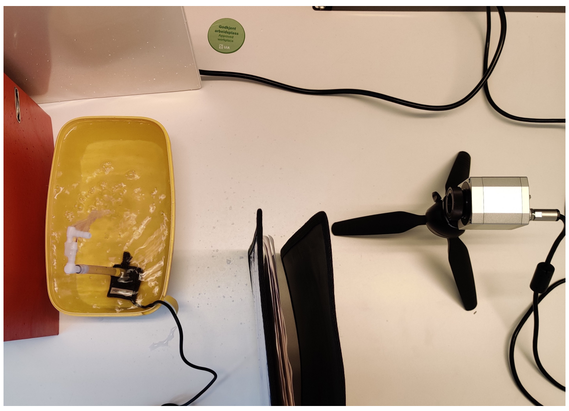


Figure 3.22: Top view of the test stand.

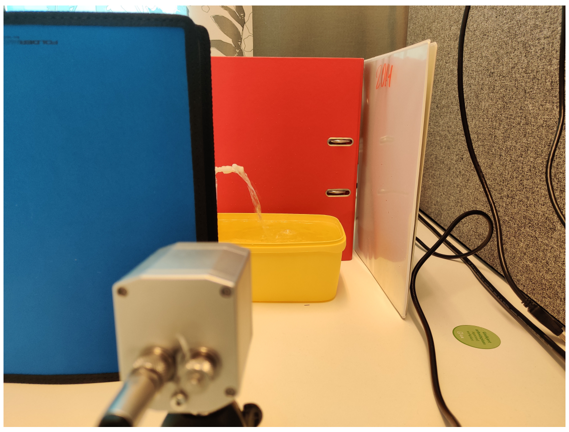


Figure 3.23: Camera view of the test stand.

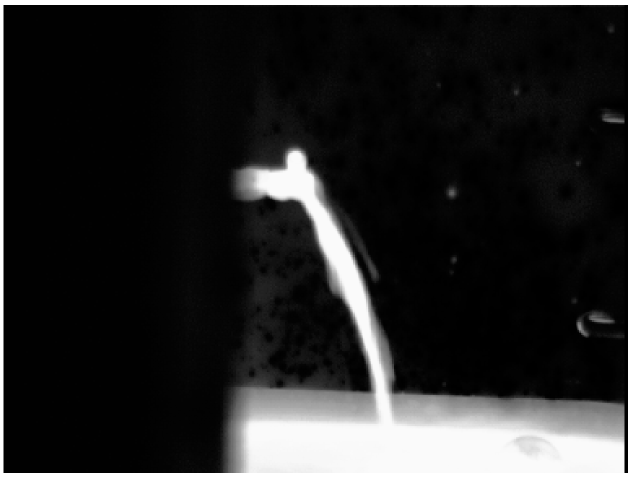


Figure 3.24: Grayscale thermal image from the test stand.

Chapter 4

Results

4.1 Computer vision algorithms

In this section, the results from the different computer vision functions and algorithms are presented. All of the data presented here have been generated from testing with the experimental tapping beam test-stand shown in section 3.4 and synthetically developed images for the tapping beam shape classification.

4.1.1 Overall results

The IR camera application yielded in a functional program that takes in the thermal image and temperature matrix from the Optris IR camera, processes it through the computer vision algorithms and returns a set of data about the tapping beam that is sufficient to solve the scope and objectives set for this project, referring to section 1.3.

Final testing of the algorithms has been done on the NVIDIA Jetson Nano single-board computer, running the Linux based operating system, Ubuntu 18.04, connected to the Optris IR camera via USB. While running the application, it was clear that the processing power of the Jetson Nano would be a limiting factor. As a result, the time between launching the file `mainRealTime.py` and the algorithms had compiled were recorded to $56[s]$. But it must be taken into consideration that the JIT compiling of `RansacFit()` takes on average $50[s]$. The table in figure 4.1 shows the average CPU and memory load as well as the processing time for each frame.

Jetson Nano performance	
CPU load [1 core]	80 [%]
Memory usage	7.5 [%]
`mainRealTime.py` computing time	0.6 [s]

Table 4.1: Average load and performance while running `mainRealTime.py` on the Jetson Nano.

4.1.2 Template matching

The template matching algorithm `TemplateMatching()` localises the specified template feature, returning the top-left corner position in the image as well as the width and height of the segmented image region. Using the OpenCV method `cv.TM_CCOEFF_NORMED` to calculate the correlation between the input image and template has proven to be a valid method for detection and localisation of specified features. Real-time implementation of the algorithm with the IR camera and experimental tapping beam have provided the results shown in figure 4.1 and 4.2.

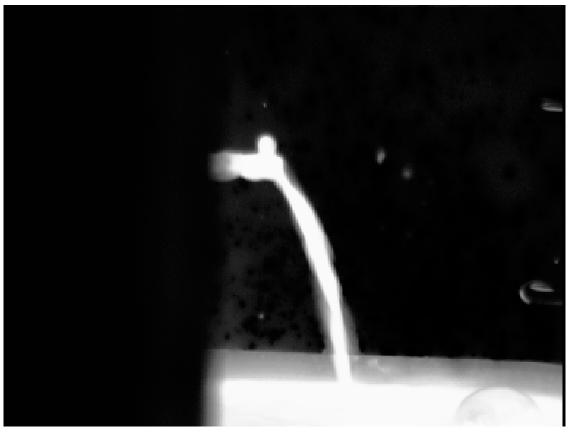


Figure 4.1: Input thermal image to the template matching.

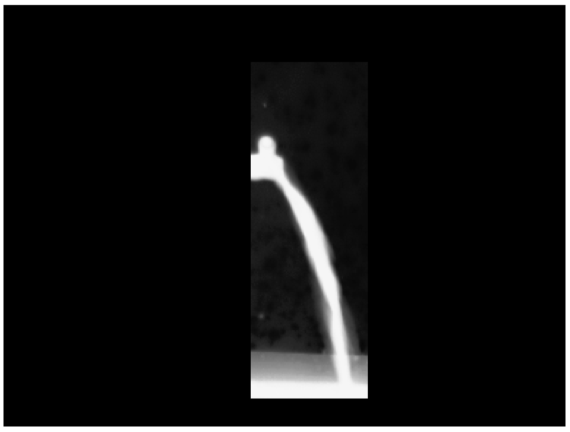


Figure 4.2: Isolated tapping beam image returned from template matching.

4.1.3 Histogram equalisation and detail enhancement

Increasing the level of detail within the tapping beam using histogram equalisation proved to be a successful method. In the early stages of development, detail enhancement on a grayscale thermal image directly from the IR camera resulted in only a minor change in detail level and did not provide the necessary separation between the tapping beam and background disturbances. A solution to this was to isolate the tapping beam ROI on beforehand, using the function `TemplateMatching()`. Histogram equalisation on the isolated image in figure 4.2 resulted in the image shown in figure 4.3.

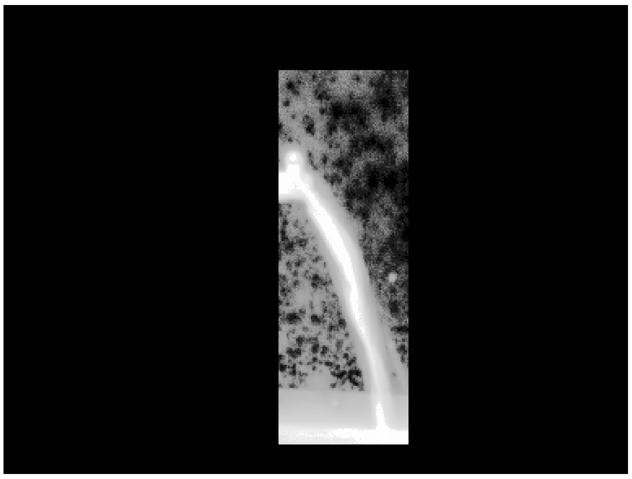


Figure 4.3: Histogram equalised image.

To substantiate the resulting image in figure 4.3, the histogram before and after equalisation can be examined. In figure 4.4, the histogram is denoted by a large collection of pixels close to 255, as well as close to grayscale value 0. This correlates to the unequalised image in figure 4.2, as most of the image pixels are black (i.e grayscale value close to 0) and the pixels within the detected template are mostly bright (i.e grayscale value close to 255). There are also some intermittent pixels with value between 50 and 200 in the unequalised histogram. Then, by looking at the equalised histogram in figure 4.5, it can be seen that the pixel values are now localised around 0 and in the range of 175 to 230. Effectively increasing the level of detail in the tapping beam.

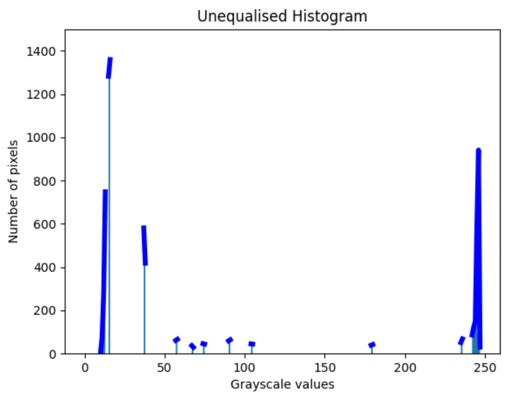


Figure 4.4: Unequalised histogram of figure 4.2.

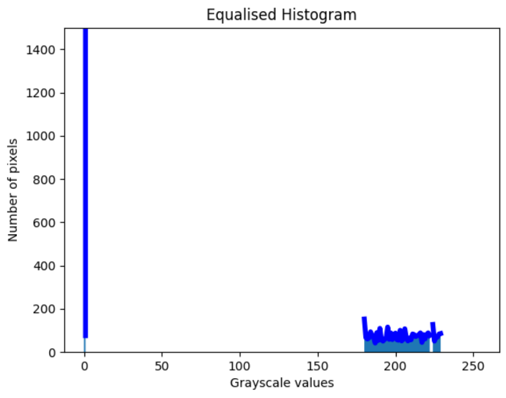


Figure 4.5: Histogram of the equalised image in figure 4.3.

By studying the cumulative distribution of the histogram after the equalisation, it can be verified that the cumulative histogram of the equalised image in figure 4.7 follows closely to a linear profile.

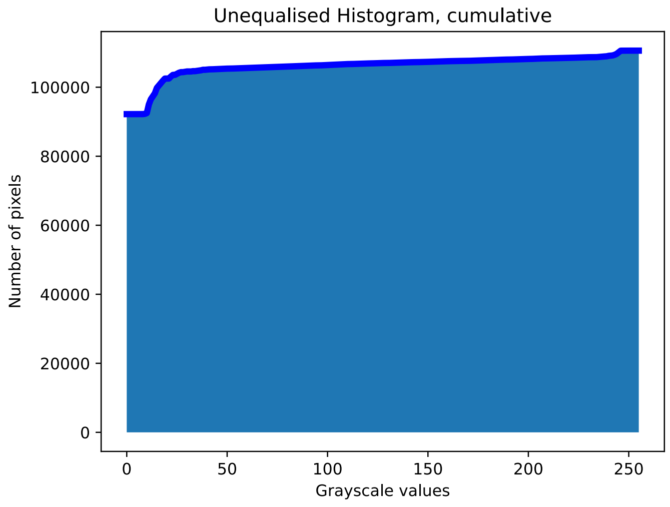


Figure 4.6: Unequalised cumulative histogram of figure 4.2.

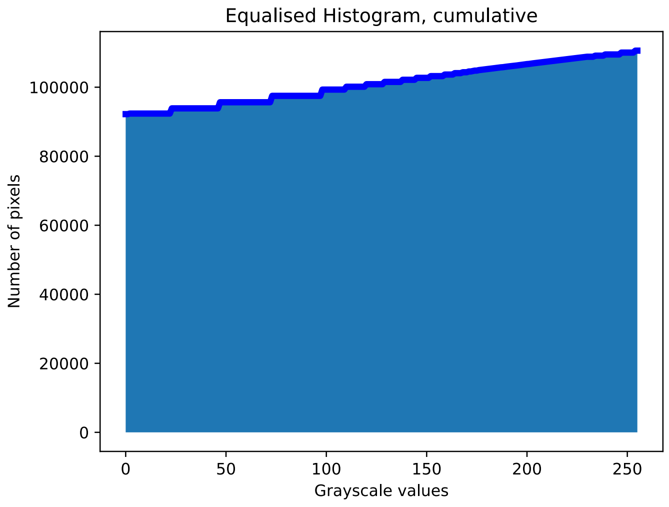


Figure 4.7: Cumulative histogram of the equalised image in figure 4.3.

4.1.4 Binary image

After equalisation, the binary image is created with function `BinaryImage()`. For this binarization, the threshold-gain variable is set to 255 as it gave the best representation of the tapping beam. Comparing the binary image to the thermal image in figure 4.1, shows that the binary representation of the tapping beam is valid.

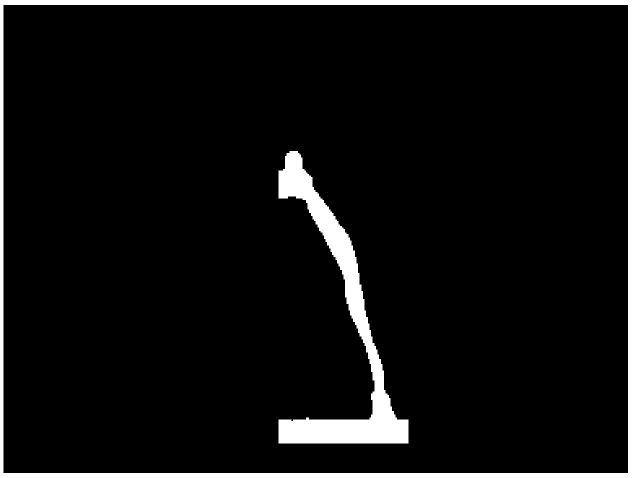


Figure 4.8: Resulting binary image

4.1.5 Inlier detection with RANSAC

The RANSAC algorithm manages to isolate the inliers of the tapping beam, with the image in figure 4.8 as the input dataset. For the results listed here, the parameters for the RANSAC algorithm is set to:

Number of iterations = 1000
Number of samples for model parameters = 4
Minimum number of inliers = 50
Inlier distance threshold = 3

Resulting in a filtered image containing the inliers of the tapping beam in figure 4.2 and the model parameters for the fitted line in table 4.3. Note that the values are based upon the image coordinate system, where the origin is located in the top left corner, and the y axis is pointing downwards.

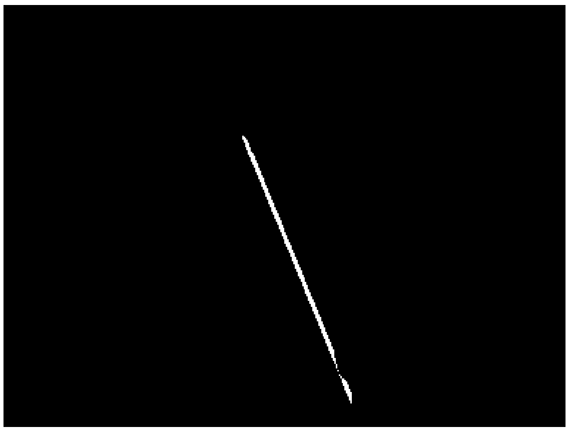


Table 4.2: Detected inliers from binary image in figure 4.8.

Model parameters,	Y(x) = Ax + B
Inclination, A	2.437
Y-axis crossing, B	-309.843

Table 4.3: Optimised parameters for the RANSAC linear model.

Just-In-Time compilation

The use of Numba to JIT compile the RANSAC function have resulted in a significant decrease in computing time, removing on average $0.861[s]$ from the execution time when compared to regular Python and NumPy.
Making the JIT compiled version approximately 3 times faster when used in real-time with the tapping beam test stand. To further prove the value of Numba and JIT compiling of Python code, a test was conducted. The RANSAC algorithm was timed with its regular NumPy functions and with the JIT compiled functions while increasing the number of iterations in the RANSAC function from 500 to 15000. The results from this test is shown in figure 4.9.

Average execution time:	
With JIT	0.389 [s]
Without JIT	1.25 [s]

Table 4.4: Average execution time of the RANSAC-function, with and without JIT-compiling.

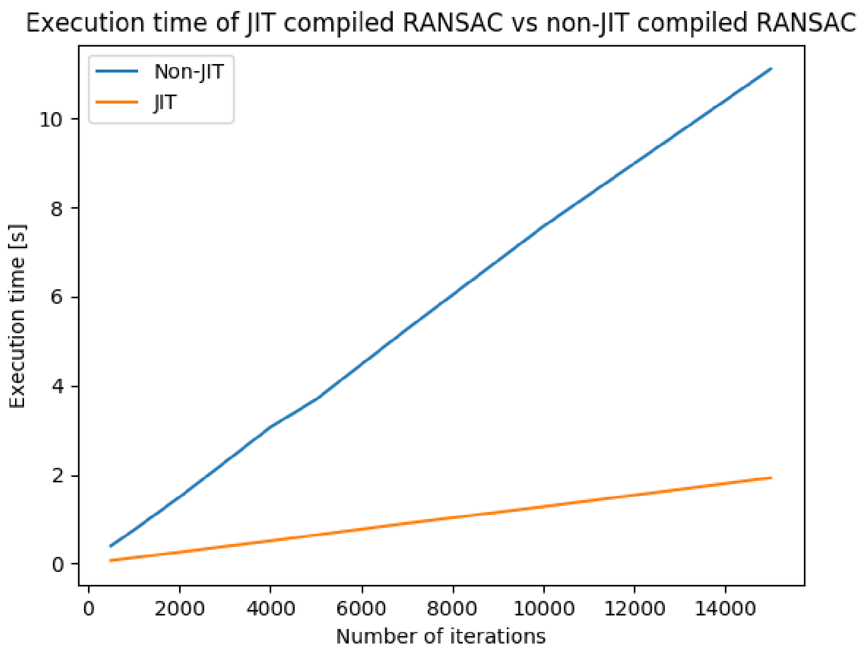


Figure 4.9: Effects of JIT-compilation with increasing computational load.

4.1.6 Beam shape classification

The edge detection algorithm used to classify the tapping beam as either continuous or split is tested using a synthetically developed image; based upon images of a split tapping beam from the plant in Thamshavn. This is mostly due to limitations of the tapping beam test-stand in combination with the available images from the physical tapping beam. The challenges with the split tapping beam are further explained in the discussion. Figure 4.10 shows the characteristics of a split tapping beam, and figure 4.11 shows the synthetic image. In total, 5 images were created to test the edge detection performance, one of them is shown in the figure below, and the rest are attached in the appendix B.2.3.

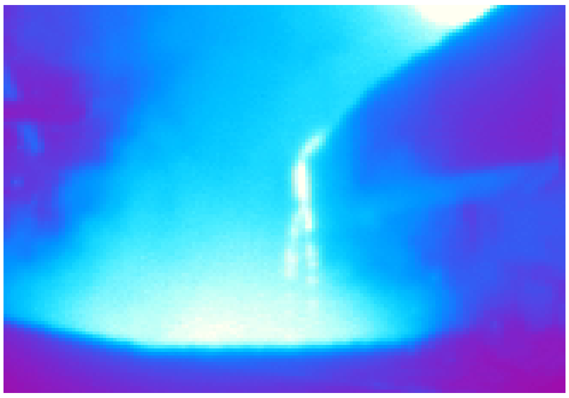


Figure 4.10: Observation of a split tapping beam at the plant in Thamshavn.

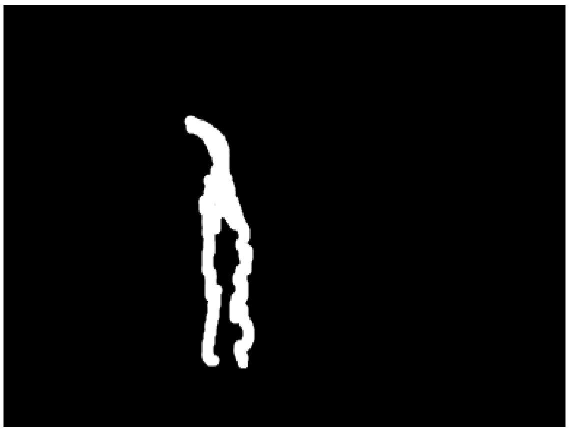


Figure 4.11: Synthetically created image of a divided tapping beam.

Table 4.5 lists the number of detected edges in figure 4.11, where the horizontal image vectors are sampled for every 20 pixels vertically. Note that the y-coordinate column is based upon the template coordinate system. The average number of detected edges in this specific image is 3, categorising it as a *split tappingbeam*.

Detected edges:	
Y-coordinate	Number of edges
0	0
20	3
40	2
60	2
80	4
100	4
120	4
140	4
160	4

Table 4.5: Number of detected edges

4.2 Human Machine Interface and database

This section presents the results from the web-based HMI and data communication to the computer vision application `mainRealTime.py`.

4.2.1 Overall results

Development of the HMI application `mainFlask.py`, has resulted in a minimalist interface for the computer vision application, providing the users with the information that the computer vision algorithms produce. The information is displayed in real-time through video-streaming routes and parsing of *JSON* data from the `home.py` view-script to the HTML template. Structuring the front-end layout with *Bootstrap* made it easy to define columns and position of these elements, allowing the HMI to responsively fit the layout to the specific browser window size. An image of the HMI home screen is attached in appendix B.5. Figure 4.12 and 4.13 shows the video stream from the IR camera and the tapping beam status labels.

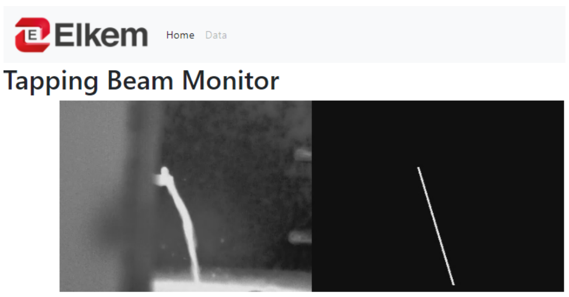


Figure 4.12: HMI video stream.

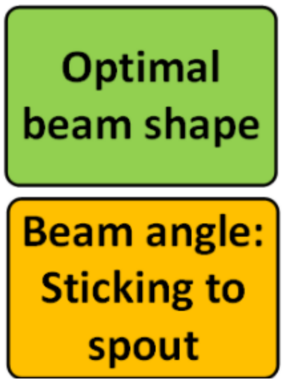


Figure 4.13: HMI status labels.

4.2.2 SQLite database

The SQLite database provided a simple framework for sharing data between applications running in separate threads and a stable platform for logging data over an extended time. A sample of the logged temperature and image data is available in appendix B.4.2 and B.4.2. Both the `mainRealTime.py` and `mainFlask.py` runs in their own separate thread while data is transferred in real-time, using the multi-thread capabilities of SQLite.

4.2.3 Temperature data

Displaying the temperature data with a line chart from the JavaScript package *Chart.js*, resulted in a responsive graph with automatic updating of temperature values and the option the select and deselect specific data.

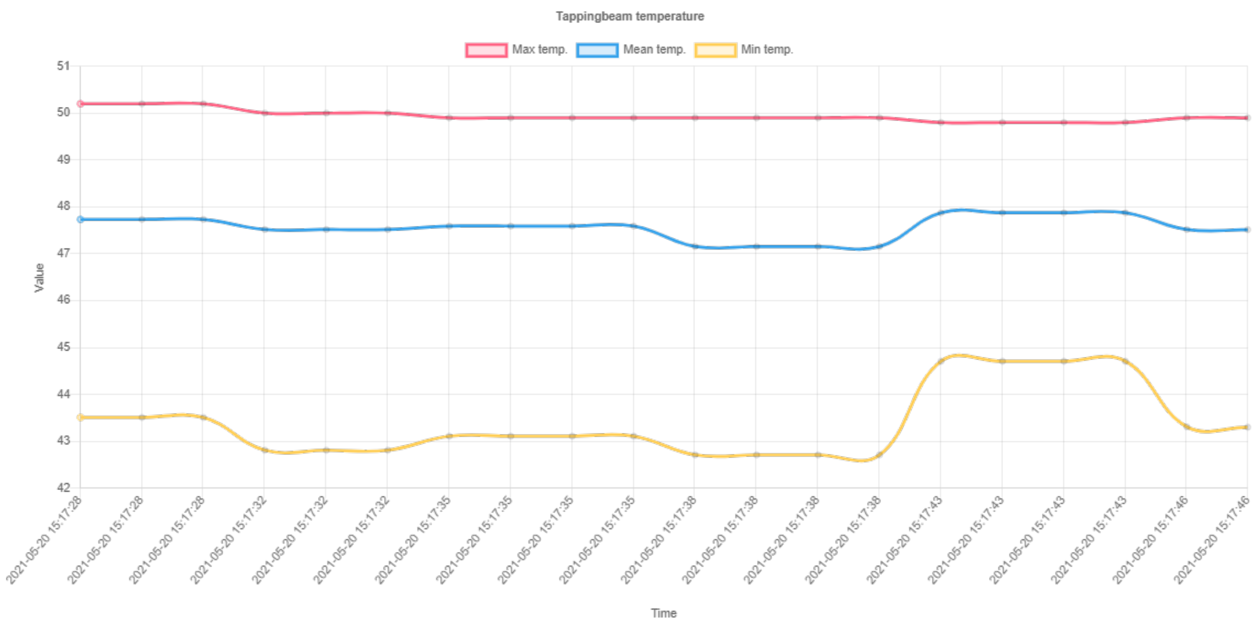


Figure 4.14: HMI temperature graph made with Chart.js

Chapter 5

Discussion and further work

The purpose of this chapter is to discuss the presented results, observations made throughout the project and the application framework as a whole. Possible solutions and improvements are also presented as further work under each section.

5.1 Position of IR camera at Elkem Thamshavn

In the early stages of development, video and image data from the physical tapping beam at Elkem's plant in Thamshavn, Orkanger, were used to develop the first iterations of the feature tracking, histogram equalisation and binary image algorithms. Data was accessed via a remote login using a Virtual Private Network (VPN) connection, this made it possible to directly interact with the IR camera software and capture image data of the tapping process. But despite the ease of accessibility, the quality and level of detail in the images did not suffice to be used for any further development with RANSAC and beam shape classification. The most significant challenge with the IR images from Thamshavn was the poor resolution of the tapping beam in the image, as a result of the long distance between the tapping area and the IR camera. The low resolution made it difficult to extract detailed information about the tapping beam, nonetheless develop functional algorithms. This was also the primary motivation for building the tapping beam test stand, as it provided a simple platform for the development, testing and verification of the algorithms.

Further work: Fit the IR camera with a telephoto lens that can make the tapping beam ROI cover more of the image area, i.e. increasing the number of pixels in the width and height of the tapping beam. Or simply dismantle the camera from its wall-mount and position it close to the tapping beam before recording sets of image data. For instance, it could now be possible to develop algorithms for slag detection and estimation of the tapping beam's mass flow rate.

5.2 Detection of tapping beam

During testing of the template matching algorithm developed in this project to localise and isolate the tapping beam, it became clear that the OpenCV function `cv.matchTemplate()` had some limitations in regards to the robustness of the template matching. These limitations were visible once testing with the tapping beam test stand started.

In the development phase, images from the IR camera at the plant in Thamshavn were used to validate if the OpenCV-function would be adequate to find the position of the tapping beam. A series of video frames from the IR camera at Thamshavn was then tested in the OpenCV template matching function, returning the position of the tapping beam for every frame. Thus, indicating that it would be a valid method for finding the location of the tapping beam. However, testing with the tapping beam test-stand proved otherwise, as the OpenCV template matching function was susceptible to small changes in the shape of the test beam that occurred when varying the pump voltage to simulate the different beam shapes. Resulting in losing the position of the tapping beam. Seeing as this is a potential issue for the overall functionality of the IR camera application, it would be of interest to revise the method for localisation.

Further work: By gathering an image dataset from the tapping beam, it is possible to create training data for a neural network that can localise the position of the tapping beam. This method is called regression, and it can find the image coordinates of features defined in the training dataset. As an example, the NVIDIA Deep Learning Institute offers an introductory level course on AI on the Jetson Nano, where one of the sections in the course is to train a neural network for regression [42]. In this course, the basics of *deep learning* and neural networks are explained, and the pre-trained ResNet-18 network is trained with data gathered through a connected USB web camera. The computational load necessary to train and use the network is handled by the NVIDIA 128-core Maxwell GPU of the Jetson Nano. A motivating factor for this solution is that the use of neural networks and *deep learning* in the process industry is already in development through projects like SINTEF's COGNITWIN [43].

5.3 Application structure

The suggested application structure presented in this thesis is perhaps not the most optimal in regards to computational load and in keeping a high frame rate video stream from the IR camera to the HMI. As it is presented in section 3.2.6, the IR camera is initiated in `mainRealTime.py` together with all of the other computer vision functions. There the thermal image frames are only written to the GStreamer pipeline once every iteration of the while-loop in `mainRealTime.py`, resulting in a low frame-rate stream of the original thermal image in the HMI. Which is not optimal for the operators working in the tapping area.

Further work: Since the execution time of RANSAC and the tapping beam detection algorithm's does not suffice for a video feed of 30 [FPS], a solution is to restructure the application, so the IR camera is initiated by the HMI-application and streams images to `mainRealTime.py` via GStreamer. This would yield slower feedback on parameters such as tapping beam existence and temperature but an increased frame rate of the original thermal image in the HMI. This suggestion would also require two GStreamer pipelines to be established.

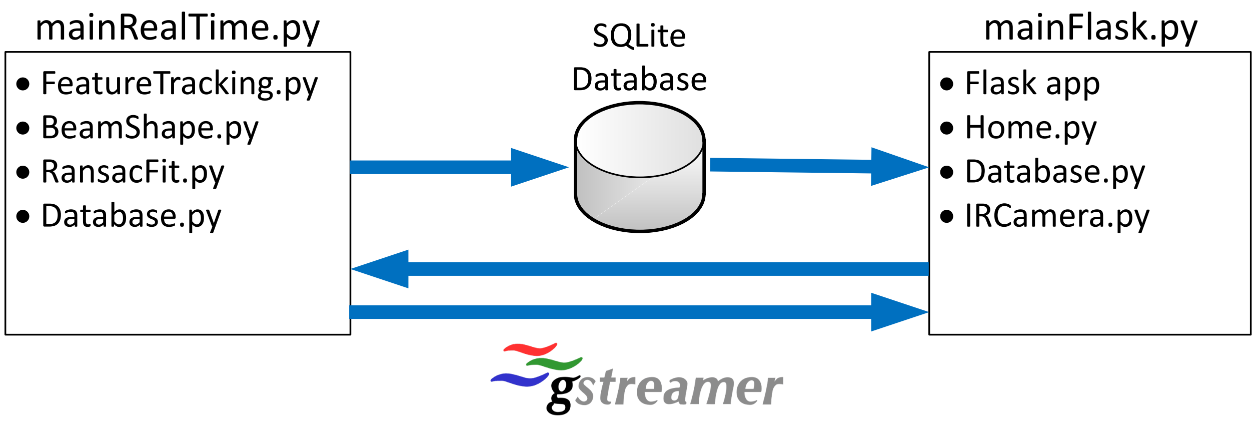


Figure 5.1: Suggested application structure

5.4 Jetson Nano performance

Throughout testing and developing the computer vision algorithms in this project, the Jetson Nano single-board computer has been used as the main platform. The reason for choosing the Nano was the possibility of using the GPU to perform training and processing of neural network algorithms that could perform classification and regression on images. As the application presented in section 3 is structured, only the CPU of the Nano is used for computing, making it equivalent to a Raspberry Pi 4. Even though Numba and JIT-compiling increases the performance quite significantly, it lacks support for the complete OpenCV library and some NumPy functions. With this in mind, it is a possibility that utilising the GPU for parallel computing of the OpenCV and NumPy functions would significantly boost the applications real-time performance. An example on the benefits of using OpenCV with GPU acceleration is shown in article [44].

Furhter work: For accelerating the performance NumPy functions, the open-source CuPy library offers GPU optimising using NVIDIA CUDA cores. Where as OpenCV already has a CUDA toolkit available. For it to be installed, the flag `WITH_CUDA=ON` must have been set in the initial build of OpenCV using CMake. Then combining these functions with the suggested application structure discussed in the previous section 5.3.

Chapter 6

Conclusion

The research and methods developed in this project aimed to solve the challenge of analysing a complex industrial process with a high level of uncertainties and disturbances, while providing the operators with real-time feedback on the specific conditions of the process. A application that uses an industrial IR camera to extract information about the ferrosilicon tapping beam was developed. Requirements for the application is that the tapping beam must be the most considerable feature in the image, i.e. represented by as many pixels as possible, and a template image of the specific tapping beam must be captured. The computer vision algorithms can then be launched, and data on the tapping beam gets saved to the database. After validation of the image and temperature data, the tapping beam HMI can be launched. With the HMI running, live feed of the tapping beam thermal image and detected inliers, status labels, and temperature graph is available as a locally hosted website.

Throughout this project, several different subjects have been investigated, evaluated and tested. These include; feature localisation, image detail enhancement, binary image thresholding, inlier detection with RANSAC and edge detection to determine beam shape.

- **Feature localisation:** Finding the position of the tapping beam and isolating the specific ROI was solved using a function within OpenCV. And as presented in the discussion at section 5.2, this method proved to be a suitable solution when testing with low resolution image data sets from the physical tapping beam at the plant in Thamshavn, but proved otherwise when testing with the tapping beam test stand. Arguably, it could have been the test stand that gave the template matching such poor performance. Still, when taking into account the uncertainties and process disturbances such as smoke and human operators blocking the camera line-of-sight, it can be concluded that more research and testing is necessary to develop a robust algorithm specific to tapping beam detection.

- **Detail enhancement:** Increasing the level of detail in the image of the isolated tapping beam was necessary to create a robust definition between the background and foreground objects. Histogram equalisation, where the cumulative distribution of the equalised image follows a linear model, has proved to be a good algorithm for increasing detail and contrast. This algorithm is necessary when dealing with images from the physical ferrosilicon tapping beam at the plant in Thamshavn, as the beam is surrounded by high-temperature smoke. It can be concluded that detail enhancement helps to better represent the tapping beam, but needs to be validated with images where the actual tapping beam is the most significant feature.

- **Binary image thresholding:** Transforming the equalised image into a binary representation consisting only of pixel values 0 and 255 was necessary for the inlier detection with RANSAC and the edge detection algorithm for the beam shape classification. This was done by calculating an intensity value for thresholding with the max and mean image intensities and an adjustable gain parameter. The gain parameter made it possible to manipulate the thresholding value to give a better binary representation of the tapping beam. The algorithm worked as intended and has yielded credible results.

- **Inlier detection:** The inlier detection algorithm was developed on the assumption that the tapping beam mostly follows a vertical line, which gave the reason for using a RANSAC algorithm. This was also confirmed with the image data set from the tapping beam at the plant in Thamshavn. The resulting algorithm iterates through the input dataset and finds the tapping inliers with the specified distance threshold and a minimum number of inliers. With the detected inliers, the tapping beam angle is calculated and saved to the database. Using RANSAC to filter out the tapping beam profile and finding the inliers for a 2D line proved to be a successful implementation. But it needs to be tested on a physical ferrosilicon tapping beam.

- **Beam shape classification:** Classification of the tapping beam by sampling horizontal image vectors of the isolated tapping beam and counting the number of edges in each vector gave good results. But further testing is necessary, as the images for testing this algorithm are produced synthetically.

To summarise, the research conducted in this project, along with the methods and software developed, has answered the project scope and questions of research presented in section 1.3. A method for measuring the temperature, algorithms for detecting and extracting information about the tapping beam and an HMI for visualisation of the data has been developed. Despite the challenges met and the simplifications that were done to achieve results within the given time frame. A system that enhances the level of information about the tapping beam in real-time for the process operators and logs data for the metallurgists were developed. The application can also serve as a platform for gathering image and temperature data for developing more advanced algorithms, with the COGNITWIN project as an example.

Bibliography

[1] K. Sogner, Creative Power, En. Elkem, Oslo 2014: Messel Forlag AS, ISBN: 9788276311242.

[2] Elkem ASA. (2021). "About Elkem," [Online]. Available: https://www.elkem.com/about-elkem/ (visited on 03/15/2021).

[3] ——, (2021). "Sustainability," [Online]. Available: https://www.elkem.com/sustainability/ (visited on 03/13/2021).

[4] Global Silicones Council. (2016). "Socio-economic evaluation of the global silicones industry," [Online]. Available: https://sehsc.americanchemistry.com/Socio-Economic-Evaluation-of-the-Global-Silicones-Industry-Final-Report.pdf (visited on 01/20/2021).

[5] T. Hannesson. (2016). "The si process," [Online]. Available: https://www.elkem.com/globalassets/iceland/si-process.pdf (visited on 05/20/2021).

[6] Elkem ASA, General information about the plant in thamshavn, 2021.

[7] A. Szajewska, "Development of the thermal imaging camera (tic) technology," Procedia Engineering, vol. 172, pp. 1067–1072, 2017, Modern Building Materials, Structures and Techniques, ISSN: 1877-7058. DOI: https://doi.org/10.1016/j.proeng.2017.02.164. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877705817306707.

[8] A. Glowacz and Z. Glowacz, "Diagnosis of the three-phase induction motor using thermal imaging," Infrared Physics and Technology, vol. 81, pp. 7–16, 2017, ISSN: 1350-4495. DOI: https://doi.org/10.1016/j.infrared.2016.12.003. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1350449516306259.

[9] Z. Zhang, L. Bin, and Y. Jiang, "Slag detection system based on infrared temperature measurement," Optik, vol. 125, no. 3, pp. 1412–1416, 2014, ISSN: 0030-4026. DOI: https://doi.org/10.1016/j.ijleo.2013.08.016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0030402613011790.

[10] Environmental-Expert. (2021). "Elkem carbon - model elsep - elkem søderberg electrode paste," [Online]. Available: https://www.environmental-expert.com/products/elkem-carbon-model-elsep-elkem-soderberg-electrode-paste-516916 (visited on 05/18/2021).

[11] M. Miranda-Martínez, J. R. Campello-García, and D. Castaño-Laviana, "Development of soderberg carbon electrodes bonded with a carbon nanofibre reinforced coal-tar pitch," Refractories world forum, vol. 11, no. 1, pp. 63–67, 2018. [Online]. Available: https://www.refractories-worldforum.com/paper?article_id=100689.

[12] A. S. Hauksdóttir, A. Gestsson, and A. Vésteinsson, "Current control of a three-phase submerged arc ferrosilicon furnace," Control Engineering Practice, vol. 10, no. 4, pp. 457–463, 2002, Mechatronics, ISSN: 0967-0661. DOI: https://doi.org/10.1016/S0967-0661(01)00104-6. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0967066101001046.

[13] Teknisk Ukeblad, Per-Ivar Nikolaisen, Slik kuttet elkem utslipp tilsvarende 150.000 dieselbiler på rekordtid, [Online; accessed April 28, 2021], 2014. [Online]. Available: https://img.gfx.no/1702/1702551/_MG_0431_6212.jpg.

[14] Store Norske leksikon. (2021). "Infrarød stråling," [Online]. Available: https://snl.no/infrar%5C%C3%5C%B8d_str%5C%C3%5C%A5ling (visited on 04/29/2021).

[15] Wikipedia Commons, Em spectrum, [Online; accessed April 30, 2021], 2021. [Online]. Available: https://commons.wikimedia.org/wiki/File:EM_spectrum.svg.

[16] Optris GmbH. (2021). "What web cams and ir cameras have in common," [Online]. Available: https://www.optris.global/how-thermal-imaging-cameras-and-thermography-work (visited on 01/22/2021).

[17] Optotherm, Inc., Microbolometers, [Online; accessed April 30, 2021], 2018. [Online]. Available: https://www.optotherm.com/microbolometers.htm.

[18] M. Kimata, "Uncooled infrared focal plane arrays," IEEJ Transactions on Electrical and Electronic Engineering, vol. 13, no. 1, pp. 4–12, 2018. DOI: https://doi.org/10.1002/tee.22563. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/tee.22563.

[19] Python Software Foundation, What is python? executive summary. [Online]. Available: https://www.python.org/doc/essays/blurb/ (visited on 05/08/2021).

[20] The SciPy community, What is numpy? [Online]. Available: https://numpy.org/doc/stable/user/whatisnumpy.html (visited on 05/08/2021).

[21] LLVM Project, The llvm compiler infrastructure. [Online]. Available: https://llvm.org/ (visited on 05/08/2021).

[22] Towards data science: Puneet Grover, What is numpy? [Online]. Available: https://miro.medium.com/max/700/0*bJ6XIUE05phjWZgz (visited on 05/08/2021).

[23] OpenCV team. (2021). "About - OpenCV," [Online]. Available: https://opencv.org/about/ (visited on 05/25/2021).

[24] GStreamer Team, What is gstreamer? [Online]. Available: https://gstreamer.freedesktop.org/documentation/application-development/introduction/gstreamer.html?gi-language=c (visited on 05/08/2021).

[25] Pallets Projects, Flask: Foreword. [Online]. Available: https://flask.palletsprojects.com/en/1.1.x/foreword/ (visited on 05/08/2021).

[26] Simplilearn - Online Certification Training Course Provider. (2021). "What is gitlab and how to use it," [Online]. Available: https://www.simplilearn.com/tutorials/git-tutorial/what-is-gitlab (visited on 05/10/2021).

[27] OpenCV-Python Tutorials. (2020). "Opencv-python tutorials: Histogram equalisation," [Online]. Available: [https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_template_matching/py_template_matching.html] (visited on 05/03/2021).

[28] ——, (2020). "Opencv-python tutorials: Template matching," [Online]. Available: [https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_template_matching/py_template_matching.html] (visited on 05/03/2021).

[29] A. Manzur. (2019). "Got outliers? ransac them!" [Online]. Available: https://medium.com/@angel.manzur/got-outliers-ransac-them-f12b6b5f606e (visited on 05/04/2021).

[30] MathWorld–A Wolfram Web Resource: Weisstein, Eric W. (2021). "Convolution," [Online]. Available: https://mathworld.wolfram.com/Convolution.html (visited on 05/14/2021).

[31] M. Ottestad, "Lectures in instrumentation: Edge Detection Concepts in 1-D," Lecture notes from MAS506 Instrumentation at the University of Agder, 2021.

[32] TeamGantt. (2021). "TeamGantt," [Online]. Available: https://www.teamgantt.com/ (visited on 05/18/2021).

[33] Atlassian. (2021). "What is kanban?" [Online]. Available: https://www.atlassian.com/agile/kanban (visited on 05/18/2021).

[34] ——, (2021). "Trello," [Online]. Available: https://trello.com/ (visited on 05/18/2021).

[35] ——, (2021). "What is Agile?" [Online]. Available: https://www.atlassian.com/agile (visited on 05/18/2021).

[36] Optris GmbH, Infrared camera optris pi 400i. [Online]. Available: https://www.optris.global/thermal-imager-optris-pi-400i-pi-450i (visited on 05/06/2021).

[37] NVIDIA Corporation. (2021). "Jetson Nano Developer Kit," [Online]. Available: https://developer.nvidia.com/embedded/jetson-nano-developer-kit (visited on 05/23/2021).

[38] Evocortex, Irimagerdirectsdk. [Online]. Available: https://evocortex.org/products/irimagerdirect-sdk/ (visited on 05/06/2021).

[39] Python Software Foundation, Ctypes — a foreign function library for python. [Online]. Available: https://docs.python.org/3/library/ctypes.html (visited on 05/06/2021).

[40] Evocortex, Irimagerdirectsdk, linux installation. [Online]. Available: http://documentation.evocortex.com/libirimager2/html/Installation.html (visited on 05/06/2021).

[41] Python Software Foundation. (2021). "Db-api 2.0 interface for sqlite databases," [Online]. Available: https://docs.python.org/3/library/sqlite3.html (visited on 05/16/2021).

[42] NVIDIA Corporation: Deep Learning Institute. (). "Getting started with ai on jetson nano," [Online]. Available: https://courses.nvidia.com/courses/course-v1:DLI+S-RX-02+V2/about (visited on 05/21/2021).

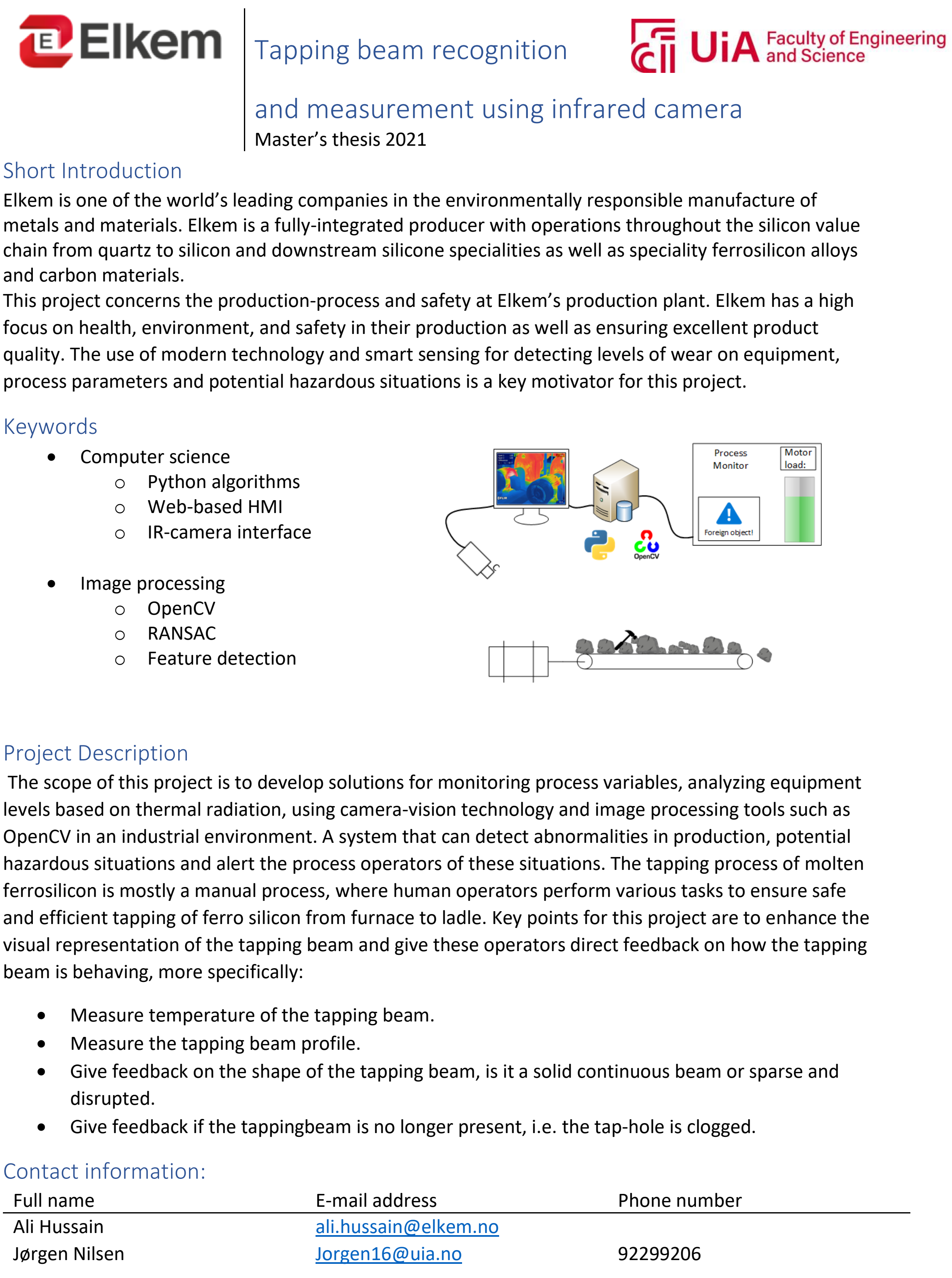
[43] SINTEF. (2021). "Cognitwin - cognitive plants through proactive self-learning hybrid digital twins," [Online]. Available: https://www.sintef.no/projectweb/cognitwin/ (visited on 05/21/2021).

[44] K. Pulli, A. Baksheev, K. Kornyakov, and V. Eruhimov, "Realtime computer vision with opencv: Mobile computer-vision technology will soon become as ubiquitous as touch interfaces.," Queue, vol. 10, no. 4, pp. 40–56, Apr. 2012, ISSN: 1542-7730. DOI: 10.1145/2181796.2206309. [Online]. Available: https://doi.org/10.1145/2181796.2206309.

Appendix A

Administrative

A.1 Project description



Tapping beam recognition

and measurement using infrared camera

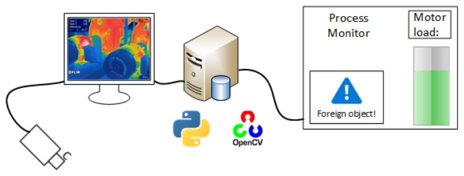
Master's thesis 2021

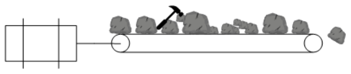
Short Introduction

Elkem is one of the world's leading companies in the environmentally responsible manufacture of metals and materials. Elkem is a fully-integrated producer with operations throughout the silicon value chain from quartz to silicon and downstream silicone specialities as well as speciality ferrosilicon alloys and carbon materials.
This project concerns the production-process and safety at Elkem's production plant. Elkem has a high focus on health, environment, and safety in their production as well as ensuring excellent product quality. The use of modern technology and smart sensing for detecting levels of wear on equipment, process parameters and potential hazardous situations is a key motivator for this project.

Keywords

- Computer science
  - Python algorithms
  - Web-based HMI
  - IR-camera interface
- Image processing
  - OpenCV
  - RANSAC
  - Feature detection





Project Description

The scope of this project is to develop solutions for monitoring process variables, analyzing equipment levels based on thermal radiation, using camera-vision technology and image processing tools such as OpenCV in an industrial environment. A system that can detect abnormalities in production, potential hazardous situations and alert the process operators of these situations. The tapping process of molten ferrosilicon is mostly a manual process, where human operators perform various tasks to ensure safe and efficient tapping of ferro silicon from furnace to ladle. Key points for this project are to enhance the visual representation of the tapping beam and give these operators direct feedback on how the tapping beam is behaving, more specifically:

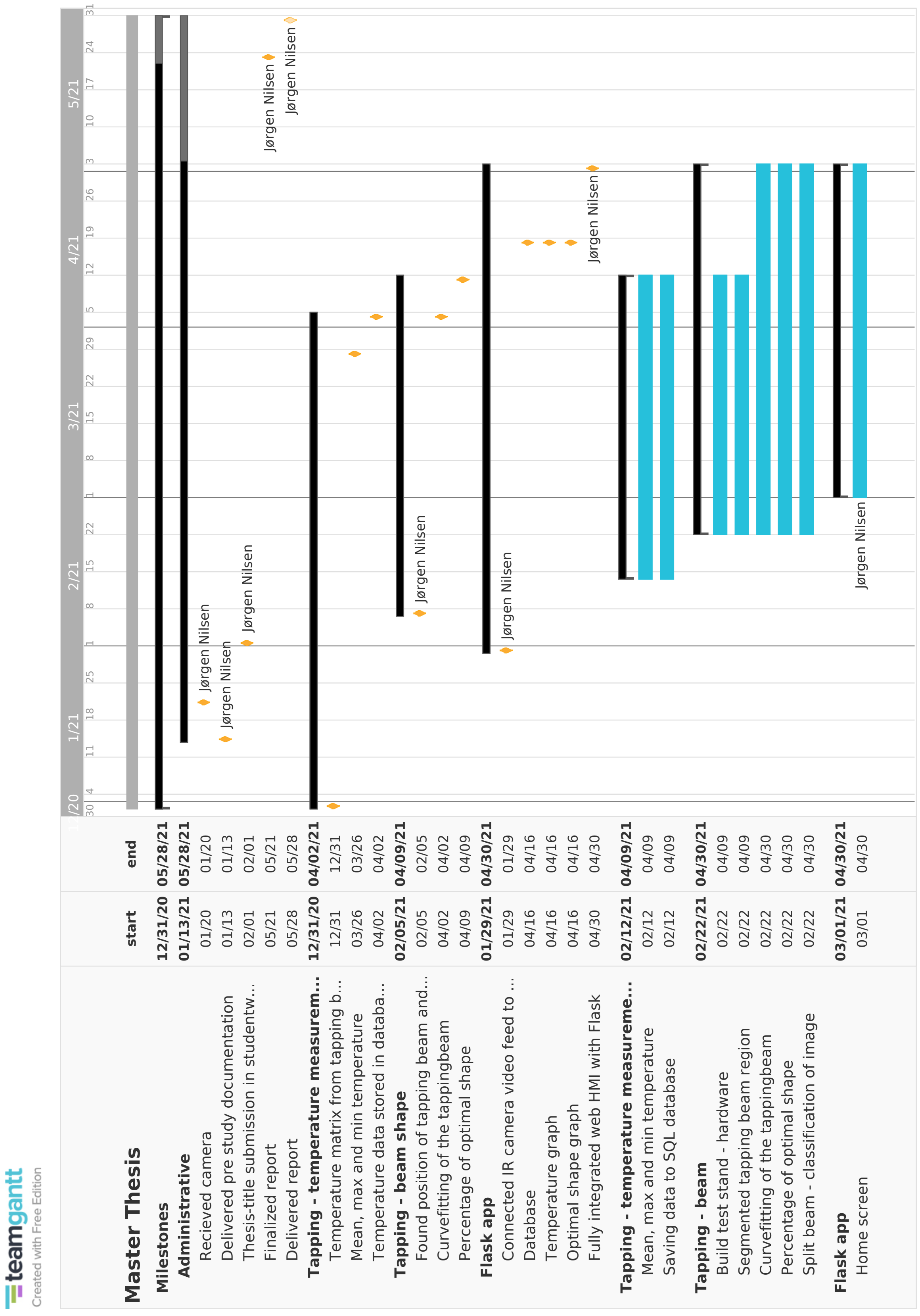
- Measure temperature of the tapping beam.
- Measure the tapping beam profile.
- Give feedback on the shape of the tapping beam, is it a solid continuous beam or sparse and disrupted.
- Give feedback if the tappingbeam is no longer present, i.e. the tap-hole is clogged.

Contact information:

Full name	E-mail address	Phone number
Ali Hussain	ali.hussain@elkem.no	
Jørgen Nilsen	Jorgen16@uia.no	92299206

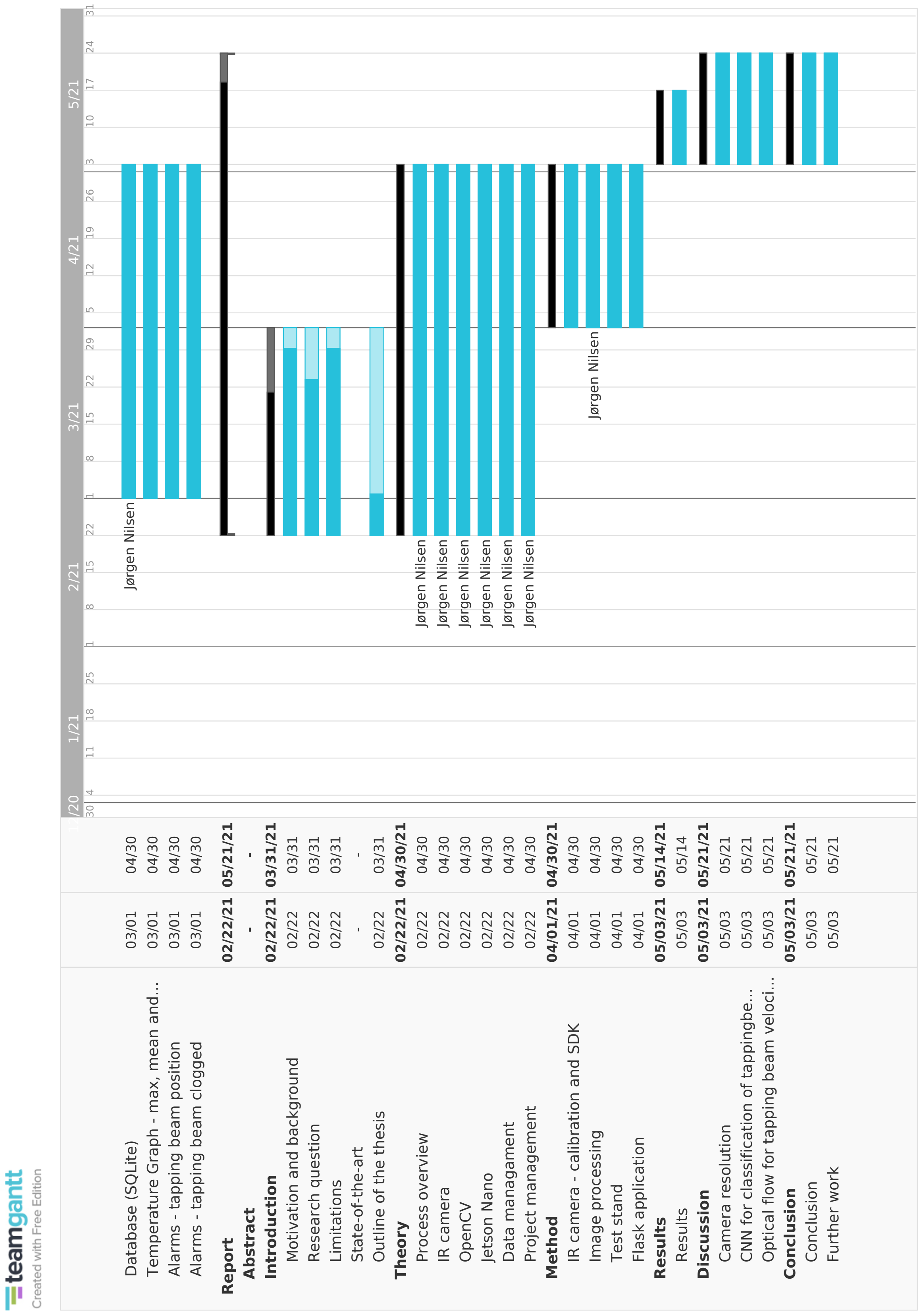
Figure A.1: Project description

A.2 Gantt chart



Master Thesis	start	end
Milestones	**12/31/20**	**05/28/21**
Administrative	**01/13/21**	**05/28/21**
Recieved camera	01/20	01/20
Delivered pre study documentation	01/13	01/13
Thesis-title submission in studentw...	02/01	02/01
Finalized report	05/21	05/21
Delivered report	05/28	05/28
Tapping - temperature measurem...	**12/31/20**	**04/02/21**
Temperature matrix from tapping b...	12/31	12/31
Mean, max and min temperature	03/26	03/26
Temperature data stored in databa...	04/02	04/02
Tapping - beam shape	**02/05/21**	**04/09/21**
Found position of tapping beam and...	02/05	02/05
Curvefitting of the tappingbeam	04/02	04/02
Percentage of optimal shape	04/09	04/09
Flask app	**01/29/21**	**04/30/21**
Connected IR camera video feed to ...	01/29	01/29
Database	04/16	04/16
Temperature graph	04/16	04/16
Optimal shape graph	04/16	04/16
Fully integrated web HMI with Flask	04/30	04/30
Tapping - temperature measureme...	**02/12/21**	**04/09/21**
Mean, max and min temperature	02/12	04/09
Saving data to SQL database	02/12	04/09
Tapping - beam	**02/22/21**	**04/30/21**
Build test stand - hardware	02/22	04/09
Segmented tapping beam region	02/22	04/09
Curvefitting of the tappingbeam	02/22	04/30
Percentage of optimal shape	02/22	04/30
Split beam - classification of image	02/22	04/30
Flask app	**03/01/21**	**04/30/21**
Home screen	03/01	04/30

Figure A.2: Gantt chart, page 1.



Database (SQLite)	03/01	04/30
Temperature Graph - max, mean and...	03/01	04/30
Alarms - tapping beam position	03/01	04/30
Alarms - tapping beam clogged	03/01	04/30
Report	**02/22/21**	**05/21/21**
Abstract	**-**	**-**
Introduction	**02/22/21**	**03/31/21**
Motivation and background	02/22	03/31
Research question	02/22	03/31
Limitations	02/22	03/31
State-of-the-art	-	-
Outline of the thesis	02/22	03/31
Theory	**02/22/21**	**04/30/21**
Process overview	02/22	04/30
IR camera	02/22	04/30
OpenCV	02/22	04/30
Jetson Nano	02/22	04/30
Data managament	02/22	04/30
Project management	02/22	04/30
Method	**04/01/21**	**04/30/21**
IR camera - calibration and SDK	04/01	04/30
Image processing	04/01	04/30
Test stand	04/01	04/30
Flask application	04/01	04/30
Results	**05/03/21**	**05/14/21**
Results	05/03	05/14
Discussion	**05/03/21**	**05/21/21**
Camera resolution	05/03	05/21
CNN for classification of tappingbe...	05/03	05/21
Optical flow for tapping beam veloci...	05/03	05/21
Conclusion	**05/03/21**	**05/21/21**
Conclusion	05/03	05/21
Further work	05/03	05/21

Figure A.3: Gantt chart, page 2.

Appendix B

System

B.1 System charts

B.1.1 Tapping process timeline

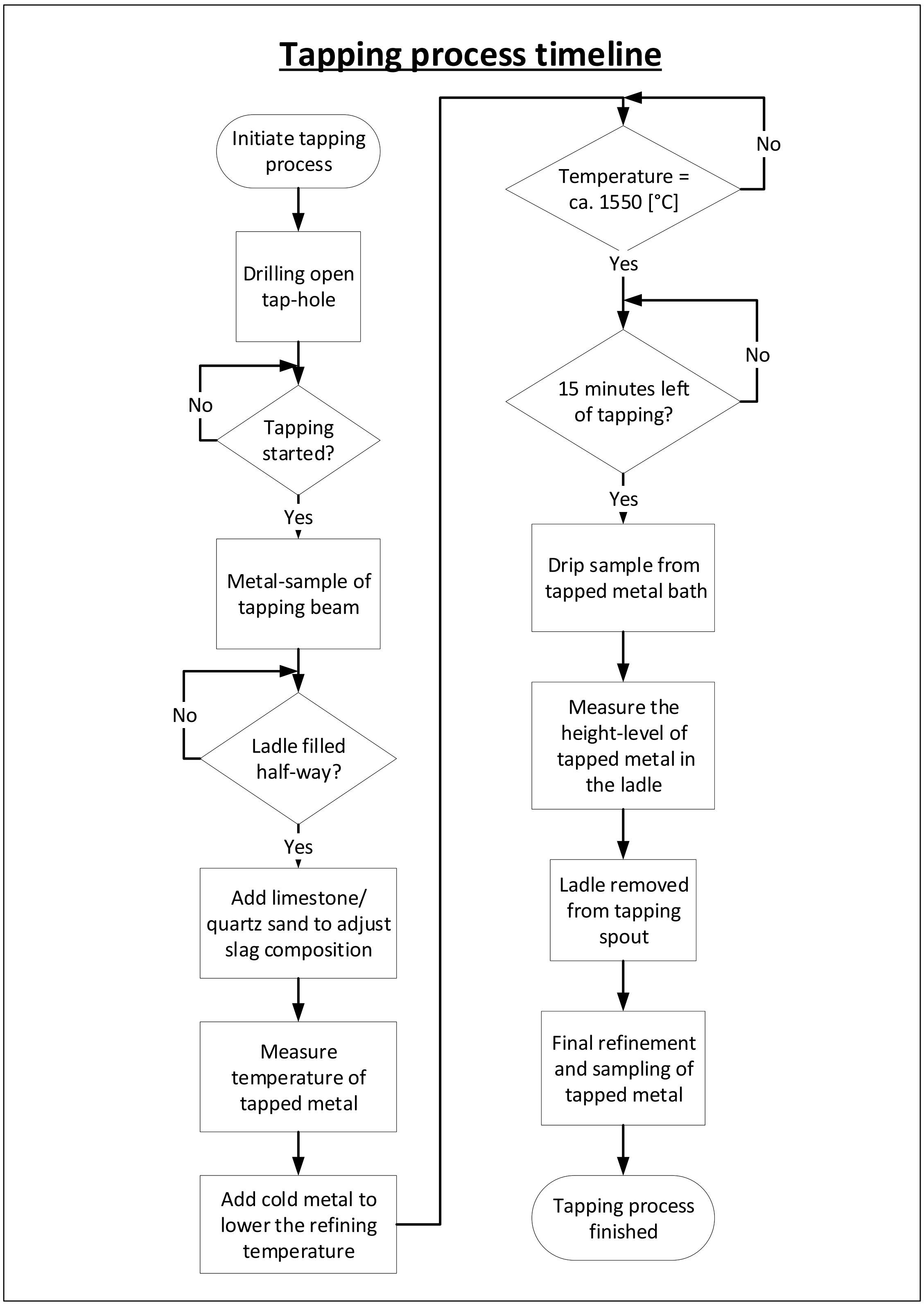
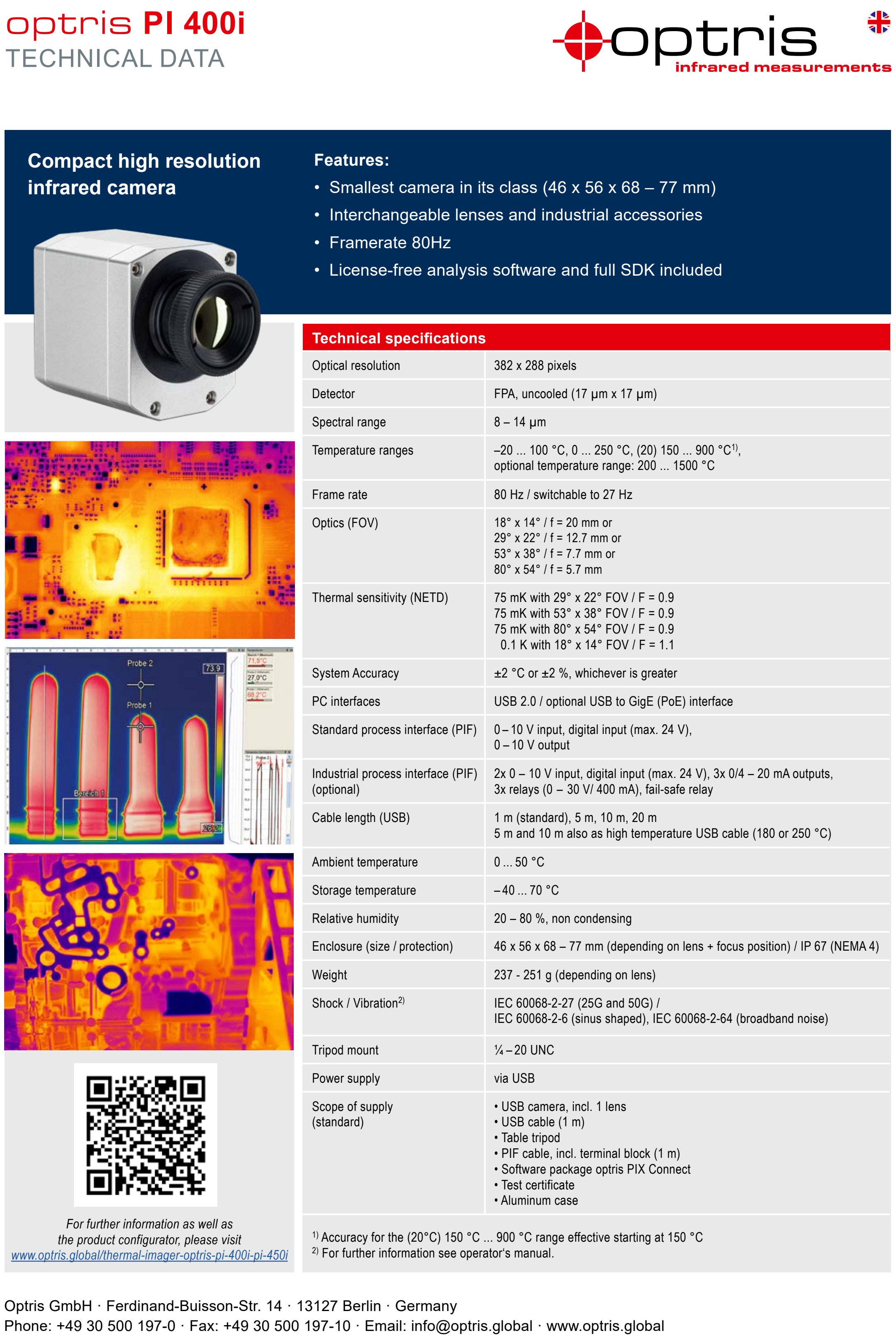

Tapping process timeline
Initiate tapping process
Drilling open tap-hole
No
Tapping started?
Yes
Metal-sample of tapping beam
No
Ladle filled half-way?
Yes
Add limestone/ quartz sand to adjust slag composition
Measure temperature of tapped metal
Add cold metal to lower the refining temperature
No
Temperature = ca. 1550 [°C]
Yes
No
15 minutes left of tapping?
Yes
Drip sample from tapped metal bath
Measure the height-level of tapped metal in the ladle
Ladle removed from tapping spout
Final refinement and sampling of tapped metal
Tapping process finished

Figure B.1: Tapping process manual operations timeline.

B.2 Datasheets

B.2.1 Optris PI 400i



optris **PI 400i**
TECHNICAL DATA

optris infrared measurements

Compact high resolution infrared camera

Features:

- Smallest camera in its class (46 x 56 x 68 – 77 mm)
- Interchangeable lenses and industrial accessories
- Framerate 80Hz
- License-free analysis software and full SDK included

Technical specifications	
Optical resolution	382 x 288 pixels
Detector	FPA, uncooled (17 µm x 17 µm)
Spectral range	8 – 14 µm
Temperature ranges	–20 ... 100 °C, 0 ... 250 °C, (20) 150 ... 900 °C[1], optional temperature range: 200 ... 1500 °C
Frame rate	80 Hz / switchable to 27 Hz
Optics (FOV)	18° x 14° / f = 20 mm or 29° x 22° / f = 12.7 mm or 53° x 38° / f = 7.7 mm or 80° x 54° / f = 5.7 mm
Thermal sensitivity (NETD)	75 mK with 29° x 22° FOV / F = 0.9 75 mK with 53° x 38° FOV / F = 0.9 75 mK with 80° x 54° FOV / F = 0.9 0.1 K with 18° x 14° FOV / F = 1.1
System Accuracy	±2 °C or ±2 %, whichever is greater
PC interfaces	USB 2.0 / optional USB to GigE (PoE) interface
Standard process interface (PIF)	0 – 10 V input, digital input (max. 24 V), 0 – 10 V output
Industrial process interface (PIF) (optional)	2x 0 – 10 V input, digital input (max. 24 V), 3x 0/4 – 20 mA outputs, 3x relays (0 – 30 V/ 400 mA), fail-safe relay
Cable length (USB)	1 m (standard), 5 m, 10 m, 20 m 5 m and 10 m also as high temperature USB cable (180 or 250 °C)
Ambient temperature	0 ... 50 °C
Storage temperature	– 40 ... 70 °C
Relative humidity	20 – 80 %, non condensing
Enclosure (size / protection)	46 x 56 x 68 – 77 mm (depending on lens + focus position) / IP 67 (NEMA 4)
Weight	237 - 251 g (depending on lens)
Shock / Vibration[2]	IEC 60068-2-27 (25G and 50G) / IEC 60068-2-6 (sinus shaped), IEC 60068-2-64 (broadband noise)
Tripod mount	¼ – 20 UNC
Power supply	via USB
Scope of supply (standard)	• USB camera, incl. 1 lens • USB cable (1 m) • Table tripod • PIF cable, incl. terminal block (1 m) • Software package optris PIX Connect • Test certificate • Aluminum case

[1] Accuracy for the (20°C) 150 °C ... 900 °C range effective starting at 150 °C
[2] For further information see operator's manual.

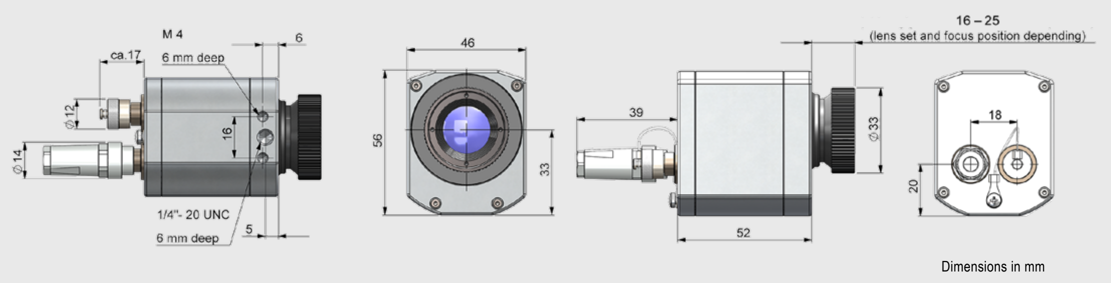
For further information as well as the product configurator, please visit www.optris.global/thermal-imager-optris-pi-400i-pi-450i

Optris GmbH · Ferdinand-Buisson-Str. 14 · 13127 Berlin · Germany
Phone: +49 30 500 197-0 · Fax: +49 30 500 197-10 · Email: info@optris.global · www.optris.global

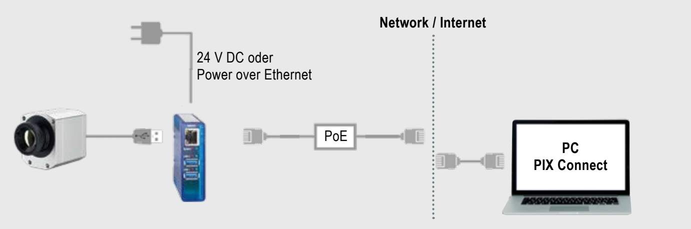
Figure B.2: Optris PI 400i datasheet, page 1.

optris **PI 400i**

Dimensions



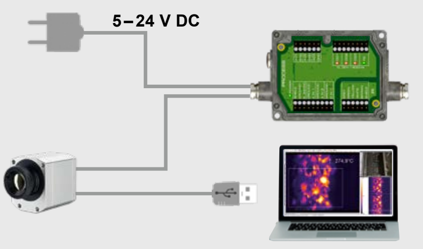
Process integration



optris USB-Server Gigabit 2.0

- Network connection via Gigabit Ethernet
- Full TCP/IP support incl. routing and DNS
- Two independent USB ports
- Power via PoE or external voltage supply at 24 – 48 V DC
- Galvanic isolation 500 V_{RMS}
- Remotely configurable via web based management

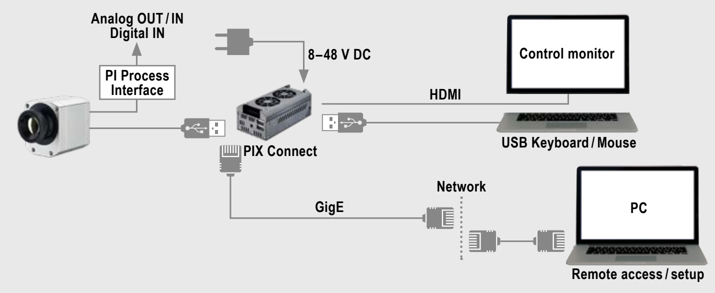
For further information please visit
www.optris.global/usb-server-gigabit



optris Industrial Process Interface

- Use of camera for process monitoring in industrial environments
- Continous fail safe monitoring of imager, software and cable connections
- 3 analog/ alarm outputs,
  2 analog inputs,
  1 digital input,
  3 alarm relays,
  1 fail-safe relay

For further information please visit
www.optris.global/neu-industrial-process-interface



optris PI NetBox

- Miniature PC as add-on to the PI series for stand-alone system
- Integrated hardware and software watchdog
- Connections: 2x USB 2.0, 1x USB 3.0, 1x Mini-USB 2.0, Micro-HDMI, Ethernet (Gigabit Ethernet), micro SDHC / SDXC card

For further information please visit
www.optris.global/pi-netbox

Figure B.3: Optris PI 400i datasheet, page 2.

B.2.2 IR Camera application flowchart

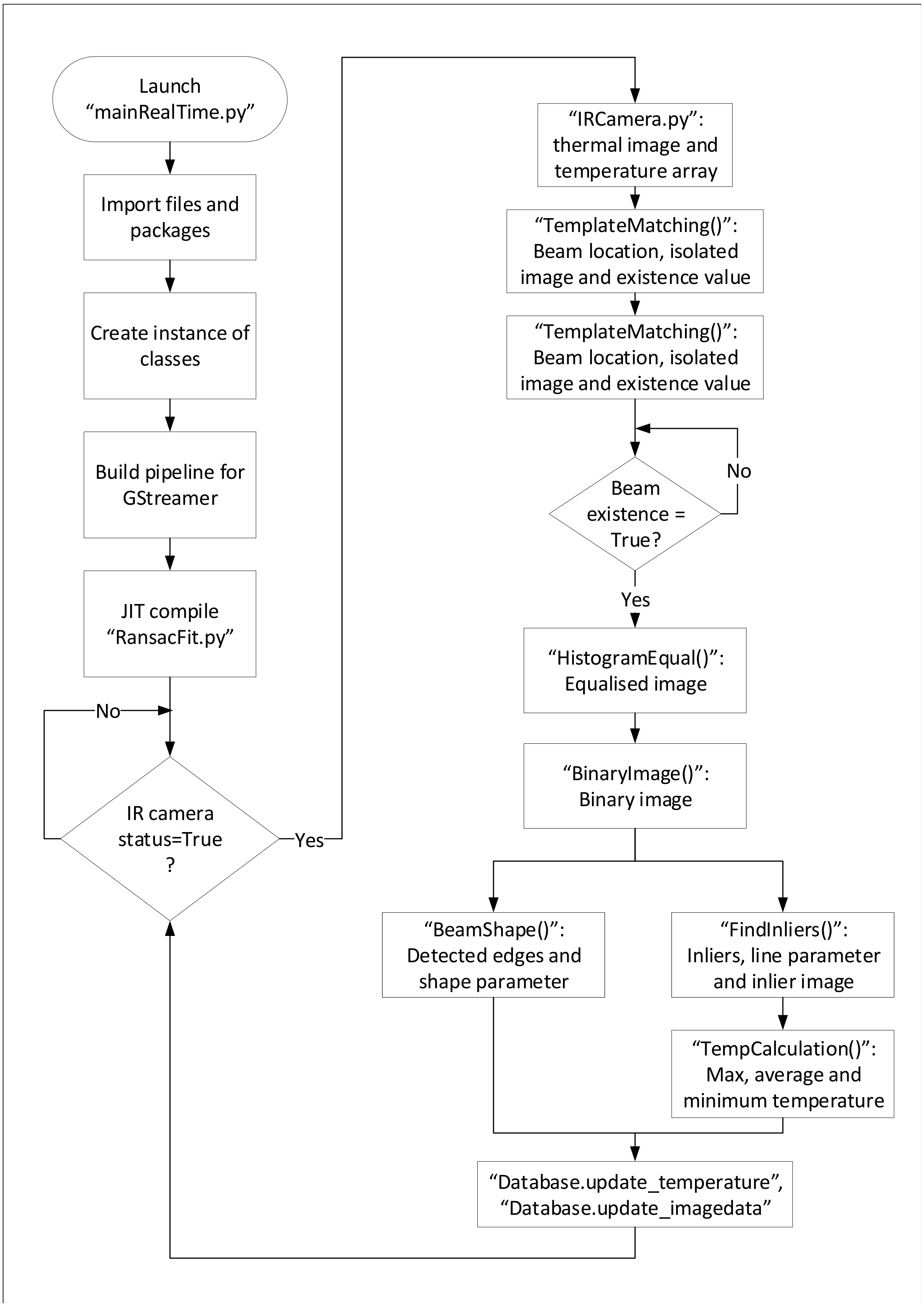

Launch "mainRealTime.py"
Import files and packages
Create instance of classes
Build pipeline for GStreamer
JIT compile "RansacFit.py"
No
IR camera status=True ?
Yes
"IRCamera.py": thermal image and temperature array
"TemplateMatching()": Beam location, isolated image and existence value
"TemplateMatching()": Beam location, isolated image and existence value
Beam existence = True?
No
Yes
"HistogramEqual()": Equalised image
"BinaryImage()": Binary image
"BeamShape()": Detected edges and shape parameter
"FindInliers()": Inliers, line parameter and inlier image
"TempCalculation()": Max, average and minimum temperature
"Database.update_temperature", "Database.update_imagedata"

Figure B.4: Camera application process flowchart

B.2.3 Synthetically developed images

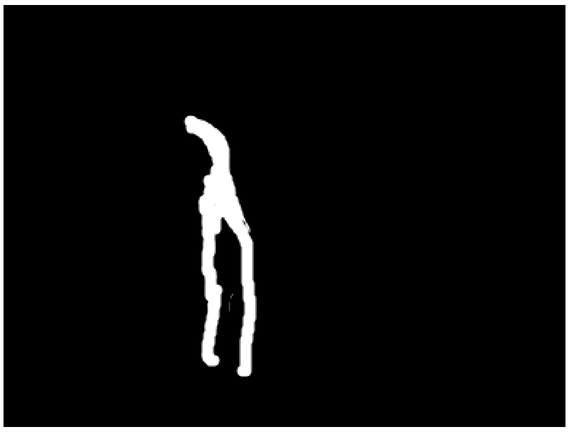


Table B.1: Synthetically developed image number 2.

Detected edges:	
Y-coordinate	Number of edges
0	0
20	3
40	2
60	2
80	4
100	4
120	4
140	7
160	4

Table B.2: Number of detected edges in image number 2.

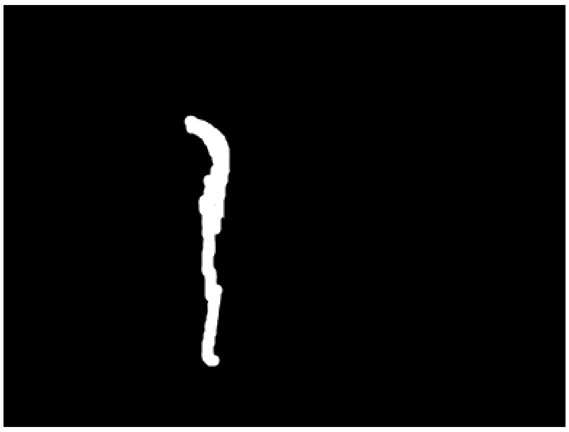


Table B.3: Synthetically developed image number 3.

Detected edges:	
Y-coordinate	Number of edges
0	0
20	3
40	2
60	2
80	2
100	2
120	2
140	3
160	2

Table B.4: Number of detected edges in image number 3.

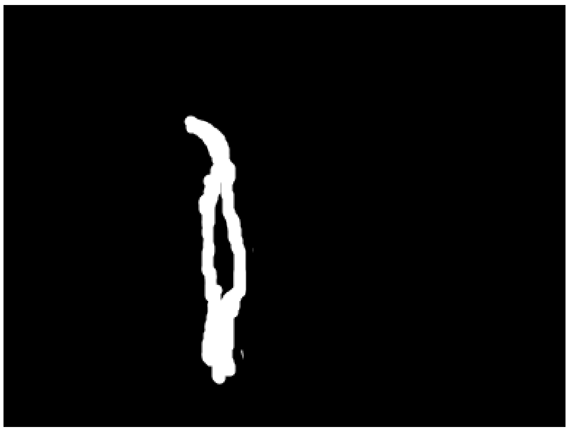


Table B.5: Synthetically developed image number 4.

Detected edges:	
Y-coordinate	Number of edges
0	0
20	3
40	2
60	4
80	4
100	6
120	4
140	2
160	2

Table B.6: Number of detected edges in image number 4.

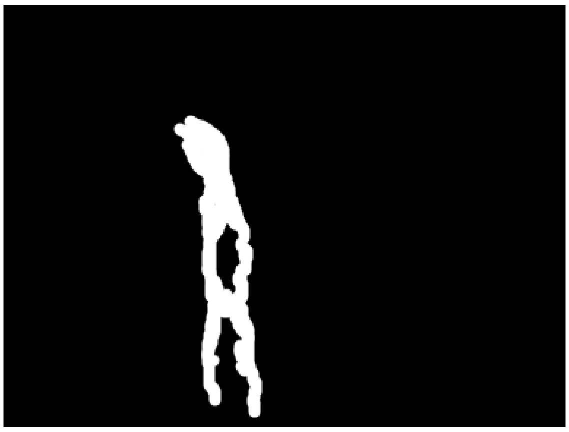


Table B.7: Synthetically developed image number 5.

Detected edges:	
Y-coordinate	Number of edges
0	0
20	3
40	2
60	2
80	4
100	4
120	4
140	2
160	4

Table B.8: Number of detected edges in image number 5.

B.3 Human Machine Interface

B.3.1 Flask web-HMI homepage

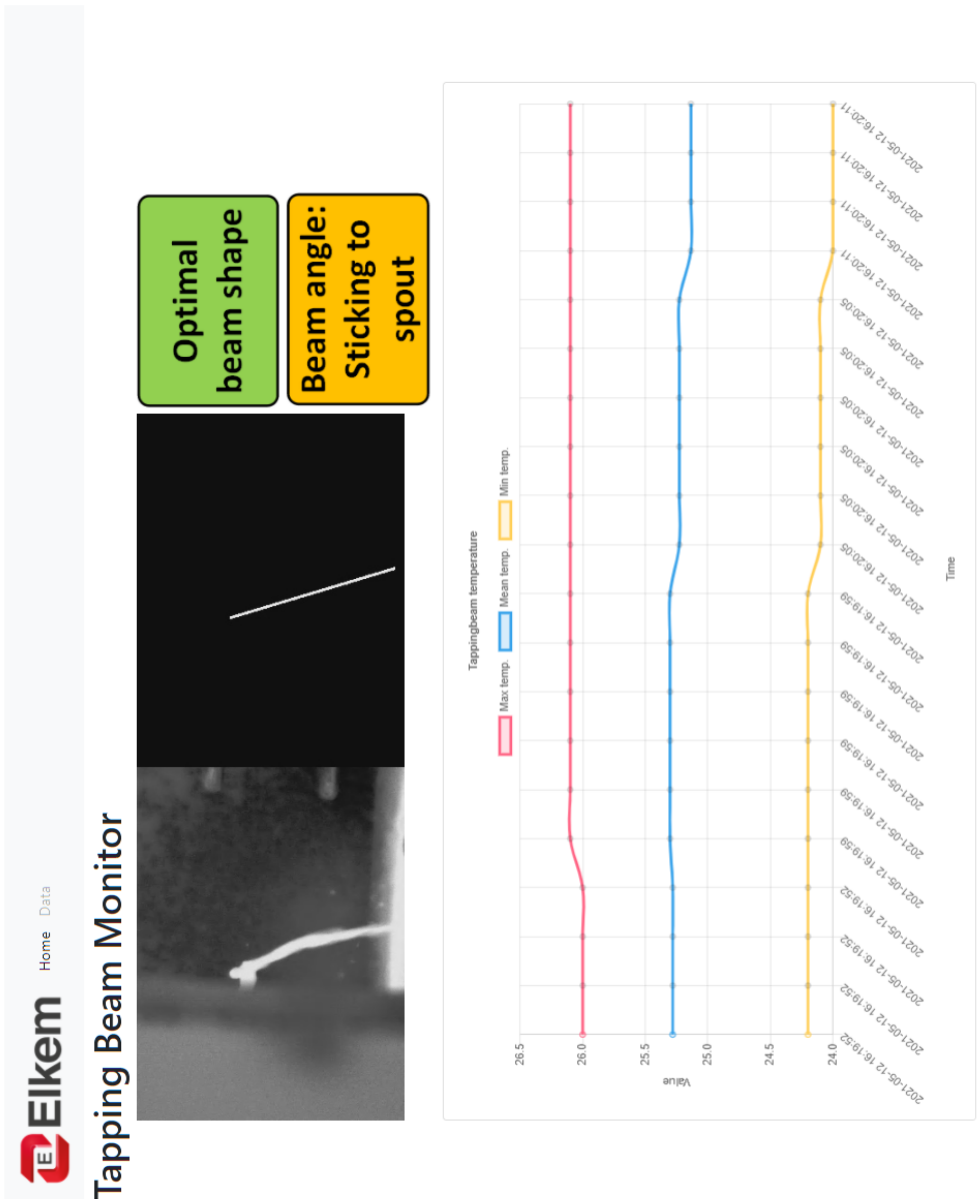
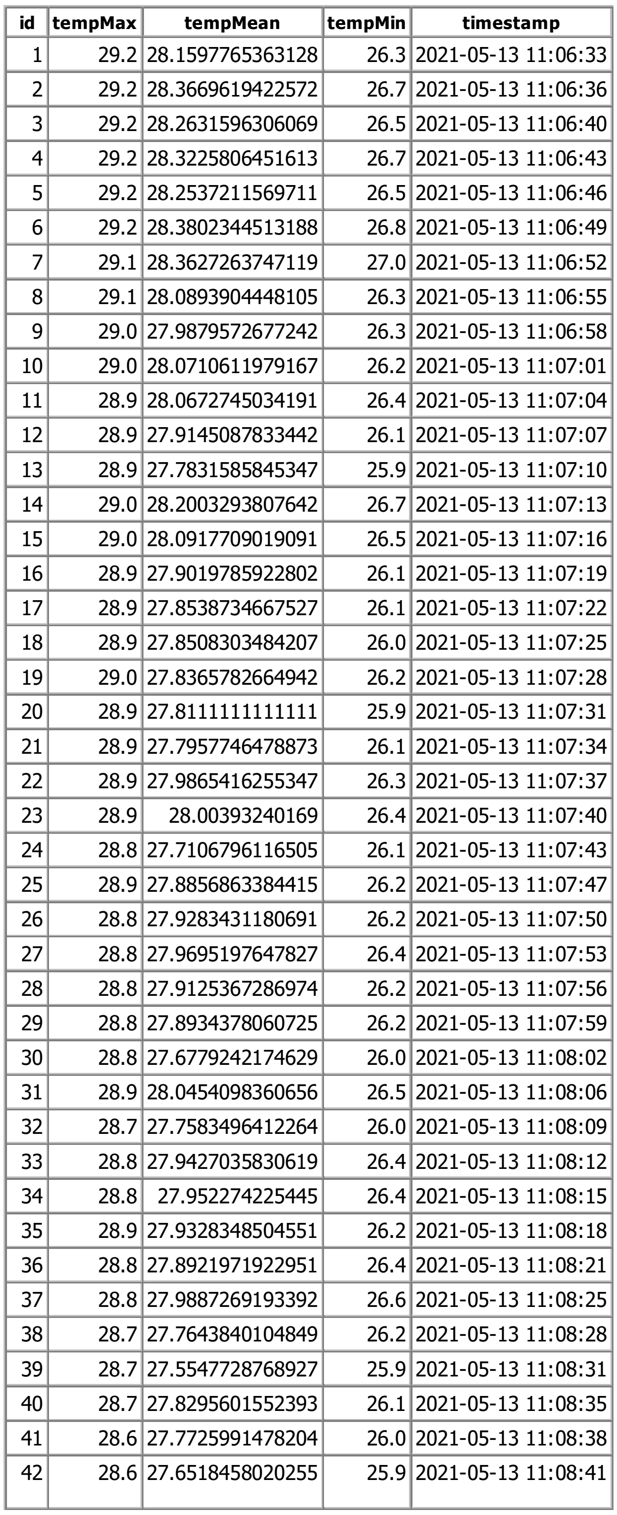


Figure B.5: HMI homepage

B.4 Database file

B.4.1 Temperature table

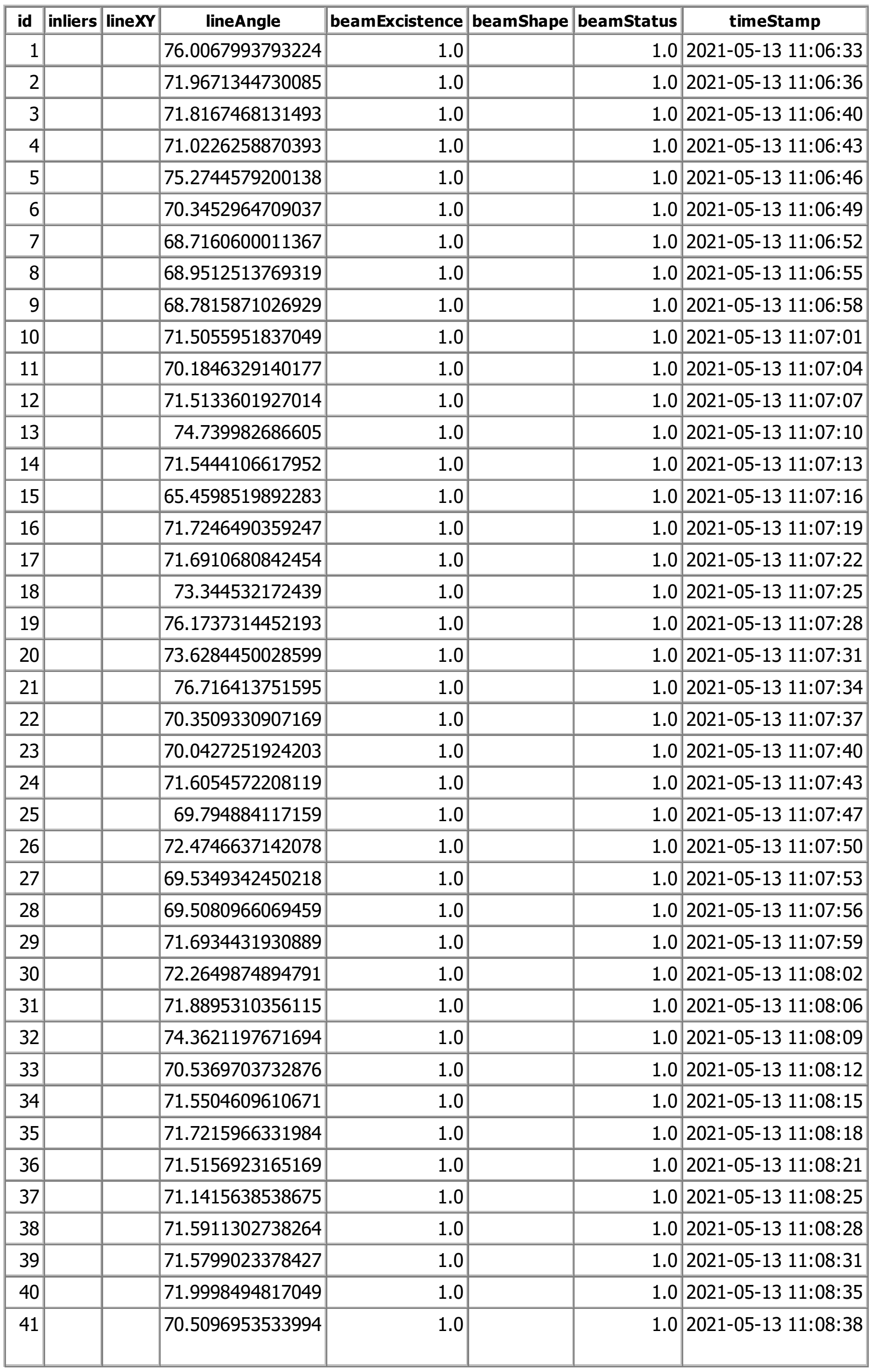


id	tempMax	tempMean	tempMin	timestamp
1	29.2	28.1597765363128	26.3	2021-05-13 11:06:33
2	29.2	28.3669619422572	26.7	2021-05-13 11:06:36
3	29.2	28.2631596306069	26.5	2021-05-13 11:06:40
4	29.2	28.3225806451613	26.7	2021-05-13 11:06:43
5	29.2	28.2537211569711	26.5	2021-05-13 11:06:46
6	29.2	28.3802344513188	26.8	2021-05-13 11:06:49
7	29.1	28.3627263747119	27.0	2021-05-13 11:06:52
8	29.1	28.0893904448105	26.3	2021-05-13 11:06:55
9	29.0	27.9879572677242	26.3	2021-05-13 11:06:58
10	29.0	28.0710611979167	26.2	2021-05-13 11:07:01
11	28.9	28.0672745034191	26.4	2021-05-13 11:07:04
12	28.9	27.9145087833442	26.1	2021-05-13 11:07:07
13	28.9	27.7831585845347	25.9	2021-05-13 11:07:10
14	29.0	28.2003293807642	26.7	2021-05-13 11:07:13
15	29.0	28.0917709019091	26.5	2021-05-13 11:07:16
16	28.9	27.9019785922802	26.1	2021-05-13 11:07:19
17	28.9	27.8538734667527	26.1	2021-05-13 11:07:22
18	28.9	27.8508303484207	26.0	2021-05-13 11:07:25
19	29.0	27.8365782664942	26.2	2021-05-13 11:07:28
20	28.9	27.8111111111111	25.9	2021-05-13 11:07:31
21	28.9	27.7957746478873	26.1	2021-05-13 11:07:34
22	28.9	27.9865416255347	26.3	2021-05-13 11:07:37
23	28.9	28.00393240169	26.4	2021-05-13 11:07:40
24	28.8	27.7106796116505	26.1	2021-05-13 11:07:43
25	28.9	27.8856863384415	26.2	2021-05-13 11:07:47
26	28.8	27.9283431180691	26.2	2021-05-13 11:07:50
27	28.8	27.9695197647827	26.4	2021-05-13 11:07:53
28	28.8	27.9125367286974	26.2	2021-05-13 11:07:56
29	28.8	27.8934378060725	26.2	2021-05-13 11:07:59
30	28.8	27.6779242174629	26.0	2021-05-13 11:08:02
31	28.9	28.0454098360656	26.5	2021-05-13 11:08:06
32	28.7	27.7583496412264	26.0	2021-05-13 11:08:09
33	28.8	27.9427035830619	26.4	2021-05-13 11:08:12
34	28.8	27.952274225445	26.4	2021-05-13 11:08:15
35	28.9	27.9328348504551	26.2	2021-05-13 11:08:18
36	28.8	27.8921971922951	26.4	2021-05-13 11:08:21
37	28.8	27.9887269193392	26.6	2021-05-13 11:08:25
38	28.7	27.7643840104849	26.2	2021-05-13 11:08:28
39	28.7	27.5547728768927	25.9	2021-05-13 11:08:31
40	28.7	27.8295601552393	26.1	2021-05-13 11:08:35
41	28.6	27.7725991478204	26.0	2021-05-13 11:08:38
42	28.6	27.6518458020255	25.9	2021-05-13 11:08:41

1

Figure B.6: Database temperature data.

B.4.2 Image data table



id	inliers	lineXY	lineAngle	beamExcistence	beamShape	beamStatus	timeStamp
1			76.0067993793224	1.0		1.0	2021-05-13 11:06:33
2			71.9671344730085	1.0		1.0	2021-05-13 11:06:36
3			71.8167468131493	1.0		1.0	2021-05-13 11:06:40
4			71.0226258870393	1.0		1.0	2021-05-13 11:06:43
5			75.2744579200138	1.0		1.0	2021-05-13 11:06:46
6			70.3452964709037	1.0		1.0	2021-05-13 11:06:49
7			68.7160600011367	1.0		1.0	2021-05-13 11:06:52
8			68.9512513769319	1.0		1.0	2021-05-13 11:06:55
9			68.7815871026929	1.0		1.0	2021-05-13 11:06:58
10			71.5055951837049	1.0		1.0	2021-05-13 11:07:01
11			70.1846329140177	1.0		1.0	2021-05-13 11:07:04
12			71.5133601927014	1.0		1.0	2021-05-13 11:07:07
13			74.739982686605	1.0		1.0	2021-05-13 11:07:10
14			71.5444106617952	1.0		1.0	2021-05-13 11:07:13
15			65.4598519892283	1.0		1.0	2021-05-13 11:07:16
16			71.7246490359247	1.0		1.0	2021-05-13 11:07:19
17			71.6910680842454	1.0		1.0	2021-05-13 11:07:22
18			73.344532172439	1.0		1.0	2021-05-13 11:07:25
19			76.1737314452193	1.0		1.0	2021-05-13 11:07:28
20			73.6284450028599	1.0		1.0	2021-05-13 11:07:31
21			76.716413751595	1.0		1.0	2021-05-13 11:07:34
22			70.3509330907169	1.0		1.0	2021-05-13 11:07:37
23			70.0427251924203	1.0		1.0	2021-05-13 11:07:40
24			71.6054572208119	1.0		1.0	2021-05-13 11:07:43
25			69.794884117159	1.0		1.0	2021-05-13 11:07:47
26			72.4746637142078	1.0		1.0	2021-05-13 11:07:50
27			69.5349342450218	1.0		1.0	2021-05-13 11:07:53
28			69.5080966069459	1.0		1.0	2021-05-13 11:07:56
29			71.6934431930889	1.0		1.0	2021-05-13 11:07:59
30			72.2649874894791	1.0		1.0	2021-05-13 11:08:02
31			71.8895310356115	1.0		1.0	2021-05-13 11:08:06
32			74.3621197671694	1.0		1.0	2021-05-13 11:08:09
33			70.5369703732876	1.0		1.0	2021-05-13 11:08:12
34			71.5504609610671	1.0		1.0	2021-05-13 11:08:15
35			71.7215966331984	1.0		1.0	2021-05-13 11:08:18
36			71.5156923165169	1.0		1.0	2021-05-13 11:08:21
37			71.1415638538675	1.0		1.0	2021-05-13 11:08:25
38			71.5911302738264	1.0		1.0	2021-05-13 11:08:28
39			71.5799023378427	1.0		1.0	2021-05-13 11:08:31
40			71.9998494817049	1.0		1.0	2021-05-13 11:08:35
41			70.5096953533994	1.0		1.0	2021-05-13 11:08:38

1

Figure B.7: Database image data.

Appendix C

Code

C.1 Computer vision algorithms

C.1.1 mainRealTime.py

```python
from src import IRCamera, BeamDetector, FeatureTracking, Database,
    RansacFit
from datetime import datetime
from time import sleep, time
import numpy as np
import cv2
import os

path = os.path.dirname(os.path.abspath(__file__))

# Template root folder
template_root = 'template/frame2904.png'

# Initialize variables
thresh_gain = 500  # binary image threshold gain, tune this value
    between 0 and 255 for better binary representation of tapping beam
heigth_top = 0 # top heigth of the beam shape edge detection algorithm
heigth_bottom = 0 # bottom heigth of the beam shape edge detection
    algorithm
thresh_inlier = 3 # inlier distance threshold for the RANSAC algorithm
iterations = 1000 # number of iterations in the RANSAC algorithm
samples = 4 # number of samples per iteration to fit the RANSAC model
inliers = 50 # minimum number of inliers for the RANSAC model


# Create instance of classes
beamDetector = BeamDetector.BeamDetector(thresh_gain, heigth_top,
    heigth_bottom)

featureTracking = FeatureTracking.FeatureTracking(template_root)
```

```python

ransacFit = RansacFit.RansacFit(thresh_inlier, iterations, samples,
    inliers)

irCamera = IRCamera.IRCamera(path)

path_db = os.path.join(path,'webapp/database/data.db' )
database = Database.Database(path_db)


# Frame dimensions
framerate = 10
frame_width = 384
frame_heigth = 288

# Create videowriter as a SHM source, two images side-by-side
out = cv2.VideoWriter("appsrc ! video/x-raw, format=BGR ! queue !
    videoconvert ! video/x-raw, format=BGRx ! nvvidconv ! omxh264enc !
    video/x-h264, stream-format=byte-stream ! h264parse ! rtph264pay pt
    =96 config-interval=1 ! udpsink host=0.0.0.0 port=5000",  0,
    framerate, (frame_width*2, frame_heigth))

# Initiate loop parameters
i = 0
k = 0
beam_existence_old = 1
beam_existence_old_old = 1

if __name__ == '__main__':

    # Compile RansacFit.py with JIT
    ransacFit.InitJIT((frame_heigth, frame_width))

    # Check camera status
    status = irCamera.status()

    while status is True:

        # Add sleep time to the program - used to lower the CPU usage
        sleep(0.2 - time() % 0.2)

        # Get thermal image and array from IR camera
        image_thermal, array_temp = irCamera.getThermal()


        # Find the tapping beam region and check for existence of the
            tappingbeam
        loc, image_tapping, beam_existence = featureTracking.TemplateMatching(image_thermal)

        # Calculate the average of the beam existence parameter, used to filter out transient values
        beam_average_existence = np.mean([beam_existence, beam_existence_old, beam_existence_old_old])

        if beam_average_existence == 1:

            # Equalize the histogram of the image
            image_eq = beamDetector.HistogramEqual(image_tapping)

            # Create a binary image
            image_binary = beamDetector.BinaryImage(image_eq)

            # Calculate the beam-shape paraemters
            beam_data, beam_status = beamDetector.BeamShape(loc, image_binary)


            # Use RansacFit to find the inliers
            inliers,line_xy, image_inliers, theta_line = ransacFit.FindInliers(image_binary)


            # Detect mean, max and min temperature in the tappingbeam
            temp_max, temp_mean, temp_min = beamDetector.TempCalculation(array_temp, image_inliers)

            # Convert to BGR format
            image_inliers = cv2.cvtColor(np.array(image_inliers), cv2.COLOR_GRAY2BGR)

            # Horizontaly stack the images before sending them to the GStreamer pipeline
            image_GST = np.hstack((image_thermal, image_inliers))

        else:

            # Thermal image side-by-side, if the template matching fails
            image_GST = np.hstack((image_thermal,image_thermal))

            # Initialize the remaining values
            temp_max, temp_mean, temp_min = 0,0,0
            inliers, line_xy, theta_line, beam_data, beam_status = 0, 0, 0, 0, 0

        # Write to SHM
        out.write(image_GST)

        # Save data to database, for every 5th iteration
        if i >= 5:
            # Time
            timestamp = datetime.now().replace(microsecond=0)

            # Update temperature data
            database.update_temperature((temp_max, temp_mean, temp_min, timestamp))

            # Update image date
            database.update_imagedata((inliers, line_xy, theta_line, beam_existence, beam_data, beam_status, timestamp))

            i = 0

        i = i + 1

        # Update existence parameter
        beam_existence_old = beam_existence
        beam_existence_old_old = beam_existence_old
```

C.1.2 IRCamera.py

```python
#! /usr/bin/env python3
from ctypes.util import find_library
import numpy as np
import ctypes as ct
import cv2
import os
import sys
import time

class IRCamera(object):
    """
    IRCamera object initializes the thermal camera and
    returns the thermal image and the temperature array
    """
    def __init__(self,source):
        """
        Initialize camera variables
        """
        # create log dir if non existent
        if not os.path.exists('./src/log'):
            os.mkdir('./src/log')

        if not os.path.exists('./src/images'):
            os.mkdir('./src/images')


        # load library
        if os.name == 'nt':
            #windows:
            path_dll = os.path.join(source, "bin\\x64\\libirimager.dll"
                )
            self.libir = ct.CDLL(path_dll)

            #init vars for Windows
            self.pathXml = ct.c_char_p(b'./config/20022030.xml')
            self.pathFormat = ct.c_char_p(b'./config/')
            self.pathLog = ct.c_char_p(b'./log/logfilename')
            print('===== Windows OS =====')

        else:
            print('DLL path:',source)
            #linux:
            self.libir = ct.cdll.LoadLibrary(ct.util.find_library("
                irdirectsdk"))

            #init vars for Linux
            path_xml = os.path.join(source, "config/20022030.xml" )
            self.pathXml = ct.c_char_p(path_xml.encode())
            self.pathFormat = ct.c_char_p()
            path_log = os.path.join(source, "log/logfilename" )
            self.pathLog = ct.c_char_p(path_log.encode())
            print('===== Linux OS =====')


        # init vars for thermal camera
        self.palette_width = ct.c_int()
        self.palette_height = ct.c_int()
        self.thermal_width = ct.c_int()
        self.thermal_height = ct.c_int()
        self.serial = ct.c_ulong()

        # instance of EvoIRFrameMetadata class
        self.metadata = self.EvoIRFrameMetadata()

        # init lib
        self.retLib = self.libir.evo_irimager_usb_init(self.pathXml, self.pathFormat, self.pathLog)
        print('===== Init IR imager library =====')
        if self.retLib != 0:
            print("error at init")
            exit(self.retLib)

        # get the serial number
        ret = self.libir.evo_irimager_get_serial(ct.byref(self.serial))
        print('serial: ' + str(self.serial.value))

        # get thermal image size
        self.libir.evo_irimager_get_thermal_image_size(ct.byref(self.thermal_width), ct.byref(self.thermal_height))
        print('thermal width: ' + str(self.thermal_width.value))
        print('thermal height: ' + str(self.thermal_height.value))

        # init thermal data container
        self.np_thermal = np.zeros([self.thermal_width.value * self.thermal_height.value], dtype=np.uint16)
        self.npThermalPointer = self.np_thermal.ctypes.data_as(ct.POINTER(ct.c_ushort))

        # get palette image size, width is different to thermal image width duo to stride alignment!!!
        self.libir.evo_irimager_get_palette_image_size(ct.byref(self.palette_width), ct.byref(self.palette_height))
        print('palette width: ' + str(self.palette_width.value))
        print('palette height: ' + str(self.palette_height.value))

        # init image container
        self.np_img = np.zeros([self.palette_width.value * self.
            palette_height.value * 3], dtype=np.uint8)
        self.npImagePointer = self.np_img.ctypes.data_as(ct.POINTER(ct.
            c_ubyte))


        """
        Initialize calibration variables
        """
        # Initilize camera parameters
        self.f = 0.013  # focal length of the 029 lens on the Optris
            PI400i
        self.Sx = 384   # camera resolution in the x-axis (horisontal)
        self.Sy = 288   # camera resolution in the y-axis (vertical)

        # Camera intrinsic calibration matrix
        self.mtx =  np.array([[  -self.f/self.Sx,   0,
            self.Sx/2],
                              [   0,                     -self.f/self.Sy,
                                  self.Sy/2],
                              [   0,                     0,
                                  1]])
        # Camera distortion calibration matrix
        self.dist = 0

        # Splits up calibration into making a map, and doing the
            remapping in calibrate
        # member function
        # This is faster than doing cv2.undistort()
        self.map1, self.map2 = cv2.initUndistortRectifyMap(
            self.mtx,
            self.dist,
            np.eye(3),
            self.mtx,
            (self.Sx, self.Sy),
            cv2.CV_32FC1
        )

    def status(self):
        """
        Returns a boolean value for the initialization status of the
            camera
        """

```

```python
        if self.retLib == 0:
            status = True
            print('===== IR camera ready ======')

        else:
            status = False
            print('Error at init. Camera status:',status)

        return status




    def getThermal(self):
        """
        Function for returning the thermal image and temperature array
        """

        ret = self.libir.evo_irimager_get_thermal_palette_image_metadata(self.thermal_width, self.thermal_height, self.npThermalPointer, self.palette_width, self.palette_height, self.npImagePointer, ct.byref(self.metadata))

        if ret != 0:
            print('error on evo_irimager_get_thermal_palette_image ' + str(ret))


        # get the palette image
        thermal_image = self.np_img.reshape(self.palette_height.value, self.palette_width.value, 3)[:,:,::-1]

        # get the thermal array
        thermal_vector = self.np_thermal/10. - 100.

        thermal_array = thermal_vector.reshape(self.thermal_height.value, self.thermal_width.value)


        return thermal_image, thermal_array

    def calibrate(self,image):
        """
        Perform calibration of input OpenCV image
        """
        # Make copy of input image

        # # Perform color calibration
        # image_float = np.array(image, float) * self.gain_matrix

        # # Clip to keep values between 0 and 255
        # # Using [:] sets the value of object "image" without making a new object
        # # i.e. just replaces in-object which does not need a return statement
        # calibrated[:] = np.clip(image_float, 0, 255)

        # Apply camera distortion calibration
        calibrated = image.copy()
        calibrated[:] = cv2.remap(calibrated, self.map1, self.map2, cv2.INTER_LINEAR)
        return calibrated


    class EvoIRFrameMetadata(ct.Structure):
        """
        Structure that contains additional frame information from the camera
        """

        # Define EvoIRFrameMetadata structure for additional frame info
        _fields_ = [
            ("counter", ct.c_uint),
            ("counterHW", ct.c_uint),
            ("timestamp", ct.c_longlong),
            ("timestampMedia", ct.c_longlong),
            ("flagState", ct.c_int),
            ("tempChip", ct.c_float),
            ("tempFlag", ct.c_float),
            ("tempBox", ct.c_float),
        ]




if __name__ == "__main__":

    path = os.path.dirname(os.path.abspath(__file__))

    # Crop area, tapping test
    class Template():
        x = 145
        y = 40
        w = 80
        h = 230

    xyTemplate = Template()

    # instance of IRCamera class
    irCamera = IRCamera(path)

    # capture and display image till q is pressed
    t, i = 0, 0
    while chr(cv2.waitKey(1) & 255) != 'q':


        # get thermal image and temperature array
        image_thermal, array_thermal = irCamera.getThermal()

        # Index thermal image array to create template image
        image_template = cv2.cvtColor(image_thermal, cv2.COLOR_BGR2GRAY)[xyTemplate.y:(xyTemplate.y + xyTemplate.h), xyTemplate.x:(xyTemplate.x + xyTemplate.w)]


        # Save images
        cv2.imwrite('../src/images/frame{}.png'.format(i), image_template)
        print('Captured!')
        i +=1
```

C.1.3 FeatureTracking.py

```

import os
import numpy as np
import matplotlib.pyplot as plt
import cv2
import time


class FeatureTracking(object):
    """
    Class for tracking the tapping beam, uses template matching finding
       the tapping beam region.
    """

    def __init__(self,templatePath):
        """
        Initialize variables
        """
        self.templatePath = templatePath


    def TemplateMatching(self,image):
        """
        Template matching for finding the specific image region where
           the tapping beam is located.
        Requires a template of the tapping beam, use Template.py to
           create the desired template image.
        """
        # Read template
        template_image = cv2.imread(self.templatePath)

        # Convert template image to grayscale
        template_image = cv2.cvtColor(template_image,cv2.COLOR_BGR2GRAY)

        # Convert CV image to grayscale
        image_gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

        # Get template height and width
        template_w, template_h = template_image.shape[::-1]

        # Get image height and width
        image_w, image_h = image_gray.shape[::-1]

        # Convert CV image to grayscale
        image_gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

        # Perform match operations
        res = cv2.matchTemplate(image_gray,template_image,cv2.TM_CCOEFF_NORMED)
        # Specify a threshold for the matching
        threshold = 0.8

        # Store the coordinates of matched area in a numpy array
        loc = np.where( res >= threshold)[::-1]

        # Base of the segmented image
        image_segmented = np.zeros(image_gray.shape,np.uint8)

        if bool(loc[0].any()) is False:
            loc = (0, 0, image_w, image_h)
            beam_existence = 0

            # Isolate the tapping ROI
            image_segmented = image_segmented


        else:
            loc = (int(round(np.mean(loc[0]),0)),int(round(np.mean(loc[1]),0)),template_w, template_h)
            beam_existence = 1
            # Isolate the tapping ROI
            image_segmented[loc[1]:(loc[1] + loc[3]), loc[0]:(loc[0] + loc[2])] = image_gray[loc[1]:(loc[1] + loc[3]), loc[0]:(loc[0] + loc[2])]


        return loc, image_segmented, beam_existence


    def BoundingBox(self,image,loc,theta):
        """
        Draw a bounding box over the tappingbeam grayscale image.
        The coordinates for the bounding box is numbered ccw,
        starting with the top left corner.
        """

        # Define top left corner, width and heigth
        pt0, width, heigth = (loc[0],loc[1]), loc[2], loc[3]

        # Calculate the absolute value of the tapping beam angle
        absDelta = np.ceil(np.abs(heigth/np.tan(theta[0])))

        # Correct the offset
        if theta[1] < 90:
            delta = absDelta
        else:
            delta = - absDelta

        # Define corners
        pt1 = [pt0[0], pt0[1]]
        pt2 = [pt0[0] + delta, pt0[1] + heigth]
        pt3 = [pt0[0] + width + delta, pt0[1] + heigth]
        pt4 = [pt0[0] + width, pt0[1]]

        # Create list of corners
        corners = [np.array([ pt1, pt2, pt3, pt4 ], dtype = int)]

        # Draw the contour
        image_contour = cv2.drawContours(np.array(image),corners
            ,-1,(255,255,255),2)

        return image_contour
```

C.1.4 BeamDetector.py

```
import numpy as np



class BeamDetector(object):

    def __init__(self,thresh_gain = 100,heigth_top = 0, heigth_bottom =
        0):
        """
        Initialize the variables.
        """
        # BinaryImage
        self.thresh_gain = thresh_gain

        # BeamShape
        self.heigth_top = heigth_top # starting position for the edge
            detection
        self.heigth_bottom = heigth_bottom # djust stop position for the
             edge detetion

    def BinaryImage(self,image):
        """
        Binary image thresholding based upon temperature/image
            brightness.
        This algorithm is based upon the idea that the tapping beam
            always
        exists in the brightest image pixels, within some margin.
        """
        # Base of the binary image
        image_binary = np.zeros(image.shape,np.uint8)
        image_white = np.uint8(255)

        # Find the pixels with maximum brightness
        i_mean = np.mean(image)
        i_max = np.amax(image)
        i_thresh = np.ceil(i_max - i_mean/i_max*self.thresh_gain)
        indices = np.where(image >= i_thresh)

        # print('Threshold:',i_thresh,'Max:',i_max,'Mean:',i_mean)

        # Converts the coordinates
        coordinates_x = np.transpose(indices[0])
        coordinates_y = np.transpose(indices[1])

        image_binary[coordinates_x,coordinates_y] = image_white

        return image_binary



    def HistogramEqual(self,image):
        """
        Function for normalizing the histogram
        https://www.imageeprocessing.com/2011/04/matlab-code-histogram-
           equalization.html
        """

        # Create histogram vector
        histogram = np.zeros(256)
        buffer, _ = np.histogram(image,
                                 bins=np.arange(256),
                                 range=(0, 1))

        histogram[0:buffer.size]=buffer

        # Cumulative distrobution of the histogram
        cdf = histogram.cumsum()

        # Create masked array, excluding indices where the value is zero
        cdf_m = np.ma.masked_equal(cdf,0)

        # Scale the histogram to values between 0 and 255
        cdf_m = (cdf_m - cdf_m.min())*255/(cdf_m.max()-cdf_m.min())

        # Fill inn the blank indexes with the value = 0
        cdf = np.ma.filled(cdf_m,0).astype('uint8')

        # Create the new equalised image
        image_eq = cdf[image]

        return image_eq

    def BeamShape(self, loc, image):
        """
        Slices the tapping beam into horizontal vectors with even
           spacing,
        detects the edges by convolving a filter with the image vector.
        """
        # Define top left corner, width and heigth
        pt0, width, heigth = (loc[0],loc[1]), loc[2], loc[3]


        # Isolate the pixels of the tappingbeam, found with template.
        image_isolated = np.zeros((heigth,width))
        image_isolated = image[pt0[1]:(pt0[1] + heigth), pt0[0]:(pt0[0]
            + width)]

        # For local testing
        # image_isolated[:,10:20] = 0
        # image_isolated[:,30:40] = 0

        # Increment vector for iterating through the image in the y-
            direction
        y_increment = np.arange(self.heigth_top, (heigth + self.
            heigth_bottom), 10)

        # Empty array for storing the y-position and the detected number
             of edges
        beam_data = np.zeros((len(y_increment),2))

        # Initialize counter variable
        k = 0

        # Iterating through the image in the y-direction, row-by-row.
        for i in y_increment:

            # Convolve the extracted image data with a Gaussian
                derivative filter
            row = image_isolated[i,:]
            row_convolve = np.convolve([1,0,-1], row)

            # Find peaks in dataset, inspired by: https://tcoil.info/
                find-peaks-and-valleys-in-dataset-with-python/
            row_convolve_peak = (np.diff(np.sign(np.diff(np.abs(
                row_convolve)))) < 0).nonzero()[0] + 1

            # Count number of edges in row
            edges = len(row_convolve_peak)

            # Save data
            beam_data[k,:] = np.array([i,edges])

            # Update counter
            k += 1

        # Remove noise, only accept data between 0 and 8
        indices = np.where( beam_data[:,1]>8 )
        coordinates_x = np.transpose(indices[0])

        # Set the invalid values to zero
        beam_data[coordinates_x, 1] = 0

        # Calculate the mean of the tappingbeam shape-values
        beam_value_mean = np.mean(beam_data[:,1])

        if 3 >= beam_value_mean > 1:

            beam_status = 1 # Optimal shape

        elif beam_value_mean > 3:

            beam_status = 0 # Sparse shape

        else:

            beam_status =  2 # Fault, check beam_data matrix or adjust the y-increment vector in BeamDetector.BeamShape()

        return beam_data, beam_status



    def TempCalculation(self, array_thermal, image_binary):
        """
        Calculates the mean, max and min temperature of the tapping beam.
        """

        # Find the pixel coordinates for the tapping beam
        indices = np.where(image_binary == 255)

        # Converts the coordinates
        coordinates_x = np.transpose(indices[0])
        coordinates_y = np.transpose(indices[1])

        coordinates_y = np.delete(coordinates_y, np.where(coordinates_y >= 382))

        # Empty array
        temp_region = np.zeros(image_binary.shape, np.uint8)

        # Isolate the tapping beam temperature
        temp_region = array_thermal[coordinates_x, coordinates_y]

        # Calcualte mean, max and min temperature
        temp_mean = np.mean(temp_region)
        temp_max = np.max(temp_region)
        temp_min = np.min(temp_region)

        return temp_max, temp_mean, temp_min
```

C.1.5 RansacFit.py

```
import numpy as np
import numba as nb
import cv2
from numba import jit
from numba.experimental import jitclass


spec = [('thresh',nb.float32),
        ('iterations',nb.uint32),
        ('samples',nb.uint32),
        ('inliers',nb.uint32)]
@jitclass(spec)
class RansacFit(object):

    def __init__(self, thresh = 3, iterations = 1000, samples = 4,
        inliers = 50):

        """
        Initialize the variables for RansacFit(thresh),
        where thresh is the inlier threshold for the RANSAC algorithm.
        """
        self.thresh = np.float32(thresh)
        self.iterations = np.uint32(iterations)
        self.samples = np.uint32(samples)
        self.inliers = np.uint32(inliers)

    def InitJIT(self, data_size):

        """
        Initializes the JIT - compiler.
        Run this function directly after the instansiation of the
            RansacFit class.
        Note that the data input to this function should be of the same
            size and type
        as the data that is to be used in the FindInliers function
        """
        # Create sample data
        data = np.ones(data_size, dtype = np.uint32)

        data[30:150,150:170] = np.uint32(255)
        data[100:150,100:150] = np.uint32(255)

        # Rund the Ransac function with sample data
        initInliers, _, _, _ = self.FindInliers(data)

        # If sucsessfull, print to terminal
        if initInliers is not None:
            print('===== JIT compiled RansacFit.py =====')


    def FindInliers(self, data):

        """
        Finds the inliers in a dataset based upon a y=ax+b line model.
        Takes in a binary image of the segmented tappingbeam, returns
            the
        position for.
        """

        # Find the matrix positions where data == 255
        indices = np.where(data == 255)

        # Converts the coordinates
        data_x, data_y = np.transpose(indices[1]), np.transpose(indices
            [0])

        # Create data matrix - flipped axis for using image coordinate
            system
        data_xy = np.column_stack((data_x,data_y))

        # Robustly find inlier data with self written RANSAC algorithm
        bestModel, bestInliers= self.fit_ransac(data_xy, max_iters =
            self.iterations, samples_to_fit = self.samples,
            inlier_threshold = self.thresh, min_inliers = self.inliers)

        # Generate coordinates of estimated line model
        line_x_min = np.uint32(np.min(bestInliers[:,0])) # lowest index
            value
        line_x_max = np.uint32(np.max(bestInliers[:,0])) # highest index
             value

        line_x = np.arange(start = line_x_min, stop = line_x_max, dtype
            = np.float64) # arange integers from lowest to highest
        length_x = len(line_x) # length of line
        line_y = np.zeros(length_x, dtype = np.float64) # empty zeros
            vector for y-value


        bestModel_a = bestModel[0,0]
        bestModel_b = bestModel[1,0]*np.ones(length_x, dtype = np.
            float64)
        line_y = bestModel_a*line_x + bestModel_b

        # Calculate the absolute angle for the line model
            # startPoint is supposed to be the point on the RANSAC'ed
                line
            # that is closest to the tappinspout (top right corner).
            # endPoint is supposed to be the point on the RANSAC'ed line
            # at the bottom of the image.

        if line_y[0] == np.max(line_y):
            startPoint = (line_x[-1], line_y[-1])
            endPoint = (line_x[0], line_y[0])

        elif line_y[0] == np.min(line_y):
            startPoint = (line_x[0], line_y[0])
            endPoint = (line_x[-1], line_y[-1])

        # Find the delta-distance between startPoint and endPoint in x-
            and y-direction
        deltaX = (endPoint[0] - startPoint[0])
        deltaY = (endPoint[1] - startPoint[1])

        theta_deg = np.arctan2(deltaY,deltaX)*180/np.pi


        # Base of the binary image
        image_binary = np.zeros(data.shape, dtype = np.uint8)
        image_white = np.uint8(255)

        # Create binary image of the inliers
        index_length = len(bestInliers[:,0])
        index_x = np.zeros((index_length,1), dtype = np.uint32)
        index_y = np.zeros((index_length,1), dtype = np.uint32)
        index_x[:,0] = bestInliers[:,0]
        index_y[:,0] = bestInliers[:,1]

        # Iterate through the image matrix
        for ii in range(index_length):
            image_binary[index_y[ii,0], index_x[ii,0]] = image_white

        image_binary = np.copy(image_binary)

        return bestInliers, bestModel, image_binary, theta_deg



    def fit_lsq(self, X, y):

        """
        Source: https://medium.com/@iamhatesz/random-sample-consensus-bd2bb7b1be75
        Fits model for a given data using least squares.
        X should be an mxn matrix, where m is number of samples, and n is number of independent variables.
        y should be an mx1 vector of dependent variables.
        """

        b = np.ones((X.shape[0], 1), np.uint32)
        A = np.hstack((X, b)).astype(np.float64)

        theta = np.linalg.lstsq(A, y.astype(np.float64), rcond=-1)[0]

        return theta



    def evaluate_model(self, X, y, theta, inlier_threshold):
        """
        Source: https://medium.com/@iamhatesz/random-sample-consensus-bd2bb7b1be75

        Evaluates model and returns total number of inliers.
        X should be an mxn matrix, where m is number of samples, and n is number of independent variables.
        y should be an mx1 vector of dependent variables.
        theta should be an (n+1)x1 vector of model parameters.
        inlier_threshold should be a scalar.
        """


        X = X.copy()
        b = np.ones((X.shape[0], 1), np.uint32)
        y = y.copy()
        A = np.hstack((y, X, b))
        theta = np.append(-1., theta)

        distances = np.abs(np.sum(A*theta, axis=1)) / np.sqrt(np.sum(np.power(theta[:-1], 2)))
        inliers = (distances <= inlier_threshold).astype(np.bool_)
        num_inliers = np.count_nonzero(inliers)

        return num_inliers, inliers



    def fit_ransac(self, data, max_iters, samples_to_fit,
        inlier_threshold, min_inliers):
        """
        Source: https://medium.com/@iamhatesz/random-sample-consensus-bd2bb7b1be75
        """
        # Create X and y vector from the dataset
        data_len = len(data[:,0])
        X = np.zeros((data_len,1), dtype = np.uint32)
        y = np.zeros((data_len,1), dtype = np.uint32)
        X[:,0] = data[:,0]
        y[:,0] = data[:,1]

        # Declare variables for RANSAC algorithm
        best_model = np.zeros((2,1),dtype = np.float64)
        best_inliers = np.zeros((X.shape[0], 1), dtype = np.bool_)
        best_model_performance = np.uint32(0)
        num_samples = X.shape[0]



        # Iterate through the data set with random points
        for i in range(max_iters):
            sample = np.random.choice(num_samples, size=samples_to_fit, replace=False)
            model_params = self.fit_lsq(X[sample], y[sample])
            model_performance, model_inliers = self.evaluate_model(X, y, model_params, inlier_threshold)

            if model_performance < min_inliers:
                continue

            if model_performance > best_model_performance:
                best_model = model_params
                best_model_performance = model_performance
                best_inliers[:,0] = model_inliers


        # Find the index for each inlier, achieve proper array dimensions
        best_inliers_index = np.zeros((best_model_performance, 1), dtype = np.uint32) # create zeros vector
        best_inliers_index[:,0] = np.where(best_inliers)[0].astype(np.uint32) # transfer the inlier indexes into the new vector

        inliers_x = np.zeros((best_model_performance,1), dtype = np.uint32) # create zeros vector
        inliers_y = np.zeros((best_model_performance,1), dtype = np.
        uint32) # create zeros vector

        inliers_x[:] = X[best_inliers_index.reshape(
            best_model_performance)] # transer image position of the
            inliers in x-direction
        inliers_y[:] = y[best_inliers_index.reshape(
            best_model_performance)] # transer image position of the
            inliers in y-direction

        inliers_xy = np.hstack((inliers_x, inliers_y)) # combine the
            inliers into a nx2 array


        return best_model, inliers_xy
```

C.1.6 Database.py

```
from sqlite3 import Error
from datetime import datetime
import numpy as np
import sqlite3
import io
import os

class Database(object):


    def __init__(self,dbPath):
        """
        Initalize variables
        """
        self.dbPath = dbPath

        # Register a new datatype
        def adapt_array(arr):
            """
            http://stackoverflow.com/a/31312102/190597 (SoulNibbler)
            """
            out = io.BytesIO()
            np.save(out, arr)
            out.seek(0)
            return sqlite3.Binary(out.read())

        def convert_array(text):
            out = io.BytesIO(text)
            out.seek(0)
            return np.load(out)

        # Converts np.array to TEXT when inserting
        sqlite3.register_adapter(np.ndarray, adapt_array)

        # Converts TEXT to np.array when selecting
        sqlite3.register_converter("array", convert_array)

        # Temperature data table
       temp_table = """ CREATE TABLE IF NOT EXISTS temperature (
        id              integer PRIMARY KEY,
        tempMax         float                   NOT NULL,
        tempMean        float                   NOT NULL,
        tempMin         float                   NOT NULL,
        timestamp       text                    NOT NULL); """



        # Image data table
        image_table = """ CREATE TABLE IF NOT EXISTS imagedata (
        id              integer PRIMARY KEY,
        inliers         arr                         NOT NULL,
        lineXY          arr                         NOT NULL,
        lineAngle       float                       NOT NULL,
        beamExcistence  float                       NOT NULL,
        beamShape       arr                         NOT NULL,
        beamStatus      float                       NOT NULL,
        timeStamp       text                        NOT NULL); """



        # create a database connection
        self.conn = self.create_connection()

        # create tables
        if self.conn is not None:
            # create temperature table
            self.create_table(temp_table)

            # create image data table
            self.create_table(image_table)

            print("===== Created connection to database =====")

        else:
            print("Error! cannot create the database connection.")


    def create_connection(self):
        """ create a database connection to a SQLite database """
        conn = None
        try:
            conn = sqlite3.connect(self.dbPath, check_same_thread =
               False, detect_types=sqlite3.PARSE_DECLTYPES)
            return conn

        except Error as e:
            print(e)

        return conn

    def create_table(self, create_table_sql):
        """ create a table from the create_table_sql statement
        :param conn: Connection object
        :param create_table_sql: a CREATE TABLE statement
        :return:
        """
        try:
            c = self.conn.cursor()

            c.execute(create_table_sql)

        except Error as e:
            print(e)

    def update_temperature(self, data):

        sql = ''' INSERT INTO temperature(tempMax, tempMean, tempMin, timestamp)
                  VALUES(?,?,?,?) '''

        cur = self.conn.cursor()
        cur.execute(sql, data)
        self.conn.commit()
        return cur.lastrowid

    def update_imagedata(self,data):

        sql = ''' INSERT INTO imagedata(inliers, lineXY, lineAngle, beamExcistence, beamShape, beamStatus, timestamp)
                  VALUES(?,?,?,?,?,?,?) '''

        cur = self.conn.cursor()

        cur.execute(sql, data)

        self.conn.commit()
        return cur.lastrowid

    def select_data(self):

        cur = self.conn.cursor()
        cur.execute("SELECT * FROM temperature ORDER BY id DESC LIMIT 1")

        rows_temperature = cur.fetchall()

        cur.execute("SELECT * FROM imagedata ORDER BY id DESC LIMIT 5")

        rows_imagedata = cur.fetchall()

        return rows_temperature, rows_imagedata

if __name__ == '__main__':


    database = Database(r'/home/elkem/GitLab/elkemvision/webapp/database
        /data.db')

    _, beam_data = database.select_data()
    beam_existence = beam_data[0][4]
    beam_angle = beam_data[0][3]
    print(beam_angle)
```

C.2 Human Machine Interface

C.2.1 mainFlask.py

```python
#!/usr/bin/env python

from webapp import app


if __name__ == '__main__':
    app.run(host='0.0.0.0', debug=False)
```

C.2.2 home.py

```python
#!/usr/bin/env python

from webapp import app, path
from src import Database
from flask import render_template, url_for, Response
import numpy as np
import cv2
import io
import random
import json
import time
from datetime import datetime
import os




# Connect to database
path_db = os.path.join(path,'database','data.db' )
database = Database.Database(path_db)

# Capture UDP stream
cap = cv2.VideoCapture("udpsrc port=5000 ! application/x-rtp ! queue !
    rtph264depay ! h264parse ! nvv4l2decoder ! nvvidconv ! videoconvert
    ! appsink")

# Defines the
beam_angle_good = np.array([0,72])
beam_angle_bad = np.array([72,120])

# Yield images
def frame_thermal():
    while True:
        ret, frame = cap.read()
        frame_split = frame[:,0:383]
        if frame_split is not None:
            ret, frame_split = cv2.imencode('.jpg', frame_split)
            frame_split = frame_split.tobytes()
            yield (b'--frame\r\n'b'Content-Type: image/jpeg\r\n\r\n' +
                frame_split + b'\r\n')

def frame_binary():
    while True:
        ret, frame = cap.read()
        frame_split = frame[:,383:767]
        if frame_split is not None:
            ret, frame_split = cv2.imencode('.jpg', frame_split)
            frame_split = frame_split.tobytes()
            yield (b'--frame\r\n'b'Content-Type: image/jpeg\r\n\r\n' + frame_split + b'\r\n')

def frame_shape():
    while True:
        _, beam_data = database.select_data()
        beam_existence = np.mean(np.array([beam_data[0][4], beam_data[1][4], beam_data[2][4], beam_data[3][4], beam_data[4][4]]))
        beam_status = np.mean(np.array([beam_data[0][6], beam_data[1][6], beam_data[2][6], beam_data[3][6], beam_data[4][6]]))


        if beam_existence == 1 and beam_status == 1:
            frame = cv2.imread('webapp/static/HMI_beam_shape_optimal.png',1)

        elif beam_existence == 1 and beam_status == 0:
            frame = cv2.imread('webapp/static/HMI_beam_shape_split.png',1)

        elif beam_existence == 0:
            frame = cv2.imread('webapp/static/HMI_beam_shape_stopped.png',1)

        else:
            frame = None

        if frame is not None:
            ret, frame = cv2.imencode('.png', frame)
            frame = frame.tobytes()
            yield (b'--frame\r\n'b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n')


def frame_angle():

    while True:
        _, beam_data = database.select_data()
        beam_existence = np.mean(np.array([beam_data[0][4], beam_data[1][4], beam_data[2][4], beam_data[3][4], beam_data[4][4]]))
        beam_angle = np.mean(np.array([beam_data[0][3], beam_data[1][3], beam_data[2][3], beam_data[3][3], beam_data[4][3]]))

        if beam_existence == 1 and beam_angle_good[0] < beam_angle <= beam_angle_good[1]:
            frame = cv2.imread('webapp/static/HMI_beam_angle_free_pouring.png',1)

        elif beam_existence == 1 and beam_angle_bad[0] < beam_angle <= beam_angle_bad[1]:
            frame = cv2.imread('webapp/static/HMI_beam_angle_sticking.png',1)

        elif beam_existence == 0:
            frame = cv2.imread('webapp/static/HMI_beam_shape_stopped.png',1)

        else:
            frame = None


        if frame is not None:
            ret, frame = cv2.imencode('.png', frame)
            frame = frame.tobytes()
            yield (b'--frame\r\n'b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n')


@app.route('/')
def index():
    """ Home template """
    return render_template('home.html')


@app.route('/camera_feed_thermal')
def camera_feed_thermal():
    """Video streaming route."""
    return Response(frame_thermal(), mimetype='multipart/x-mixed-replace; boundary=frame')


@app.route('/camera_feed_binary')
def camera_feed_binary():
    """Video streaming route."""
    return Response(frame_binary(), mimetype='multipart/x-mixed-replace; boundary=frame')


@app.route('/beam_status_image')
def beam_status_image():
    """ Beam status image """
    return Response(frame_shape(), mimetype='multipart/x-mixed-replace;
        boundary=frame')


@app.route('/beam_angle_image')
def beam_angle_image():
    """ Beam angle image """
    return Response(frame_angle(), mimetype='multipart/x-mixed-replace;
        boundary=frame')


@app.route('/beam-temp')
def beam_temp():
    """ Temperature-data route """
    def generate_temperature_data():
        while True:
            temp_data, _ = database.select_data()
            json_data = json.dumps(
                {'time': temp_data[0][4], 'value': temp_data[0][1:4]})
            yield f"data:{json_data}\n\n"
            time.sleep(1)

    return Response(generate_temperature_data(), mimetype='text/event-
        stream')
```

C.2.3 base.html

```html
<!doctype html>
<html lang=en>

<head>
    <!-- Bootstrap CSS -->
    <link rel="stylesheet" href="static/bootstrap/css/bootstrap.min.css"
        >

    <title>{% block title %}{% endblock %}</title>
</head>

    <body>
        <nav class="navbar navbar-expand-lg navbar-light bg-light">
            <div class="container-fluid">
              <a class="navbar-brand" href="#"><img src="{{url_for('
                 static', filename='elkem-logo.png')}}"></a>
              <button class="navbar-toggler" type="button" data-bs-
                 toggle="collapse" data-bs-target="#navbarNav" aria-
                 controls="navbarNav" aria-expanded="false" aria-label=
                 "Toggle navigation">
                <span class="navbar-toggler-icon"></span>
              </button>
              <div class="collapse navbar-collapse" id="navbarNav">
                <ul class="navbar-nav">
                  <li class="nav-item">
                    <a class="nav-link active" aria-current="page" href=
                       "/">Home</a>
                  </li>
                  <li class="nav-item">
                    <a class="nav-link disabled" href="#" tabindex="-1"
                       aria-disabled="true">Data</a>
                  </li>
                </ul>
              </div>
            </div>
          </nav>
        {% block home %}

        {% endblock %}


            <!-- Option 1: Bootstrap Bundle with Popper -->
        <script src="static/bootstrap/js/bootstrap.bundle.min.js"></
            script>

        <!-- Option 2: Separate Popper and Bootstrap JS -->

        <!-- <script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.5.4/dist/umd/popper.min.js" integrity="sha384-q2kxQ16AaE6UbzuKqyBE9/u/KzioAlnx2maXQHiDX9d4/zp8Ok3f+M7DPm+Ib6IU" crossorigin="anonymous"></script>
        <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta1/dist/js/bootstrap.min.js" integrity="sha384-pQQkAEnwaBkjpqZ8RU1fF1AKtTcHJwFl3pblpTlHXybJjHpMYo79HY3hIi4NKxyj" crossorigin="anonymous"></script> -->


    </body>

</html>
```

C.2.4 home.html

```
{% extends "base.html" %}

{% block title %} Home Page {% endblock %}

{% block home %}

<body>
    <h1>Tapping Beam Monitor</h1>
</body>
<body>
    <!-- Image stream-->
    <div class="container">
        <div class="row">
          <div class="col-md-4">
            <img src="camera_feed_thermal" alt="Raw thermal image">
          </div>

          <div class="col-md-4">
            <img src="camera_feed_binary" alt="Binary thermal image">
          </div>

          <div class="col-md-4">
            <div class="row">
                <div class="col-12"><img src="beam_status_image" alt="
                   Tappingbeam"></div>
                <div class="col-12"><img src="beam_angle_image" alt="
                   Tappingbeam"></div>
            </div>
          </div>

        </div>
      </div>
</body>


<!-- Temperature plot  -->
<body>
    <div class="container">
        <div class="row">
            <div class="col-12">
                <div class="card">
                    <div class="card-body">
                        <canvas id="canvas_temp"></canvas>
                    </div>
                </div>
            </div>
        </div>
    </div>
    <!--suppress JSUnresolvedLibraryURL -->
    <script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.4.0/
       jquery.min.js"></script>
    <!--suppress JSUnresolvedLibraryURL -->
    <!-- <script src="https://cdnjs.cloudflare.com/ajax/libs/twitter-
       bootstrap/4.3.1/js/bootstrap.min.js"></script> -->
    <!--suppress JSUnresolvedLibraryURL -->
    <script src="https://cdnjs.cloudflare.com/ajax/libs/Chart.js/2.8.0/
       Chart.min.js"></script>

    <!-- Chart JavaScript -->
    <!-- <script src="static/js/chart.min.js"></script> -->

    <script>
        $(document).ready(function () {
            const config = {
                type: 'line',
                data: {
                    labels: [],
                    datasets: [{
                        label: ["Max temp."],
                        backgroundColor: ['rgba(255, 99, 132, 0.2)'],
                        borderColor: ['rgba(255, 99, 132, 1)'],
                        data: [],
                        fill: false,
                    }, {
                        label: ["Mean temp."],
                        backgroundColor: ['rgba(54, 162, 235, 0.2)'],
                        borderColor: ['rgba(54, 162, 235, 1)'],
                        data: [],
                        fill: false,
                    }, {
                        label: ["Min temp."],
                        backgroundColor: ['rgba(255, 206, 86, 0.2)'],
                        borderColor: ['rgba(255, 206, 86, 1)'],
                        data: [],
                        fill: false,
                    }],
                },
                options: {
                    responsive: true,
                    title: {
                        display: true,
                        text: 'Tappingbeam temperature'
                    },
                    tooltips: {
                        mode: 'index',
                        intersect: false,
                        enabled: false
                    },
                    hover: {
                        mode: 'nearest',
                        intersect: true
                    },
                    scales: {
                        xAxes: [{
                            display: true,
                            scaleLabel: {
                                display: true,
                                labelString: 'Time'
                            }
                        }],
                        yAxes: [{
                            display: true,
                            scaleLabel: {
                                display: true,
                                labelString: 'Value'
                            }
                        }]
                    }
                }
            };

            const context = document.getElementById('canvas_temp').getContext('2d');

            const lineChart = new Chart(context, config);

            const source = new EventSource("/beam-temp");

            source.onmessage = function (event) {
                const data = JSON.parse(event.data);
                if (config.data.labels.length === 20) {
                    config.data.labels.shift();
                    config.data.datasets[0].data.shift();
                    config.data.datasets[1].data.shift();
                    config.data.datasets[2].data.shift();


                }
                config.data.labels.push(data.time);
                config.data.datasets[0].data.push(data.value[0]);
```

```
                    config.data.datasets[1].data.push(data.value[1]);
                    config.data.datasets[2].data.push(data.value[2]);
                    lineChart.update();
                }
            });
        </script>
        </body>

{% endblock %}
```