EtherCAT-enabled depth camera for industrial applications

DIPLOMARBEIT

Ausgeführt zum Zwecke der Erlangung des akademischen Grades eines
Diplom-Ingenieurs (Dipl.-Ing.)

unter der Leitung von

Univ.-Prof. Dr.sc.techn. Georg Schitter
Proj.-Ass. Dipl.-Ing. Peter Gsellmann

eingereicht an der
Technischen Universität Wien
Fakultät für Elektrotechnik und Informationstechnik
Institut für Automatisierungs- und Regelungstechnik

von
Tobias Steinegger
Matrikelnummer: 01326489

Wien, im Jänner 2024

Technische Universität Wien
Karlsplatz 13, 1040 Wien, Österreich

Acknowledgement

First, I would like to thank my advisor Univ.-Prof. Dipl.-Ing. Dr.sc.techn. Georg Schitter for the inspiring discussion and the possibility to conduct this diploma thesis. Furthermore, I am deeply grateful to Dipl.-Ing. Peter Gsellmann for his unwavering support, guidance, and invaluable feedback throughout the research process. Not only did his expertise and encouragement contribute significantly to the creation of this work, but I can also say that I made a new friend during the process of creating it. I am also very grateful to the OSL-gang who have shared their knowledge, ideas, and experiences. The time we spent together always motivated and inspired me throughout the thesis.

To all my friends, who are not individually named here, but who have been an integral part of my life, thank you for sticking with me through thick and thin. Your friendship has made this academic endeavor all the more fulfilling.

A big thank you to Aunt Christi and Uncle Knut for their constant encouragement and support. They always had my back and took care of me, especially in the hardest times. I am also grateful to my sister Kathrin and her husband Andreas for their understanding, motivation and emotional support. Their belief in my abilities was a driving force in overcoming the biggest challenges. Finally, I owe my parents, Monika and Werner, a debt of gratitude that cannot be expressed in words. Their tireless support, their sacrifices and their love were the foundation on which my academic endeavors were born and grew. Without their help, my studies and therefore this Master's thesis would not have been possible. Thank you from the bottom of my heart.

Abstract

The increasing importance of 3D vision in the realm of industrial automation, driven by the challenges towards Industry 4.0, demands a fieldbus-capable depth camera. This master's thesis evaluates the integration of depth information from a 3D camera via the EtherCAT fieldbus into a programmable logic controller (PLC). Additionally, the developed fieldbus-capable depth camera system is integrated into a collaborative industrial robot system, subject to the DIN ISO/TS 15066 standard, to define conditions and create comparability with the performance of existing systems in the collaboration space.

The proposed EtherCAT-capable camera module consists of three main components: a depth camera, a single-board computer, and an EtherCAT Slave Controller (ESC). Thereby, a depth image segmentation method enables the transmission of high-resolution depth images via EtherCAT. In the human-robot collaboration application, a five-axis robotic arm operates within the field of view of the depth camera, monitoring the minimum distance between humans and robots. Safety stops are executed to avoid collisions. The detection time is then utilized as a benchmark for comparing the system's performance with state-of-the-art technologies.

The implemented camera system allows for data transmission rates exceeding 25 Mbit s^{-1}. Through the utilization of various tested data reduction methods, detection times of less than 25 ms are realized. The camera system employed in a human-robot collaboration application, coupled with data reduction techniques, results in a reduced minimal safety distance, surpassing existing state-of-the-art applications in the collaboration space by 75 mm.

Kurzfassung

Die wachsende Bedeutung der 3D-Vision im Bereich der industriellen Automatisierung, getrieben durch die Herausforderungen von Industrie 4.0, erfordert eine Feldbus-fähige Tiefenkamera. Diese Masterarbeit bewertet die Integration von Tiefeninformationen aus einer 3D-Kamera über den EtherCAT-Feldbus in einer speicherprogrammierbaren Steuerung (SPS). Darüber hinaus wird das entwickelte Feldbus-fähige Tiefenkamerasystem in eine kollaborative Industrieroboteranwendung integriert, das dem Standard DIN ISO/TS 15066 unterliegt, um Bedingungen zu definieren und Vergleichbarkeit mit der Leistungsfähigkeit bestehender Systeme im Kollaborationsraum zu schaffen.

Das vorgeschlagene EtherCAT-fähige Kamerasystem besteht aus drei Hauptkomponenten: einer Tiefenkamera, einem Einplatinencomputer und einem EtherCAT Slave Controller (ESC). Dabei ermöglicht eine Tiefenbildsegmentierungsmethode die Übertragung hochauflösender Tiefenbilder mittels EtherCAT. In der Mensch-Roboter-Kollaborationsanwendung aktuiert ein Fünf-Achs-Industrieroboter innerhalb des Sichtfelds der Tiefenkamera und überwacht den minimalen Abstand zwischen Menschen und Roboter. Die Erkennungsszeit wird anschließend als Maßstab verwendet, um die Leistung des Systems mit modernsten Technologien zu vergleichen.

Das implementierte Kamerasystem ermöglicht Datenübertragungsraten von über 25 Mbit s^{-1}. Durch die Anwendung verschiedener Datenreduktionsmethoden werden Erkennungszeiten von weniger als 25 ms realisiert. Das in einer Mensch-Roboter-Kollaborations Anwendung implementierte Kamerasystem, in Verbindung mit Datenkomprimierungsmethoden, führt zu einer reduzierten minimalen Sicherheitsdistanz, die bestehende Anwendungen im Kollaborationsraum um 75 mm übertrifft.

Contents

List of Tables

List of Figures

Acronyms

µC	Microcontroller.
ADS	Automation Device Specification.
API	Application Programming Interface.
DMA	Direct Memory Access.
DPRAM	Dual Port Random Access Memory.
EEPROM	Electrically Erasable Programmable Read-Only Memory.
ENI	EtherCAT Network Information.
EPU	EtherCAT Processing Unit.
ESC	EtherCAT Slave Controller.
ESI	EtherCAT Slave Information.
ESM	EtherCAT State Machine.
EtherCAT	Ethernet for Control Automation Technology.

FMMU	Fieldbus Memory Management Unit.
FoV	Field of View.
GPIO	General Purpose Input/Output.
MISO	Main In and Subnode Out.
MOSI	Main Out and Subnode In.
PDI	Process Data Interface.
PDO	Process Data Objects.
PHY	Physical Interface.
PLC	Programmable Logic Controller.
SPI	Serial Peripheral Interface.
SSI	Slave Information Interface.
ToF	Time of Flight.
TwinCAT3	The Windows Control and Automation Technology 3.
XML	Extensible Markup Language.

CHAPTER 1

Introduction

One of the most current topics in industrial production nowadays is Industry 4.0 or the ”Industrial Internet of Things” [1]. Industry 4.0 includes areas such as automation, big data, cloud computing, machine learning, and image processing, which have led to significant changes in existing production processes. For the latter, industrial applications are not just limited to classic RGB or mono images. The integration of depth information into systems, the so-called 3D vision [2], is also part of Industry 4.0. 3D vision comprises the acquisition, evaluation, and processing of sensor-based 3D information for the control of mechanical systems or processes. In this way, 3D vision enables new possibilities and solutions for existing problems that could not be solved with classic 2D vision approaches.

1.1 Motivation

The range of applications that rely on 3D vision has grown significantly over the last few years [3] and it’s potential is far from exhausted. An example of the use of 3D vision is bin picking [4, 5]. Bin picking is the robot-based separation of randomly provided objects in a bin. It is a core problem in computer vision and robotics, where depth data are used to guarantee a precise gripping process by means of pose estimation. Pose estimation based on 3D vision will also be used in depalletization [6, 7]. During depalletization, the pallets loaded with cartons are unloaded one by one. These systems have increased steadily as a result of the growth of sectors such as logistics, warehousing, and supply chains. Another area of industrial 3D vision application is inspection [8]. Inspection systems are often used in industrial quality control, for example, to check certain characteristics of products. The inspection covers both functional aspects of a product, e.g. whether all components are fitted to an assembly, as well as aesthetic properties, e.g. checking for scratches on a polished surface.

Collaborative applications involving humans and robots also necessitate the integration of 3D vision. Human-robot collaboration workspaces are becoming increasingly important to improve work efficiency, flexibility, and overall productivity in production facilities [9, 10]. State-of-the-art approaches in the field of collision avoidance use depth cameras to monitor the environment around robotic systems, enabling a safe and collaborative working environment.

There are different types of depth cameras. Stereo vision cameras, for example, use the human depth vision principle by means of stereo disparity of two 2D images from different positions [11]. The depth perception of stereo vision cameras is based on a geometric approach called triangulation. Structured light cameras use a light source, such as a LED or a laser, to cast a narrow light pattern onto the surface and detect distortions of the illuminated patterns [12]. Based on the distortion of the pattern, the camera geometrically reconstructs a 3D image.

Time of Flight (ToF) is another technology for the area-wide acquisition of depth data [13][11]. A ToF camera illuminates the scene with a modulated infrared light source and observes the reflected light. The phase shift between the emitted light and the reflection is measured and converted to distance.

The interface for depth cameras is usually USB or Ethernet. However, the latest standard for data exchange in industrial automation between the field and control level is the fieldbus [14, 15]. A fieldbus is a bus system that connects sensors, actuators, electric motors, switches, valves, etc. to a control unit, such as a Programmable Logic Controller (PLC), to exchange information. This means that a data connection is established from a PLC to various device participants through bus lines [16].

Today's demands on various field components in Industry 4.0 require modularity and versatility. This aims to improve efficiency, productivity, and connectivity across different elements of the industrial landscape. It therefore stands to reason that camera systems for industrial plants should also be modular and connectable [17].

1.2 Research goals

The main objectives of this research are two main topics: The first task is to develop a methodology to make a depth camera fieldbus compatible, as shown in Figure 1.1. Subsequently, the second task is the implementation of the fieldbus-capable depth camera in an industrial application to create comparability with existing state-of-the-art camera systems.

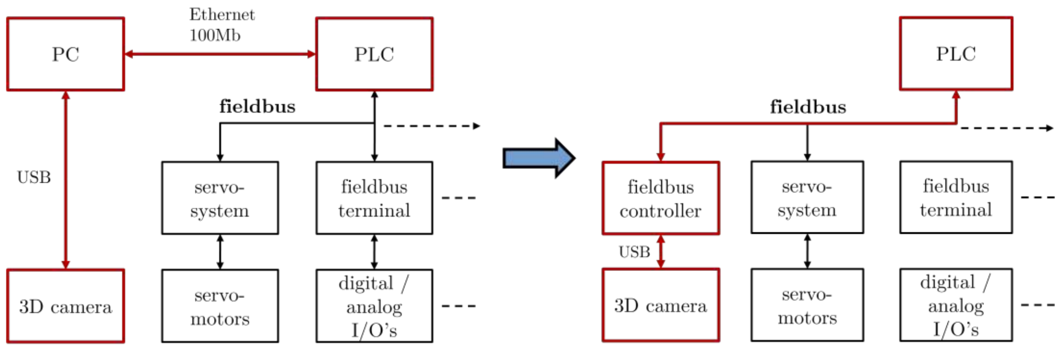


Figure 1.1: The schematic overview on the left shows a typical industrial application involving a 3D camera [18, 19]. All components at field level communicate with the PLC via fieldbuses. The 3D camera uses a separate interface for data exchange. The depth data is then exchanged with the PLC via Ethernet. The schematic overview on the right shows the estimated approach, which aims to integrate the 3D camera into the fieldbus.

High-resolution depth cameras generate depth images with a data volume of several kilobytes to several megabytes [20, 21]. These cameras typically work with frame rates of between 30 fps and 60 fps. In order to achieve efficient data transmission between the camera system and the PLC, a fieldbus is required that guarantees a data transmission rate that matches the specifications of the depth camera. By integrating the system components into a fieldbus that supports such high data rates, high-resolution depth images can be embedded in the application.

The necessary criteria for human-robot collaboration applications are defined in the DIN ISO/TS 15066 standard [22]. This makes it possible to establish the comparability of existing systems in the collaboration space under defined framework conditions. In particular, this standard specifies the minimum safety distance between robots and humans. The detection time of the sensor system plays a decisive role here, as shorter detection times lead to shorter safety distances in the collaborative workspace. This optimization increases productivity and efficiency in the execution of processes. An existing system in the field of human-robot collaboration is from Veo Robotics [23]. Their Veo FreeMove safety system is for industrial work cells with human-robot collaboration.

The system monitors work cells in 3D and implements dynamic speed and distance monitoring in accordance with DIN ISO/TS 15066, which enables safe interaction between humans and robots. The parameters defined here serve as a benchmark for the desired depth camera system.

For the implementation of the application, low latency in combination with a high data rate are therefore among the most important specifications of the fieldbus [22]. Ethernet for Control Automation Technology (EtherCAT) is a real-time Ethernet, disclosed in the IEC 61158 standard [16], that meets these requirements.

As already mentioned, there are different types of depth cameras. ToF depth cameras are characterized in particular by their high depth accuracy, resolution, and frame rate. Furthermore, ToF cameras are compact, lightweight, and relatively inexpensive compared to other camera systems. Although ToF depth cameras have poor depth determination in highly reflective or transparent materials as well as strongly illuminated environments, the advantages over other camera systems outweigh the disadvantages in a human robot collaborating application. For this reason, a ToF camera is preferred for implementation [11, 24].

The primary objective is to integrate a ToF depth camera into the EtherCAT fieldbus. The aim is to achieve a maximum data transfer rate with minimum latency. In addition, the EtherCAT-compatible depth camera is to be integrated into a human-robot collaboration application. The task of the human-robot collaboration application is to monitor the safety distance between the robot and humans and to initiate safety stops if necessary. The existing state-of-the-art system from Veo Robotics serves as a reference for evaluation.

1.3 Outline

In order to give more insight into the implementation of an EtherCAT-capable ToF depth camera, Chapter 2 of the thesis elaborates on the EtherCAT fieldbus and its components. In this context, a process data interface serves as the interface between the camera and the EtherCAT device. Subsequently, the indirect ToF technology is discussed. These topics are laying the groundwork for formulating two research questions. Chapter 3 is dedicated to describing the components used. In Chapter 4, the focus is on the implementation of the EtherCAT-enabled camera module and the realization of the human-robot collaboration application. Chapter 5 delves into the discussion of the results of the depth camera and the application, including possible methods for data reduction. Finally, Chapter 6 presents a conclusion and an outlook for potential further research directions.

CHAPTER 2

State of the Art

This chapter focuses on the state of the art technologies in camera-based industrial robotics. First, 3D vision systems are described, in particular Time of Flight (ToF) cameras. Afterwards the functional principle and implementation of the Ethernet for Control Automation Technology (EtherCAT) fieldbus are discussed. Here, the focus will shift towards the individual components of the EtherCAT fieldbus. Finally, an introduction to the Serial Peripheral Interface (SPI) is provided.

2.1 3D vision

In recent years, there has been notable growth in the field of industrial applications utilizing 3D vision. The array of applications is continuously expanding and has not yet reached its full potential [3]. 3D vision is the capture and use of three-dimensional information from target objects in applications [25]. The image of the target object in a 3D vision system is a three-dimensional point cloud with coordinates showing the position of each pixel in space. It simultaneously provides data in the X, Y, and Z planes, as well as rotation information (around each of the axes). Both on the camera and on the PC system, the 3D information can be calculated.

There are various methods for obtaining comprehensive 3D data, such as ToF principle, Structured Light, Stereo Vision, and Laser Triangulation. ToF camera technology is characterized by its compact size and low power consumption [11, 24]. ToF cameras also operate in a wide range of lighting conditions, from bright sunlight to complete darkness, making them suitable for a variety of environments. In addition to their accuracy and reliability, ToF cameras are also known for their fast response times. This is critical for applications that require fast response times, such as collision avoidance in robotic systems. For this reason, tof technology is a prerequisite when choosing a depth camera for use in the collaboration space.

2.1.1 Time of Flight (ToF)

ToF technology is used for a variety of applications, including robot navigation, 3D perception, people counting, and object detection [26, 27]. ToF distance sensors use the time it takes for photons to travel between two points to calculate the distance between the points. ToF technology is divided into two different methods, direct and indirect ToF. In both methods, the intensity and distance are measured simultaneously for each pixel in a scene.

Direct ToF sensors emit short light pulses [28]. These typically last only a few nanoseconds. The time it takes for a portion of the emitted light to return is measured. The principle of direct ToF measurement is simple, but places high demands on the light source at the transmitter, the image sensor at the receiver, and the synchronization and timing circuits. The transmitter must be able to generate short pulses, and the image sensor in the receiver must use high-sensitivity optical detection to detect weak optical signals. This results in a small number of sensor elements in one device, which is used for single or a few point measurement.

There are different approaches for modulating the light in an indirect ToF camera. A simple approach is to use continuous wave modulation. Indirect continuous wave ToF sensors emit continuous modulated light [29, 30]. The phase difference between emitted and reflected light is measured to calculate the distance to an object. This technology allows for higher depth resolution, the use of low-frequency light, and greater illumination duration. It is also relatively easy to scale up the process to sensor arrays with multiple pixels. This makes indirect ToF sensors suitable for high-resolution 3D cameras. The operating principle of an indirect ToF sensor is shown in Figure 2.1.

The continuous wave ToF camera measures the time difference t_d between the transmitted signal and the returning signal by estimating the phase offset $\phi = 2\pi f \cdot t_d$ between the fundamental waves with the modulation frequency f_{mod} of these two

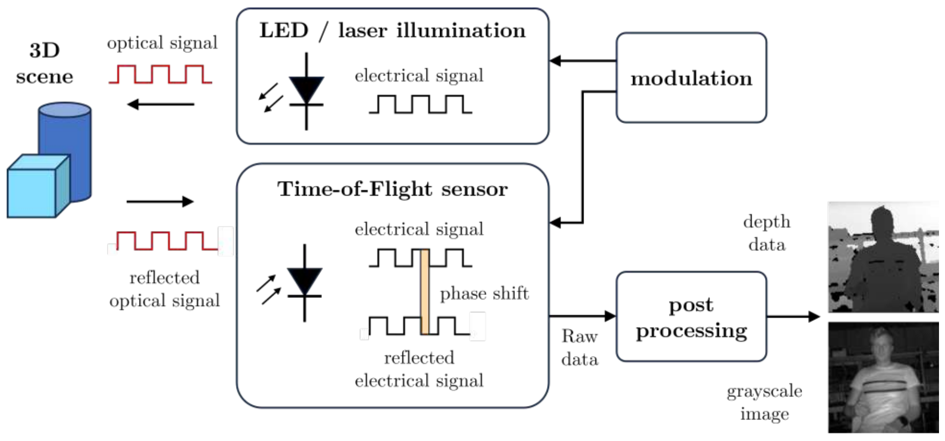


Figure 2.1: The working principle of an indirect ToF depth sensing system using continuous wave modulation [31]. A light source emits modulated light. When the light hits an object, it is reflected and detected by the ToF sensor. Based on the phase shift between the transmitted and received light, the distance of the object to the camera can be determined. Sensor arrays are used to generate depth images covering a bigger area in front of the camera. In addition to the depth image a gray image can be constructed, based on the received amplitudes of the reflected signals.

signals [32]. The depth d is calculated from the phase offset ϕ and the speed of light c:

$$d = \frac{c \cdot \phi}{4\pi f_{mod}}. \tag{2.1}$$

Phase measurement errors occur due to photon shot noise, readout circuit noise, and multipath interference [33]. The effect of these errors on the depth estimate can be reduced by using a high modulation frequency. A disadvantage of a high modulation frequency is that the clearly measurable range is shorter. This problem can be circumvented by using multiple modulation frequencies. The lowest modulation frequency provides a large range without ambiguity, but larger depth errors. Higher modulation frequencies are used to reduce depth errors. An example of this scheme with two different modulation frequencies is shown in Figure 2.2. The final depth estimate is calculated by weighting the phase estimates for the different modulation frequencies, assigning a higher weight to the higher modulation frequencies.

The depth (d), which denotes the distance between the camera and the depth point, can be converted to point cloud information. This information of a particular pixel represents its real-world coordinates (x,y,z) in space on a reference frame. Often applications use only the z-image map (depth map) instead of the full point cloud. The calculation of the point cloud requires knowledge of the intrinsics of the lens and distortion parameters. These parameters are estimated during the geometric calibration of the camera. In this process, the coordinates of each pixel *pcd* are calculated via the depth value of the pixel $d(i, j)$, where i and j are the indices for the row and column in the depth image, the focal lengths of the camera f_x, f_y and the optical centers c_x, c_y, respectively:

$$pcd(i,j) = \begin{cases} z = d(i,j) \\ x = \dfrac{(j - c_x) \cdot z}{f_x} \\ y = \dfrac{(j - c_y) \cdot z}{f_y} \end{cases} . \tag{2.2}$$

Depth processing can be done on the camera module itself or in a processor elsewhere in the system.

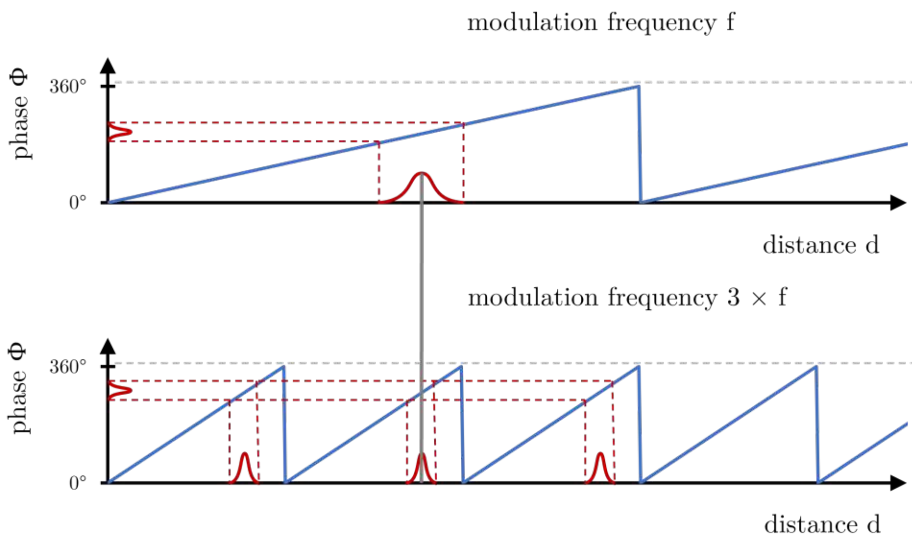


Figure 2.2: The effect of phase error on depth estimation [33]. The upper graph has a lower modulation frequency for depth calculation. This allows a higher range, but the phase error has a stronger effect on the depth calculation. The lower graph uses 3 times the frequency of the upper graph. This reduces the maximum range by a factor of 3. However, a more accurate depth calculation is possible here. When using several modulation frequencies in one camera system, high ranges can be achieved with high accuracy. However, this yields increased computational effort.

In order to process the data of the camera system in various applications, different transmission systems can be used. It must be ensured that the bandwidth of the system is sufficient for the amount of data produced by the camera. In order to be able to dimension the transmission medium, the required data transmission rate of the camera is therefore necessary. This can be calculated from the following camera parameters:

- Data transmission rate DTR,
- Frame rate FRR,
- Pixels per line (image width) W_{image},
- Pixels per row (image height) H_{image},
- Bits per pixel (depth resolution) R_{depth},

$$DTR = FRR \cdot W_{image} \cdot H_{image} \cdot R_{depth}. \tag{2.3}$$

Depth cameras often use interfaces such as USB or Ethernet for data transmission.

2.2 EtherCAT

The latest standard for data exchange in industrial automation between the field and control level is the fieldbus. One of the most advanced and fastest fieldbuses is EtherCAT. EtherCAT is a real-time Ethernet technology [34]. The EtherCAT protocol, which is disclosed in the IEC 61158 standard [16], operates on the principle of a master-slave system and is suitable for real-time hard and soft requirements in automation technology [35]. The EtherCAT operating principle is shown in Figure 2.3.

The telegram sent by the master passes through all nodes. In an EtherCAT fieldbus system, each slave reads the output data assigned for it from the frame and simultaneously writes its input data to the frame as the frame passes through the system. Only the EtherCAT master in a segment is allowed to actively send an EtherCAT frame. All other participants only forward the frames. This avoids unpredictable delays and guarantees the real-time capability of EtherCAT.

The master uses a standard Ethernet medium access controller without an additional communication processor. This means that a master can be installed on any hardware platform that provides an Ethernet port.

EtherCAT slaves use an EtherCAT Slave Controller (ESC) for processing. So the data processing is completely done in hardware, which makes the performance of the network deterministic. Telegrams are only delayed by hardware cycle times. If a frame reaches the last node of a segment, the slave detects an open port and sends the telegram back to the master.

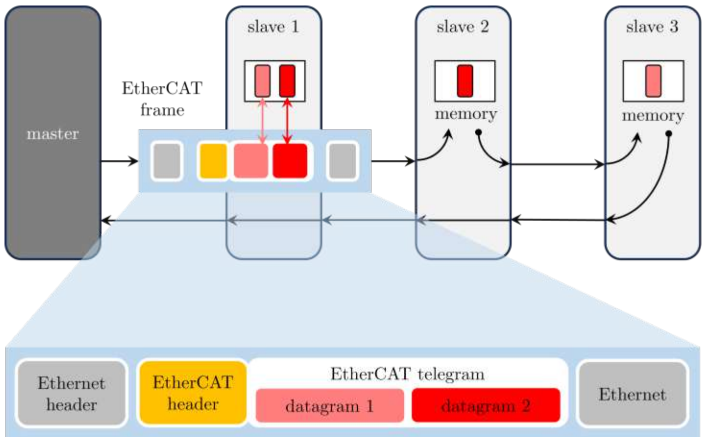


Figure 2.3: Illustration of an exemplary EtherCAT application [34]. The master sends an EtherCAT frame to the three slaves. The EtherCAT frame contains two datagrams, which results due to two different process mapping tasks in the application. Each slave reads it's associated output data in the respective datagram from the frame and writes its input data to the correct datagram while the telegram passes through each slave. Slave three is the last connected slave, whereby it recognizes an open port and thus sends the EtherCAT frame back to the master.

2.2.1 System Architecture

The basic EtherCAT network architecture is shown in Figure 2.4. The EtherCAT master and the EtherCAT slaves use a standard Ethernet port for communication. Different topology types such as line, tree, star and daisy chain are possible for EtherCAT networks. The network configuration information is stored in the EtherCAT Network Information (ENI) file. The ENI is created based on EtherCAT Slave Information (ESI) which are provided by the vendors for each device [36].

EtherCAT Slave Information (ESI)

An ESI serves as a configuration file in EtherCAT systems, in which the functionality and configuration parameters of EtherCAT slave devices are described in detail. ESI files, which are based on Extensible Markup Language (XML), are provided by the slave device manufacturer or vendor and contain information such as device type, supported mailbox protocols, synchronization settings, and mapping of its input and output process data (Process Data Objects (PDO)) that the master needs to configure and communicate with slaves [37].

EtherCAT Network Information (ENI)

An ENI serves as the central configuration file in EtherCAT systems and has the task of describing the network topology, the slave positioning and the process data structures for all EtherCAT slaves. Typically, this file is generated by a EtherCAT configuration tool, which defines the structure of the network by scanning it for devices or manually integrating the slave devices EtherCAT. In addition, the configuration tool analyzes information from the provided ESI files associated with each slave device. The ENI file, which is essential for the EtherCAT master functionality, enables the configuration and initialization of the network. It encapsulates all the relevant details that the master needs to communicate with each slave device in the network [37].

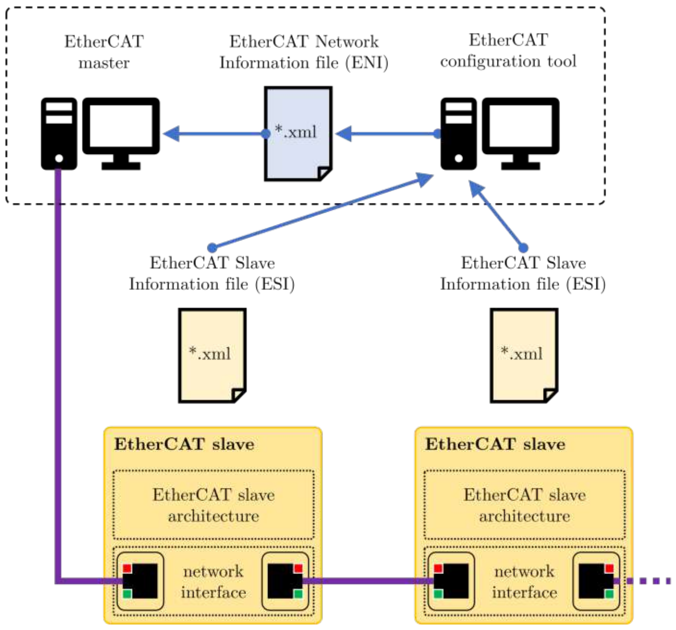

EtherCAT master
EtherCAT Network Information file (ENI)
EtherCAT configuration tool
*.xml
EtherCAT Slave Information file (ESI)
*.xml
EtherCAT Slave Information file (ESI)
*.xml
EtherCAT slave
EtherCAT slave architecture
network interface
EtherCAT slave
EtherCAT slave architecture
network interface

Figure 2.4: Graphical representation of the EtherCAT Network Architecture [36]. The architecture of each slave defined in the EtherCAT network is described in the ESI file created for this slave. An EtherCAT configuration tool summarizes this information in a single ENI File and makes it available to the EtherCAT master. This provides the master with all the necessary information to communicate with each device within the network.

2.2.2 Frame

EtherCAT uses standard Ethernet frames [38]. EtherCAT data are embedded in the data field of the Ethernet frame. The telegram is terminated by the Frame Check Sequence (FCS). FCS refers to a checksum that is added to a Ethernet frame to enable error detection.

The EtherCAT data contain an EtherCAT header and one or more EtherCAT datagrams. The structure of a complete EtherCAT frame can be seen in Figure 2.5. Up to 15 datagrams can be included in the EtherCAT data. If the whole Ethernet frame is smaller than 64 B, one to 32 padding bytes are inserted at the end of the EtherCAT data to to ensure the minimum length of the Ethernet frame. A datagram consists of a datagram header, the data to be read or written, and a working counter. The working counter is incremented when an EtherCAT node is successfully addressed and a read, write, or read and write operation is successfully executed. It is possible to specify a certain value for the working counter for each datagram, which is expected as soon as the telegram passes through all interconnected nodes. By comparing the expected operational counter value with the actual received value after the telegram has passed through all nodes, the master can check the successful processing of an EtherCAT datagram. The maximum amount of data is 1486 B. If more bytes have to be sent, multiple EtherCAT frames are necessary. The interpretation of the individual bits of the EtherCAT header and the datagram header are given in Table 2.1.

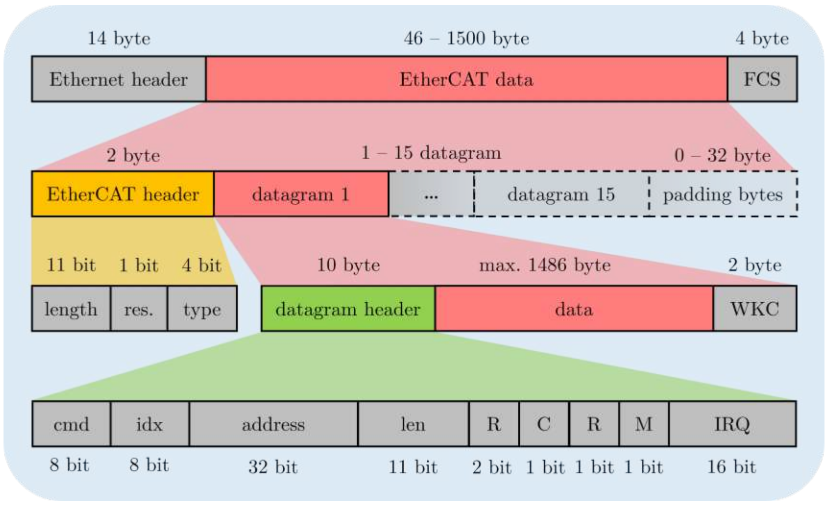


Figure 2.5: Structure of an EtherCAT frame [38]. The red marked area indicates the location of the transmitted data in the EtherCAT frame. The EtherCAT header, marked orange, and the datagram header, marked green, are described in more detail in Table 2.1.

The addressing area specified in the datagram header is separated into two addressing modes: Device addressing and logical addressing. Three device addressing modes are available: auto-increment addressing, configured station address, and broadcast.

Logical addressing supports the bit-wise assignment of data and reduces unnecessary communication contents during data processing. Each slave uses a mapping unit, the

Fieldbus Memory Management Unit (FMMU), to map data from the logical process data image to its local address space. For this, a 4 GB address space is available. The master configures the FMMUs of each slave during the start-up. Using the configuration information of its FMMUs, a slave knows which parts of the logical process data image should be assigned to which local address space. Each slave is assigned one or more addresses in this address space when the network starts up. By giving several slaves an address in the same area, these slaves can be addressed via a single datagram. Since the information on the desired data access is completely contained in the datagrams, the master can decide when to access which data. This allows it to control processes with different cycle times at the same time.

Table 2.1: EtherCAT header and the datagram header description [38].

EtherCAT Header		
Field	Data type	Description
Length	11 Bit	Length of the EtherCAT datagram
Res.	1 Bit	Reserved, set to 0
Type	4 Bit	Protocol type
Datagram Header		
Field	Data type	Description
Cmd	BYTE	Length of the EtherCAT datagram
Idx	BYTE	Reserved, set to 0
Address	BYTE[4]	Device addressing and logical addressing
Len	11 Bit	Length of the data of this datagram
R	3 Bit	Reserved, set to 0
C	1 Bit	Protocol type
M	1 Bit	Multiple EtherCAT datagrams: 0: Last EtherCAT datagram 1: A following EtherCAT datagram exists
IRQ	WORD	The EtherCAT event request registers of all slave devices are combined using a logical OR operation

2.2.3 Master

An EtherCAT master is a device that manages and controls an EtherCAT network [34, 37]. This is typically a PC, an embedded microprocessor, a Programmable Logic Controller (PLC) or a motion controller that communicates with and is responsible for controlling the devices connected to the network. An Ethernet port of an on-board

Ethernet controller or a standard network card serves as the interface between the control unit and the fieldbus. The EtherCAT master is the control point for the network. It is also responsible for managing the network clock and synchronization across the network, as well as error handling mechanisms. Furthermore, the EtherCAT master uses data from the ENI and ESI files to identify and understand the structure of the network and the capabilities of the connected slaves.

The Ethernet controller is connected to the master Direct Memory Access (DMA). This means that data transfer between the master and network memory does not require any CPU power. The power requirement of the master CPU is thus not determined by the EtherCAT connection, but by the desired master application. Thereby, the process image is already fully sorted, as the process data is not configured in the master but in the slaves. The peripheral devices insert their data at the corresponding position in the passing frame and read the data intended for them. When the frame with the input information is received again, the Ethernet controller can also copy it directly into the working memory of the computer by means of DMA. Thus, no active copying is required by the CPU.

2.2.4 Slave

An EtherCAT slave is a device that is integrated into an EtherCAT network and is controlled by an EtherCAT master [34, 36, 37]. Typically, EtherCAT slave devices include sensors, drives, actuators or other automation devices that are assigned to specific tasks within the system. To enable EtherCAT frame processing, these slave devices require an ESC. Each slave device in the network is automatically assigned a unique address, which is used for unique communication with the master. The functionality of a slave device is defined by the ESI file. The EtherCAT slave contains three main layers: the Physical Interface (PHY), the data link layer, and the application layer. The architecture of an EtherCAT slave can be seen in Figure 2.6.

The PHY and the network interface is defined in the Ethernet standard [39]. It contains components to process fieldbus signals, applies signals from the ESC to the network and forwards data from the network to the ESC. The network interface consists of three main components:

1. Plugs: Cable connectors such as RJ45 connectors or M12 D-code connectors can be used. EtherCAT cables are usually shielded twisted pair enhanced category 5 cables (CAT 5e STP) or better.

2. Magnetics: Pulse transformers are used for galvanic isolation between network and slave.

3. PHY: connects the link layer device (ESC) to a physical medium such as an optical fiber or copper cable. The hardware functions implemented in the chip are used to transmit and receive Ethernet frames.

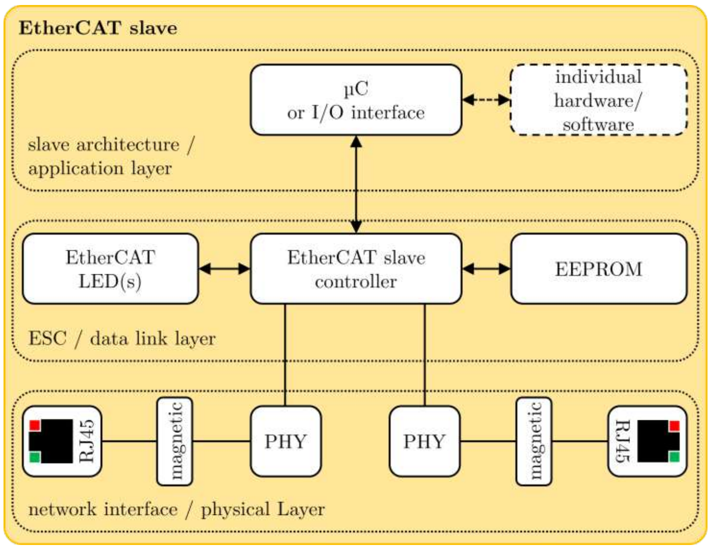


Figure 2.6: Block diagram of an ESC [36]. It consists of three layers: PHY, data link layer and the application layer. The PHY forwards the data from the network to the data link layer and makes the data from the data link layer available to the network. The data link layer manages the EtherCAT protocol in real-time. It serves as interface for the data exchange between the EtherCAT network and the local application. The application layer consists of hardware, communication software, and device-specific software to ensure the data exchange between the data link layer and the slave application. This can be realized for example by a µC.

The second level of an EtherCAT slave is the data link layer. This layer handles the EtherCAT protocol in real-time by processing the EtherCAT frames on the fly and providing the interface for data exchange between the EtherCAT master and the slave's local application controller via registers and a Dual Port Random Access Memory (DPRAM). It contains the ESC, an Electrically Erasable Programmable Read-Only Memory (EEPROM) and status LED‘s.

1. ESC: The ESC is a chip for EtherCAT communication. It is explained in more detail in Section 2.2.4

2. EEPROM: The EEPROM, also known as the Slave Information Interface (SSI), stores hardware configuration details for the ESC. These configuration parameters are loaded into the ESC's internal registers during its power-up sequence. This action activates the Process Data Interface (PDI) so that the DPRAM can interact

with the nearby host controller. The EEPROM can be modified by a configuration tool via EtherCAT using the information provided by the ESI file. If authorized access is granted, the µC can also read and write to the EEPROM. All interactions with the EEPROM are channeled through the ESC via the Inter-Integrated Circuit (I2C) data bus.

3. EtherCAT LEDs: LEDs provide information from the ESC and the application status.

The application layer consists of hardware, communication software, and device-specific software. A local µC can meet the mentioned requirements. This controller handles the EtherCAT State Machine (ESM), process data exchange with the slave application, and mailbox-based protocols for acyclic data exchange. The µC performance depends solely on the device application, not on the EtherCAT communication.

EtherCAT Slave Controller (ESC)

The ESC usually uses an Application-Specific Integrated Circuit (ASIC) or a Field Programmable Gate Array (FPGA). Direct integration into the processor is also possible.

The ports connect the ESC to other EtherCAT slaves and the master. EtherCAT slaves support two to four ports. The logical ports are numbered 0-1-2-3 or A-B-C-D. A physical topology of an EtherCAT network is always structured as a logical ring, as can be seen in Figure 2.7.

The ESCs are always connected to the master via Port 0 and to the following slave via Ports 1 to 3. The EPU, which is located behind Port 0, takes over the frame processing. The returning frames are directly forwarded to the next Port (as shown at Port 1) or returned to Port 0 (as shown at Port 2). Each port has a auto-forwarder and a loop-back-function. The auto-forwarder receives the Ethernet frames, performs a check of the frames and forwards them to the loop-back function. The loop-back function forwards Ethernet frames to the next logical port if there is no link at a port, if the port is not reachable, or if the loop for the port is closed. The loop back function of Port 0 forwards the frames to the EPU.

EtherCAT Processing Unit (EPU)

The EPU receives, analyzes, and processes the EtherCAT data [38]. It enables and coordinates the access to the internal registers and to the memory area of the ESC. The memory area of the ESC can be addressed by the EtherCAT master and via the

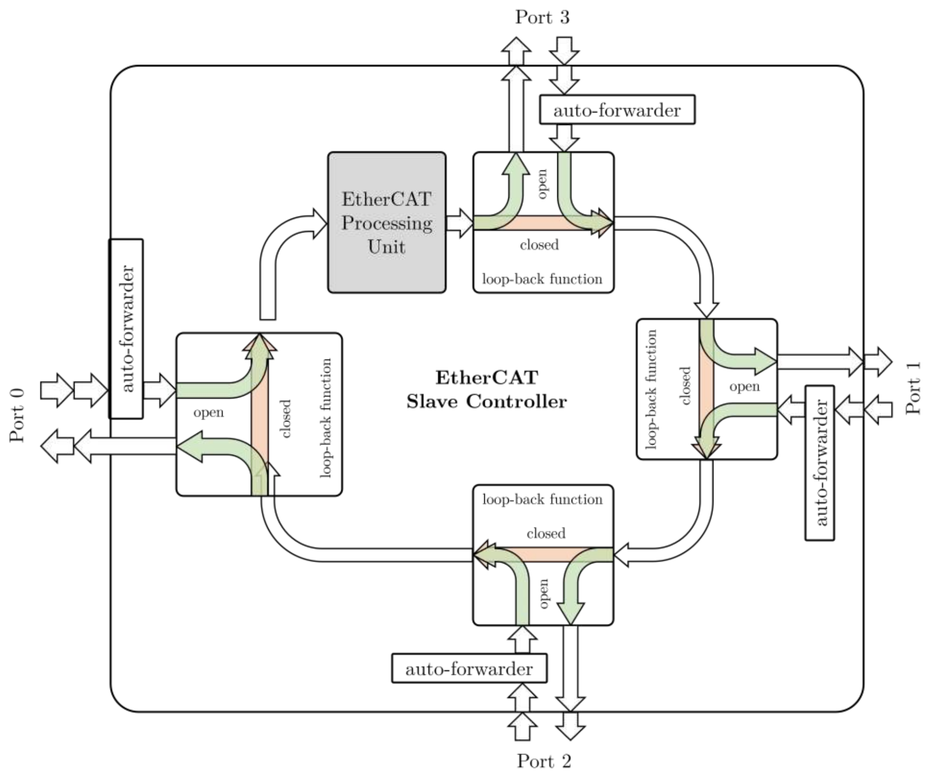


Figure 2.7: Processing order of a four-port-ESC [36]. Each port has an auto-forwarder and a loop-back function. The auto-forwarder receives the Ethernet frames, performs a check of the frames and forwards them to the loop-back function. The loop-back function forwards Ethernet frames to the next logical port. If there is no link at a port, the port is not reachable or the loop for the port is closed, the loop-back function forwards the Ethernet frames to the next logical port. The EPU is always located after Port 0. The master is always connected via Port 0.

PDI by the local application. Figure 2.8 shows the main components of the EPU.

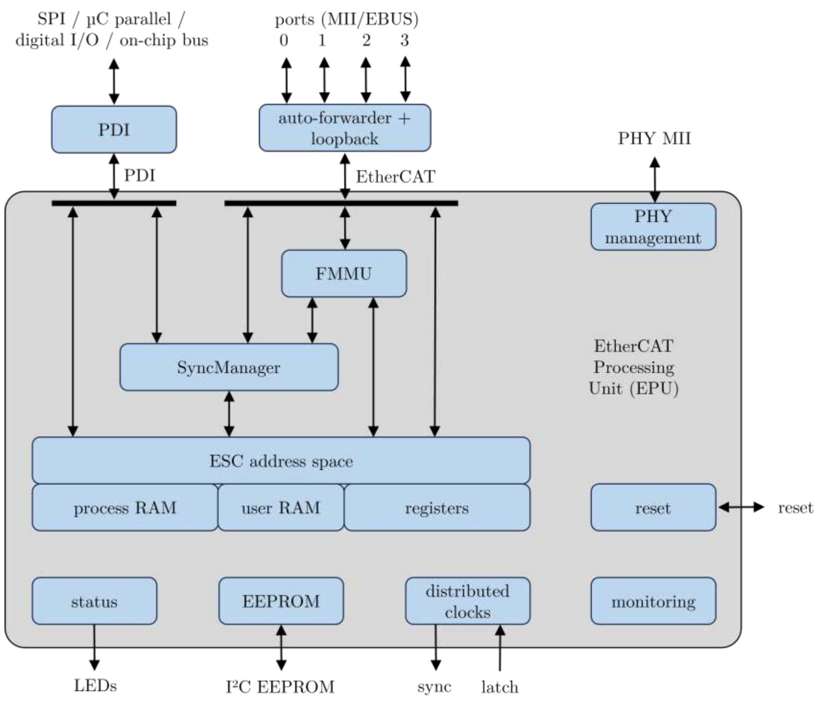


Figure 2.8: Block diagram of an EPU [38]. The ESC address space (Dual Port Random Access Memory) is accessible from both directions, the EtherCAT network and the PDI. SyncManager managers prevent simultaneous access to the memory. FMMUs map the logical addresses of the EtherCAT frame to the local address space in the EPU. Other important components of the EPU are PHY Management, Monitoring, Distributed Clocks, EEPROM, Status Block and a Reset controller.

The following descriptions provide a brief overview of the various components and functions of an EPU.

- **Physical Layer (PHY) Management**
  The task of the PHY management is to read and write the control and status registers of the PHY in order to configure each PHY before operation, and to monitor link status during operation.

- **SyncManager**
  SyncManagers are responsible for a consistent data exchange between the EtherCAT master and the EtherCAT slaves. To prevent simultaneous access to DPRAM by the EtherCAT network (master) and the PDI (local µC), a mechanism is needed to protect the data. This task is performed by the SyncManager.

If the slave uses FMMU, the SyncManagers for the corresponding data blocks are located between the DPRAM and the FMMU. To guarantee data consistency two modes can be used, mailbox mode and buffered mode.

In mailbox mode the EtherCAT master and the µC only get access to the buffer when the other has finished its access. If the sender writes to the buffer, the buffer is locked for writing until the receiver has read it out. Mailbox mode is typically used to exchange acyclic data such as parameter settings.

Buffered mode is typically used for cyclic data exchange. In buffered mode, the EtherCAT master and µC can access the communication buffer simultaneously. This makes this mode suitable for the exchange of process data. The sender can always update the content of the buffer whereby the receiver always receives the latest buffer content. Three buffers of identical size are physically used for the buffered mode and therefore require three times the process data size assigned in DPRAM. The start address and size of the first buffer are configured in the SyncManager configuration. The addresses of this first buffer are used by the master device and by the µC for reading and writing the data. Depending on the state of the SyncManager, accesses to the address range of the first buffer are redirected to one of the three buffers. Therefore other SyncManagers must be configured in such a way that they do not address the memory area of the second and third buffer.

The standard configuration includes four SyncManagers. One for acyclic data output (mailbox out, master to slave), one for acyclic data input (mailbox in, slave to master), one for cyclic data output (process data out, master to slave) and one for cyclic data input (process data in, slave to master). The standard configuration plus the calculation of the DPRAM size can be seen in Table 2.2.

Table 2.2: Standard SyncManger configuration and DPRAM size calculation [36].

SyncManager	Buffer Count	Length [Byte]	Total Length [Byte]
Mailbox Output	1	L_MbxOut	$1 \cdot$ L_MbxOut
Mailbox Input	1	L_MbxIn	$+$ $1 \cdot$ L_MbxIn
Outputs	3	L_Out (TxPDO)	$+$ $3 \cdot$ L_Out
Inputs	3	L_In (RxPDO)	$+$ $3 \cdot$ L_In
			$=$ Σ DPRAM size

- **Fieldbus Memory Management Units (FMMU)**
  FMMUs are used to map logical addresses bit by bit or byte by byte to physical addresses of the ESC. In the start-up phase, the master configures the FMMUs of each slave to set which area of the logical process data image is to be assigned

to which local address space. Each FMMU channel maps a continuous logical address space to a continuous physical address space of the slave device. While the telegram is passing through the device, the FMMU can extract specific data for the terminal and also insert data into the telegram.

- **Dual Port Random Access Memory (DPRAM)**
  An EtherCAT slave can have an address space of up to 64 kB. The first block of 4 kB, `0x0000-0x0fff`, is used for registers and for user memory. The memory address `0x1000-0xffff` is used as process data memory. The size of the process data memory depends on the device. The ESC address area is directly addressable by the EtherCAT master or an attached µC.

- **Process Data Interface (PDI)**
  An EtherCAT slave can have several types of PDIs. The different types of PDI are: Digital I/O, SPI slave, 8-16 bit µC, on-chip bus, and multipurpose I/O.

- **Status**
  The status block provides ESC information and application status information. It controls the external LEDs.

- **Electrically Erasable Programmable Read-Only Memory (EEPROM)**
  A non-volatile memory is used for the ESC configuration and the device description.

- **Distributed Clocks**
  Distributed clocks allow precisely synchronized generation of output signals, precisely synchronized reading of inputs and generation of time stamps for events. Synchronization can involve the entire EtherCAT network.

- **Monitoring**
  The monitoring unit contains blocks for counting errors and contains watchdogs. Watchdogs are functions that monitor correct process data communication. Error counters help to analyze errors.

- **Reset**
  The integrated reset controller monitors the power supply and controls external and internal resets.

2.2.5 EtherCAT State Machine (ESM)

The state of the EtherCAT slave is controlled via the ESM [36, 38] In each state, special functions can be executed in the EtherCAT slave. A distinction is made between five different states: *Init*, *Pre-Operational*, *Safe-operational*, *Operational*, and *Bootstrap*.

The functional principle is shown in Figure 2.9.

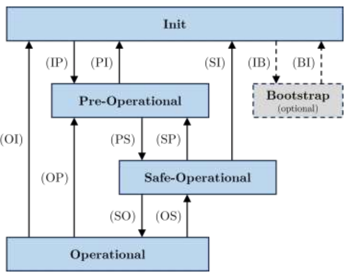


Figure 2.9: Illustration of the different states and the possible transitions of an ESM [36].

Init

The EtherCAT slave is in *Init* state after power-on. The EtherCAT master initializes the SyncManager for mailbox communication. Neither mailbox nor process data communication is possible.

Pre-Operational (*Pre-Op*)

At the transition from *Init* to *Pre-Op* the EtherCAT slave checks the correctness of the mailbox initialization. In *Pre-Op* state mailbox communication is possible. The EtherCAT master initializes the SyncManager for process data and the FMMU channels. If the EtherCAT slave supports a configurable mapping, the EtherCAT master also initializes the PDO mapping or the SyncManager PDO assignment. In addition, the settings for the process data transfer as well as possible terminal-specific parameters that deviate from the default settings are transferred.

Safe-Operational (*Safe-Op*)

During the transition from *Pre-Op* to *Safe-Op* the EtherCAT slave checks whether the SyncManager channels for the process data communication. Furthermore the EtherCAT slave copies current input data into the corresponding DPRAM areas of the ESC before the state change is completed. In *Safe-Op* state mailbox and process data communication is possible. The input data are already updated cyclically. However, the data outputs of the EtherCAT slave are not yet active.

Operational (*Op*)

During the transition from *Safe-Op* to *Op* the EtherCAT master activates the output of the EtherCAT slave. In the *Op* state full process data and mailbox communication is possible.

Boot

The *Boot* state can only be reached via the *Init* state. Here, an update of the slave firmware can be done. In the *Boot* state only mailbox communication via the File-Access over EtherCAT protocol is possible.

2.3 Serial Peripheral Interface (SPI)

As already mentioned in Section 2.2, different PDIs can be used for communication between the ESC and the application layer. One of the PDIs is the SPI. SPI facilitates full-duplex communication, allowing simultaneous transmission and reception of data [40]. With support for high-speed data transfers and straightforward connectivity, this interface is well-suited for the transmission of depth data. In the course of this sub-chapter, the SPI is discussed in more detail.

SPI is a synchronous, full duplex main-subnode-based interface for synchronous serial short distance communication between integrated circuits [41]. The SPI topology consists of one main/master device and one or multiple subnodes/slave devices. Both the main and the subnode can transmit data at the same time. The 4-wire SPI interface has four signals: Clock (SPI CLK, SCLK), Chip Select/Slave Select (CS/SS), Main Out and Subnode In (MOSI), Main In and Subnode Out (MISO). Figure 2.10 shows the functionality of the SPI.

The master generates the clock signal that provides the synchronization of the data transmission. The chip select signal from the main is used to select the subnode, and it is usually an active `Low` signal. This signal is pulled `High` to disconnect the subnode from the SPI bus. Each subnode has an individual chip select signal from the main. MOSI and MISO are the lines for data transmission. MOSI transmits data from the main to the subnode, and MISO transmits data from the subnode to the main. On the subnode side, the data lines are named serial data output (SDO) or serial data input (SDI).

The SPI distinguishes between four different modes, in which they differ in clock polarity (CPOL) and clock phase (CPHA). The period when the chip select is `High` and goes `Low` at the beginning of the transmission and when chip select is `Low` and goes `High` at the end of the transmission is called idle state. The CPOL bit sets the polarity of the clock signal during the idle state. The CPHA bit selects the clock phase. Either a rising or falling edge is used to sample and/or shift the data, depending on the CPHA

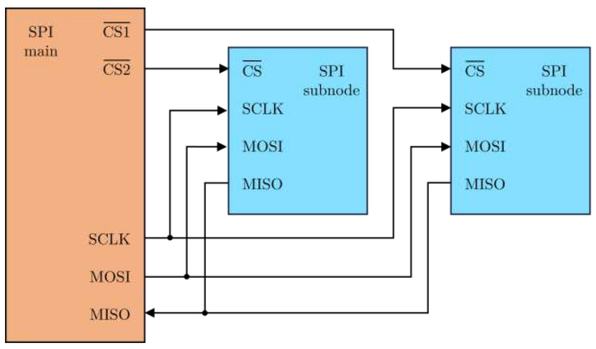


Figure 2.10: Blockdiagram of a SPI network with one SPI main and two SPI subnodes in regular mode [41]. Each subnode has it's one inverted Chip Select ($\overline{\text{CS}}$) connection. The main sets the Clock (SCLK) in the SPI network. Data between the main and the selected subnode can be simultaneously transmitted via two lines. MOSI transmits data from the main to the subnode, and MISO transmits data from the subnode to the main.

bit. The requirements of the subnode regarding clock polarity and clock phase must be fulfilled in the main.

As already mentioned multiple subnodes can be used with a single SPI main. The subnodes can be connected in regular mode or daisy-chain mode. In regular mode, every subnode has an individual chip select from the main. The clock and data are available for the selected subnode as soon as the associated chip select signal is enabled (pulled `Low`). If several chip select signals are enabled, the data on the data lines will be corrupted because the main cannot identify which subnode is transmitting the data.

In daisy-chain mode, the sub-nodes are implemented with a common chip select signal so that the data flows from one sub-node to the next. In this arrangement, all sub-nodes receive the SPI clock simultaneously. The main data line is connected directly to the first sub-node, which then transmits data to the subsequent sub-nodes in sequence. This means that the number of clock cycles required for transmission is directly linked to the position of the sub-node in the chain as the data pass through the daisy chain.

2.4 Research Question

Research shows that 3D vision is becoming increasingly important in industrial automation. The potential of application areas is constantly expanding and far from being exhausted. In particular, the use of high-resolution depth images in industrial applications to detect objects or people is steadily increasing. Typically, data from depth cameras are transmitted to controllers or PC systems via USB or Ethernet interfaces. Some of the goals of Industry 4.0 are to meet needs in a customer-oriented sense, emphasizing speed, (cost-) efficiency, simplicity, reliability, and modularity. Fieldbuses, in general, meet these needs. One of the most modern fieldbuses is EtherCAT. This fieldbus is characterized by its short cycle times and its high data transfer rate. It also guarantees real-time and is versatile. There are now countless fieldbus components such as actuators, analog and digital input and output terminals, 2D cameras, and much more, which communicate via this bus with a controller in the context of an industrial application. 3D vision can contribute to improving industrial applications, as more information is available. As fieldbus systems are state of the art in industry and depth cameras usually have a USB or Ethernet interface, integrating depth cameras into a fieldbus would be in line with Industry 4.0. This leads to the first research question of this master thesis:

Research Question 1

Is it feasible to embed depth information from a 3D camera via fieldbus within a PLC, while maintaining the specifications of the embedded camera system?

To prove feasibility, the proposed system is integrated and tested in an industrial application. Safety applications are particularly suitable for evaluation, since certain requirements for the system are defined in the standards. Therefore, an application in the field of collision avoidance is developed. The relevant valid standard is DIN ISO/TS 15066 Robots and robotic devices - Collaborating robots [22]. This technical specification defines the safety requirements for collaborative industrial robot systems and the working environment and supplements the requirements and instructions for the operation of collaborative industrial robots given in ISO 10218-1 and ISO 10218-2 [42, 43]. With regard to the camera system, the DIN ISO/TS 15066 standard primarily evaluates the recognition time, which results from the latency and the frame rate. Modern safety camera systems specify a latency of less than 100 ms at a frame rate of 30 fps. This will be used as motivation for the second research question:

Research Question 2

Is it feasible to use the developed camera system in a human robot collaboration application, while complying with the specifications of existing camera systems?

CHAPTER 3

System setup for the transmission of depth data via EtherCAT

This chapter describes the system setup for the transmission of depth data via Ethernet for Control Automation Technology (EtherCAT) and the selection of the different components of the EtherCAT-capable camera. First, an overview of the system structure is given. Subsequently, the used EtherCAT master is described. Finally, the components of the EtherCAT slave are presented.

The aim of the system is to integrate depth data from a 3D camera into the fast Ethernet fieldbus EtherCAT, making it accessible within a Programmable Logic Controller (PLC). The setup system consists of an EtherCAT master and an EtherCAT slave. The slave is the EtherCAT-capable depth camera and consists of three components: a 3D camera, a Microcontroller (µC) device, and an EtherCAT Slave Controller (ESC) device. The depth camera transmits the data via a bus to a µC. The µC processes and forwards the depth information via a process data interface to an ESC. The ESC is directly connected to the controller via the EtherCAT fieldbus. If a frame passes through the slave, the slave writes the data to the frame and sends them back to the master. Thus, the depth data has been transmitted to a PLC via a fieldbus. Figure 3.1 gives a basic overview of the proposed system.

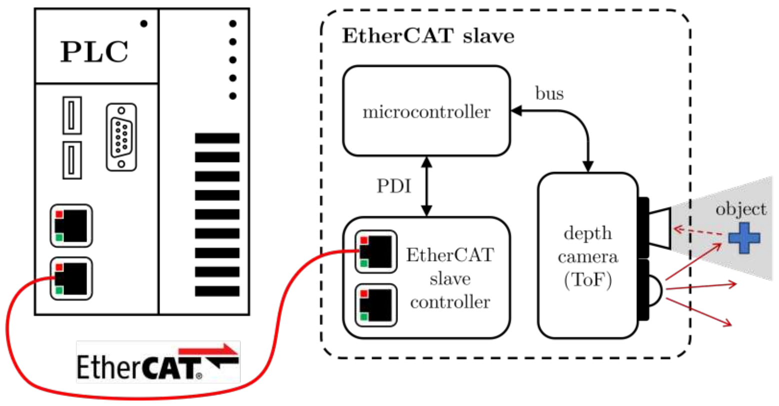


Figure 3.1: System overview of the EtherCAT master and EtherCAT slave. The PLC is directly connected over EtherCAT with the EtherCAT slave. The slave consists of three components: ESC device, µC and 3D camera. This 3D camera uses ToF technology to generate depth information. A light source emits infrared light. If the light hits an object, it is reflected and sent back to the camera. A sensor detects the reflected light. Amplitude modulation is used to determine the distance between the camera and the object. The 3D camera shares this information with a µC via bus. The µC processes the data and sends it via a Process Data Interface to the ESC, where it can be polled by EtherCAT frames.

3.1 Master device

The EtherCAT Master is a desktop PC with the Windows 10 operating system and the The Windows Control and Automation Technology 3 (TwinCAT3) software from Beckhoff Automation GmbH [44]. TwinCAT3 transforms a PC-based system into a real-time controller with one or more run-time systems. TwinCAT3 is structured into three main components: TwinCAT3 eXtended Automation Engineering (TwinCAT3 XAE), TwinCAT3 eXtended Automation Runtime (TwinCAT3 XAR), and TwinCAT3 Functions.

The TwinCAT3 XAE is an engineering tool that enables the programming and configuration of hardware. The standard version of TwinCAT3 includes all programming languages of the IEC 61131-3 standard [45], as well as functionalities such as program debugging and control hardware diagnostics.

The TwinCAT3 XAR is a real-time capable runtime, which is used to control in the field level of a system. In addition to an operating-system-independent runtime, the operating system which always runs on PC-based controllers enables the execution of other applications as well.

TwinCAT3 Functions enables the modularity of the system, which means that individual and demand-oriented projects can be created. The basic system can be extended with a wide range of automation functions. Within the application of human robot collaboration, the following *Vision* functions is relevant. It is used for image processing, configuration of cameras and programming of image analysis [46].

TwinCAT3 XAE is integrated in a Microsoft Visual Studio environment. TwinCAT3 XAE enables the inputs and outputs of the PLC to be linked with the inputs and outputs of the I/Os [47]. TwinCAT3 I/Os collect cyclic data from different fieldbuses in the process images. Tasks drive the corresponding fieldbuses, which can be operated with different cycle times, on a CPU. Tasks are executed cyclically and can be weighted with priorities. Tasks with higher priority can interrupt tasks with lower priority, which is why tasks with short cycle times should have high priority. The transport layer between the TwinCAT3 XAE and the TwinCAT3 XAR is the Automation Device Specification (ADS) communication protocol [38]. The TwinCAT3 architecture of an exemplary application can be seen in Figure 3.2.

For multi-core processor systems, TwinCAT3 offers the possibility to isolate individual cores. This allows different TwinCAT3 tasks to be assigned to a core that is isolated for real-time use. For communication between the master and the slaves, a network card is required that supports the TwinCAT3 driver for Ethernet cards. In the following the system-relevant Desktop PC components are specified:

- Processor: Intel(R) Core(TM) i5-6600 CPU 3.30GHz 4 Cores,
- Network card: Beckhoff FC9001-0010 Ethernet-Network card.

3.2 Slave device

The requirements for the EtherCAT slave are: high data transmission rate, low latency, compact, moderate implementation effort, low cost, and easy to reproduce. Limitations are only given by the EtherCAT fieldbus and the EtherCAT master. Since the standard EtherCAT data transmission allows a maximum speed of $100\,\mathrm{Mbit\,s^{-1}}$ [34], it is important to select an EtherCAT-capable camera system so that this limit is not exceeded [34]. Due to the modularity and versatility of EtherCAT slaves, the system should be far below the $100\,\mathrm{Mbit\,s^{-1}}$ mark. The limiting of the data transfer rate can be done by the choice of the 3D camera or by a software solution. In this regard, a µC should be chosen that offers a suitable interface for configuration and programming, an easy connection to the 3D camera and a Process Data Interface (PDI) interface for the EtherCAT controller. In addition the µC should be powerful enough to handle the amount of depth data provided by the depth camera. The data transfer rate should also

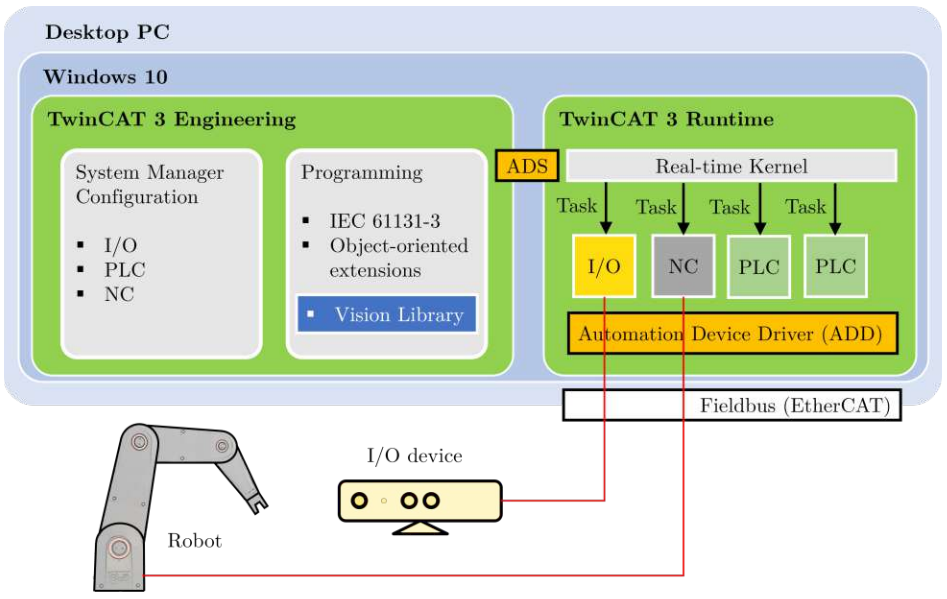


Figure 3.2: Overview of the TwinCAT3 implementation on a desktop PC with an exemplary application [47]. The TwinCAT3 eXtended Automation (XA) architecture consists of two main components, the TwinCAT3 engineering tool and the TwinCAT3 runtime. The eXtended Automation Engineering (XAE) is the development environment of TwinCAT3. It offers different programming languages and libraries like the vision library. A compiler for translation is also included in TwinCAT3. The eXtended Automation Runtime (XAR) is the real-time environment in which the TwinCAT3 modules created are loaded, executed, and managed. In this figure, two PLC modules, one I/O module, and one NC module are executed. These are cyclically performed by the so-called tasks. Such TwinCAT3 tasks can be distributed to different cores of a CPU to increase performance. In this application, two slaves are connected to the master via a fieldbus. The communication with the respective modules is cyclic.

not be limited by the process data interface and by the hardware of the ESC. Based on these requirements, the components of the EtherCAT capable camera system can be selected.

3.2.1 Microcontroller device

The µC is the core of the EtherCAT slave. It serves on the one hand as an interface from the 3D camera to the ESC, on the other hand for processing the depth data and as a user interface for settings and programming. This means that the µC must contain an interface for a depth camera and an ESC. The µC used for this master thesis is the Raspberry Pi 4 Model B [48]. The Raspberry Pi is a single board computer, which can be operated with the operating system Raspberry Pi OS. Raspberry Pi OS is a free operating system based on Debian (Linux), optimized for Raspberry Pi hardware. It supports most programming languages like C, C++, Java, HTML, Python etc. It also offers an easy-to-use user interface for programming. The Raspberry Pi has two Micro HDMI ports for this purpose. In addition, it is also possible to access the system with the network protocol Secure Shell (SSH). Due to the Linux-based operating system, a high availability of camera drivers is given, which facilitates the commissioning of these. The single-board computer offers two USB 2, two USB 3 and an Ethernet interface for the connection of a depth camera. It also has an Serial Peripheral Interface (SPI), which allows process data to be exchanged with ESCs that support this process data interface. The physical connection is provided via the General Purpose Input/Output (GPIO) pins of the Raspberry Pi. Figure 3.3 serves to illustrate the Raspberry Pi 4 Model B.

Figure 3.3: Raspberry Pi 4 Model B.

The system-relevant properties of the Raspberry Pi 4 Model B used are listed below:

- Processor Broadcom BCM2711 Chip

64 bit Quad Core ARM v8 Cortex-A72 1,5 GHz,

- Random Access Memory 4 GB LPDDR4,
- Gigabit Ethernet,
- 2x USB 3.0 port and 2x USB 2.0 port,
- 40 Pin GPIO Header,
- 2x micro HDMI,
- Power supply 5V/3,0A DC via USB Type C jack,
- Micro SD format for loading the operating system and for data storage.

The advantages of the Raspberry Pi 4 Model B as an EtherCAT slave are its versatile connectivity, the user-friendly interface and the powerful processor. The decisive factor, however, is that there is an ESC board developed for the Raspberry Pi, the EasyCAT HAT from AB&T Srl.

3.2.2 EtherCAT slave controller (ESC)

For the single-board computer Raspberry Pi 4 Model B an ESC board exists. The EasyCAT HAT from AB&T Srl allows a Raspberry microcontroller to become an EtherCAT Slave [49]. The communication between the single board computer and the ESC takes place via the SPI of the Raspberry Pi's GPIO pins. The power supply of the EasyCAT HAT is also done via the GPIO pins. The ESC LAN9252 from Microchip [50] is integrated on the AB&T Srl board. The LAN9252 is a two or three-port ESC with integrated Physical Interface (PHY). Each PHY contains a full duplex 100BASE-TX transceiver and supports 100 Mbit s^{-1} operation. The ESC of the LAN9252 includes 4 kB of Dual Port Random Access Memory (DPRAM) and three Fieldbus Memory Management Unit (FMMU). The board also has 4 kB of Electrically Erasable Programmable Read-Only Memory (EEPROM) to define the EtherCAT slave. The ESC includes four SyncManagers. These allow the exchange of data between the EtherCAT master and the local application. Each SyncManager's direction and mode of operation, buffered mode, or mailbox mode, are configured by the EtherCAT master. An SPI slave controller provides a synchronous slave interface with a low pin count. It allows access to the System Control and Status Registers (CSR), internal First In First Out (FIFO) buffers, and memory. Bit lanes are supported with a clock rate of up to 80 MHz. AB&T Srl also provides a tool, called *EasyConfigurator*, to write the EEPROM and thus define the slave. However, only 256 B can be defined with this tool.

Figure 3.4 shows the EasyCAT HAT.

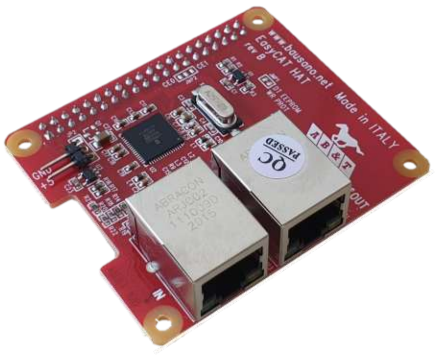


Figure 3.4: ESC device: AB&T Srl EasyCAT HAT.

3.2.3 3D camera

The requirements for the 3D camera in this system are: a data transfer rate much lower than 100 Mbit s^{-1} due to the EtherCAT limitations, the use of ToF technology because of the advantages for the human robot collaboration application as discussed in Section 2.1, the availability of a driver for Linux based operating systems to operate on the Raspberry Pi, and connection via USB or Ethernet. The flexx2 from pmd technologies is a 3D imaging ToF USB camera [20]. It offers software drivers for Windows and Linux operating systems. With a resolution of 224×172 pixels ($W_{image} \times H_{image}$), a maximum frame rate FRR of 60 fps and a measuring range of 0.1 m to 4 m at a depth resolution of $<= 1\,\%$ of measured distance it achieves a maximum data transfer rate DTR, according to Equation (2.3), of

$$\begin{aligned} DTR &= W_{image} \cdot H_{image} \cdot FRR \cdot R_{depth} \\ &= 224 \cdot 172 \cdot 60\,\text{fps} \cdot 16\,\text{bit} = 36.99\,\text{Mbit}\,\text{s}^{-1}, \end{aligned} \tag{3.1}$$

using a depth information R_{depth} of 16 bit per pixel. The camera includes a IRS2381 Infineon REAL3 3D Image Sensor based on pmd technology with a viewing angle (Horizontal $\times$ Vertical) of $56° \times 44°$. Figure 3.5 shows the flexx2.

Since ToF technology is not limited to certain distances, it increases flexibility in various applications. The flexx2 driver offers several pre-defined modes that the user can select. Depending on the use case, the distance and frame rate can be changed to obtain a suitable image capture. The mode can be changed on-the-fly and requires one frame for the change. The "5" modes (five sub-frames) are better suited for short distances or when the processor load is a problem. The mode "5" cases require about

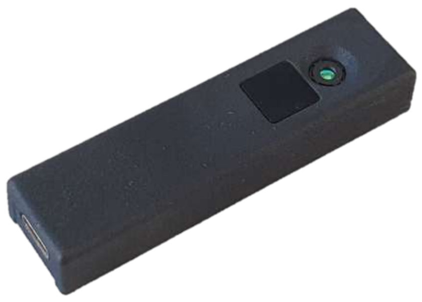


Figure 3.5: 3D ToF USB camera: flexx2.

30 % less processing than the mode nine cases. The "9" modes (nine sub-frames) have greater range and depth quality because they use two modulation frequencies, but they need more processing power. On the Raspberry Pi 4 Model B, all modes work without dropping a frame rate. The different modes are listed in Table 3.1.

Table 3.1: Different settings of the flexx2 [20].

Frequency	Mode Name	Exposure Time	Phases	FPS	Approximate Max Range
20 MHz	MODE_5_15FPS	1040 µs	5	15 fps	3 m
20 MHz	MODE_5_30FPS	500 µs	5	30 fps	2.5 m
20 MHz	MODE_5_45FPS	310 µs	5	45 fps	2 m
20 MHz	MODE_5_60FPS	220 µs	5	60 fps	1 m
20 MHz–60 MHz	MODE_9_5FPS	1500 µs	9	5 fps	4 m
20 MHz–60 MHz	MODE_9_10FPS	760 µs	9	10 fps	3.5 m
20 MHz–60 MHz	MODE_9_15FPS	500 µs	9	15 fps	3 m
20 MHz–60 MHz	MODE_9_20FPS	390 µs	9	20 fps	2.5 m
20 MHz–60 MHz	MODE_9_30FPS	220 µs	9	30 fps	2 m

CHAPTER 4

Implementation of the EtherCAT-capable depth camera in an industrial application

This chapter describes the implementation of the depth camera, flexx, in the Ethernet for Control Automation Technology (EtherCAT) fieldbus as well as its integration into the human-robot collaboration application. First, the concept for transferring the depth data via EtherCAT is presented in Section 4.1. Then the implementation of the EtherCAT Slave Controller (ESC) EASYCAT HAT on the Raspberry Pi is discussed in Section 4.2. Subsequently, the connection of the flexx2 to the Raspberry Pi is described in Section 4.3, which leads to the complete integration of the depth camera in EtherCAT, presented in Section 4.4. Afterwards, the depth data processing in The Windows Control and Automation Technology 3 (TwinCAT3) is outlined in Section 4.5. This results to the integration of an EtherCAT-capable camera into an application in the field of collision avoidance, presented inSection 4.6.

4.1 Concept for the transmission of depth data via EtherCAT

Images or depth images have a high memory requirement. This depends on the resolution and the bit depth used. A depth image of the flexx2 has 38.528 pixels [20]. With a measurement range of 0.1 m − 4 m at a depth accuracy of $\leq 1\,\%$ of the measured distance. 2 B per pixel are sufficient to define all the depth information provided by the flexx2. This results in a total byte count of 77 056 B per frame.

As mentioned in Section 3.2.2, the ESC LAN9252 [50] has a Dual Port Random Access Memory (DPRAM) of 4 kB. The first block of address space, `0x0000-0x0fff`, is used for registers and user memory (mailbox communication). The address space, `0x1000-0x1fff` is available for DPRAM. Due to the buffered mode this results in a maximum definable memory area for cyclic data exchange of

$$DPRAM_{available} = \frac{DPRAM_{total}}{3} = \frac{4096\,\mathrm{B}}{3} \approx 1365\,\mathrm{B}. \tag{4.1}$$

To be able to transmit a depth image of the flexx2 without loss, a method must be developed that allows 77 056 B per frame to be sent via the relatively limited DPRAM of 1365 B. The considered approach can be seen in Figure 4.1.

The depth images sent by the flexx2 are transmitted as line-sorted 1D arrays to the Raspberry Pi via USB. Thereby, the 1D array is divided into segments of equal size. The depth data are then written to the EASYCAT HAT's DPRAM via SPI. In the ESC, a two byte input variable is defined for each pixel in a segment. The definition of the input variable is given by the Electrically Erasable Programmable Read-Only Memory (EEPROM) configuration. If an EtherCAT frame passes the ESC, the data of a segment are written to the EtherCAT frame and forwarded to the controller. During this reading process the data of the next segment are written into another buffer of the DPRAM. This is then read with the next EtherCAT frame. The process is repeated until the entire depth image has been transferred.

Within the controller, the input variables of the camera device in the I/O area are linked to a defined segment array in the Programmable Logic Controller (PLC). To reconstruct the one-dimensional depth array, the position of the sent segment must be known. Therefore, the address of the first element in the segment is also sent with the depth data. Once the entire depth image has been transmitted, the entire process can be repeated.

The attempted approach requires multiple EtherCAT frames per depth image. These are sent cyclically by TwinCAT3. The minimum cycle time of TwinCAT3 is 50 µs. To keep the latency of the camera system as low as possible, the number of required frames is minimized, whereby a maximum of 1365 B per frame can be transmitted. In addition, the segmentation process should be as simple as possible. If the total number of bytes per depth image is divisible by the number of pixels per transmitted segment, this simplifies the implementation.

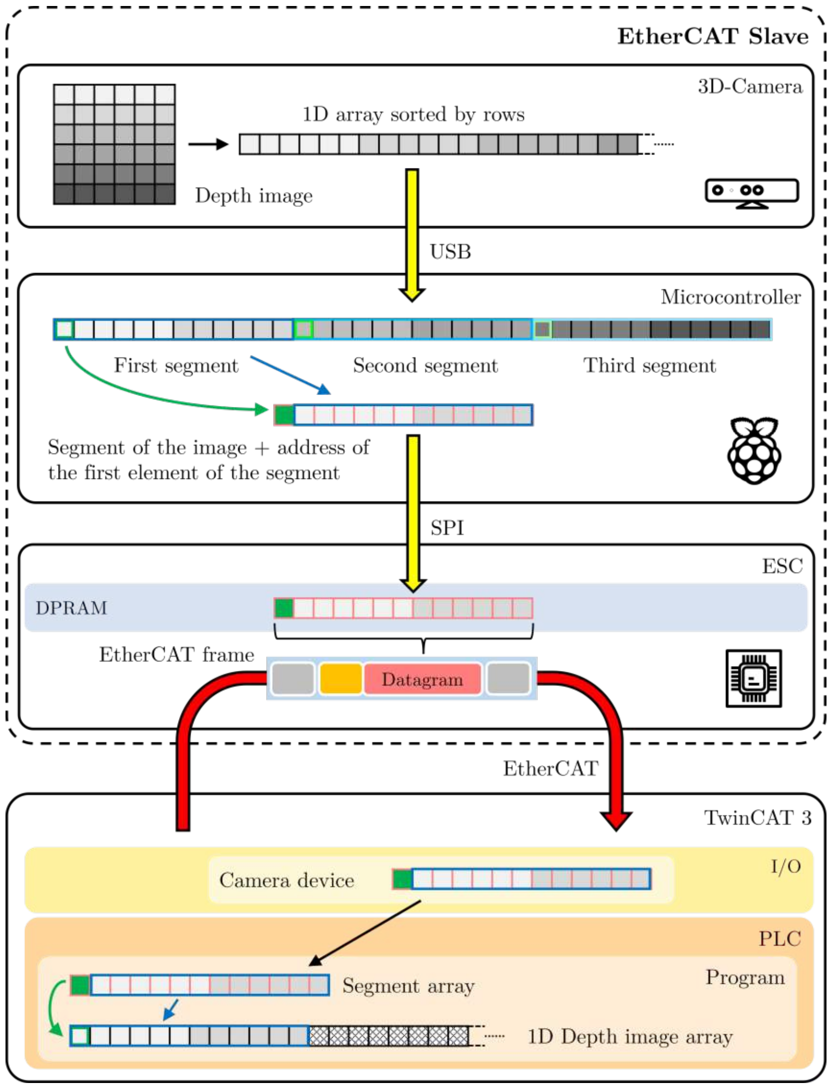


Figure 4.1: Representation of the implemented methods to transmit depth data via EtherCAT. A depth camera sends the depth image as a row sorted 1D array via USB to the µC. The image here has an exemplary resolution of 36 pixels. In the µC, only one segment and the address of the first element in the segment is sent to the ESC via SPI. Here, the array is divided into three segments. When an EtherCAT frame passes the slave, the data and the address are read from the DPRAM and sent to the controller. The data received in the I/O area are linked to an array of the size of the DPRAM of the ESC. Using the segment address, the depth image can be reconstructed in the form of a 1D array. This process is repeated in this case 3 times per depth image. After that, a new frame of the depth camera can be transferred.

A data transmission of 301 pixel per segment results in an utilization of slightly more than 44 % of the DPRAM as well as an easy implementation of the segmentation. The flexx2 has 172 lines per depth image. This results in exactly 128 required EtherCAT frames for one depth image with a theoretical total latency of 6.4 ms. With the depth information of two byte per pixel plus two byte address information, a memory area of 604 B is defined in the ESC. Better utilization of the DPRAM would be possible, however, as can be seen in Section 5.3, the segmentation of 301 pixels per EtherCAT frame allows comparability of the different implemented data reduction methods.

4.2 Implementation of the EASYCAT HAT on the Raspberry Pi

The EASYCAT is directly connected via the GPIO ports of the Raspberry Pi 4. Since the board of the ESC is directly above the heat sinks of the Raspberry Pi after mounting, this results in a low heat dissipation of the system. To avoid performance losses due to overheating, the components are actively cooled. In addition, a heat sink is mounted directly on the ESC. The setup used is shown in Figure 4.2.

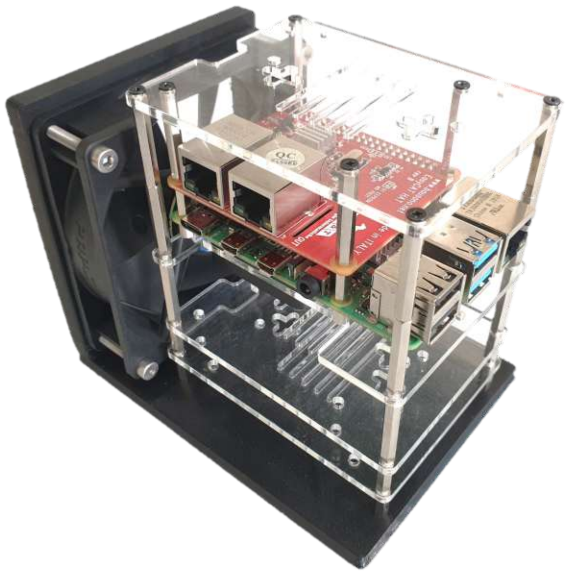


Figure 4.2: Implementation of the Raspberry Pi 4 and the EASYCAT HAT with active cooling to avoid performance losses due to overheating.

The aim of the implementation is not only to transmit the depth data but also to maintain the desired frame rate of the depth camera in the PLC and to minimize the latency of the entire transmission. This requires the fastest possible processing of the implemented methods. The Raspberry Pi 4 Model B is overclocked for this purpose. The following configuration parameters are set in the Raspberry Pi:

- `over_voltage=6`: Sets an overvoltage to the base voltage of the CPU,
- `arm_freq=2000`: Maximum clock speed of the CPU,
- `gpu_freq=750`: Maximum clock speed of the GPU,
- `force_turbo=1`: Forces the maximum value of the CPU clock permanently.

4.2.1 EEPROM programming

As already mentioned in Section 2.2.1, EtherCAT slaves are defined by so called EtherCAT Slave Information (ESI) files in Extensible Markup Language (XML) format. The EASYCAT HAT from AB&T Srl is delivered with a configuration software called *EasyConfigurator*. However, this only allows for a configuration of the EEPROM with a maximum of 256 B. 604 B of memory is required to implement the desired method. Therefore, with the *EasyConfigurator* software, only a template is created in XML format, which is then processed with a Python script designed for this slave. In the ESI, the structure of the object dictionary and the corresponding behavior of the entries are based on the Modular Device Profile (MDP) [51] of EtherCAT. When using MDP objects, the inputs (index `0x6000 - 0x6FFF`) are mapped to Process Data Objects (PDO) mapping objects (index `0x1A00 - 0x1BFF` for TxPDO mapping) and assigned to the respective SM via index `0x1C10` (SM0) for PDO assignment.

For each pixel in a segment and for the address of the first pixel in the depth data array two bytes are needed. Thus, the variables in the slave are defined as Unsigned Integer (UINT) with 16 bits. The utilized method sends 301 pixels and one address per EtherCAT frame. This results in a total of 302 defined input variables of type UINT. Two TxPDO are created for this purpose. The first PDO with the index `0x1A00` contains the address of the first element of the transmitted segment. The second PDO with the index `0x1A01` defines the sent 301 pixel of the depth image. The indexing of the depth data is based on the input variable designation. The PDO of the address is implemented in XML format as follows:

Listing 4.1: PDO of the address in XML format

```
<TxPdo Fixed="1" Mandatory="1" Sm="0">
  <Index>#x1A00</Index>
  <Name>Address</Name>
  <Entry>
    <Index>#x6000</Index>
    <SubIndex>0</SubIndex>
    <BitLen>16</BitLen>
    <Name>address</Name>
    <DataType>UINT</DataType>
  </Entry>
</TxPdo>
```

In the following the implementation of the first two data of a line is shown:

Listing 4.2: PDO of the data in XML format

```
<TxPdo Fixed="1" Mandatory="1" Sm="0">
  <Index>#x1A01</Index>
  <Name>InputData</Name>
  <Entry>
    <Index>#x6001</Index>
    <SubIndex>0</SubIndex>
    <BitLen>16</BitLen>
    <Name>data1</Name>
    <DataType>UINT</DataType>
  </Entry>
  <Entry>
    <Index>#x6002</Index>
    <SubIndex>0</SubIndex>
    <BitLen>16</BitLen>
    <Name>data2</Name>
    <DataType>UINT</DataType>
  </Entry>
  ...
</TxPdo>
```

To make the slave configuration available to the master, the slave ESI file must be available to TwinCAT3. The EEPROM of the EASYCAT HAT can be written by different tools. TwinCAT3 allows the configuration and writing of EEPROMs of EtherCAT capable slave devices. This requires a functioning EtherCAT connection between master and slave, and the existing EtherCAT Network Information (ENI) file. For the configuration of the EtherCAT slave in TwinCAT3, the device in the I/O of the Solution Explorer in a TwinCAT3 project must be scanned. The detected slave can then be configured by updating the EEPROM with the defined ESI file. To adopt the configuration of the EEPROM, the slave device must be removed and added again by scanning the entire device.

4.2.2 Configuration of the SPI

AB&T Srl [49] offers an implementation of the SPI on a Raspberry Pi based on the programming language C++. For the communication between the Raspberry Pi 4 Model B and the ESC, the LAN9252, the Broadcom BCM2835 library [52] is implemented. This is a C library for Raspberry Pi. It provides access to GPIO and other IO functions on the Broadcom BCM2835 chip, as well as on the BCM2711 chip. This allows control and communication with external devices such as the ESC LAN9252. The program structure of the SPI implementation consists of two header files (*EasyCAT.h* and *camera.h*) and two C++ files (*main.cpp* and *EasyCAT.cpp*).

camera.h

Using the *EasyConfigurator*, a header file is created which describes the slave. The data types and designations of the input and output variables of the EtherCAT slave are described, as well as the total number of bytes used in the memory of the slave. However, since the *EasyConfigurator* can define a maximum data range of 256 B, the input variables in the camera.h header file are configured using a Python script. Listing 4.3 shows the structure of the input data.

Listing 4.3: Input Variables of the EtherCAT slave.

```
#define TOT_BYTE_NUM_ROUND_IN 604
typedef union
{
   uint8_t  Byte [TOT_BYTE_NUM_ROUND_IN];
   struct
   {
      uint16_t    address;
      uint16_t    data1;
      uint16_t    data2;
      ...
      uint16_t    data301;
   }DataIn;
} PROCBUFFER_IN;
```

EasyCAT.h

The EasyCAT.h header file contains various addresses for accessing different registers of the ESC LAN9252. Furthermore, all commands of the flag register of the ESC LAN9252 as well as the EtherCAT State Machine (ESM) are defined. Additionally the write and read commands of the ESC and the SPI are specified. The EasyCAT class represents the instance of the ESC on the Raspberry Pi. It contains various constructors, variables and functions, which allow the data exchange. Listing 4.4 shows the structure of the EasyCAT class. It is described in more detail in *EasyCAT.cpp*.

Listing 4.4: EasyCAT class

```
class EasyCAT
{
  public:
    EasyCAT();
    EasyCAT(uint8_t   SCS);
    EasyCAT(SyncMode  Sync);
    EasyCAT(uint8_t   SCS, SyncMode Sync);

    bool           Init      ();
    unsigned char  SlaveTask ();
    void           CameraTask();

    PROCBUFFER_OUT  BufferOut;
    PROCBUFFER_IN   BufferIn;

  private:
    void           SPIWriteRegisterDirect   (unsigned short Address,
                                             unsigned long DataOut);
    unsigned long  SPIReadRegisterDirect    (unsigned short Address);
    void           SPIWriteRegisterIndirect (unsigned long  DataOut,
                                             unsigned short Address,
                                             unsigned char Len);
    unsigned long  SPIReadRegisterIndirect  (unsigned short Address,
                                             unsigned char Len);

    void           SPIReadProcRamFifo();
    void           SPIWriteProcRamFifo();

    uint8_t    SCS_;
    SyncMode   Sync_;
};
```

EasyCAT.cpp

All functions of the EasyCAT class are defined in the *EasyCAT.cpp* file. These are described in more detail below.

`EasyCAT()`
This is the standard constructor of the EasyCAT class. It defines the private variables `SCS` and `Sync`. `SCS` is the SPI Chip Select. A selection can be made between the two available SPI pins of the Raspberry Pi GPIO. The variable

`Sync` selects the synchronization mode of the EtherCAT slave. It is defined as an enumeration in the *EasyCAT.h* header file. The default mode is `ASYNC`. In this case, the slave operates autonomously based on its own cycle and is not synchronized with the EtherCAT cycle.

`bool Init()`
Executing the `Init()` function is necessary to be able to transfer data via SPI. It starts the initialization of the bcm2835 driver, sets up the GPIO pins for SPI access, selects the chip select defined in the constructor and also the SPI data mode. In addition, the clock divider can be selected. This is an integer divisor that is used to set the SPI clock. The core clock is the reference here. In addition, the `Init()` function uses the private functions `SPIWriteRegisterDirect`, `SPIReadRegisterDirect`, `SPIWriteRegisterIndirect` to select the synchronization mode, to perform an ESC reset, or to query the *Ready* flag. If the initialisation is successful, the return value is `true`.

`unsigned char SlaveTask()`
The `SlaveTask()` function is the standard task of the ESC. First, the watchdog status and the ESM status are requested. Then the output variables are read from the DPRAM and the input variables are written to the DPRAM. For this, the private functions `SPIReadProcRamFifo()` and `SPIWriteProcRamFifo()` are used. The return value is the current status of the ESM.

`void CameraTask()`
The `CameraTask()` function is based on the `SlaveTask()` function, but only input variables can be written to the DPRAM using the private function `SPIWriteProcRamFifo()`. In addition, the watchdog and ESM status are ignored. This allows considerably faster transmission of input data via the SPI.

`PROCBUFFER_OUT BufferOut`
The output variables are defined in the header file *camera.h*.

`PROCBUFFER_IN BufferIn`
The input variables defined in the header file *camera.h*.

main.cpp

The *main.cpp* file is used to run the application. A variable of type `EasyCAT` must be defined for the implementation of the EASYCAT HAT. This allows the transfer of data through SPI. Successful initialization is crucial for data transmission. The variables created in the header file *camera.h* are written via the EasyCAT class. The utilized task writes the data to the DPRAM of the ESC. It is necessary for a continuous data exchange to have the data updated cyclically, for example with a loop.

4.3 Software implementation of the flexx2 on the Raspberry Pi

The flexx2 is supplied with the *Royale* Application Programming Interface (API) [53]. This software package is a camera framework for Time of Flight (ToF) cameras and is written entirely in the C++ programming language. Therefore, the camera can be controlled via a high-level interface. In Listing 4.5, the functions required for the depth data obtained from the camera are described using a simple implementation.

Listing 4.5: Simple implementation of the *Royale* API for capturing depth data of a 3D camera.

```
#include <stdio.h>
#include <royale.hpp>
using namespace std;

class MyListener : public royale::IDepthDataListener {
   public:
      MyListener(){}
      void onNewData(const royale::DepthData *data) override {
         ...
      }
};

int main()
{
   unique_ptr<MyListener> listener;
   ...
```

The libraries *stdio.h* and *royale.hpp* are required for the implementation. The `IDepth-DataListener` class provides the listener interface to retrieve depth data from the *Royale* API. The `onNewData(...)` function is called with every update of the *Royale* framework frame. The structure `DepthData` defines the depth data that is delivered via the callback. This data contains a 3D point cloud with the size of the depth image (width, height). The coordinate pointer returns a line-sorted array with the size of (width x height x 4). For each pixel, the x, y and z coordinates are available in meters, as well as the confidence (in the range from 0 (bad) to 1 (good)). Based on this confidence, the user can decide whether to use the 3D point or not. The point cloud uses a right-handed coordinate system (x -> right, y -> down, z -> in image plane).

```
   ...
   royale::CameraManager manager;
   unique_ptr<royale::ICameraDevice> cameraDevice;

   auto camlist = manager.getConnectedCameraList();
   if (!camlist.empty()){
      cameraDevice = manager.createCamera(camlist[0]);}
   if (cameraDevice == nullptr){
      return 1;}

   if (cameraDevice->initialize() != royale::CameraStatus::SUCCESS){
      return 1;}
   ...
```

The class `ICameraDevice()` is the main interface for communication with the ToF camera system. The `CameraManager` is responsible for recognizing and creating instances of ICameraDevices, one for each connected (supported) camera device. The function `getConnectedCameraList()` attempts to establish a connection to each plugged-in camera, asks for its unique serial number, and returns the list of connected camera modules. The cameras found are transferred to an object of type `ICameraDevice` by calling `createCamera()`. This implementation assumes that only one camera is connected. The function `initialize()` initializes the camera device and sets the first available use case. It is necessary to call the initialize method before working with the camera device.

```
   ...
   listener.reset(new MyListener());
   if (cameraDevice->registerDataListener(listener.get()) != royale::
   CameraStatus::SUCCESS){
      return 1;}
   ...
```

The function `registerDataListener()` registers the listener on the camera device.

```
   ...
   if (cameraDevice->startCapture() != royale::CameraStatus::SUCCESS){
      return 1;}

   cin.get();

   if (cameraDevice->stopCapture() != royale::CameraStatus::SUCCESS){
      return 1;}
   return 0;
}
```

The function `startCapture()` starts the video capture mode, based on the specified operation mode. The function `stopCapture()` stops the video capture mode.

As described in Section 3.2.3, the flexx2 offers a selection of different operating modes. These can also be changed during image capture.

Listing 4.6: Implementation of the operation mode selection.

```
auto selectedUseCaseIdx = 7;
royale::Vector<royale::String> useCases;
auto status = cameraDevice->getUseCases(useCases);

if (status != royale::CameraStatus::SUCCESS || useCases.empty()){
   return 1;}

if (cameraDevice->setUseCase(useCases.at(selectedUseCaseIdx)) !=
royale::CameraStatus::SUCCESS){
   return 1;}
```

The function `getUseCases(useCases)` returns all use cases that are supported by the connected device. After checking the existing use cases, the camera mode can be set with the function `setUseCase(useCases.at(selectedUseCaseIdx))`. The variable `selectedUseCaseIdx` specifies the index of all use cases. For the flexx2, index 7 corresponds to the MODE_9_30FPS mode. If the mode is not selected, the default mode is MODE_9_5FPS.

4.4 Integration of flexx2 in EtherCAT

The considered approach, depicted in Figure 4.1, is now realized based on the implementations described in Section 4.2 and Section 4.3. For this purpose, the *main.cpp* file and the flexx2 implementation code are merged. Thereby, function `void sendFrame(float*, int )` is implemented for segmented transmission. The structure of the function is shown in Listing 4.7.

Listing 4.7: Function for sending depth data segment by segment via EtherCAT.

```
void sendFrame(float* depthData, int numOfPixels){

   size_t numOfInputs = TOT_BYTE_NUM_IN/2-1;

   for (size_t address = 0; address < (numOfPixels/numOfInputs);
   address++) {
      EASYCAT.BufferIn.DataIn.address = static_cast<uint16_t>(address)
   ;
      EASYCAT.BufferIn.DataIn.data1 = static_cast<uint16_t>
         (*(depthData+2+(address*numOfInputs+0)*4)*1000.0);
      EASYCAT.BufferIn.DataIn.data2 = static_cast<uint16_t>
         (*(depthData+2+(address*numOfInputs+1)*4)*1000.0);
         ...
      EASYCAT.BufferIn.DataIn.data301 = static_cast<uint16_t>
         (*(depthData+2+(address*numOfInputs+300)*4)*1000.0);

      EASYCAT.CameraTask();
   }
}
```

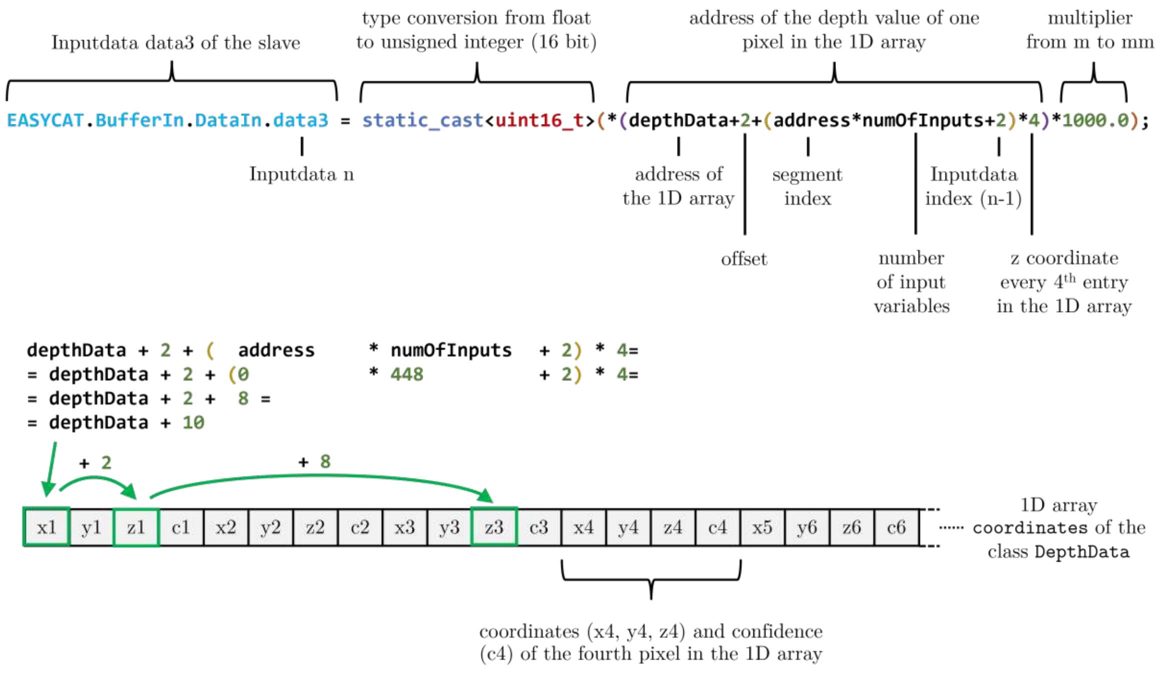
The arguments passed are the pointer to the depth data, the ESC object of type Easy-CAT, and the number of pixels to be transferred per depth image. The pointer points to the 1D array `coordinates` of the structure `DepthData`. The function `sendFrame(...)` iterates over all segments, assigns the corresponding depth values to the ESC variables and calls the function `CameraTask()` after the assignment. Therefore the values are written to the DPRAM of the ESC. The address variable sent with each segment contains its index. The assignment of individual depth values to the ESC variables is described in Figure 4.3.

The function `sendFrame(...)` is called in the function `onNewData(..)` of the data listener. As this function is automatically called by the *Royale* API after every frame update, no loop is required for the cyclical writing of data. The listener in Listing 4.5 is extended as shown in Listing 4.8.

Listing 4.8: Extension of the data listener for the attempted approach.

```
class MyListener : public royale::IDepthDataListener {
   public:
      MyListener(){}

      void onNewData(const royale::DepthData *data) override {
         int numOfPixels = data->width * data->height;
         float* depthData = (float*)data->coordinates;
         sendFrame(depthData, EASYCAT_, numOfPixels);
      }
};
```



Inputdata data3 of the slave
type conversion from float to unsigned integer (16 bit)
address of the depth value of one pixel in the 1D array
multiplier from m to mm
EASYCAT.BufferIn.DataIn.data3 = static_cast<uint16_t>(*(depthData+2+(address*numOfInputs+2)*4)*1000.0);
Inputdata n
address of the 1D array
segment index
Inputdata index (n-1)
offset
number of input variables
z coordinate every 4th entry in the 1D array
depthData + 2 + (address * numOfInputs + 2) * 4=
= depthData + 2 + (0 * 448 + 2) * 4=
= depthData + 2 + 8 =
= depthData + 10
+ 2
+ 8
x1 y1 z1 c1 x2 y2 z2 c2 x3 y3 z3 c3 x4 y4 z4 c4 x5 y6 z6 c6
1D array coordinates of the class DepthData
coordinates (x4, y4, z4) and confidence (c4) of the fourth pixel in the 1D array

Figure 4.3: Assignment of the individual depth data for every segment. The input data `data3` is shown here as an example. As the depth data is stored in an array of type `float` and the input variables are of type `uint16_t`, a type conversion is performed for each variable. Before this, the value is multiplied by a factor of 1000 to convert the unit from m to mm. The address of the depth value in the 1D array results from the address of the 1D array, an offset, the index of the current segment of the depth image, the number of input variables of the slave and the input data index.

The standard constructor of the `EasyCAT` class is used in the *main.cpp* file. This means that the asynchronous mode is defined when writing the DPRAM and pin 24 of the Raspberry Pi's GPIO. The following settings have been made in the `bool EasyCAT::Init()` function of the *EasyCAT.cpp* file:

- `bcm2835_spi_setDataMode(BCM2835_SPI_MODE0);`
  This function sets the polarity ($CPOL = 0$) and the clock phase ($CPHA = 0$) of the SPI.

- `bcm2835_spi_setClockDivider(20);`
  This function sets the clock divider factor of the SPI. With a set base clock of 750 MHz, an SPI clock divider of 20 results in an SPI clock frequency of 37.5 MHz. This is also the minimum possible factor at a clock frequency of 750 MHz. The maximum SPI clock frequency of this system is approximately 38 MHz. Stable data transmission is no longer possible at higher clock frequencies.

During runtime of the application, it must be guaranteed that each segment of the depth image is transmitted via EtherCAT. As the asynchronous mode is set in the ESC, the cycle time of the PLC must be shorter than the shortest time required to transfer a segment via SPI in order to prevent data loss. Therefore, the cycle time of the PLC is selected based on the measured SPI transmission times. Figure 4.4 shows the times of 50000 measurements. It can be seen that a transmission task requires at least 0.15 ms. It can be assumed that a cycle time of 0.1 ms is sufficient for loss-free data transmission.

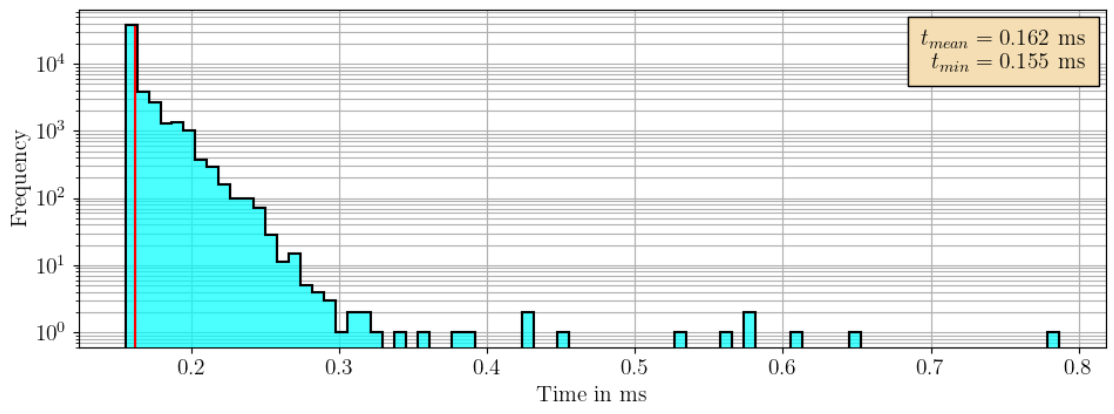


Figure 4.4: SPI transmission time for a segment. The times of 50000 transmissions are shown. The red line shows the average value of all times.

4.5 Implementation of the camera device in TwinCAT3

As already mentioned in Section 4.4, the PLC garantees a cycle time of 0.1 ms. In TwinCAT3, the cycle time is adjusted in the real-time settings. As different program structures have different processing times, it is advantageous to map these to different tasks. The cycle times of the tasks can be customised according to requirements.

The merging of the individual segments into a depth image is realised with a code designed in Structured Text (ST). This is shown in Listing 4.9. The data is written to the corresponding position in the 1D array via a simple loop. iterating through each depth data input variable in the segment. The address of the input variable specifies the segment index. The 1D array is defined as a global variable.

Listing 4.9: Implementation of the merging of individual data segments into a complete depth image. A segment consists of 301 input variables for depth data (`inputArray`) and one input variable containing the segment index (`address`).

```
PROGRAM PROCESS_DATA
VAR
   inputArray AT%I* : ARRAY[1..301] OF UINT;
   address AT%I*: UINT;
   i: UINT;
END_VAR

FOR i := 1 TO 301 BY 1 DO
    GVL.depthData[address * 301+i] := inputArray[i];
END_FOR
```

4.6 Implementation of the camera device in an industrial application

The application in which the EtherCAT-capable depth camera will be integrated is in the field of collision avoidance for robotic systems. The safety distances between robots and people in the collaboration space is evaluated according to the DIN ISO/TS 15066:2016 [22] standard based on the measured detection times of the camera system [22]. This safety margin allows the system to be benchmarked and creates comparability with state-of-the-art applications.

ISO/TS 15066:2016 describes the safety requirements for the integration of collaborative industrial robot systems. The operating characteristics of collaborative robot systems differ significantly from those of conventional robot systems in that in collaborative robot operation the operators can work in the immediate vicinity of the robot system. Energy is applied to the drive elements of the robot and physical contact between an operator and the robot system can occur within the collaboration space.

With this operating mode of velocity and distance monitoring, the robot system and operator may move simultaneously in the collaboration space. Risk reduction is achieved by maintaining at least the safety distance between the operator and the robot without interruption. If the distance is reduced to a value below the safety distance, the robot system stops. If the operator moves away from the robot system, the robot system can automatically resume movement as long as at least the safety distance is maintained. If the robot system reduces its velocity, the safety distance is reduced accordingly.

With variable values, the maximum permissible velocities and the safety distances can be set continuously on the basis of the relative velocities and distances of the robot system and the operator. On the contrary, with constant values, the maximum permissible velocity and the safety distance must be determined by the risk assessment as the most unfavorable cases in the entire course of the application. This means that a reduction in velocity depending on the measured distance between the robot and the human is conducive to maintaining operation.

The safety distance, S_p, at the time t_0 is described in the standard DIN ISO/TS 15066:2016 by the following equation:

$$S_p(t_0) = S_h + S_r + S_s + C + Z_d + Z_r \tag{4.2}$$

with the parameters:

S_h	The contribution to the safety distance due to the change in the position of the person.
S_r	The contribution to the safety distance that can be attributed to the response time of the robot system.
S_s	The contribution to the safety distance due to the stopping distance of the robot system.
C	The penetration distance, as defined in ISO 13855. This is the distance by which a body part can penetrate the detection zone (the sensor field) before it is detected.
Z_d	The position uncertainty of the operator in the collaboration space, measured by the presence detection device, resulting from the measurement tolerance of the sensor system.
Z_r	The position uncertainty of the robot system resulting from the accuracy of the system for measuring the robot position.

With the definition of the safety distance according to Equation (4.2), a concept of a safety application can be designed and implemented.

4.6.1 Experimental setup

A description of the structure is essential for designing the experimental application. The main components of the setup include the EtherCAT-capable depth camera, a 5-axis gripper arm robot and a desktop PC with TwinCAT3. The depth camera is attached to the superstructure of a 5-axis gripper arm robot. The Field of View (FoV) of the camera covers part of the robot's working area. The robot only operates in this area. The robot and depth camera, including the labelled FoV, are shown in Figure 4.5.

The robot used is a 5-axis robolink RL-DP-5 gripper robot from igus [54]. Each axis of the robot is controlled by a stepper motor with incremental encoder. These are connected directly to EtherCAT terminals for stepper motors with incremental encoders. An EtherCAT bus coupler connected to the terminals ensures the EtherCAT capability of the robot.

The pixel information of the depth camera transmitted via EtherCAT only refers to the distance between the pixel and the camera. To reconstruct a three-dimensional image, Equation (2.2) in Section 2.1.1 are required. To do this, the intrinsic parameters of the camera must be known. These can be obtained via the *Royale* API of flexx2. Listing 4.10 shows the code structure. The `getLensParameters(...)` function of the camera device returns the intrinsic parameters.

Listing 4.10: Output of the intrinsic parameters via the *Royale* API of the flexx2.

```
royale::LensParameters parameters;
if (royale::CameraStatus::SUCCESS != cameraDevice->getLensParameters(
    parameters)) {
    return 1;}
cout << "Lens focal length: " << lens.focalLength.first << " / " <<
    lens.focalLength.second << endl;
cout << "Principal point: " << lens.principalPoint.first << " / " <<
    lens.principalPoint.second << endl;
```

Due to the fixed mounting of the camera on the robot body, the extrinsic camera parameters are transferred to the robot system by a simple transformation. The transformation matrix is measured and read manually. The camera coordinates are transformed into the origin of the first axis of the robot. The position of the coordinate origin is shown in Figure 4.6 a). The coordinate transformation between the camera coordinate system and the robot coordinate system for each pixel is as follows:

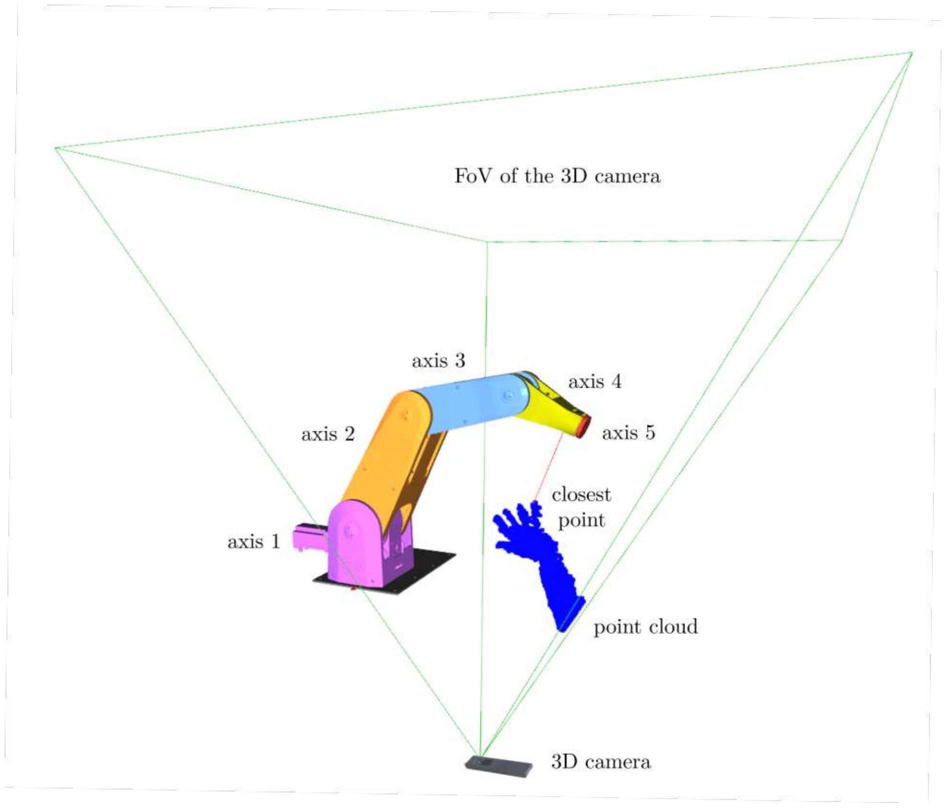


Figure 4.5: Graphical representation of the experimental setup. The flexx2 depth camera is mounted on the superstructure of a 5-axis gripper arm robot. The green lines represent the FoV of the camera. This shows how an arm reaches into the working area of the robot. The closest point of the detected and calculated point cloud to the robot defines the distance between the robot and the object.

$$\begin{bmatrix} x_r \\ y_r \\ z_r \\ 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0 & -80 \\ 0 & -0.9890 & 0.1478 & 286 \\ 0 & 0.1478 & 0.9890 & -525 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix}. \tag{4.3}$$

$\{x_r, y_r, z_r\}$ are the coordinates of the robot and $\{x_c, y_c, z_c\}$ are the coordinates of the camera. All coordinates are specified in millimeters.

4.6.2 Functional principle of the application

Since the minimum safety distance in the collaboration space depends on the robot velocity, two modes are implemented when moving the axes, the *stop mode* and the *reduced velocity mode*. The robot performs simple positioning movements with axes 2, 3 and 4 only in order to remain in the camera's FoV. All axes should stop when an object falls below the safety distance. The velocity mode should also be activated if an object falls below a certain distance. In this mode, all axes should reduce the velocity linearly to the object-robot distance, whereby the lower bound is the threshold distance of the *stop mode* and the upper bound is selected based on the robot dynamics.

The functional principle of the application is explained in more detail in Figure 4.6. Axis 1 is not moved in order to guarantee that the robot remains in the camera's FoV. Axis 5 only allows a rotational movement of the end effector and thus has no influence on the translation. Therefore, it will also stay deactivated. On this basis, three line segments are formed in the centre of each robot segment. The coordinates in the space of both end points of each segment are the axis positions in the space. These can be determined by the forward kinematics of the robot. This allows the line segments to be calculated in space.

The smallest distances of an object in space to each line segment can be determined using its orthogonal projection onto each segment. The smallest distance of all projections is the smallest distance between the object and the robot. For the robot model, an offset is chosen for each line segment based on the element of a segment that is furthest away from the line. This results in a cylindrical approximation of the individual robot segments. The resulting deviation from the model to the real robot system is irrelevant in terms of collision avoidance, as the dimensions of the model are greater than or equal to the actual robot dimensions.

The point cloud generated by the robot can be uniquely identified with the approximated model, making it possible to distinguish between the robot and the object in space. The point in the point cloud that does not belong to the robot and has the

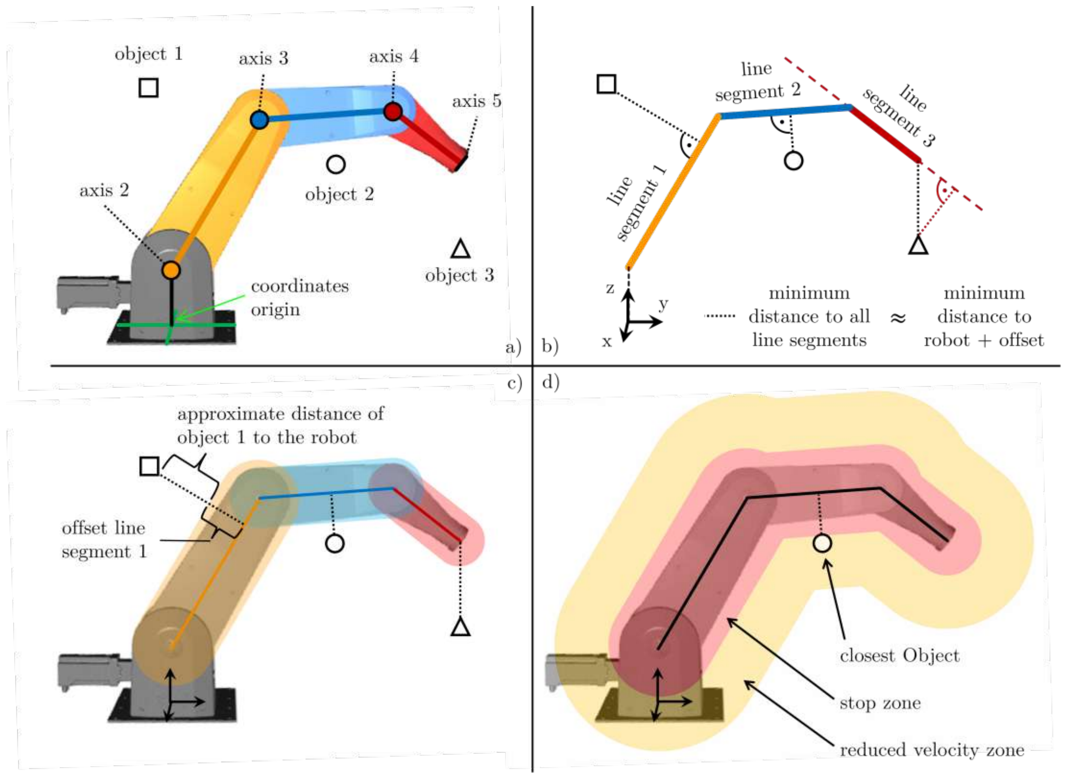


Figure 4.6: Graphical representation of the functional principle of the safety application for collaborating robot systems. Figure a) shows the robot and the axes used for the trajectory. There are also three objects in the immediate vicinity. Line segments are formed from the coordinates of the axis positions in space, which can be determined by the forward kinematics of the robot, as shown in figure b). From the orthogonal projection of all objects onto all segments, the smallest distance of each object to the robot can be determined, whereby an offset is defined for each line segment, which approximates the robot model, as shown in figure c). The positions of the objects in space are given by the point cloud generated by the depth camera. Figure d) shows the closest object to the robot and the zones for the *stop mode* and *reduced velocity mode*.

smallest distance to the approximated robot model is therefore the smallest distance between any object in space and the robot. In this way, it is also possible to distinguish whether the closest object to the robot is located in the *stop mode* zone, in the *reduced velocity mode* zone or outside both zones.

4.6.3 Direct Kinematic of the robot

In order to be able to calculate the line segments of the robot, the axis position in space must be defined. This can be done using direct kinematics.

The kinematic model of a manipulator can be obtained in the form of a kinematic chain by using systematic schemes for joints and rigid links. This is also known as direct kinematics or forward kinematics. A kinematic chain is obtained by determining the geometric quantities and the kinematic parameters. These parameters define the relative position and orientation of a joint in relation to a neighbouring joint depending on the variable coordinates [55].

The Denavit-Hartenberg convention [56, 57], for example, can be used to determine these geometric variables and kinematic parameters. This scheme specifies the minimum number of parameters required to describe the geometry of a link between two joints and the joint variables. The following DH parameters are defined to describe the five-axis robot used in this application.

Table 4.1: DH parameter of the application.

Index i	θ_i	d_i	a_i	α_i
1	0 rad	157 mm	0 mm	$-\pi/2$ rad
2	$-\pi/2$ rad	0 mm	350 mm	0 rad
3	$+\pi/2$ rad	0 mm	270 mm	0 rad
4	$-\pi/2$ rad	0 mm	0 mm	$-\pi/2$ rad
5	0 rad	190 mm	0 mm	0 rad

The DH parameters θ_i are added to the joint angles of the respective axes in the application. When the individual axis positions are determined in 3-dimensional space, line segments can be formed between the axis points. Each point of the point cloud can thus be projected onto each line segment. This makes it possible to determine the closest point or object to the robot.

4.6.4 Implementation of the application in TwinCAT3

The entire safety application is implemented in TwinCAT3. The various interacting processes are described in four structured programming organisation units (POUs) of the program type. A programming object requires an extension in the sense of a method. The structure of the PLC project is shown in Figure 4.8.

In the `PROCESS_DATA` program, the depth data segments are reassembled into a complete depth image in the sense of a 1D array sorted in rows. The input variables of the camera device in the I/O area are linked to an input array defined in `PROCESS_DATA`. The `MAIN` program is responsible for controlling robot processes. The actuators are linked to the defined axis variables here. `KINEMATIC` uses the Denavit-Haarenberg convention to calculate the axis coordinates in space. `POINTCLOUD` calculates the point cloud using depth data and performs various operations to eliminate irrelevant points caused by surrounding objects, noise, etc. The `KINEMATIC` and `POINTCLOUD` programs are called up in the `MAIN` program.

Data are exchanged between programs via global variables. These are defined in the Global Variable List (GVL) `GVL`. For the application, the TwinCAT3 libraries `Tc2_MC2` for axis control and `Tc3_Vision` for displaying the depth data are added to the PLC project.

Two tasks with different cycle times are referenced for processing the program structures. To ensure fast depth data processing, the program `PROCESS_DATA` is referenced to a task with a cycle time of 0.1 ms. Due to the increased computational effort of point cloud processing, the remaining programs are processed cyclically with 10 ms.

The programs `MAIN` and `POINTCLOUD` are explained in more detail below. A more detailed structural overview is shown in Figure 4.7.

MAIN

In addition to the program calls of `KINEMATIC` and `POINTCLOUD`, a state machine is implemented for robot control. A state variable of type `STATE` specifies the current state. The states are as follows:

- **POWER** activates the axes,
- **HOMING** performs an initialization run based on the incremental encoders of the axes,
- **INIT_POS** moves the robot to the initial position,

- **DRIVE1** moves the robot to position 1,
- **DRIVE2** moves the robot to position 2.

An axis override control is also implemented in `MAIN`. This controls the override variables and thus the velocity of all axes based on the distance of the closest point. The axis override control is activated as soon as all axes have been initialized, and the actual joint angles of the incremental encoders are read out.

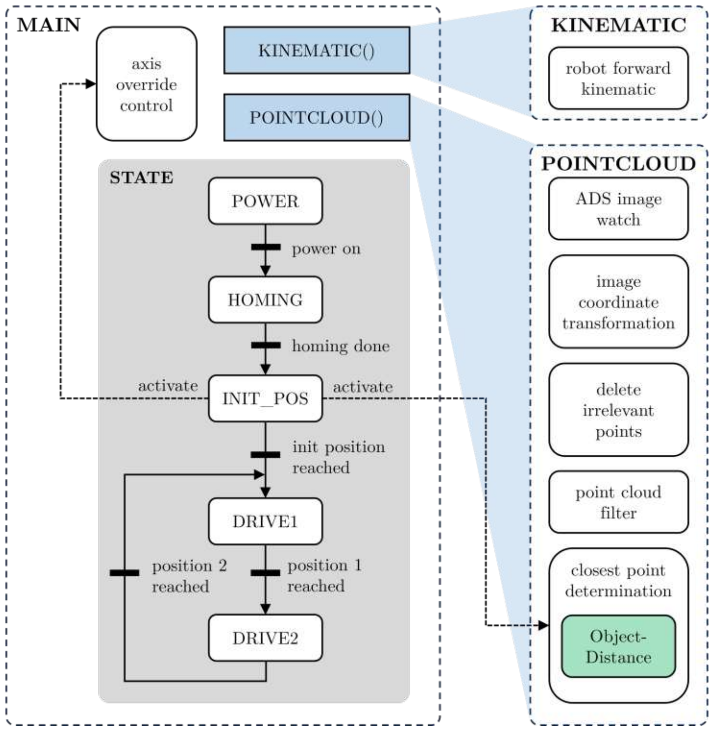


Figure 4.7: Display of the various sub-processes of the programs `MAIN`, `KINEMATIC` and `POINTCLOUD`. After initializing the robot, the functions for determining the nearest point and for axis control are activated. The `ObjectDistance` method calculates the smallest distance for each relevant point in the point cloud.

POINTCLOUD

`POINTCLOUD` is used to process and prepare the depth data, which is ultimately used to determine the nearest point.

The TwinCAT3 Automation Device Specification (ADS) Image Watch function is used for graphical display of depth images. It retrieves the images from the PLC and displays them in the development environment. The 1D depth data field is formatted and normalized as a grayscale image.

The point cloud is generated using the depth image and the intrinsic of the camera. An image coordinate transformation is required to integrate the camera data into the robot system. This is described in Equation (4.3).

To be able to distinguish between robot and object, the robot model is removed from the point cloud. This is possible by forming cylindrical areas around the line segments between the axes. Irrelevant points are also eliminated to save computing power. These include, for example, points on the ceiling or the superstructure.

ToF cameras are rather sensitive to edges, strongly reflective and black objects. Because the robot model used has a black surface, a strong outbreak occurs in the surrounding area. This requires some sort of noise suppression. A filter implemented for this purpose deletes points from a point cloud if there are too few nearby neighbors. Manually adjustable parameters such as window size, maximum permissible point distance, or minimum number of points per window are determined experimentally.

The remaining points of the point cloud are used for the closest point determination. This function is activated only after the robot system is initialized. The `ObjectDistance` method determines the nearest segment of each point, the orthogonal projection and its distance. This allows the point to be determined at the smallest distance from the robot.

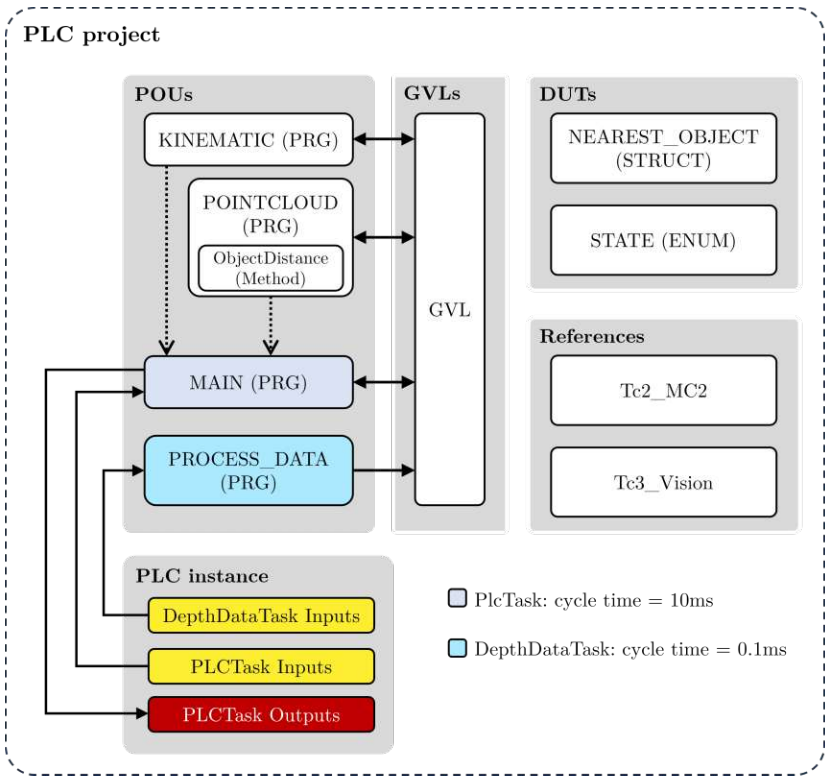


Figure 4.8: Structured overview of the PLC project. The input variables of the camera device are linked to an input array defined in `PROCESS_DATA`. The depth data is then sorted into a 1D array for depth data defined in `GVL` based on the address sent. `POINTCLOUD` processes the depth data into a point cloud and uses the kinematic chain calculated in `KINEMATIC` and the method `ObjectDistance` to calculate the closest point to the robot. This is saved together with all important information in `GVL` as structure `NEAREST_OBJECT`. `MAIN` communicates with the robot system and controls it based on the distance of the closest object.

CHAPTER 5

Experiments and Results

This chapter discusses the experiments in more detail and presents the results. First, in Section 5.1 the viability of the proposed integration approach of depth data from a 3D camera into the Ethernet for Control Automation Technology (EtherCAT) fieldbus is demonstrated. Thereafter, the latency and frame rate of the flexx2 in The Windows Control and Automation Technology 3 (TwinCAT3) are then evaluated in Section 5.2. Subsequently, the data reduction methods described in Section 5.3 are intended to reduce latency and increase the frame rate of the camera measured in TwinCAT3. Finally, Section 5.4 defines the resulting safety distances in collaboration space based on the total detection times of the flexx2.

5.1 Evaluation of the transmitted depth image via EtherCAT to TwinCAT3

The first objective is to transfer depth images to a controller via EtherCAT. For this evaluation, the entire flexx2 image resolution (224 × 172 pixels) is transmitted through EtherCAT. The depth defined resolution is 1 mm and is transmitted with 16 bit per pixel. As flexx2 has a range of 0.1 m - 4 m with an accuracy of 1 % of the measured distance, this type of transmission can be described as virtually loss-free in terms of depth resolution. The camera is configured to use of double modulation frequencies at 30 fps.

The transmitted depth image is shown in the right image of Figure 5.1. The Automation Device Specification (ADS) image watch function displays the depth image. In addition, the depth data is exported to a Python file via ADS and visualized as point cloud using Open3D, shown in the left image of Figure 5.1.

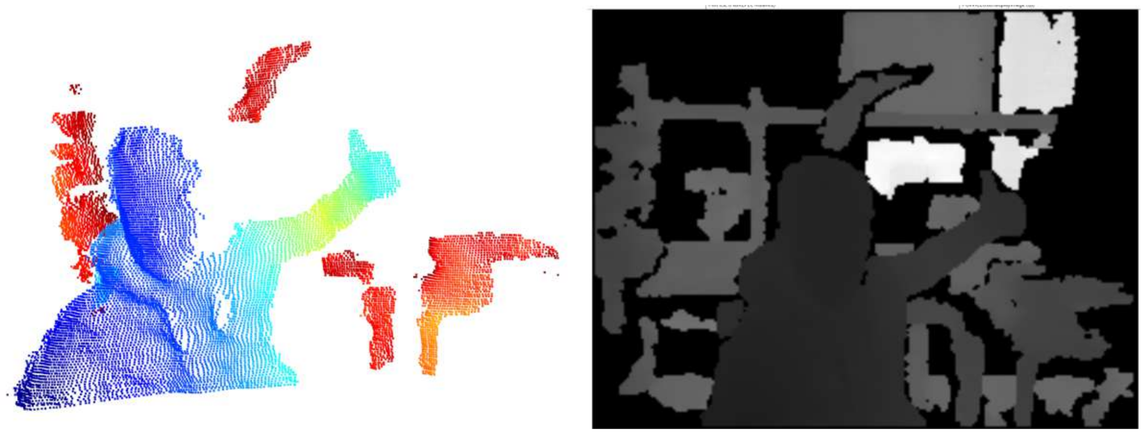


Figure 5.1: Display of the depth data transmitted via EtherCAT. The left-hand image shows the point cloud of the depth image. The depth data is exported to a python file via ADS and visualized using the Open3D library. The right image shows the depth image as a gray scale image displayed via the ADS image monitoring function.

5.2 Evaluation of the latency and frame rate of flexx2 in TwinCAT3

The safety distances for collaborative robots specified in the DIN ISO/TS 15066 standard [22] are determined in relation to the measuring system by the detection time, the position uncertainty of the operator in the collaboration space, and the penetration depth. In this regard, the position uncertainty and the penetration depth are determined by the calibration of the camera, the accuracy of the coordinate transformation between the robot and the camera system, the resolution of the camera and its depth resolution. The detection time results from the latency of the camera, including the transmission system and the frame rate. Moreover, it is essential for determining the safety distances in the collaborating workspace.

To determine the detection time, an infrared LED is mounted in the camera's Field of View (FoV). This LED emits infrared light with the same wavelength of 940 nm as the flexx2 transmitter. If the LED emits light in the depth image, no depth data is available at the position of the LED. The reason for this is that the modulated light from the camera interferes with the light from the LED. As a result, the sensor array cannot detect any phase difference between the emitted and captured light. Therefore, the measured distance is 0 m. If the LED does not emit any light, only the modulated reflected light is received by the depth camera, so the distance between the camera and the LED can be measured.

When sending the output signal to switch on the LED, the time is saved in a time variable in TwinCAT3. If the switched-on LED is detected in a depth image, this time is saved in a second time variable. The difference results in the detection time of the EtherCAT-capable camera system. This measurement process is repeated with a random time delay until a sufficient number of measurements is reached (10,000 measurements), in order to guarantee statistical significance of the results.

5.2.1 Setup

In addition to the depth camera, a EK1100 EtherCAT bus coupler from Beckhoff Automation GmbH [58] is connected to the EtherCAT fieldbus to measure the detection time. The EL2008 digital output terminal [59], also from Beckhoff Automation GmbH, is mounted on the bus coupler. The terminal output voltage is 24 V. The TSAL6100 infrared LED [60] is connected to this output terminal and is located in the field of view of the flexx2. The forward voltage of the LED is 1.35 V with a forward current of 100 mA and a illumination time of 20 ms. The LED is connected in series with a 5 W, $R = 220\,\Omega$ resistor to limit the current. The setup is shown in Figure 5.2.

The infrared LED is firmly screwed to the structure of the application for all measurements, which results in a fixed position of the LED in the depth image. The distance between the LED and the camera is 0.51 m.

In order to evaluate only the camera system, a low latency of the light source components is essential. The LED has a rise and fall time of 15 ns. The EL2008 output terminal switches with a turn-on time of 60 µs and a turn-off time of 300 µs. A transmission time of 1 µs is specified for the EK1100 bus coupler for ten modules with 32 bit input/output each. As the latency of the light source should be kept as low as possible in order to approximately evaluate only the camera system, only switch-on times are measured due to the output terminal. This results in a total maximum latency of approximately 61 µs of the light source.

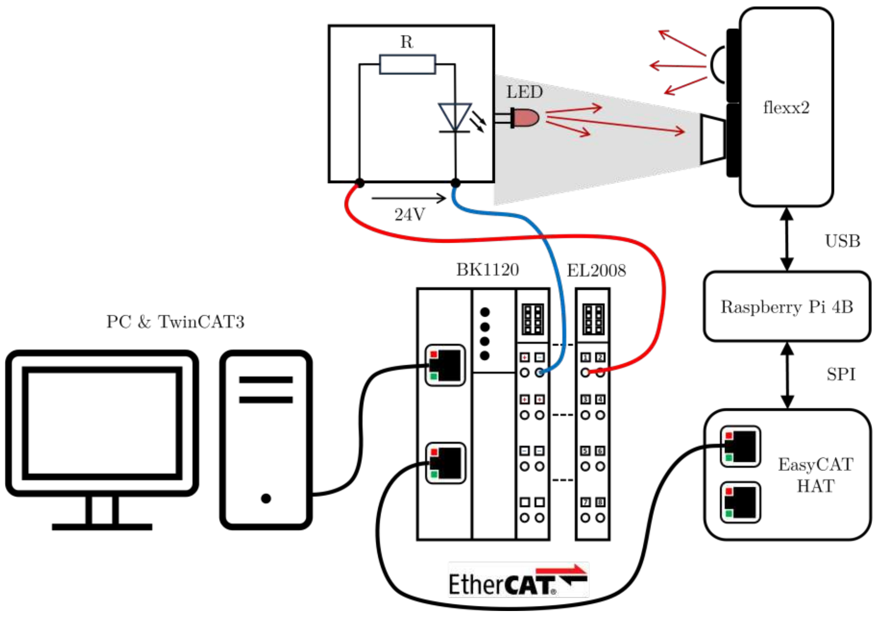


Figure 5.2: System overview of the setup to measure the detection time. A EK1100 EtherCAT bus coupler [58] with the EL2008 output terminal from Beckhoff Automation GmbH [59] and the depth camera it connected to PC with TwinCAT3 via the EtherCAT field bus. The terminal EL2008 switches an infrared LED, which is mounted in the field of view of the flexx2, on and off. This switching process can be seen in the depth image. To determine the latency of the camera system, the time from the moment the LED is switched on until it is detected in the depth image is measured.

5.2.2 Implementation

Evaluation of latency and frame rate is implemented in the program `PROCESS_DATA`. The frame rate is determined by measuring the time required to transmit all segments of 1000 images. To calculate the average frame rate over these 1000 frames, 1000 is divided by the measured time.

When measuring the latency, the time that elapses between the LED being switched on and the recognition in the depth image is measured. A 5×5 grid is viewed in the image where the LED is located. This is shown in Figure 5.3. The mean value of all distances in the grid is calculated. If this is below the value of a defined threshold, the LED is turned on, as some pixels on the grid have the value 0. If the average value is above the threshold, the LED is switched off. Since the mean value of the 5×5 grid, which is the mean distance of this 5×5 grid, is approximately 230 mm when the LED is switched on and approximately 470 mm when it is switched off, a threshold mean value of 350 mm of the 5×5 grid has proven to be well suited for the measurements.

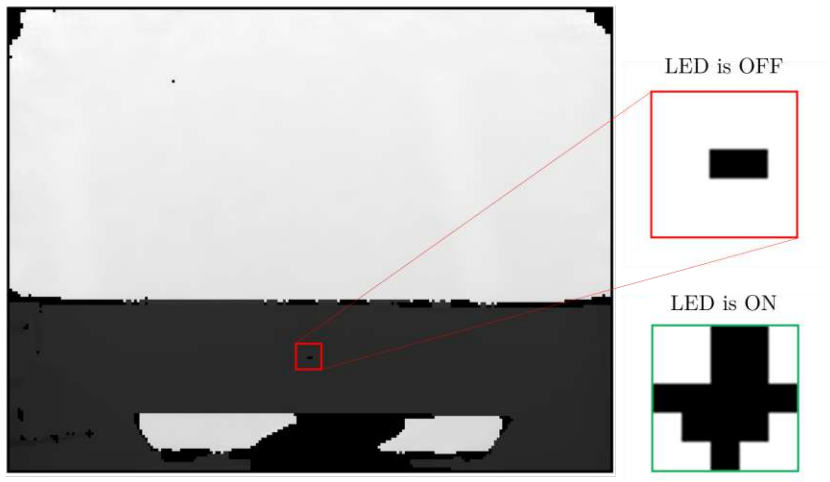


Figure 5.3: Depth image for measuring the latency of the camera in TwinCAT3. The red window shows the area around the infrared LED when it is switched off. Two pixels remain black due to the transparent surface of the LED. In this case, no modulated light from the depth camera is reflected. The green window shows the area around the LED when it is emitting light. Due to the interference between the light from the LED and the light from the depth camera, the sensor array can no longer detect a phase difference. The measured distance is therefore 0. The detection time of the camera results from the time measurement between switching on the LED and detection in the depth image.

5.2.3 Results

The detection times of four different camera modes are measured. For the first two modes, the camera operates at 20 fps and 30 fps, respectively, and double frequency modulation. Modes three and four operate with a single modulation frequency at 45 fps and 60 fps, respectively. In all modes, 10,000 measured values are recorded. The results are shown in Figure 5.4. The average frame rate of the camera in TwinCAT3 is 20.1 fps in mode MODE_9_20FPS, 28.6 fps in mode MODE_9_30FPS, 41.5 fps in mode MODE_5_45FPS, and 41.6 fps in mode MODE_5_60FPS.

5.2.4 Discussion

The results shown in Figure 5.4 are now discussed and analyzed in more detail. The histograms show the detection times of the depth camera in TwinCAT3 over 10000 measured values in order to guarantee statistical significance of the results. In addition, the the mean value is displayed as a red line. The measured values show that the average detection time is significantly higher than the reciprocal of the frame rate of the camera. Furthermore, a high variance in the values can be noted. For a better understanding of the measured values, a thought experiment is carried out:

If the transmission latency is negligible, one would expect the histogram to resemble a uniform distribution on an interval corresponding to the reciprocal of the frame rate, and the mean value of all measured detection times to correspond to the value of half the frame rate. However, if the latency is not negligible but constant, all values of the uniform distribution would increase by the latency. If the latency time varies, the interval of the uniform distribution would extend. This can be observed in the top histogram of Figure 5.4.

Except the mode MODE_9_20FPS, the time interval between lowest and highest measured detection time in Figure 5.4 do not correspond to the reciprocal of their frame rate. This is not only due to the variance of the latency, but much more to the type of implementation. As already explained in Section 4.3, the Royal Framework calls the function `onNewData(...)` of the data listener for a new frame. The frame is calculated beforehand from the measured sub-frames. The depth processing thread of the *Royale* library may block while waiting for the function `onNewData(...)` to return. If this function exceeds the elapsed time between capturing frames, there may be a delay between the capture and the function `onNewData(...)` for the next frame. The *Royale* Application Programming Interface (API) may even skip frames to catch up.

In the `onNewData(...)` function, the data is sent via Serial Peripheral Interface (SPI). If the transmission via SPI of an entire frame is not completed before the next

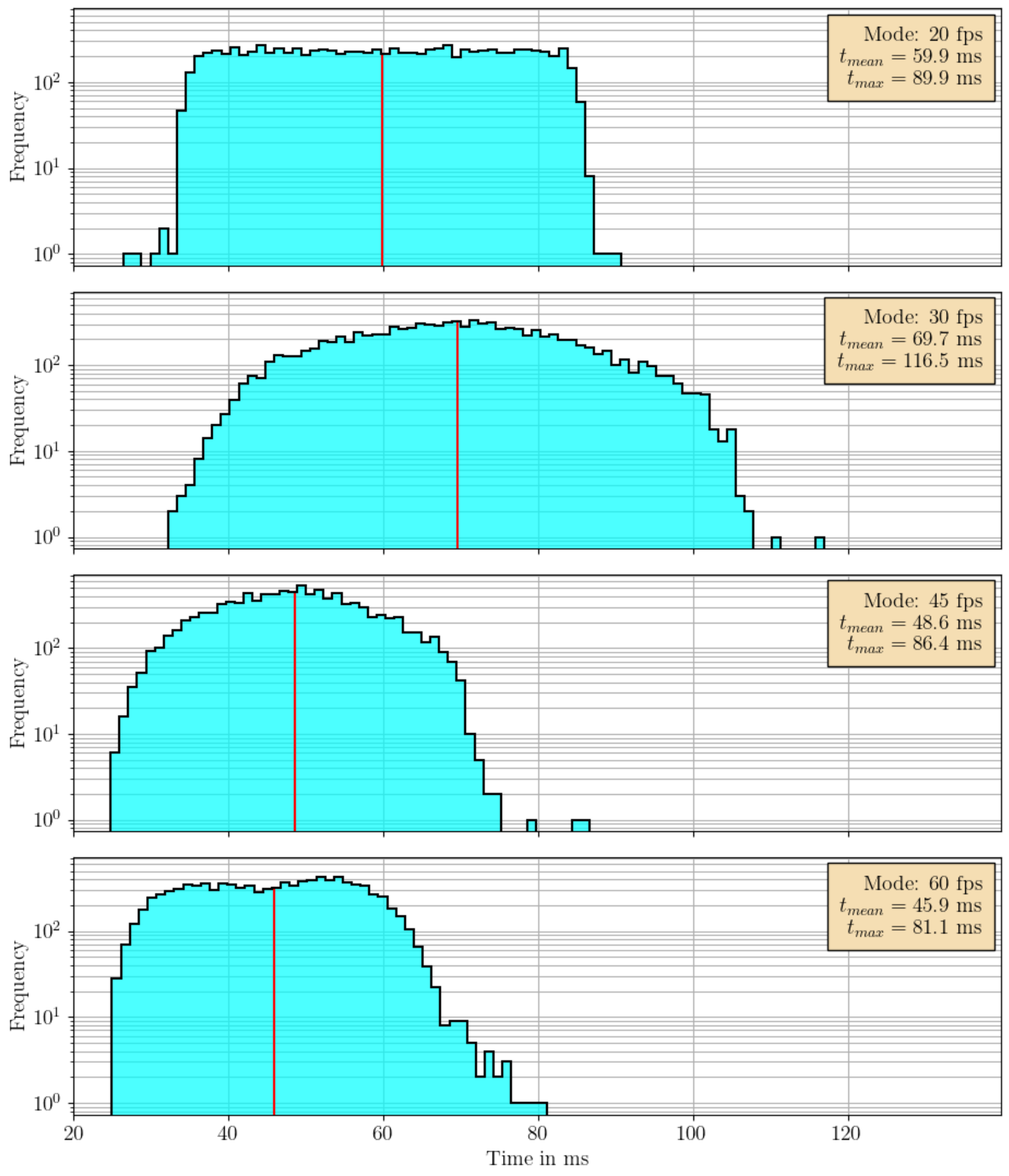


Figure 5.4: Measured detection times, defined as the time span between switching the LED on to detecting the LED in the depth image. The results of 10 000 measured values from four different operating modes are displayed. The histograms contain the measured values from MODE_9_20FPS, MODE_9_30FPS, MODE_5_45FPS and MODE_5_60FPS. The red line shows the mean value of all measured values.

frame is processed, there is a delay in frame processing or even a frame drop. This is also clearly visible in TwinCAT3. In the mode MODE_9_30FPS, a frame rate of 28.6 fps is measurable instead of the desired 30 fps.

It is also noticeable that individual detection times are significantly longer than the other values. To be able to describe this better, the times for a transmission via SPI are measured on the Raspberry Pi. This is shown in Figure 5.5. The times apply to a transmission of 604 B.

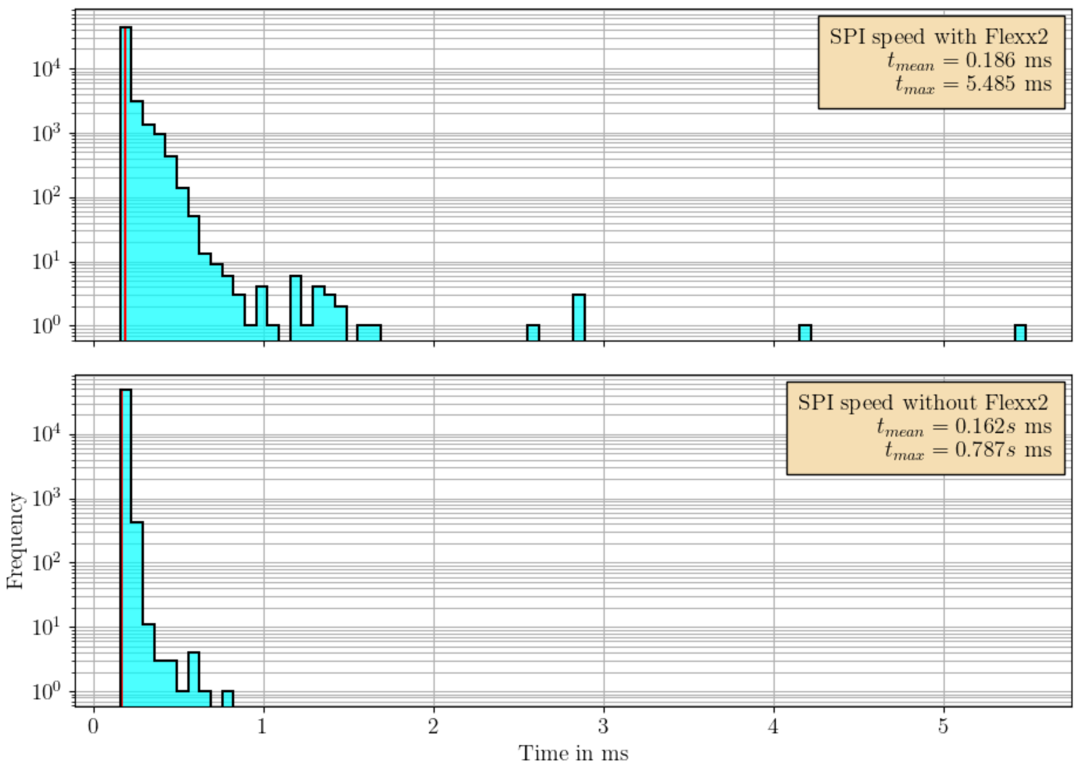


Figure 5.5: SPI transmission times measured on the Raspberry PI 4 Model B with the camera running (MODE_9_30FPS mode) and with the camera switched off. 604 bytes are transferred. For the measurement, 50000 values are recorded.

Transmission times are recorded with and without the camera running. It can be clearly seen that the camera influences the SPI transmission. It is assumed that the limited computing power of the Raspberry Pi 4 Model B has a significant influence on the SPI transmission. Individual SPI transmissions are also notable to take significantly longer than average. This could also be the reason for the increased duration of individual detection times. However, there is a lower threshold in the transmission time of the SPI. This means that the transmission time of a frame with this implementation can only be reduced by reducing the number of segments per frame. The detection time is therefore directly related to the number of transmitted segments per depth frame.

5.3 Evaluation of the detection times of flexx2 in TwinCAT3 using data reduction methods

The detection times of flexx2 in TwinCAT3 can be reduced by the number of segments per depth image, provided the segment size is constant. This can be implemented in two different ways while maintaining the frame rate. Either the number of pixels can be reduced or the number of bits per pixel. Both methods are explained in more detail in this chapter.

5.3.1 Implementation

One method to lower the data transfer rate of the flexx2 is to reduce the number of bits per pixel. The smallest number of bits that an EtherCAT slave variable can have is 8 (1 B). The largest unsigned integer that can be displayed is 255, which means that if a pixel is represented by just one byte, a depth of 0 m to 2.55 m can be displayed with a depth resolution of 1 cm, for example. Thus, this method sets all values greater than 2.55 m to 0 m. Therefore, the data of a depth image is reduced by a factor of two.

Another method of reducing the data transfer rate of the flexx2 is to reduce the number of pixels per depth image. This is achieved, for example, with a minimum pool operation. This is shown in Figure 5.6.

For example, a 2×2 window is used to iterate over the depth image. The smallest value except 0 m is retained, the rest is ignored. In this case, the resolution of a flexx2 depth image is reduced from 224×172 pixels to 112×86 pixels. This means that a 2×2 pooling operation reduces the data volume of a depth image by a factor of four, a 4×4 pooling operation by a factor of 16.

On the one hand, the minimum pooling operation leads to noise suppression in the depth image, and, on the other hand, this method is suitable for applications in the field of collaborative robot systems. If the entire workspace can be captured three-dimensional by depth cameras, the dimensions of all captured objects or persons remain the same or are even enlarged by the minimum pooling operation. By enlarging the objects, safety stops would be triggered earlier, and thus complies with the requirements given in collaborative workspaces.

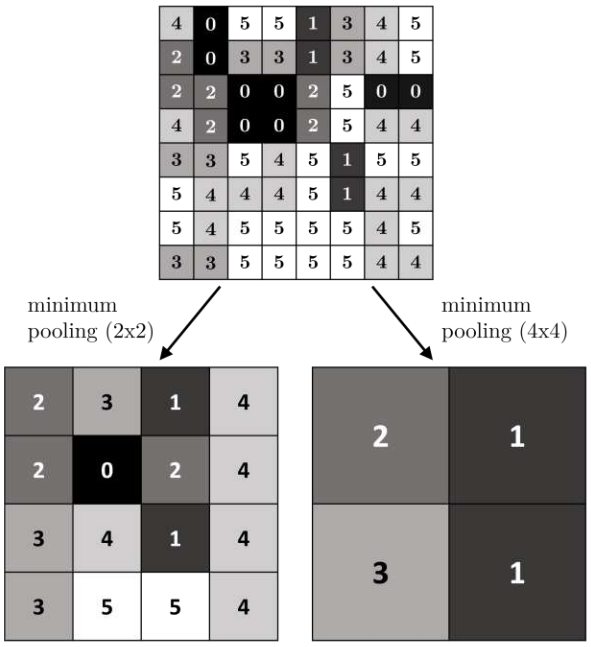


Figure 5.6: Display of two minimum pooling operations from one depth image. The 16x16 pixel depth image in the upper half of the figure shows the depth values displayed as a grayscale image. This image is processed once with a 2×2 pooling operation and once with a 4×4 pooling operation. The smallest value in the pooling window other than 0 is retained, the remaining pixels are ignored.

5.3.2 Results

When evaluating the data reduction methods, the 3 different modes MODE_9_30FPS, MODE_5_45FPS and MODE_5_60FPS are compared. The methods used are the reduction of the depth information to 8 bit per pixel, a 2×2 minimum pooling operation and a 4×4 minimum pooling operation. The results of the recognition times for the three different data reduction methods are shown in Figure 5.7, Figure 5.8 and Figure 5.9, respectively.

5.3.3 Discussion

The average values of all data rates measured in TwinCAT3 over 1000 frames correspond to the data rate of the specified mode. This means that the processing of the `onNewData()` function of the camera framework does not cause any delays or frame losses.

The method to reduce the depth data halves the number of bytes per depth image. The minimum pooling operation 2×2 reduces the bytes per depth image by a factor of four, the minimum pooling operation 4×4 by a factor of 16. In Figure 5.7, Figure 5.8, and Figure 5.9, a direct regression of the expected value, as well as the maximum value of all measured detection times, is recognizable in direct dependence of the number of segments transmitted via EtherCAT per depth image.

If the amount of data per depth image is reduced, fewer segments and therefore EtherCAT frames are required for transmission. As there is a latency with every SPI transmission and every EtherCAT frame also causes a latency depending on the cycle time, the total latency of the depth camera is reduced by reducing the amount of data per depth image.

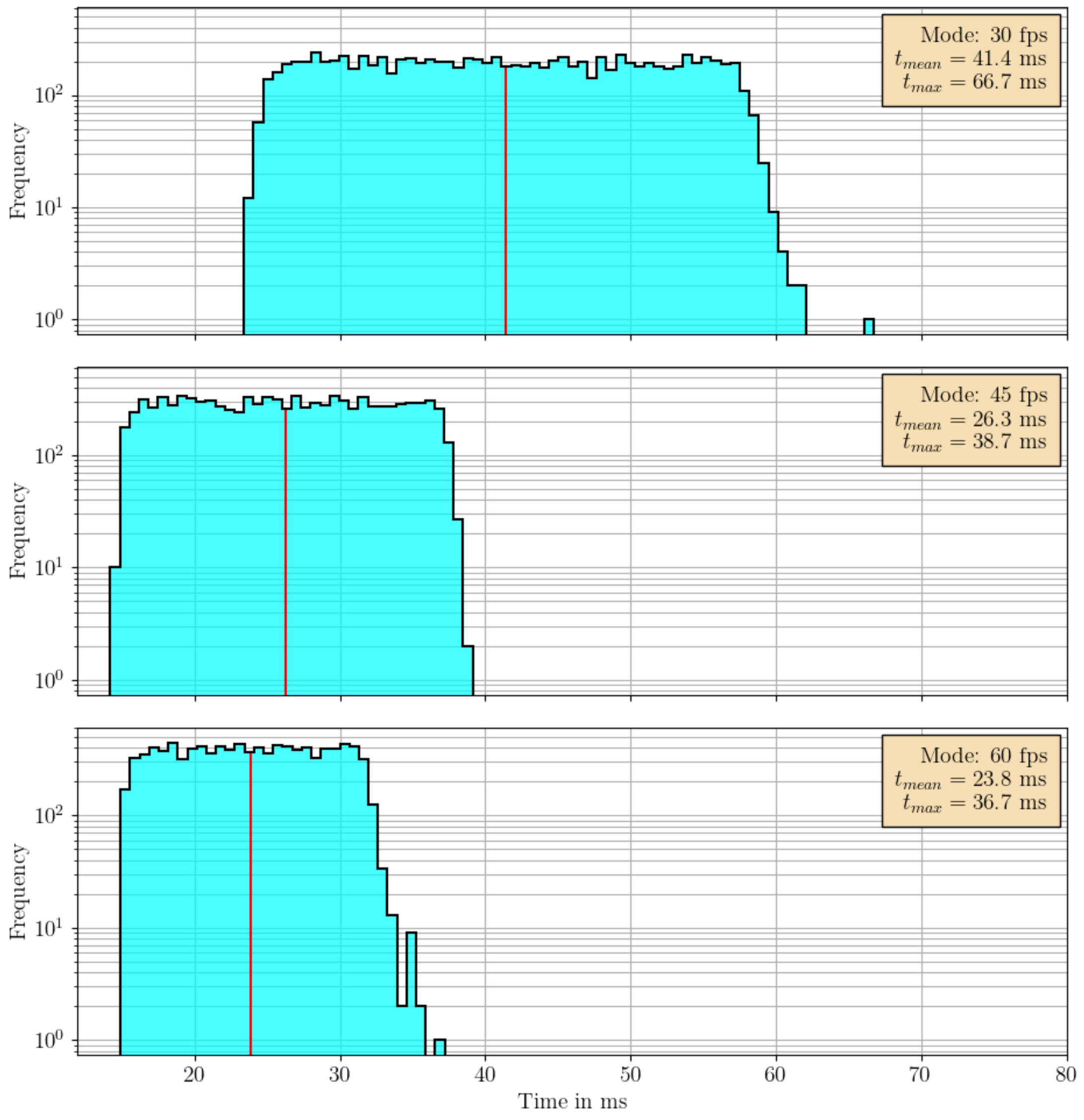


Figure 5.7: Measured detection times using the reduction of depth information to 8 bit per pixel. The results of 1000 measured values from three different operating modes are displayed. The upper histogram contains the measured values from MODE_9_30FPS, the second from MODE_5_45FPS, and the lowest from MODE_5_60FPS. The red line shows the mean value of all measured values.

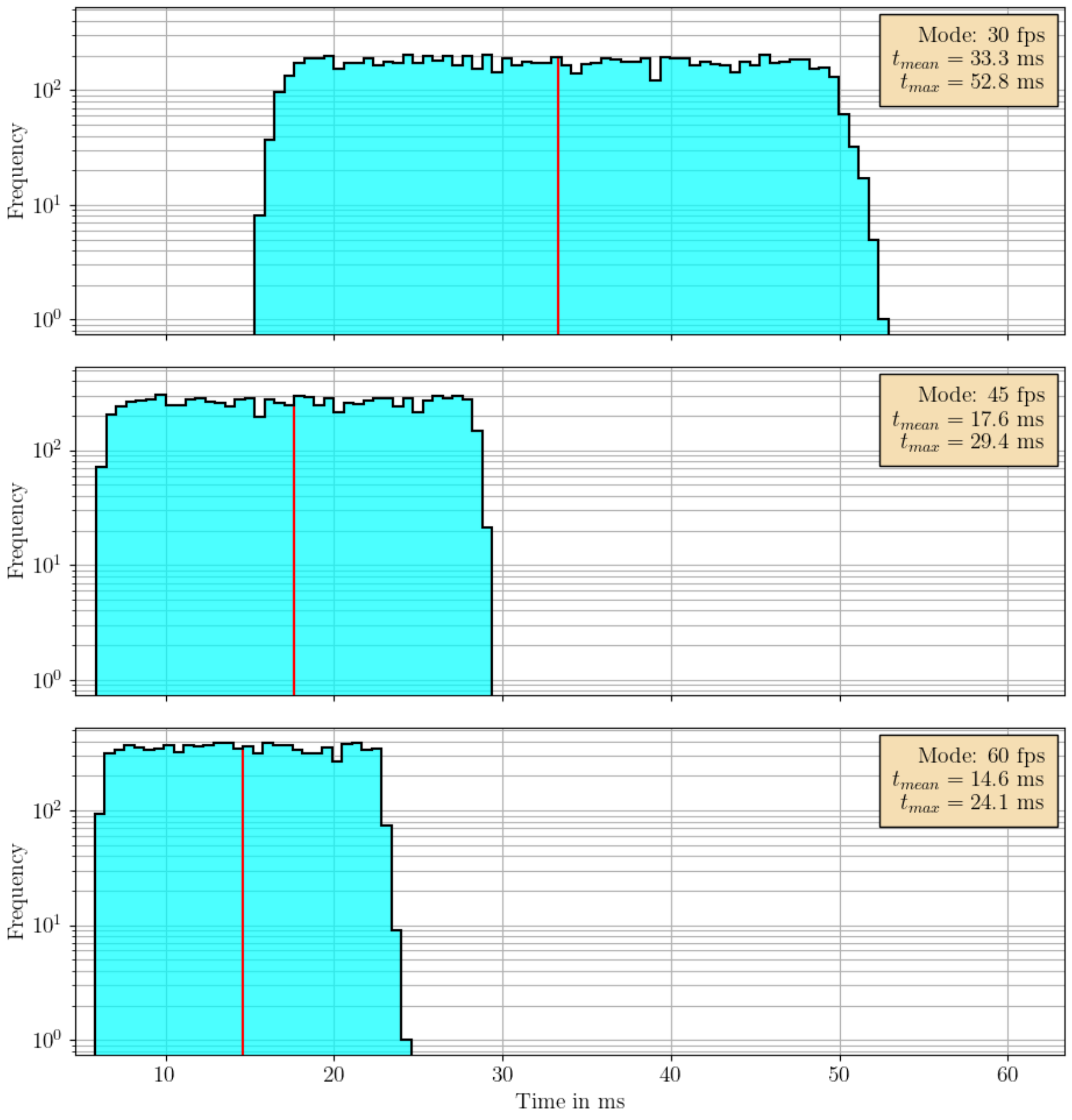


Figure 5.8: Measured detection times using the 2 × 2 minimum pooling operation. The results of 1000 measured values from three different operating modes are displayed. The upper histogram contains the measured values from MODE_9_30FPS, the second from MODE_5_45FPS, and the lowest from MODE_5_60FPS. The red line shows the mean value of all measured values.

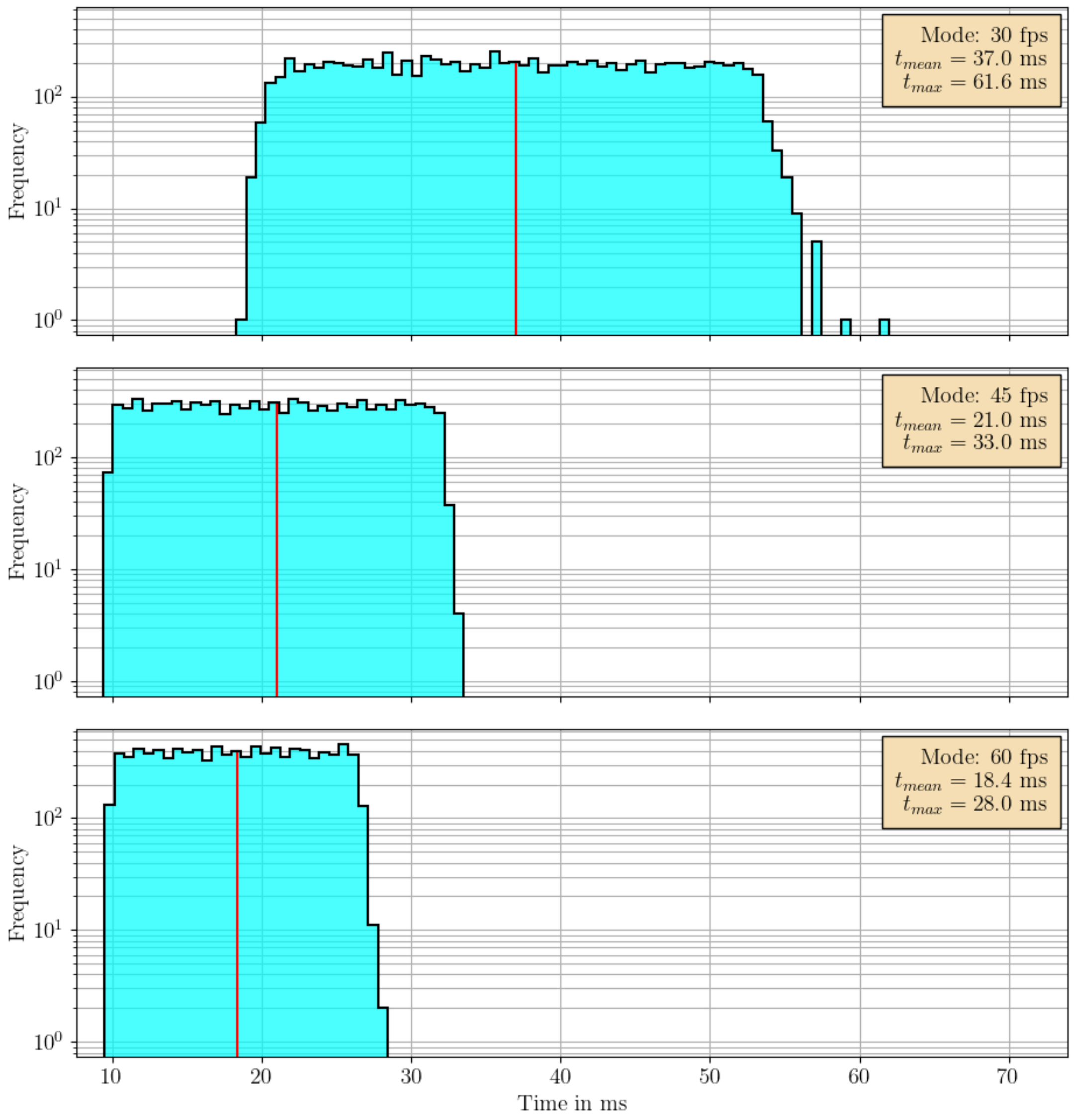


Figure 5.9: Measured detection times using the 4 × 4 minimum pooling operation. The results of 1000 measured values from three different operating modes are displayed. The upper histogram contains the measured values from MODE_9_30FPS, the second from MODE_5_45FPS, and the lowest from MODE_5_60FPS. The red line shows the mean value of all measured values.

5.4 Evaluation of safety distances in the application for collaborative robot systems

When determining the minimum safety distance, the camera system only has an influence on the parameters S_h, S_r, C and Z_d of Equation (4.2) according to [22]. For the following description of the contributions of the minimum safety distance, only the detection time of the camera system is taken into consideration.

5.4.1 Specification of the parameters for calculating the safety distances

The variable S_h represents the contribution to the distance caused by the movement of the operator from the actual time until the robot comes to a standstill. According to [22], if the velocity of a person v_h is not monitored, the design of the system must assume that v_h is $1.6\,\mathrm{m\,s^{-1}}$ in the direction in which the distance decreases the most. A constant value for S_h using the estimated human velocity ($1.6\,\mathrm{m\,s^{-1}}$) can be estimated using the following equation:

$$S_h = v_h \cdot T_r = 1.6\,\mathrm{m\,s^{-1}} \cdot T_r. \tag{5.1}$$

T_r is the detection time of the robot system, including the time for detecting the operator's position, the time for processing this signal, and the time for activating a stop of the robot, but excluding the time for the robot to come to a standstill.

The variable S_r represents the contribution to the distance resulting from the movement of the robot after a person has entered the sensor field until the control system triggers a stop. S_r depends on the directional velocity v_r of the robot towards a person in the collaboration space. To simplify the system, v_r is assumed to be a constant value defined as the highest velocity in all areas of the robot, which is the worst case in the human-robot collaboration application. The tool center point is the area of the robot that has the highest velocity during the process when all axis are aligned at maximum velocity. In this application, axes 2, 3 and 4 of the 5-axis robolink RL-DP-5 gripper robot from igus [54] are moved during the process, resulting in a maximum velocity of the tool center point as follows:

$$\begin{aligned} v_r &= v_{TCP1} + v_{TCP2} + v_{TCP3} \\ &= \frac{2\pi}{360^\circ} \cdot [\omega_{max2} \cdot (a_2 + a_3 + d_5) + \omega_{max3} \cdot (a_3 + d_5) + \omega_{max4} \cdot d_5] \\ &= \frac{2\pi}{360^\circ} \cdot [15\,^\circ\,\mathrm{s}^{-1} \cdot (0.35\,\mathrm{m} + 0.27\,\mathrm{m} + 0.23\,\mathrm{m}) + 30\,^\circ\,\mathrm{s}^{-1} \cdot (0.27\,\mathrm{m} + 0.23\,\mathrm{m}) + \\ &+ 20\,^\circ\,\mathrm{s}^{-1} \cdot 0.23\,\mathrm{m}] \\ &\approx 0.565\,\mathrm{m\,s}^{-1}. \end{aligned} \tag{5.2}$$

ω_{max2}, ω_{max3} and ω_{max4} are the maximum angular velocities of axes 2, 3 and 4 measured in degrees. a_2, a_3 and d_5 are the dimensions of the DH convention in Section 4.6.3. According to DIN ISO/TS 15066, a constant value for S_r can be estimated using the following equation:

$$S_r = v_r \cdot T_r. \tag{5.3}$$

For the contributions C and Z_d of Equation (4.2), the depth resolution of the flexx2 is assumed at a maximum measuring distance of 2 m. The maximum penetration depth C and the positional uncertainty Z_d of a person with this camera system is half the diagonal of a voxel. A voxel is calculated from the depth resolution z, the x and y dimensions of a pixel at the maximum distance. This is shown in Figure 5.10.

The maximum z-value of the voxel is calculated from the maximum measured depth d_{max} for the collaboration space and the depth resolution:

$$z_{max} = d_{max} \cdot 0.01 = 2\,\mathrm{m} \cdot 0.01 = 20\,\mathrm{mm}. \tag{5.4}$$

The maximum x and y values of a pixel are calculated from the FoV, the resolution p, and the maximum depth measured d_{max}:

$$x_{max} = \frac{2 \cdot \tan \frac{FOV_x}{2} \cdot d_{max}}{p_x} = \frac{2 \cdot \tan \frac{56^\circ}{2} \cdot 2\,\mathrm{m}}{112} \approx 18.99\,\mathrm{mm}, \tag{5.5}$$

$$y_{max} = \frac{2 \cdot \tan \frac{FOV_y}{2} \cdot d_{max}}{p_y} = \frac{2 \cdot \tan \frac{44^\circ}{2} \cdot 2\,\mathrm{m}}{86} \approx 18.79\,\mathrm{mm}. \tag{5.6}$$

However, these relations do not take into account the lateral noise of the camera. As there is no information on the lateral noise of the flexx2, a lateral noise of one pixel

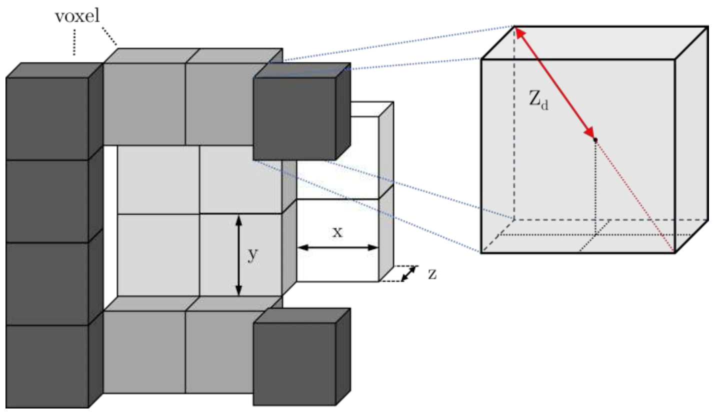


Figure 5.10: 3D grayscale image of the depth data displayed as a voxel grid. Closer points are darker, and far-off points are lighter. Every voxel defines a specific area in space depending on the resolution of the depth camera and the measured distance. The position uncertainty Z_d of a person in this camera system is half a diagonal of a voxel. The lateral noise of the camera is in this figure ignored.

at a distance of 2 m is assumed according to the information in [61]. The contributions C and Z_d are determined as follows:

$$\begin{aligned} C = Z_d =& \sqrt{[(1+0.5)\cdot x_{max}]^2 + [(1+0.5)\cdot y_{max}]^2 + z_{max}^2} \\ =& \sqrt{(1.5\cdot 18.99\,\text{mm}]^2 + (1.5\cdot 18.79\,\text{mm})^2 + 20\,\text{mm}^2} \\ \approx& 44.8\,\text{mm}. \end{aligned} \tag{5.7}$$

However, the contributions C and Z_d depend on the type of implementation. For example, the values for x and y change in the 2×2 minimum pooling operation:

$$\begin{aligned} C = Z_d =& \sqrt{[(1+1)\cdot x_{max}]^2 + [(1+1)\cdot y_{max}]^2 + z_{max}^2} \\ =& \sqrt{(2\cdot 18.99\,\text{mm}]^2 + (2\cdot 18.79\,\text{mm})^2 + 20\,\text{mm}^2} \\ \approx& 57.1\,\text{mm}. \end{aligned} \tag{5.8}$$

For the 4×4 minimum pooling operation, the contributions C and Z_d are as follows:

$$\begin{aligned} C = Z_d =& \sqrt{[(1+2)\cdot x_{max}]^2 + [(1+2)\cdot y_{max}]^2 + z_{max}^2} \\ =& \sqrt{(3\cdot 18.99\,\text{mm}]^2 + (3\cdot 18.79\,\text{mm})^2 + 20\,\text{mm}^2} \\ \approx& 82.6\,\text{mm}. \end{aligned} \tag{5.9}$$

In the method for reducing the depth resolution, a depth step of 1 cm is added to z_{max}:

$$\begin{aligned} C = Z_d = &\sqrt{[(1+0.5)\cdot x_{max}]^2 + [(1+0.5)\cdot y_{max}]^2 + (10\,\text{mm} + z_{max})^2} \\ = &\sqrt{(1.5\cdot 18.99\,\text{mm}]^2 + (1.5\cdot 18.79\,\text{mm})^2 + (10\,\text{mm} + 20\,\text{mm})^2} \\ \approx &50.1\,\text{mm}. \end{aligned} \tag{5.10}$$

5.4.2 Results

With the definition of all contributions necessary for the evaluation of the camera system to determine the safety distance S_p in the collaboration space, this can be specified as follows:

$$S_p = S_h + S_r + C + Z_d = (1.6\,\text{m}\,\text{s}^{-1} + 0.565\,\text{m}\,\text{s}^{-1})\cdot T_r + C + Z_d. \tag{5.11}$$

The results for the safety distance regarding the discussed data reduction methods with different frame rates of the flexx2 can be seen in Table 5.1.

Table 5.1: Safety distances according to standard DIN ISO/TS 15066 of measured detection times.

method	frame rate	S_h	S_r	C	Z_d	S_p
no method	20 fps	144 mm	51 mm	45 mm	45 mm	284 mm
	30 fps	186 mm	66 mm	45 mm	45 mm	342 mm
	45 fps	138 mm	49 mm	45 mm	45 mm	277 mm
	60 fps	130 mm	46 mm	45 mm	45 mm	265 mm
8 bit/pixel	30 fps	107 mm	38 mm	50 mm	50 mm	245 mm
	45 fps	62 mm	22 mm	50 mm	50 mm	184 mm
	60 fps	59 mm	21 mm	50 mm	50 mm	180 mm
pool 2×2	30 fps	99 mm	35 mm	57 mm	57 mm	247 mm
	45 fps	53 mm	19 mm	57 mm	57 mm	186 mm
	60 fps	45 mm	16 mm	57 mm	57 mm	175 mm
pool 4×4	20 fps	84 mm	30 mm	83 mm	83 mm	280 mm
	45 fps	47 mm	17 mm	83 mm	83 mm	229 mm
	60 fps	39 mm	14 mm	83 mm	83 mm	217 mm

The *stop mode* is activated in the application when the safety distance S_p is reached. The lowest safety distance is achieved with the data reduction method minimum pooling 2×2 at a frame rate of 60 fps. The minimum safety distance due to the depth camera is 175 mm. The task is now to determine the distance until the reduced-velocity mode becomes effective. This is determined on the basis of the braking ramps of all axes of the robot at maximum velocity and the maximum velocity of the person. The velocity v_r is multiplied by the maximum braking time. v_s is the velocity of the robot during the stop, from the activation of the stop command until the robot comes to a standstill. As v_s is not monitored, v_s is selected as the maximum velocity of the robot. This results in the calculation of the distance for activating *reduced velocity mode* with the lowest safety distance of Table 5.1 ($S_p = 175\,\mathrm{mm}$):

$$\begin{aligned} d_v =& S_p + (v_h + v_r + v_s) \cdot T_s \\ =& 0.175\,\mathrm{m} + 1.6\,\mathrm{m\,s^{-1}} + 0.565\,\mathrm{m\,s^{-1}} + 0.565\,\mathrm{m\,s^{-1}}) \cdot 0.3\,\mathrm{s} \\ =& 0.994\,\mathrm{m}. \end{aligned} \tag{5.12}$$

If a person falls below the distance to the robot of d_v, the velocities of the axes are linearly reduced until the safety distance S_p is reached, where all axes are stationary. Since the velocity of people and the robot in the collaboration space is not tracked, the *reduced velocity mode* is a method to ensure a continuous transition between procedure and the *stop mode*.

In order to create a comparison of the implemented system with the state of the art, the specifications of the human-robot collaboration system from Veo Robotics [23] are applied to the implemented system. The recognition time is specified as 100 ms and the resolution as 25 mm at a distance of 3 m. This leads to a safety distance of:

$$\begin{aligned} S_p =& S_h + S_r + C + Z_d \\ =& (1.6\,\mathrm{m\,s^{-1}} + 0.565\,\mathrm{m\,s^{-1}}) \cdot 0.1\,\mathrm{s} + 0.025\,\mathrm{m} \cdot \frac{2\,\mathrm{m}}{3\,\mathrm{m}} + 0.025\,\mathrm{m} \cdot \frac{2\,\mathrm{m}}{3\,\mathrm{m}} \\ \approx& 250\,\mathrm{mm}. \end{aligned} \tag{5.13}$$

5.4.3 Discussion

The results in Table 5.1 indicate that a higher frame rate contributes to a decreased feasible safety distance between the robot and human. Additionally, utilizing data reduction methods leads to a reduction in the safety distance. Despite the minimum pooling operation 4×4 achieving the lowest detection time of 24.1 ms, the calculated safety distance is larger compared to other data reduction methods. This discrepancy arises from the lower resolution, causing higher inaccuracies. The most favorable outcome is obtained with the minimum pooling operation 2×2, resulting in a safety distance of 175 mm. Consequently, the safety distance of the implemented system is 75 mm less than of the state-of-the-art system, and thus show the feasibility of the proposed integration approach.

CHAPTER 6

Conclusion and Outlook

This chapter summarizes the system and implementation of the Ethernet for Control Automation Technology (EtherCAT) enabled depth camera and evaluates its performance in a human-robot collaboration application. In addition, the research questions are raised again and discussed. Finally, the chapter concludes with a perspective, referring to possible future research and suggesting improvements to the system.

6.1 Conclusion

The motivation of this thesis is the simplification of components in Industry 4.0 and their standardized integration into the field level. In the course of this project, a depth camera, the flexx2 from pmd technologies [20], is converted for the use with EtherCAT. A Raspberry Pi 4 serves as a Microcontroller (µC) that segments the depth data from the flexx2 connected via USB 3.0 and sends it to the EtherCAT Slave Controller (ESC) via Serial Peripheral Interface (SPI). On the ESC of the EASYCAT HAT board [49], 602 B are defined for the data and 2 B for the segment address. The EtherCAT master, The Windows Control and Automation Technology 3 (TwinCAT3), reassembles the segments sent via EtherCAT frames into a complete depth image. The cycle time is 0.1 ms to ensure that all segments of a depth image are transmitted as fast as possible.

Various modes are tested for the operation of the depth camera, which differ in the number of modulation frequencies used, the range of depth information, and the frame rate. The maximum possible data transmission rate is decisive for the transmission of depth images.

In the course of this diploma thesis, two research questions are addressed. The first question is as follows:

Research Question 1

Is it feasible to embed depth information from a 3D camera via fieldbus within a Programmable Logic Controller (PLC), while maintaining the specifications of the embedded camera system?

Yes, depth images can be transmitted with a resolution of 224×172 pixels, a depth information of 16 bit/pixel and a frame rate of 30 fps. The use of one modulation frequency allows a maximum SPI data transfer rate of approximately 25 Mbit s^{-1}. However, if two modulation frequencies are used to calculate depth, the maximum transmission rate of the SPI is reduced to 17 Mbit s^{-1}. The reason for this is the influence of the computing power utilization of the Raspberry Pi 4 Model B on the SPI. When the depth camera is switched off, data transfer rates of over 28 Mbit s^{-1} are possible via SPI.

In order to establish comparability with current depth camera systems, the flexx2 is integrated into a human-robot collaboration application. Veo Robotics' collaboration space system [23] with a specified detection time of 100 ms for objects smaller than 17 mm at a distance of two meters from the measuring system serves as a benchmark. On this basis, the second research question can be formulated as follows:

Research Question 2

Is it feasible to use the developed camera system in a human-robot collaboration application while complying with the specifications of existing camera systems?

Yes, the EtherCAT-enabled camera system is integrated into a human-robot collaboration application with detection times of less than 28 ms for objects with a detectable size of 57 mm at a distance of two meters. In the implementation, different methods for data reduction are compared to counteract the limited data transfer rate. The safety distances between humans and robots, as defined in the DIN ISO/TS 15066 standard [22] for collaborative robot systems, are up to 75 mm smaller with the implemented EtherCAT-capable depth camera in combination with various data reduction methods than with the reference system from Veo Robotics. The resulting smaller safety distance therefore enables a more efficient and fluid way of working in the collaboration space.

6.2 Outlook

With regard to future research, it is desirable to aim for overall data transmission performance. The focus here is particularly on the maximum data transfer rate, the latency, and the real-time capability of the implemented camera system.

The maximum data transfer rate of the system is limited by the SPI, which in turn depends on the selected mode. The limited computing power of the Raspberry Pi determines the maximum data transfer rate of the SPI. Therefore, a more powerful single-board computer or a Field Programmable Gate Array (FPGA) could ensure faster transmission via SPI and lower latencies. In this case, the selected Process Data Interface (PDI), i.e. the SPI, would be the bottleneck. Serial communication between the µC and the ESC is a limiting factor. Parallel interfaces can improve the data transfer rate and reduce latencies, enabling the transmission of higher-resolution images and higher frame rates.

The real-time capability of systems plays a decisive role in robotics. However, the connected USB camera and the Linux-based operating system of the Raspberry Pi do not enable real-time capability due to unpredictable latencies. Further development of the EtherCAT-capable camera in this area is therefore desirable in industrial automation.

Bibliography

[1] J. Pistorius, *Industrie 4.0 – Schlüsseltechnologien für die Produktion*. Springer Berlin Heidelberg, 2020.

[2] B. Z. Yuan and M. G. Rodd, “Tutorial: Computer vision—towards a three-dimensional world,” *Engineering Applications of Artificial Intelligence*, vol. 2, no. 2, pp. 94–108, Jun. 1989.

[3] V. Alonso, A. Dacal-Nieto, L. Barreto, A. Amaral, and E. Rivero, “Industry 4.0 implications in machine vision metrology: an overview,” *Procedia Manufacturing*, vol. 41, pp. 359–366, Jan. 2019.

[4] W. Yan, Z. Xu, X. Zhou, Q. Su, S. Li, and H. Wu, “Fast Object Pose Estimation Using Adaptive Threshold for Bin-Picking,” *IEEE Access*, vol. 8, pp. 63 055–63 064, 2020.

[5] J. Su, Z. Y. Liu, H. Qiao, and C. Liu, “Pose-estimation and reorientation of pistons for robotic bin-picking,” *Industrial Robot*, vol. 43, no. 1, pp. 22–32, Jan. 2016.

[6] P. Opaspilai, S. Vongbunyong, and A. Dheeravongkit, “Robotic System for Depalletization of Pharmaceutical Products with 3D Camera,” *ICSEC 2021 - 25th International Computer Science and Engineering Conference*, pp. 422–427, 2021.

[7] P. Doliotis, C. D. McMurrough, A. Criswell, M. B. Middleton, and S. T. Rajan, “A 3D perception-based robotic manipulation system for automated truck unloading,” *IEEE International Conference on Automation Science and Engineering*, vol. 2016-November, pp. 262–267, Nov. 2016.

[8] C. Koch, Z. Zhu, S. G. Paal, and I. Brilakis, "Machine vision techniques for condition assessment of civil infrastructure," *Integrated Imaging and Vision Techniques for Industrial Inspection: Advances and Applications*, pp. 351–375, Sep. 2015.

[9] B. Schmidt and L. Wang, "Depth camera based collision avoidance via active robot control," *Journal of Manufacturing Systems*, vol. 33, no. 4, pp. 711–718, Oct. 2014.

[10] Terabee, *Building TOF Sensor Arrays for Collision Avoidance Systems*, 2022. [Online]. Available: `https://www.terabee.com/an-introduction-to-building-time-of-flight-sensor-arrays-for-collision-avoidance-systems-in-mobile-robots/` (visited on 12/18/2023).

[11] L. Li, "Time-of-Flight Camera - An Introduction," 2014. [Online]. Available: `https://api.semanticscholar.org/CorpusID:18376723` (visited on 12/18/2023).

[12] P. Zanuttigh, C. D. Mutto, L. Minto, G. Marin, F. Dominio, and G. M. Cortelazzo, "Time-of-flight and structured light depth cameras: Technology and applications," *Time-of-Flight and Structured Light Depth Cameras: Technology and Applications*, pp. 1–355, Jan. 2016.

[13] C. Bamji *et al.*, "A Review of Indirect Time-of-Flight Technologies," *IEEE Transactions on Electron Devices*, vol. 69, no. 6, pp. 2779–2793, Jun. 2022.

[14] J. P. Thomesse, "Fieldbus technology in industrial automation," *Proceedings of the IEEE*, vol. 93, no. 6, pp. 1073–1101, Jun. 2005.

[15] S. K. Sen, *Fieldbus and Networking in Process Automation*. CRC Press, Apr. 2021.

[16] DIN EN IEC 61158-1, *Industrielle kommunikationsnetze - feldbusse - teil 1: Überblick und leitfaden zu den normen der reihen iec 61158 und iec 61784*, Standard, Apr. 2020.

[17] Beckhoff Automation GmbH, *Industriekameras für die industrielle Bildverarbeitung | Beckhoff Österreich*. [Online]. Available: `https://www.beckhoff.com/de-at/produkte/vision/kameras/` (visited on 12/18/2023).

[18] A. Pochyly, T. Kubela, V. Singule, and P. Cihak, "3D vision systems for industrial bin-picking applications," in *Proceedings of 15th International Conference MECHATRONIKA*, IEEE, 2012, pp. 1–6.

[19] E. Shojaei Barjuei, E. Courteille, D. Rangeard, F. Marie, and A. Perrot, "Real-time vision-based control of industrial manipulators for layer-width setting in concrete 3D printing applications," *Advances in Industrial and Manufacturing Engineering*, vol. 5, p. 100 094, Nov. 2022.

[20] pmdtechnologies ag, *pmd flexx2 3D Camera Development Kit – pmdtechnologies ag*, 2023. [Online]. Available: `https://3d.pmdtec.com/en/3d-cameras/flexx2/` (visited on 12/18/2023).

[21] SICK AG, *safeVisionary2 - Sichere Kamerasysteme.* [Online]. Available: https://www.sick.com/at/de/catalog/produkte/safety/sichere-kamerasysteme/safevisionary2/c/g568562?tab=overview (visited on 12/18/2023).

[22] DIN ISO/TS 15066:2016, *Roboter und robotikgeräte – kollaborierende roboter*, Standard, Dec. 2017.

[23] VEO ROBOTICS, INC, *Veo Robotics*, 2022. [Online]. Available: https://www.veobot.com/ (visited on 12/18/2023).

[24] FRAMOS GmbH, *Advantages and Disadvantages of Time-of-Flight Cameras*, 2023. [Online]. Available: https://www.framos.com/en/articles/advantages-and-disadvantages-of-time-of-flight-cameras (visited on 12/18/2023).

[25] P. Linton, M. J. Morgan, J. C. Read, D. Vishwanath, S. H. Creem-Regehr, and F. Domini, "New Approaches to 3D Vision," *Philosophical transactions of the Royal Society of London. Series B, Biological sciences*, vol. 378, no. 1869, 2023.

[26] Y. He and S. Chen, "Recent Advances in 3D Data Acquisition and Processing by Time-of-Flight Camera," *IEEE Access*, vol. 7, pp. 12 495–12 510, 2019.

[27] M. Stec, V. Herrmann, and B. Stabernack, "Using Time-of-Flight Sensors for People Counting Applications," *Conference on Design and Architectures for Signal and Image Processing, DASIP*, vol. 2019-October, pp. 59–64, Oct. 2019.

[28] H. Seo *et al.*, "Direct TOF Scanning LiDAR Sensor with Two-Step Multievent Histogramming TDC and Embedded Interference Filter," *IEEE Journal of Solid-State Circuits*, vol. 56, no. 4, pp. 1022–1035, Apr. 2021.

[29] A. P. Jongenelen, D. G. Bailey, A. D. Payne, A. A. Dorrington, and D. A. Carnegie, "Analysis of errors in ToF range imaging with dual-frequency modulation," *IEEE Transactions on Instrumentation and Measurement*, vol. 60, no. 5, pp. 1861–1868, May 2011.

[30] Y. He, B. Liang, Y. Zou, J. He, and J. Yang, "Depth Errors Analysis and Correction for Time-of-Flight (ToF) Cameras," *Sensors (Basel, Switzerland)*, vol. 17, no. 1, Jan. 2017. [Online]. Available: /pmc/articles/PMC5298665/%20/pmc/articles/PMC5298665/?report=abstract%20https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5298665/.

[31] H. Plank, G. Holweg, T. Herndl, and N. Druml, "High performance Time-of-Flight and color sensor fusion with image-guided depth super resolution," *Proceedings of the 2016 Design, Automation and Test in Europe Conference and Exhibition, DATE 2016*, pp. 1213–1218, Apr. 2016.

[32] M. Hansard, S. Lee, O. Choi, and R. Horaud, *Time-of-Flight Cameras* (SpringerBriefs in Computer Science). London: Springer London, 2013.

[33] P. O'Sullivan and N. L. Dortz, *Time of flight system design: System overview*, 2021. [Online]. Available: https://www.embedded.com/time-of-flight-system-design-system-overview/ (visited on 12/18/2023).

[34] EtherCAT Technology Group, *EtherCAT Technology Group | EtherCAT*. [Online]. Available: https://www.ethercat.org/de/technology.html (visited on 12/18/2023).

[35] F. Häfele, “Topologievarianten von EtherCAT und deren Einfluss auf die Systemeigenschaften,” *atp edition*, vol. 50, no. 12, pp. 38–42, 2008.

[36] *Slave Implementation Guide Document: ETG.2200*, Version 3.1.0, EtherCAT Technology Group, Nov. 2018.

[37] acontis technologies, *Introduction to EtherCAT Technology and the EtherCAT Protocol*, 2021. [Online]. Available: https://www.acontis.com/en/what-is-ethercat-communication-protocol.html (visited on 12/18/2023).

[38] *Grundlagen TwinCAT 3*, Version 1.0.2, Beckhoff Automation GmbH, Nov. 2023.

[39] IEEE 802.3, “IEEE Standard for Ethernet,” *IEEE Std 802.3-2022 (Revision of IEEE Std 802.3-2018)*, pp. 1–7025, 2022.

[40] I. I. B. Jamaludin and H. B. Hassan, “Design and Analysis of Serial Peripheral Interface for Automotive Controller,” *2020 IEEE Student Conference on Research and Development, SCOReD 2020*, pp. 498–501, Sep. 2020.

[41] *SPI Interface*, Analog Devices, Miguel Usach, 2015. [Online]. Available: https://www.analog.com/media/en/technical-documentation/application-notes/an-1248.pdf (visited on 12/18/2023).

[42] DIN EN ISO 10218-1, *Robotik - Sicherheitsanforderungen - Teil 1: Industrieroboter (ISO/DIS 10218-1.2:2021)*, Standard, Sep. 2021.

[43] DIN EN ISO 10218-2, *Robotik - Sicherheitsanforderungen für Robotersysteme in industrieller Umgebung - Teil 2: Robotersysteme, Roboteranwendungen und Integration von Roboterzellen (ISO/DIS 10218-2:2020)*, Standard, Mar. 2021.

[44] Beckhoff Automation GmbH, *TwinCAT 3 | Automatisierungssoftware | Beckhoff Österreich*. [Online]. Available: https://www.beckhoff.com/de-at/produkte/automation/twincat/ (visited on 12/18/2023).

[45] DIN EN IEC 61131-3, “Speicherprogrammierbare Steuerungen - Teil 3: Programmiersprachen (IEC 65B/1229/CDV:2023),” Standard, Sep. 2023.

[46] *Manual TF7000 - TF7300 TwinCAT 3 Vision*, Version 1.5.0, Beckhoff Automation GmbH, Dec. 2022.

[47] *Handbuch TwinCAT3 Produktübersicht*, Version 1.13.1, Beckhoff Automation GmbH, Nov. 2023.

[48] Raspberry Pi Foundation, *Raspberry Pi 4 Model B specifications – Raspberry Pi*, 2021. [Online]. Available: https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/ (visited on 12/18/2023).

[49] AB&T Srl, *EtherCAT® & Raspberry*. [Online]. Available: `https://www.bausano.net/it/hardware/ethercat-raspberry.html` (visited on 12/18/2023).

[50] *LAN9252 - 2/3-Port EtherCAT® Slave Controller with Integrated Ethernet PHYs*, Document Number DS00001909A, Microchip Technology Inc., Jan. 2015.

[51] *EtherCAT Modular Device Profile Part 1: General MDP Device Model ETG.5001.1 S (D)*, Version V0.10.0, EtherCAT Technology Group, Oct. 2021.

[52] M. McCauley, *bcm2835: C library for Broadcom BCM 2835 as used in Raspberry Pi*, 2021. [Online]. Available: `https://www.airspayce.com/mikem/bcm2835/` (visited on 12/18/2023).

[53] *Royale API Documentation*, Version 5.1.0.1781, pmdtechnologies ag, Mar. 2023.

[54] igus GmbH, *robolink® RL-DP-5 | 5 Freiheitsgrade (DOF) | 790mm Reichweite*. [Online]. Available: `https://www.igus.at/product/20239?artNr=RL-DP-5` (visited on 12/18/2023).

[55] A. Toquica, L. O. Martinez, R. Rodriguez, A. C. Chavarro, and T. Cardozo, "Kinematic modelling of a robotic arm manipulator using matlab," *Journal of engineering and applied sciences*, vol. 12, no. 7, 2017.

[56] M. Ceccarelli, *Fundamentals of Mechanics of Robotic Manipulation* (Mechanisms and Machine Science). Cham: Springer International Publishing, 2022, vol. 112.

[57] R. Di Gregorio, *Kinematics and Robot Design I, KaRD2018*. MDPI - Multidisciplinary Digital Publishing Institute, Sep. 2021, p. 229.

[58] *Documentation EK110x-00xx, EK15xx EtherCAT Bus Coupler*, Version 4.4, Beckhoff Automation GmbH, Sep. 2023.

[59] *Documentation EL20xx, EL2124 Digital Output Terminals*, Version 5.6, Beckhoff Automation GmbH, Mar. 2023.

[60] *EtherCAT Modular Device Profile Part 1: General MDP Device Model ETG.5001.1 S (D)*, Document Number 81009, Vishay Semiconductor GmbH, May 1999.

[61] G. Halmetschlager-Funek, M. Suchi, M. Kampel, and M. Vincze, "An empirical evaluation of ten depth cameras: Bias, precision, lateral noise, different lighting conditions and materials, and multiple sensor setups in indoor environments," *IEEE Robotics and Automation Magazine*, vol. 26, no. 1, pp. 67–77, Mar. 2019.

Eigenständigkeitserklärung

Hiermit erkläre ich, dass die vorliegende Arbeit gemäß dem Code of Conduct, insbesondere ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel, angefertigt wurde. Die aus anderen Quellen direkt oder indirekt übernommenen Daten und Konzepte sind unter Angabe der Quelle gekennzeichnet. Die Arbeit wurde bisher weder im In– noch im Ausland in gleicher oder in ähnlicher Form in anderen Prüfungsverfahren vorgelegt.

Datum	Tobias Steinegger