1 Fundamentals

1.1 Concept and Introduction

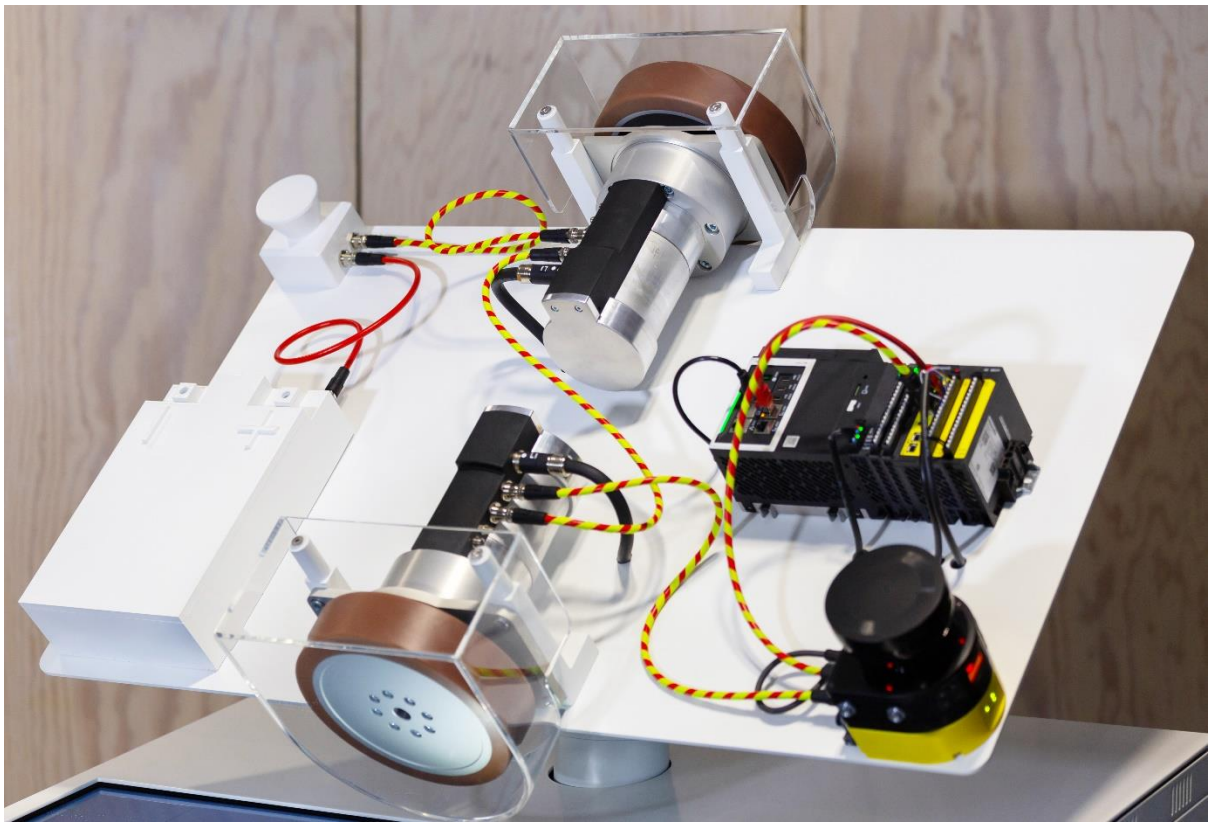


Figure 1 Components Display and Hardware Overview

This project consists of the development of an autonomous mobile robot that integrates the latest hardware offerings from Bosch Rexroth, demonstrating their potential in both performance and streamlined communication. The robot is composed of essential components: a laser scanner, two motors creating a differential drive, and (possibly in the future) a camera. These components come together to create an autonomous mobile robot. The main inspiration of the project is the showcase of the capabilities of its technology while emphasizing the simplicity and efficiency of communication lines within the robotic system. This robot can be used to further test and optimize other Rexroth products and services, such as the ROKIT Locator and the ROKIT Navigator. Figure 1 shows the first version of this robot, in which the hardware, its connections, and the fundamental functionality can be seen.

1.2 Components

1.2.1 ctrlX CORE

Figure 2 ctrlX CORE

The Bosch Rexroth ctrlX CORE serves as the central processing unit and nerve center of the mobile robot. With its powerful computing capabilities and versatile architecture, the ctrlX CORE controls the robot's operations, processing sensor data, executing algorithms, and coordinating movement.

1.2.2 ctrlX SAFEX

Figure 3 ctrlX SAFEX-C.12

The ctrlX SAFEX is responsible for building the safety communication on top of the existing data bus lines, serving as the safety controller for the robot. This device manages the safety functions of the motors and laser scanner. With its advanced safety functionalities and integrated architecture, the ctrlX SAFEX monitors and ensures compliance with safety protocols, safeguarding both the robot and its surroundings. The ctrlX SAFEX builds an FSoE (Fail Safe over EtherCAT) signal on top of the existing EtherCAT communication, which enables the safety capabilities of the other components connected to the robotic system.

1.2.3 SICK microScan3

Figure 4 SICK microScan3

The SICK microScan3 is a safety laser scanner renowned for its exceptional reliability and performance in industrial environments. The scanner was chosen to ensure the safety side of the mobile robot, the microScan3 integrates seamlessly into our robotic systems, employing the power of FSoE (Fail Safe over EtherCAT) communication protocol. This integration simplifies the setup process, aligning perfectly with other components of the robot that also utilize FSoE.

1.2.4 ROKIT with Elmo Drive

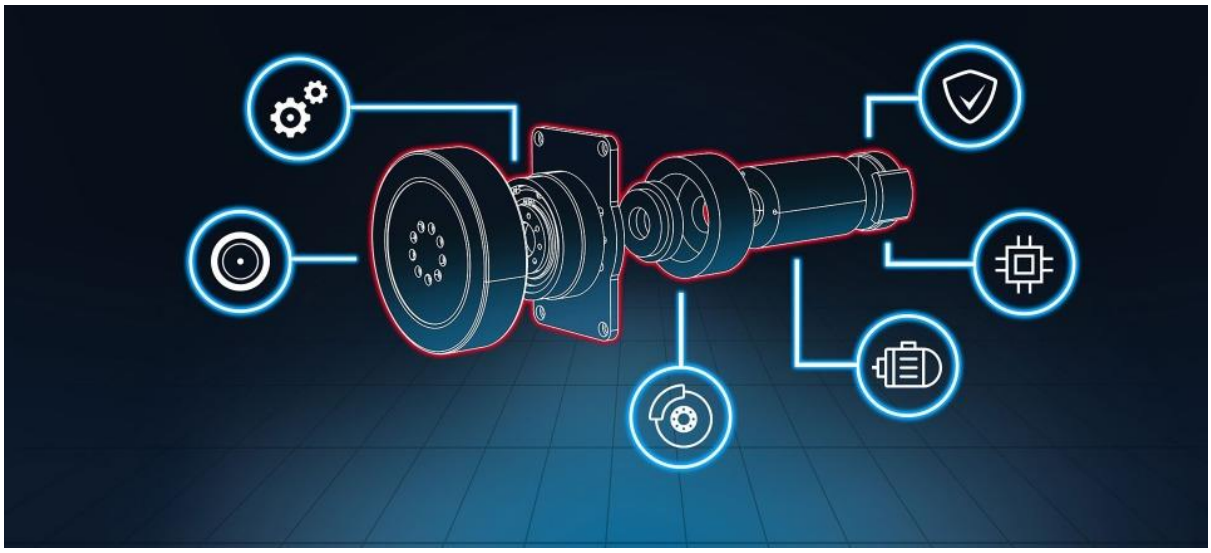


Figure 5 ROKIT Explosion Diagram

The robot uses a combination of two ROKIT motors as its driving mechanism. They are assembled and controlled independently from one another, thus creating a differential drive system. The ROKIT consists of multiple integrated components as seen in Figure 5, namely the wheel, planetary gearbox, safety parking brake, motor, safety encoder, and motion controller.

1.3 Hardware Topology

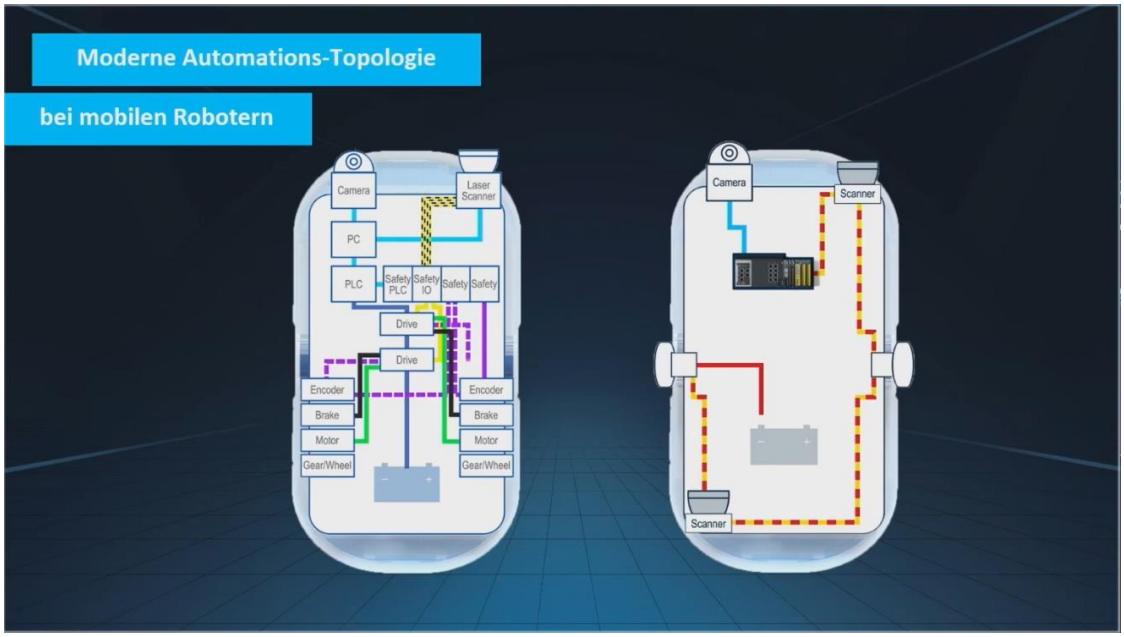


Figure 6 Hardware Topology Overview

One of the main inspirations for the project was the optimization of communication lines. With a single data cable, it is possible to control all the devices required for the safe control of the robot. Figure 6 shows the hardware topology of two different mobile robots. The left robot shows the typical devices and connections needed for such a robot. By implementing protocols on top of the existing data lines, the topology can be simplified. Such an example can be seen on the right side of Figure 6.

1.3.1 Communication Protocol – EtherCAT

EtherCAT (Ethernet for Control Automation Technology) is a high-performance industrial communication protocol renowned for its real-time capabilities and minimal communication latency, making it ideal for time-critical control systems, such as the one needed for the robot. The flexibility of this protocol, allows the communication of the robot to be simplified. A single cable carries the information for all devices. The devices can be connected in a daisy-chain arrangement

1.3.2 SAFETY and FSOE

Ensuring the safety of mobile robots, especially in the presence of humans, is paramount due to the potential risks of collisions, entrapment, or other hazardous situations. Integrating robust safety measures is crucial to mitigate these risks and ensure safe operation in shared spaces. Functional Safety over EtherCAT (FSoE) provides a standardized method to implement safety functions within the EtherCAT network, enabling real-time monitoring and control of safety-related processes. By integrating FSoE, mobile robots can achieve a higher level of safety integrity, enabling features such as safe motion control, collision avoidance, and emergency stop functionalities. This combination of EtherCAT's high-performance communication capabilities with FSoE's safety functionalities ensures rapid response to safety hazards, minimizing the risk of accidents and enhancing overall safety for operators and bystanders alike. No further data line is required in the robot since the FSoE communication travels within the EtherCAT network.

1.4 Required Software for Development

1.4.1 ctrlX WORKS

CtrlX WORKS is the gateway to the communication with the ctrlX Core. Once the ctrlX Core has been connected through Ethernet to the host PC, it will appear on the available devices. Once the connection has been established, the web-based tool for the control of the ctrlX Core can be opened by entering the IP address of the device in the preferred browser.

The software can be downloaded from the following link:

https://developer.community.boschrexroth.com/t5/Store-and-How-to/ctrlX-WORKS/ba-p/16448

1.4.2 ctrlX I/O

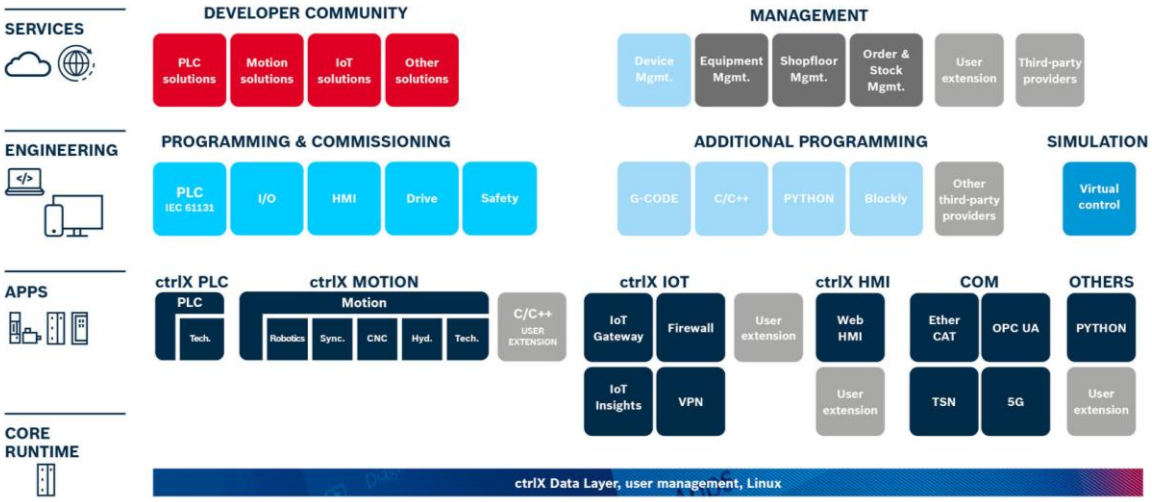


Figure 7 ctrlrX Apps and Services

CtrlX IO Software is found on one layer deeper than the web-based interface. Nevertheless, through this software, it is possible to connect directly to the ctrlX Core, as well as its peripherals and web-based interface. It is possible to synchronize the changes made in ctrlX I/O with everything else. In this software, the different communication protocols between the multitude of devices are defined and configured. This is used to configure the EtherCAT and FSoE communication.

This software can be downloaded as an App on the ctrlX Core web interface. In case, it is not connected to the internet, the app can be downloaded from the Bosch Rexroth website.

So that the ctrlX IO can recognize the connected devices. Additional XML files are needed:

- BoschRexroth_ctrlX_SAFETY_SAFEX_C12_C15_ModulesECAT.xml
- ElmoPlatinum_10_V120027_S.xml
- SICK_microScan3.xml

1.4.3 ctrlX PLC

The ctrlX PLC software is found on the same engineering layer as the ctrlX IO. The PLC is where most of the logic is found. Similar to the ctrlX IO, the PLC communicates to the ctrlX Core and the connected devices. The logic found in the program describes what the robot does with the incoming information. Further applications can be used to aid the control of the robot.

The software is available to download directly from the ctrlX core interface or under the following link:

https://developer.community.boschrexroth.com/t5/Store-and-How-to/ctrlX-AUTOMATION-PLC/ba-p/13298

1.4.4 ctrlX SAFETY Engineering

The SAFETY Engineering software is necessary for configuring the diverse ports of the ctrlX SAFETY. Furthermore, the software is needed to define the devices, which will be connected to the SAFETY device via FSoE, their slaver number, and more.

Similar to the other ctrlX Software, this one can be downloaded from the ctrlX collaboration rooms:

https://www.boschrexroth.com/de/de/myrexroth/collaboration/collaboration-rooms/?path=%2FCtrlx-Automation%2FctrlX_SAFETY%2FSAFETY_Engineering_APPS&search=&page=1&filter=

The version used for this development is version 1.7.6.9778. This version has not been released at the time of writing this protocol.

1.4.5 Elmo Application Studio II

This software is used to configure the needed parameters of the ROKIT motor. Once the parameters are configured via USB, the communication with the motor will be executed with EtherCAT.

The software can be downloaded with the following link:

https://www.elmomc.com/product/easii/

The version used for this development is version 2.9.0.13. This version has not been released at the time of writing this protocol.

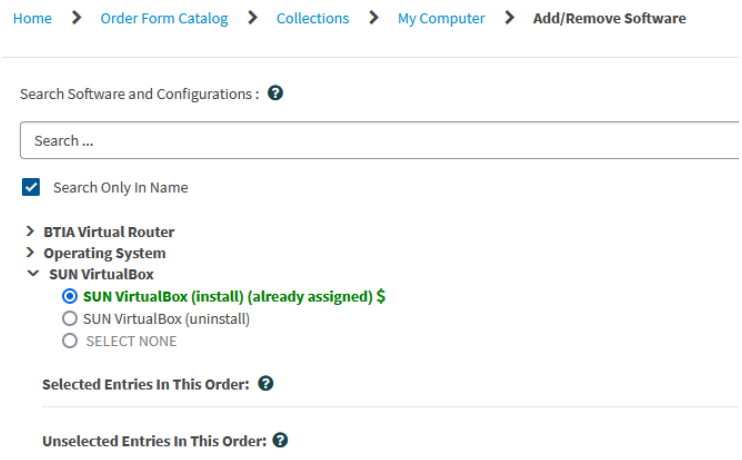
1.4.6 SICK Safety Designer

To have proper control over the SICK microScan3 the initial settings need to be configured over USB. The Safety Designer software is needed for this. It can be downloaded from the following link:

https://www.sick.com/de/en/catalog/digital-services-and-solutions/sick/safety-designer/p/p444968

1.4.7 Virtual Machine

The ethernet communication between the host PC and crlX Core is a closed one. If a parallel connection to multiple networks through two adapters is not possible, e.g. simultaneous ethernet and WIFI, access to the Internet will not be possible while connected to the ctrlX Core. Such is the case for most Bosch Laptops. For this reason, it is recommended to have a virtual machine with Windows. The Virtual Machine can be downloaded through the IT service of Bosch or with the following link:



https://www.virtualbox.org/wiki/Downloads

The Windows ISO file can be downloaded from this website:

https://www.microsoft.com/de-de/software-download/windows10

2 ctrlX Core

2.1 Initial Configuration

2.1.1 Software and Firmware Information

When development started, the hardware was running ctrlX OS 1, which is the operating system version out of the box. At its base, it's running Ubuntu Core20.

The firmware was updated to ctrlX OS 2.2 to get the newest version and avoid bugs. This version is running Ubuntu Core22.

2.1.2 Credentials

Username: boschrexroth

Password: boschrexroth

The password, user/admin rights, and other credentials can be changed under Settings >>Users & Permissions on the web interface of the hardware.

2.1.3 License

Some software will require additional licenses to work properly. One such software is the ctrlX PLC software. To upload and validate the license files, go to Settings > Licenses and upload the files. Temporary rights can also be enabled with the gear icon. However, this will disable the hardware after the time limit has been reached.

2.1.4 App Installation

To install an app, open the web interface, first go to Settings > Apps. If the ctrlX CORE is connected to the internet, apps can be downloaded without having the actual app file. Otherwise, the file containing the app needs to be uploaded to the hardware on the same user interface with the “Install from file” option in Figure 8. App file extensions include, but are not limited to, “.app”, “.snap”. The status of the ctrlX CORE needs to be changed to “Service” to be able to install apps.

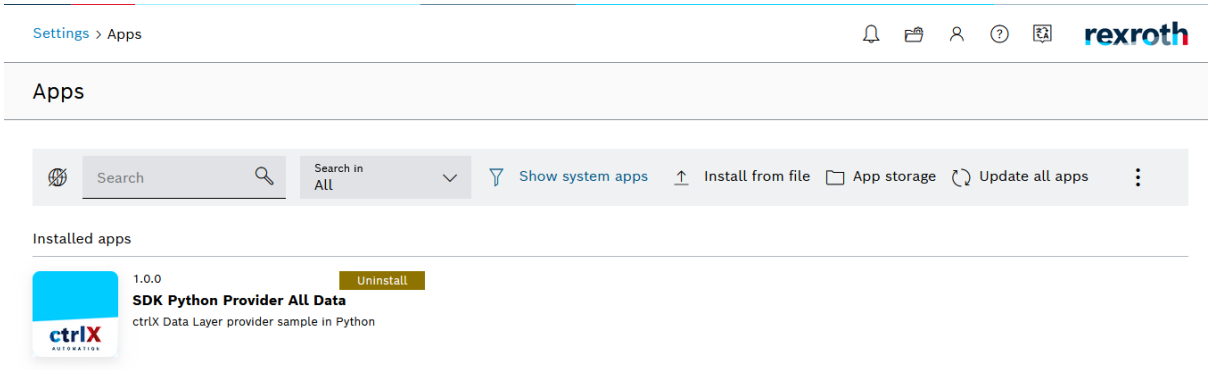


Figure 8 App Installation ctrlX CORE

Communication

2.1.5 Static IP-Address

By setting a static IP address for the communication with the ctrlX Core, the connection seems to be more stable. Without the static IP address, the communication to the ctrlX apps (like ctrlX IO and

ctrlX PLC) commonly leads to problems. For this reason, a static IP address is recommended. Figure 9 shows how to do this. Go to Control Panel → Network and Internet → Network Connections click on the properties of your Ethernet adapter. From this window open the properties of the TCP/IPv4 connection. The static address has to be adapted to the one used for the ctrlX Core. In this case, the IP address 192.168.1.10 was used.

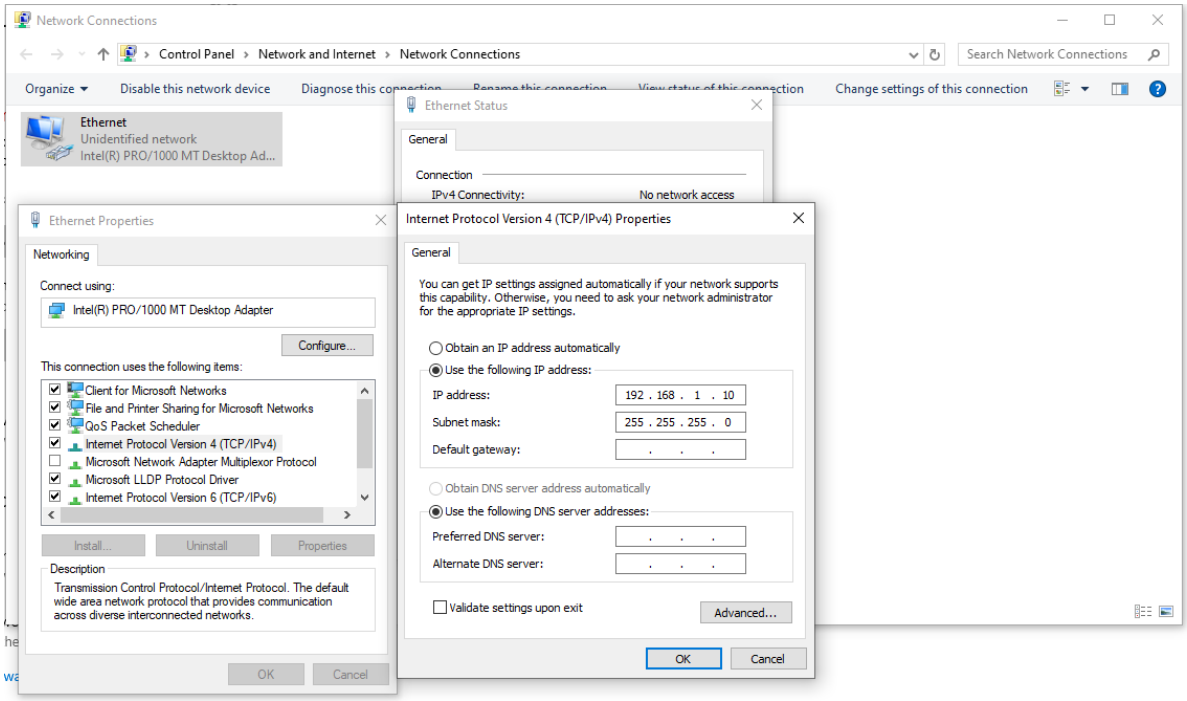


Figure 9 Static IP-Address

2.1.6 Static IP-Address Virtual Machine

As explained in Subchapter 1.4.7 a Windows virtual machine is recommended to be able to stay connected to the internet on the host machine, while still being able to communicate to the ctrlX Core over Ethernet. The static IP address is once again recommended for a more stable connection. The same approach should be used as in the last chapter on the virtual machine.

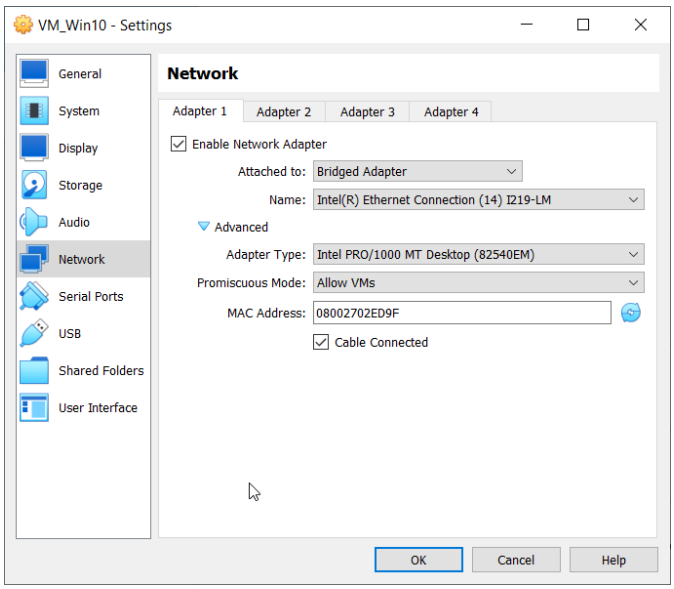


Figure 10 Network Settings Virtual Machine

To be able to connect to an Ethernet device, set the Network settings of the virtual machine to “Bridged Adapter” as seen in Figure 10.

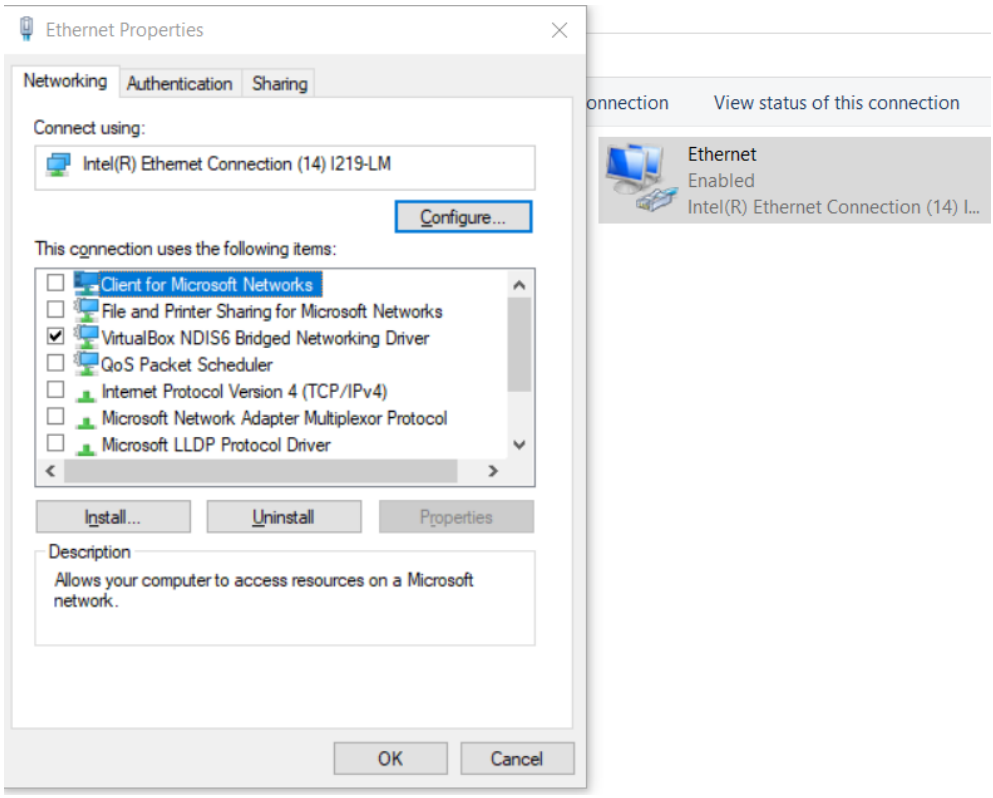


Figure 11 Host Machine Ethernet Settings

Since the Ethernet adapter has priority over the WIFI adapter, this will be used for the Internet connection. For that reason, it is recommended to disable all properties of the Ethernet adapter of the host machine, except for the ones related to the Virtual Machine as seen in Figure 11.

2.2 ROKIT Locator Installation

Please follow this guide to install the ROKIT Locator app into the ctrlX Core:

https://developer.community.boschrexroth.com/t5/Store-and-How-to/How-to-run-the-Rexroth-ROKIT-Locator-on-ctrlX-CORE-X7/ba-p/85296

3 Hardware Configuration

3.1 Connections

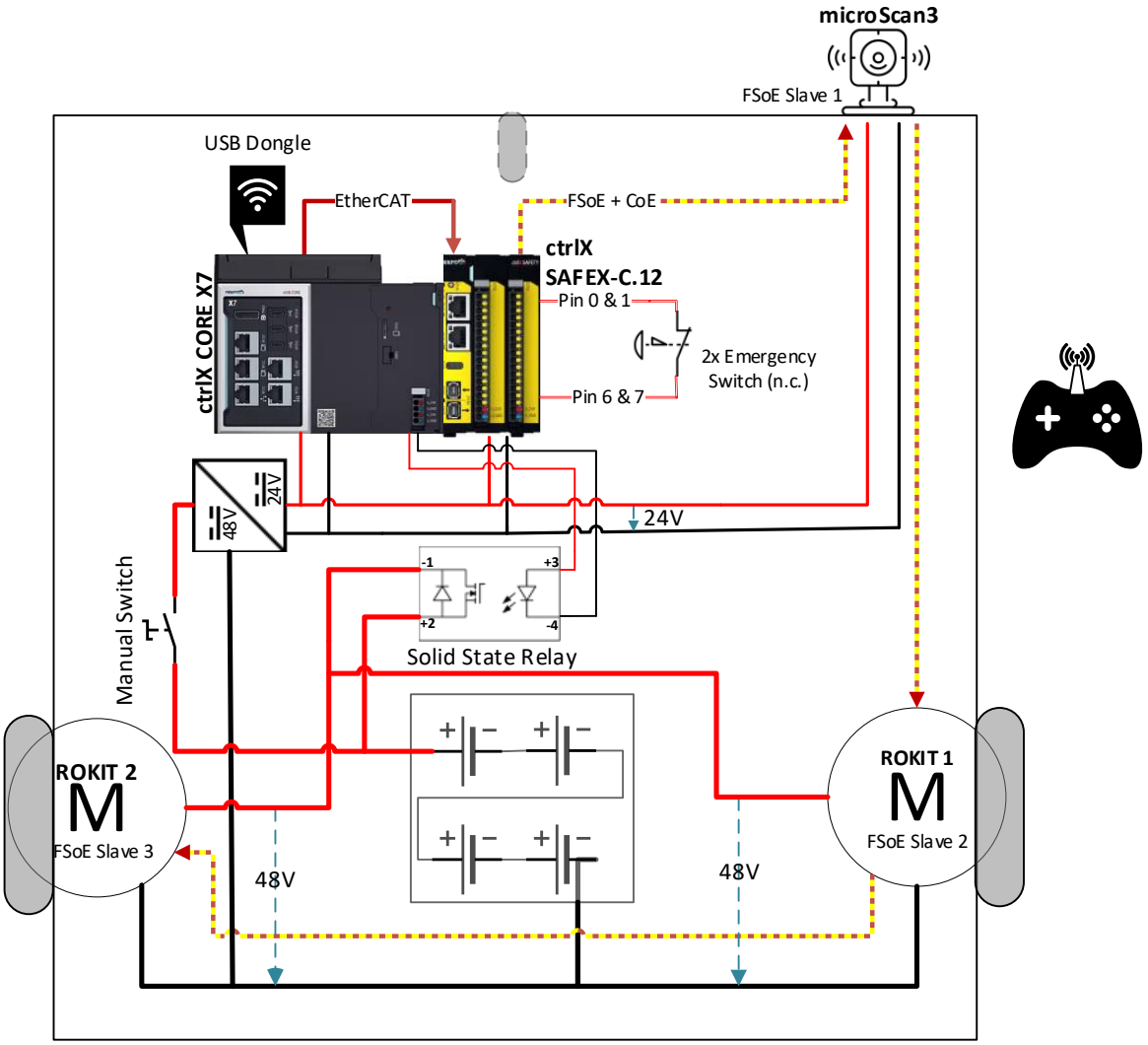


Figure 12 Hardware Connections

Electrical:
The autonomous robot is fully powered by four 12V Lead-acid batteries connected in series, amounting to a total nominal voltage of 48V. To enable the current flow, the manual switch has to be turned to the on position. This voltage is stepped down to 24V by a DC-DC converter. The 24V powers on the crtrlX CORE X7, ctrlX SAFEX-C.12, and the SICK microScan3. The ctrlX CORE can enable the bottom Pins which close the circuit connected to the solid state relay. This circuit carries a 48V Votlage, which powers the two ROKIT motors. Lastly, there is an emergency switch connected between Pins 6&7(XG32) and Pins 0&1(XG31) of the SAFEX-C.12. If opened, all safety features send a stop signal.

Communication:
The main communication to the devices is through their respective USB communication (ctrlX CORE is through Ethernet-XF10). The ctrlX Core (XF50) is connected to the SAFEX (IN-XF28) module via EtherCAT. The SAFEX builds an FSoE (Fail Safe over EtherCAT) and a CoE (CAN over EtherCAT) communication over the established EtherCAT bus. The communication is chained to the microScan3

and the two ROKIT motors through their respective EtherCAT IN and EtherCAT OUT ports. Finally, a remote controller is connected to the ctrlX CORE with a USB dongle.

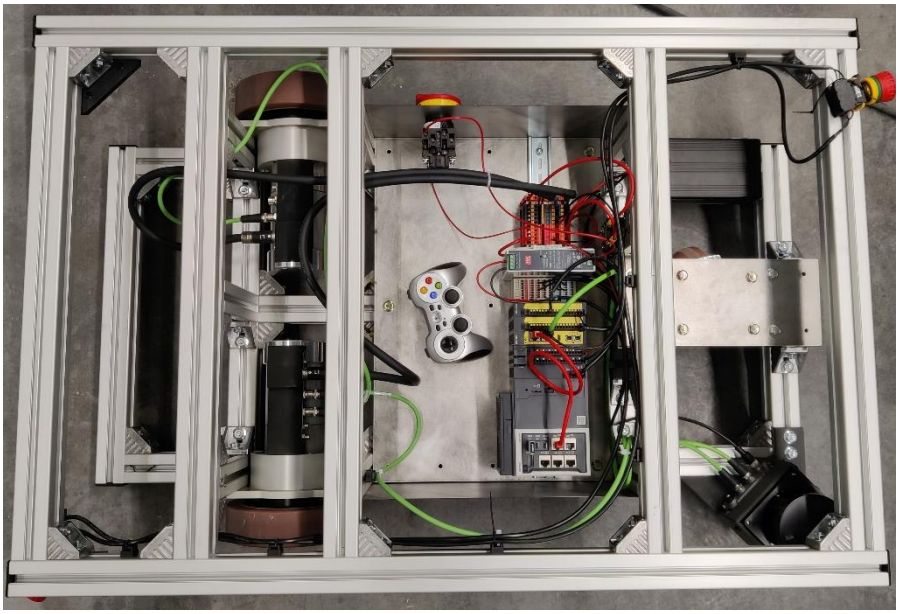


Figure 13 Final Assembly of Mobile Robot

Figure 13 shows how the hardware seen in Figure 12 looks like when finally assembled. The green cable follows the communication line with EtherCAT carrying both CoE (CAN over EtherCAT) and FSoE (Fail Safe over EtherCAT) protocols.

3.2 CtrlX IO Configuration

For the next step, make sure you have the necessary XML files. These are needed, so that the ctrlX Core recognizes the devices connected to the EtherCAT network. To install the XML file to the ctrlX IO software, go to “Tools >> Device Repository”. This will open the window seen in Figure 15. Press the “Install…” button, select the XML file and the file should install automatically. Repeat this step for all the required devices on the EtherCAT network.

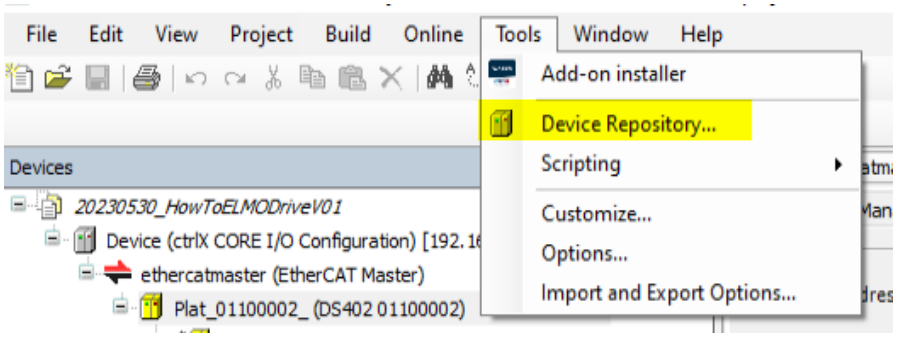


Figure 14 Open Device Repository

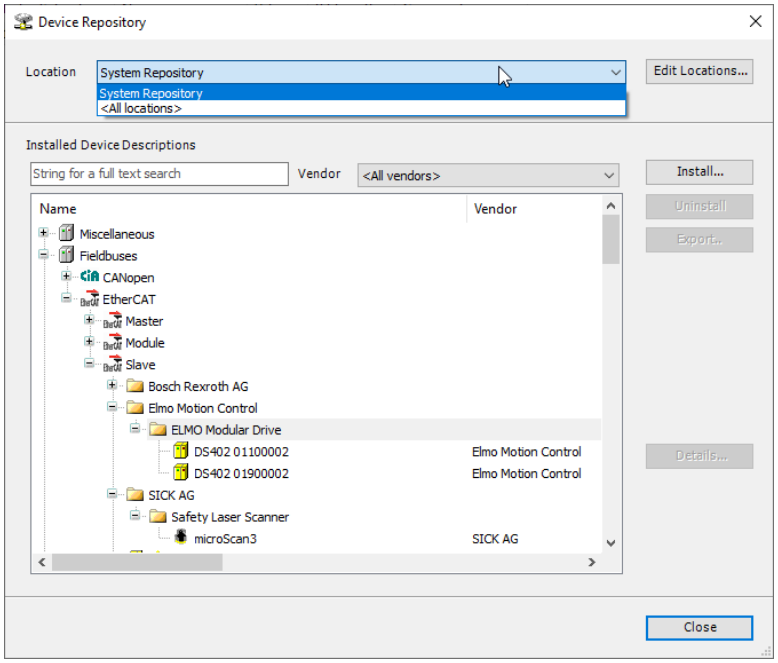


Figure 15 Device Repository -- Installing XML file

Once the XML files have been installed, the configuration can proceed. Right-click on the ethercatmaster module. Once more settings appear, click on the “Scan for Devices…” option. This will pop up the window seen in Figure 16. The EtherCAT network will be scanned and the available devices will be seen on the right side. Make sure, you are connected to the ctrlX Core and the devices are connected like Figure 12. Insert all newly scanned devices.

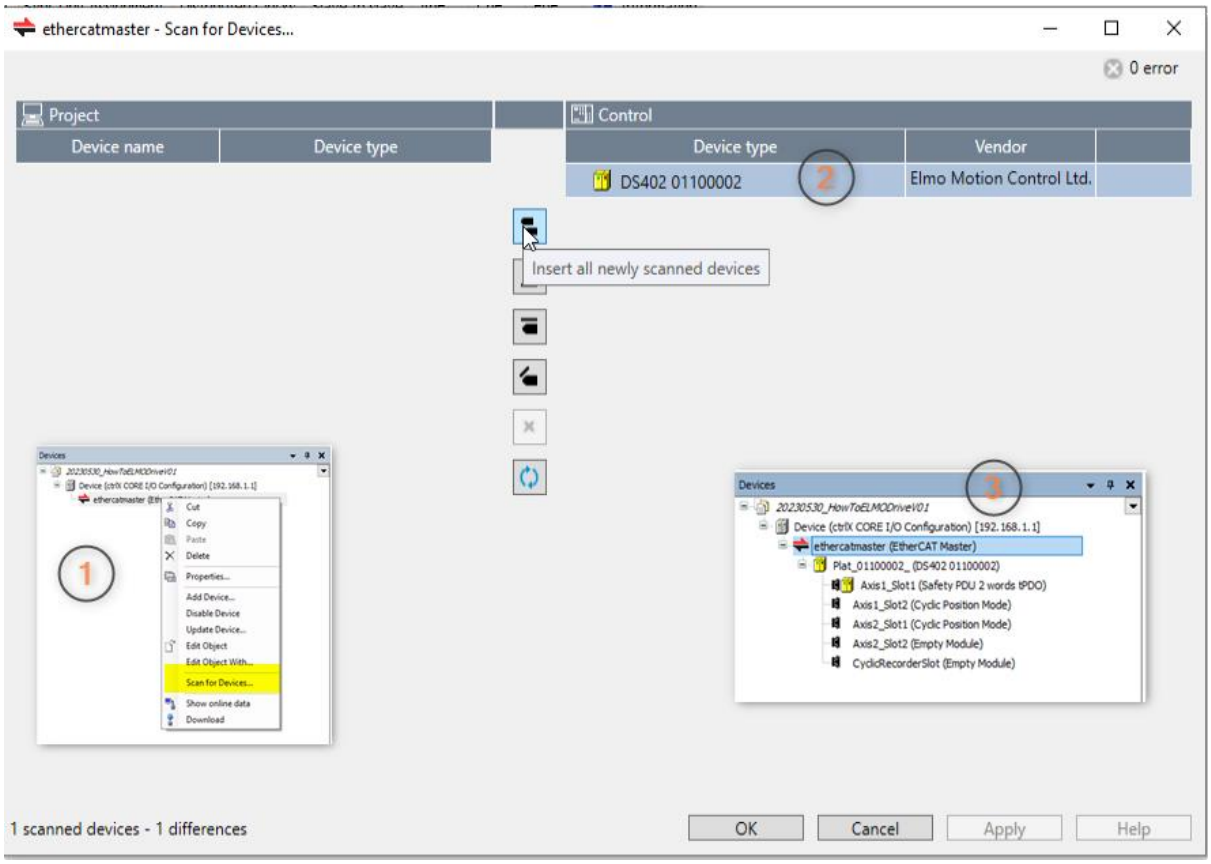


Figure 16 EtherCAT Scan for Devices...

The devices will now show on the left, under the “Devices” tab (the name can be modified, after being scanned). One more step is important for the ROKIT motors and SAFEX. Check the highlighted setting in Figure 17. They should be in “Cyclic Velocity Mode”, with the value set to 9. If this is not the case, right-click on the “Axis1_Slot2” of each motor, and click on “Plug Device”. The plug device dialog should open. Select the desired mode and click “Plug Device”.

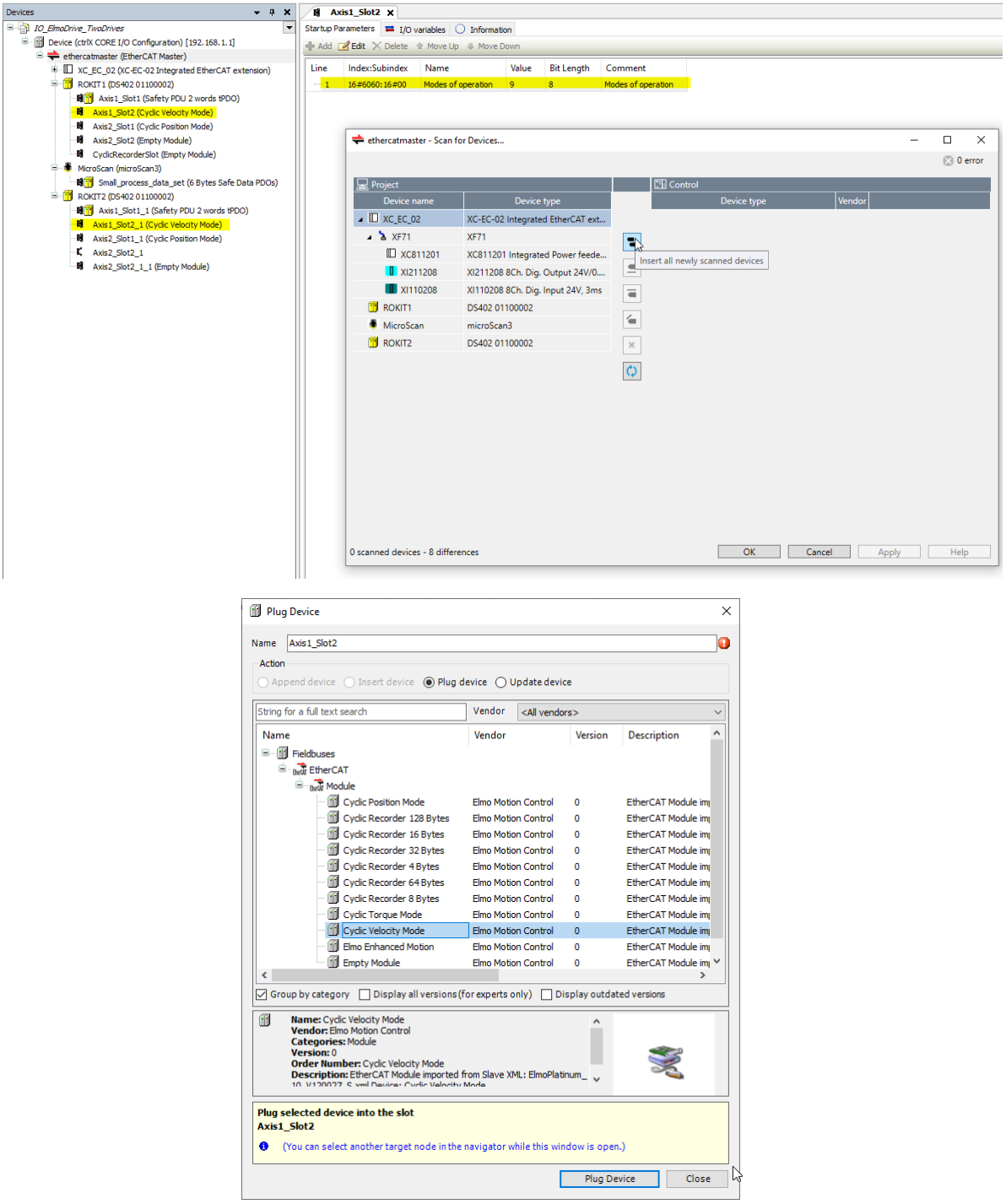


Figure 17 Plug Device CtrlX IO

To update the settings on the CtrlX Core, press the download button on the CtrlX IO ribbon.

The same step to set the velocity mode on the ROKIT motors is necessary for the ctrlX SAFEX. This time, the FSoE connections need to be defined. Since three devices will use this protocol, two ROKIT motors and the laser scanner, three connections are necessary. They can be configured in the same manner as the velocity mode of the ROKIT motors. The FSoE connections need to correspond to the data type and length of the device connected in that order. For this robot, the first device is the laser scanner, which sends/receives 6 Bytes as a safety message. The first FSoE connection should be defined with the same values and length. See Figure 18 reference.

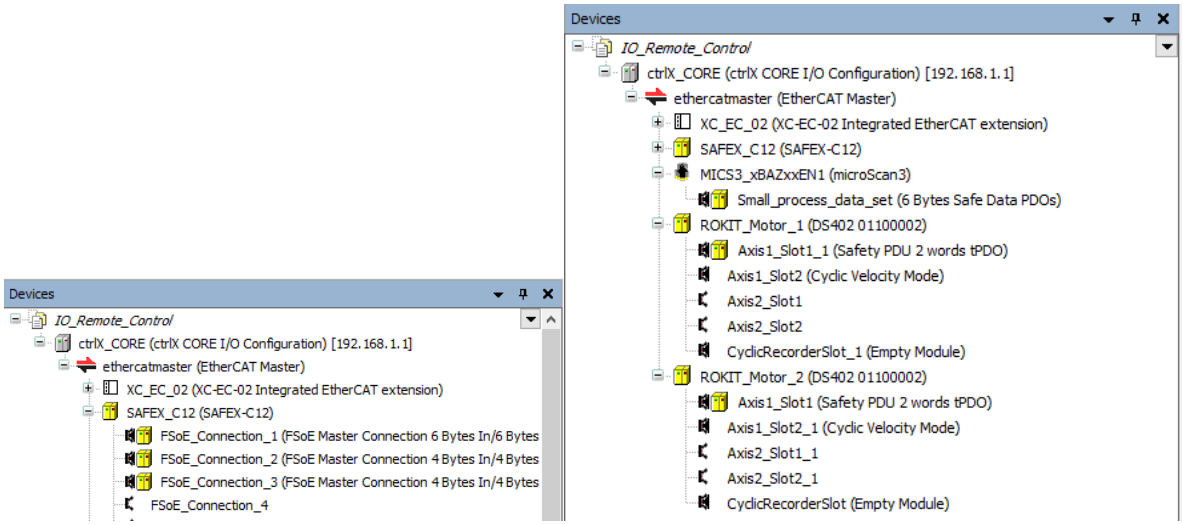


Figure 18 ctrlX IO Device Configuration

3.2.1 Slave to Slave

One last step is necessary to establish a successful FSoE connection. The inputs and outputs of the defined FSoE connections need to be manually allocated to each other, meaning the outputs of one device will be read as inputs on the other side. This configuration can be found under ethercatmaster >>Slave to Slave. Define the configuration as the one shown in Figure 19. A bigger version can be found in Appendix 6.1. Once this step is completed, download the configuration to the ctrlX CORE once again. The ctrlX SAFEX might show an “Invalid input configuration” error. Follow the steps described in Chapter 3.6 to transfer the configuration to the SAFEX.

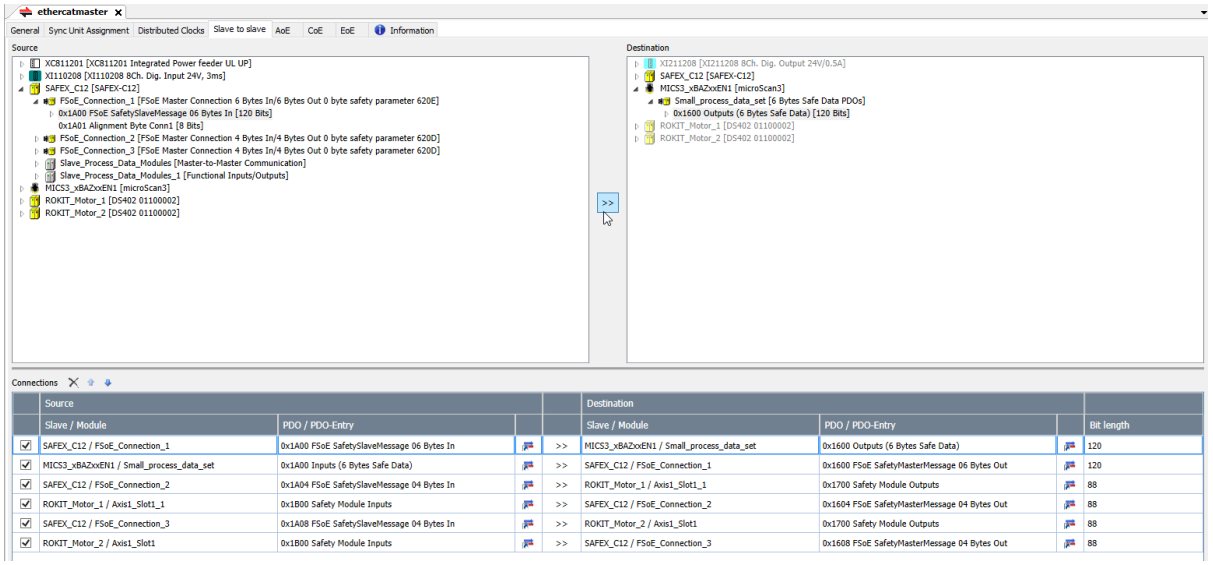


Figure 19 CtrlX IO Slave to Slave Configuration

3.3 ROKIT Setup

Power on the motor and connect it to the PC over USB. Launch *Elmo Application Studio II.* Go to the System Configuration page, select the Serial Port USB of the motor, and connect to it.

3.4 Elmo Drive FSoE

The safety configuration should take place first since it overrides all the settings of the drive.

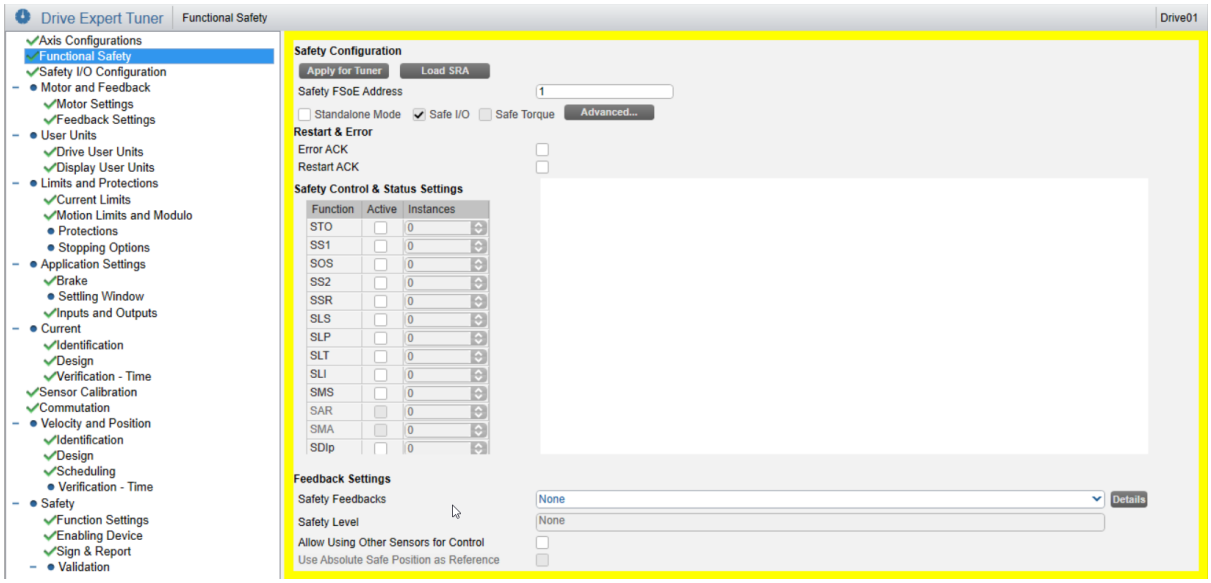


Figure 20 Elmo Functional Safety

Under “Drive Setup and Motion” go to “Drive Expert Tuner” and open the functional safety tab Figure 20. The safety configuration can be loaded from a file or manually set up. To load the configuration, check the ribbon at the top for the “Parameter” tab and select “Download textual file for drive”. Select the desired file, normally with a file ending “.pprm” and load it. The slave FSoE address might have to be changed, depending on the application. The drive needs to not be in “Operation” mode while downloading the configuration (this can be changen on the ethercatmaster window of the ctrlX Core webinterface). Ensure that the STO setting is active with 1 instance after loading the file (Figure 20 does not show this).

To save the safety configuration, go to the “Sign & Report” window, and on the top of the window, where the ribbon is, press the “Apply SRA” setting. Activate the Remote Control before confirming. The drive will now reboot and reconnect. Copy the shown CRC number. This will be needed for Subchapter 3.6.

Once the safety configuration has been saved and the drive reconnected, go to the Motion – Single Axis tab on the Drive Setup and Motion page. If the connection was successful, you should see some values changing in this tab, as well as a red/green indicator, representing the status of the motor. Change the drive mode to “Velocity [UM=2]”.Go to the terminal, and type the following: “AF=1”. One line after, type “SV” to save the setting into the motor (see Figure 21). This will enable to control of the motor through an external signal, e.g. EtherCAT and FSoE.

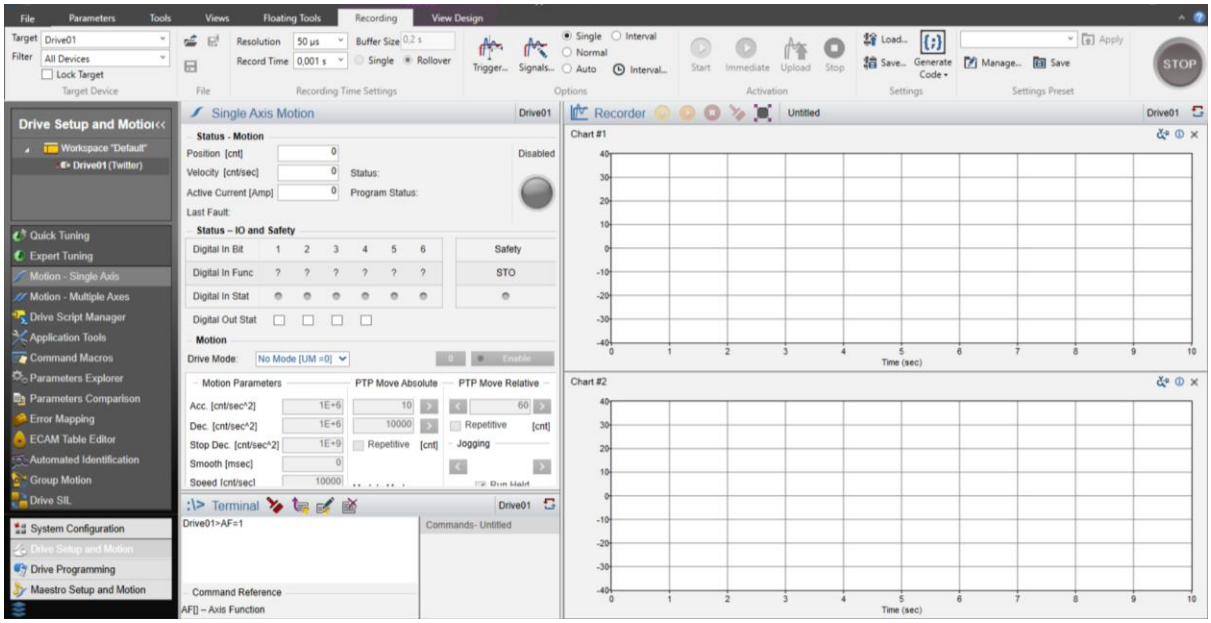


Figure 21 Elmo Configuration Setup

Repeat the steps mentioned above for the second motor. Keep in mind the FSoE slave address and the CRC number.

3.5 SICK MicroScan 3

The SICK microScan3 also needs to be configured to enable the EtherCAT and FSoE communication. First, connect the laser scanner to the computer through USB. Open the Sick *Safety Designer* program. Go to the "Search for Devices" option. The connected device should pop up. Open the settings for the devices. The following video shows a good overview of how to configure the laser scanner (keep in mind the video shows the Ethernet configuration and the one needed is EtherCAT):

https://www.youtube.com/watch?v=ywQ8oVl2cCg&t

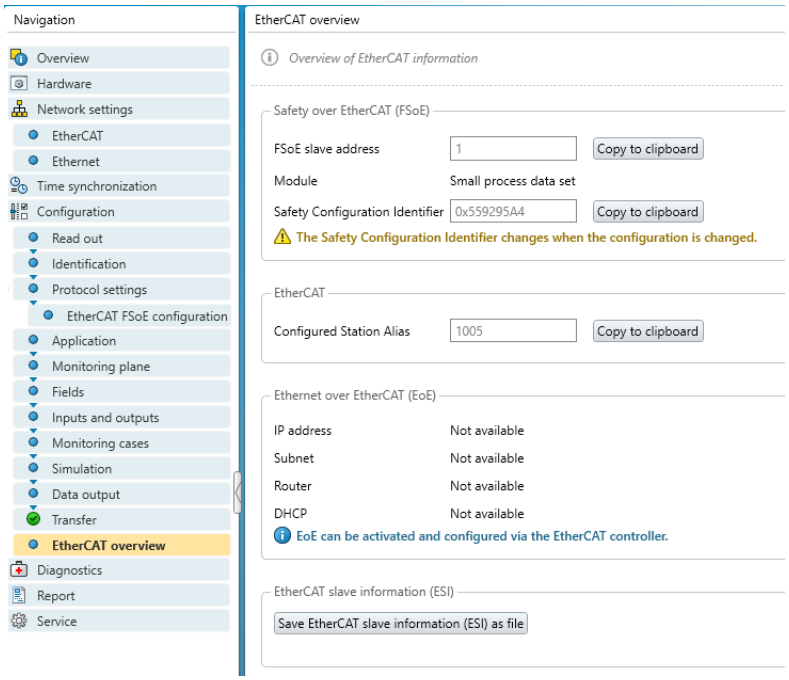


Figure 22 EtherCAT Overview SICK Configuration

Go to the EtherCAT tab under “Network Settings” and change the station alias to the one used in the system (Figure 22 shows the different tabs and the overview communication). This is normally automatically read from the hardware if the scanner is connected to the EtherCAT network in the desired order and the network has been set up in the ctrlX IO. Furthermore, enter the FSoe Slave Address under “EtherCAT FSoE configuration”.The final communication overview should look like the one shown in Figure 22. Copy the Safety Configuration Identifier for later use.

Next, the monitoring cases and other settings need to be configured. Settings like whether the laser will be moving or stationary, the kind of field, the height and resolution (hand, leg, etc.), and more settings, depending on the desired application.

The monitoring cases are defined areas, in which the laser scanner will send a warning or alarm, to let the system know, there is something within the defined area. Figure 23 shows the two monitoring areas for the mobile robot. If an object is detected in the yellow area, the robot should slow down and within the red area, it should completely stop. The monitoring cases should also be defined, so the systems output this information over FSoE under “Monitoring cases”.

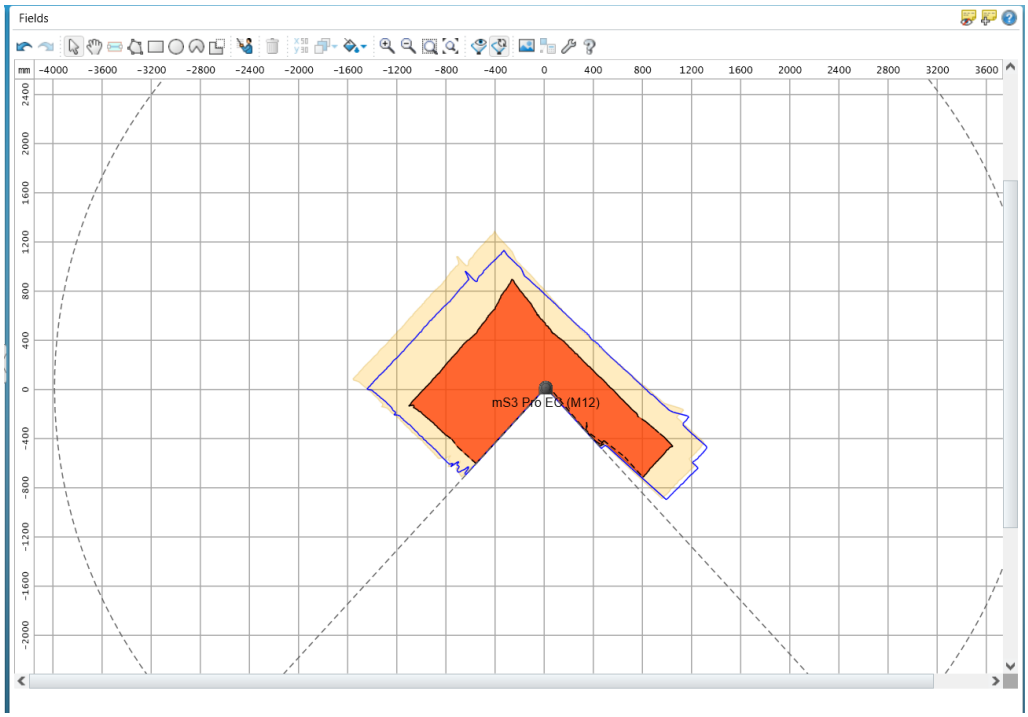


Figure 23 Monitoring FIelds for Mobile Robot

Once everything has been configured, transfer the configuration to the device. A protocol will be created. This protocol needs to be verified to enable the new configuration on the laser scanner. The following password is also necessary to complete the transfer.

Password: SICKSAFE

3.6 SAFETY Engineering

The SAFETY Engineering software is where the safety communication comes together.

Figure 24 Safety Engineering Devices

From the device library on the right side, drag and drop the necessary devices for the FSoE network. Place them in the order they are connected. Devices 1, 2 & 3 should also follow the same order as the FSoE slave numbers, meaning device 1, should be FSoE slave 1, and so on. Furthermore, a (double) emergency stop is inserted to send a stop signal to all FSoE slaves. Figure 24 shows the devices used for the robot, following the order they are connected to each other (see Figure 12 for reference).

The parameters of each device can be manually configured. These configuration needs to be the same one, as the one set up in the corresponding device software. These parameters include the FSoE slave address, the safety message length (profile), the CRC, and others. The CRC numbers copied from the device's hardware should be inserted here. For an unknown reason, the laser scanner only works, if the CRC number is set to 0, and not the corresponding CRC. See Appendix 6.3 for the full settings.

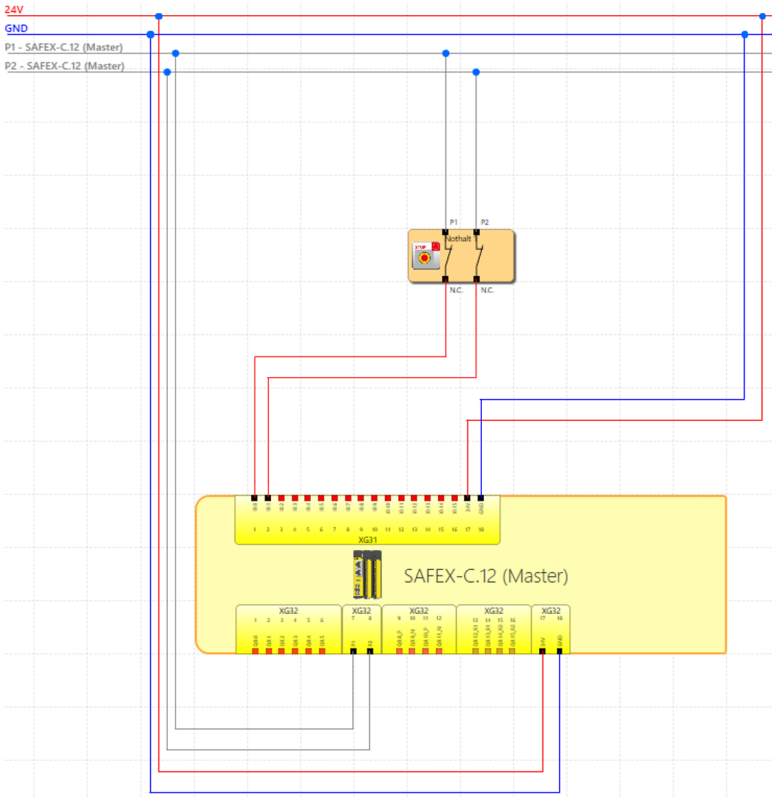


Figure 25 CtrlX SAFEX Wiring Scheme

Figure 25 (larger version in Appendix 6.2) shows the wiring scheme of the ctrlX SAFEX. Pulse signals 1 & 2 are connected to the XG31 pins 0 & 1 respectively, interrupted by a (double normally closed) emergency stop. The origin of the pulse signals are pins XG32 6 & 7, corresponding to the pulse signals 1&2.

The wiring schemes of the other devices do not need to be defined, because they are only connected to the EtherCAT network. However, the functionality of the safety function in each device still needs to be configured. This configuration needs to correspond to the one done before in each device's software. The configuration can be found in Appendixes 6.3 and 6.4. These configurations show which bits are set to high or low, depending on multiple situations. For example, the two drives turn off their STO if the emergency stop is pressed.

3.6.1 Transfer Data to ctrlX SAFEX

Once the configuration of each device, as well as the safety network has been completed, the configuration can be transferred to the hardware. First compile the current workspace to find possible errors, such as plausibility errors. Use the USB-C port for communication with the hardware. Connect the hardware to the developer's computer and select the correct COM port. A window will appear, asking for the serial code of the device. Enter the serial code and establish the connection. Send the configuration to the hardware. In the newer versions of ctrlX Safety Engineering, a password might be needed to overwrite the current configuration. Use the following password if asked for one. Once the configuration has been properly transferred, the SAFEX will reboot. After a couple of minutes, the current status of the hardware should be "run" and the light indicator should be blinking green. Lock the configuration, and the blinking should stop.

Password: boschrexroth

3.6.2 Checking for FSoE Functionality

If the SAFEX indicator is green, everything should be working correctly.

However, to fully check the functionality of the FSoE communication by trying to turn on the motors. Without the FSoE, the STO will not be enabled, disabling the rotation of the wheels completely. Furthermore, the configured devices on the ctrlX IO (see 3.2.1) have multiple FSoE bits allocated to the status of the connection. If the status is not being updated, the most likely error is the Slave to Slave configuration on the ctrlX IO. By connection to the ctrlX CORE via the IO or PLC apps, the live data can be shown, thus being able to see if the status bits are high or low.

If the FSoE communication cannot be established, the SAFEX will probably go into "RunIntern" status. Unfortunately, no further information is given, when this is the case. Common errors include, some configuration is not correct on the ctrlX IO, e.g. Slave to Slave, the CRC is not the correct one or the FSoE Slave Address is also incorrect. The steps in Figure 26 need to be followed to establish the connection. More information about errors can be gathered from the device-specific software. The *Elmo Application Studio II* has a "Safety Monitoring" tool, which might give more insight into the problem. One instance showed for example, that the Elmo drive was sending a reset signal, every time the connection reached the "Parameter" step. For this reason, the SAFEX was stuck in the "RunIntern" mode.

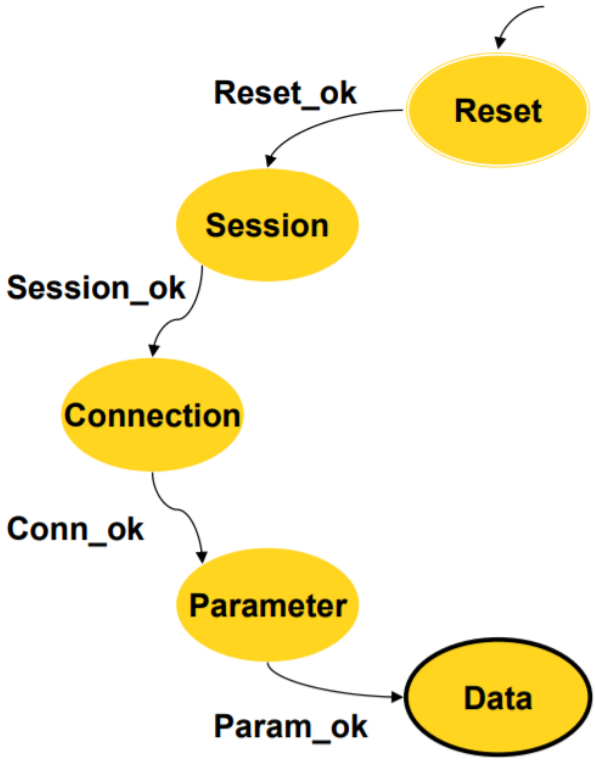


Figure 26 FSoE Connection Steps

4 App Build Environments & SDKs

4.1 Introduction

The ctrlX Core X7 has a USB-C port. This port is mainly used for data transfer or for the use of a Wifi dongle to connect the Core to the internet. Nevertheless, it is possible to use this port for other purposes. One of them would be to establish a 2.4 GHz wireless connection with a remote controller. The inputs of the controller would then be used to steer the movement of the wheel, thus having control over the mobile robot. However, such an implementation is not inherently available to the software on the core. For this reason, the development of an app was necessary.

“The SDK is a Software Development Kit for the development of software applications (APPS) for ctrlX AUTOMATION devices, especially for Linux-based ctrlX OS devices. It enables the development of APPS in various high-level languages” https://developer.community.boschrexroth.com/t5/Store-and-How-to/SDK-for-ctrlX-AUTOMATION/ba-p/16509

The development of apps on the SDK is based on Snaps and snapcraft. For this reason, a basic understanding of these is recommended.

4.2 Software Version

IMPORTANT Make sure the software versions of the different libraries are compatible with one another, as well as the devices used (ctrlX Core). This is one of the most common errors when developing for the SDK.

The versions used for the USB Remote Controller App are:

- ctrlX WORKS WRK-V-0120.8
  - App Build Environment: Ubuntu 20.04.6 LTS app-builder-amd64 ttyS0
- ctrlx-automation-sdk-1.18.0
- ctrlx-datalayer-1.9.1
- ctrlX Core X7 with ctrlX OS 1

4.3 Setup Overview

For a video guide, follow the steps in this video:

https://www.youtube.com/watch?v=9f7_FId2PK4

The video explains the following procedure to get started:

1. **Install ctrlX WORKS with the function “App Build Environment”**

2. **Download and run the proxy (PX.exe)**

3. **Create a new App Build Environment (QEMU VM)**

4. **From ctrlX WORKS create and start a new App Build Environment (QEMU VM) instance.**
   Wait until the VM is shutting down automatically after the initialization phase. Then restart your VM. For more information see ctrlX WORKS Feature App Build Environment.

5. **When ctrlX WORKS enables the SSH link click this link to start an SSH session.**
   On the password prompt enter `boschrexroth`

6. **Install the latest ctrlX AUTOMATION SDK from GitHub**
   Call this script:

   `~/scripts/install-sdk.sh`

   a. The script downloads ctrlx-automation-sdk.zip from GitHub and unzips it to the folder /home/boschrexroth/ctrlx-automation-sdk/

   It is possible that the newest library is not compatible with the device used. Therefore, check which version is necessary. This seems to be a major problem with the ctrlx-datalayer library. A manual installation of the libraries might be necessary.

7. **Change to project folder /home/boschrexroth/ctrlx-automation-sdk/samples-cpp/hello.world**

8. **Build the snap**
   To build a snap for a ctrlX CORE X7 enter:
   `./build-snap-amd64.sh`

   As a result a snap file `sdk-cpp-hello.world_...._amd64.snap` respectively `sdk-cpp-hello.world_...._amd64.snap` will be created. Download this file.

9. **Install the snap in the device**
   First, change the device to Setup

   Go to the Web interface of the ctrlX Core → Settings →Apps.
   Allow the installation from an unknown source under ⋮ → Settings

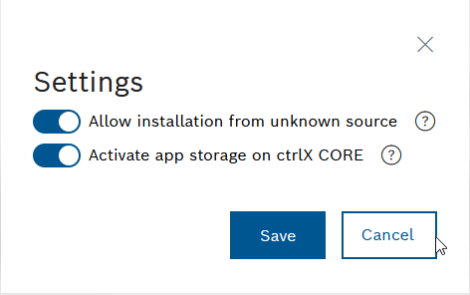


Figure 27 Allow installation from unknown sources

   Install the snap. (Install from file)

Information from YouTube video and https://boschrexroth.github.io/ctrlx-automation-sdk/quick-start-guide.html

Video alternative:

https://tube.video.bosch.com/playlist/dedicated/3324/0_abae1zc6/0_1q11yejj

4.4 Test and Debugging

The Web Interface should interrupt the installation of the snap if an error in key parameters is missing. A common mistake is not having the correct base version for the snap. This can be changed

in the “snapcraft.yaml” file. Figure 28 shows an exert of the snapcraft.yaml file from the sdk-py-provider-alldata app. Here the core20 is used for the base of the snap. This is the Ubuntu version running in the ctrlX Core. If ctrlX Core OS 2.2 is used, base Core22 should be used.

```
name: sdk-py-provider-alldata
title: SDK Python Provider All Data
version: 1.0.0
summary: ctrlX Data Layer provider sample in Python
description: |
  ctrlX Data Layer provider in Python publishing nodes of all data types
base: core20
confinement: strict
grade: stable
```

Figure 28 Snapcraft.yaml Base

4.4.1 Logbook for Debugging

The Web Interface Logging function can be used to print prompts from the installed app. Such prompts need to be explicitly defined in the source code. This can include status updates, notifications, as well as errors and warnings.

Enable the developer information under Diagnostics→Logbook→Settings.

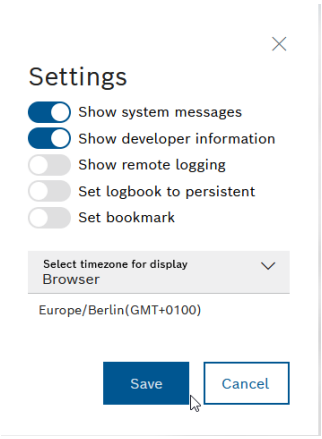


Figure 29 Log: Show Developer Information

Once this has been done. The print text from the source code of the installed snap should appear in the logbook. To make the logbook look more seamless, it can be filtered after the app is installed.

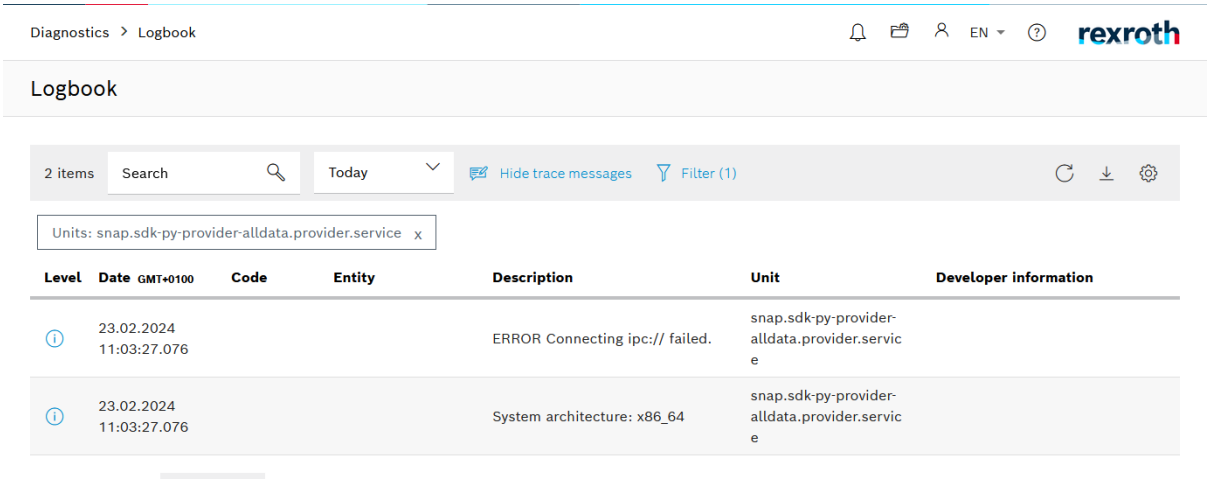


Figure 30 Logbook Filtered after Snap

4.5 Snap USB Remote Controller

The source code was mostly based on the following sample from Bosch Rexroth:

https://developer.community.boschrexroth.com/t5/Store-and-How-to/Using-the-libusb-library-within-a-snap-to-access-USB-devices/ba-p/52477

The link also provides instructions on how to install the needed libraries and build the snap. Nevertheless, before the snap can be built, the SDK environment has to be established as stated in Chapter 4.3.

Once the SDK environment has been correctly established with the correct library version, a sample Python code can be copied. The copy can then be modified with the code provided in the link, to enable USB access. However, once the snap was built, many problems were encountered. Most of them are stated in the following chapter. It is possible, that the code provided by the link, works directly. This will probably depend on the different versions used.

4.5.1 Datalayer.Provider-All-Data as Base

```
apps:
  provider:
    command: bin/main.py
    plugs:
      - network
      - datalayer
    daemon: simple
    passthrough:
      restart-condition: always
      restart-delay: 10s

parts:
  provider:
    plugin: python
    source: .
    stage-packages:
      - libzmq5
      - ctrlx-datalayer

plugs:
  datalayer:
    interface: content
```

Figure 31 Datalayer.provider-all-data yaml File

After multiple tests, it was established, that with the version used, the code provided by Bosch Rexroth for the USB communication was unable to establish a reliable connection to the datalayer and USB interface. The main problem seems to be the communication over IPC (see 4.6.2). For this reason, it was decided to use another sample as a base for the communication with the datalayer.

The Python code used as the base for the communication was the “datalayer.provider.all-data” sample. This sample provided the foundation for the datalayer communication. It is unknown, why the communication with the datalayer works in this sample and not the other. The most likely cause is once again the version compatibility of the multiple libraries.

Figure 31 shows the yaml file used for building the snap. Here it becomes apparent which interfaces (plugs) and packages are used by the program. This section needs to be expanded to add the

functionality to read USB inputs. The program linked in Chapter 4.5 was combined with this foundation, to enable such a functionality.

4.5.2 PyUSB

To enable the hardware to read the Inputs from the USB dongle, the PyUSB was needed. This library allows access to the USB port with Python. This library needs to be included in the snap building process. These addition were made to the snapcraft.yaml file to explicitly declare, which libraries and interfaces the snap required (Figure 32). The interfaces “raw-usb” and “hardware-observe” allow access to the USB port on the ctrlX Core. Furthermore, when building the snap, the newest version is installed (“pip3 install pyusb”). The library LibUSB is also needed. This library is also prone to errors, due to compatibility and the program not finding the library.

```
apps:
  provider:
    command: bin/main.py
    plugs:
      - network
      - datalayer
      - raw-usb
      - hardware-observe
    environment:
      LD_LIBRARY_PATH: $SNAP/lib/x86_64-linux-gnu
    daemon: simple
    restart-condition: always
    passthrough:
      restart-delay: 30s
```

```
parts:
  provider:
    plugin: python
    source: .
    override-build:
      snapcraftctl build

      pip3 install pyusb
    build-packages:
      - libusb-1.0-0
    stage-packages:
      - libzmq5
      - ctrlx-datalayer
      - libusb-1.0-0
    python-packages:
      - pyusb
```

Figure 32 PyUSB library for Snap Build

4.6 Common Problems

4.6.1 Library ctrlX-Datalayer Not Found

A common error with the SDK development is the access to the datalayer. Most times, the problem originates from the compatibility of the library with the device. If access to the ctrlX Datalayer is necessary, it is recommended to test one of the samples included in home/boschrexroth/scripts/ctrlx-automation-sdk/samples-python (or samples-cpp)/… The sample used as a base for the remoter controller was the “alldata.provider.service” example.

```
Cleaning up priming area
Cleaning up staging area
Cleaning up parts directory
*EXPERIMENTAL* --target-arch for core20 enabled.
Setting target machine to 'arm64'
Pulling compiled
+ snapcraftctl pull
Failed to fetch stage packages: Error downloading packages for part 'registernode': The package 'ctrlx-datalayer' was not found..
make[2]: *** [CMakeFiles/registerNode.dir/build.make:87: registerNode] Error 2
make[2]: *** Deleting file 'registerNode'
make[1]: *** [CMakeFiles/Makefile2:79: CMakeFiles/registerNode.dir/all] Error 2
make: *** [Makefile:130: all] Error 2
~/ctrlx-automation-sdk/samples-cpp/datalayer.register.node
 *  Terminal will be reused by tasks, press any key to close it.
```

Figure 33 Error - ctrlx-datalayer was not found

This problem seems to be very common. Multiple factors could be the cause of the problem. However, trying to uninstall the ctrlx-datalayer and reinstalling it again, seems to work. Please consider the different versions. The latest one might not be the most suitable one. Another option is to start a completely new SDK and try installing the library again.

The following forum links provide a better insight into the problem:

Error SDK V1.18. Not able to snap, because the package 'ctrlx-datalayer' was not found:

https://developer.community.boschrexroth.com/t5/SDK/Error-SDK-V1-18-Not-able-to-snap-because-the-package-ctrlx/m-p/66989#M679

SDK USB Mouse Data Layer:

https://developer.community.boschrexroth.com/t5/SDK/SDK-USB-Mouse-Data-Layer/m-p/56504#M520

Python Datalayer - Error importing ctrlxdatalayer:

https://developer.community.boschrexroth.com/t5/SDK/Python-Datalayer-Error-importing-ctrlxdatalayer/m-p/60878

4.6.2 IPC Connection Failed

The connection between snaps running on the ctrlX Core and the datalayer is established to the IPC (or TCP for external connections). Nevertheless, the communication does not seem to work every time. Even with the samples provided by the App Build Environment, this communication seems to also be affected by compatibility issues with the different versions. It is recommended to test the samples with different versions, to find the most suitable version.

Helpful forum links:

https://developer.community.boschrexroth.com/t5/ctrlX-CORE/Problems-with-establishing-a-connection-between-a-TCP-Client-on/td-p/24746

4.6.3 LibUSB Not Found

One last problem seems to appear rather frequently. Once the snap has been built and installed, it has a hard time finding the LibUSB library. This library is necessary to enable access to the USB port. This problem was fixed by explicitly defining the library path (see Figure 32: $SNAP/lib/x86_64-linux-gnu). Furthermore, by using the command “pip3 install pyusb” during the build step, it is ensured to have the latest PyUSB library. This problem still persists on the virtual Core.

Helpful forum links:

https://developer.community.boschrexroth.com/t5/ctrlX-CORE/libusb-on-ctrlX-CORE/td-p/52185

4.7 OS Update

The firmware of the ctrlX Core X7 was updated during the development process. For this reason, the snaps had to be adapted and built once again. As explained in Chapter 4.4, the older OS used the Ubuntu Core20. The newer ctrlX CORE has the ctrlX OS V2.2, based on the Ubuntu Core22. This line has to be changed to make the snap compatible with the newer version. The snap can be simply built again and the functionality should be the same.

4.7.1 Library Version and Path

Through the OS update, libraries can become non-compatible with the newer OS version. For this reason, it is recommended to use the same SDK version for developing snaps for this version. In this case ctrlx-automation-sdk-2-2. Once the libraries have been updated, it should work once again.

One further problem was found when updating the program to the newer version. The problem stated in Subchapter 4.6.3 appeared once again. The older solution seems not to work. The cause was that the newer OS version stores the snaps in a slightly different path.

```
140         backend = usb.backend.libusb1.get_backend(
141             find_library=lambda x: "/snap/sdk-py-provider-alldata/current/usr/lib/x86_64-linux-gnu/libusb-1.0.so.0")
142             #find_library=lambda x: "/snap/sdk-py-provider-alldata/current/lib/x86_64-linux-gnu/libusb-1.0.so.0")
```

Figure 34 Change in Library Path

Figure 34 shows the slight change in the library path. The green comment shows the old library path. The newer version seems to store its libraries inside a “usr” folder.

For more information see the post on the forum:

https://developer.community.boschrexroth.com/t5/ctrlX-CORE/LibUSB-not-found-on-version-ctrlX-OS-2-Ubuntu-core22/m-p/91829#M2460

4.8 Controller Specific Settings

4.8.1 USB Device Identifier

```
# Find the connected usb device using the Vendor ID and the Product ID
dev = ''
try:
    dev = usb.core.find(idVendor=0x046d, idProduct=0xc21f, backend=backend)
except usb.core.NoBackendError as e:
    self.full_data.set_string("something wrong with libusb")
    print("something wrong with libusb")
    return

if not dev :
    print("controller not found")
    print("trying again in 5s")
    self.full_data.set_string("controller not found")
    time.sleep(5.0)
    self.start_reading()
```

Figure 35 USB idVendor and idProduct

Once the libraries have been found and loaded correctly and the snap has connected successfully to the datalayer, the program will search for the controller. The controller used for this app is the Logitech F710. The device is identifiable by two numbers. The idVendor, which should be company specific (Logitech), and the idProduct, most likely product specific (remote controller F710). If the idVendor and idProduct values of the USB device connected don’t match the ones in Figure 35, the program will not find the controller and no inputs will be read.

4.8.2 Decodification

If the controller is found, the inputs of the controller will be read each time a new input has been sent from the controller. If the controller reads any input, the value for all buttons, joysticks, and triggers is sent as an array. This array can be seen in Figure 36. The datalayer node “full-data”, displays all the data received. Nevertheless, the program needs to decode this array, to know which button was pressed and additionally, the value attached to it for analog inputs. Once the array has been identified, the other nodes are filled with the corresponding data. Each element of the array corresponds to one or multiple inputs of the controller. In this figure, the B-button was pressed (element #4 = 32) and the left joystick was tilted all the way to the front (#9 = 255).

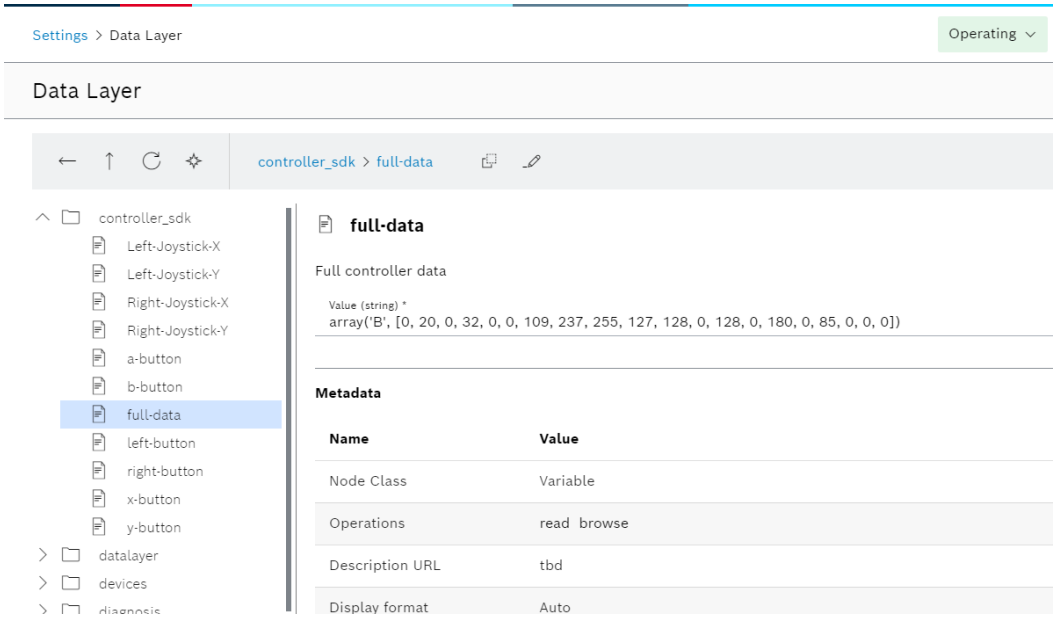


Figure 36 Controller data input

The decodification was only performed for the face and shoulder buttons, as well as the joysticks. All the face buttons share the same array index. Depending on what button was pressed, the value of this array element is added. For example, the left shoulder button adds the value “1” and the b-button adds the value “32”. If both buttons are pressed simultaneously, the output for this array element would be “33”. By adding a case and an “AND” logical operator, the program knows which button was pressed. Each joystick is separated into two elements, representing the X and Y axes. Since these values are analog, they range from [0…255], 128 being the center of the axis. For simplicity purposes, the joystick values were transformed to a range of[-1…1], 0 being the center.

```
#Face and shoulder Buttons decodification
    clicked_button = int(data[3])
    if(clicked_button & 1):
        self.left_button.set_bool8(True)
    if(clicked_button & 2):
        self.right_button.set_bool8(True)
    if(clicked_button & 32):
        self.b_button.set_bool8(True)
    if(clicked_button & 128):
        self.y_button.set_bool8(True)
    if(clicked_button & 64):
        self.x_button.set_bool8(True)
    if(clicked_button & 16):
        self.a_button.set_bool8(True)

#Joysticks Decodification
    l_joystick_x_scaled = (int(data[6])/128)-1
    self.l_joystick_x.set_float32(float(l_joystick_x_scaled))

    l_joystick_y_scaled = (int(data[8])/128)-1
    self.l_joystick_y.set_float32(float(l_joystick_y_scaled))

    r_joystick_x_scaled = (int(data[10])/128)-1
    self.r_joystick_x.set_float32(float(r_joystick_x_scaled))

    r_joystick_y_scaled = (int(data[12])/128)-1
    self.r_joystick_y.set_float32(float(r_joystick_y_scaled))
```

Figure 37 Controller Input Decodification

5 CtrlX PLC

The PLC program contains the logic of the mobile robot. Here is where all the inputs, outputs, and case-dependent reactions are defined.

5.1 Data Layer - EtherCAT Devices

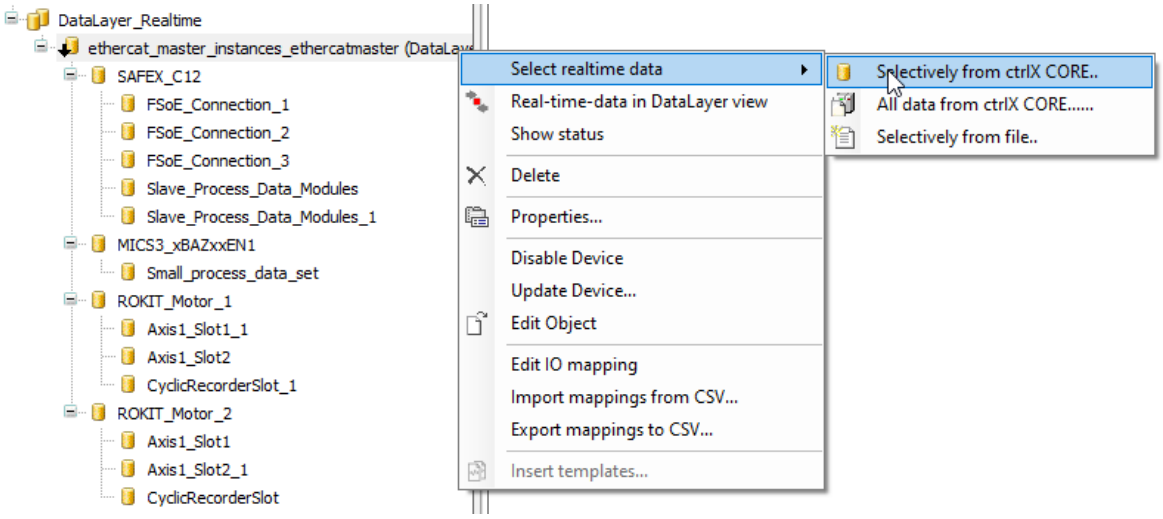


Figure 38 Importing EtherCAT Datalayer Devices to PLC

The easiest way to access the data of the devices connected is through the datalayer. The datalayer is shared between all ctrlX CORE apps running on the hardware. By importing the data from the devices connected to the EtherCAT network, this data will become available on the PLC program. Follow the steps in Figure 38 to import the device-specific data. The devices should follow the same structure as the one defined in the ctrlX IO configuration.

5.2 PLC Tree Overview

The PLC Program consits of the muliple programs, functions and functions block. An overview of them can be seen in Figure 39. The routine consits of mainly two tasks running parallel to eachother:

- microScan: Checks the defined non-safe areas, triggered by the Laser Scanner if an object is detected within them. If true, the velocity is decreased or set to zero.
- PLC_PRG: This is the main task of the program, where most of the logic is found. The remote controller inputs are read, the function to enable the control of the motore drives is called, the velocity transofrmation function is called and the motor velocity is also read.

Furthermore the program consits of the following function blocks:

- Fb_VelmodeCoE_2: This functions reads the statusword of the motor drives and (if enabled) sends the corresponding controlword to enable the motor control. Futhermore, the desired speed is also sent here.
- AngVelToTwistM/TwistMtoAngVel: These functions transforms the desired linear and angular velocity of the robot to motor velocites and vice versa.

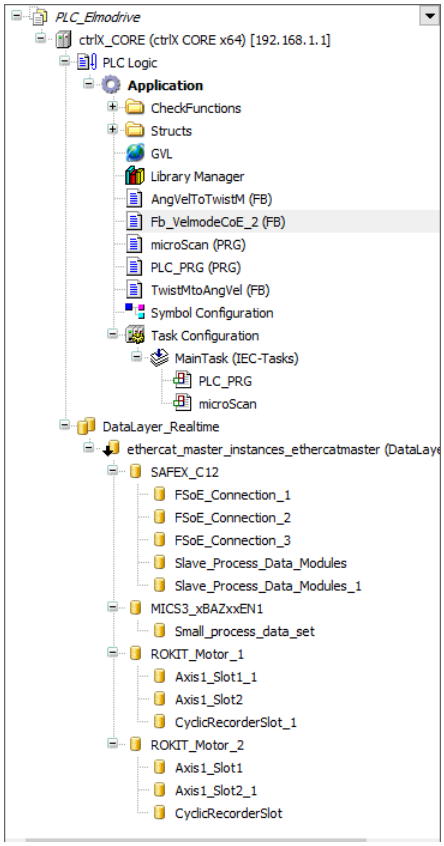


Figure 39 PLC Tree Program Overview

5.3 Laser Scan Measurement

```
IF NOT(Microscan_StatusWord1.RunModeActive) THEN
    //Sick microscan3 Sicherheitsfunktion angehalten
    strSensor := 'Scanner safey function not active';

END_IF

IF NOT(Microscan_StatusWord3.ApplicationError ) THEN
    //Applikationsfehler - Fehler beheben + Neustart Sicherheitsfunktion
END_IF

IF NOT(Microscan_StatusWord3.DeviceError ) THEN
    //Gerätefehler - Fehler beheben + Neustart Gerät
END_IF

IF(Microscan_StatusWord2.NonSafeCutOffPath01) AND (Microscan_StatusWord2.NonSafeCutOffPath03) AND Microscan_StatusWord1.RunModeActive THEN
    //kein Objekt erkannt
    strSensor := 'No object';
    strDrive := 'Fast movement';
    dLaserRangeFactor:= 1;

ELSIF (NOT(Microscan_StatusWord2.NonSafeCutOffPath01)) AND (Microscan_StatusWord2.NonSafeCutOffPath03) AND Microscan_StatusWord1.RunModeActive THEN
    //Object im entfernten/großen Überwachungsfeld erkannt
    strSensor := 'Object in large distance';
    strDrive := 'Slow movement';
    dLaserRangeFactor:= 0.5;
ELSIF Microscan_StatusWord1.RunModeActive THEN
    //Object im nahen/kleinen Überwachungsfeld erkannt
    strSensor := 'Object in small distance';
    strDrive := 'No movement';
    dLaserRangeFactor:= 0;
END_IF
```

Figure 40 Laser Scan

This secondary program. Reads the alarm bits set on the laser scanner module. If the alarms have been triggered, an object was detected within that range. This creates a velocity factor, which decreases or completely stops the motors.

5.4 Controller Input

```
b_Executel := TRUE;
fbReadNodel(Execute:= b_Executel, NodeName:= str_address_Yjoystick, Value:= r_Yjoystick);
IF (fbReadNodel.Done = TRUE) OR (fbReadNodel.Error = TRUE) THEN
    b_Executel := FALSE;
    fbReadNodel(Execute:= b_Executel, NodeName:= str_address_Yjoystick, Value:= r_Yjoystick);
END_IF
```

Figure 41 PLC Read Data from Datalayer

The first function of the main program is reading the data from specific datalayer nodes. These nodes are where the remote controller inputs are. If the remote controller snap was installed correctly, the nodes should update, as soon as a button of the controller is pressed. The function shown in Figure 41 is repeated for every node read.

5.4.1 Controller Overview

Figure 42 Remote Controller Input Overview

Figure 42 shows a picture of the remote controller used (Logitech F710) and what each relevant controller input does. To have control over the robot, the first step is to press the “A” button to enable the power to the motors. Once the motors have their power supply enabled, the robot can be controlled. Nevertheless, as a safety measure, the control is only enabled if the left shoulder button is pressed (hold). When these two requirements are fulfilled, the robot’s wheels will move according to the desired speed, according to the joysticks. The Y-axis of the left joystick represents the linear velocity of the robot and the X-axis of the right joystick controls the angular velocity. The speed of the wheels is increased by double while the right shoulder button is pressed. If the laser scanner detects an object within the defined non-safe areas, the velocity of the robot will be decreased or fully stopped. To overwrite these constraints, hold down the “B” button.

5.5 Motor Drive Control

Due to safety features, the ROKIT motor drives have to follow a strict flowchart, to power them on and enable their control. Figure 43 shows the state order. If a step is skipped, the motor will not turn on and will probably lead to an error. The current state of the drive is known through the "statusword" and the desired next state can be sent through the "controlword".

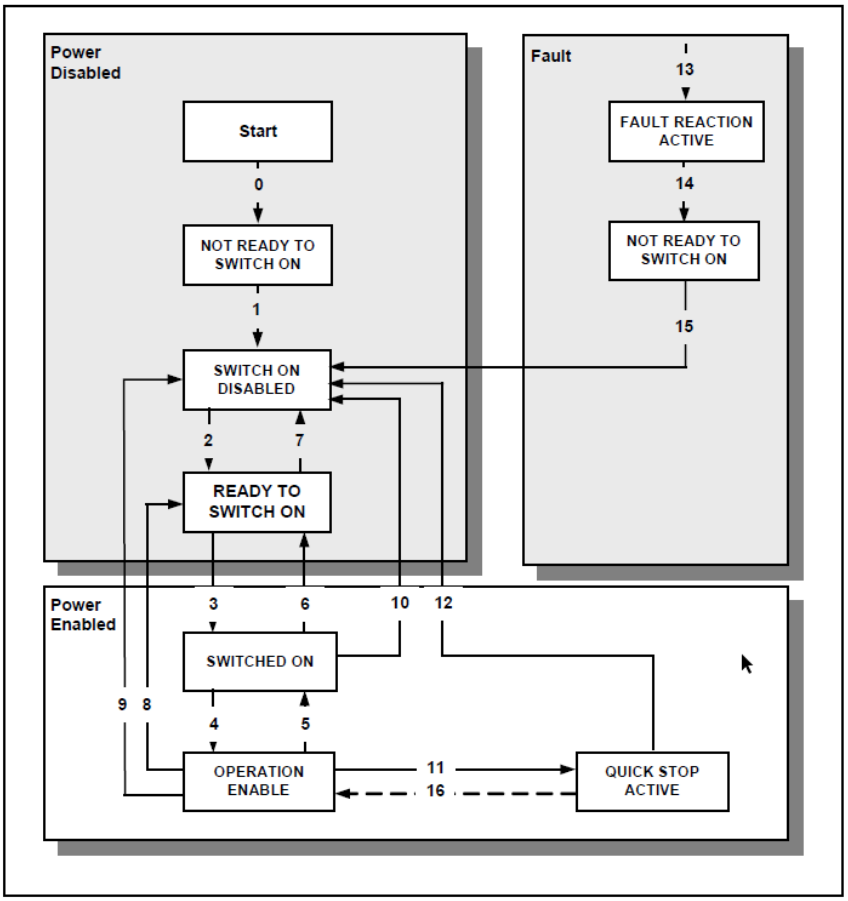


Figure 43 Elmo Drive State Flowchart

5.5.1 Statusword

Bit	Description
0	Ready to switch on
1	Switched on
2	Operation enabled
3	Fault
4	Voltage enabled
5	Quick stop
6	Switch on disabled
7	Warning
8	Manufacturer specific
9	Remote
10	Target reached
11	Internal limit active
12 - 13	Operation mode specific
14 - 15	Manufacturer specific

Figure 44 Elmo Drive Statusword Bits

The statusword of the Elmo drive describes, which state the drive is the current one. The status word also gives additional information, e.g. if the remote control is enabled or warnings/error alarms. Bit 9 (remote) should be high if the remote control is enabled (as explained in Subchapter 3.4). Through the statusword, it is known, which step should be the following (see Controlword). If the PLC is connected to the host PC and running on the hardware, it is possible to see the live data. This data is found under the EtherCAT devices >> Elmo Drive (ROKIT) >> Axis1_Slot2.

5.5.2 Controlword

The Controlword is used to set the motor drive into the next step of the flowchart (see Figure 43). Nevertheless, for the change in state to take place, it needs to be a legal move. If for example, the . motor is currently in step 1, it is not possible to change to step 7. If such a change is sent with the cotntrolword, this will only be ignored. If the "A" button is presed, the motor drive is turned on. However, in the background, the motor drive goes through all the steps until "Operation Enabled" has been reached, only changing to the next step, when the correct statusword is sent in return. This process happens in the "Fb_VelmodeCoE_2" function block.

Command	**Bits of the *controlword***					**Transitions**
	7	**3**	**2**	**1**	**0**	
	Fault Reset	**Enable Operation**	**Quick Stop**	**Enable Voltage**	**Switch On**	
Shutdown	0	X	1	1	0	2, 6, 8
Switch ON	0	0	1	1	1	3*
Switch ON	0	1	1	1	1	3**
Disable Voltage	0	X	X	0	X	7, 9, 10, 12
Quick Stop	0	X	0	1	X	7, 10, 11
Disable Operation	0	0	1	1	1	5
Enable Operation	0	1	1	1	1	4, 16
Fault Reset	↑	X	X	X	X	15

Figure 45 Controlword Bit Description

5.5.3 Desired Speed

The desired speed depends on three main factors, the "turbo" button, the laser scanner alarm, and the motor control enabler. The first factor shown in Figure 46 is the turbo factor. While the right shoulder button is pressed, the target velocity is doubled. The next factor taken into consideration is the one set by the laser scanner. The scanner field alarms can halve the target speed, set it to zero, or just leave it as is (more information in Chapter 5.3). The final factor sets the velocity to zero unless the left shoulder button is pressed, thus preventing the accidental movement of the robot.

Finally, the target velocity is transformed to motor velocities by the "TwistMtoAngVel" function block and sent to the motor drives.

```
//If right shoulder pressed -- Enable turbo

IF (b_rtrigger=TRUE) THEN
    r_turbo_factor := 2;
    ELSE
        r_turbo_factor := 1;
END_IF

//If b button pressed -- Ignore Laser Limitation
IF (b_Bbutton=FALSE) THEN
    r_Yjoystick := r_Yjoystick * r_scale_factor * r_turbo_factor * dLaserRangeFactor;
    r_Xjoystick := r_Xjoystick * r_scale_factor * r_turbo_factor * dLaserRangeFactor;
    ELSE
        r_Yjoystick := r_Yjoystick * r_scale_factor * r_turbo_factor;
        r_Xjoystick := r_Xjoystick * r_scale_factor * r_turbo_factor;
END_IF

//If Left shoulder pressed -- Enable speed
IF(b_lTrigger=TRUE) THEN
    tMessageIN.LinVel := r_Yjoystick;
    tMessageIN.AngVel := r_Xjoystick ;
    ELSE
        tMessageIN.LinVel:= 0;
        tMessageIN.AngVel:= 0;
END_IF

//Update Incoming Twist Message
TwistIn(tTempMessage := tMessageIN, diOmegaR => diDesOmegaR, diOmegaL => diDesOmegaL);
```

Figure 46 Target Speed Scale Factor

5.5.4 Read Current Motor Speed

```
// READ CURRENT VELOCITY
//  Read velocity on the CoE Bus
//  A counter is used to read every three cycles, it seems to not be able to read values more often. Velocity default unit RPM
//Right Motor
IF 2 = counterR THEN
    fbECATCoeReadSdo.Execute      := TRUE;
    fbECATCoeReadSdo.MasterName   := ADR(strMasterName);
    fbECATCoeReadSdo.SlaveAddress := ROKITMotor1Addr;
    fbECATCoeReadSdo.Index        := by_RIndex;
    fbECATCoeReadSdo.SubIndex     := bySubIndex;
    fbECATCoeReadSdo.ValueAdr     := ADR(diReadValueR);
    fbECATCoeReadSdo.SizeOfValue  := SIZEOF(diReadValueR);
    diCurrentSpeedR := diReadValueR;

    counterR:= 0;
END_IF

    fbECATCoeReadSdo(); //Call function with the set parameters
    counterR := counterR + 1;

    IF TRUE = fbECATCoeReadSdo.Done THEN
        fbECATCoeReadSdo.Execute      := FALSE;
    END_IF

     IF TRUE = fbECATCoeReadSdo.Error THEN
      fbECATCoeReadSdo.Execute      := FALSE; // Error handling
    END_IF
```

Figure 47 Current Velocity Motor

Unlike the target motor speed, which can be directly set through the datalayer module, the current speed of the motor must be requested explicitly. The current motor speed can be obtained by reading the data sent through the CoE (CAN over EtherCAT) communication under a specific index. This specific index is the 0x606C (not seen in Figure 47, defined in the variable declaration). Other parameters can be read and possibly also written in this manner. Each motor needs to be read independently (identifiable by the EtherCAT slave address number).

For an unknown reason, the read command gets stuck if the velocity is read every cycle (around every 20 ms). For this reason, the current motor velocities are read every 3 cycles (every 60 ms).

5.6 Velocity Transformation

5.6.1 Transformation Overview

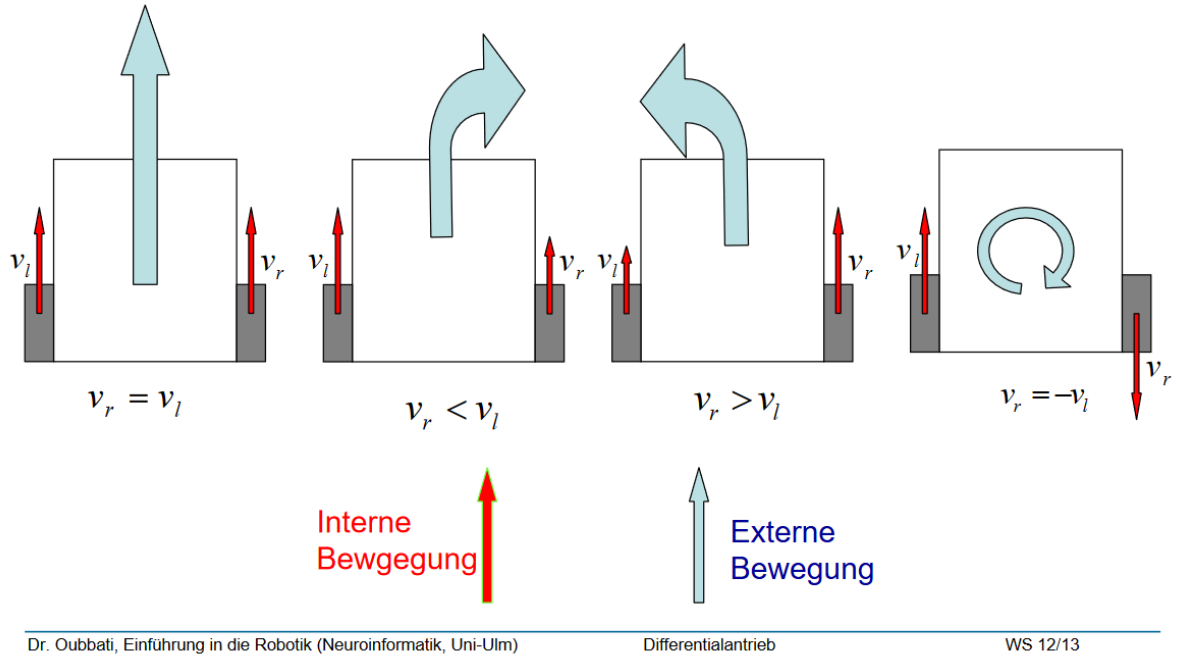


Figure 48 Differential Robot Drive Movement Overview (Differentialantrieb, Dr. Oubbati Uni-Ulm)

Figure 48 gives an overview of what kind of movement the wheels' linear velocities cause on the robot. This figure shows four different cases:

1. Both wheels move at the same velocities → The robot moves only linearly.
2. The right wheel moves slower than the left one → The robot moves linearly with a rotation to the right.
3. The right wheel moves faster than the left one → The robot moves linearly with a rotation to the left.
4. The wheels move at equal but opposite velocities → The robot will turn without any linear component (in this case to the right).

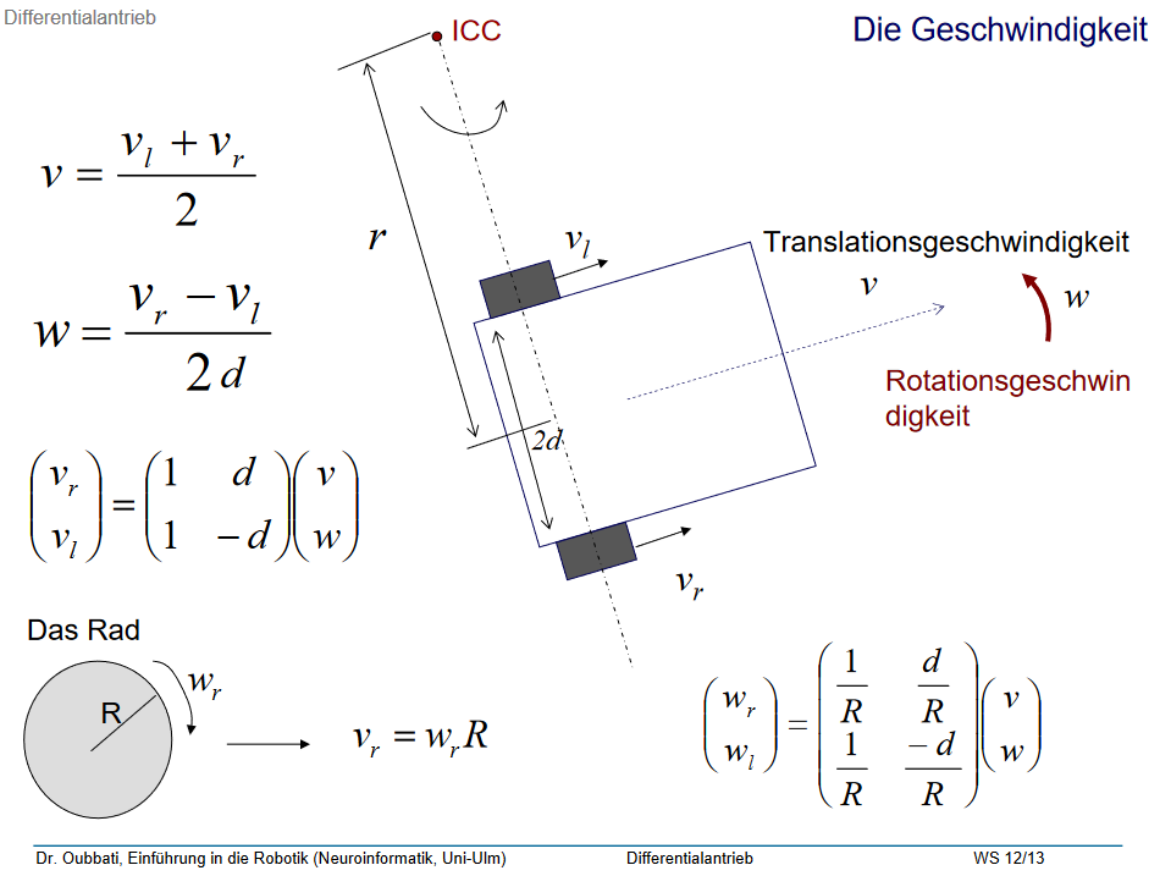


Figure 49 Axis Transformation Differential Drive (Differentialantrieb, Dr. Oubbati Uni-Ulm)

5.6.2 AngVelToTwistM Function Block

This function transforms the angular velocities of each wheel (motor) into the global linear and angular velocities of the robot. The linear velocity of a wheel can be calculated as follows:

$$v_r = \omega_r \cdot R \tag{5.1}$$

v_r	Right wheel linear velocity
ω_r	Right wheel angular velocity
R	Radius

Nevertheless, before the linear velocity of the wheel can be calculated with Equation (5.1), the gear ratio of the motor needs to be considered. The reason for that is the encoder measures the motor rotation, not the wheel rotation.

The linear velocity of the robot can be derived from the average linear velocity of both wheels.

$$v = \frac{v_l + v_r}{2} \tag{5.2}$$

v	Robot's linear velocity
v_l	Left wheel linear velocity
v_r	Right wheel linear velocity

The angular velocity of the robot is determined by the difference in angular velocities of the left and right wheels. This is calculated using the formula:

$$\omega = \frac{v_r - v_l}{2} \tag{5.3}$$

ω Robot's Angular velocity
v_l Left wheel linear velocity
v_r Right wheel linear velocity

This transformation is summarized on the left side of Figure 49.

5.6.3 TwistMtoAngVel Function Block

This function has the opposite purpose of the last function: Outputing the angular velocities of each motor using the linear and angular velocities of the robot as inputs. By using equations (5.2)and (5.3), it is possible to solve for ω_l and ω_r. This will result into two equations, which can be summarized as a matrix:

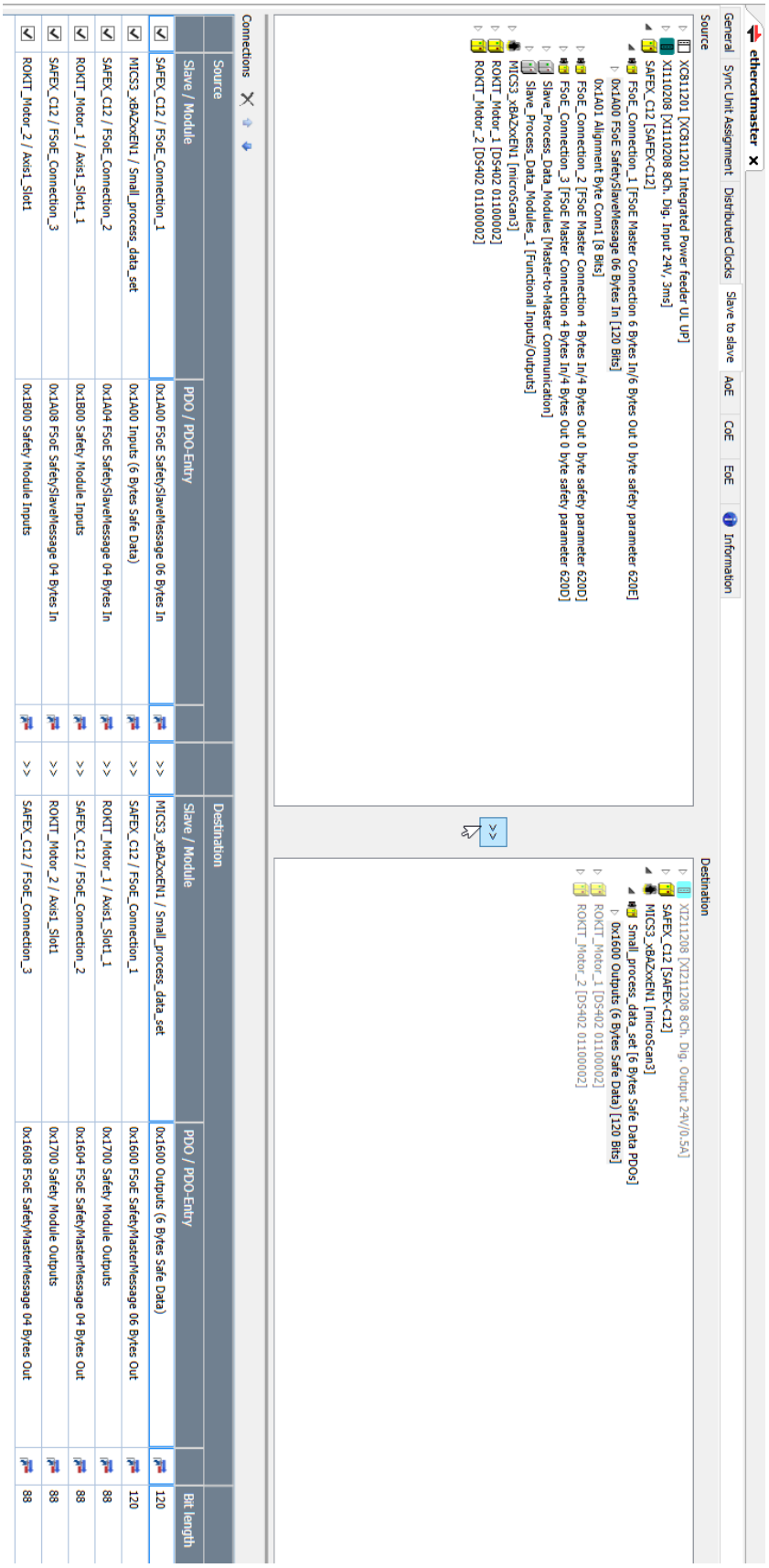
$$\begin{pmatrix} \omega_r \\ \omega_l \end{pmatrix} = \begin{pmatrix} \frac{1}{R} & \frac{d}{R} \\ \frac{1}{R} & \frac{-d}{R} \end{pmatrix} \begin{pmatrix} v \\ \omega \end{pmatrix} \tag{5.4}$$

ω_r Right wheel angular velocity
ω_l Left wheel angular velocity
R Wheel radius
d Distance to the center point between both wheels
v Robot's linear velocity
ω Robot's angular velocity

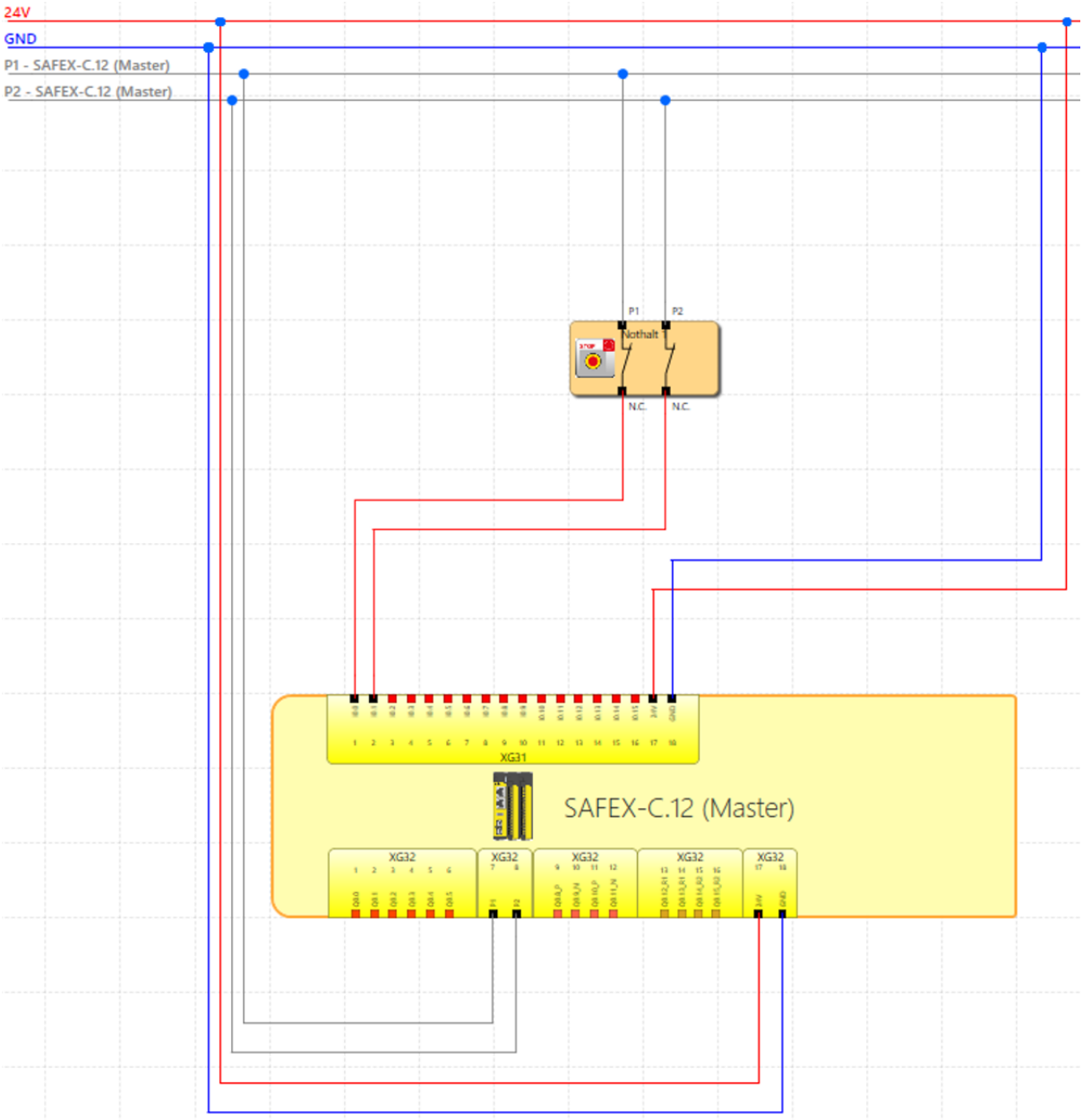
This matrix can also be seen at the bottom right of Figure 49. To send the target velocity for the motors, the gear ratio needs to be considered once more as the final step.

6 Appendix

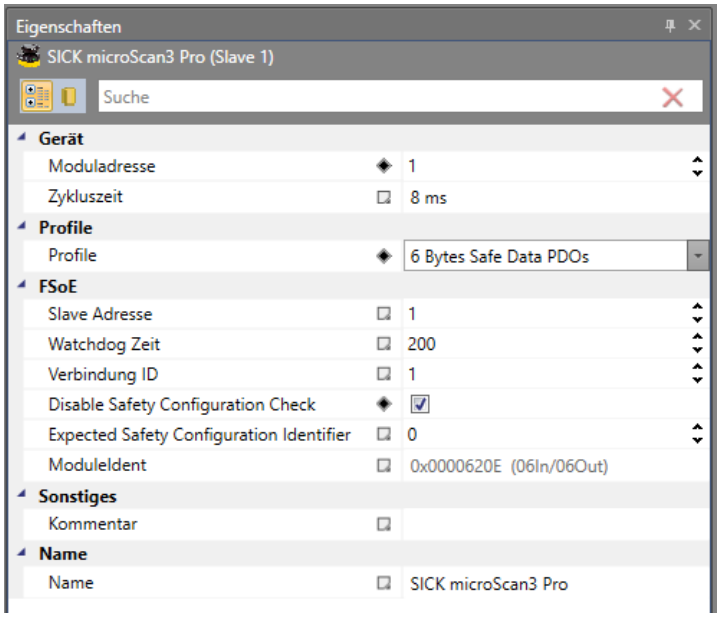
6.1 CtrlX IO Slave to Slave Configuration

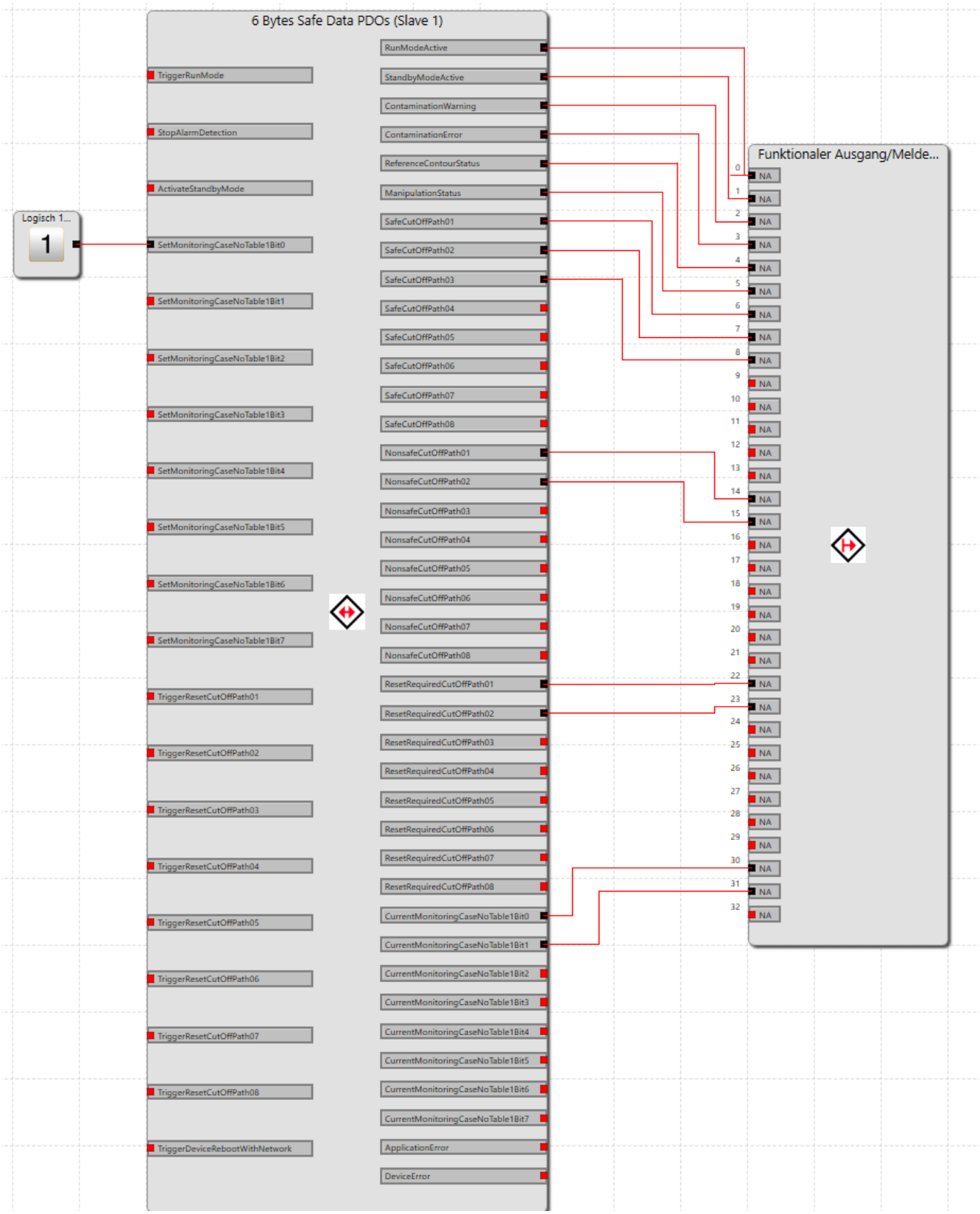


6.2 SAFETY Wiring Diagram

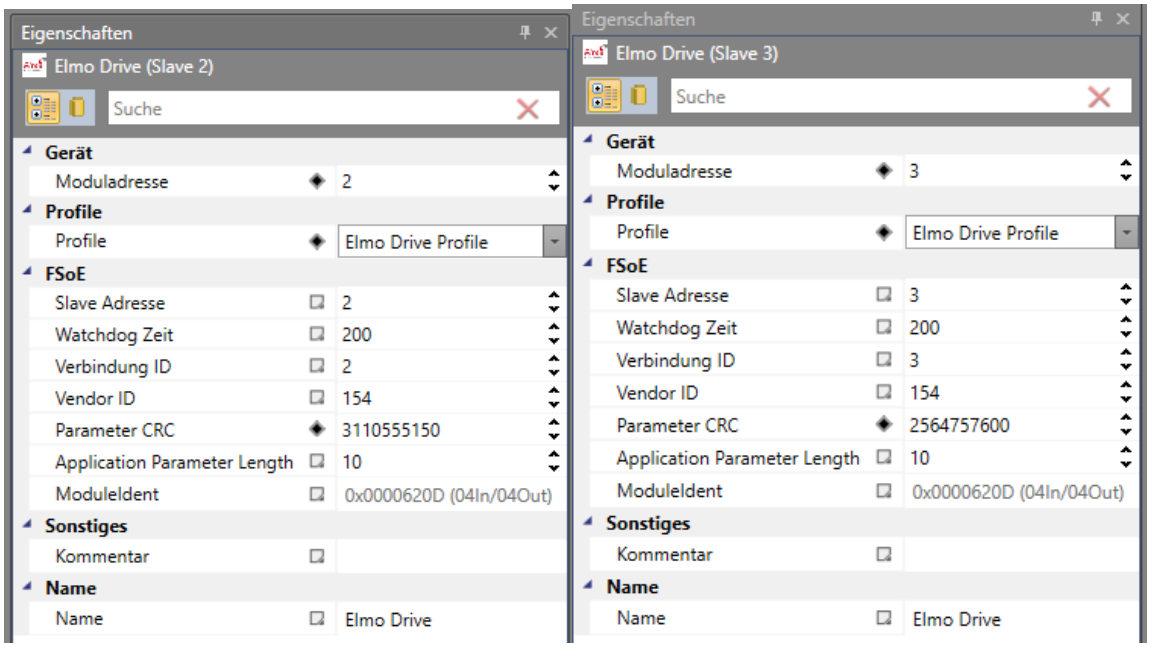


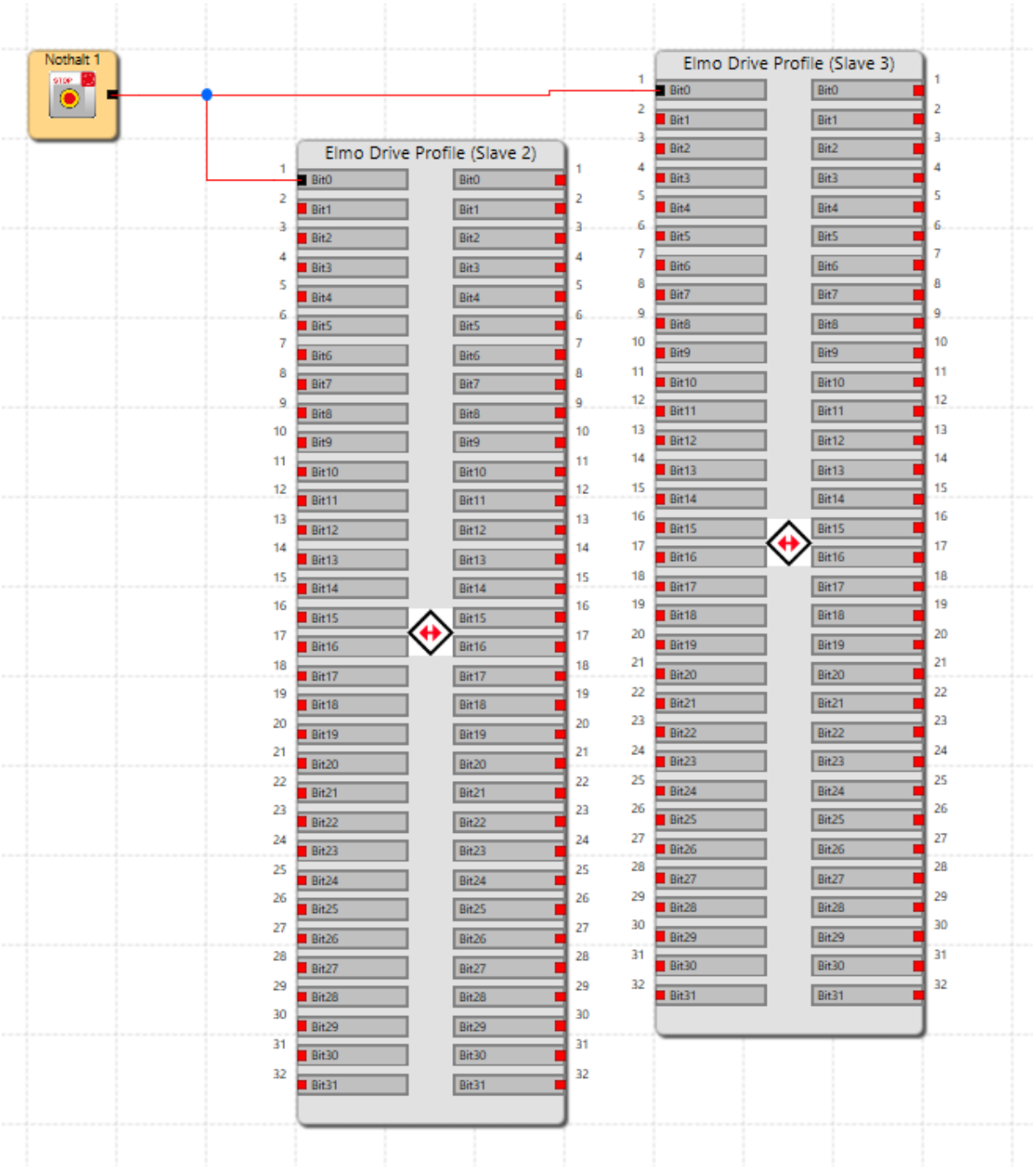
6.3 SAFETY Device Configuration SICK




6 Bytes Safe Data PDOs (Slave 1)
Logisch 1...
1
TriggerRunMode
StopAlarmDetection
ActivateStandbyMode
SetMonitoringCaseNoTable1Bit0
SetMonitoringCaseNoTable1Bit1
SetMonitoringCaseNoTable1Bit2
SetMonitoringCaseNoTable1Bit3
SetMonitoringCaseNoTable1Bit4
SetMonitoringCaseNoTable1Bit5
SetMonitoringCaseNoTable1Bit6
SetMonitoringCaseNoTable1Bit7
TriggerResetCutOffPath01
TriggerResetCutOffPath02
TriggerResetCutOffPath03
TriggerResetCutOffPath04
TriggerResetCutOffPath05
TriggerResetCutOffPath06
TriggerResetCutOffPath07
TriggerResetCutOffPath08
TriggerDeviceRebootWithNetwork
RunModeActive
StandbyModeActive
ContaminationWarning
ContaminationError
ReferenceContourStatus
ManipulationStatus
SafeCutOffPath01
SafeCutOffPath02
SafeCutOffPath03
SafeCutOffPath04
SafeCutOffPath05
SafeCutOffPath06
SafeCutOffPath07
SafeCutOffPath08
NonsafeCutOffPath01
NonsafeCutOffPath02
NonsafeCutOffPath03
NonsafeCutOffPath04
NonsafeCutOffPath05
NonsafeCutOffPath06
NonsafeCutOffPath07
NonsafeCutOffPath08
ResetRequiredCutOffPath01
ResetRequiredCutOffPath02
ResetRequiredCutOffPath03
ResetRequiredCutOffPath04
ResetRequiredCutOffPath05
ResetRequiredCutOffPath06
ResetRequiredCutOffPath07
ResetRequiredCutOffPath08
CurrentMonitoringCaseNoTable1Bit0
CurrentMonitoringCaseNoTable1Bit1
CurrentMonitoringCaseNoTable1Bit2
CurrentMonitoringCaseNoTable1Bit3
CurrentMonitoringCaseNoTable1Bit4
CurrentMonitoringCaseNoTable1Bit5
CurrentMonitoringCaseNoTable1Bit6
CurrentMonitoringCaseNoTable1Bit7
ApplicationError
DeviceError
Funktionaler Ausgang/Melde...
0 NA
1 NA
2 NA
3 NA
4 NA
5 NA
6 NA
7 NA
8 NA
9 NA
10 NA
11 NA
12 NA
13 NA
14 NA
15 NA
16 NA
17 NA
18 NA
19 NA
20 NA
21 NA
22 NA
23 NA
24 NA
25 NA
26 NA
27 NA
28 NA
29 NA
30 NA
31 NA
32 NA

6.4 Elmo Drive SAFETY Configuration




Nothalt 1
STOP
Elmo Drive Profile (Slave 2)
Elmo Drive Profile (Slave 3)
Bit0
Bit1
Bit2
Bit3
Bit4
Bit5
Bit6
Bit7
Bit8
Bit9
Bit10
Bit11
Bit12
Bit13
Bit14
Bit15
Bit16
Bit17
Bit18
Bit19
Bit20
Bit21
Bit22
Bit23
Bit24
Bit25
Bit26
Bit27
Bit28
Bit29
Bit30
Bit31