OsloMet – the Metropolitan University
Faculty of Technology, Art and Design
Department of Machinery, Electronics and Chemistry
PO Box 4 St. Olavs plass, 0130 Oslo
Phone: 67 23 50 00

PROJECT NO. 2218

AVAILABILITY 25/05/2025

BACHELOR THESIS

TITLE OF BACHELOR THESIS	DATE
Unmanned inspection of remote installations	25/5-2022
	NUMBER OF PAGES / ATTACHMENTS 76/53
AUTHORS Henrik Westgaard, Vebjørn Berstad Ivar Øksendal, Tommy Tran	INTERNAL SUPERVISOR Nils Sponheim

PERFORMED IN COLLABORATION WITH	EXTERNAL GUIDE
ABB	Valentinos Kongezos Waqas Ikram

SUMMARY
Automation in industry is generally a static process without flexibility that performs the same operation over and over without input. With recent development in artificial intelligence there is opportunity to automate processes that have previously required human intervention. This new technology builds on artificial neural networks (ANN) which mimics the biological process of decision making. Creating technology capable of giving itself inputs to produce outputs that fall in line with—or better than—informed human decisions open up for an entirely new branch of automation. An issue with adaptive technology is that it may have unexpected outcomes and become inconsistent in its performance. In an industrial setting with untrained personnel this can produce an unsafe workplace with potentially disastrous consequences.Therefore developing a consistent solution with redundancy is a challenging task. Specifically; image recognition as a tool for error location and identification will be a critical part of this process and should therefore be extremely reliable. In this thesis we pursue a solution to an automated camera system for object detection using machine learning and computer vision. The system detects instruments and uploads images and videos to a cloud server. Such a system requires a robust and reliable solution that works in different scenarios.
One of the main focuses in this thesis is training a functional AI to send inputs to a PLC. This is the continued work of a series of projects where the end goal is to make a fully functional system for inspection of unmanned remote installations.

3 KEYWORDS
Machine learning
PLC
Programming

OSLOMET

Abstract

English
Automation in industry is generally a static process without flexibility that performs the same operation over and over without input. With recent development in artificial intelligence there is opportunity to automate processes that have previously required human intervention. This new technology builds on artificial neural networks (ANN) which mimics the biological process of decision making. Creating technology capable of giving itself inputs to produce outputs that fall in line with—or better than—informed human decisions open up for an entirely new branch of automation. An issue with adaptive technology is that it may have
unexpected outcomes and become inconsistent in its performance. In an industrial setting with untrained personnel this can produce an unsafe workplace with potentially disastrous consequences. Therefore developing a consistent solution with redundancy is a challenging task. Specifically; image recognition as a tool for error location and identification will be a critical part of this process and should therefore be extremely reliable. This thesis proposes a potential solution to such a problem. It will continue the work proposed by a previous student body at OsloMet in 2021. In this thesis we pursue a solution to an automated camera system for object detection using machine learning and computer vision. The system detects instruments and uploads images and videos to a cloud server. Such a system requires a robust and reliable solution that works in different scenarios.
One of the main focuses in this thesis is training a functional AI to send inputs to a PLC. This is the continued work of a series of projects where the end goal is to make a fully functional system for inspection of unmanned remote installations.

Norwegian
Automasjon i industri er vanligvis en statisk prosess uten fleksibilitet som utfører den samme operasjonen igjen og igjen uten input. Med ny utvikling innenfor kunstig intelligens er det nye muligheter for å automatisere prosessene som tidligere har krevd menneskelig innblanding. Denne nye teknologien bygger på kunstige nevrale nettverk som etterligner beslutningsevnen til biologiske prosesser. Å lage teknologi i stand til å gi seg selv input som produserer output på linje med—eller bedre enn—informerte menneskelige besluttninger åpner for helt nye grener av automasjon. Et problem med adaptiv teknologi er at det kan lede til uventede utfall og bli inkonsistent i dens atferd. I en industriell setting med utrent personalia kan dette lede til en utrygg arbeidsplass med potensielt katastrofale konsekvenser. Derfor er det vanskelig å utvikle en konsistent løsning med redundans. Spesifikt; bildegjenkjenning som et verktøy for oppdaging av feil og identifisering vil være en kritisk del av denne prosessen, og bør derfor være ekstremt pålitlig. Denne avhandlingen foreslår en potensiell løsning til et slikt problem. Den vil videreføre arbeidet foreslått av en tidligere studentgruppe ved OsloMet i 2021. I denne thesen foreslår vi en løsning til et automatisert kamera system for gjenstands deteksjon, ved bruk av maskinlæring og kunstig syn. Systemet oppdager instrumenter og laster opp bilder og video til et skylagrings bibliotek. Et slikt system krever en robust og pålitlig løsning som virker i forskjellige scenarioer. En av hovedfokusene i denne thesen er å trene en funksjonell AI til å sende input til en PLC. Dette er fortsettelsen av arbeidet i en rekke prosjekter hvor sluttmålet er å lage et fullt funksjonelt system for inspeksjon av ubemannede avsidesliggende installasjoner.

Foreword

This Bachelor thesis is presented on behalf of ABB, by four electrical engineering students at the Oslo Metropolitan University. The project was made possible by the collaborative effort of ABB and the OsloMet faculty.

The students involved in the project share a background in automation technology, as well as a general interest in artificial intelligence and robotics. This interest prompted the search for an industry partner that would accommodate a project that allowed the group to apply their knowledge while remaining a learning challenge.

ABB has hosted this thesis to establish an industry connection for the students involved in the project. Providing a very valuable experience of working with a larger company within the respective field of study.

The project detailed in this thesis has provided the students with an opportunity to put their educational learnings to the test, and apply them to a real world problem that pushes the boundary of current technological development.

Acknowledgements

We would like to extend our sincere gratitude and thanks to Valentinos Kongezos and Waqas Ikram who represented ABB, and provided technical expertise and guidance, and Nils Sponheim from OsloMet Faculty who supervised the project and maintained guidelines for appropriate workload and expected outcome of concluded work.

Ivar Øksendal

Tommy Tran

Henrik Westgaard

Vebjørn Berstad

OSLOMET

Table of contents

List of Figures

List of Tables

Introduction

1 Introduction

With automation technology becoming more and more prevalent both within the commercial and industrial market, there is an opportunity to develop new and exciting technology. For industry, most automation is hard coded, and leaves little room for adjustment. This means the processes cannot be altered to account for outside factors, or take in new inputs while it is running. This is where the application of artificial intelligence can help improve the systems exponentially.

The solution the 2021 group concluded was a rail system with an attachable camera moving along a single axis patrolling an area of interest. The system was hard coded, meaning all actions were predetermined and no AI was involved in the process. For further reference see the full 2021 thesis report.

This project's development picks up where the 2021 project finishes. The 2022 students need to familiarise themselves with the previous group's work to be able to understand the system and make potential alterations before continuing the work towards a finished product.

To maintain good structure and consistent workload a Gantt diagram (see appendix) was made to accommodate individual skill and interest. Weekly meetings were scheduled for the students to present their work for the ABB representatives who provided the students with feedback and new tasks. At project start the restrictions imposed by the government to combat the outbreak of the Corona virus made having meetings in person a non-option, so digital meetings were held. These restrictions also meant that the hardware was inaccessible for a large portion of the project, and the students needed to make preparations to complete the project with minimal hardware interaction. For the later part of the project the system was also moved to a new location at Fornebu from the old office at Ole Deviks vei, during which it was also unavailable. This prompted a challenge for the students that meant more time needed to be spent on planning rather than execution. For this reason the goal of the project was to make an artificial intelligence (AI) integration for the image recognition, instead of further developing the physical system apart from a few aspects pertaining to system orientation.

In classic programming a system running code will execute in order of operations in a predictable fashion, producing the same result in every loop given the same conditions. For this project the group wishes to combine the use of classic code through PLC’s and innovative machine learning algorithms to produce a system capable of executing predictable and predetermined movement with a flexible AI that allows for on the go adjustments based on a changing input field.

It was important to develop new methods of integrating industrial technology with a more object-oriented programming language. PLC programming—specifically the AC500 unit—uses ABB automation builder which has fewer open source development libraries available than Python.

This thesis aims at presenting the work performed by the students in its entirety; structured in parts that describe the different components, research and work leading up to project completion.

1.1 Objective

The project is focused on developing a functional model that displays proof of concept within the given guidelines. It is encouraged for the reader to familiarise themselves with the criteria set by ABB for this project before continuing. A full list of criteria can be found in Appendix 7.1 Project requirements. As per Nils’ request scenario 4 remains an optional task.

The finished project should deliver the following features:

- Automatic AI based image recognition
- Labelling of detected items
- Automated patrols
- Recording and cloud storage of video feed
- Live video feed
- Mapping of patrolled area
- Functional GUI

Optionally the project should deliver a system capable of:

- Identifying change since the last patrol.

1.2 System overview

The system can be broken up into four major parts consisting of the Festo rail system (1.2.1), Camera (1.2.2), Network (1.2.3), and Software (1.2.4). These components each provide an integral part to the system as a whole and will be briefly introduced and evaluated for the reader to familiarise themselves with the project.

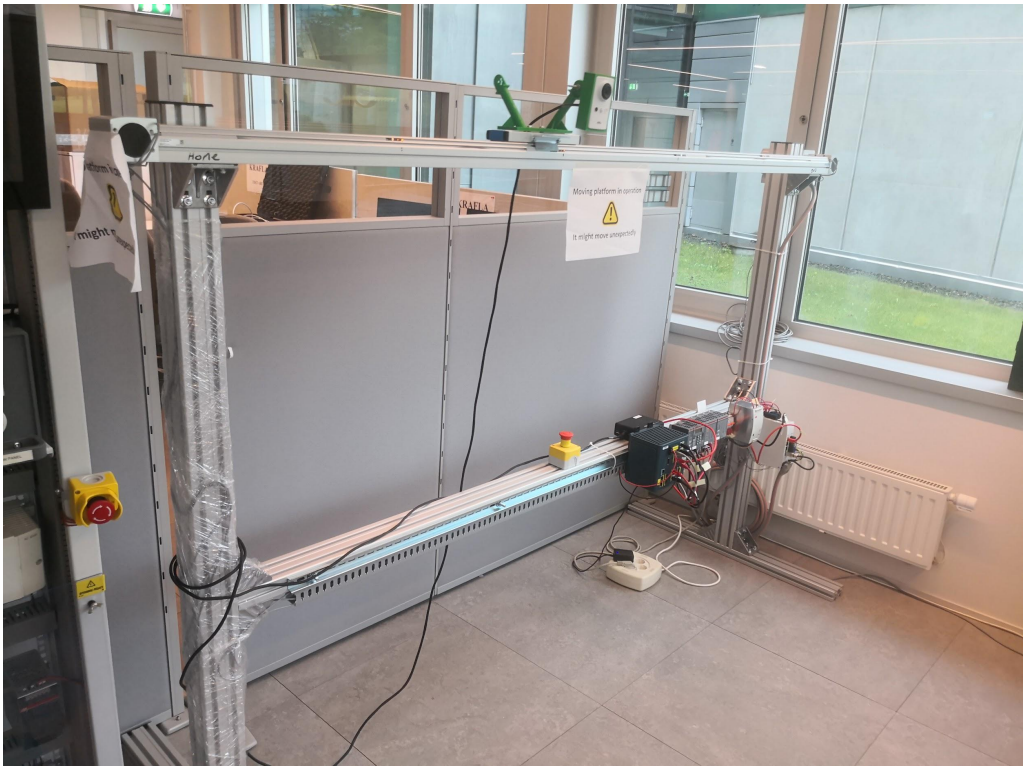


Figure 1: Picture of the system

1.2.1 Festo rail system

The rail system is a 2.0x1.6x0.2m mounting structure that houses a moving platform. It has a servo motor and motor drive connected to the conveyor belt that moves the attached platform, as well as proximity sensors that calculate the position of the platform.

The full list of specifications delivered by ABB for the rail system has been unaltered from the 2021 project and lists as following:

Table 1: List of rail and motor components provided by ABB (Tjernes, S., Listaul, M., & Mørk Madsen, T., 2021)

Item	Supplier	Description
ELGA-TB-G-80-2000-0H+6SB6SE4NS TANDREMSAKSE	Festo	2m Conveyor
EMMT-AS-80-S-LS-RM	Festo	Servo motor
CMMT-AS-C2-3A-PN-S1	Festo	Motor Drive
SMT-8M-A-PO-24V-E-7,5-OE	Festo	Proximity sensor
NEBM-M23G15-EH-5-Q10N-R3LEG14	Festo	Cable from drive to motor
Schneider Electric Surface Mount Mushroom Head Emergency Button - NO/NC, Twist to Reset, 40mm	RS-Online	E-Stop
RS PRO, 2 Way Joystick Switch Knob, Spring Return, IP65 Rated, 250V ac	RS-Online	Joystick
Axis Camera M1065-L	Dustin	Camera

1.2.2 Camera

The camera model used in this project is the Axis-M1065-L. This camera offers a 1920x1080 pixel resolution which is standard resolution for most commercial and industrial displays. For image sensor and lens we have the following specifications:

Camera	
Image sensor	1/3" progressive scan RGB CMOS
Lens	Fixed focal length, 2.8 mm, F2.0 Horizontal field of view: 110° Vertical field of view: 61° M12 mount, fixed iris, IR corrected

Figure 2: Camera specifications (ABB, 2017)

1.2.3 Network

The network is the connection between all components in the system.

- The LAB-PC serves as the system interface that controls the I/O of most components
- The PLC controls the speed of the motor drive
- Two cameras provide input for the LAB-PC
- Two switches for cameras and system connections respectively.

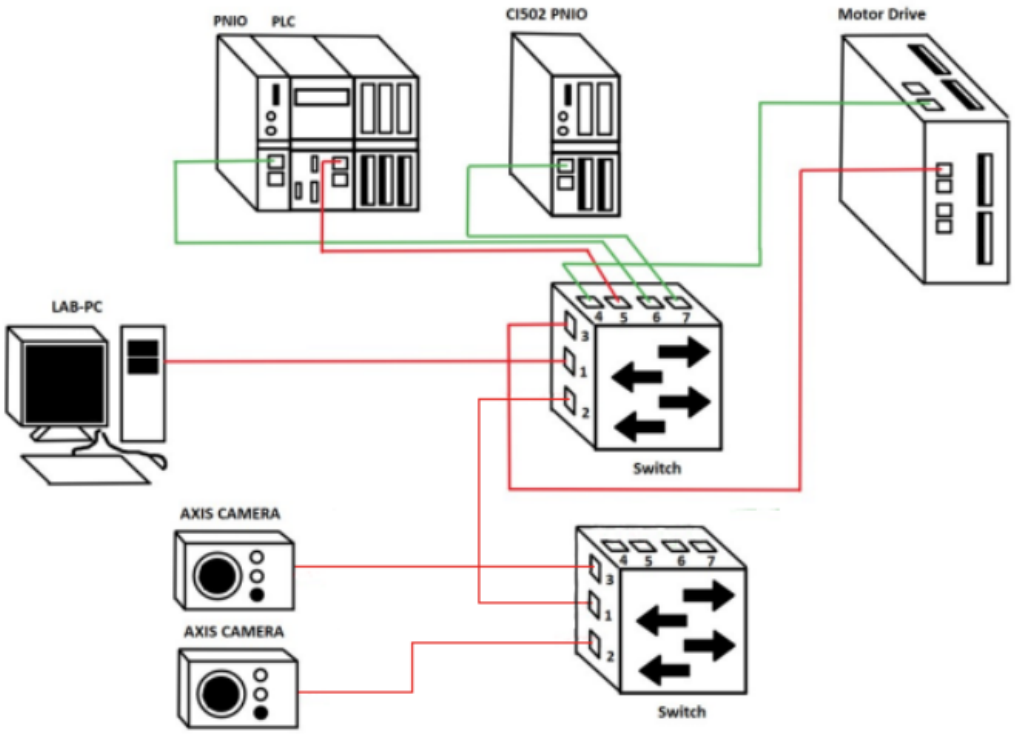


Figure 3: System setup. Red cables are physical connections and green cables are profinet connections (Tjernes et al., 2021)

1.2.4 Software

- Python
- ABB - Automation Builder
- Roboflow
- Festo Automation Suite
- TinkerCAD
- Cura

For image recognition Python provides large open source libraries and packages aimed at performing image recognition and labelling with a high degree of accuracy, and with constantly improving models.

Automation builder is based on CodeSys which uses C# as its base programming language. Python makes it possible to communicate with the AC500 PLC through packages developed to create easy integration. To interface PLC and Python the PyModbus library was used to allow for easy communication between programming languages.

Roboflow is used to develop frameworks for image recognition algorithms.

Festo automation suite is used to interface ABB automation builder with the AC500 PLC

TinkerCAD and Cura are Computer Aided Design (CAD) tools that are easy-to-use modelling software used for creating 3D printable structures.

1.3 Research

At project start the group was expected to perform a full risk assessment of the 2021 project. To do this the system limitations were first established and then evaluated for potential risks. This was done to determine the individual limits, from hardware to code. Each part of the system will have its own limits evaluated within their respective subcategory. The categories consist of the rail system, camera, network, and software all consisting of further subcategories pertaining to the individual parts.

1.3.1 System limitations

The system was running at 0.4 m/s on a two-metre-long rail, with an AXIS camera mounted. The system is controlled on a human-machine interface (HMI) remotely, with a stationary camera providing surveillance.

The limitations were grouped for easy identification and to classify which of the major components would bottleneck the systems performance the most.

Table 2: List of components and limitations

Limitation	Component
Velocity	Motor
Torque	
Deceleration	
Accuracy (Encoder)	
Environmental	Camera
View	
Weight	
Visibility	
Environmental	Rail system
Mobility	
Space	
Scalability	
Environmental	PLC
I/O	
Image processing	Software
Native processing	
Overfitting	
Cost	All

1.3.1.1 Motor limitation

As far as minimum speed limitations go, there are no tangible limits to the system as we can theoretically set the speed as low as we require.

The maximum speed for our rail is 5 m/s (Festo, 2016), but currently it's running at 0.4m/s. The current setup could theoretically fulfil ABB's requirement of being able to move at 1m/s, but the problem is that the prototype rail is only 2 metres long and because the setup does not have any brakes installed it will potentially overshoot. The previous bachelor group concluded that moving the camera at a speed of 1 m/s would result in a 200-300mm stopping distance (Tjernes et al., 2021).

As per the manual we can see that there is a trade-off in acceleration given by the mass of the object being moved by the rail system.

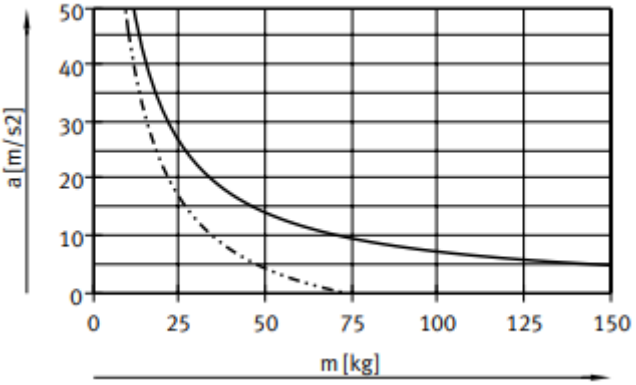


Figure 4: Shows the acceleration given the mass of the object (Festo, 2016, p16)

Another limitation of the system is if the torque of the motor isn't enough to provide a sufficient acceleration. This is not an issue with the current 2 m belt, but will be a limitation if we consider a larger conveyor belt. See 7.8 in appendix for calculations.

1.3.1.2 Camera limitation

The camera is limited by its field of view. The current system has a field of view of 110 degrees vertically, and 61 degrees horizontally. This creates limited visibility horizontally, and therefore also the environment surrounding the work area. The mounted AXIS camera's limited field of view may also cause a problem when paired with the speed of the motor and rail system, by making the accuracy of the detection lower than required for the system to be trusted.

The external environment will impact the camera's ability to see depending on multiple factors. Some of these environmental obstacles can be:

- Lighting
- Humidity
- Vibrations
- Visibility
- Depth of field
- Exposure

1.3.1.3 Rail system limitation

The previous bachelor thesis group recommended after discussion to use a rail system as the mobile platform, with an onboard motor solution. During the prototype phase ABB decided to use an

off-board motor solution. When using an off-board motor solution the scalability of the machine becomes a major limitation, as it makes the length of the toothed belt difficult to scale, and it makes rail switches complex, or impossible.

The space of the machine is another limitation of the machine, as it requires a huge amount of space. This does not hinder any of the requirements set by ABB.

If the current setup provided by ABB was to be used, the rail system may have some errors meeting the requirements. If the rail system is deployed in places which require a longer rail/conveyor, there might be changes needed for different parts of the system such as motor and belt.

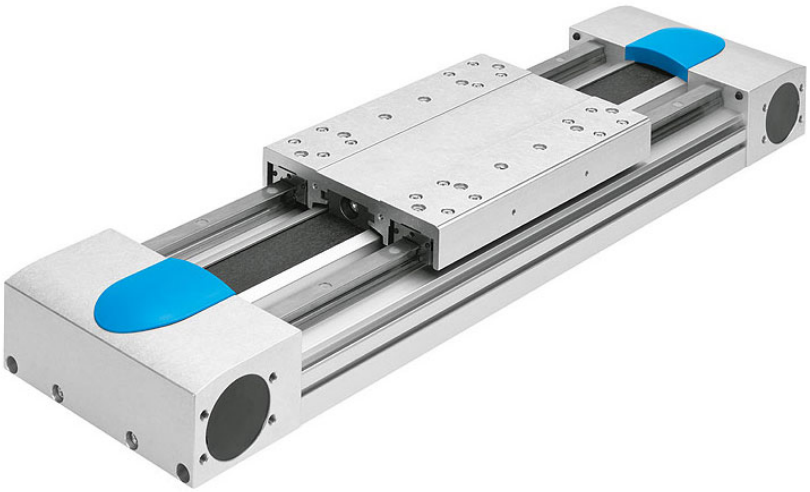


Figure 5: Toothed belt and spindle axes ELGA (FESTO, 2015)

1.3.1.4 PLC limitation

The programmable logic controller (PLC) that is used is the ABB AC500. From ABB's website it is described as "a reliable and powerful platform to design and create scalable, cost-effective and flexible automation solutions".

The PLC has environmental limitations which could cause hazards if placed in an environment with too high temperature or vibrations. The machine from the thesis "Unmanned inspection of remote installations" is designed to be placed in different environments, with varying temperature and vibration, which makes the environmental limitations of the PLC something to be considered.

As described by ABB this PLC is scalable by adding modules, but at the moment there are a set limitations on inputs and outputs available, both on the ethernet and profinet I/O module.

Figure 6: AC500 PLC unit (ABB, 2022)

1.3.1.5 Software limitations

The previous group decided to use OpenCV and TensorFlow with Keras as libraries for the project. OpenCV is a library used for image processing and is mainly used for real time computer vision. The group has used OpenCV in Python, but it can also be used in C/C++ and Java. OpenCV is built as an infrastructure for computer vision. TensorFlow is a low-level framework for machine learning, and Keras is an object oriented neural network library that is running on top of TensorFlow for easy development of machine learning models.

Figure 8: OpenCV, Keras, Tensorflow

One of the main limitations is the time used on image processing. OpenCV supports video processing, but the machine today takes pictures that are updated with a two second interval. The two second interval can be a major limitation for expanding the rail beyond two metres, as that would create limited visibility of the area that is being surveilled. This could lead to the machine not registering the

different instruments, or the machine would have to slow down to compensate, but that would result in an ineffective machine with limited visibility.

This limitation could be solved with cloud processing as the processing time would be negligible, but that would create a cyber security threat for the machine. The machine as of project start is running on native processing, and it works well on the small prototype as there are very few images to be processed, but it would cause a problem to have a video stream or pictures being processed continuously. OpenCV does not support cloud processing.

In the bachelor thesis “Unmanned inspection of remote installations” by the 2021 group, it is mentioned that the image classification only has a 20% accuracy in the new environment when only using pictures from the lab. The model later gets updated with pictures from the relevant environment, but this may indicate that the model is overfitted.

This solution can meet the original requirements, but there are uncertainties whether it would handle the image processing fast enough if there were to be multiple cameras added, or a video feed instead of pictures. If the image processing were to become faster, either by being more efficient or more processing power, the motor would be able to move with a higher velocity, and reach the requirement of 1 m/s.

1.4 Supportive Literature

To develop a general understanding and expertise of the subject at hand, the group has used the following works as major contributors in learning and solving potential problems throughout the project. These have served as guidelines on how to approach the task and provided the group a higher level of competency than at the start of the project.

- Andrew Ng - Machine Learning Yearning
- Deep Learning with Python, Second Edition - François Chollet

Andrew Ng has authored multiple books on the subject of machine learning, deep learning, and Python and is considered a renowned scientist within the field with a total of 200747 citations, and an h-index of 134 as of writing this report.

François Chollet is a renowned software engineer and researcher at Google, specialising in AI. He has authored the Keras library in Python as well as multiple books on deep learning. His works are commonly cited within the respective field.

Background and Theory

2 Background and Theory

For this section the report will provide a more in-depth explanation to the background and theory of the project. The reader is encouraged to familiarise themselves with its contents to provide context for later system implementation and decisions.

2.1 Network communication

2.1.1 Network

A network is defined as “a system of computers and peripherals that are able to communicate with each other” (Merriam-Webster, 2022). The internet is a type of network that connects devices. The most popular types of networks are LAN and WAN, but there are many other types of networks such as VPN (Virtual Private Network) and SAN (Storage Area Network).

2.1.2 LAN and WAN

A Local Area Network (LAN) is a collection of connected devices in one physical location, such as a home or an office (Cisco, 2022). When the LAN finds its way to other LANs using a network router it is considered a WAN. A Wide-Area Network (WAN) extends over a large geographical area and connects individual users or multiple LANs.

2.1.3 Networking hardware

To establish a network a specifically designed network hardware is used. The most commonly used devices are the router and switch. These two act as network nodes in their respective types of networks. The difference between these two devices is how they identify, send and receive to different devices connected to it.

A switch works as a central place for computers and every other wired network device on the network to talk and share information with each other. If the switch is connected to a router with a cable, this will allow it to have access to the internet. The main difference between router and switch is how they treat information and how they forward this information to the end-device. The switch finds the end-device using their respective MAC address while routers use IP-addresses to find their end-device.

Power over Ethernet (PoE) is a technology that allows users to deliver power over an ethernet cable. This is a good solution for places that have limited space, where power adapters or power sockets will not be needed. Devices such as IP-cameras, access-points and alarm systems use PoE as a standard.

2.1.4 IP addresses

An Internet Protocol Address (IP-Address) allows other computers and network devices to send and receive information. Each of the devices in a network has its own unique IP-address which allows them to identify the respective devices.

As of now the default address is IP Version 4 (IPv4) in this version there are around 4.3 billion unique addresses, which is formatted in a 32-bit number, displayed in a dotted decimal notation. Since the

number of network devices keeps on increasing IP Version 6 (IPv6) has been developed. IPv6 uses a 128-bit alphanumeric value to identify an endpoint device.
IPv6 has $4.3 \bullet 10^{38}$ unique addresses.

2.1.5 TCP/IP

The Transmission Control Protocol/Internet Protocol (TCP/IP) is the industry-standard communications protocol for interconnecting hosts, the internet and other computer networks (Parziale, L., et al. 2006). The Protocol is an abbreviation of the most used protocol Transmission Control Protocol (TCP) and Internet Protocol (IP) among other protocols in this suite.

TCP/IP is a connection-oriented transport protocol that sends data as an unstructured stream of bytes. By using sequence numbers and acknowledgment messages (Cisco, 2005). TCP/IP also recognises if a package is duplicated or sent multiple times, and will delete them.

2.1.6 ModBus

Modbus is an open-source serial communication protocol developed by Modicon. Modbus was developed for the use of transmitting information over serial lines between electronic devices. In a Modbus network there is a Modbus client(master) and one or multiple Modbus servers(slaves). The Master is the one to write and read the information supplied from the Modbus Slaves.

Modbus sends data as a series of ones and zeros called bits. With Modbus they are transferred as voltage where negative voltage is zero and positive voltage is one. Modbus sends these data packages extremely fast with a speed of 9600 baud (bits per second)(*What Is Modbus and How Does It Work?*, n.d.).

2.1.7 Profinet

PROFINET is an open Industrial Ethernet solution based on international standards. It is a communication protocol designed to exchange data between controllers and devices in an automation setting (*What Is PROFINET*, n.d.).

Profinet uses standard ethernet cable as its communication medium. In a network, Profinet components and other devices in the network are still able to use normal Ethernet protocols. Profinet is most often used in industrial settings where the delay and order of the data frame is precise and anticipated.

2.2 Hardware

2.2.2 PLC

A programmable logic controller (PLC) is a special form of microprocessor-based controller that uses a programmable memory to store instructions and to implement functions such as logic, sequencing, timing, counting and arithmetic in order to control machines and processes (Bolton, 2011, p. 14). A PLC's task is to receive signals from different sensors, process these signals using the programmed logic, and perform actions based on the programmed logic, often this includes sending the processed signals to actuators.

An advantage of using a PLC is the versatility of the system. They consist of various components that are suited across a wide range of applications, and are easily available. The most essential components are the central processing unit, and the input/output module (I/O module).

- The CPU interprets the signals and performs the logical operations according to the program stored in the memory
- The I/O module receives information from the external devices, and communicates the signals to the external devices.
- The power supply unit converts A.C. voltage to low D.C. voltage (5V) and supplies it to the modules.
- The memory unit stores the program, and data from the inputs and to the outputs.
- The programming device is used to program the PLC, often a computer is used for this purpose.
- The communication interface is used to receive and transmit data to other devices in networks.

A PLC is suitable for a lot of applications, but as with all technology, not suitable for every application. One drawback is the processing power of a PLC. They are not able to handle complex data, or operations with discrete inputs efficiently. This hinders the PLC being used for heavy processing operations, such as artificial intelligence.

2.3 Software

2.3.1 Python

Python is a high-level programming language created by Guido Van Rossum in 1991 (Srinath, R, 2017). The language was developed with the intention of improving the readability of the code and reducing the lines needed, making it easy to understand. It's also open source, so there is no limitation in using this language.
This has made it a popular programming language for a vast amount of tasks, such as machine learning. Because of its popularity, a large community is constantly improving it with each iteration. It has a large number of libraries for machine learning and artificial intelligence built-in.

It's compatible with Windows, Linux, macOS, Android operating systems as well as others (Python, n.d.).

2.3.1.1 CUDA

CUDA is a parallel computing platform and programming model created by NVIDIA (Oh, 2012).

2.3.1.2 Pytorch

Pytorch is a high-level framework used in machine learning that makes developing and training of model prototypes easy to do. The library is object oriented, and development of different model architectures and training can be compared to building with LEGO blocks. Pytorch wraps the torch C-backend to a Python UI. It supports the use of the GPU acceleration tool CUDA for faster training and implementation of machine learning models.

2.3.1.3 OpenCV

OpenCV (Open Source Computer Vision Library) is an open source software library used for computer vision and machine learning (OpenCV, n.d.). It is primarily used for computer vision in real-time, such as live video footage and image analysis, and works well with Python. The library has

multiple different user interfaces (UI), including a Python UI, and is supported by the GPU acceleration tool CUDA, to help accelerate the image processing done in OpenCV.

2.3.1.4 Google Colab

Google Colab is a cloud programming service for writing and executing Python code through the browser. It provides access to powerful computing resources such as NVIDIA Tesla K80 and T4 GPUs (graphics processing unit), free of charge.
This is a widely used service for users, and there are many tools available and free to use, such as algorithms for machine learning and computer vision.

2.3.1.5 Pymodbus

Pymodbus is a Python library used for full modbus protocol implementation. It includes functions for connecting to a modbus network, and reading and writing to registers.

2.3.2 PLC programming

2.3.2.1 ABB automation builder

ABB automation builder is a software solution for integrated development of PLC's and other automation products from ABB.

2.3.2.2 Programming language

To program a microprocessor-based system a set of programming instructions have to be developed into machine code to represent the program instructions. The international standard IEC 61131-3 for programmable logic controllers defines five standard programming languages used in PLC programming.

- Ladder programming (LD), graphical
- Function block diagram (FBD), graphical
- Structured text (ST), textual
- Instruction list (LI), textual
- Sequential function chart (SFC), graphical

Structured text is a high-level language that is based on the programming language pascal, and is similar to popular programming languages such as C. Structured text allows for easier development of complex algorithms, than the graphical languages defined in IEC 61131-3, although it may be more complicated than a graphical language.

In combination with structured text it is possible to use Function block diagram (FBD) programming language. This language describes the relationship between the input and output of programming blocks. The blocks are freely interconnected and form programming networks. The software comes with standard libraries for blocks such as the logic gate AND, OR and SR/RS-latches, but it is also possible to develop custom blocks using ST to develop easy to read, but complex FBD. The developers of the different brands of PLC's also have developed libraries for more complex programming and different scenarios.

2.3.2.3 Registers

The term register is used for an electronic device in which data can be stored. An internal relay is such a device (Bolton, 2015, p.189). Registers are usually stored in groups of 8, 16 or 32 bits, and used for data storage for data currently used by the central processing unit (CPU). Compared to hardware memory units it stores less data and is faster accessible for the CPU to gather and process.

2.4 Machine learning

2.4.1 Deep learning neural networks

Artificial intelligence (AI) is an old field that can be described as attempting to automate the cognitive process of humans. Machine learning is a subfield of artificial intelligence that attempts to create models purely from exposure to training data, or input data with associated output data. This is a different approach than traditional programming where the aim is to create a model which creates its own output data. The process of creating these models from exposure to the training data is called learning. Machine learning has multiple subfields that expand on this idea.

Deep learning is a specific subfield of machine learning: a new take on learning representations from data that puts an emphasis on learning successive layers of increasingly meaningful representations (Chollet, 2018, p. 8). The deep in deep learning isn't a reference to any kind of deeper understanding achieved by the approach; rather, it stands for this idea of successive layers of representation.

These layers are often represented by nodes illustrated as the circles in figure 19, and are interconnected as stacked layers. This approach was developed with inspiration on how the human brain was believed to work, but are not models of the brain, hence the name neural networks from neurobiology. A more appropriate name could be layered representation learning. In deep learning, layered representations are trained on neural networks.

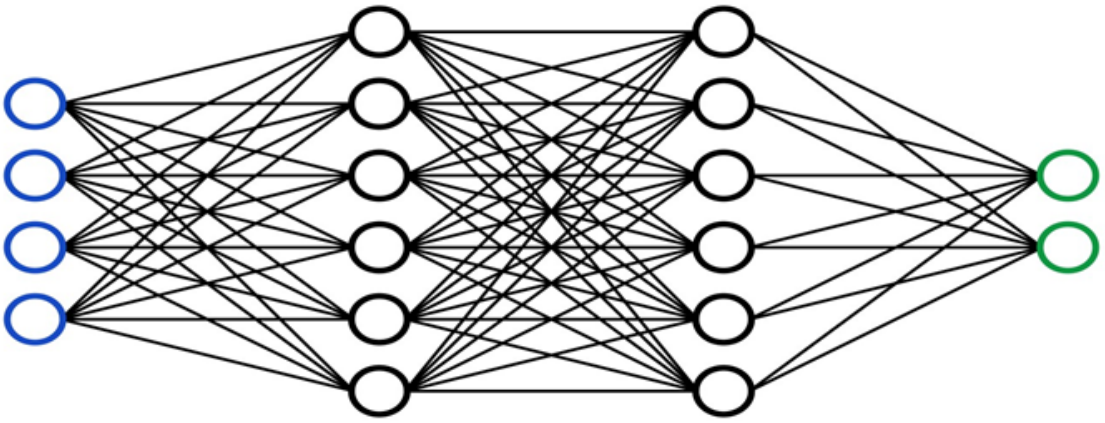


Figure 9: Illustration of neural network from Deep Learning with Python (Chollet, 2021)

As said, each layer is represented by nodes. Each node is a simple data transformation, or geometric function, and the layer is a set of geometric functions on inputs from the previous layer, with outputs connecting it to the next layer. The weights of the layer describe how the data is transformed, and are stored in matrices. In other words, the layers are parameterized by their corresponding weight matrices.

The objective of the learning is finding the weights of each layer, so that the network correctly predicts the inputs to their associated outputs. These networks may consist of unimaginable amounts of weights, and each weight affects the behaviour of the other layers and weights, therefore training these networks are process heavy operations that seemed impossible not long ago.

2.4.1.1 Training

To be able to adjust the weights of the model, it is essential to be able to measure the difference from actual output of the targets, and the true targets associated with the input. A loss function, also known as a cost function, or objective function, is used to fulfil this operation. The loss function takes the predictions of the network and the true target and computes a distance score, capturing how well the network has done on this specific example (Chollet, 2018, p. 10).

This signal is used in a feedback loop to adjust the weights and biases of each layer to develop a more useful representation. The aim is for each iteration to produce a lower loss score than the previous score; this process is done by the optimizer, and is called backpropagation. This process is known as the training loop, and is iterated until a sufficient loss score is reached. Figure 20 shows an illustration of the training loop.

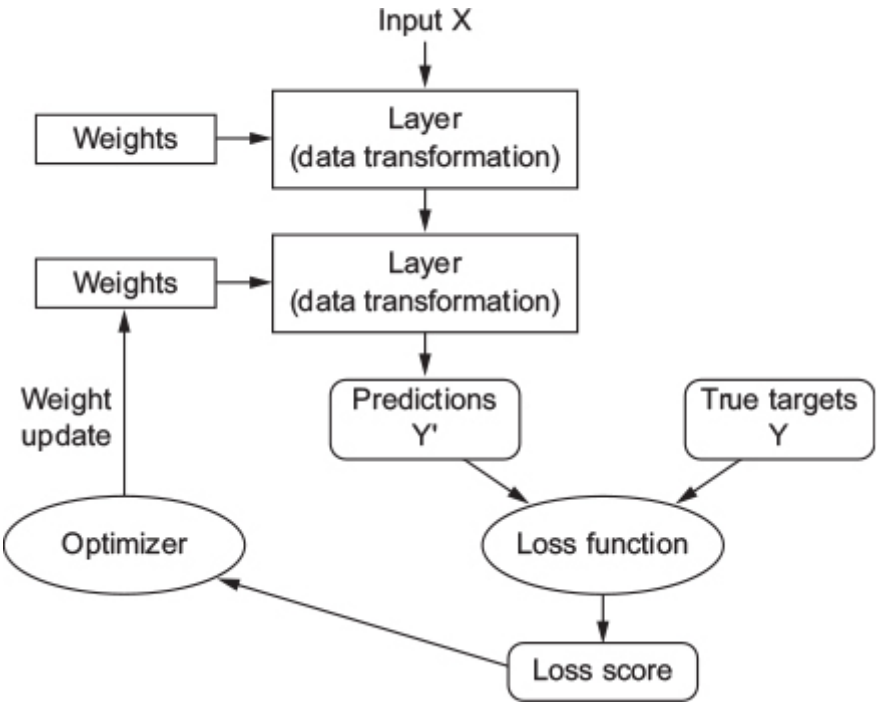


Figure 10: Flowchart showing training process (Chollet, 2018, Fig. 1.9)

Gradient descent and Backpropagation
Before training the weight matrices are initialised in a step called random initialisation, where each weight is assigned a random value. As previously stated, the aim is to update each weight to reduce the loss score until the model has a sufficient representation. The challenge is to update each weight so the loss score lessens, since there may be millions of weights in a model that needs to be updated.

Each operation done in the model is differentiable, in other words it can be derived. This property is taken advantage of to calculate the gradient of the loss function with regards to the network's coefficients, and moving the coefficients a step in the opposite direction from where the gradient is pointing, and thereby decreasing the loss score. The step is also known as the learning rate, and is important to set at an appropriate rate to increase the speed of the learning without sacrificing the performance of the model, and avoiding wrong convergence.

Example from Chollet (2018) of one iteration with gradient descent:

1. Draw a batch of training samples “x” and corresponding targets “y”.

2. Run the network on “x” to obtain predictions “y_pred”.
3. Compute the loss of the network on the batch, a measure of the mismatch between “y_pred” and “y”.
4. Compute the gradient of the loss with regard to the network's parameters (a backward pass).
5. Move the parameters a little in the opposite direction from the gradient – for example $W = W - (step * gradient)$ – thus reducing the loss on the batch a bit.

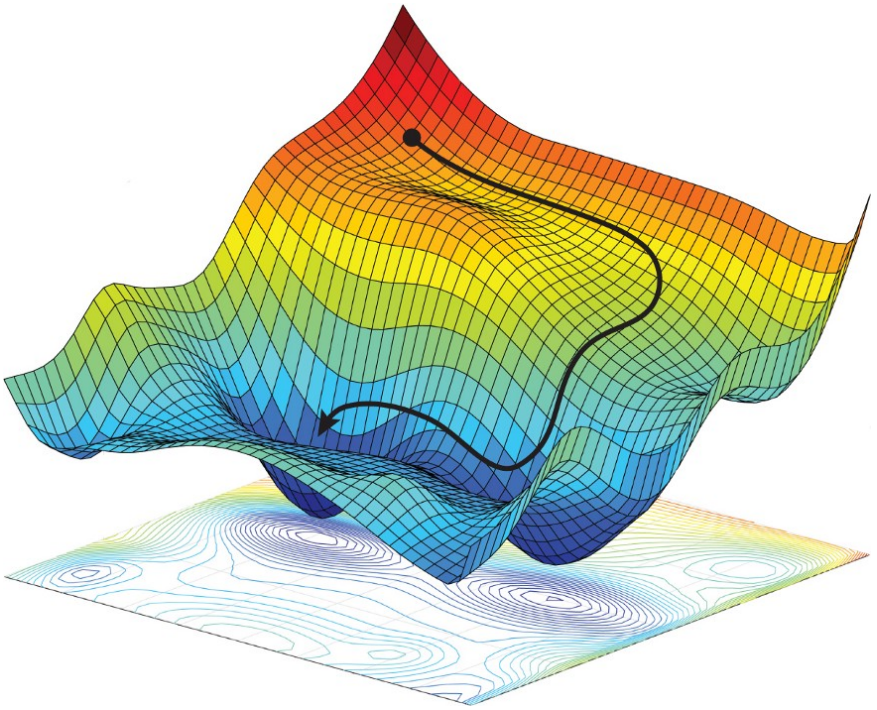


Figure 11: Showing gradient descent (A. Amini et al., 2018)

Backpropagation applies the chain rule: $f(g(x)) = f'(g(x)) + g'(x)$, to the computation of the gradient descent values, and starts from the top layer working its way backwards to the bottom layers of the model. It calculates each parameter's contribution, and increases the positive, and decreases the negative parameters contribution to the loss score.

Generalisation and overfitting
During training it is exposed to pre labelled data over many iterations to train the model. This training data becomes known to the model, and often the model performs with a better accuracy when exposed to this data contrary to new unseen data, this is called overfitting. Overfitting a model refers to a model that models the training data too well, and therefore performs suboptimally on unseen data. This is a common problem during development of deep learning models, and multiple solutions are used to tackle this problem, such as but not limited to reducing the network size, training over fewer epochs and adding dropout layers. A model that performs with the same accuracy on unseen data as the training data is called a generalised model, and is the goal of artificial intelligence.

2.4.1.2 Convolution neural networks

The current industry standard for image recognition is done by Convolutional Neural Networks (CNN). These networks utilise a method that lowers the required computational power drastically by sampling the area of interest and producing a response map that accurately represents the input data. The CNN architecture consists mainly of three different types of layers stacked together. These are convolutional layers, pooling layers and fully connected layers.

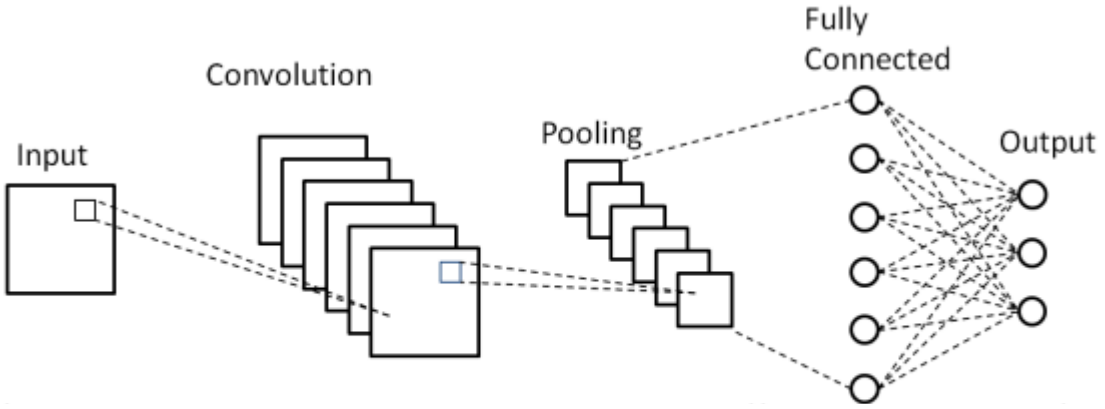


Figure 12: Convolution and pooling example (Chollet, 2018, p. 122)

The fundamental difference between a densely connected layer and a convolutional layer is this: Dense layers learn global patterns in their input feature space, whereas convolutional layers learn local patterns (Chollet, 2018, p. 122). This is done by extracting small patches of the input, often 3x3 or 5x5 pixels, applying filters and creating a response map.

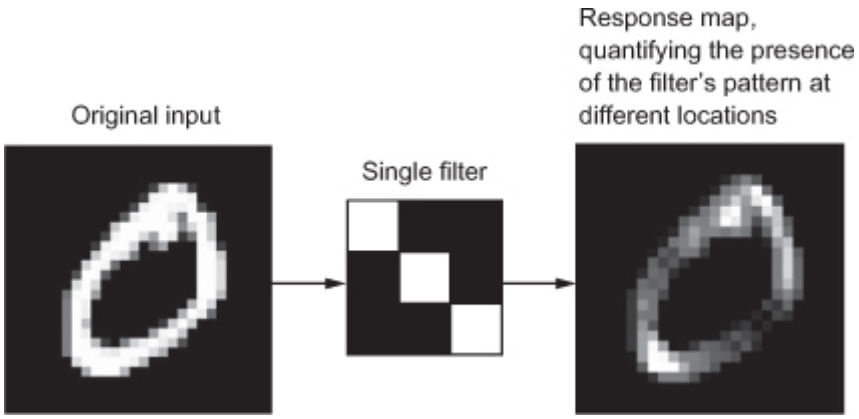


Figure 13: Filter to response map illustration (Chollet, 2018, fig. 5.3)

This makes the pattern's translation invariant, which means that the model will recognize the pattern anywhere on a picture. Conversely a densely connected layer is not translation invariant, and would have to relearn the pattern if encountered upon a new location on the picture.

A CNN learns spatial hierarchies of patterns, meaning that the first layers will learn local patterns, or features, and later layers will learn larger patterns made up of the features on the previous layers. This allows CNN's to efficiently learn increasingly complex and abstract visual patterns in the real world.

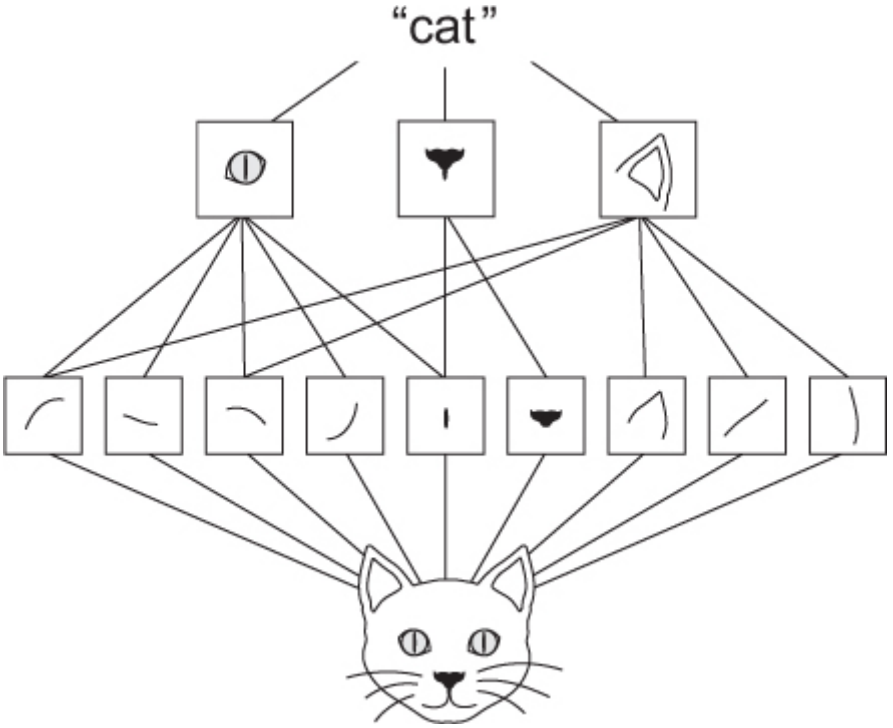


Figure 14: Shows layered features that combine to larger images (Chollet, 2018, Fig. 5.2)

2.4.1.3 YOLOv5

YOLOv5 (You Only Look Once) is an algorithm for object detection that divides images into a grid system. Each cell in the grid is responsible for detecting objects within itself. (YOLOv5 Documentation, 2020). It is a convolutional neural network (CNN) that is known for its high speed and accuracy. The YOLOv5 implementation was done in Pytorch while YOLOv4 was developed using another framework called DarkNet (Bochkovskiy, A., Wang, C. Y., & Liao, H. Y. M., 2020). There was no paper released with v5, but the architecture is very similar to the one of v4. The main difference is that v5 is proven faster than v4 on the COCO dataset, while maintaining accuracy.

The YOLOv5 architecture is split into four different sizes: YOLOv5s, YOLOv5m, YOLOv5l and YOLOv5x. YOLOv5s is the smallest one, and offers lower latency at the cost of mAP, as seen in figure 15.

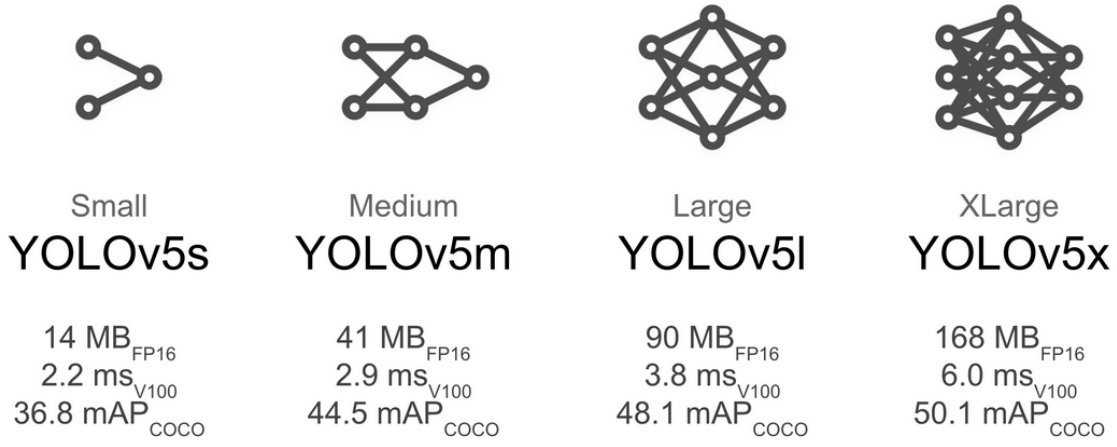


Figure 15: Image showing the four different YOLOv5 model sizes (YOLOv5 Documentation)

Mean average precision (mAP)
Mean average precision is used to determine the performance of an object detection model, like YOLOv5. The mAP consists of three measurements, the precision, the recall and the Intersection over Union (IoU) measurements.

The precision measures the number of times the model predicts correctly versus when it predicts false, and therefore does not measure the number of times it doesn't guess when it should have.

$$Precision = (Correct\ guesses)/(Correct\ guesses + False\ guesses)$$

The recall measures the model's ability to predict true positives, and disregards false positives. A true positive guess is a prediction that detects the object when it is on the image, a false positive is when it detects an object that is not on the images, and so on.

$$Recall = (True_{positive})/(True_{positive} + False_{negatives})$$

These two measurements are plotted on a curve and called the precision-recall curve, and the average precision(AP) is gathered from taking the integral under the precision-recall curve.

The last measurement is the IoU, which measures the intersection of the labelled bounding box and the predicted bounding box of the model. The prediction and recall measurements are taken at different thresholds of the IoU and the AP is calculated individually for each class at every IoU threshold. Then the mAP is calculated as the AP of each class at every IoU threshold.

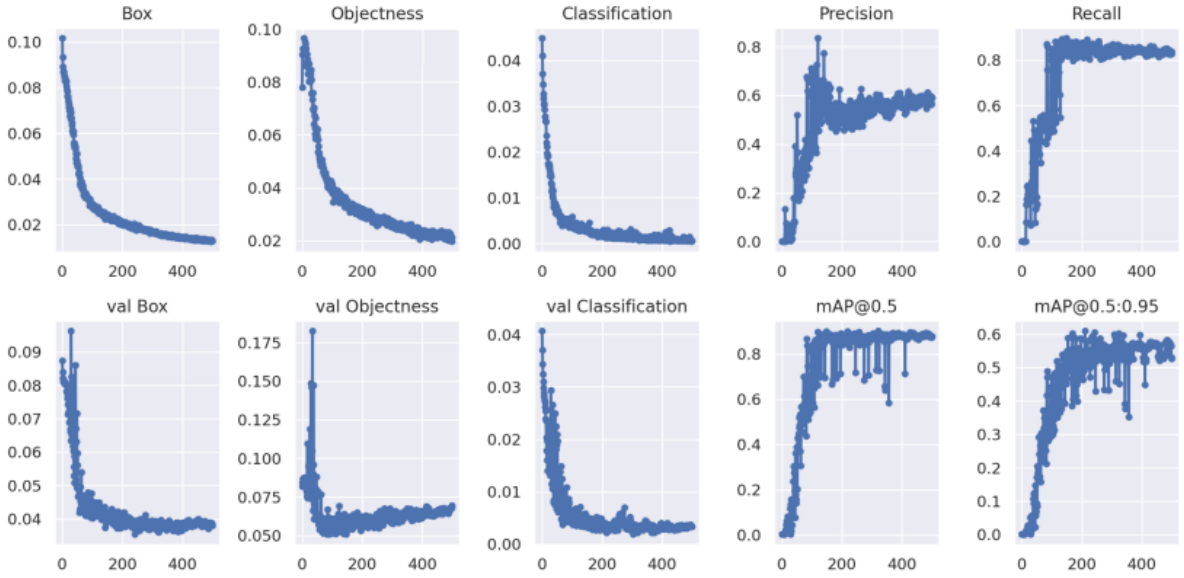


Figure 16: Graphs of how box loss, objectness loss, classification loss, precision, recall, and mAP could look like (Kasper-Eulaers, Hahn, Berger, Sebulonsen, Myrland and Kummervold, 2021).

The "Box" graph in figure 16 represents the model's ability to find the centre of the objects with the bounding box. The "Objectness" graph is a measure of the probability that an object exists in the region of interest. The "Classification" graph represents how good the model is at predicting the correct classes of a given object (Kasper-Eulaers et al., 2021).

2.4.2 Dataset

A dataset is a collection of data that is treated as a single unit by a computer. It contains many separate pieces of data that can be used to find predictable patterns when training an algorithm. It's important to understand that computers and humans understand data in completely different ways, so the data in a dataset should be made uniform and understandable for a computer. In order to achieve this, the bad images need to be removed and the good images need to be annotated so that the computer knows what it's looking for.

Datasets are usually split into a training set, a validation set and a training set.

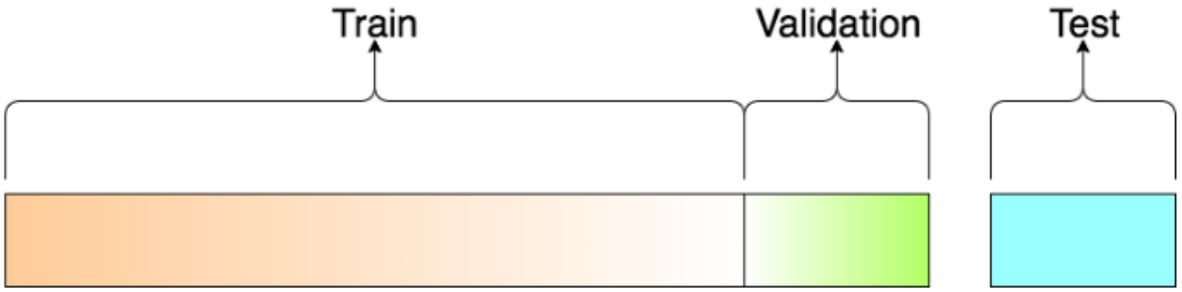


Figure 17: Visualisation of dataset splits (Towards Data Science, 2022)

The training set is used to train the model to find features and patterns in the data, also known as learning. This data becomes known to the model, and is therefore not used to combat overfitting, a unknown validation set is used. The validation set is used to validate the model's performance (Xu, Y., & Goodacre, R., 2018).

To accurately test how well a model works, the testing set is usually separated from the data. The test set is also separated from the rest of the data, and is used to test the model after the training is complete. It is designed to give unbiased information on the model's performance.

2.4.2.1 Augmentation

Augmentation of the dataset is the practice of applying different transformations to synthetically expand a training set (DeVries, T., & Taylor, G. W., 2017). There are many different ways of augmenting your data, and which method is most beneficial depends on what kind of dataset you have. For a visual dataset where the data contains images, augmentations like cropping, rotation, adjusting saturation, brightness and so on can prove beneficial.

This procedure can help combat overfitting, as this increases the variation and prevents the model from memorising the training data, and at the same time improves the chances of detecting the objects in other scenarios.

It is important that the dataset consists of high quality and clean pictures to avoid confusing the model, and it's also important to have a large enough dataset to avoid overfitting the model. There's no set amount of images you need in your dataset as it all depends on what kind of model you want, but as a rule of thumb you want a clean dataset with many high quality images.

OSLOMET

Implementation and System

3 Implementation and system

3.1 System overview

3.1.1 System introduction

We started our project by researching the system from the previous bachelor group. This work culminated in the two papers; systems limitations which is included in the first chapter as an introduction to the system, and a risk assessment which is included as an attachment 6: risk analysis in our thesis. After getting familiar with the system, we as a group were a part of deciding, along with ABB, what the requirements for the thesis became, and how we wanted to develop the system.

It was decided that the thesis would revolve around PLC programming, machine learning and further development of the camera system. The thesis got split into three mandatory scenarios and one optional scenario.

- **Scenario 1 (mandatory):** Simple video recording without image recognition.
  Automatic patrolling to record the video of the surroundings.

- **Scenario 2 (mandatory):** Video recording with image recognition.
  Automatic patrolling to detect the surroundings and identify the list of objects detected.

- **Scenario 3 (mandatory):** Show live video of the object.
  The user shall be able to specify the name of the object from the list created in Scenario 2. The camera system shall then automatically go to the location where the object is installed and show the live view to the user.

- **Scenario 4 (optional):** Show a list of changes with images. This shall be built on Scenario 2.

After receiving the scenarios the group started researching the main subjects which were identified to be PLC programming, Python programming, machine learning and network communication. During the research phase many different options were considered and studied, where several solutions were added in appendix.

During researching we identified the challenges of the thesis, and the phase concluded in a proposed design of the system, which includes, a communication solution between the LAB-PC and PLC, what programming language and libraries to use, the machine learning algorithm, and a proposed redesign of the camera solution and mount. These proposals were based on research, and the group's understanding of ABB's desired direction of the system.

- **Communication protocol**
  - ModBus TCP/IP communication between LAB-PC and PLC

- **Programming**
  - Python
  - Libraries
    - pyModBus
    - OpenCV
    - pyTorch

- **Machine learning**
  - YOLOv5
  - Roboflow

- **Camera solution**
  - Multiple camera solution
  - Adjustable mount for multiple cameras

It was identified that a PLC would not have the processing power, nor ready made frameworks to handle deep learning, and image processing. Our solution was to include the LAB-PC in the loop of the system to handle the process heavy operations, as we would have access to more process power in the form of a GPU, and frameworks and libraries for deep learning are available. The PLC would still be the main component of the system, by receiving signals from the LAB-PC and processing the received signals to control the program.

3.1.2 System flowchart

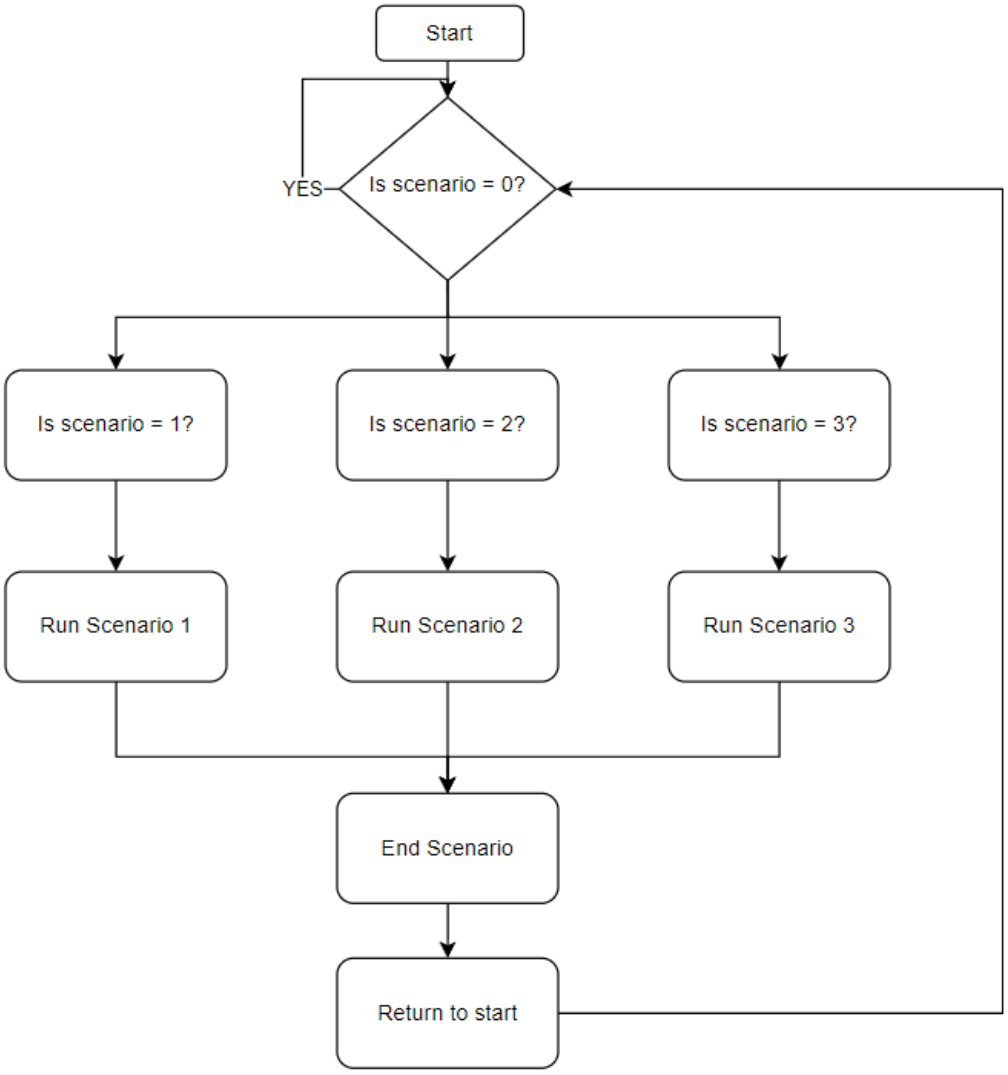


Figure 18: Flow chart of program control flow

3.2 Image recognition with Python

3.2.1 Dataset

Creating a good dataset is not just taking a bunch of images and throwing them into a model. In order to get a good performing dataset, you have to carefully choose what kind of images you want.

Our dataset consists of almost 2000 images. These images are split into approximately 1400 training images, 400 validation images and 200 test images. There are also about 100 background images in the dataset. Background images contain no objects and are added to the dataset to combat false positives.

Almost all of the images in the dataset are taken at a distance between 0.5 metres and 3 metres, as this is the work area of the system, as described in the project requirements (see appendix. 7.1.).

Blurry and low-quality images were deleted to avoid confusing the model.

The group was tasked with making a general model that would work in different installations and scenarios. In order to achieve this, a few factors had to be considered. The images had to be taken at:

- Different locations
- Different times of day and with different lighting
- Different angles
- Different distance from object

Having images taken at different locations is very beneficial for the models performance. This makes sure that the model doesn't start recognising the background instead of the object itself, making the model more generalised.

The first datasets we tested contained images taken in the same room at OsloMet campus, and at roughly the same time of day with similar lighting. The later versions of the dataset contained images from different rooms at campus and also at both ABB offices (the old one and the new one).

Having pictures taken from different angles was a priority from early on. The latest version of the dataset contains dozens upon dozens of images from all angles, except for angles it would not see realistically during operation, like the backside.

The dataset consists of 5 different classes for the instruments, as shown in figure 19.

Figure 19: From left to right: Pressure_Analog, Pressure_Rosemount, Temp_ABB, Temp_ABB, harvester and Pressure_ABB

3.2.2 Roboflow

There are several tools out there for annotating images and making datasets. Figure 20 shows annotation done in Roboflow on an image containing four objects. Annotating is done by marking the objects with a bounding box and assigning a class. Roboflow was chosen because of its intuitive design and ease of use.

All the images the group took of the instruments were uploaded to Roboflow and then annotated with bounding boxes and assigned classes, then resized to 640x360 to fit the model. An important thing to remember when annotating images, is being precise when placing the bounding boxes. This is because the model compares the labelled bounding boxes (IoU) to the predicted bounding boxes when validating. The images were then augmented with a rotation of 90 degrees both clockwise and counter-clockwise. After this, the dataset was generated and training could start.

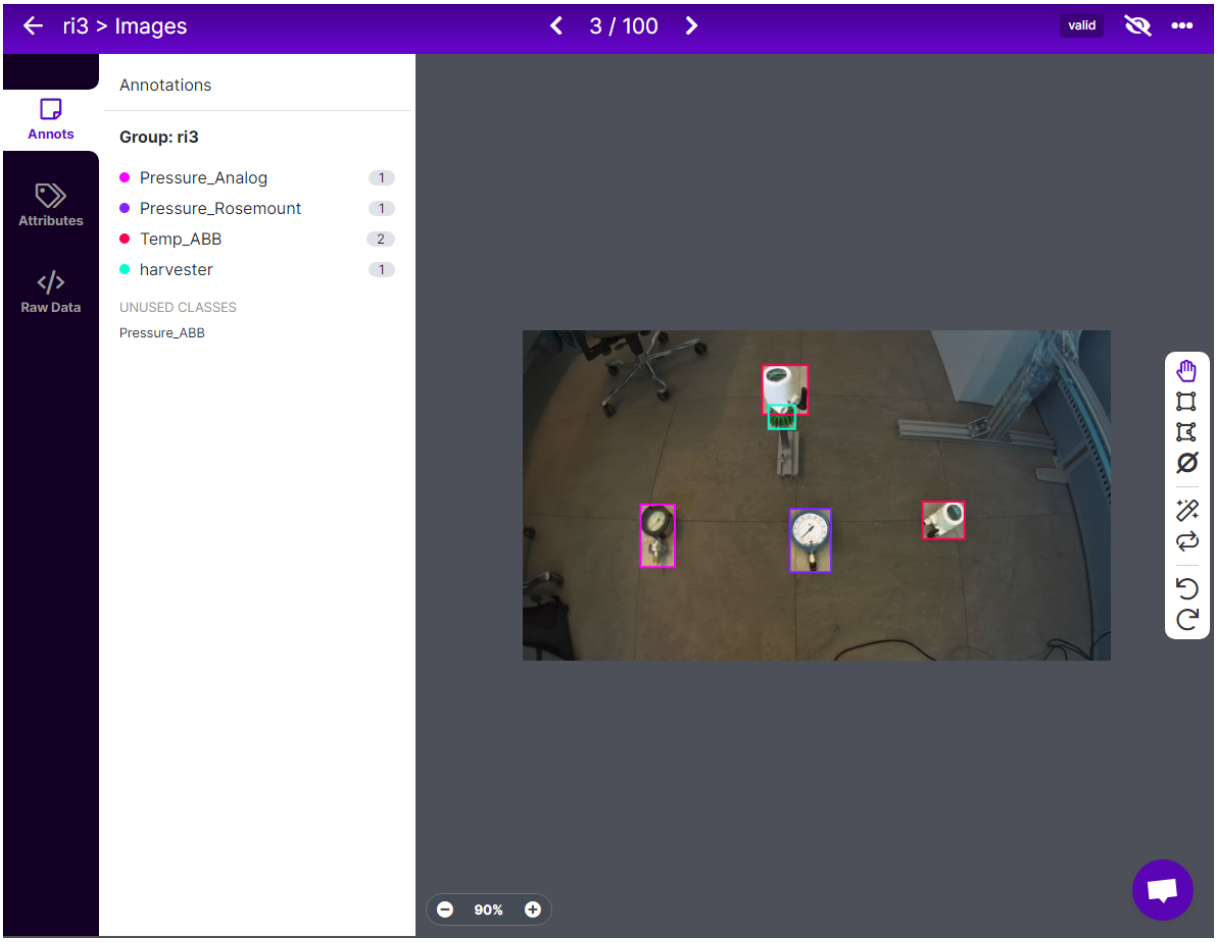


Figure 20: Annotating in Roboflow

3.2.3 YOLOv5

Because the camera is moving along a rail, the group had to find a fast working model for image recognition, while still being accurate enough to find all the instruments. The group did some research on different algorithms, and landed on using YOLOv5. This algorithm comes in different model sizes, and testing was done to find the optimal one for our use. We mainly tested the YOLOv5s algorithm and the YOLOv5m algorithm. We were able to process higher quality images with YOLOv5m, but this came at a cost of frames per second (FPS). As the camera is mounted on a moving rail, it was necessary for the PC to be able to produce enough FPS for the model to recognise the objects.

The YOLOv5s model was chosen and training could start. This was done using a Google Colab cloud computer.

Many different models were trained with different training parameters like epochs and batch size, and a couple of different datasets. One of the datasets only had pictures taken at ABB's current location at Fornebu, while the other datasets had pictures of the instruments taken at OsloMet, ABB's old building at Ole Deviks vei and of course also at Fornebu.

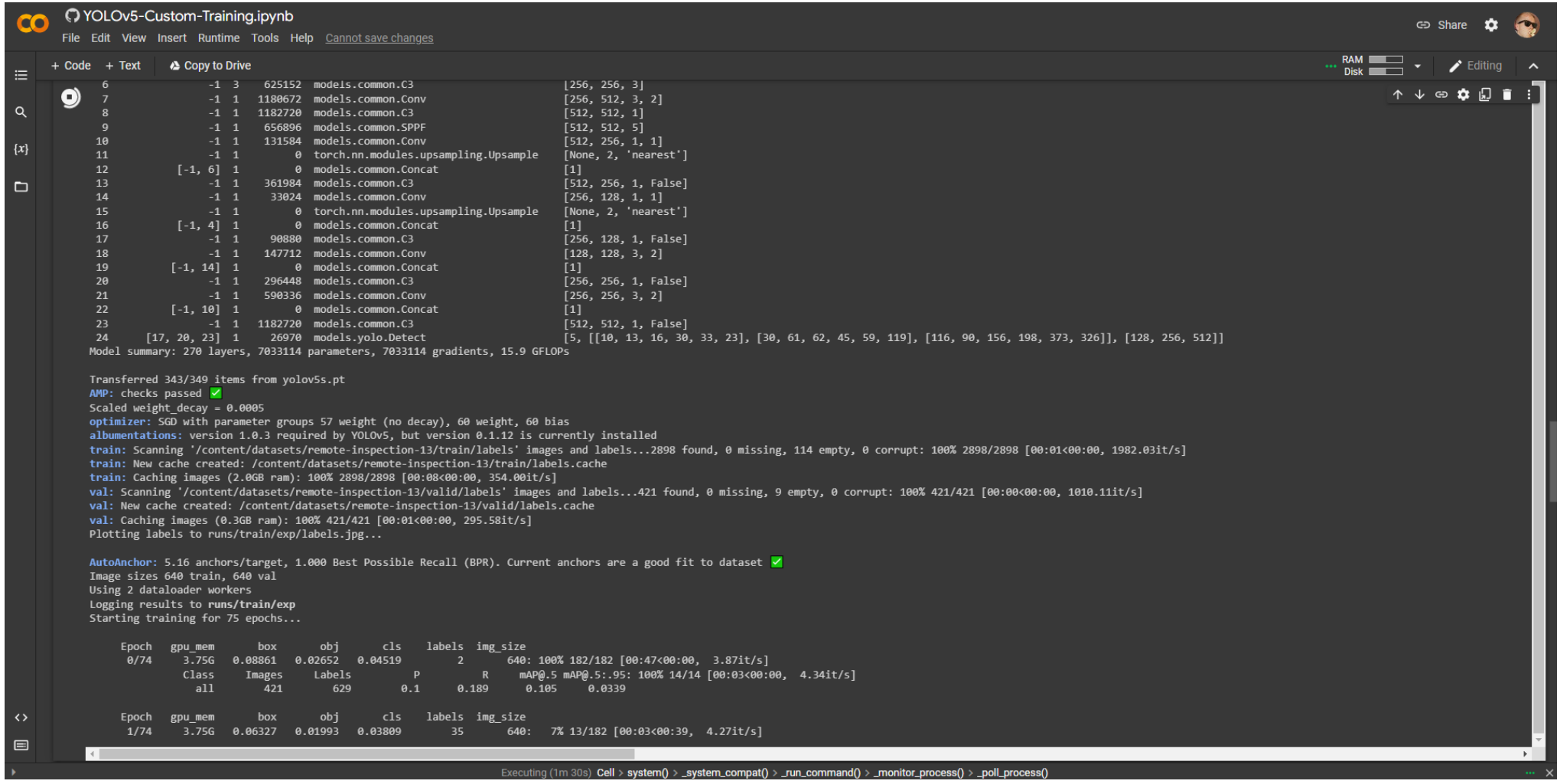


Figure 21: YOLOv5 training in progress on a Google Colab cloud computer

Figure 21 shows a screenshot of training in progress on Google Colab. When connecting to the cloud computer, you don't know what GPU you will get. Most of the time the group got connected to a computer with a NVIDIA Tesla K80 and sometimes we got NVIDIA Tesla T4. From our testing, the T4 trained the model about 5x as fast as with the K80.

3.3 PLC

3.3.1 Hardware

The hardware for the PLC was set up and installed as the previous group's bachelor thesis work. During our research of the system, it was concluded that the setup would fit our thesis work without adding further to it, as the hardware already supported the Modbus TCP/IP communication protocol we were going to implement.

The PLC system was assembled by four modules:

- CPU (PM5650)
- Digital/Analog input module (DA501)
- Profinet I/O module (CI502)
- Profinet interface (CM570-PNIO)

Figure 22: The modules of the PLC AC500 (Tjernes et al., 2021)

3.3.2 Software

From the previous bachelor thesis work, the group had set up all the different modules in ABB automation builder, and the communication between the modules, sensors and motor drive.

We needed to implement the modbus TCP/IP server protocol on the PLC to be able to communicate with the LAB-PC. This was done by adding the communication protocol in the project tree under the ethernet port in use, in our case that was ETH1.

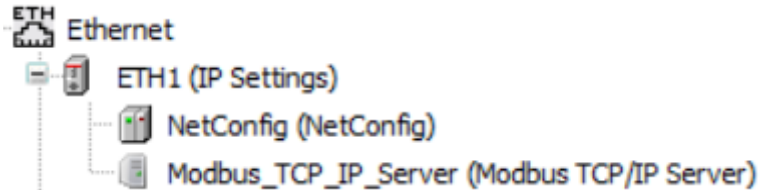


Figure 23: Modbus server protocol added in the project tree.

3.3.3 Programming

The previous bachelor group had made a time-scheduled patrol program, where the slide would stop at each proximity sensor located along the rail.

Our first action was to run their program, and see if it worked as expected and described in their thesis. This was done by booting up the system and logging into the PLC, then running their program through their virtual human machine interface (HMI), which was successful.

We wanted to create our system design by reusing one of the function blocks they had designed with logic gates, and reuse the variables connected through profinet to the motordrive. The function block we decided to reuse was named “Check E_S & RDY”, which is a redundant system for checking whether the motor is ready to run, based on the emergency stop (e-stop), digital e-stop and the motor drive.

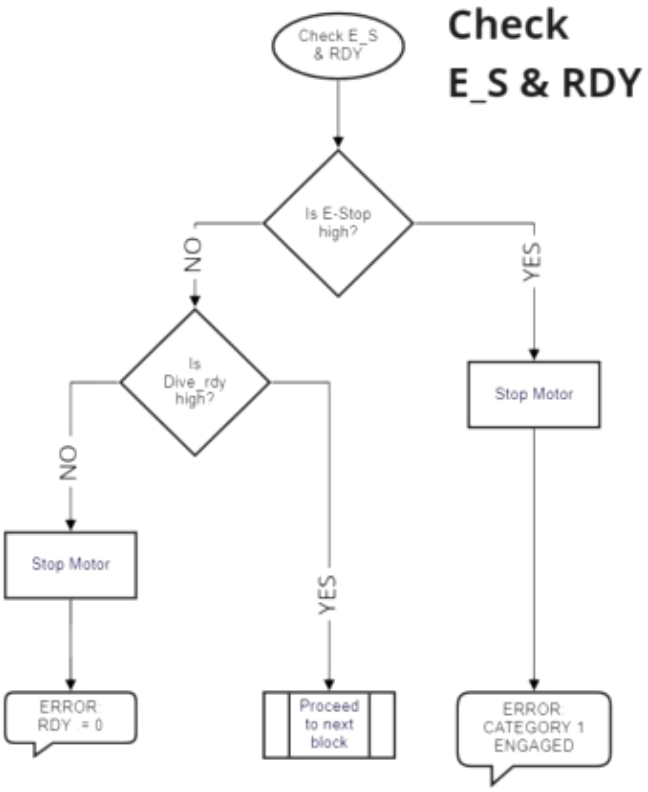


Figure 24: Flowchart of Check E_S & RDY (Tjernes et al., 2021)

We were tasked with creating three different mandatory scenarios. The group decided that this would be done with three programs in the PLC to make the system easy to understand. We started the programming by identifying the main similarities and differences between the scenarios by creating a flowchart (see appendix 7.9.) for each scenario, so we could identify the function blocks(FB) that were essential for each program.

One of the main components was implementing communication between the PLC and the LAB-PC through the use of modbus TCP/IP protocol. We wanted to be able to read and write to the PLC, and therefore we needed to use the holding registers of modbus which are 16 bit registers, located on the address space 40001-49999, and addresses %MWx in ABB automation builder. These registers were chosen since they are able to be read from and written to. We created a global variables list with the variables to use as communication between the PLC and LAB-PC. We had problems using every %MW address, and solved this by using every fourth address. The problem occured since we were sending to many bits to the registers.

```
VAR_GLOBAL
    //Input
    scenario_MB AT %MB0: WORD;
    e_stop_MB AT %MB4: WORD;

    start_MB AT %MB8: WORD;
    reset_MB AT %MB12: WORD;
    speed_MB AT %MB16: WORD;

    schedulStart_MB AT %MB20: WORD;
    schedulTime_MB AT %MB24: WORD;

    objFound_MB AT %MB28: WORD;
    objPosition_MB AT %MB32: WORD;

    //Output
    position_MB AT %MB40: WORD;
    recordScenario1_MB AT %MB44: WORD;
    recordScenario2_MB AT %MB48: WORD;

END_VAR
```

Figure 25: Addresses used for modbus communication on PLC

We started programming the PLC by making FB for tasks that would be used as communication between the PLC and LAB-PC. The holding registers addresses on automation builder are of the integer data type WORD, and we wished to use this as a boolean value for communication from the LAB-PC to the PLC. To be able to do this we created a function block called “wordBool" in ST.

```
IF wordInput > 0 THEN
    boolOutput := TRUE;
    ELSIF wordInput = 0 THEN
        boolOutput := FALSE;
    END_IF
```

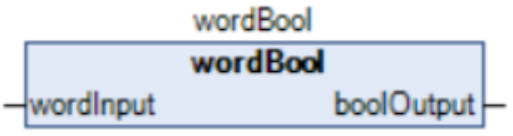


Figure 26: ST and FB of wordBool

The next FB we created was the opposite operation, converting boolean value to data type WORD. This was needed for communication from the PLC to the LAB-PC. We designed this to be used as a boolean value where zero is false and one is true.

```
IF boolInput = TRUE THEN
    wordOutput := 1;
ELSE
    wordOutput := 0;
END_IF
```

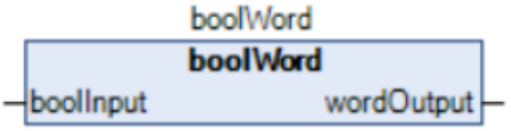


Figure 27: ST and FB of boolWord

In scenario 1 and 2 we were going to receive the time interval in minutes from the LAB-PC, and this needed to be converted to the TIME format, t#(minutes)m. There was no conversion built in from WORD to TIME, so we created a FB "wordTime" in ST converting WORD to string, and then manipulating the string with the built in concatenate function, CONCAT(STR1, STR2), and outputting the interval in time format.

```
intervalString := WORD_TO_STRING(intervalWord);
intervalString := CONCAT(CONCAT('t#', intervalString),'m');
intervalTime := STRING_TO_TIME(intervalString);
```

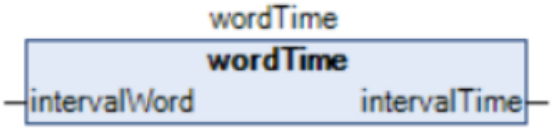


Figure 28: ST and FB of wordTime

These three FB's concluded the FB's needed for modbus communication between the PLC and LAB-PC

Our next action was creating a FB "recallSlide" for recalling the slide back to the start position(left), as this position would be used as the home between each scenario. The encoder data from the motor drive was needed to get the position of the slide, and decide whether the slide should move or not. We added a 50 mm margin for error at each end, as the slide needed space to decelerate to not hit the end of the rail. The encoder data was received in metres, and converted to millimetres for easier interpretation. Then the encoder data was converted from data type UDINT to WORD and placed on the holding register, since the position of the slide would be used continuously on the LAB-PC. This FB would be activated by a reset input variable that would reset its own value after returning home, to exit the recall program we would later design.

```
modbusVariables.position_MB := UDINT_TO_WORD(encoder_data / 1000);

IF modbusVariables.e_stop_MB = 1 THEN
    move_right := FALSE;
    move_left := FALSE;
ELSIF reset = TRUE AND modbusVariables.position_MB > 50 THEN
    move_right := FALSE;
    move_left := TRUE;
ELSIF modbusVariables.position_MB < 50 THEN
    move_right := FALSE;
    move_left := FALSE;
    modbusVariables.reset_MB := 0;
END_IF
```

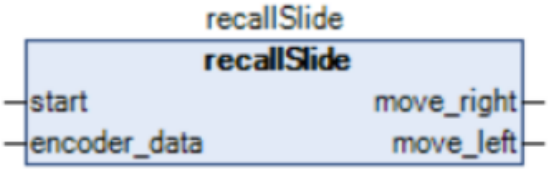


Figure 29: ST and FB of recallSlide

Scenario 1 and 2 would need a clock that would output a pulse at a given interval. We decided to design the FB timeInterval with built-in FB's and logic gates in a function block diagram(FBD). The design has two inputs, timeStart and intervalWord, and output intervalPulse. The variable timeStart activates the clock until the variable is set to false. The input variable intervalWord would be the interval received in data type WORD from the LAB-PC. This signal goes into the FB wordTime, which converts it into data type TIME and sets the delay time(PT) on the timer on FB(TON). TON's output turns TRUE if IN is TRUE and PT has passed. Q then resets the IN signal with an RS-latch, causing the Q to turn off and the process starts over.

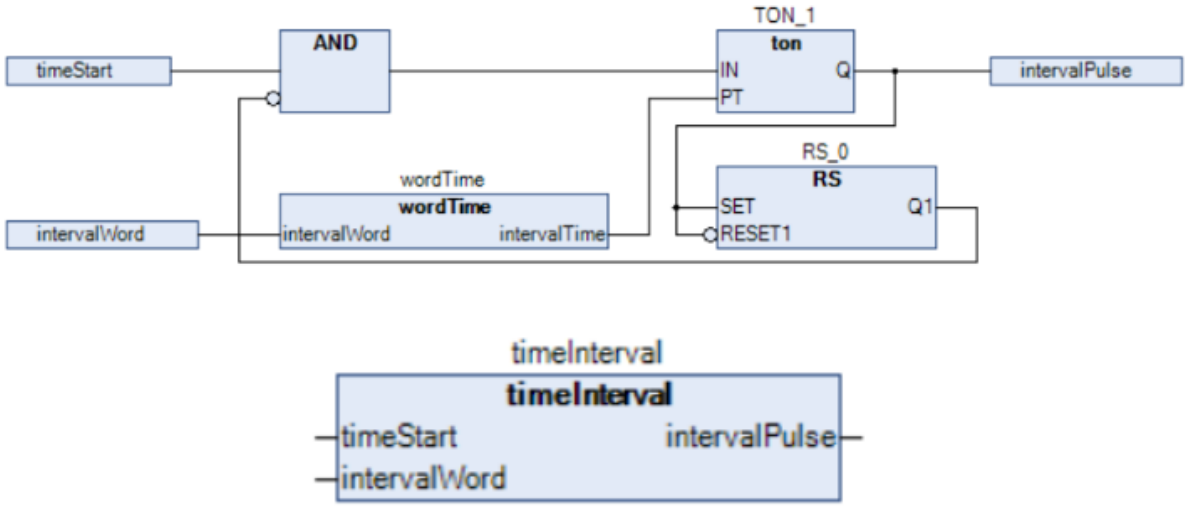


Figure 30: FBD and FB of timeInterval

The next FB startMotor was designed to move the slide in scenario 1 and 2. We added 50 mm as a margin for collision like we did in the FB recallSlide. This FB has one input "Start" which activates FB, and outputs the desired direction of movement based on the position.

```
modbusVariables.position_MB := UDINT_TO_WORD(encoder_data/1000);

IF start THEN
    IF modbusVariables.position_MB > 1950 THEN
        move_right := FALSE;
        move_left := TRUE;

    ELSIF modbusVariables.position_MB < 50 THEN
        move_right := TRUE;
        move_left := FALSE;
    END_IF
END_IF
```

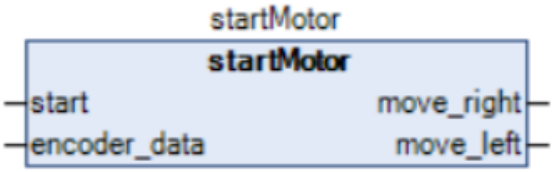


Figure 31: ST and FB of startMotor

The last FB we created was for scenario 3. The FB receives a desired position(objPosition_MB) as input, and calculates a position interval ±25mm. Then the FB checks whether the slide is in between said interval, smaller, or higher, and decides movement direction based on the answer. This FB is activated by a start input, and receives the desired position and encoder data through inputs. To be able to compare the different data types we had to convert objPosition and encoder data to REAL.

```
modbusVariables.position_MB := UDINT_TO_WORD(encoder_data/1000);

position := UDINT_TO_REAL(encoder_data/1000);
objPosition := WORD_TO_REAL(modbusVariables.objPosition_MB);
positionLower := objPosition - 25;
positionUpper := objPosition + 25;

IF start THEN
    IF positionLower < position AND position < positionUpper THEN
        move_right := FALSE;
        move_left := FALSE;
    ELSIF modbusVariables.position_MB < modbusVariables.objPosition_MB THEN
        move_right := TRUE;
        move_left := FALSE;
    ELSIF modbusVariables.position_MB > modbusVariables.objPosition_MB THEN
        move_right := FALSE;
        move_left := TRUE;
    END_IF
END_IF
```

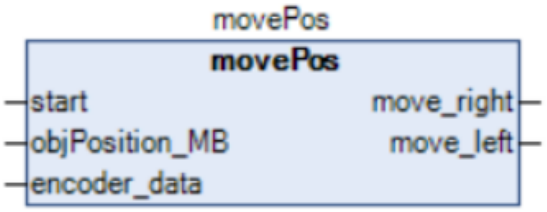


Figure 32: ST and FB of movePos

With the FB's we needed made, and the built-in FB's and logic gates we built our scenarios in a FBD. Scenario 1 and 2 is the same on the PLC, and scenario 3 has a FBD designed for its purpose. In the first two scenarios it was essential to open the camera feed and start recording when the patrol mode was activ, and close when it was done. This was solved through a series of latches connected to a modbus holding register.

The PLC is designed to work with a GUI running on the LAB-PC controlling the flow of the program through the modbus variables start_MB, reset_MB and scenario_MB.

```
IF modbusVariables.reset_MB = 1 THEN
    modbusVariables.recordScenario1_MB := 0;
    modbusVariables.recordScenario2_MB := 0;
    recall();
ELSIF modbusVariables.start_MB  = 1 THEN
    IF modbusVariables.scenario_MB = 1 THEN
        scenario_1();
    ELSIF modbusVariables.scenario_MB = 2  THEN
        scenario_2();
    ELSIF modbusVariables.scenario_MB = 3 THEN
        scenario_3();
    END_IF
END_IF
```

Figure 33: Flow control of scenarios

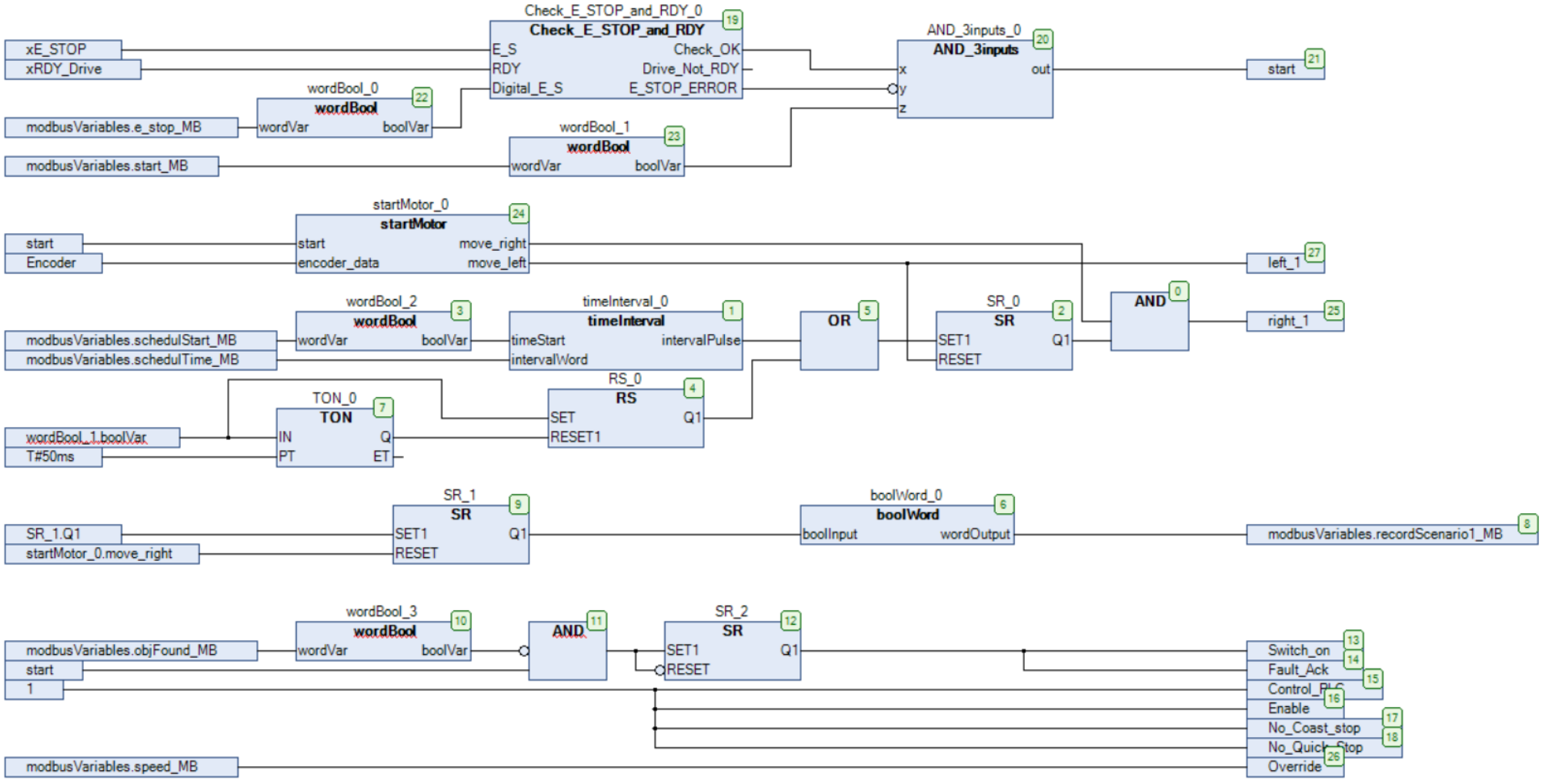


Figure 34: FBD of scenario 1 and 2.
SR_1.Q1 should be SR_0.Q1

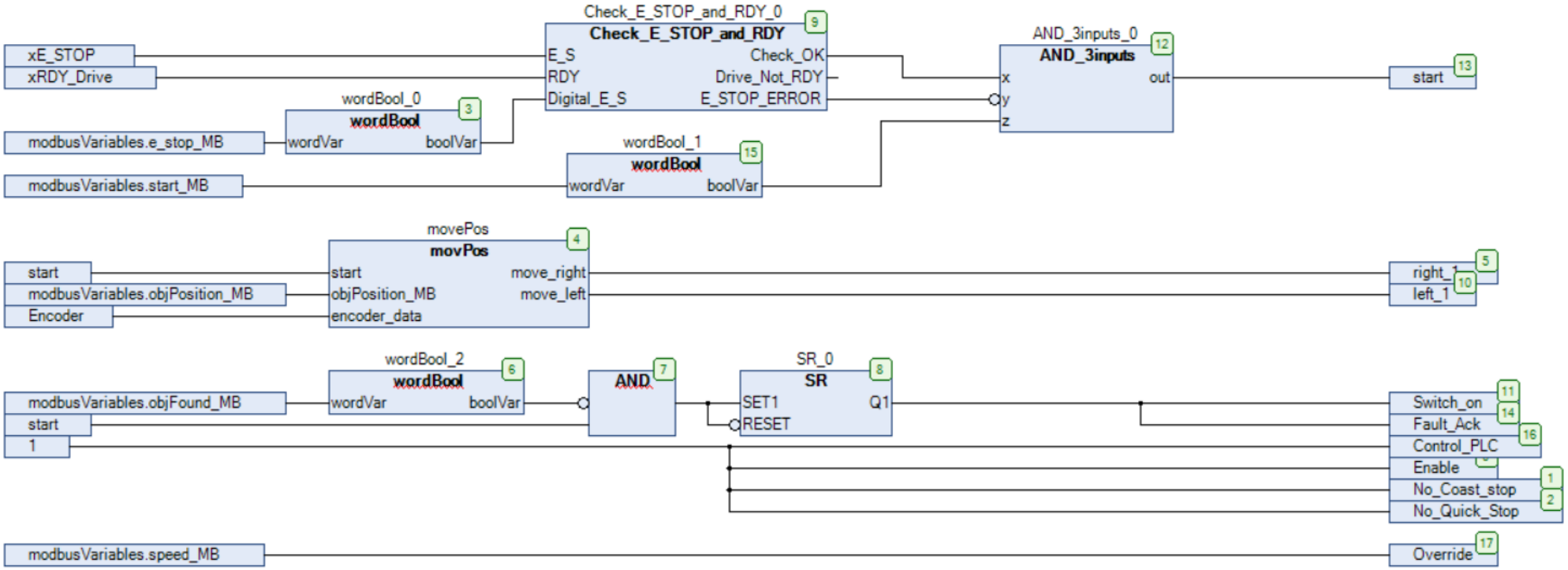


Figure 35: FBD of scenario 3

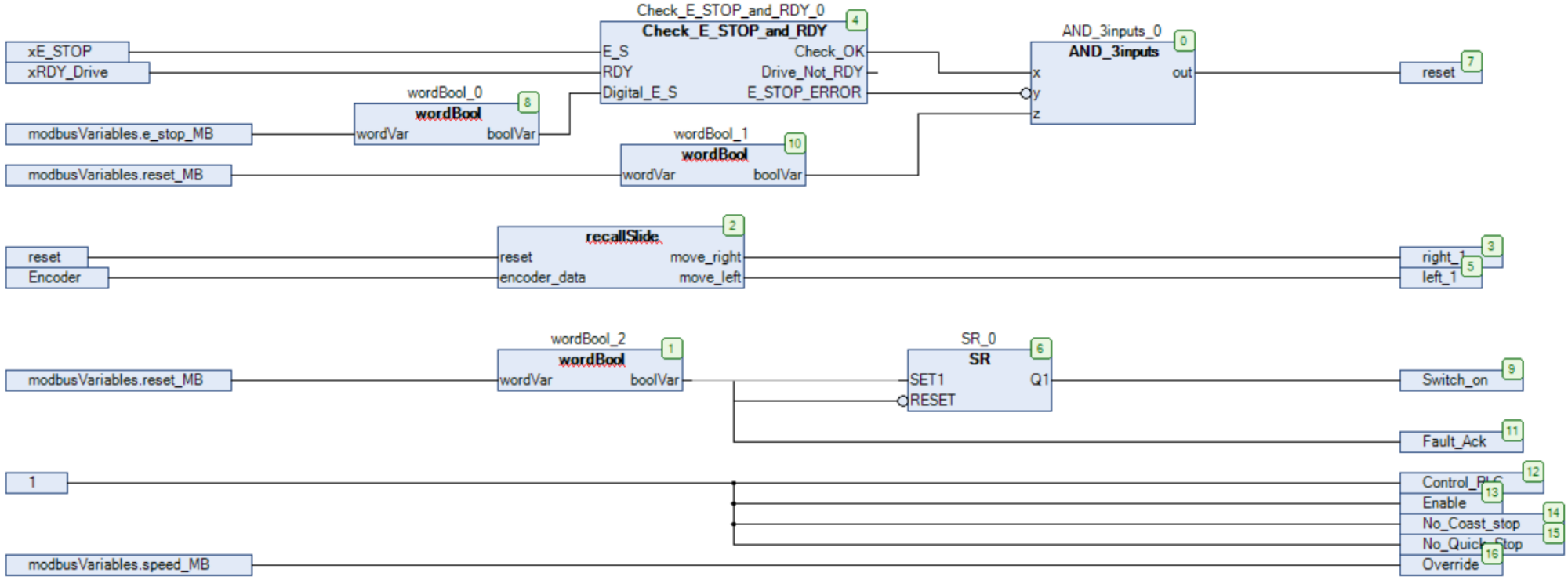


Figure 36: FBD of recall program

3.4 Programming

3.4.1 Pymodbus

We decided to use the Python library “pymodbus” for communication between the LAB-PC and PLC. The ethernet cables we needed were wired by the previous bachelor group, so all we needed to do was connect through the library function with the LAB-PC configured as a client(master). We needed to be able to read and write to the holding register.

- from pymodbus.client.sync import ModbusTcpClient #Import client class
- client = ModbusTcpClient('IPaddress') #Connect to PLC through IP address
- client.write_registers(address,[value,0]) #Write to register
- client.read_holding_registers(address).registers #Read from register

3.4.2 AI model

To implement the AI model in Python, we first tried to use the function “detector()” from YOLOv5, but experienced low frame per second(FPS) and a significant delay on the camera feed. We decided to customise a detector function based on Nikolai Høirup Nielsen’s (niconielsen32, n.d.) already existing solution. Our module class is included in attachment x with detailed comments.

3.4.3 Cloud drive

One of the project's main tasks is being able to record and take pictures while moving for scenario 1 and 2. At the end of the scenarios it is going to automatically send video and picture files to a cloud service. In our case we are using Azure Storage and OneDrive for our cloud service. In the code we are only uploading it to Azure but with help of Microsoft Power Automate we are also copying the files over to OneDrive. This can also be done to other cloud services such as DropBox or Google Drive as well. The group decided to go with Azure Storage for our main cloud service because it is more adapted for the future and lays a better foundation for development.

3.4.4 Graphical user interface

The graphical user interface (GUI) is an interface the user can interact with to manage components and values through electrical signals. The interface uses buttons mapped to inputs and outputs connected to corresponding functions for the user to control. This allows the user to interact with the system without prior knowledge of how the code or system functions are set up. We created the GUI using Python’s integrated library, Tkinter. This library is already built into the Python standard library.

The decision to use Tkinter over other libraries such as PyQt5 or Kivy was made because Tkinter was already installed in the standard Python framework and did not need any additional packages to be installed. Tkinter has been used for many years and has a lot of documentation and examples. Since Tkinter is an old library it also follows with older fonts and aesthetics, unlike PyQt5 which is used to build a more modern GUI. Tkinter can also be used to make modern-like GUI, but requires more effort. The group chose to go for Tkinter because it had all the functionality needed for the system. The GUI was designed to be intuitive and easy to use. It covers all the functionality of the system without it being overwhelming.

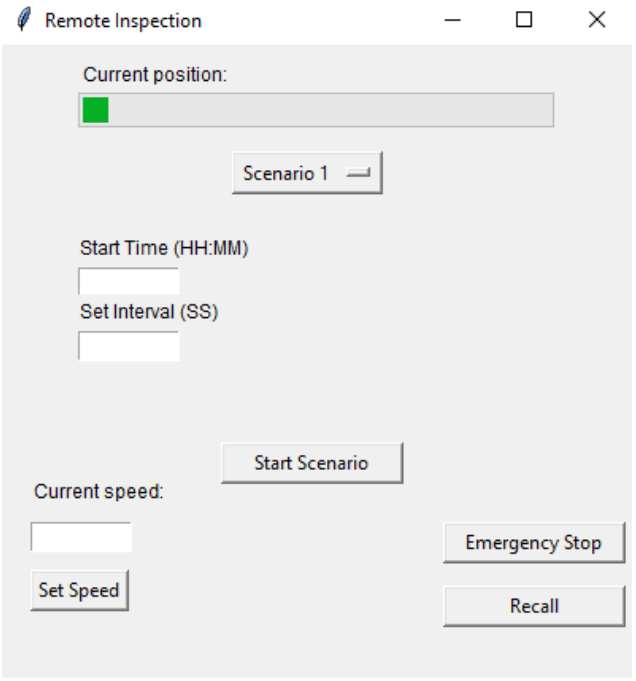


Figure 37: The GUI in Scenario 1 which was made with Tkinter

Tkinter works by making a main loop and waits for buttons to be pressed or action to be made. However this can be a problem if we were to implement the Image Recognition part of the project. Since both of these parts are a loop, only one of them can run at the same time. Problems such as not being able to interact with the GUI while the camera is running can occur.

In Figure 51 below, it is illustrated how the GUI works with the individual scenarios. The main loop which is the GUI, is used to initialise the different scenarios. Once a scenario is initialised the GUI will go into that loop and stop the GUI from being interactable. Once a scenario is rested the GUI will start working again, and the user can start another scenario.

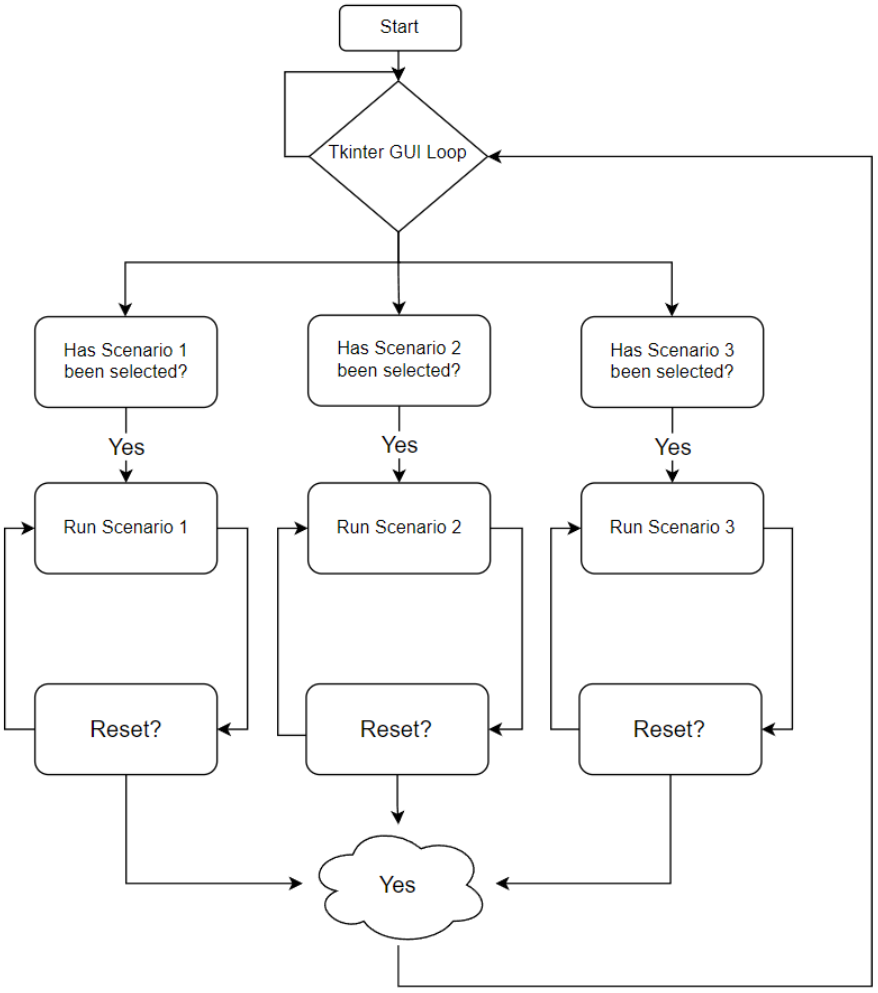


Figure 38: Flowchart of GUI loop

3.5 Camera and mount

As the group was not able to obtain a second camera due to unavailability in the suppliers of ABB, the implementation for the system only uses a single camera solution, but will address better alterations for future work in the discussion section. For demonstrational purposes this does not pose an issue, as our goals can still be completed with a single camera.

The decision to rotate the camera back to an upright position was implemented as this gives a larger horizontal field of view. The system without alterations and the camera mounted upright would then provide a stationary 110° by 61° field of view solution travelling along the rail, as shown below in illustration 39 and 40.

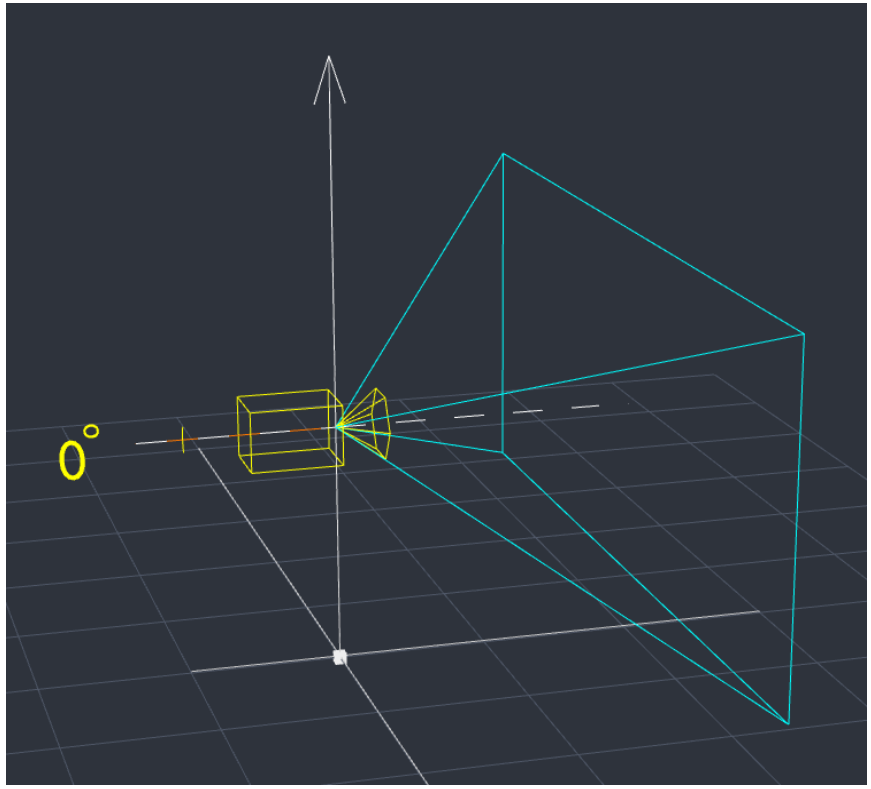


Figure 39: Camera mounting solution 1

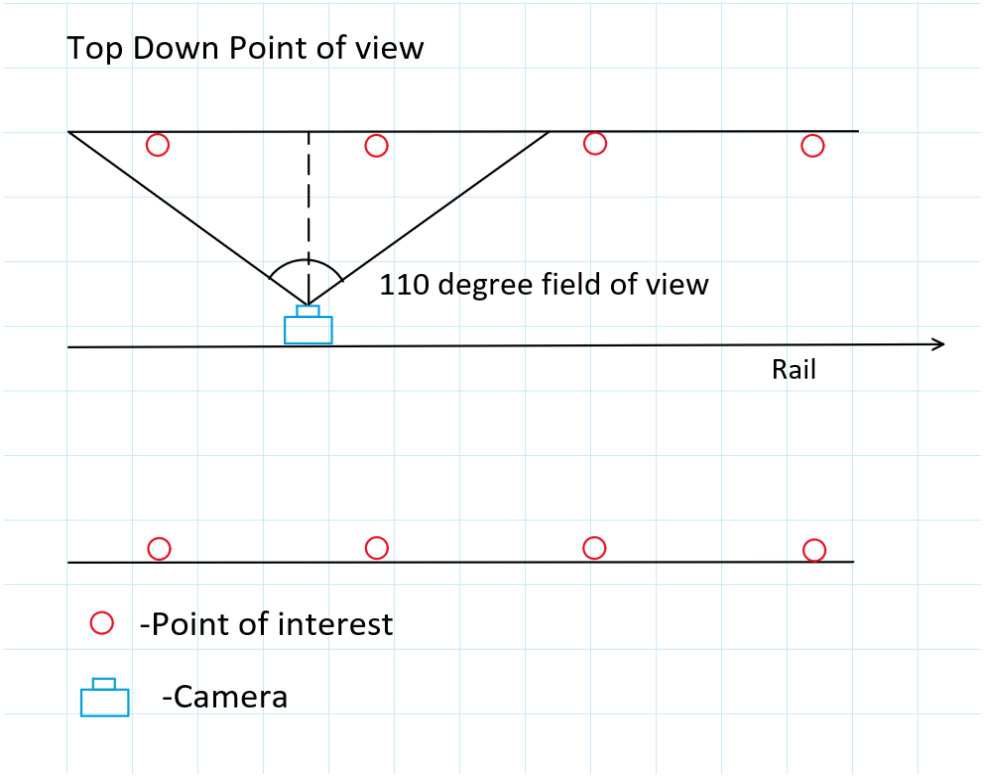


Figure 40: Top down camera point of view

The 2022 group was tasked by ABB to produce a solution that can be flexible to different mounting angles. To allow multiple angles a camera housing unit was 3D printed with corresponding tilt values. This housing matches the measurements of the AXIS camera and is fastened with a single M6 bolt through the back of the casing. Below the allocated attachment point there is a hole for the PoE cable.

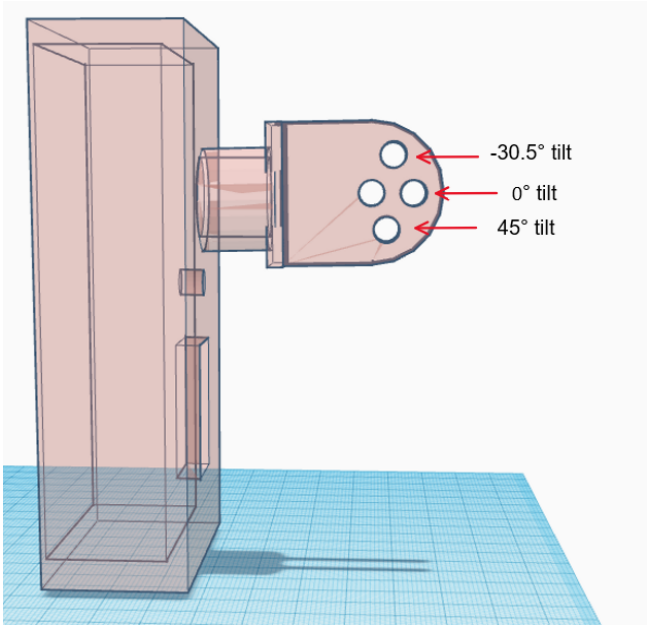


Figure 41: Camera mount side view

Figure 42: M6 Screws

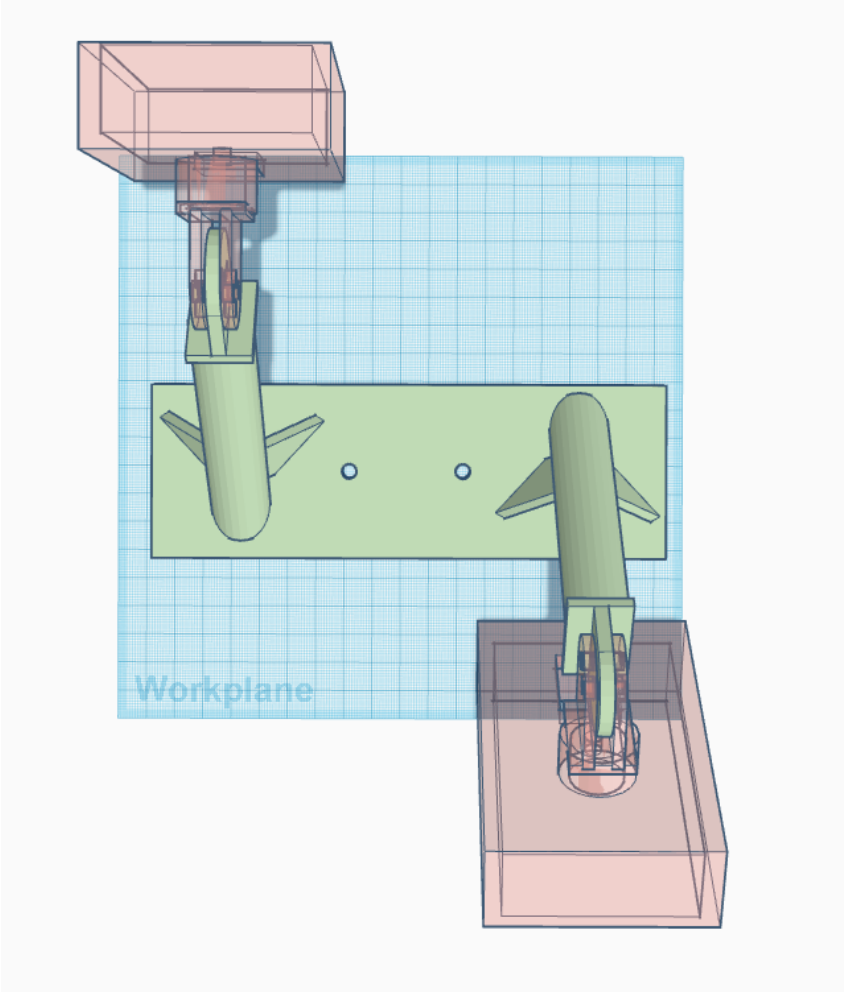


Figure 43: Top down view of camera mount

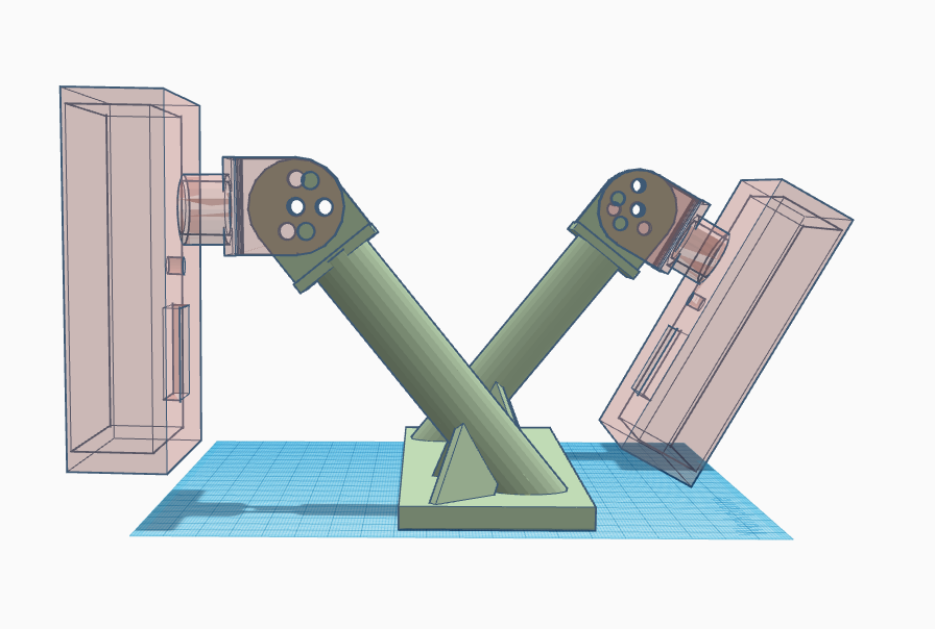


Figure 44: Camera mount side view

Figure 45: Final mount solution

The mount was modelled in TinkerCAD, a Computer Aided Design program aimed at providing a simpler version of AutoCAD, while still maintaining functionality for producing easy designs. The 3D print model was then imported to Cura, where it can be sliced and loaded for print with the Ultimaker 2+ machine. It was printed in PLA plastic filament at 0.1 layer height to give a stable and robust mounting platform with an adjustable camera housing unit. The camera housing is held in place by 2 M6 bolts and secured with washers and wing nuts. The final mounting solution provides the option of mounting two cameras in opposite directions with adjustable angles for a -30.5°, 0° and 45° tilt.

OSLOMET

Results

4 Results

This part of the thesis will state the results of the project work.

4.1 Model

The first versions of the datasets contained almost exclusively images from the same room at OsloMet campus. The images were also taken at roughly the same time of day with similar lighting. This led to the model being very overfitted to images from that specific room, and it was not performing well on images from different locations.

The worst performing datasets contained images taken at short distances. Many of these images were also blurry and of low quality. The solution to this was taking images from different rooms at campus as well as images from ABB's old office at Ole Deviks vei and the new office at Fornebu. The images were taken at different times of day and with different lighting. The performance of the model improved drastically when adding several other images taken at different angles and from different distances, and removing blurry and low quality ones.

The group spent a lot of time training models with different datasets and tweaking epochs and batch_size to find the best performing model. After many rounds of testing it was concluded that the model with images from OsloMet campus and ABB's old building at Ole Deviks vei, as well as the images taken at Fornebu, trained with epochs=75 and batch size=16 yielded the best results.

Figure 44 shows the same image as in figure 29 in 3.2.7 after the best performing model has been applied to it. The number after the name of the class represents the confidence level on a scale from 0 to 1, where 1 is 100%. As we can see, the model works very well and it finds all the classes with a high confidence level.
One thing to note is that the same class is used on two of the instruments because they are identical. The only difference is the harvester attached to one of them.

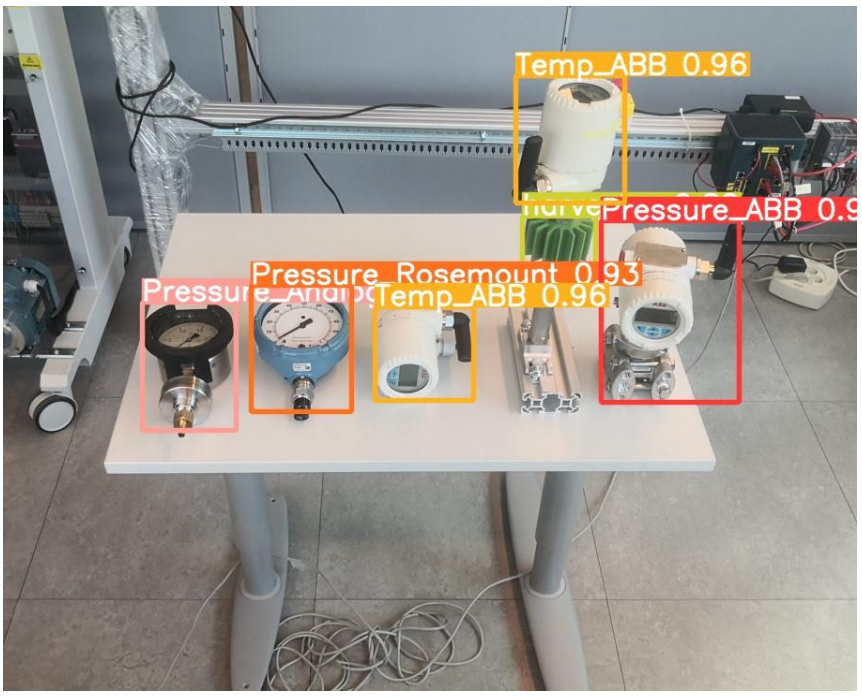


Figure 46: The same image as in figure 19 in 3.2.1 after the model has been applied to it

Figure 47 shows the results from training the model with 50, 75 and 100 epochs and batch size set to 16. As we can see, the mAP improves swiftly until 20 to 30 epochs, and then improves slightly until plateauing after around 50 epochs. 100 epochs didn't improve the results much in terms of mAP.

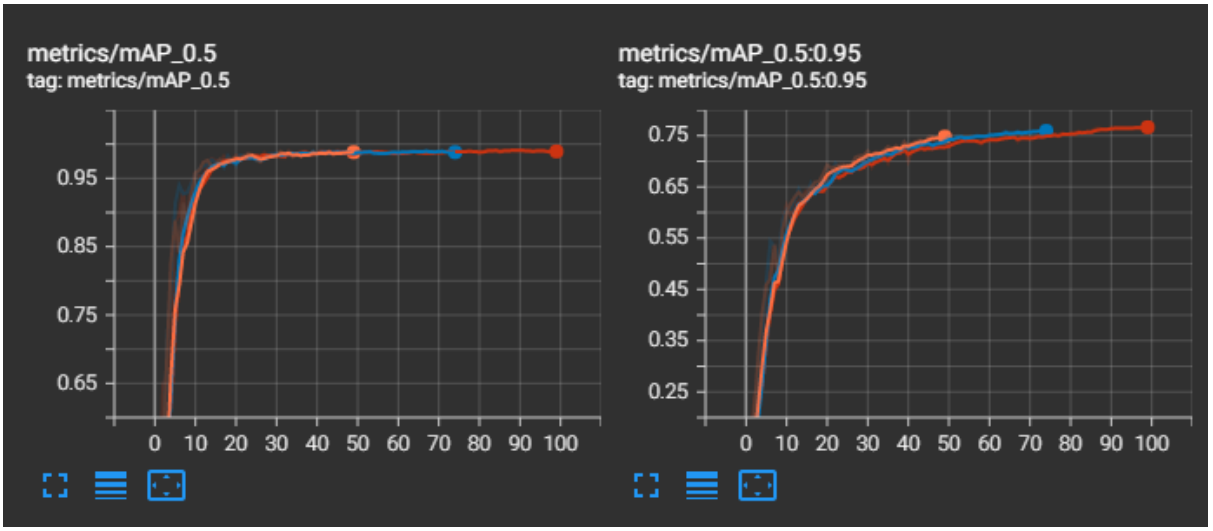


Figure 47: Graphs showing mAP (mean average precision) for different epochs. The x-axis shows the number of epochs

The three models trained with 50 epochs, 75 epochs and 100 epochs all perform pretty similarly when it comes to predictions, as shown in figure 46 below. These are all models with the latest dataset.

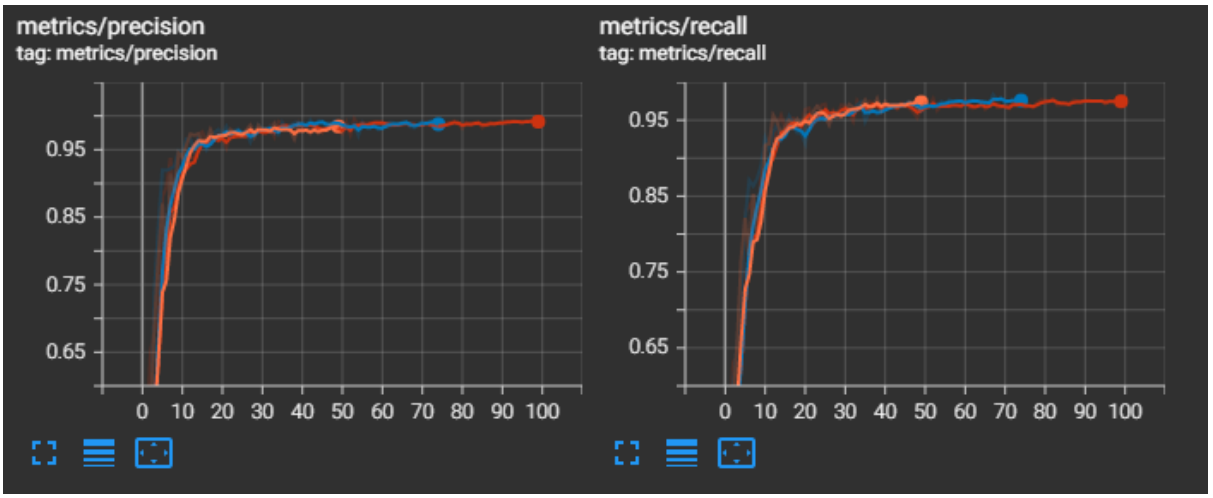


Figure 48: Graphs showing precision and recall for different epochs

As shown on the graphs in figure 49, the models that were trained with more epochs perform better, but more epochs often means higher risk of overfitting, so the models had to be tested in practice.

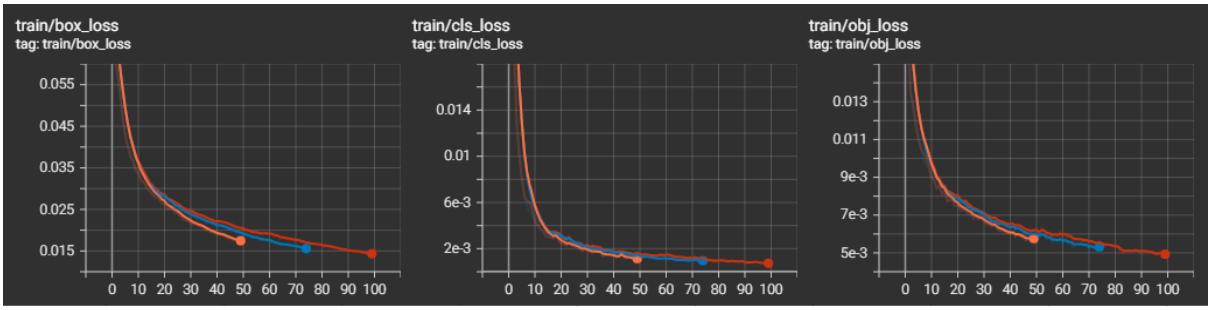


Figure 49: Graphs showing loss functions for different epochs

Figure 50 shows the comparison between two datasets. The blue graph represents the dataset used in the current model and the orange graph represents one of the earlier datasets. The model with the latest dataset reaches a good mAP early and maintains this, while the model with the early version of the dataset struggles to reach good mAP.

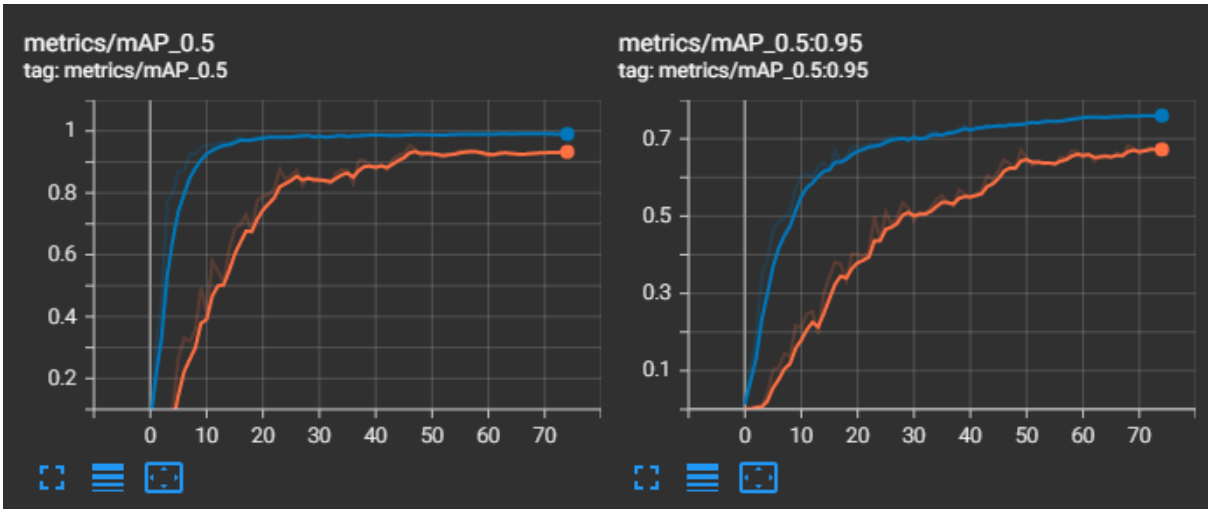


Figure 50: Graphs show mAP of two datasets. The blue graph represents the latest dataset and the orange graph represents one of the earlier datasets

Looking at the precision graph in figure 51, it is also obvious that the model with the latest dataset is superior to the one with the old dataset when it comes to correct prediction. The recall graph in the same figure shows that the model with the latest dataset is also way better at predicting true positives and disregarding false positives.

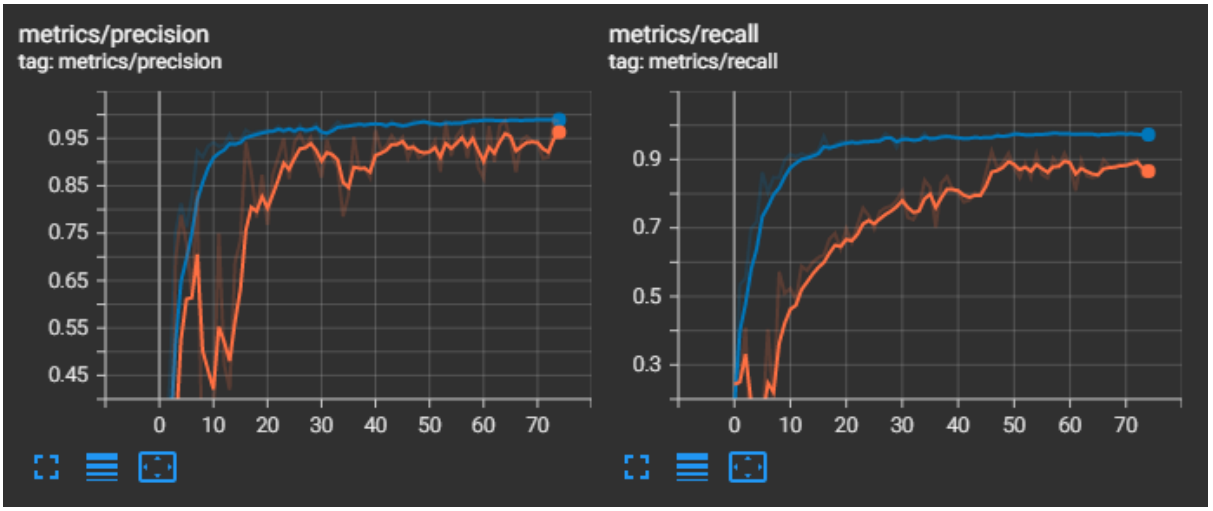


Figure 51: Graphs show precision and recall for two datasets. The blue graph represents the latest dataset and the orange graph represents one of the earlier datasets

Figure 52 shows the comparison between the two datasets when it comes to the model's ability to find the centre of the object (box_loss), how well it's predicting the correct class (cls_loss) and how good of a prediction it makes when it comes to the probability of there being an object present. Looking at the cls_loss graph, we see that the older dataset actually performs a bit better than the newer dataset after about 60 epochs when it comes to predicting the class. The reason for that is that the older dataset only contains three classes while the newer one contains five classes, three of which can look similar from some angles.

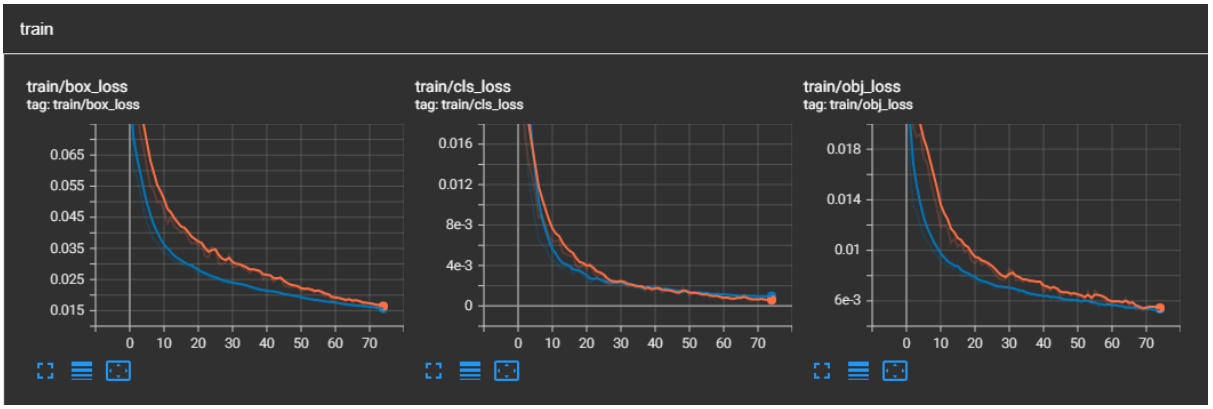


Figure 52: Graphs show loss functions for two datasets. The blue graph represents the latest dataset and the orange graph represents one of the earlier datasets

4.2 PLC and GUI

The hardware portion of the PLC and system was already installed in the previous bachelor group and worked as intended. There was one proximity switch that was defective, but was not needed for our project as positions were not hardcoded. This was therefore not replaced. In the last week of the project we got an error with the communication between the PLC and motor drive over Profinet, but the error was rare. We were not able to identify the problem, but it is believed that there is a faulty wire.

Orally we were given the task of setting the speed of the slide to 1 m/s, up from 0.4 m/s. The hindrance was the deceleration of the slide, but the issue was addressed by implementing a 50 mm margin for the declaration length at each end of the rail.

Each scenario connects and opens the camera feed. During the previous bachelor thesis there was a problem with a delay on the feed ranging from 5 seconds to 5 minutes. We also experienced a delay, but were able to shorten it to constantly approximately 1 second. This was done by clearing the buffer of frames received from the camera feed after processing each frame, and implementing GPU accelerated processing. The delay is within an acceptable range, but should be improved.

We were tasked by ABB with building three different scenarios. The first two scenarios are identical on the PLC, but with the AI model running on scenario 2. The third scenario being storing and moving to chosen object position, and all scenarios should be triggered from a GUI.

The first two scenarios were essentially the same, but with and without the AI model running, and some different variables were used. Both scenarios would start from the GUI at a given time and interval. We tested this portion of the scenario by letting it run over several hours while we were at ABB without missing a patrol cycle. The scenarios also opened the camera feed and saved the recording for scenario 1 image for scenario 2, and uploaded it to Cloud Storage. The AI model worked with good accuracy and recall when the slide was moving in scenario 2, even with different lighting. Scenario 2 also saved the rail position and label in a text file for use in scenario 3. If an already found object were detected the position and label were not saved multiple times.

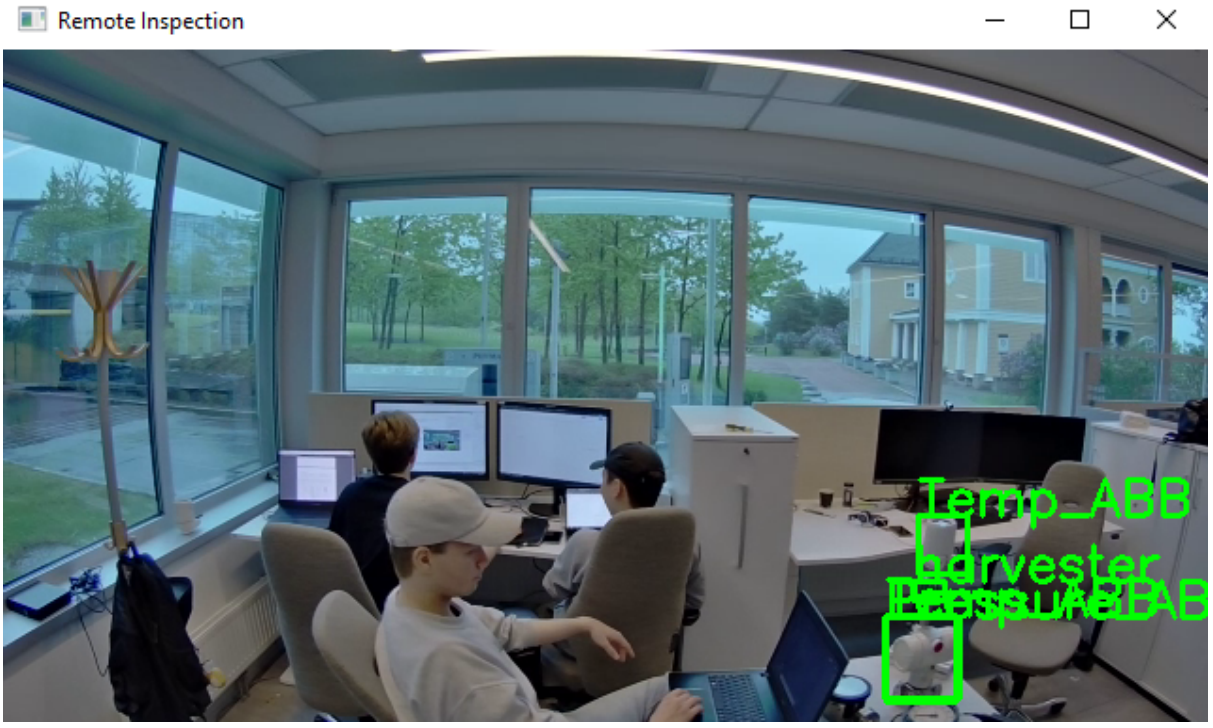


Figure 53: Still image from scenario 2 with AI model

Scenario 3 moved to the desired location, and opened the camera to inspect the object. The list of rail positions and object labels received from scenario 2 were updated and functional.

The GUI updated in real-time with data from the PLC, and sent boolean variables instantly to PLC. We never lost data between the PLC and GUI. It had buttons for each scenario, and was responsive and intuitive to use. There was a small delay when accessing the camera feed in each scenario, but it was negligible.

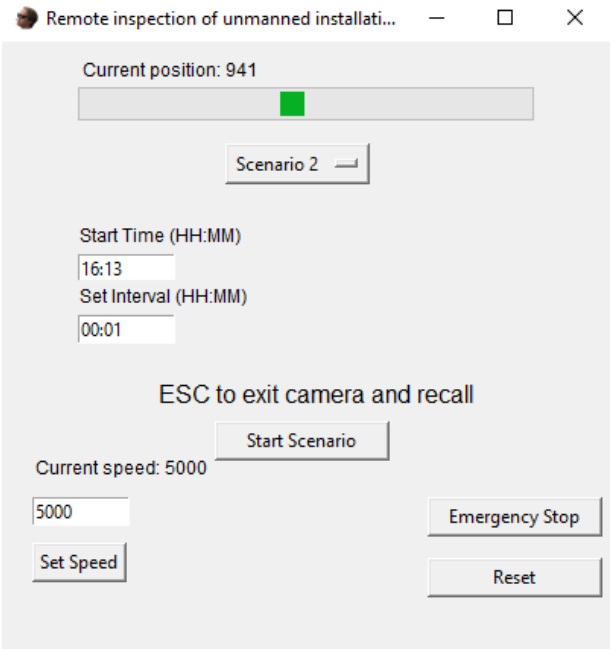


Figure 54: GUI

Unfortunately we were not able to implement the optional scenario 4 due to time limitations, but have made a foundation that could be continued on. Overall the result was successful, as the system worked according to the requirements we received.

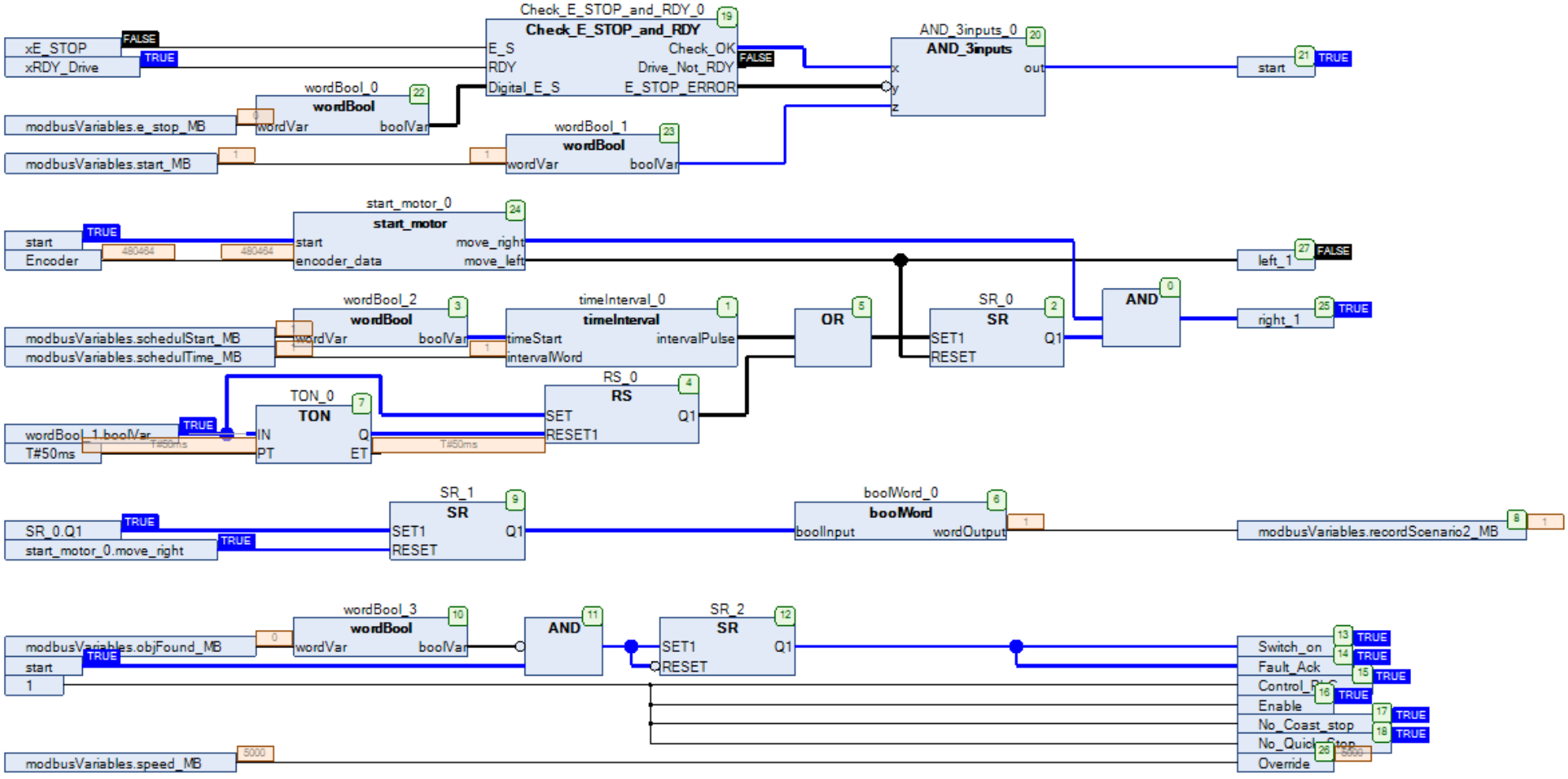


Figure 55: Scenario 2 running in PLC

4.3 Cloud Storage

One of our main objectives was to upload files to a cloud storage service. The recording from scenario 1 and 2 got uploaded to azure, and after a brief delay got uploaded to OneDrive. During testing there were no pictures that did not get uploaded to OneDrive. Before uploading the pictures and videos got named in the format yyyy-dd-month-hhmm. Azure Storage and OneDrive have good compatibility, and it proved easy to copy files from one to the other through Microsoft Power Automate.

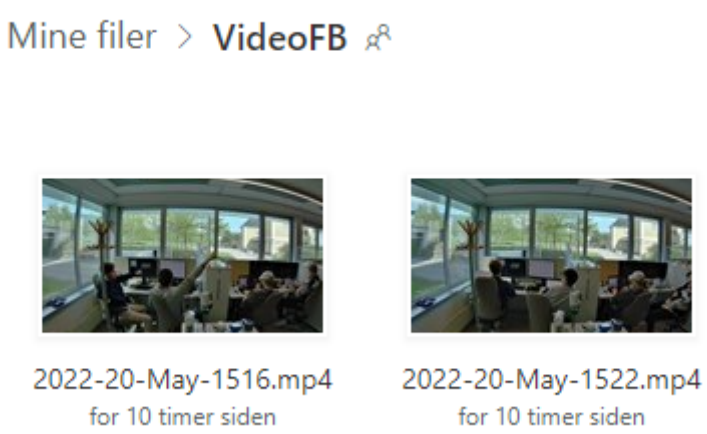


Figure 56: Videos in OneDrive

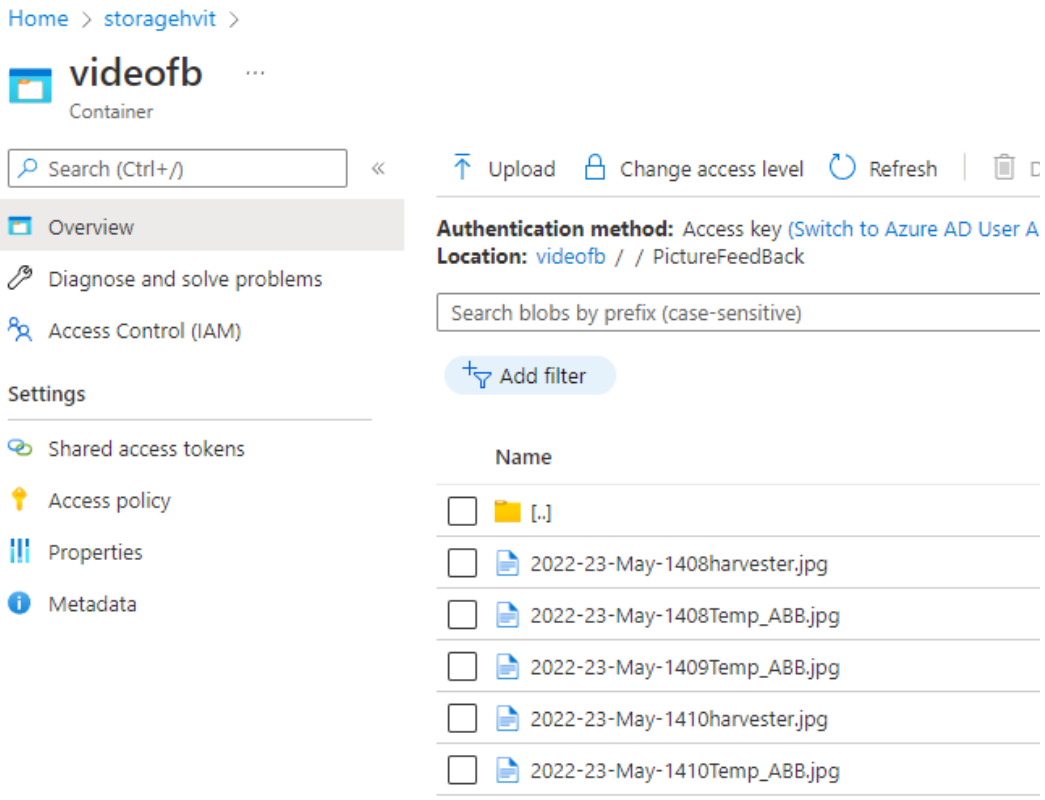


Figure 57: Pictures in Azure Storage

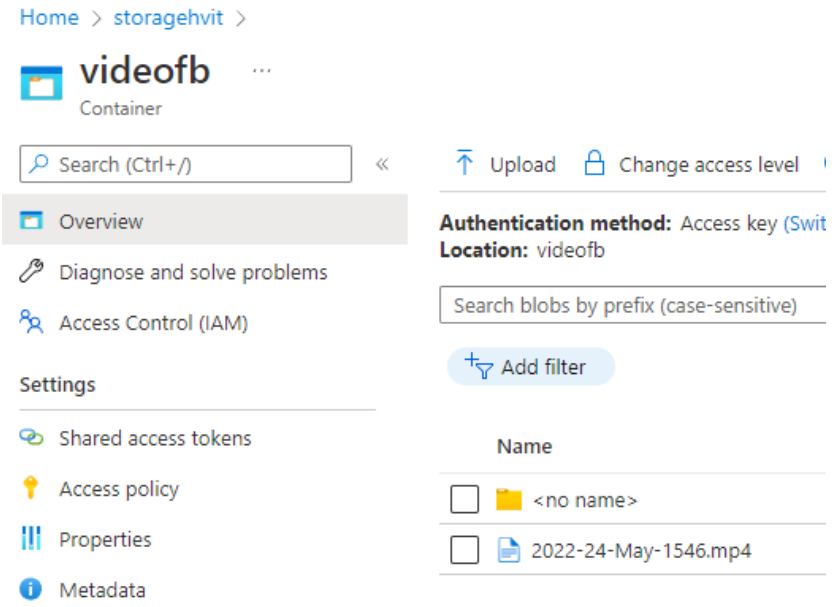


Figure 58: Video in Azure Storage

4.4 Camera mount and angles

The mount provides an adequate solution to provide the desired stability and mobility, while remaining easy to install and with easy change of angular setup. The plastic from the 3D print remains somewhat fragile and would not be suitable for a finished product, but is sufficient for demonstrating a prototype at three different angles for all suggested mounting solutions.

OSLOMET

Discussion

5 Discussion

In this chapter we will discuss potential improvements to the system and ideas for future work.

5.1 Model

The model works excellent in the current lab setup and works pretty good on images taken from different environments. This is a result of the majority of images in the dataset being taken from that same lab.

Adding more data to the dataset from other places, especially realistic work environments, would likely greatly improve the overall performance.

As mentioned earlier in the report, the LAB-PC struggled to run the larger YOLOv5 model architectures. This is because of outdated components in the computer. A more powerful computer with a GPU more suited for handling this kind of task would be able to run a much more accurate model.

Another thing to mention is training the model to look for specific properties on the instruments, such as wireless antennas, brand logos et cetera. Having a model that manages to categorise instruments based on their respective characteristics may prove valuable.

5.2 PLC and programming

Even if the system is functional, there are some changes the group would like to do. We wanted to implement this sooner but due to the time limit we only got to research it. Some of the main changes the group wanted to do was threading and delay.

Python threading is a way to have different parts of the program run simultaneously. Which means that the GUI could run in the background even if the camera was open since both are a continuous loop. Threading was a way to workaround this problem and simplify the code.

Another part of the code we wanted to optimise is Image Recognition and delay. As mentioned already in the results, we got a delay due to hardware limitation and most likely over-processing in the code. We wanted to implement a better way to buffer the images and minimise the delay even more than 1 second that we have right now. From researching the problem we discovered that this was a problem many people had encountered with openCV and IP cameras, but we were not able to reduce the delay to where it was negligible. This delay will become a limitation if the slide were to move faster, and therefore be solved in the future.

5.3 Cloud Storage

Some of the concern regarding Cloud storage is storage, after using the program for a while the storage will fill up and more space is needed. One thing to note is that if we need more space on cloud services, we either need to pay a subscription fee or pay for additional storage space.

Currently the system is planned to be mounted off-shore, this can be a complication for the cloud storage part of the code. In the code we are directly sending both the video and picture files straight to the cloud without storing it locally on the LAB-PC. Without access to the internet the files are getting deleted right away without any way to retrieve them. A solution to this is to store it locally until a network connection is established, and then deleting the files after the files are uploaded to the cloud.

5.4 Camera mount and angles

When the next stage of development begins the system will ideally be ceiling mounted, so the group has made suggestions for a ceiling mounted system. It will always maintain a full view of the instrument wall if the length:height ratio of 2.8:5 is maintained, and the camera has a negative tilt of 30.5°. For the AXIS camera this is the ideal ratio to achieve the highest possible PPM, while still maintaining a zero vertical loss in field of view.

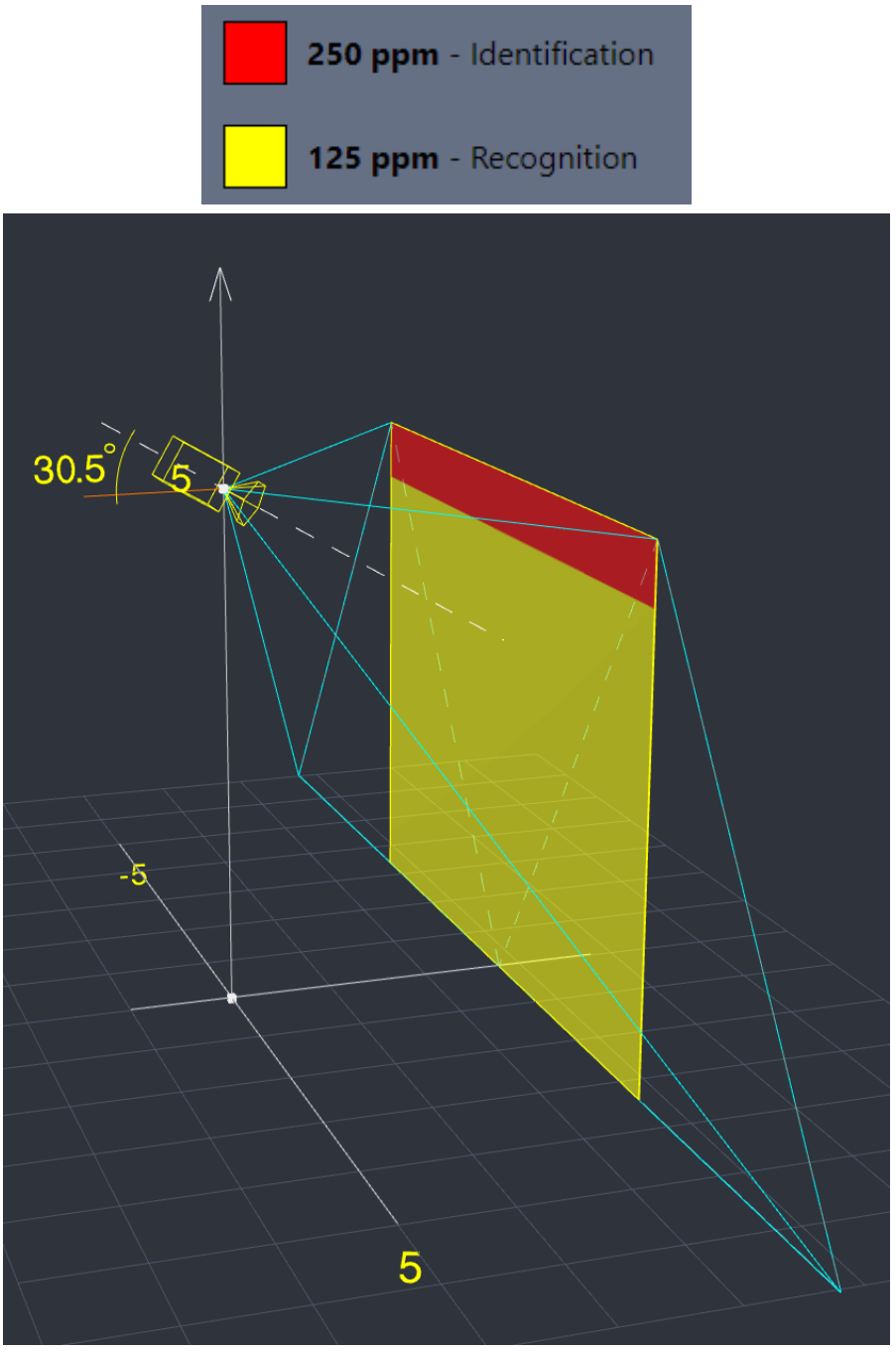


Figure 59: Ideal tilt with classification zones (Lens Calculator, n.d.)

An in-depth study for alternative camera and mounting solutions can be found in 7.2 Attachment 1: Camera and mount. For a more optimised system a camera with a higher resolution would provide the

system with a higher PPM count. An Increased resolution would also require more processing power, but would make recognition at further distances possible without the drop in field of view.

5.5 Future research and development

5.1.1 Optional scenario

Due to time limitations we were not able to complete the last scenario we were given. Scenario 4 was an optional task proposed for the system to track changes within the system instruments through pictures of analog gauges.

Inputs: time and frequency
Output: Images of the differences found uploaded to OneDrive or Dropbox.

Most modern transmitters and gauges are wireless and can send values to a computer, but old analog gauges can not. Installations are typically designed for a lifespan of 30 years, meaning that a lot of analog gauges are still in use. A model trained to read analog gauges is therefore beneficial.

From the other scenarios we already have made essential parts of this scenario, including the PLC program from scenario 1 and 2, the pictures of the images saved in scenario 2, the upload function used in the first two scenarios, and the instruments position used in scenario 3. Due to processing power we were not able to use the camera resolution to full extent, and realised this would prove a major hindrance in spotting differences in instrument differences.

5.1.2 Onboard vs offboard motor solution

The group was tasked with improving the model in all areas. One of these tasks was discussing improvements to the current motor system. As touched on in the motor limitations chapter in 1.3.1.1, the current motor has some limitations when it comes to mobility because the current system features an off board motor. This means that for a bigger rail system the motor would have to be upgraded as well. The approximate power requirement calculations can be found in 7.3 in the appendix.

An onboard motor solution would solve these problems because the onboard motor would only have to power the camera and mount, but this presents some new problems as well. The main thing to consider is how to power such a motor. One solution is a battery, but a battery would need to recharge at some point. This could be done when the camera is in its “home”-position. Another solution would be to have the rail powering the system, but this also causes new challenges. The finalised system should be a system that is deployable in many different installations, and should be able to run outdoors. Rain could cause electrical problems to the rail and potentially damage the system.

These are all things to take into consideration. The group decided not to pursue an onboard solution as the offboard solution does its job fine for now.

5.1.3 Technical specifications for LAB-PC

When first inspecting the lab at ABB, we were supplied with a LAB-PC. This LAB-PC was somewhat outdated. Some of the hardware that limited the systems performance was the graphical processing unit (GPU) and central processing unit (CPU). The PC was equipped with two NVIDIA GTX 980Ti and an Intel i7-5820x. Both of these components are over 6 years old and have older electrical

architecture. This means support for newer and more optimised drivers is not available. We encountered some problems with the old hardware. One of our main issues was getting both of the GPUs working together, and when we got our project working with the GPU instead of the CPU, we realised it was only using one of them.

A rare occurrence with the LAB-PC was memory errors. While troubleshooting and testing we got a blue screen error resulting in some progress loss. We noticed that it only occurred if we were using multiple Python code editors at the same time, and refrained from doing that again.

As mentioned we did not get a hold of another camera, and therefore did not implement multiple video streams, but if this were to be done in the future the GPU should be upgraded.

Conclusion

6 Conclusion

To sum up the project; the group was able to fulfil all the mandatory requirements, and if not for a time constraint would have been capable of completing the optional task. This lays a solid foundation for future work.

The current model works very well and is considered a success. It does a good job at detecting all of the objects, especially when the system is mounted where it is now. A small loss in performance when testing the model on images taken in different areas was noticed, but it still managed to detect all objects with a satisfactory confidence level.

The drawback of this project was the inaccessibility of hardware that ABB was not able to obtain for us to implement into the system. The lack of two cameras to run the system with, meant there wasn't an opportunity to test the system on multiple camera feeds. Adding a secondary camera would add considerable strain to the LAB-PC hardware, which may have resulted in a different solution to account for the extra processing power needed. Integrating a lightsource to ensure a well lit work area was also not implemented and will make it impossible to operate the system in work areas without external lighting sources.

The startup phase of the project included a considerable time invested into research of the system, and how we wanted to develop the system. This made the implantation phase easier without many unexpected problems encountered.

The group has completed a well-structured and well-planned project with evenly distributed workload, good teamwork and management. Making decisions as a group with weekly meetings has helped everyone be part of the process.

Appendix

7 Appendix

7.1 Project Requirements

Batch 2022
Project Checklist:

Table 3: Project requirements

Ref	Description	Project Objective	Status
1	**Action items:** Read the report from the previous students. Conduct a safety review of the setup as-is. ● What safety mechanisms are in place today? ● What needs to be done to improve the safe operations of the test setup? **Deliverable**: ● Proposed new design/functions/procedures to improve overall safe operations of the test setup.	Mandatory	

2	**Action items:** What are the limitations of the existing setup? ● Hardware limitations (e.g., motor) ● Software limitations (AI part) ● Automation limitations (e.g., PLC related) ● Environmental limitations ● Electrical hazards ● Mechanical hazards **Deliverable:** Based on original requirements: ● Can this setup be used to meet the original requirements? If not, why? ● Can this setup be modified to meet the original requirements? What additional components will be required? ● What will be the minimum and maximum time required to conduct a remote inspection if we have the following rail tracks in a loop? ● 250m ● 500m ● 1000m ● 2000m Assuming everything else remains the same and we need to process the images as the rail cart moves on the track.	Mandatory	

3	**Action items:** Today, we have only one camera on the test setup with one field of view. How can we increase the field of view? ● Target 180deg view on both sides **Deliverable:** ● New camera or additional camera? ● How to mount? ● How can we integrate video feeds on same timescale? ● How will this be handled in image processing.	Mandatory	
4	**Action items:** ● How many cameras are needed for the target design - See Figure1? ● Which camera model is required? ● Construct the installation mount for the camera.	Mandatory	
5	**Action items:** Scenario to implement: GET TO KNOW THE SURROUNDINGS ● Scenario 1: Simple video recording without image recognition. Automatic patrolling to record the video of the surroundings. **Inputs**: time and frequency **Output**: recorded video uploaded to OneDrive or Dropbox. This shall be a function which the user can enable/disable via the GUI.	Mandatory	

6	**Action items:** Scenario to implement: GET TO KNOW THE OBJECTS IN THE SURROUNDINGS ● Scenario 2: Video recording with image recognition. Automatic patrolling to detect the surroundings and identify the list of objects detected. **Inputs**: time and frequency **Output**: Images of the objects found uploaded to OneDrive or Dropbox. Give names to the objects and automatically create a document which lists all the objects detected and at what time. This shall be a function which the user can enable/disable via the GUI.	Mandatory	
7	**Action items:** Scenario to implement: OBJECTS LOCATION MAPPING AND LIVE VIEW ● Scenario 3: Show live video of the object. The user shall be able to specify the name of the object from the list created in Scenario2. The camera system shall then automatically go to the location where the object is installed and show the live view to the user. **Inputs**: object name **Output**: live view on the screen This shall be a function which the user can enable/disable via the GUI.	Mandatory	

8	**Action items:** Scenario to implement: IDENTIFY WHAT HAS CHANGED SINCE LAST INSPECTION? ● Scenario 4: Show a list of changes with images. This shall be built on Scenario 2. **Inputs**: time and frequency **Output**: Images of the differences found uploaded to OneDrive or Dropbox. This shall be a function which the user can enable/disable via the GUI.	Optional	

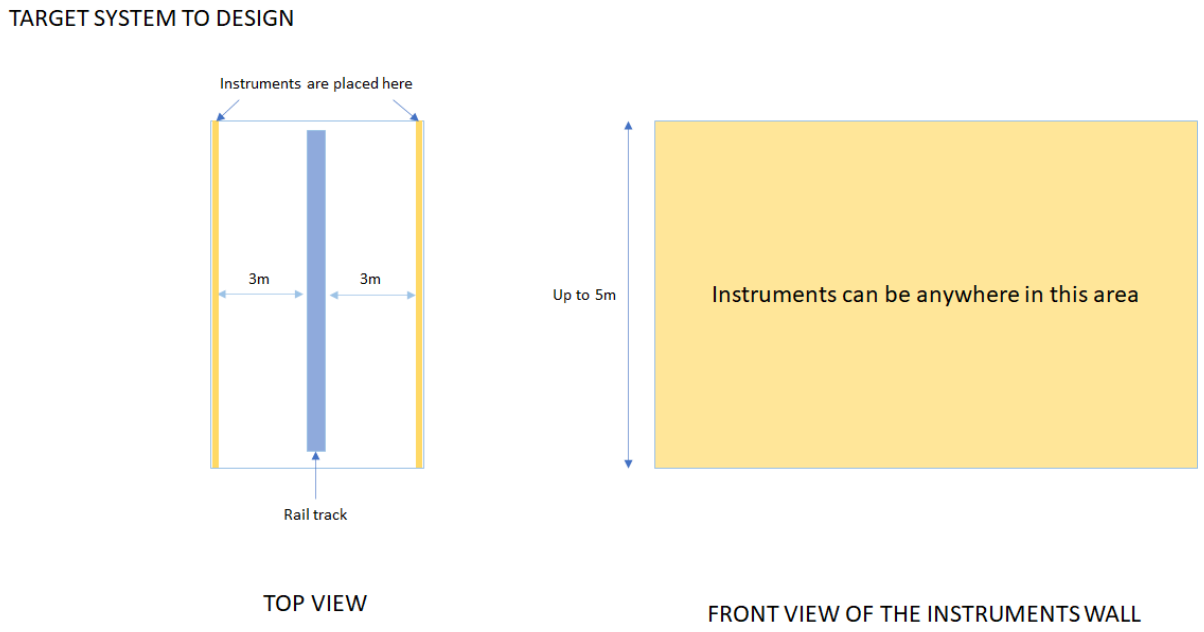
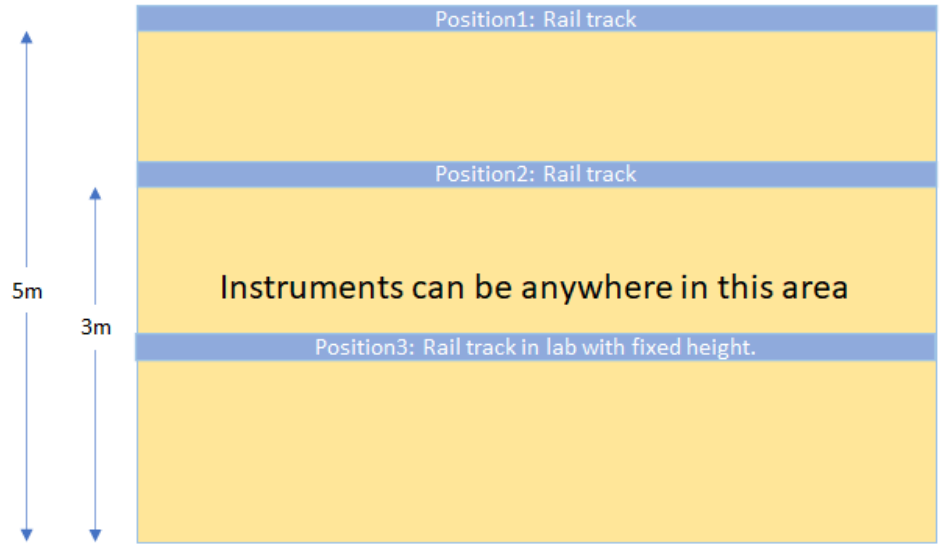


Figure 60: a). Target System to design



Design the system in a manner that it will work with minimum alterations if the rail is mounted in Position1 and Position2. During the project, you will only test Position3.

Figure 61: b). Target System with rail track

7.2 Attachment 1: Camera and mount

All configurations assume a camera with a 110° horizontal and 61° vertical field of view, with a 1920x1080 pixel resolution. For different camera specifications new calculations can easily be made following the formulas listed in these configurations.

1. **Two camera solution:**
   To increase the field of view to 180° in both directions an additional camera can be added opposite the original camera. This provides the system with a total field of view of 220°, but does not fully satisfy the request of 180° in both directions. See figure 8.

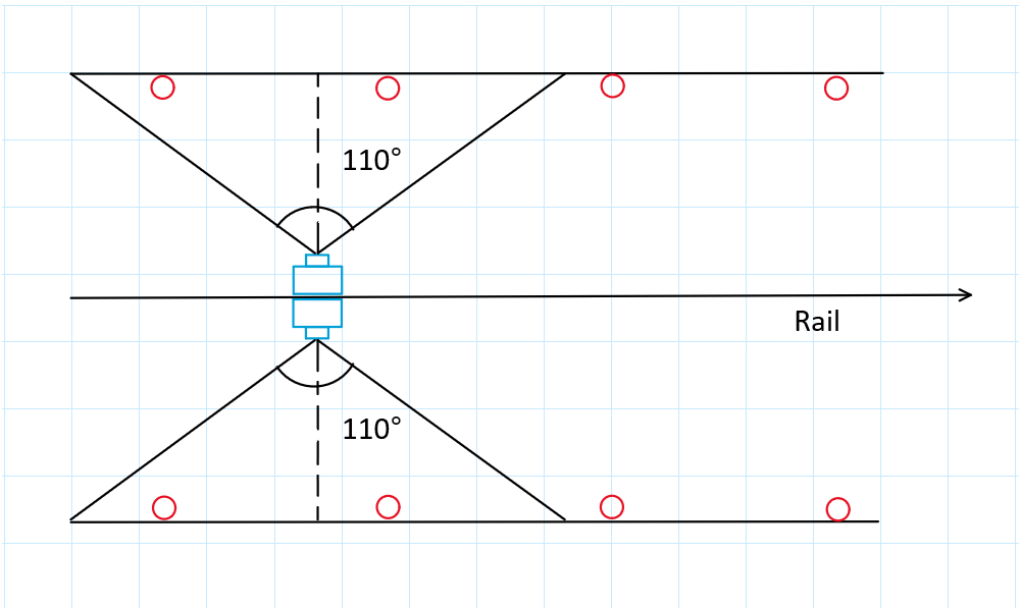


Figure 62: Two Camera orientation at 110°

2. **Wide angle lens solution:**
   A lens can be used to increase the field of view of the single camera to 180°, when combined with solution 1 the system has a 180° degree field of view in both directions.

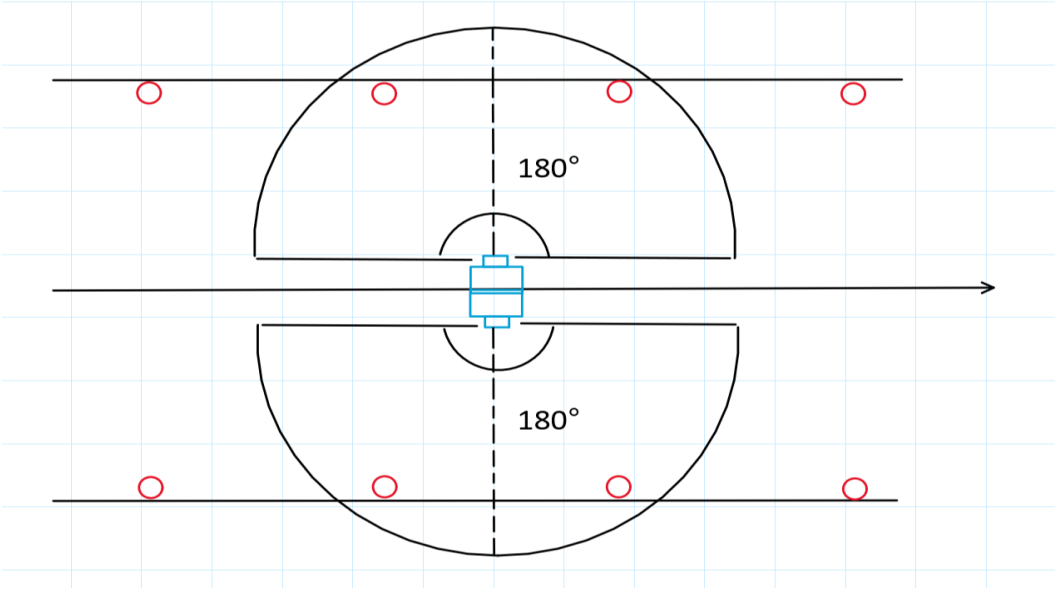


Figure 63: Two Camera orientation wide angle lens 180°

A wide angle lens solution will reduce focal length which distorts the image to fit a wider view onto the same size sensor. This decrease in focal length reduces the effective viewing distance.

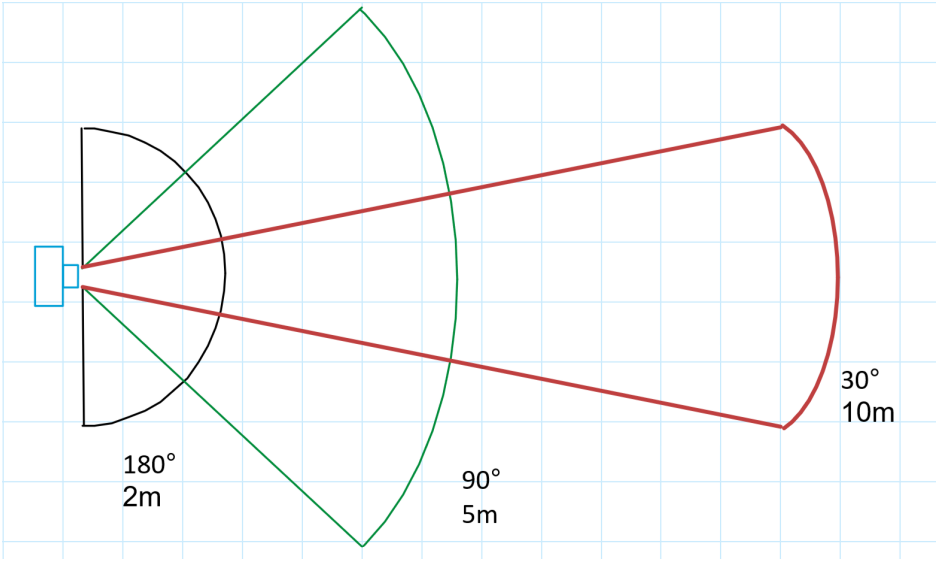


Figure 64: Showing loss of angular width with wider angle lens

3. **Overlapping FoV for two camera solution:**
For increased field of view, without decrease of focal length, a two or more camera solution can be made. Cameras are mounted perpendicularly with a 45° and 135° tilt relative to rail. The resulting setup will provide a 220° FOV in one direction. This setup will have a blind area relative to the distance between the aperture of the respective cameras. Given the size of the camera when placed perpendicularly, the AXIS cameras will approximately have a minimum 6 centimetre gap.

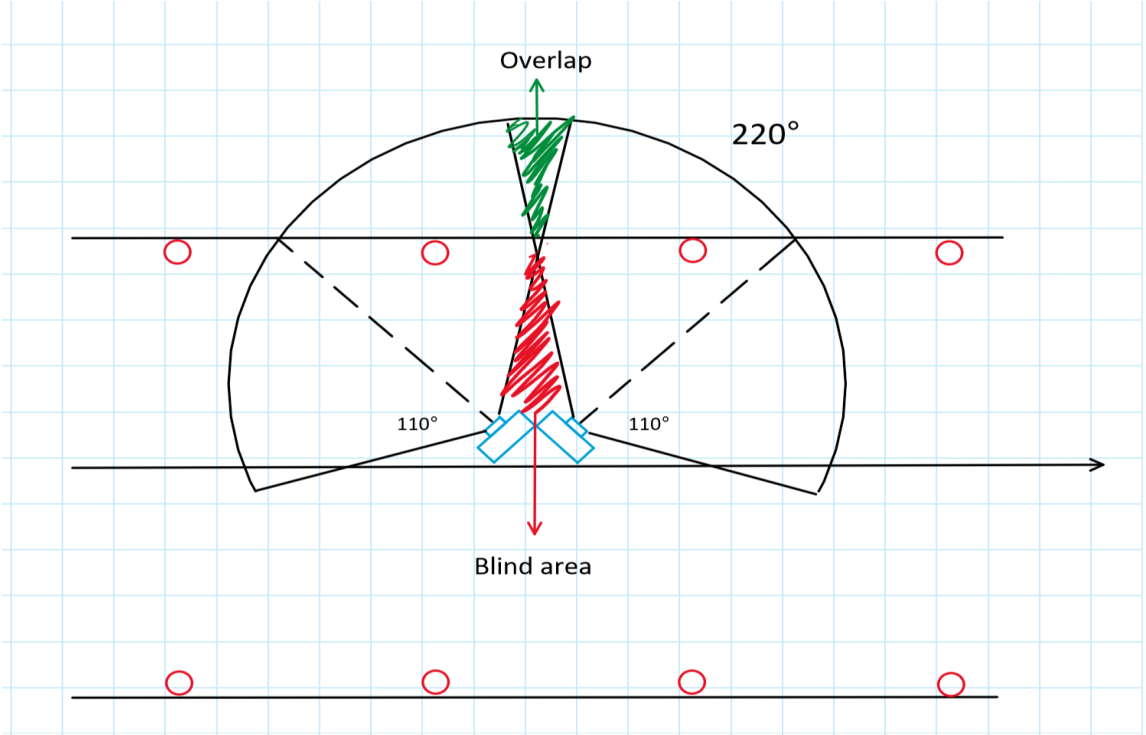


Figure 65: Showing an overlapping field of view solution

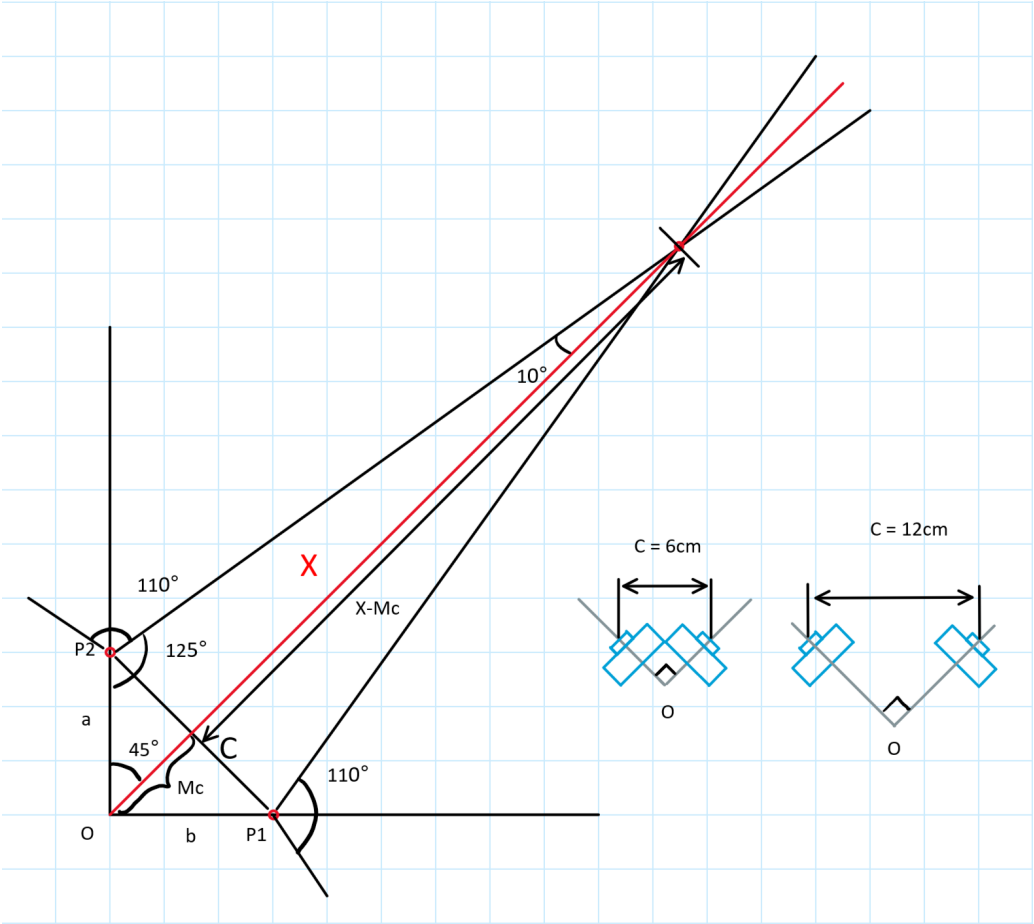


Figure 66: Showing calculations for point of overlap between cameras when mounted perpendicular

A generalised solution can be found for "x"—the distance the blind area reaches from origo to the intersection of the line segments produced by a 110 degree field of view with cameras placed perpendicularly.
x is found by applying law of sines:

$$\frac{x}{sin(125°)} = \frac{\frac{C}{\sqrt{2}}}{sin(10°)}$$

$$x = \frac{c}{\sqrt{2}} \frac{sin(125°)}{sin(10°)}$$

For ease of installation using median of C as a reference point to measure distance to end of blind area can be done by subtracting m_c from x.

m_c is found by applying Apollonius' theorem to the right angle triangle:

$$m_c = \sqrt{\left(\frac{C}{2}\right)^2 + \left(\frac{C}{\sqrt{2}}\right)^2 - C * 3\sqrt{2} * cos(45°)}$$

Solving for $C = 6cm$ produce the following results:

$$x - m_c = 20.01cm - 3cm = 17.01cm$$

For a two camera solution standing upright with a 6cm distance between camera aperture, there will be a blind area between cameras extending approximately 17cm out from the middle point between both cameras.

4. **Overlapping two camera solution for 180°:**
For a 180° FOV setup with minimised blind area, an alteration to solution 3 can be applied by changing the relative angle of the cameras. By decreasing the relative camera angle from 90° to 50°. The resulting camera angle and total field of view will give:

$$90° - 40° = 50°$$
$$220° - 40° = 180°$$

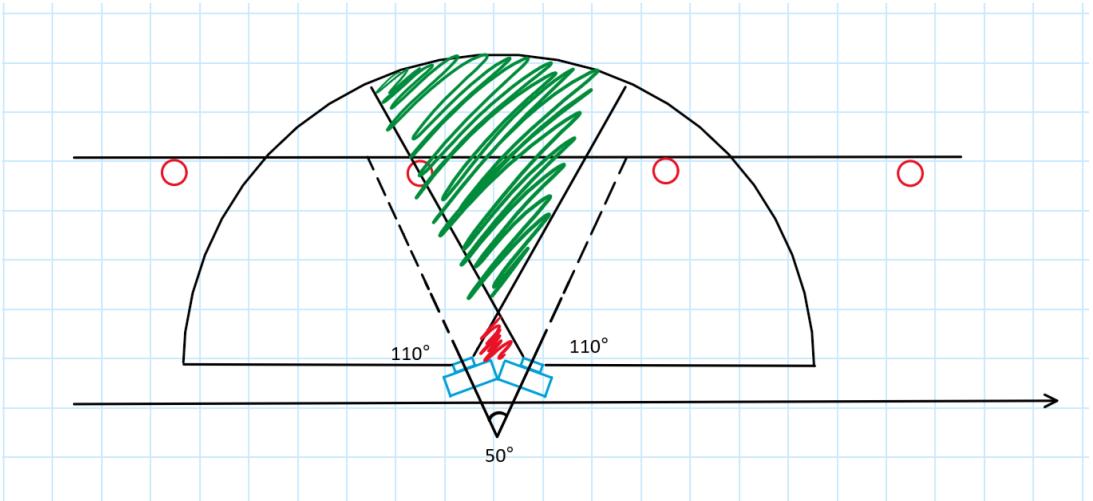


Figure 67: Showing overlapping camera solution for two cameras

Following the same reasoning as in solution 3 and using the same notation for reference the blind area is:

$$\frac{x}{sin(125°)} = \frac{\frac{C}{\sqrt{2}}}{sin(30°)}$$

$$x = \frac{c}{\sqrt{2}} \frac{sin(125°)}{sin(30°)}$$
$$x = 6\sqrt{2}cos\left(\frac{7\pi}{36}\right)$$
$$x \approx 6.95cm$$
$$a = b = \frac{6sin(65°)}{sin(50°)} \approx 7.099$$
$$m_c = \sqrt{\left(\frac{C}{2}\right)^2 + (a)^2 - C * b * cos(65°)}$$
$$m_c \approx 6.43cm$$
$$x - m_c = 6.95 - 6.43 = 0.52cm$$

The blind area for this configuration will extend 0.52cm away from the middle point between cameras.

To account for difficult angles or obstructions a swivel mounted camera was suggested. This configuration would allow for a greater degree of freedom when installed in areas with bends, or difficult angles. However to reduce potential points of failure, the decision to not add more moving parts to the rail was made.

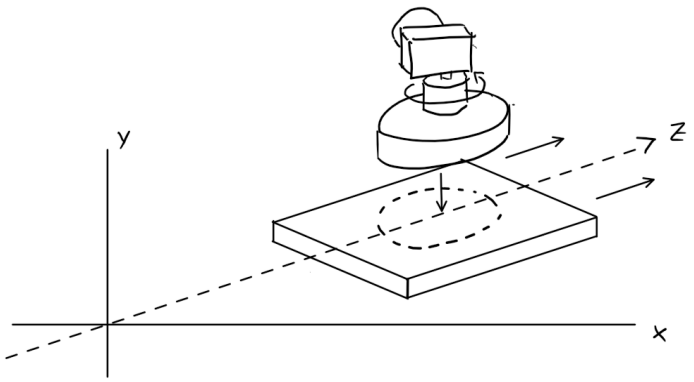


Figure 68: Mount illustration

Initial design had a small lens hood to reduce glare in the event that the camera was mounted in an area where there was bright light present. After the first revision the lens hood was removed as a redundancy and for easier mobility and an unobscured line of sight.

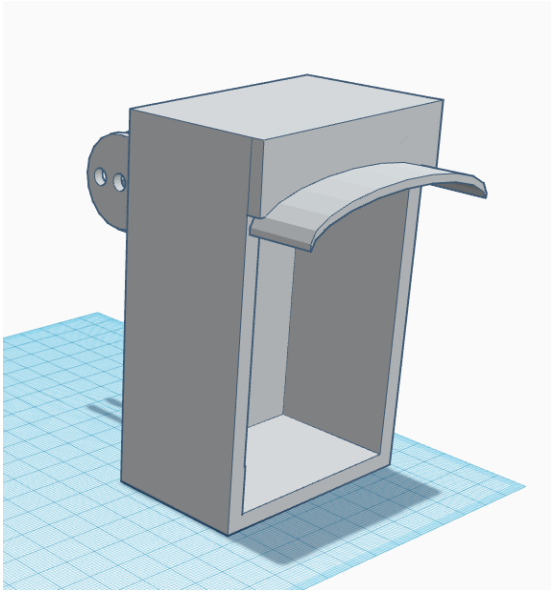


Figure 69: Mount solution

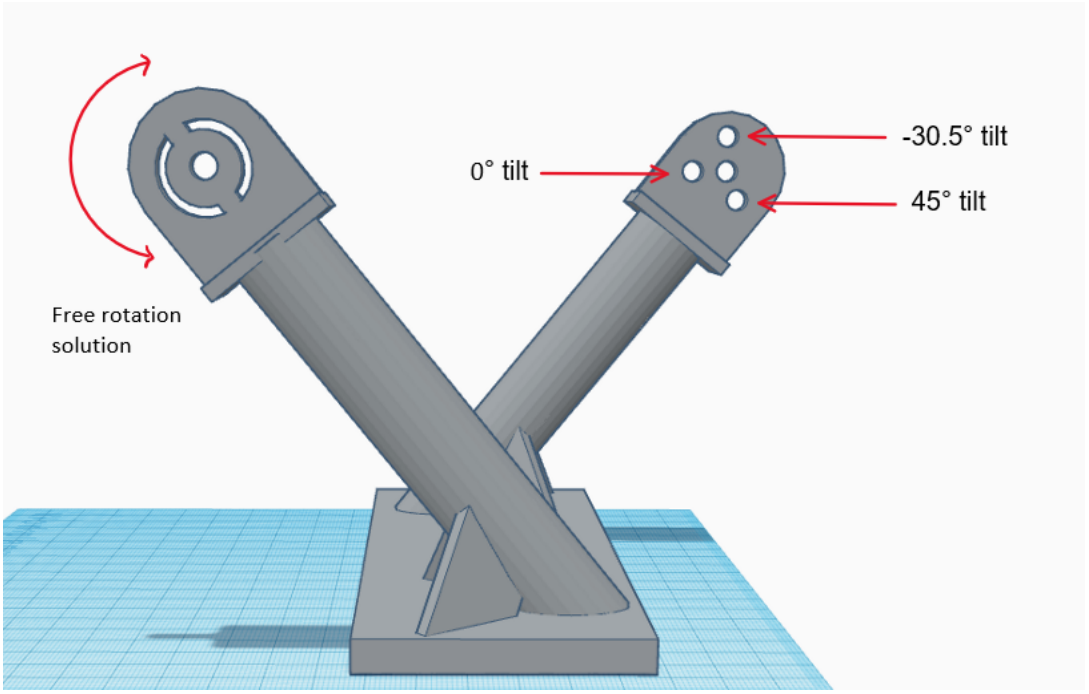


Figure 70: Mount solution

7.3 Attachment 2: QR code

QR codes are either printed directly on the corresponding part, or as a separate poster. QR codes are printed in a variety of different configurations. The size of the individual QR codes are determined by the size of the string the code represents. A string with more characters produces a higher density QR code. A 25x25 code can hold a string of 47 alphanumeric characters, and a 29x29 can hold 77. These are some of the most common QR formats in use.

Figure 71: Three different versions of QR codes (Autopilot, 2011)

The calculations for QR codes will assume that the camera is viewing a flat surface, no post processing has been applied to video feed, no mechanical zoom, perfect lighting conditions and no angular distortion. The resulting values will be for the minimum required size of the printed QR code given distance. To account for external factors like lighting, temperature or distortion, separate calculations need to be made.

A generalised model for determining the size of required QR code can be made. The model can be broken down into 5 steps:

1. Find furthest distance camera has to view
2. Find the horizontal and vertical field of view in metres.
3. Find pixels per metre (PPM) at that distance
4. Find if the largest value for PPM is vertical or horizontal
5. Find required QR code size.

Step 1:
By applying pythagoras' theorem the furthest distance a camera needs to view on any wall of a and b (height, length) can be found. A 3 by 5 room will be used as an example for ease of demonstration.

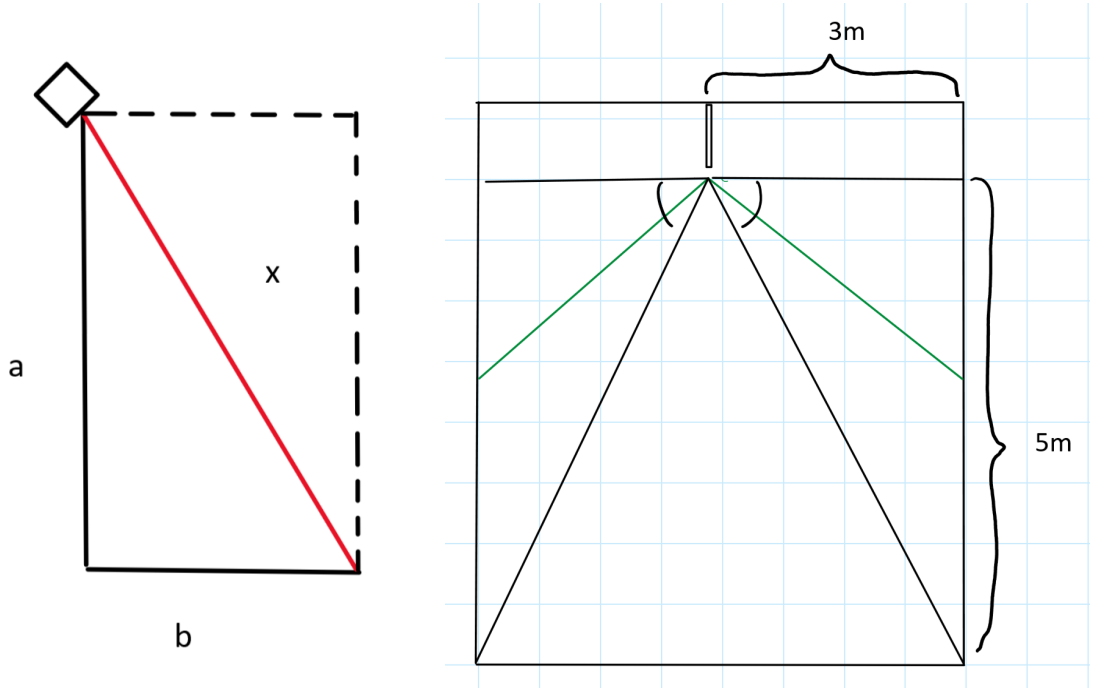


Figure 72: Left to right: illustration of pythagoras' theorem for camera in corridor, head on view of a corridor with two camera solution

The furthest point the camera needs to detect will be a ceiling mounted system at 5 metre, with a 3 metre clearing on both sides as illustrated above. Pythagoras' theorem can be applied to solve for x:

$$x = \sqrt{a^2 + b^2}$$

Solve for x gives:

$$x = \sqrt{5^2 + 3^2}$$
$$x = 5.8\text{m}$$

Step 2:
Determine the PPM (pixels per metre) at x metre the horizontal field of view needs to be known in metres at the given distance from camera to object. Law of sines is applied and the length of $X_{5.8m} = 2b = 16.6m$ is found.

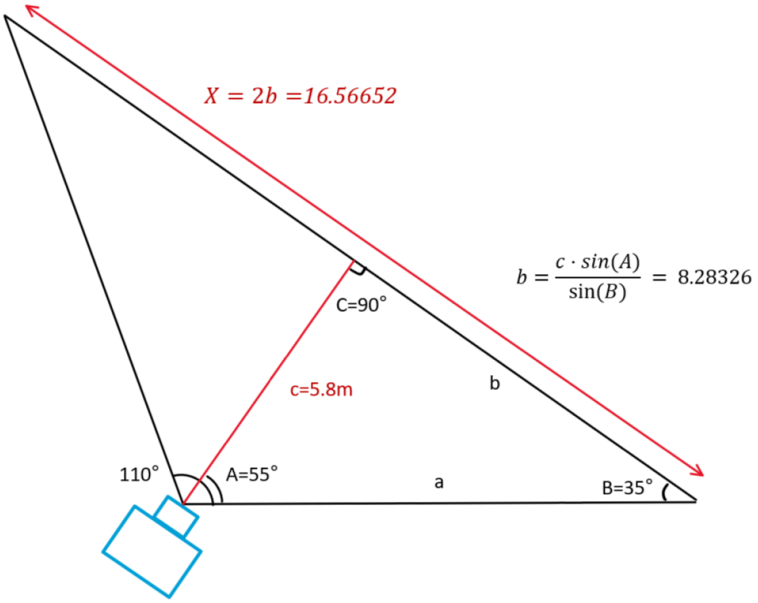


Figure 73: Law of sines for width of FOV at 5.8m

Step 3:
To find PPM we need to divide the amount of pixels by field of view.

$$PPM = \frac{Pixels}{Field\ of\ view\ in\ metre}$$

For any camera with 1920x1080 pixel resolution gives

$$PPM_{5.8m} = \frac{1920}{16.6} = 115.7$$

For the camera to be able to read the QR code at 5.8m the QR code needs to be printed with a pixel density higher than the minimum of 115.7PPM.

Step 4:
Applying the same logic to the vertical field of view produces the following values:

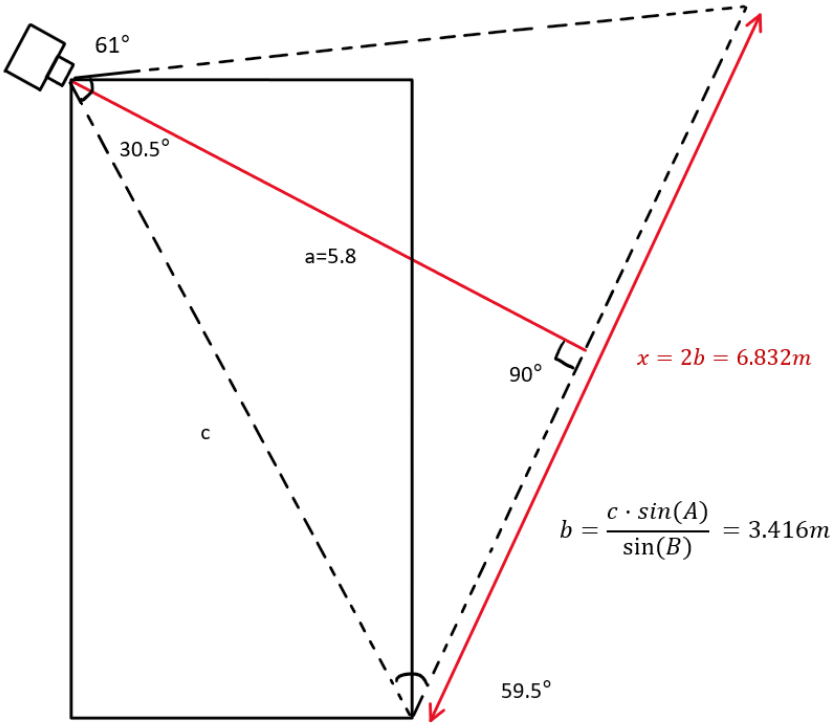


Figure 74: Law of sines for height of FOV at 5.8m

$PPM_{vertical} = \frac{1080}{6.832} = 158.08$
$PPM_{vertical} > PPM_{horizontal}$
$158.08 > 115.7$

QR codes are printed in square formats which means the lowest value PPM needs to be applied to the entirety of the print to maintain consistency. The larger value is discarded and the last step is only applied to the smaller value.

Step 5:
To determine the minimum size required for the printed QR code we need to determine the QR format, then divide QR format size by pixels per metre:

$$\frac{QR\ format\ size}{PPM} = minimum\ size\ required\ in\ metre$$

For a 25x25 QR format print we get the following:

$$\frac{25}{115.7} = 0.216m$$

Minimum vertical QR code size at 5.8m is $\frac{25}{158.08} = 0.158m$

The system should be capable of recognising QR codes for relevant objects. These
The AXIS M1065-L uses a 2.8mm fixed focal length lens, with optical zoom capabilities and an aperture of F2.0. This limits the potential detail at longer distances. Clarity of detail can be measured

in pixels per metre or PPM. Lower PPM values give less detail. For a camera to recognize a QR code it needs a higher PPM than the size of the individual "nodes" of the QR code.

This means in ideal conditions the smallest a 25x25 QR code could be printed and still be readable by the AXIS M1065-L camera is 21.6cm wide.

A generalised formula expressing the ratio of pixel density and distance from aperture to point of interest with the AXIS camera can be expressed as the function:

$$y = -38.4x + 338.6 \; for \; 3 \geq x \geq 5.8$$

Calculations for minimum distance to the wall will be 3m. Solving for x=3 we get 223.3ppm.

The distance from aperture to QR code size ratio scales linearly. This means the QR-code needs a pixel density lower than 115.7ppm at the bottom of the wall, and 223.3ppm at the top. The distance to pixel density scales linearly, so any point on the wall between these two can be found in the function $y = -38.4x + 338.6$ for $3 \geq x \geq 5.8$ where y is the pixel density given x point on the instrument wall.

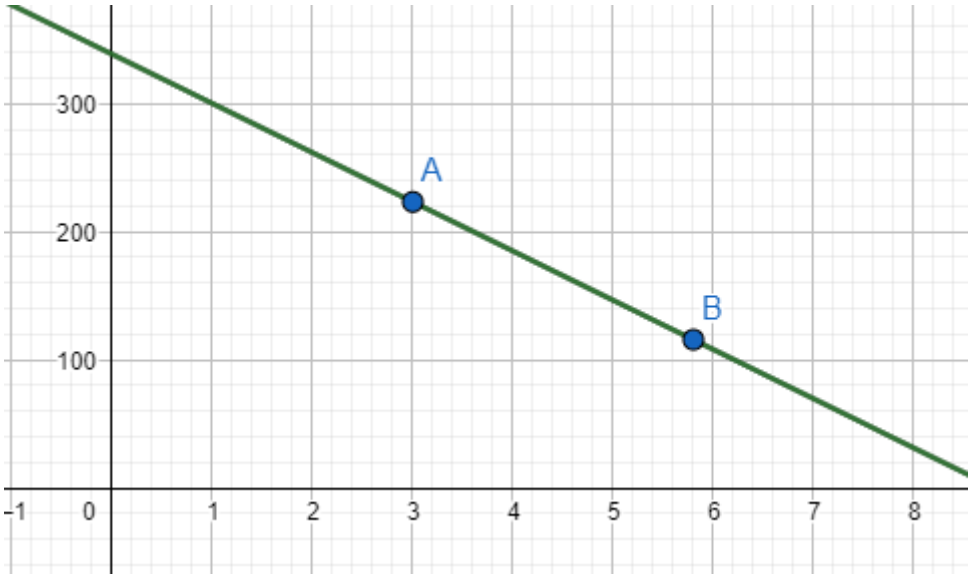


Figure 75: QR code size ratio

This generalised formula provides easy calculation for any installation distance or height.

Inversely the size of printed QR code will scale with the QR format I.E version 1, 2 and 3 etc. See figure 71.
This ratio can be expressed as a linear function of print size:
$y = 0.85x + 0.35, \; where \; x = QR \; code \; format, \; and \; y = QR \; print \; size$

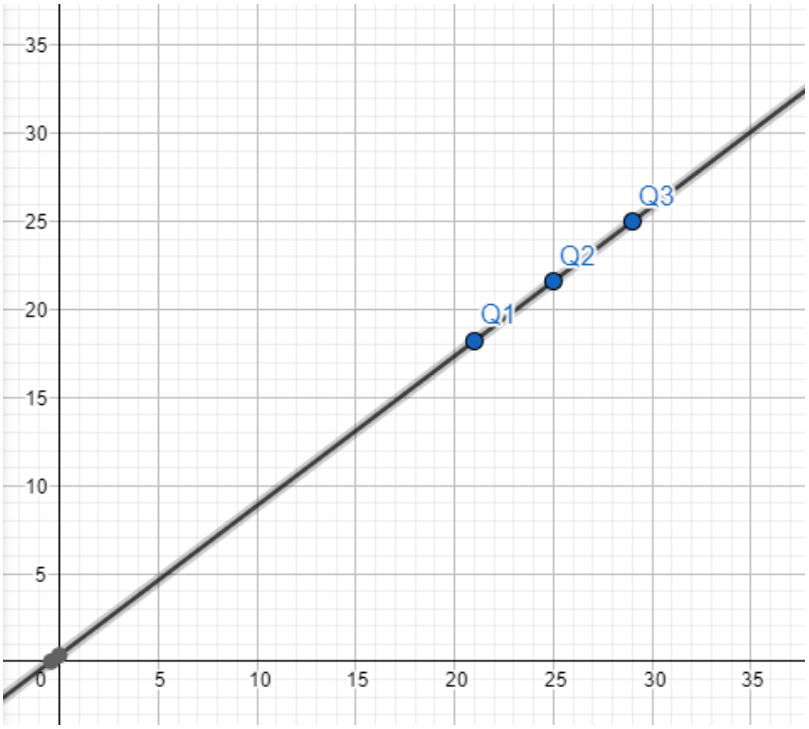


Figure 76: QR code Q1, Q2 and Q3 graphed, y as print size x as QR code format

Solving for examples Q1, Q2 and Q3:

For Q1=21x21 QR-code:

$$\frac{21}{115.7} = 0.182m = 18.2cm$$

For Q2=25x25 QR-code:

$$\frac{25}{115.7} = 0.216m = 21.6cm$$

For Q3=29x29 QR-code:

$$\frac{29}{115.7} = 0.25m = 25cm$$

The calculations are done with the assumption that the viewing conditions are perfect and a 1 to 1 scale of print size to image sensor applies. In a real scenario this will not apply, as there are outside factors like lighting, angular distortion, film grain or obstructions. Therefore scaling the size of all prints by a factor would be advised. No tests have been performed to determine a reasonable scaling factor.

At perfect conditions a 25x25 code would need to be 21.6 by 21.6cm. Which is already bigger than some of the components they are trying to locate. This means it is a highly inefficient way of identifying the objects.

Using a digital zoom as an alternative to enlarge the image “can detect the full view at all times, however, fails to provide additional visual information when zoomed” (Weerasinghe et al., 2004).

A mechanical zoom can be used to alter the focal point of the camera to enhance the image quality and PPM at the desired area, but at the loss of field of view. An alternative option for future development could be to implement a lens zoom functionality to enlarge and read QR codes

7.4 Attachment 3: Image stitching

The camera has a 1920x1080 resolution meaning it has a 16:9 aspect ratio. Different tilt configurations produce different aspect ratios with multiple camera solutions. To provide a minimum 180° FOV multiple cameras need to be used. As illustrated in suggestion 3 and 4, there will be an overlapping field of view. When running image recognition on a system with multiple cameras, having more than one camera can give duplicate detection of the same object. To avoid this an image stitching algorithm needs to be applied to the video feed before running the image recognition algorithm.

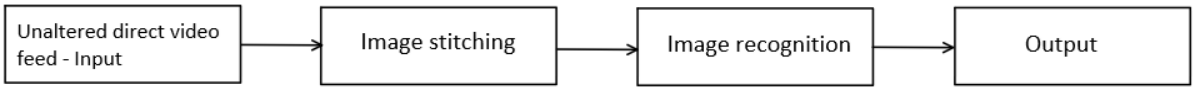


Figure 77:Image stitching system

When the aspect ratio is changed by adding multiple camera feeds, there is an increase in pixels that need to be processed. This creates a larger demand for processing power from the system. Manually viewing the video feed with a higher or lower aspect ratio than the monitor it is being viewed upon will result in cropping and loss of screen size optimization. Therefore the post stitched video feed should maintain close to a 16:9 aspect ratio and a 1920x1080 resolution.

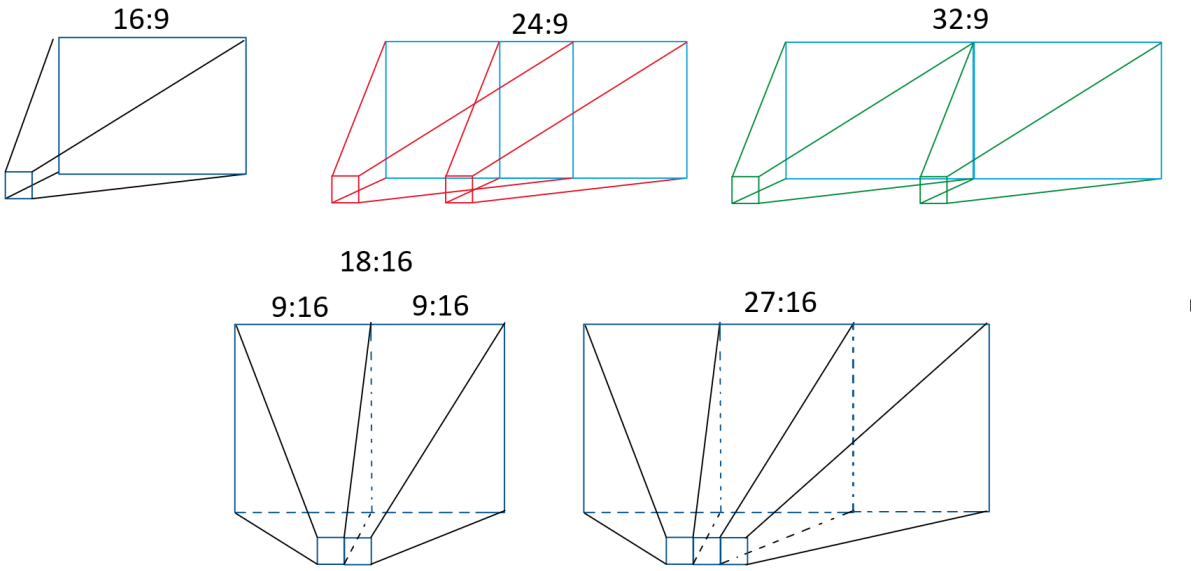


Figure 78: Aspect ratio examples

(a) Input image pair (b) Before refinement (c) After refinement

Figure 79: Image stitching example (Kang et al., 2019)

Stitching can be done in multiple ways, but usually involves a process of finding reference points of high contrast in both images that can be geometrically warped to produce an image that appears seamless. In modern methods this process uses AI to generate these points of reference for optimal splicing.

Due to hardware inaccessibility this project never adopted a multiple camera solution to its process, and therefore will not be detailed further.

7.5 Attachment 4: Lighting

For the camera to achieve sufficient viewing light, external lighting sources could be added to maintain consistent illumination.
For the camera to be able to perform any kind of image recognition it needs sufficient lighting to produce consistent and reliable video feeds. The degree lighting affects the system's ability to recognise objects is determined by multiple factors, such as sample data, obstructions

Camera illumination for computer vision:

Low lighting poses an obstacle for an automated camera running a computer vision AI algorithm. Therefore we need to come up with a solution that gives consistent lighting with repeatable conditions for the camera to work with. We want to implement a wireless or near wireless solution that is ready for use without any connections to the local power or lighting grid. This means we need to find our own lighting solution for the camera rail system.

Problems:
- Low lighting
- Inconsistent lighting
- Power supply
- Weight

Requirements:
- Consistent lighting
- Infrequent maintenance
- Visible and IR light
- Wireless

Potential solutions:

Turning on PIR sensor lights at location:

We want to simulate human movement with infrared light to trick PIR sensors to turn on local lighting. Initially this seemed like a good solution, as most installation sites would have local motion activated lighting that could be utilised. A PIR (passive infrared) sensor works by having a pyroelectric sensor tuned to detect infrared light in the spectrum of human body heat, which is approximately 10µm to 14µm. As we can see in this graph, that only makes up a very small portion of the infrared spectrum.

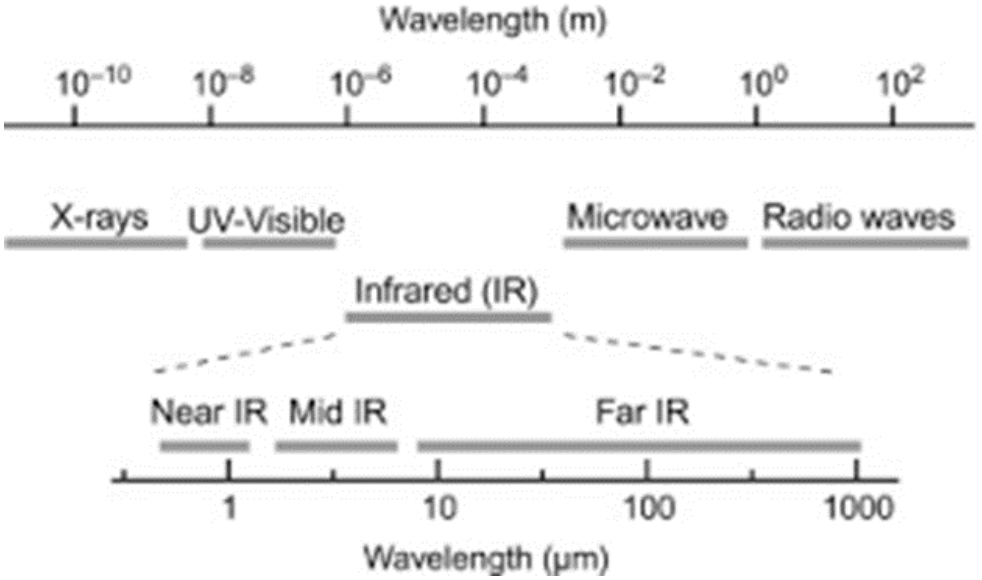


Figure 80: Showing infrared light spectrum (*Infrared Light - An Overview*, n.d.)

This means we can't trigger these sensors with just any type of IR light, and the photonic quantity needs to be high enough for it to consistently reach the sensor to trigger a response.

Most IR LEDs are not made to simulate human body temperature, as there would be interference from any human interacting with the receiver. So an infrared emitting diode would have to be made specifically to broadcast a 10-14µm wavelength signal.

Alternatively we could simulate human body temperature with an incandescent bulb that emits light at around 37 degrees or hotter, to increase the amount of photons in the 10-14µm spectrum. As a hotter source would emit more photons we could increase temperature to increase likelihood of detection.

However, after careful consideration, we concluded that using a diode or light source to trigger a PIR sensor for turning on light often meant adding more problems to our project, so we abandoned the idea for a simpler one.

IR camera with or without IR LED emitter:

The next natural step was to consider using the infrared spectrum for video feed in low lighting scenarios. This would be a viable solution in most workplaces where there is natural light coming in, or by attaching an IR light emitter source onto our camera rail system. IR cameras are able to pick up a lower wavelength signal than visible light, meaning there is less energy required to illuminate. Most IR cameras can detect light from 1µm to 14µm, which is usually present even at night through ambient lighting.

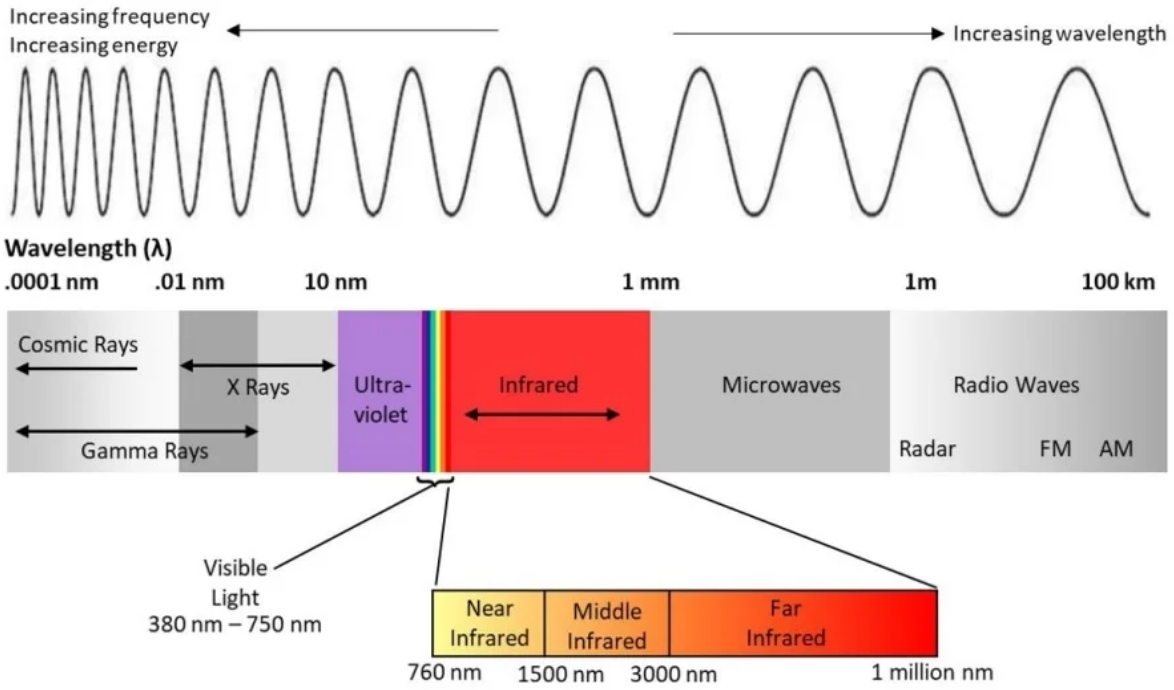


Figure 81: Infrared wavelength scale (Radiant Vision Systems, 2019)

This means we can illuminate our workspace efficiently with less energy than what would be required for visible light. The only downside to this, is that we lose the ability to use colour as a factor in our training data, and should we decide to use both visible and IR light, we will have to train our AI on two different sets of data.

Rail system light source:

The obvious solution would be to simply add lights to the rail system. Either at each end of the rail, or directly to the moving platform.

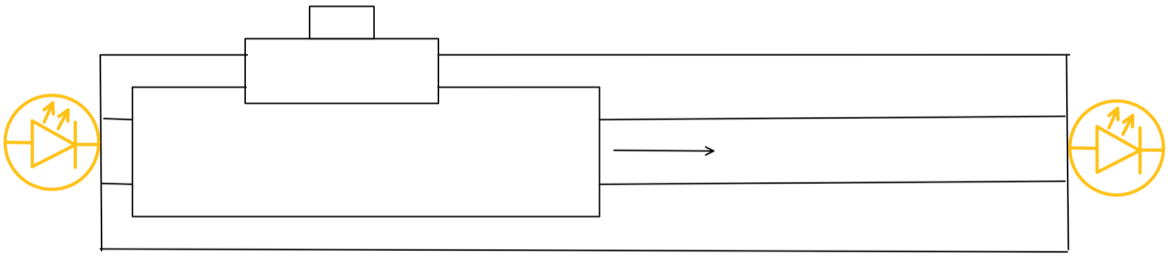


Figure 82: Illustration of light source solution

However this means we either need to be connected to a power supply, or we need to run it off an independent voltage source. The downside to this, is that it will require intervals of maintenance to swap batteries for the rail system to remain functional.

7.6 Attachment 5: Raspberry Pi

To lower the processing power needed by the lab computer a separate system on a chip (SoC) could be used to handle the image stitching before running the image recognition algorithm. The Raspberry Pi 4 was suggested as an option for this process. It uses a Broadcom BCM2711 chip which runs at 1.5 GHz, with 8GB SDRAM. Without a dedicated GPU and no VRAM the Raspberry Pi can not maintain a high frame rate when running on a video feed in real time. As the group never received a second camera compatible with the system this idea was not elaborated upon further.

7.7 Attachment 6 Risk analysis:

RISK ANALYSIS

This risk assessment is based on procedures from ISO 12100 (International Organisation of Standardisation). The purpose of this document is to identify risk with the machine and will conclude with whether the machine is safe or not. ISO 12100 defines risk as a combination of the probability of occurrence of harm and the severity of harm. In this context harm means physical injury or damage to a person's health.

The limit of the machine is important to establish as a framework for the risk assessment. This will help assess and evaluate the hazards that can arise when operating the machine, or during the machine's life cycle

Determination of the limits of the machine

The machine will be used for unmanned inspection of remote installations on wind farms, processing plants and pipelines.

The machine will operate in proximity with humans but will for the most part operate without human presence, as the purpose of the machine is to reduce humans working in the same area. The operators of this machine will be trained with the machine and will work remote from the machine at a control centre.

The machine does not have a safeguard and may be subjected to changes in temperature, humidity, and different environmental aspects, depending on which environment it is placed in.

The machine is a two metre long toothed belt system from Festo with four proximity sensors used as a redundant system to calculate position along with measuring revolutions in the motor.

The machine has a moveable part with a load attached (camera) that has 1 degree of freedom in the translational direction, and 0 rotational degrees of freedom.

From festo's operation manual of the toothed belt system it is specified that the speed can be set from 0.0 metres per second to 10.0 metres per second. The speed of the toothed axis belt has been limited by the PLC and can be set from 0.0 metres per second to 0.4 metres per second with a payload of approximately 0.134 kilograms (weight of Axis Camera M1065-L). This creates a maximum momentum of 0.0536 kg*m/s.

The machine has limited visibility in the direction of the movement.

The machine's image recognition will use AI (artificial intelligence) to control the motor, and may therefore be unpredictable in its movement.

The system is controlled automatically by a PLC.

The system is not entirely restricted by password or locked.

Hazard identification

- **What are the *potential* sources of harm?**

From the machine's limitations we can easily spot the potential hazards in the machine. Hazards is a potential source of harm, and harm in this context can be to either human health, or equipment damage. Harm to humans can be to either the operator, or any harm caused to humans regardless if they are interacting with the machine.

Hazards

- Operating in proximity with humans
- No safeguards
- Changes in environment
- Electrical hazard
- Mechanical hazard
    - Moveable load
    - Losing track of position of load
    - Momentum of load
- Limited visibility
- Controlled by PLC/AI
- PLC not restricted

To maintain high safety and low risk, we ideally want all hazards to fall into the category of very unlikely, but some problems are more prevalent than others.

Risks from hazards

The machine will be in proximity to humans at times. The machine's intended purpose is to reduce human interaction, but there must be maintenance done on the machine at regular intervals, another point where humans will be in proximity is when responding to an error discovered by the machine in the working environment. This hazard in combination with hazards can cause a high level of risk to human health.

- **Mechanical hazard**
    - **Danger of crushing**
        - Risk of crushing if/when a person is moving within the machine’s working area. This can happen if a person gets a body part stuck between the mounted camera and the rail.
    - **Impact**
        - Risk of impact if/when a person is moving in the machine’s working area. There’s a risk of the machine running into a person as it’s moving along the rail.
    - **Entanglement**

- Risk of entanglement if/when a person is moving in the machine's working area.

- **Electrical hazards**

  - **Electrical shock**

    - According to the Festo Repair instructions, there is a risk of fatal injury due to electrical shock and uncontrolled movement of components.
    - Sudden changes in environment

  - **Fire from faulty wiring**

    - The insulation on the wires can wear out, causing a fire hazard.
    - Sudden changes in environment

Controlled automatically by PLC / AI

  - **Unpredictable movement**

    - May cause mechanical risks, such as impact, crushing or entanglement due to unpredictable behaviour.

- **PLC not restricted/secured from unauthorised people**

  - **Changes in speed of system**

    - Changes momentum of the load and therefore increases mechanical risks

  - **Changes in system**

    - Can cause the system to behave differently than what is expected, and thus may cause substantial damage.

Table 4: Risk matrix for system

	Consequence:				
Probability:	negligible	low	moderate	critical	fatal error
very unlikely					
unlikely					
moderate					
likely					
very likely					

Risk reduction

Physical aspects

- Limit the mass and/or velocity of moving elements to reduce their momentum.
- Wires should be bundled up to reduce risk of entanglement.
- The wiring should be inspected regularly to monitor wearing.

Safeguards

- Fixed guard
- Markers or barriers to indicate the machine's workspace
- Helmet usage for workers in the area
- Sensors that detect motion inside the machine's workspace
- Emergency stop (Already in place)

Min/max speed and velocity

Limitations:

- Servo motors
- power supply
- Load bearing belt
- PLC

As far as minimum speed limitations go, there are no tangible limits to the system as we can theoretically set the speed as low as we require.
As mentioned above the upper limit for our rail is 10 m/s, but we only utilise a max of 0.4m/s

As per the manual we can see that there is a tradeoff in acceleration given by the mass of the object being moved by the rail system.

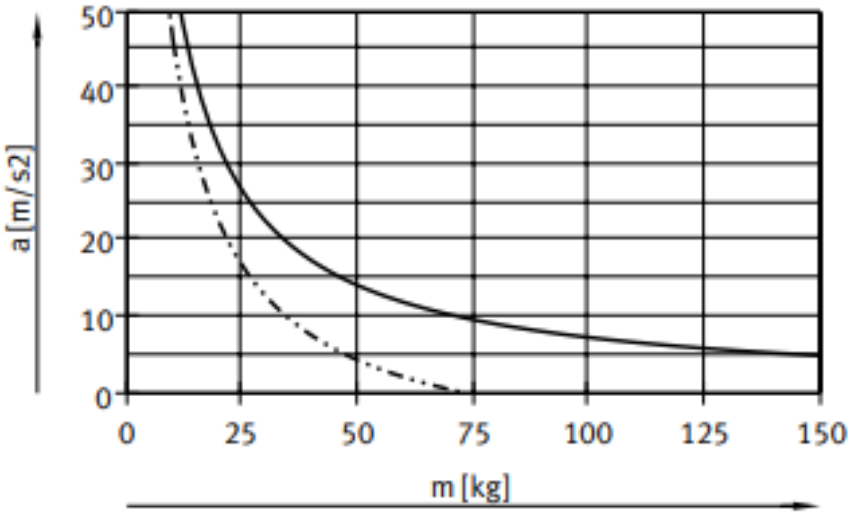


Figure 83: Mass-Acceleration graph (FESTO, 2015)

7.8 Motor limitations

Calculating power required:

LxWxT:

$500 \bullet 0.08 \bullet 0.003 = 0.12m^3$

$0.12m^3 \bullet 1360kg/m^3 = 163.2kg$

$mass \bullet \mu$ = req. belt pull

$163.2 \bullet 0.5 = 81.6$

$belt\ pull \bullet belt\ speed$ = req.power

$81.6 \bullet 60 = 4896$

$1Hp = 76kgm/s = 4560\ kgm/min$

$4896/4560 = 4896$

$1.07\ Hp = 0.798\ kW = 798\ W$

$T = torque(Nm)$
$D = roller\ diameter(m)$
$W = mass(kg)$
$g = gravity\ accel\left(\frac{m}{s^2}\right)$
$\mu = friction\ coefficient$
$F = External\ force(N)$

Calculating torque required:

$T = 1/2 \bullet D(F \bullet \mu Wg)$

$T = 1/2 \bullet 0.1(0 + 0.5 \bullet 163.2 \bullet 9.81)$

$T = 40\ Nm$

Using the maths above, we can create a table:

Table 5: Motor limitations for conveyor belt

Conveyor belt length	250m	500m	1000m	2000m
Req. power	798 W	1596 W	3192 W	6384 W
Req. torque	40 Nm	80 Nm	160 Nm	320 Nm

These numbers are only intended to give a rough idea of the motor limitations, as general numbers have been used, such as the density of rubber and the friction coefficient.

7.9 Flowchart

7.9.1 Scenario 1

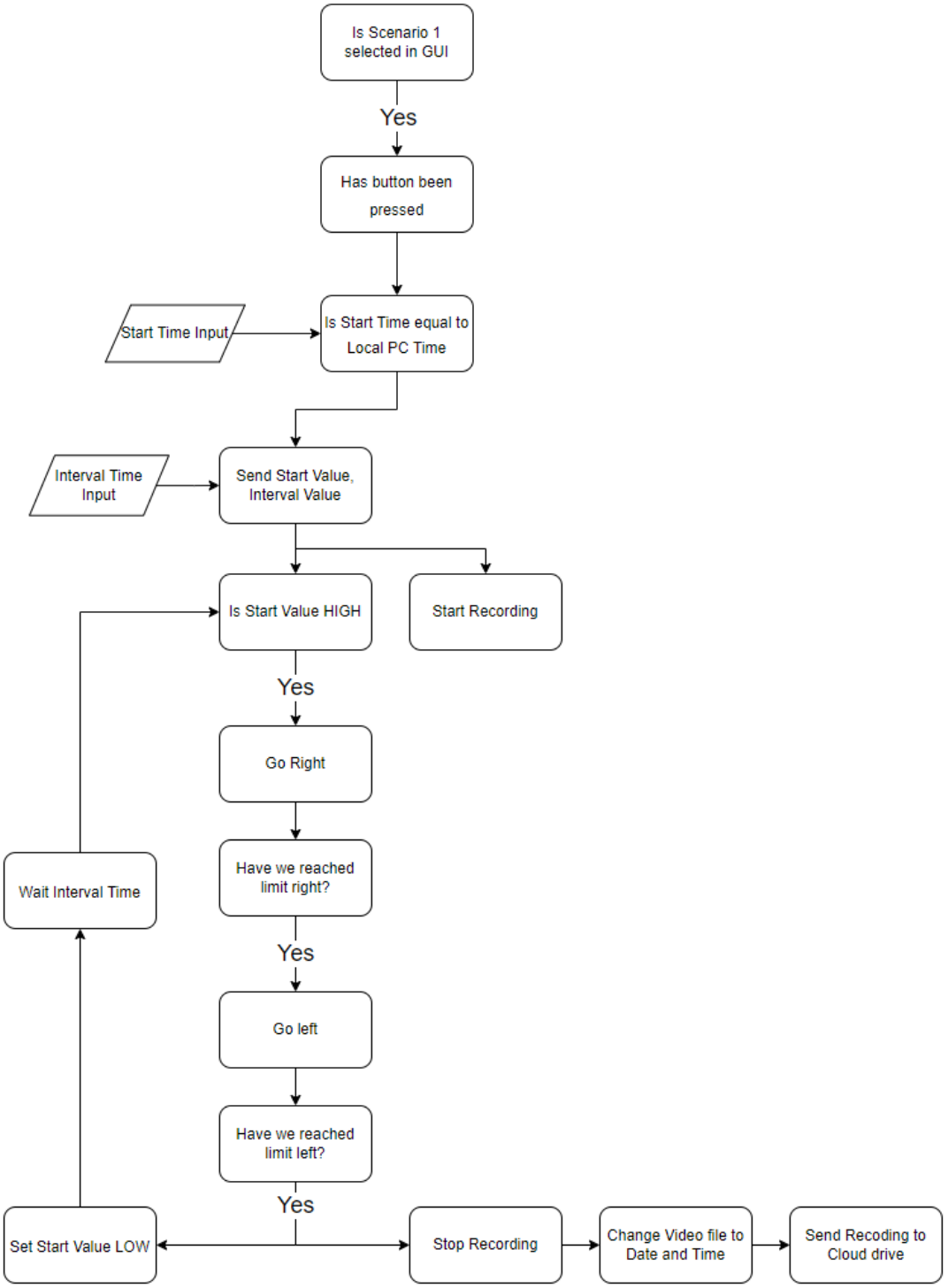


Figure 84: Flowchart of Scenario 1

7.9.2 Scenario 2

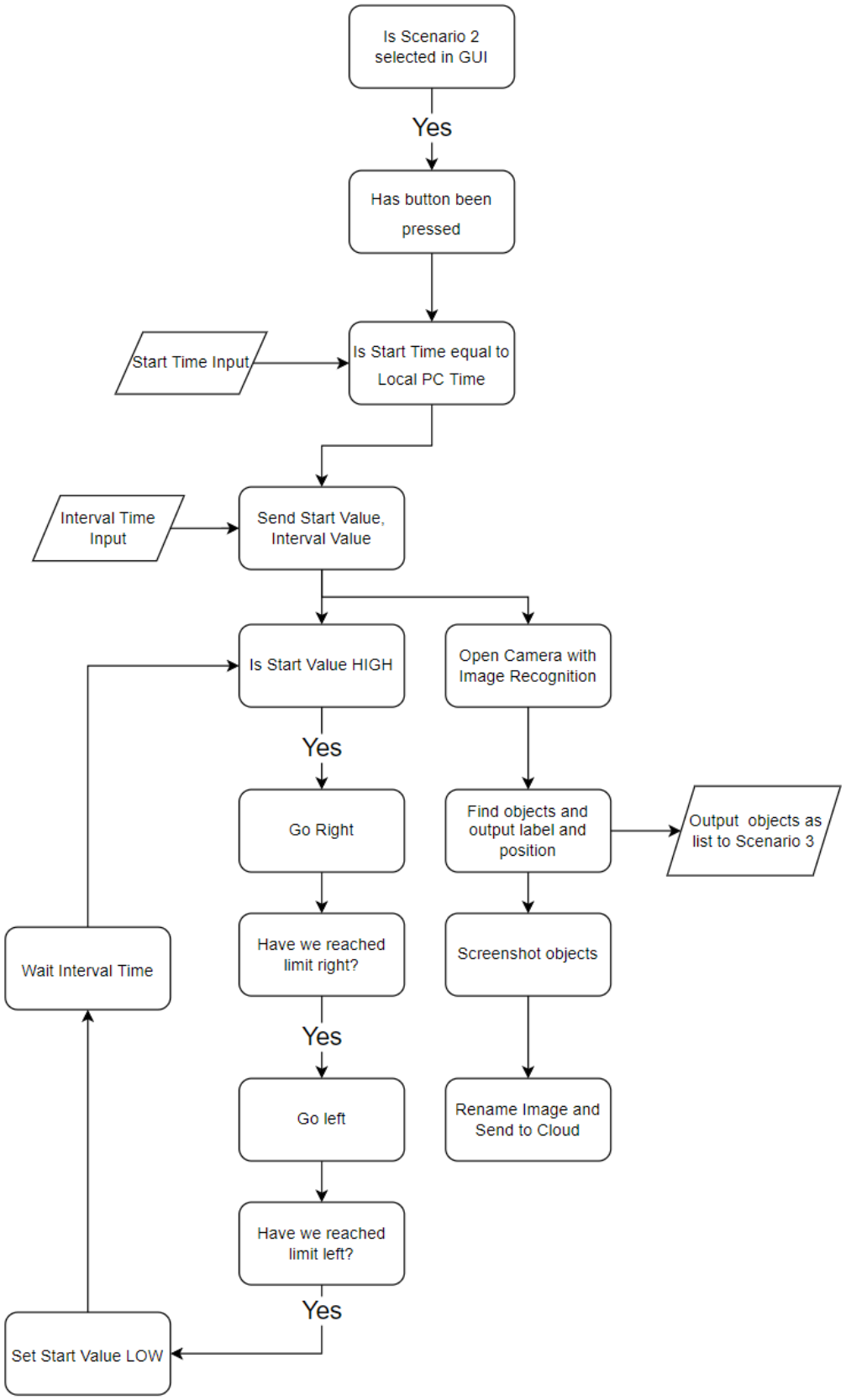


Figure 85: Flowchart of Scenario 2

7.9.3 Scenario 3

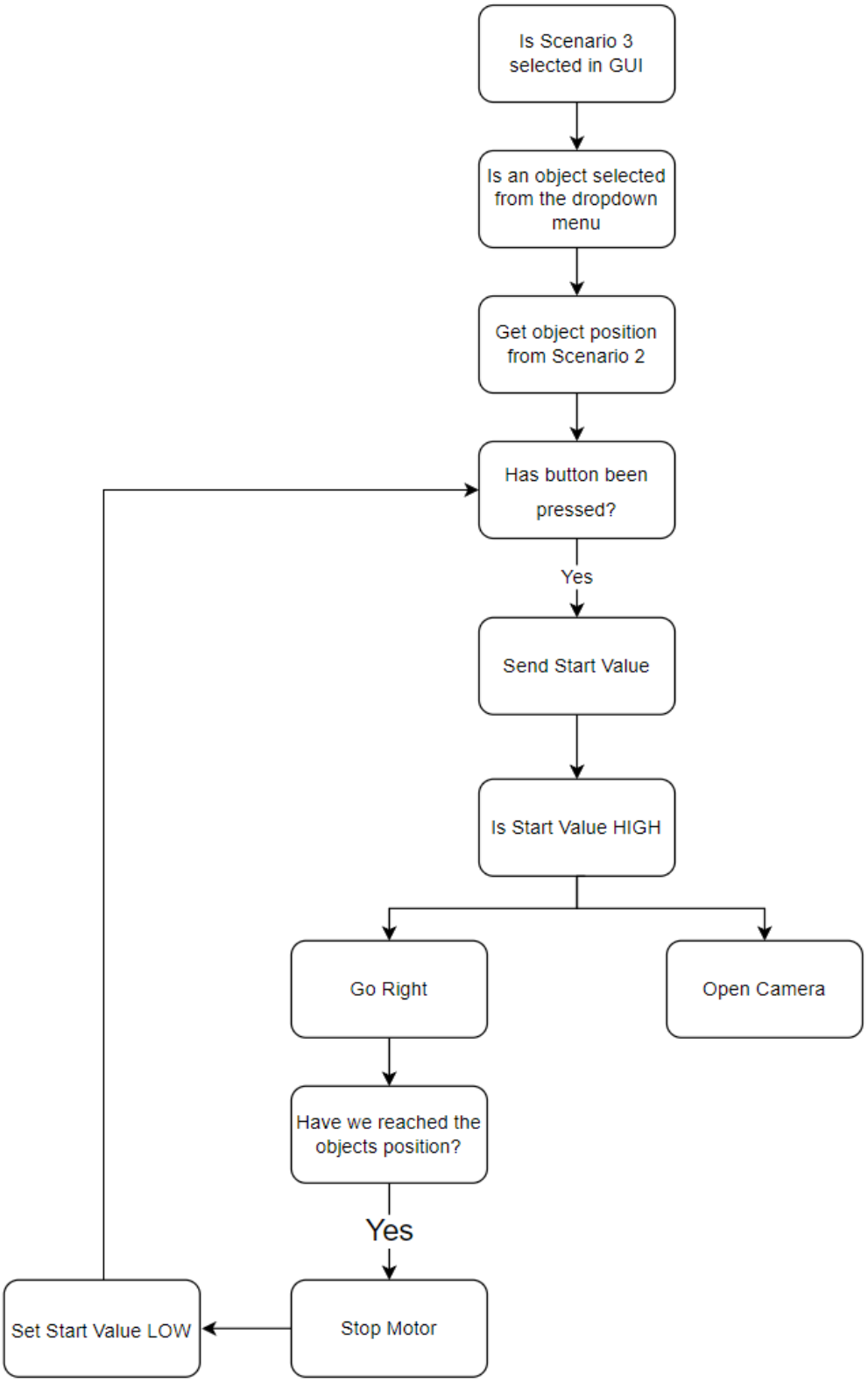


Figure 86: Flowchart of Scenario 3

7.10 Python code

7.10.1 AI model implementation

Table 6: Python code for AI implementation

```
class remoteInspection:
    """
    Class implements the Yolo5 model.
    """

    def __init__(self, capture_index, model_name):
        """
        Activates GPU processing if available.
        """
        self.model = self.load_model(model_name)
        self.classes = self.model.names
        self.device = 'cuda' if torch.cuda.is_available() else 'cpu'
        print("Using Device: ", self.device)

    def get_video_capture(self):
        '''
        return: opencv2 video capture object.
        '''


                                                                return
cv2.VideoCapture('rtsp://root:ABB1234@192.168.1.3/axis-media/media.am
p')

    def load_model(self, model_name):
        """
        Loads  custom Yolo5 model from local folder.
        :return: Trained model.
        """
        if model_name:
               model = torch.hub.load(os.getcwd(), 'custom', source =
'local', path=model_name, force_reload=True)

        return model

    def score_frame(self, frame):
        """
```

```
        :param frame: input frame in numpy/list/tuple format.
         :return: Labels and Coordinates of objects detected by model
in the frame.
        """
        self.model.to(self.device)
        frame = [frame]
        results = self.model(frame)
          labels, cord = results.xyxyn[0][:, -1], results.xyxyn[0][:,
:-1]
        return labels, cord

    def class_to_label(self, x):
        """
        :param x: numeric label
        :return: corresponding string label
        """
        return self.classes[int(x)]

    def plot_boxes(self, results, frame):
        """
         :param results: contains labels and coordinates predicted by
model on the given frame.
        :param frame: Frame which has been scored.
        :return: Frame with bounding boxes and labels plotted on it.
        """
            labels, cord = results   #Loads labels, coordinates and
confidence level into variables, cord[4] being confidence level
        n = len(labels)
          x_shape, y_shape = frame.shape[1], frame.shape[0]    #Loads
shape of frame from numpy array
          global count     #Variable to prevent jittering in the motor
from speed changes
          global speed     #Variable to prevent having to writing to
modbus register unnecessary
        global posList  #List to store position of objects detected
          global labelList     #List to store labels of corresponding
objects in posList

        if n > 0:   #Object(s) detected in frame
            for i in range(n):
                row = cord[i]
```

```
                    if row[4] >= 0.4:    #Check whether confidence level
is over set threshold

                        count += 1
                        if count > 10:
                            count = 10

                                x1, y1, x2, y2 = int(row[0]*x_shape),
int(row[1]*y_shape),      int(row[2]*x_shape),      int(row[3]*y_shape)
#Coordinates of predicted rectangle around object
                        bgr = (0, 255, 0) #Colour
                          cv2.rectangle(frame, (x1, y1), (x2, y2), bgr, 2)
#Plots rectangle
                                                        cv2.putText(frame,
self.class_to_label(labels[i]),  (x1,  y1),  cv2.FONT_HERSHEY_SIMPLEX,
0.9, bgr, 2) #Plots label of object

                        if (320*0.85) < ((x1+x2)/2) < (320*1.15): #Set
velocity of slide if detected object is within 15% of the middle of
the frame
                            speed = 1
                            client.write_registers(8,[1000,0])

                              if (320*0.98) < ((x1+x2)/2) < (320*1.02):
#Check if object detected is in the middle of frame

                                                                posRail   =
client.read_holding_registers(20).registers[0]
                            label = self.class_to_label(labels[i])

                            saveImg = 0

                             for x in range(len(posList) + 1):   #Loop to
save and sort position, and images of object

os.chdir(r"C:\Users\User3\Desktop\HVIT\yolov5\PictureFeedBack")
                                try:
                                        if((posList[x] - 200) < posRail <
(posList[x] + 200)) and labelList[x] == label:     #Checks if object
has been seen at same rail position previously
                                            break
```

```
                                elif posRail < posList[x] - 200:
                                    tempPos = posList[x]
                                    tempLabel = labelList[x]
                                    posList[x] = posRail
                                    labelList[x] = label
                                    label = tempLabel
                                    posRail = tempPos

                                    if saveImg == 0:

                                                                        timeLabel =
strftime("%Y-%d-%b-%H%M", localtime())
                                                                cv2.imwrite(label +'-'+
timeLabel + '.jpg', frame)
                                                    saveImg = 1

                                elif posRail > posList[x] + 200:

                                    if saveImg == 0:

                                                                        timeLabel =
strftime("%Y-%d-%b-%H%M", localtime())
                                                                cv2.imwrite(label +'-'+
timeLabel + '.jpg', frame)
                                                    saveImg = 1

                                    continue
                            except:
                                posList.append(posRail)
                                labelList.append(label)

                                if saveImg == 0:
                                                                        timeLabel =
strftime("%Y-%d-%b-%H%M", localtime())
                                                                cv2.imwrite(label +'-'+
timeLabel + '.jpg', frame)
                                                    saveImg = 1

                    else:   #Set speed up if no object is detected
                        count -= 1
```

```
            if count < 0:
                    count = 0
            if speed == 1 and count == 0:
                    client.write_registers(8,[5000,0])
                    speed = 0

        return frame

    def __call__(self):
        '''
        The call function of the class.
        '''

        cap = self.get_video_capture()
        cap.set(cv2.CAP_PROP_BUFFERSIZE, 0)
        assert cap.isOpened()

        while client.read_holding_registers(24).registers[0] == 1:

            ret, frame = cap.read()
            assert ret

            frame = cv2.resize(frame, (640,360))

            #start_time = time()
            results = self.score_frame(frame)
            frame = self.plot_boxes(results, frame)

            #end_time = time()
            #fps = 1/np.round(end_time - start_time, 2)
                    #cv2.putText(frame, f'FPS: {int(fps)}', (20,70),
cv2.FONT_HERSHEY_SIMPLEX, 1.5, (0,255,0), 2)

            cv2.imshow('Remote Inspection', frame)

            if cv2.waitKey(1) & 0xFF == 27:
                client.write_registers(4,[0,0])
                client.write_registers(6,[1,0])
                break

          client.write_registers(8,[5000,0])   #Set speed up if ESC is
```

```
pressed, or patrolling is done


        '''
          Saves labels and corresponding positions in .txt file when
patrolling is done
        '''
        os.chdir(r"C:\Users\User3\Desktop\HVIT\yolov5")
        np.savetxt('labelList.txt', labelList, fmt = '%s')
        np.savetxt('posList.txt', posList)

        cap.release()
        cv2.destroyAllWindows()
```

7.10.2 GUI

Table 7: Shows the code for the GUI

```
import os
import sys
import cv2
import time
import torch
import shutil
import threading
import numpy as np
import tkinter as tk
from aiModel import *
from time import *
from time import time
from tkinter import *
from tkinter import ttk
from pymodbus.client.sync import ModbusTcpClient
from azure.storage.blob import BlobServiceClient, BlobClient,
ContainerClient


#Get connection-string from Azure Storage, get which folder to upload
to in Cloud
Connect_str =
'DefaultEndpointsProtocol=https;AccountName=storagehvit;AccountKey=JU
Axzyw+MM9V12ZebQ/41UDirBzzZsLCNQ5yUQbH9Ep7Uy6u40D5RdRogjN/hLNzeduFGEH
OolST5y1WZSEgAQ==;EndpointSuffix=core.windows.net'
blob_service_client =
BlobServiceClient.from_connection_string(Connect_str)
container_name = str('videofb')
container_client =
blob_service_client.get_container_client(container_name)

#Connect to the PLC for read and write of Modbus values
#client = ModbusTcpClient('192.168.1.5')

x = 0
speed = 0
count = 0
posList = []
labelList = []
```

```
#Define Main window for tkinter and defining values
window = Tk()
value_inside = tk.StringVar(window)
current_var = tk.StringVar(window)
current_var.set("Select an object")
value_inside.set("Select a scenario")




#Send values to modbus which reset scenarios and recalls camera
def Modbus_Reset():
    client.write_registers(10,[0,0])
    client.write_registers(12,[0,0])
    client.write_registers(2,[0,0])
    client.write_registers(6,[1,0])
    client.write_registers(4,[0,0])

#Send values to modbus for emergency stop
def Modbus_ESTOP():
    client.write_registers(2,[1,0])
    client.write_registers(6,[0,0])

#Send values to modbus which changes the speed
def Set_Speed():
    label.config(text="Current speed: "+ entry.get(), font=
('Helvetica 9'))
    client.write_registers(8,[int(entry.get()),0])

#Get encoder data from PLC and process it to real time position on
GUI
def Position_number():
    encoderPos = (client.read_holding_registers(20).registers)
    position_bar['value'] = encoderPos[0]/20
    label2.config(text="Current position: "+ str(encoderPos[0]),
font= ('Helvetica 9'))
    window.after(100, Position_number) #Makes an infite loop to
update position bar

#Gets VideoFile from Scenario 1 and Finds correct path to upload to
```

```
Cloud Drive
def UploadVid_azure():
    local_path = r"C:\Users\User3\Desktop\HVIT\yolov5"
    local_name = strftime("%Y-%d-%b-%H%M", localtime()) + ".mp4"
    local_file_name ="VideoFile.mp4"
    upload_file_path = os.path.join(local_path, local_file_name)
    blob_client =
blob_service_client.get_blob_client(container=container_name,
blob=local_name)

    with open(upload_file_path, "rb") as data:
        blob_client.upload_blob(data)

    shutil.move(r'C:\Users\User3\Desktop\HVIT\yolov5\VideoFile.mp4' ,
r"C:/Users/User3/Desktop/HVIT/yolov5/VideoFeedBack/VideoFile.mp4")
    os.chdir(r"C:\Users\User3\Desktop\HVIT\yolov5\VideoFeedBack")
    os.rename(local_file_name, local_name)

#Gets ImageFile from Scenario 2 and Finds correct path to upload to
Cloud Drive
def UploadPic_azure():
    path_remove = r'C:\Users\User3\Desktop\HVIT\yolov5'
    local_picturepath =
r"C:\Users\User3\Desktop\HVIT\yolov5\PictureFeedBack"
    for r,d,f in os.walk(local_picturepath):
        if f:
            for file in f:
                file_path_on_azure =
os.path.join(r,file).replace(path_remove,"")
                file_path_on_local = os.path.join(r,file)

                blob_client =
container_client.get_blob_client(file_path_on_azure)

                with open(file_path_on_local, 'rb') as data:
                    blob_client.upload_blob(data)

    shutil.rmtree(local_picturepath)
    os.mkdir(local_picturepath)

#Forget labels and entries in the GUI for Scenario 1
```

```
def Forget_S1():
    entry2.place_forget()
    entry3.place_forget()
    label3.place_forget()
    label4.place_forget()

#Forget labels and entries in the GUI for Scenario 2
def Forget_S2():
    entry2.place_forget()
    entry3.place_forget()
    label3.place_forget()
    label4.place_forget()

#Forget labels and entries in the GUI for Scenario 3
def Forget_S3():
    dropObj.place_forget()

#Executes Scenario 1
def Scenario_1():
    Start_time = entry2.get()
    Interval = entry3.get()
    Interval = Interval.split(":")
    intervalTime = ['']
    intervalTime[0] = int(Interval[0])*60 + int(Interval[1])

    #opens camera stream and records
    cap =
cv2.VideoCapture('rtsp://root:ABB1234@192.168.1.3/axis-media/media.am
p')
    assert cap.isOpened()
    width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
    height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
    global record1
    record1 = 0

        #Sets when to start and at what interval
    if current_time == Start_time:
        client.write_registers(10,[1,0])
        client.write_registers(12,[intervalTime[0], 0])

        #Reads value from Modbus/PLC for when to start recording
```

```
    while client.read_holding_registers(22).registers[0] == 1:
        if record1 == 0:
            writer= cv2.VideoWriter("VideoFile.mp4",
cv2.VideoWriter_fourcc(*'DIVX'), 15, (width, height))

        ret, frame = cap.read()
        assert ret

        frame = cv2.resize(frame, None, fx = 1, fy = 1, interpolation
= cv2.INTER_AREA)
        #frame = cv2.resize(frame, (640,360))
        cv2.imshow('Input', frame)
        writer.write(frame)
        record1 = 1

        if cv2.waitKey(1) & 0xFF == 27:
                client.write_registers(4,[0,0])
                client.write_registers(6,[1,0])
                client.write_registers(8,[5000,0])
                break

    if record1 == 1:
        writer.release()
        record1 = 0
        UploadVid_azure()
        os.chdir(r"C:\Users\User3\Desktop\HVIT\yolov5") #Stops
recording and upload in a temp folder

    client.write_registers(4,[0,0])
    client.write_registers(6,[1,0])
    client.write_registers(8,[5000,0])
    cap.release()
    cv2.destroyAllWindows()

    window.after(300, Scenario_1)


#Executes Scenario 2
def Scenario_2():
    Start_time = entry2.get()
    Interval = entry3.get()
```

```
    Interval = Interval.split(":")
    intervalTime = ['']
    intervalTime[0] = int(Interval[0])*60 + int(Interval[1])


    global record
    record2 = 0

    if current_time == Start_time:
        client.write_registers(10,[1,0])
        client.write_registers(12,[intervalTime[0], 0])

    while client.read_holding_registers(24).registers[0] == 1:
        record2 = 1
        detector()  #Opens camera with Object Detection

    if record2 == 1:
            UploadPic_azure()
            record2 = 0

    window.after(300, Scenario_2)

#Executes Scenario 3
def Scenario_3():
    client.write_registers(16,[int(posLoad[dropObj.current()]),0])
    cap =
cv2.VideoCapture('rtsp://root:ABB1234@192.168.1.3/axis-media/media.am
p')
    assert cap.isOpened()

    while True:

        ret, frame = cap.read()
        assert ret

        frame = cv2.resize(frame, None, fx = 1, fy = 1, interpolation
= cv2.INTER_AREA)
        cv2.imshow('Input', frame)
        if cv2.waitKey(1) & 0xFF == 27:
                client.write_registers(4,[0,0])
                client.write_registers(6,[1,0])
```

```
                client.write_registers(8,[5000,0])
                break
    cap.release()
    cv2.destroyAllWindows()

#If button is clicked check which Scenario is selected and executes
#a few modbus changes in advance before executing the Scenario 1
def Scenario_start():
    #Execute Scenario 1 on button press
    if value_inside.get() == "Scenario 1":
        client.write_registers(0,[1,0])
        client.write_registers(4,[1,0])
        client.write_registers(6,[1,0])
        client.write_registers(10,[0,0])
        client.write_registers(12,[0,0])

        Start_time = ''
        Interval = ''
        intervalTime = ['']

        Scenario_1()

    #Execute Scenario 2 on button press
    elif value_inside.get() == "Scenario 2":
        client.write_registers(0,[2,0])
        client.write_registers(4,[1,0])
        client.write_registers(6,[1,0])
        client.write_registers(10,[0,0])
        client.write_registers(12,[0,0])

        Start_time = ''
        Interval = ''
        intervalTime = ['']

        open('labelList.txt', 'w').close()
        open('posList.txt', 'w').close()

        Scenario_2()

    #Execute Scenario 3 on button press
    elif value_inside.get() == "Scenario 3":
```

```
        client.write_registers(0,[3,0])
        client.write_registers(4,[1,0])
        client.write_registers(6,[1,0])
        client.write_registers(10,[0,0])
        client.write_registers(12,[0,0])

        Scenario_3()

    else:
        print('Error')  #Error if something went wrong

#A loop to update which scenario is selected and which
#labes and entries need to be placed for the right scenario
def Option_widgets():
    #Place two entries and two labels and update local time to detect
when to start
    #And for get other labels and entries
    if value_inside.get() == "Scenario 1":
        global current_time
        current_time = strftime("%H:%M", localtime())
        entry2.place(x=50, y=140)
        entry3.place(x=50, y=180)
        label3.place(x=48, y=117)
        label4.place(x=48, y=157)
        Forget_S3()

    #Place two entries and two labels and update local time to detect
when to start
    #And for get other labels and entries
    elif value_inside.get() == "Scenario 2":
        current_time = strftime("%H:%M", localtime())
        entry2.place(x=50, y=140)
        entry3.place(x=50, y=180)
        label3.place(x=48, y=117)
        label4.place(x=48, y=157)
        Forget_S3()

    #Forgets labels and entries, preload information from txt.file
    elif value_inside.get() == "Scenario 3":

        labelLoad = np.loadtxt('labelList.txt', delimiter = '\t',
```

```
dtype = 'str')
        posLoad = np.loadtxt('posList.txt', delimiter = '\t')
        dropObj['value']=labelLoad

        dropObj['state'] = 'readonly'
        dropObj.place(x=125, y=100)
        Forget_S1()
        Forget_S2()

    else:
        Forget_S1()
        Forget_S2()
        Forget_S3()
        #Forget all if error
    window.after(300, Option_widgets) #Command to loop

detector = remoteInspection(capture_index=0, model_name='best75e.pt')

#Place position bar on GUI main window
position_bar = ttk.Progressbar(window, orient='horizontal',
length=300, mode='indeterminate')
position_bar.place(x=50, y=30)

#Define Reset button
btn=Button(window, text="Reset", width=15, command =
Modbus_Reset).place(x=280, y=340)

#Define Emergency stop button
btn2=Button(window, text="Emergency Stop", width=15, command =
Modbus_ESTOP).place(x=280, y=300)

#Define set speed button
btn3=Button(window, text= "Set Speed", command =
Set_Speed).place(x=20, y=330)

#Define scenario start button
btn4=Button(window, text= "Start Scenario", width=15, command =
Scenario_start).place(x=140, y=250)

#List option and place dropdown menu to select from
Options = ["Scenario 1", "Scenario 2", "Scenario 3"]
```

```
dropdown = OptionMenu(window, value_inside, *Options).place(x=145,
y=65)

#Define dropdown meny for Scenario 3
dropObj = ttk.Combobox(window, textvariable=current_var)

#Define entries to write values
entry= Entry(window, width= 10) #Box for speed
entry.place(x=20, y=300)
entry2 = Entry(window, width= 10) #Box for Start time
entry3 = Entry(window, width= 10) #Box for Interval Time

#Defines label for speed
label = Label(window, text="Current speed:", font=('Helvetica 9'))
label.place(x=19, y=270)

#Defines label for position bar
label2 = Label(window, text="Current position:" , font=('Helvetica
9'))
label2.place(x=50,y=8)

#Defines label for start and interval time
label3 = Label(window, text="Start Time (HH:MM)",font=('Helvetica
9'))
label4 = Label(window, text="Set Interval (HH:MM)",font=('Helvetica
9'))

#Define message to stop camera and recall
label5 = Label(window, text="ESC to exit camera and recall",
font=('Helcetiva 12'))
label5.place(x=100, y=220)

#Save information from Scenario 2 to txt file
labelLoad = np.loadtxt('labelList.txt', delimiter = '\t', dtype =
'str')
posLoad = np.loadtxt('posList.txt', delimiter = '\t')
dropObj['value']=labelLoad

#Runs loop for Position bar and dropdown menu
threading.Thread(target=Position_number, args=(1,))
Option_widgets()
```

```
window.title('Remote inspection of unmanned installation') #Title of
GUI
window.geometry("400x400") #Size of GUI
window.minsize(400, 400)
window.maxsize(400, 400)
window.iconbitmap('ivitty.ico')
window.mainloop() #Executes main window
```

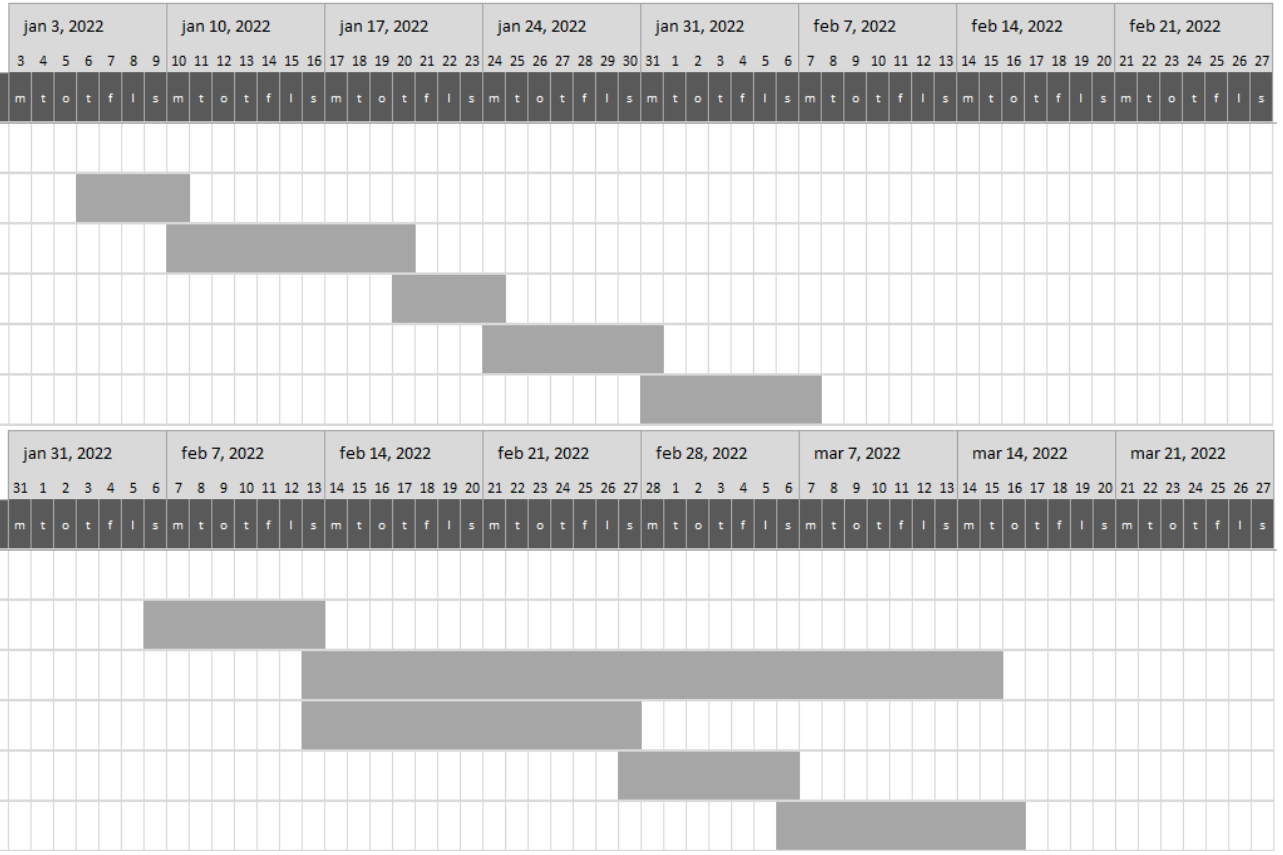
7.11 Gantt diagram

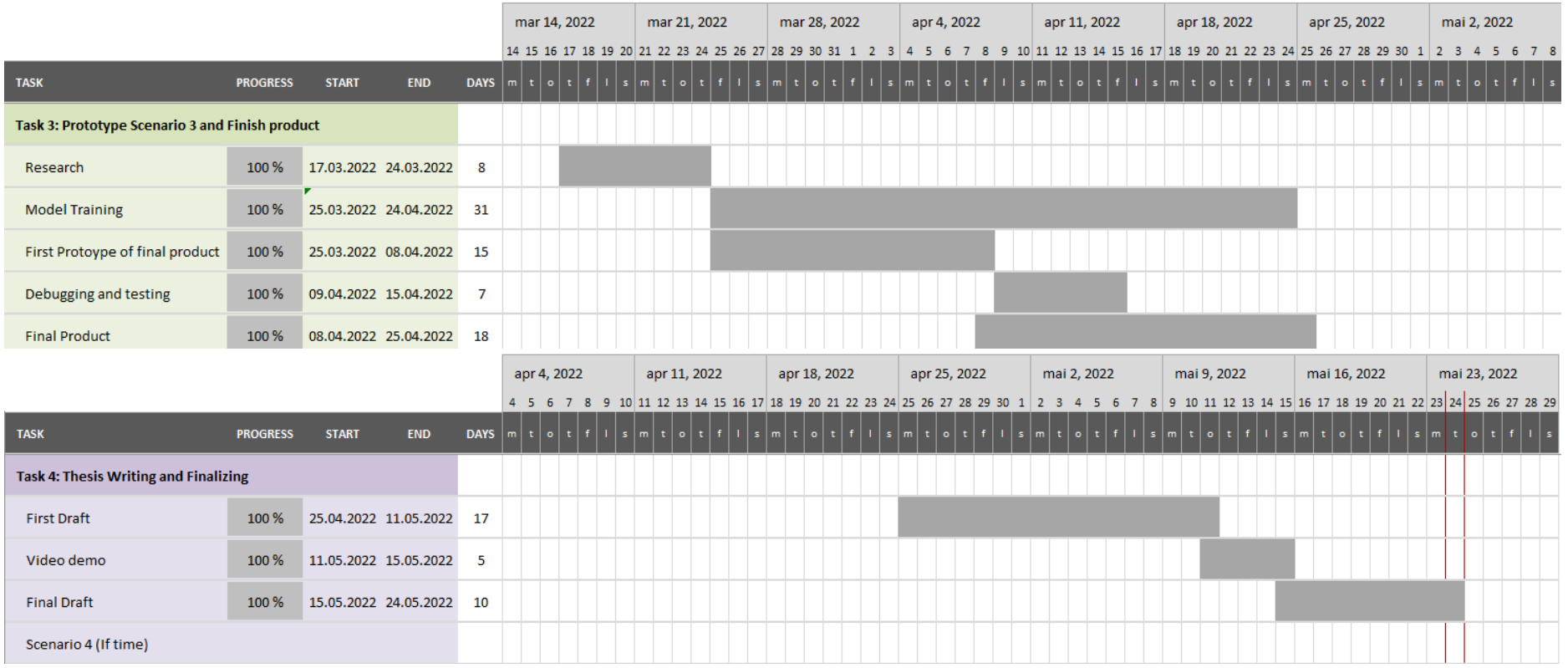
Unmanned Inspection of Remote Installations

Henrik Westgaard
Vebjørn Berstad
Ivar Øksendal
Tommy Tran

Project Start: 06.01.2022
Display Week: 1

TASK	PROGRESS	START	END	DAYS
Task 1: Background and Research				
Read previous work	100 %	06.01.2022	10.01.2022	5
Research	100 %	10.01.2022	20.01.2022	11
Risk assessment	100 %	20.01.2022	24.01.2022	5
Limitations of the system	100 %	24.01.2022	31.01.2022	8
Propose new solutions	100 %	31.01.2022	07.02.2022	8
Task 2: Prototype Scenario 1 and 2				
Research	100 %	06.02.2022	13.02.2022	8
Data gathering	100 %	13.02.2022	15.03.2022	31
First Prototype	100 %	13.02.2022	27.02.2022	15
Debugging and testing	100 %	27.02.2022	06.03.2022	8
Finished Scenario Prototype	100 %	06.03.2022	16.03.2022	11




mar 14, 2022
mar 21, 2022
mar 28, 2022
apr 4, 2022
apr 11, 2022
apr 18, 2022
apr 25, 2022
mai 2, 2022
TASK
PROGRESS
START
END
DAYS
Task 3: Prototype Scenario 3 and Finish product
Research
100 %
17.03.2022
24.03.2022
8
Model Training
100 %
25.03.2022
24.04.2022
31
First Protoype of final product
100 %
25.03.2022
08.04.2022
15
Debugging and testing
100 %
09.04.2022
15.04.2022
7
Final Product
100 %
08.04.2022
25.04.2022
18
apr 4, 2022
apr 11, 2022
apr 18, 2022
apr 25, 2022
mai 2, 2022
mai 9, 2022
mai 16, 2022
mai 23, 2022
TASK
PROGRESS
START
END
DAYS
Task 4: Thesis Writing and Finalizing
First Draft
100 %
25.04.2022
11.05.2022
17
Video demo
100 %
11.05.2022
15.05.2022
5
Final Draft
100 %
15.05.2022
24.05.2022
10
Scenario 4 (If time)

8 References

Tjernes, S., Listaul, M., & Mørk Madsen, T. (2021). Unmanned inspection of remote installations [Bachelor, Oslo Metropolitan University].

ABB, A. C. (2017). *AXIS M1065-L Network Camera User Manual*. https://www.axis.com/dam/public/92/ce/2b/axis-m1065-l--user-manual-prior-to-720-en-US-107840.pdf

Festo. (2015). *Toothed belt and spindle axes ELGA*. Retrieved from https://www.festo.com/net/hu_hu/SupportPortal/Files/443437/PSIplus_ELGA_en_V07_M.pdf

Festo. (2016). *Servo motor EMMT-AS-80: Datasheet.* Retrieved from https://www.festo.com/us/en/a/download-document/datasheet/5255426

Festo. (2016). *Toothed belt axis: Datasheet.* Retrieved from https://www.festo.com/cat/xdki/data/doc_ENGB/PDF/EN/ELGA-TB_EN.PDF

ABB. (2022). *AC500*. https://new.abb.com/plc/programmable-logic-controllers-plcs/ac500

Autopilot. (2011). *File:Qr-1.svg, File:Qr-2.svg, File:Qr-3.svg*. CC BY-SA 3.0. https://en.wikipedia.org/wiki/QR_code

Chollet, F. (2021). *Deep learning with Python*. Simon and Schuster.

Srinath, K. (2017). "Python–the fastest growing programming language." International Research Journal of Engineering and Technology (IRJET) 4(12): 354-357.

Kasper-Eulaers, M., et al. (2021). "Short Communication: Detecting Heavy Goods Vehicles in Rest Areas in Winter Conditions Using YOLOv5." Algorithms 14(4): 114. https://doi.org/10.3390/a14040114

Python.org (n.d.) *Download page.* Retrieved from https://www.python.org/downloads/

OpenCV (n.d.) *About page.* Retrieved from https://opencv.org/about/

Brownlee, J. (2018). "What is the Difference Between a Batch and an Epoch in a Neural Network." Machine Learning Mastery 20. https://deeplearning.lipingyang.org/wp-content/uploads/2018/07/What-is-the-Difference-Between-a-Batch-and-an-Epoch-in-a-Neural-Network_.pdf

Joseph Redmon. (2020). *YOLOv5 Documentation*. Ultralytics. Retrieved from https://docs.ultralytics.com/

Kang, L., Wei, Y., Jiang, J., & Xie, Y. (2019). Robust Cylindrical Panorama Stitching for Low-Texture Scenes Based on Image Alignment Using Deep Learning and Iterative Optimization. *Sensors (Basel)*, *19*(23). https://doi.org/10.3390/s19235310

Parziale, L., et al. (2006). "TCP/IP tutorial and technical overview."

Xu, Y. and R. Goodacre (2018). "On splitting training and validation set: a comparative study of cross-validation, bootstrap and systematic sampling for estimating the generalization performance of supervised learning." Journal of analysis and testing 2(3): 249-262.

DeVries, T. and G. W. Taylor (2017). "Dataset augmentation in feature space." arXiv preprint arXiv:1702.05538.

Bochkovskiy, A., et al. (2020). "Yolov4: Optimal speed and accuracy of object detection." arXiv preprint arXiv:2004.10934.

Weerasinghe, C., Nilsson, M., Lichman, S., & Kharitonenko, I. (2004). Digital zoom camera with image sharpening and suppression. IEEE Transactions on Consumer Electronics, 50(3), 777-786.

A. Amini et al. "Spatial Uncertainty Sampling for End-to-End Control". NeurIPS Bayesian Deep Learning 2018

Merriam-Webster. (n.d.). Network. In Merriam-Webster.com dictionary. Retrieved May 24, 2022, from https://www.merriam-webster.com/dictionary/network

Schneider, E. (2021, 09.29.2021). *What is Modbus and How does it work?* https://www.se.com/us/en/faqs/FA168406/

What is computer networking? (n.d.). Cisco. Retrieved May 24, 2022, from https://www.cisco.com/c/en/us/solutions/enterprise-networks/what-is-computer-networking.html

What is PROFINET. (n.d.). Retrieved May 24, 2022, from https://us.profinet.com/profinet-explained/

Tarang Shah (2017). "About Train, Validation and Test Sets in Machine Learning." Towards Data Science. Retrieved from https://towardsdatascience.com/train-validation-and-test-sets-72cb40cba9e7

What is Modbus and How does it work? (n.d.). Schneider Electric United States. Retrieved May 24, 2022, from https://www.se.com/us/en/faqs/FA168406/

Cisco (2005). *TCP/IP Overview*. Cisco. Retrieved from https://www.cisco.com/c/en/us/support/docs/ip/routing-information-protocol-rip/13769-5.html#tcp

Fred Oh. (2012, September 10). *What is CUDA*. NVIDIA Blog. https://blogs.nvidia.com/blog/2012/09/10/what-is-cuda-2/

Bolton, W. (2011). *Programmable logic controllers*. Elsevier.

niconielsen32. (n.d.). *ComputerVision/yoloCustomObjectDetection.py at fbb6d17558193c43d8bea34a0d5010b6b8b30e66 · niconielsen32/ComputerVision*. GitHub. Retrieved May 24, 2022, from https://github.com/niconielsen32/ComputerVision/blob/fbb6d17558193c43d8bea34a0d5010b6b8b30e66/yoloCustomObjectDetection.py

Infrared Light - An overview. (n.d.). ScienceDirect Topics. Retrieved May 25, 2022, from https://www.sciencedirect.com/topics/engineering/infrared-light

Radiant Vision Systems. (2019, October 22). *Near-Infrared (NIR) light sources for 3D facial recognition*. AZoOptics.Com. https://www.azooptics.com/Article.aspx?ArticleID=1666

Lens Calculator. (n.d.). *Lens calculator*. Retrieved May 25, 2022, from https://www.jvsg.com/calculators/cctv-lens-calculator/