Real-time Control of Industrial Robot Cell with PowerLink

Lloyd Tran
David Barbulovic

LUND
UNIVERSITY

Department of Automatic Control

MSc Thesis
TFRT-6048
ISSN 0280-5316

Department of Automatic Control
Lund University
Box 118
SE-221 00 LUND
Sweden

© 2018 by Lloyd Tran & David Barbulovic. All rights reserved.
Printed in Sweden by Tryckeriet i E-huset
Lund 2018

Abstract

This thesis investigated how one can build a PowerLink network and Control Nodes that act as, e.g., sensor, actuators, etc., together with B&R:s products. Establishing different PowerLink networks was also investigated such as PLC connected to Raspberry Pi2. Configuration and step-by-step introduction of what PowerLink is and how it works is also mentioned in this thesis. In addition to creating PowerLink networks and nodes, this thesis also looked into implementing OpenCV for image processing with a camera on a Raspberry Pi2 connected to a B&R PLC.

PowerLink is a deterministic Ethernet-based protocol. It works by having a Managing Node that acts as the moderator for the network. Then there are the Control Nodes, which are slaves nodes in the network.
The result showed that it is possible to create control nodes that can be controlled and read through PowerLink on a PC or PLC. The PowerLink camera node that was created was implemented on Raspberry Pi2 with OpenCV. The idea behind this was to show that it is possible to easily setup a PowerLink node. One of the problems that was experienced was that the PowerLink stack and OpenCV code that was to be implemented was written in C/C++, but this was solved by establishing a Server/Client communication between the two programs. The Raspberry Pi2 limited hardware, which made it not suitable for high performance systems.

This thesis showed that it is possible to create sensors, control and directly test them from a PC or implement them on a industrial PLC. For the users it is greatly beneficial due to the fact that a control node can execute code, e.g., filter measurements, before sending data to the PLC.

Acknowledgements

We want to thank B&R and Department of Automation Control for helping us with supplies of hardware and knowledge. A special thanks to our supervisor Anders Robertsson for help and advice during the thesis period.

Acronyms

MN - Managing Node

CN - Controlled Node

I/O - Input/Output

PLC - Programmable Logic Controller

PDO - Process Data Object

PReq - PollRequest (Frame in PowerLink)

PRes - PollResponse (Frame in PowerLink)

TPDO - Transmit Process Data Object

RPDO - Receive Process Data Object

XDC - XML Device Configuration file

XDD - XML Device Description File

UINT8 - Unsigned Integer 8 bit

OpenCV - Open Source Computer Vision Library

HSV - Hue, Saturation, Value

Contents

1

Introduction

1.1 Background

To control automation equipment, including work cell coordination and configuration of e.g., industrial robots, conveyor belts, external axes and IO-units, one often uses a Programmable Logic Controller (PLC). This in turn often requires that the user has dedicated configuration programs from the manufacturer.

If the setup of sensors, IO-units or actuators are to be changed, one must program and learn about the PLC through the supplied program from the manufacturer, which may be time consuming, as it requires the user to have the program and the right operating system in order to use it. It would be more resourceful if the user could change parameters through scripting without requiring to use a specific operating system. To implement this, openPowerLink with openConfigurator will be used [PowerLink, 2017].

Library implementation may also be a limitation as it is often a bit more complex to implement, e.g., OpenCV, firmware-update, etc, in the dedicated software for a PLC. The considered robot workcell will be controlled by a PLC supplied from B&R which is an Austrian industrial automation company [B&R, 2017b].

OpenPowerLink is an open source-based deterministic real-time Ethernet protocol. It is an alternative which in contrast to the old field-bus system CAN, runs on standard Ethernet equipment and does not require different CAN buses for different tasks.

1.2 Goal

The goal for this master's thesis project is to implement a PowerLink real-time communication between a PC and a PowerLink node. The PowerLink node should also be able to work with Automation Studio and be a working device together

with, e.g., an industrial robot. The PowerLink node can be, e.g., a Raspberry Pi2 or BeagleBone hardware with IO-units, sensors or actuators. Another PowerLink node will be an motor/sensor-drive, ACOPOS 1045, from B&R, as the task for the thesis work also concerns the motor-drive and to investigate if and how it can be directly controlled from a PC or other computer units through PowerLink. The final approach will be to setup a PowerLink communication between the Managing Node and motor-drive, e.g., where a camera connected to another that CN acts as a sensor to aid in the task of picking up an object with help of a FlexPicker controlled by ACOPOS 1045.

The goal of controlling the FlexPicker is done according to Figure 1.1. The camera node sends the coordinates of the object to be picked up to a PC. This in turn sends the coordinates of the Acopos and then controls the FlexPicker. The control of FlexPicker is done through Acopos, as it controls the motors of the robot.

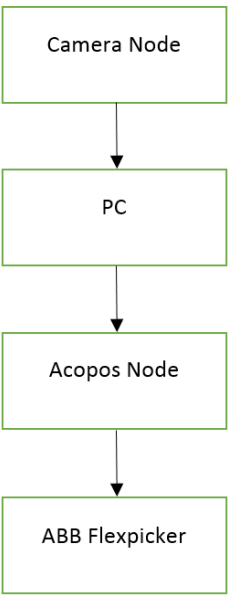

Figure 1.1 Clarification of thesis goal.

1.3 Limitations

The reason for the limitation of the scope of only creating two different kinds of PowerLink nodes, is because the thesis needs to be possible to do during the thesis

project. This thesis will also not look further into network safety, as it is outside the scope of this master thesis.

2

Theory

Understanding the problem is done by understanding the theory. In this section the theory for various subjects is described.

2.1 PowerLink

Managing- and Control Nodes

PowerLink is an Ethernet-based communication, deterministic and real-time Ethernet protocol. It is an improvement to old field-bus systems as the older technology requires different buses for different tasks [OpenPowerLink, 2017b].

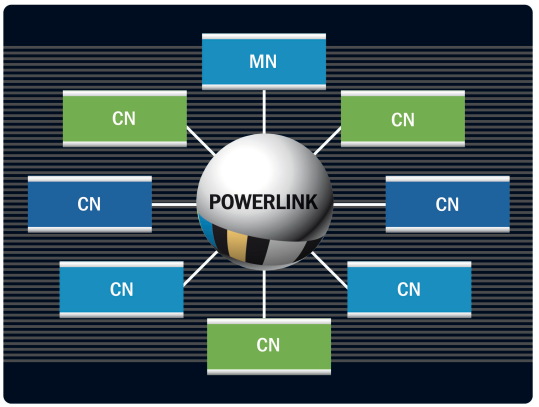

Figure 2.1 PowerLink Architecture [Ethernet.PowerLink.org, 2017a]

PowerLink works by having a Management Node (MN) and one or several Controlled Nodes (CN). The MN is the one that sets up the communication and maintains it, whereas the CN (or CN:s) receives or sends data to the MN or even to other CN:s, depending on how the PowerLink is configured. The PowerLink network

can be configured to have CN-to-CN communication without sending information through the MN first.

The Cycle

In order to achieve a deterministic network with PowerLink, timing of different procedures is required. This is done in the PowerLink cycle, which consists of two parts; the isochronous- and asynchronous phase. In the isochronous phase the real-time data exchange between the nodes appears within the cycle, whereas the asynchronous phase starts after the isochronous phase. It is through the asynchronous phase "hot plugging" is made possible, which allows adding of nodes without affecting the real-time behavior.

These phases are visualized in Figure 2.2. As the figure shows, a lot of things happen. In order to simplify the understanding of what is happening during a PowerLink cycle, a step-by-step procedure is described.

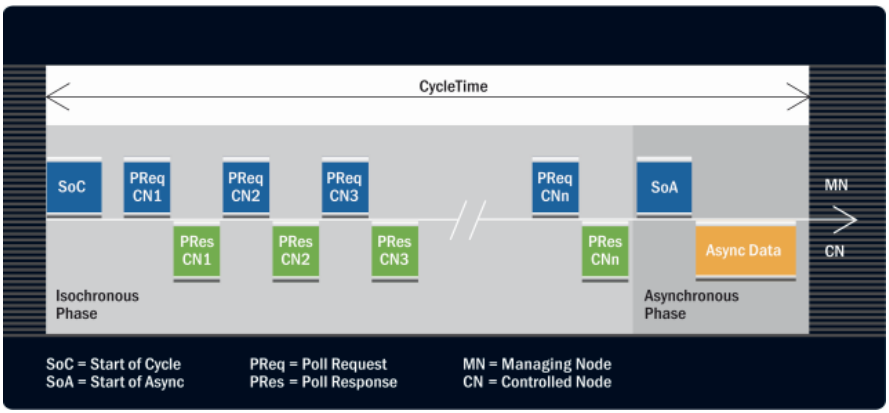

Figure 2.2 PowerLink Cycle [Ethernet.PowerLink.org, 2017b]

The MN initiates the communication by sending a SoC, Start of Cycle. Here the MN synchronizes with the CN:s. Thereafter the synchronization is completed and the isochronous phase starts. A PReq, Poll Request, is sent from the MN to the address of the CN, which includes data. The CN that has received the Poll Request from the MN, responds with PRes, Poll Response, and real-time data. This is then repeated for all registered and connected CN.

The MN starts next with sending SoA, Start of Asynchronous, which ends the isochronous phase and starts the asynchronous phase. This makes it possible to

send non time-critical data. The CN or the MN sends an Async Data, Asynchronous Send, where the asynchronous data is transported [Ethernet.PowerLink.org, 2017b].

Node Addressing

In order for the PowerLink communication to be established, nodes for the network need to be addressed. This is done by setting a unique 8-bit node ID. The range of ID 1-239 are reserved for the PowerLink CNs, while the ID 240 is reserved for the PowerLink MN. Address 0 is an invalid address, the rest of the ID:s are reserved for other purposes, e.g., diagnostic node, dummy node etc.,[OpenPowerLink, 2017d].

Process Data Objects

In a network, Process Data Objects (PDO) can hold information of , e.g., a temperature from a sensor. The node that transmits the information, sends it by using Transmit PDO (TPDO), while the receiving node registers the received PDO as an Receive PDO (RPDO) [Ethernet.PowerLink.org, 2017c].

As stated before, the PowerLink network has two parts, the iscohronous and the asynchronous. The PDO communication appears in the ischocronous part in the PowerLink network. The frames that are used are PollRequest (Preq) and PollResponse (Pres), which transmits the PDO information.

Determining the Minimum Cycle Time

A PowerLinks network minimum cycle time can be determined as described below [OpenPowerLink, 2017a].

Each frame in the network has the following size:

$$1\ SoC = 64\ Bytes$$

$$x\ PReq/Pres = x \cdot (64 - 1518)\ Bytes$$

$$1\ SoA = 64\ Bytes$$

$$1\ ASnd = 318\text{-}15185\ Bytes$$

The calculation of the time for the transmission is determined by the total amount of bits on the network, which are given by:

$$Total\ bits = Ethernet\ frame\ size\ in\ Bytes \cdot 8 + 64\ bit$$

With the number of total bits, calculating the minimum cycle time can thus be calculated as described below.

$$t_{cycle} = Bits\ on\ the\ network \cdot 10ns + Reaction\ time\ between\ all\ frames$$

The reaction time is 960ns, which is the inter-frame gap between all frames, although there is also a more generalized formula that can be used [B&R, 2015]:

$$t_{cycle} = t_{SoC} + t_{IPG} + t_{SoA} + t_{IPG} + t_{ASnd} + t_{IPG} + \sum_{MN,CNs} t_{PDO.frame.including.IPG} + \sum_{CNs} t_{hub.level}$$

Here the Inter-packet gap (IPG) is the same as the reaction time, which is 960ns. The hub level indicates the number of hubs between the MN and the CN, and the time added per hub level equals 3μs.

2.2 Hardware

B&R PLC

The Programmable Logic Controller is mainly a computer, where different tasks can be implemented, such as computation, control etc. Thanks to this, the PLC device is used extensively throughout the automation industry. The PLC device that is used is an X20 model from B&R, which allows it to expand the module with digital or analog I/O:s. One should note that the different models of the X20 PLC:s may vary significantly in capacity but have no differences visually.

Figure 2.3 The X20CP1483 PLC [B&R, 2017d]

X20CP1483 PLC This kind of PLC B&R was used in this thesis project. It has two Ethernet ports, one for the standard Ethernet and one Ethernet PowerLink port, EPL, which is needed in order to connect to the PLC through PowerLink.

B&R Bus

The B&R X20BC1483 Bus is much smaller than the PLC, but still supports external I/O:s to extend the Bus, such as digital and analog I/O:s. The X20BC1483 also supports PowerLink.

Figure 2.4 The X20BC0083 Bus [B&R, 2017c]

B&Rs Acopos 1045

Acopos 1045 servo drive is a motion controller. It gives the user the possibility to accurately control the motor position, velocity, torque etc. It is equipped with an Ethernet interface, which enables the possibility for connecting it to a PowerLink network.

Figure 2.5 Acopos 1045 [B&R, 2017a]

Raspberry Pi2

Raspberry Pi2 can be looked at as a very small computer. The board itself contains four USB ports, HDMI ports, micro SD card holder, a 900MHz quad-core ARM Cortex-A7 CPU, 1 GB RAM, 3.5mm audio jack, Camera and Display Interface [Raspberrypi, 2017a].

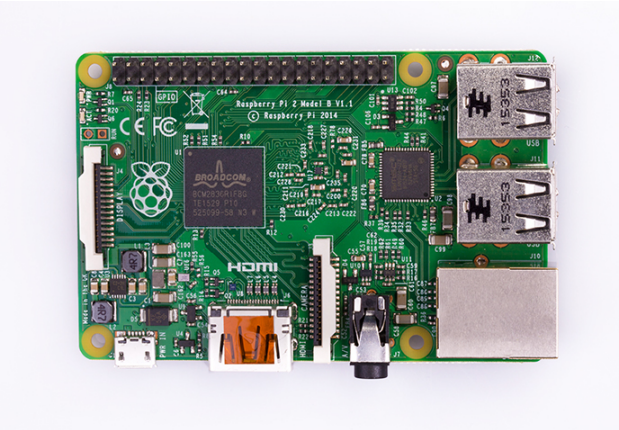

Figure 2.6 The Raspberry Pi2 board [Raspberrypi, 2017b]

BeagleBone Black

The BeagleBone Black board has similar functionality as the Raspberry Pi2 board, though the hardware is different. It consists of 512MB DDR3 RAM 800MHz, AM335x 1GHz processor, HDMI output, and 92 pin headers.

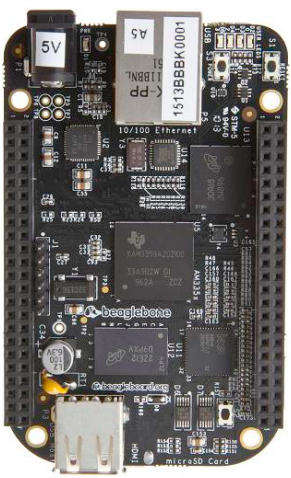

Figure 2.7 The BeagleBone black board [BeagleBone, 2017]

2.3 Software

Operating system

In this master thesis the following operating systems have been used:

- Microsoft Windows 7
- Microsoft Windows 10
- Raspbian jessie/Ubuntu
- Fedora

C/C++ & Cmake

C is a computer programming language that supports structured programming. C was originally developed to UNIX operating system [techopedia, 2017b].

C++ is the newer language, based on C, but have many more features such as e.g., object-oriented and generic programming. When a projects is starting to grow bigger then the cmake is coming in handy.

Cmake is a cross-platform build system designed to generate native build environment for many operating systems like Windows and UNIX/linux. Cmakelist.txt is a configuration file that is placed in a directory and subdirectory to generate build files [techopedia, 2017a].

When running Cmake with the configuration files it locates files, libraries and executables. For a project with multiple toolkits where each toolkit may contain several directories, Cmake will also be able to support the order of the run, e.g., executables must be built before the code is compiled and linked to final application. In other words Cmake is designed to support complex projects build with directory hierarchies and applications dependent on several libraries. Also, an advanced user can even build a makefile for a particular compiler and OS combination [Cmake, 2017].

Notepad++

Notepad++ is both a text editor and a source code editor for Microsoft Windows. Unlike the standard Notepad in Microsoft Windows, Notepad++ has more features, it supports syntax highlighting, code folding and tabbed editing. Code folding allows user to "hide" or "fold" the code user has edited and show only the subsection name and tabbed editing grants the user the possibility to open multiple files in a single window. Notepad++ is written in C++ and is a free software for any user to download [Notepad++, 2017].

Visual Studio

Visual Studio is an advanced integrated development environment (IDE) from Microsoft. It is used to develop programs for Microsoft Windows and can also be used to develop web sites, web services and web applications. Visual Studio supports many different programming languages, where the built-in languages are Visual Basic, C, C#, C++, J# and F#. Other languages are also supported in Visual Studio but has to be installed separately. Visual studio contains features like code editor, debugger and designer [Microsoft_Visual_studio, 2017].

Code editor is a feature in that supports syntax highlighting to provide feedback about compilation error with red wavy underline or green wavy underline for warning. It also supports code completion, it provides so-called visual tab-completion functionality for the user. It is done by compiling in the background [Microsoft_Visual_studio, 2017].

Debugger allows the user to set breakpoints and when the condition is met the code will be stall at the breakpoint. The user can either "step over" to run one code line at a time or the user can also "step into" a function to debug inside. While debugging the user can also inspect the value of variables by hovering the mouse pointer on top of the variable [Microsoft_Visual_studio, 2017].

Visual Studio supports design for graphical user interface (GUI) for end-users. It is built on Windows Form where the designer can drag and drop on the form to create the layout. There are many different designer tools depending on the application, e.g., Windows Form Designer for C# or visual basic, another designer are web designer for web application [Microsoft_Visual_studio, 2017].

Automation Studio

Automation Studio, developed by B&R, is a software used to program controllers, drives but also for visualization of such as HMIs. Automation Studio reminds a little bit about Visual Studio in terms of allowing the user to debug and generate data files. It also has support for different programming languages like Ladder

diagram (LD), Function Block Diagram (FBD), Sequential Functions Chart (SFC) and Structured Text (ST). Automation Studio also supports object-oriented programming in C++, ANSI C and Matlab/Simulink for simulation. The version of Automations Studio that was used during the thesis was 4.2.5 [B&R, 2008].

XML Device Description/Configuration-file (XDD-file)

An XDD or XDC file is one of the pillars of successfully building a PowerLink network. This kind of file can be described as a map of the I/O:s the device has. It contains information regarding what kind of information that should be received from the MN (Receive Process Data Object,RPDO) or transmitted (Transmit Process Data Object, TPDO). The difference between an XDD and an XDC file is very small; the XDD file has default values on its I/O:s while an XDC file has preset values. In the Listings 2.1 and 2.2, an example is given of what an XDD file contains. Please note that the XDD example below has only the described I/O:s, as it contains a lot of information which might be unnecessary for the reader.

```
<Object index="6000" name="DigitalInput_00h_AU8"
objectType="8" dataType="0005">

  <SubObject subIndex="00" name="NumberOfEntries"
   objectType="7" dataType="0005" accessType="const"
   defaultValue="4" PDOmapping="no"/>

  <SubObject subIndex="01" name="DigitalInput"
   objectType="7" dataType="0005"
   accessType="ro" PDOmapping="TPDO"/>

  <SubObject subIndex="02" name="DigitalInput"
   objectType="7" dataType="0005"
   accessType="ro" PDOmapping="TPDO"/>

  <SubObject subIndex="03" name="DigitalInput"
   objectType="7" dataType="0005"
   accessType="ro" PDOmapping="TPDO"/>

  <SubObject subIndex="04" name="DigitalInput"
   objectType="7" dataType="0005"
   accessType="ro" PDOmapping="TPDO"/>

</Object>
```

Listing 2.1 Example of input descriptions in an XDD-file

```
<Object index="6200" name="DigitalOutput_00h_AU8"
objectType="8" dataType="0005">

  <SubObject subIndex="00" name="NumberOfEntries"
  objectType="7" dataType="0005" accessType="const"
  defaultValue="4" PDOmapping="no"/>

  <SubObject subIndex="01" name="DigitalOutput"
  objectType="7" dataType="0005"
  accessType="rw" PDOmapping="RPDO"/>

  <SubObject subIndex="02" name="DigitalOutput"
  objectType="7" dataType="0005"
  accessType="rw" PDOmapping="RPDO"/>

  <SubObject subIndex="03" name="DigitalOutput"
  objectType="7" dataType="0005"
  accessType="rw" PDOmapping="RPDO"/>

  <SubObject subIndex="04" name="DigitalOutput"
  objectType="7" dataType="0005"
  accessType="rw" PDOmapping="RPDO"/>

 </Object>
```

Listing 2.2 Example of output descriptions in an XDD-file

The object index is an identifier of the object, which is given in hex (16-bit) and name is just the string name of the object. Depending of what the *objectType* has for value, it can have different meanings. If the *objectType* value is 7, this indicates that it is a variable, number 8 indicates an array of the same data type and number 9 says that the object type is a record, which means the values can have different data types. The line that says *dataType* describes what kind of data type the object/sub-object has. In the Listings 2.1 and 2.2, the data types are set to *0005*, which is unsigned8 [OpenPowerlink, 2017a].

The *accessType* line decides what kind of privileges the object should have. There are four types of privileges; *const* means that the object can only be read from the network node side, the values can not be changed during runtime and that the value is predefined. The *ro* means read-only, *wo* is write-only and lastly *rw* defines that the object has the read-write privileges.

PDOmapping describes how the information exchange is going to happen. There are five possible PDO-mapping, but only three will be mentioned in this thesis. The first is *no*, which basically says that the object can not be PDO-mapped. Next one is *RPDO*, which as explained before, indicates that the information is going to be received from the node and lastly *TPDO* which is information sent to the node.

Eclipse

Eclipse is an development environment for programming and uses mainly Java IDK. Eclipse also supports other programming language like C++, C etc. but only via plugins. It offers a range of plugins for the program, including openConfigurator which is used to build a PowerLink network [Eclipse, 2017] . Eclipse is compatible with and can be installed and run on many operating system.

openConfigurator

openConfigurator is an Eclipse plugin, and is one of the stepping stones of successfully building a PowerLink network. The plugin itself has the functionality to create a PowerLink Network by using XDD/XDC files. Creating a PowerLink network can be cumbersome if done from scratch, but nevertheless, it is pretty much straight forward by using the following recipe.

In order to create the PowerLink Network, first one must create the MN for the system, which is done aside with choosing the XDD-file. Note that the XDD-file must be written for an MN, as the corresponding file contains information if the node is a MN or CN. Next step is to add a CN, which is also done by choosing the XDD-file which holds the information about the CN. The cycle time for the network can also be chosen, but note that a too short cycle time for the PowerLink network will result in a network that would not be able to setup the communication as specifications will be violated [OpenPowerlink, 2017b].

When the MN and CN/CN:s have been added to the openConfiguration, Eclipse will generate four files and these are:

mnobd.cdc: The file contains the configuration data of the PowerLink network. This includes configuration data for the MN and CN/CN:s. The configuration of a CN is done by the configuration manager (CFM), which is part of the MN.

mnobd.txt: This file is a text-file description of mnobd.cdc, which contains all the MN and CN configuration data. This file in not crucial for the creation of a PowerLink network, as the file primarily is used for diagnostic reasons.

xap.xml: This type of file is an XML-file, and it contains information of the process image.

xap.h: The xap.h header file is one of the more interesting files. It includes the structure definition of the process image in the form of two ANSI C structures. The header file can be directly included in the openPowerLink stack and runs the desired network.

The final thing that should be mentioned, is that a PowerLink network can not have more than one active MN in the network, as it violates the idea of PowerLink, but it is possible to have one or several redundant MNs in the network. A redundant MN makes it possible to have one or more MNs in the network, put only one of them will act as a MN in the network. If, e.g., the connection to the original MN is lost, then the redudant MN will step in and control the network and thus the network can continue to operate.

The openPowerLink Source Files

As mentioned before, openPowerLink is open-source and can be implemented on the Windows or Linux operating system. For building a PowerLink network, the source files are the cornerstone for successfully building and managing a stable PowerLink network, since the stack includes the files for building and coding. These files can be directly downloaded from openPowerLink's webpage [OpenPowerLink, 2017b]. Note that there are older and newer versions of the source files and the information about differences can be acquired from openPowerLink's webpage. Libraries are built from the PowerLink stack. How it is done will be mentioned in the Sec. 3.4. The version of stack files that are used in the thesis project is version 2.5.1.

Win32 Disk Imager

The Win32 Disk Imager program writes a disk image, e.g., the operating system for a Raspberry Pi2, to a memory storage device. It can also create a backup for a removable device to an image file. It is also open software [Sourceforge, 2017].

OpenCV

Open Source Computer Vision Library, OpenCV, is open source software for computer vision. It supports a number of programing languages, such as Python, C++, C, Java and Matlab. It also has support for Linux, Windows, Mac OS and the Android operating system [OpenCV, 2017a].

In the OpenCV program *HSV* (*Hue*, *Saturation* and *Value*) is the pixel representation which is most commonly used to detect objects. *Hue* determines the colour, eg. *0* corresponds to colour red, etc. *Saturation* is the amount of grey and *Value* describes the intensity (brightness) of the colour [ThoughtCo, 2017].

Wireshark

The Wireshark software is open-software and is used for analyzing network data packages from the device that is transmitted to or sent from inside the network, as long that the devices supports Ethernet communication or PowerLink. It is also able to either in online or offline mode analyze a network. Other features are filters for the package and able to analyze USB, Bluetooth and many others protocols. Identification of what kind of packages (TCP, UDP etc.) and what kind of information is sent can be visualized in the software.

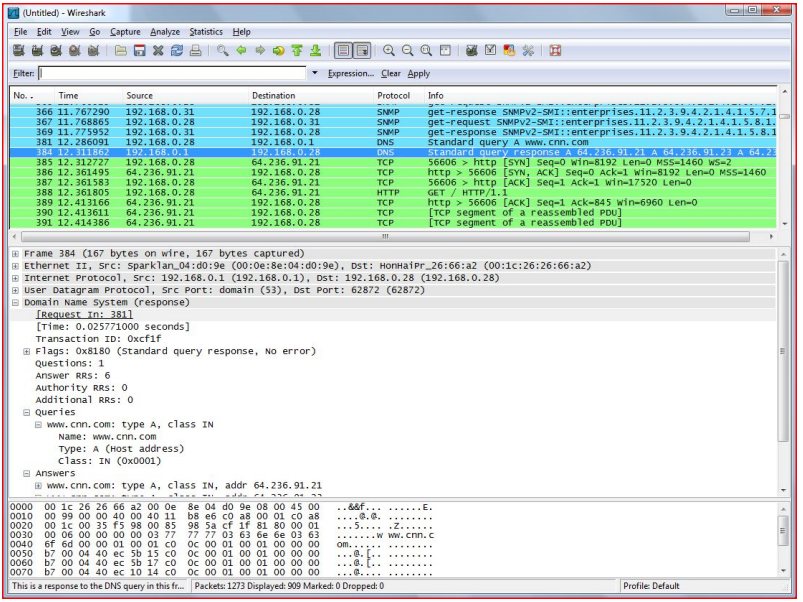

Figure 2.8 An example of a Wireshark network capture [WhireShark, 2017]

2.4 Ethernet Communication and Socketing

For sending information from one program to another, socketing is used. By using socketing it does not affect the functionality of the program and requires no major

changes in the code of the programs. The idea is to create a local Ethernet server and client network where the information can be sent, where one program will act as a server while the other one will act as a client. In this thesis, the server will recieve the information while the client sends it. When creating an Ethernet local network, it is possible to use either TCP or UDP communication protocol. The main difference between TCP and UDP is that with TCP there is handshaking and a guarantee that the data will arrive at the desired destination, whereas UDP has no form of handshaking and no guarantee of delivery.

2.5 Threading and Mutex

The main use of threads is to allow a program to do more than one thing in parallel, e.g., read and write. This allows the program to handle multiple tasks. Let us say that there are two threads, the first thread can read and the second thread can write. This allows for the program to read from a file and write to another. But if the threads are to use shared resources, e.g., read and write to the same file, mutex must be used. The reason is that there will be a guarantee that the second thread will have the ability to write to the file with updated values, thus the first thread can read the updated value and not an old value. Therefore mutex is used and this works as "wait" function. If one thread uses a resource it will be locked, then the other threads cannot use it. The other threads will have to wait for the resource to be unlocked in order to use it [randu.org, 2017].

3

Method

This chapter is divided into two parts. The first part explains how a PowerLink network is built while the second part explains the implementation of different PowerLink networks.

3.1 The steps of building a PowerLink Network

Building a PowerLink network can be described in three steps:

Step 1: Download the desired PowerLink stack and choose version of the stack. Note that older versions can have some limitations, so it is recommended to download the latest version. The stack can be downloaded at openPowerLink's webpage [OpenPowerLink, 2017b].

Step 2: Build the Libraries.

Step 3: Configure and run the network

These steps can be a bit hard for a newcomer in the PowerLink area, therefore a detailed explanation of how to do will be explained in this chapter.

3.2 Contents of the PowerLink stack

The first step is to install necessary software and create paths for the executable program. However before establishing a PowerLink network, some steps are required to be taken before the creation can be done.

There are many files in the PowerLink stack, but for building and maintaining a PowerLink network, the most interesting files reside in the folder called *apps*. In the contents of *apps*, there are additional folders, which are *commons*,

demo_cn_console, *demo_mn_console* and some more which will not be mentioned as these are not altered by the user. All the files can be easily opened with Notepad++ when using Microsoft Windows.

The following describes what the mentioned folders contain.

The Folder *Commons*

Inside the *commons* folder there is the *objdicts* folder, that contains the XDD-files for an MN and CN. For those who lack the experience of building an XDD-file from scratch, it is recommended to use the stack's XDD-files as templates, as those contain descriptions of digital input, digital output, analog input and analog output. Note that the data type of the digital I/O:s are UNSIGNED8 and not float numbers.

To program a CN to send, e.g., a float number or adding digital inputs, some command lines need to be altered/added in the the file *objdict.h*. These files reminds very much of the XDD-file for the CN. This file is what the CN uses when initializing the console for the CN. With the file, turning a CN device to a senor can be done. How the programming is done is described in Section 3.14, *PowerLink with B&R PLC MN and Raspberry Pi2 CN*.

Demo_mn_console and Demo_cn_console folders

The MN folder contains the *build* and *src* folders and a detailed explanation of what these folders contains will be now explained.

build The build folder contains two other folders that are named *linux* and *windows*. The executing program to start a PowerLink network are inside the *linux* and *windows* folder, but one can only successfully execute the program if the operating system is right.

If the PC has Linux operating system, then the execution files that should be used are inside the folder *linux* and if the PC has Windows operating system the build files that are to be used is inside the folder *windows*. A more detailed explanation on how to build a PowerLink network and run it, is mentioned in Sections 3.3 and 3.4.

src The demo_mn_console folder contains the following textfiles, *apps.c*, *apps.h*, *apps.c*, *event.c*, *event.h*, *main.c* and *xap.c*. However for programming and identifying inputs and outputs, the files that are interesting are *apps.c* and *xap.c* but the demo_cn_console folder does not have the *xap.c*.

For programming the PowerLink network, this is done in *apps.c* and how to do so will be further explained in the next subsection *Programming the Network*. The

other textfile, which is one of the most important files is the *xap.c*, as this contains what inputs and outputs the network has and is used in *apps.c*. The XDD-file described inputs and outputs can be looked in *xap.c*, if the generation of the *xap.c* has correctly been done that is. How to generate the *xap.c* is described in section 3.6, PowerLink with PC and B&R:s X20BC1483.

Programming the Network

The file *main.c* will start to initialize the system like PowerLink configuration, event handler, IP, Node address etc. It also detects all the possible Ethernet cards for PowerLink communication. Once it finishes it enters the *loopmain()* method and will react to the commands that have been sent from keyboard, "r" will reset the system, "c" will restart the cycle of the system and ESC will exit the system. The last thing *loopmain()* does is to call the method *processSync()*, which is where the *app.c* starts, before it loops again.

The file *event.c* is doing as the name implies; it handles the event for the system. It also processes the history of the events. The event handler provides the interface for posting events to other modules in the system.

In *app.c* is where the variable can be sent and received between MN and CN. In the function *initProcessImage()* a user can link an input or an output to be sent between an MN and a CN. The linking is only possible if the XDD-file has the correct configuration. The linking procedure is done as in the following:

```
obdSize = sizeof(pProcessImageIn_1->digitalIn);
varEntries = 1;
ret = oplk_linkProcessImageObject(0x6000,
                                  0x01,
                                  offsetof(PI_IN, digitalIn),
                                  FALSE,
                                  obdSize,
                                  &varEntries);
```

Listing 3.1 Link a digital input to an object directory

When the link procedure is complete, the system is ready to send and receive data, which is done in method *processSync()*. The following code will show an example:

```
/* read input image - digital outputs */
digitalOut_1 = pProcessImageOut_1->digitalOut;

/* setup output image - digital inputs */
pProcessImageIn_1->digitalIn = digitalIn_1;
```

Listing 3.2 Send and recieve data on CN app.c

The variable digitalOut_1 will receive data from pProcessImageOut_1->digitalOut (the other node) and digitalIn_1 will send data to pProcessImageIn_1->digitalIn (the other node).

3.3 Installing libpcap

The packet capture library, libpcap, enables the Raspberry Pi2/BeagleBone Black node to capture live network data [Wireshark, 2017]. If this is not installed or no form of libpcap exists on the system, then when building the console for the CN, it will give an error.

The following command is used to install libpcap in the Linux environment. To use this command, connection to the Internet has to be provided.

```
sudo apt-get install libpcap-dev
```

Listing 3.3 The command for installing libpcap in Linux environment

3.4 Building the Libraries

Building the PowerLink stack libraries is necessary since these files are crucial for successfully building a network. The libraries consist of two parts, the *Release* and *Debug* files. From the PowerLink stack protocol, the following commands can be found in the PowerLink stack documentation [OpenPowerLink, 2017c].

Note that when running on Windows operating system, Visual Studio has to be installed. Before building the libraries the following command needs to be run.

```
vcvars32
```

Listing 3.4 Presteps before building libraries on a Windows system.

The following commands build the libraries on a Linux operating system.

```
> cd <openPOWERLINK_dir>/stack/build/linux
> cmake -DCMAKE_BUILD_TYPE=Debug ../..
> make
> make install
```

Listing 3.5 Installing Debug libraries on Linux operating system

```
> cd <openPOWERLINK_dir>/stack/build/linux
> cmake -DCMAKE_BUILD_TYPE=Release ../..
> make
> make install
```

Listing 3.6 Installing Release libraries on Linux operating system

The commands for a Microsoft Windows operating system are following.

```
> cd <openPOWERLINK_dir>\stack\build\windows
> cmake -G"NMake Makefiles" -DCMAKE_BUILD_TYPE=Debug
        ..\..
> nmake
> nmake install
```

Listing 3.7 Installing Debug libraries on Windows operating system

```
> cd <openPOWERLINK_dir>\stack\build\windows
> cmake -G"NMake Makefiles" -DCMAKE_BUILD_TYPE=Release
        ..\..
> nmake
> nmake install
```

Listing 3.8 Installing Release libraries on Windows operating system

Please note that if one of the libraries, Release or Debug, is missing, this could lead to the error OPLKLIB-NOTFOUND or OPLKLIB_DEBUG-NOTFOUND when trying to build a PowerLink network.

3.5 Building the PowerLink Stack

The stack documents from openPowerLink describe how to build the demo for PowerLink. Nevertheless, this is the same commands that are used to build the network. The following commands are used to build the execution files on a Linux operating system.

```
> cd <openPOWERLINK_dir>/apps/<demo_dir>/build/linux
> cmake ../..
> make
> make install
```

Listing 3.9 Building the PowerLink execution files on Linux operating system

Here are the commands on a Windows operating system.

```
> cd <openPOWERLINK_dir>\apps\<demo_dir>\build\windows
> cmake –G"NMake Makefiles" ..\..
> nmake
> nmake install
```

Listing 3.10 Building the PowerLink execution files on Windows operating system

3.6 Running the Network

The PowerLink network can easily be setup in a Windows environment by just using the following command.

```
(Name_Of_Execution_File).exe
```

Listing 3.11 Running the PowerLink execution files on Windows operating system

To run the PowerLink network on a Linux operating system, the following command can be used.

```
./(Name Of Execution file)
```

Listing 3.12 Running the PowerLink execution files on Linux operating system

When running the program in Windows or Linux, a list of network cards will appear that are on the PC. The user has to identify which network that runs the PowerLink communication and this can be done by disabling the network one at a time and identify where the PowerLink communication will be. When running the console on a Linux machine, the Ethernet card to choose is *eth0*. It will run the PowerLink network on this Ethernet card by default.

Something that is very important to keep in mind, is that when making changes to the PowerLink network, e.g., adding features through programming, and then running it, the old code could still run. To run the newer code, follow these steps.

1. Start by deleting all the build files from build folder, but don't remove the files that came with the stack.
2. Then rebuild the new code, as described in Section 3.4.
3. Run it.

These steps can be looked at as compiling the code. An example of how it looks when initializing a PowerLink network on a Windows Operating system is given below by Figure 3.1.

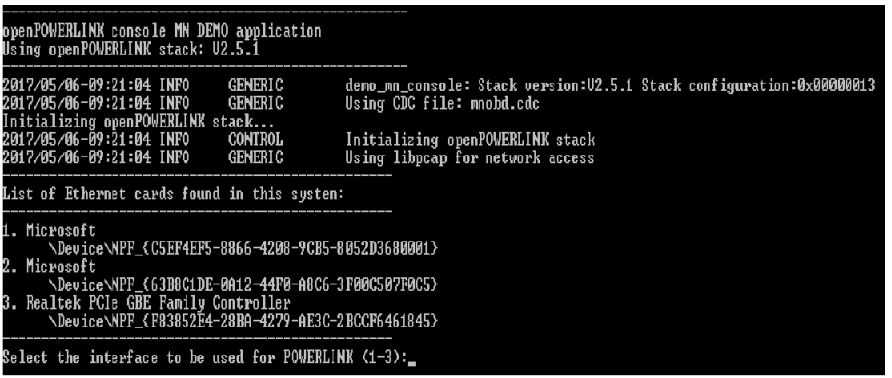

Figure 3.1 The interface of PowerLink on a Windows PC

3.7 Installing OpenCV on Raspberry Pi2

For installing the OpenCV program, the operating system *Raspian for Jessie with Pixel* was downloaded from http://raspberrypi.org and installed. The reason for it was that the tutorial for installing OpenCV on Raspberry Pi2 chose *Jessie*. To ensure that OpenCV would run smoothly, the operating system was updated and upgraded with the following commands. Note that the command *reboot* is done as it is required after the firmware update [Rosebrock, 2017]. For the commands to work, the Raspberry Pi2 needs to be connected to the Internet.

```
sudo apt-get update
sudo apt-get upgrade
sudo rpi-update
sudo reboot
```

Listing 3.13 Updating and upgrading the Raspberry Pi2 board

When the board is up to date, some packages may need to be installed; these are, e.g., *GCC 4.4.x*, *CMake 2.6* or newer version, Git, etc. These can all be downloaded with the following commands [OpenCV, 2017c].

```
sudo apt-get install build-essential
sudo apt-get install cmake git libgtk2.0-dev pkg-config
        libavcodec-dev libavformat-dev libswscale-dev
```

Listing 3.14 Installing the required packages for OpenCV

OpenCV can be built from source using cmake, although this requires some steps. An example of how the setup looks like is given below.

```
cd "path to the directory"/opencv
mkdir release
cd release
cmake -D CMAKE_BUILD_TYPE=RELEASE
-D CMAKE_INSTALL_PREFIX=/usr/local ..
```

Listing 3.15 Installing the required files

The last step is to install the run file which is done with the following command.

```
make
sudo make install
```

Listing 3.16 Install the run file

3.8 Creating Raspberry Pi2/Beaglebone Black CN

Before creating a fully functional PowerLink node, it requires initial steps to be taken. First a stable operating system has to be selected. For Raspberry Pi2, the operating system can be downloaded in form of an Ubuntu image from their web page regarding their Raspberry Pi2 tutorial, [Kalycito, 2017a]. For the BeagleBone Black, the manufacture website also gives options of different operating systems. The one that was selected was a Debian image called *Jessie* for BeagleBone Black.

To install the desired operating system on the Raspberry/BeagleBone, a microSD card and a microSD card reader is required. Together with the program Win32 Disk Imager, the image file for the operating system can be written into the microSD card from an e.g., a Windows PC. When Win32 Disk Imager is done with writing the image, take the microSD card and put it into the board. If the screen to the board and the powersupply is connected, then the operating system should boot up and

can be observed on the screen.

If the board successfully boots up, then *cmake* has to be installed. This can either be done my letting the board install *cmake* from the net or download it from cmake's webpage.

The last part is to make the board a CN, which requires that the PowerLink stack is downloaded on a usb memory card and then mount it into the usb port of the board. Then the files have to be copied from the usb to the file system on the microSD-card. The easiest way is to make a directory and copy the files from the usb onto it. After this everything is setup. The board can now be built, programmed and run with the commands previously described in the different sections above.

3.9 PowerLink with one PC as MN and another PC as CN

On the openPowerLink website one can find a pre-built system to connect two computers (Linux or Windows) using PowerLink. In this section two Windows PC:s are used in order to get a better understanding of how PowerLink works.

The setup works like this: one PC acts as MN and the other as CN. To establish this kind of PowerLink network between two PC:s requires an Ethernet cable to be connected to the network interface card, on each PC. An illustration is given in Figure 3.2.

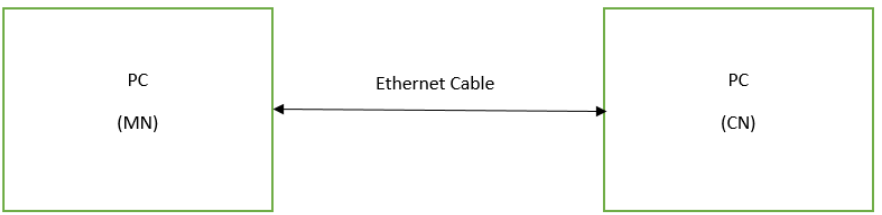

Figure 3.2 Illustration of connecting a PC to PC communication (no router/switch required)

3.10 Starting node, order of start

Starting the nodes in a particular order is not required. If one chooses to start the CN first, it will wait for the MN to start. The PowerLink communication will not start unless the MN has started.

3.11 PowerLink with PC MN and B&R:s X20BC1483 CN and openConfigurator

From the B&R webpage, it is possible to download XDD-files for the individual modules of the Bus [B&R, 2017e]. The setup is almost the same as with PC communication. The Ethernet cable runs from the PC that acts as the MN and then to the X20BC1483, which is a CN. Note that having the other way around, with the Bus as MN and PC as CN is not possible, as the Bus is just a machine with inputs and outputs, in other words a controllable device. In this case openConfigurator has to be used, as the Bus is only a device which can be set on/off by the user.

Using openConfigurator

For controlling the in- and outputs, openConfigurator has to be used, in order to make it possible to program the MN so it can control the CN. The openConfigurator plug-in creates the network by generating *mnobd.cdc*, *mnobd.txt*, *xap.xml* and *xap.h*, as mentioned in the theory chapter 2.3.

The method for using openConfigurator is done by using the following steps [Kalycito, 2017b].

1. Install Eclipse

2. Install the openConfigurator plugin from Eclipse marketplace

3. Choose new PowerLink project in Eclipse. The the program will ask for the XDD-file for the MN. Note that the XDD-file has to be written for the MN, otherwise, it will not work.

4. The next step is to add the CN. This is done by adding the XDD-file for the CN and choosing the id for it. As mentioned above, the XDD-file has to be written for the CN.
5. For the CN, map all available objects, i.e., outputs. This is can be done by selecting *Map all available objects* in *TPDO* and *RPDO*. Here it is possible to see what kind of I/O:s that the CN has.

6. Select the desired cycle time for the network. After this, the generation of the files should be successful.

The XDD-file that was used for the MN was imported from the PowerLink stack.

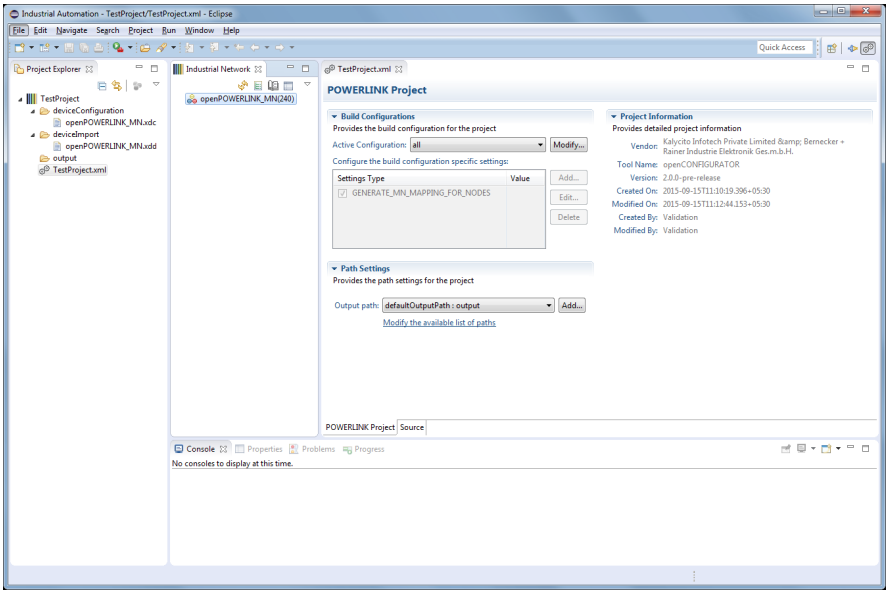

Figure 3.3 How openConfigurator looks when adding a MN [Kalycito, 2017b].

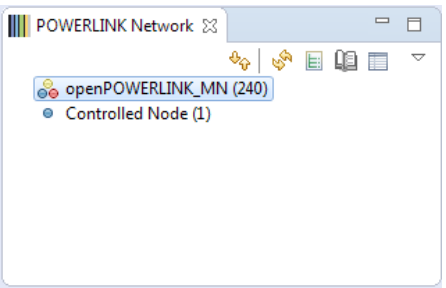

Figure 3.4 The network view of the PowerLink network after adding a CN [Kalycito, 2017b].

Controlling the Bus

After generating the files from openConfigurator, take the *xap.h* that was generated and replace the *xap.h* at *src* in the MN folder. This is done so the I/O:s get properly mapped. The next part is to program the MN so the Bus becomes controllable, in the sense of controlling the I/O:s.

Programming the MN is done, as mentioned before, in *app.c*. An example of controlling the buses's outputs is given in Section 3.6.
The example shows how the digital output and analog output are set. The code text

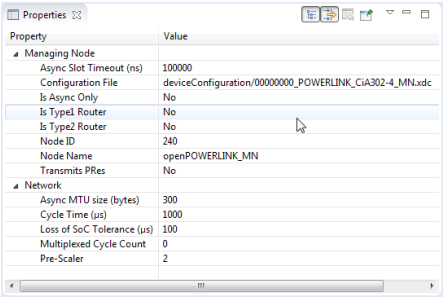

Figure 3.5 View of where the cycle time can be changed [Kalycito, 2017b].

```
 */
 //pProcessImageIn_l->CN1_DigitalOutput_00h_AU8_DigitalOutput_1 = 0;
 pProcessImageIn_l->CN1_DigitalOutput_00h_AU8_DigitalOutput_2 = 8;

 pProcessImageIn_l->CN1_AnalogueOutput_00h_AI16_AnalogueOutput_1 = 1;
 pProcessImageIn_l->CN1_AnalogueOutput_00h_AI16_AnalogueOutput_2 = 1;

//CN1_M00_DigitalOutput_00h_AU8_DigitalOutput = aNodeVar_l[0].leds
```

Figure 3.6 Example of controlling the outputs

pProcessImageIn_l->CN1_DigitalOutput_00h_AU8_DigitalOutput_2 = 8;, sets the digital output to 8, which means that output four (0x100) will be high on the X20BC1483 Bus.

pProcessImageIn_l->CN1_AnalogueOutput_00h_AI16_AnalogueOutput_1 = 1;

pProcessImageIn_l->CN1_AnalogueOutput_00h_AI16_AnalogueOutput_2 = 1;

While the code mentioned above sets the analog output to *1* which gives a *1* on the Bus and the lamp for the analog outputs glows.

3.12 PowerLink with PC as MN and ACCOPOS 1045

Running a PowerLink network with PC and ACCOPOS was not possible, as the XDD-file for ACCOPOS was not publicity available from B&R.

3.13 PowerLink with PC MN and Raspberry Pi2 CN

The first step is to build both the MN directory and the CN directory and then run MN program for PC and CN program for Raspberry Pi2. The PC has to be

connected to the Raspberry Pi2 using an Ethernet cable and the PowerLink communication is up and running but neither the MN nor CN is sending or receiving in the PowerLink stack. Applying the knowledge from previous using Listing 3.2 and changing the digitalIn_1(UINT8) to any value from 0 to 255 on the sending node and receiving node will pick it up. Confirming this can be done by printing the variable digitalOut_1 on the receiving node.

3.14 PowerLink with B&R PLC MN and Raspberry Pi2 CN

To implement a PowerLink network with a B&R PLC as MN, Automation Studio has to be used. Setting up the PowerLink network is much easier to do inside Automation Studio, as it automatically selects the desired PLC as MN inside the program. Automation Studio also has support for adding devices through importing XDD-files.

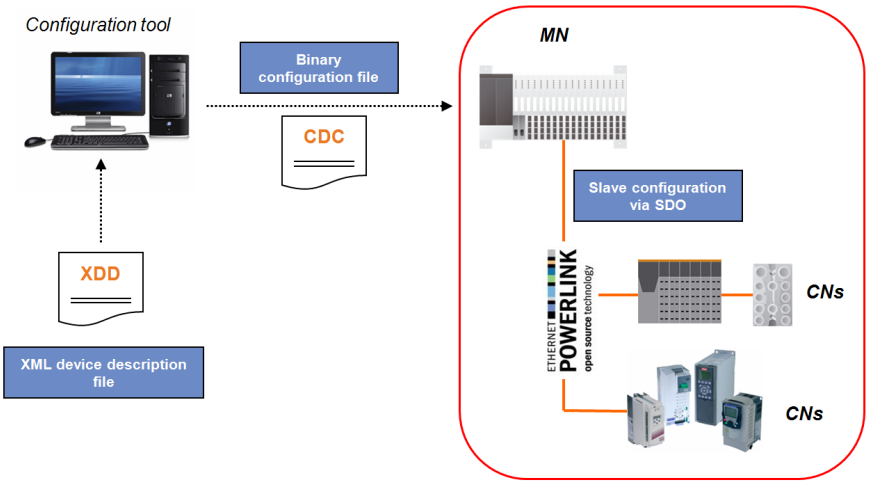

Figure 3.7 The general idea of implementing XDD-file on a B&R PLC [B&R, 2015].

The adding of a CN is done by selecting *Tools* and *Import Device* inside Automation Studio.
For reading values from the CN it requires that the user enables the read or write part of values from the CN in the *I/O configuration* in Automation Studio.

Importing the Raspberry Pi2 as a CN is done by adding the XDD-file for the node. The setup of the Raspberry Pi2 has to be done first so it can run as a CN in the

network.
In Section 3.17, it is shown how one can decide what values the CN should send to the MN, which in this case is the PLC.

```
pProcessImageIn_1->digitalIn = 3;
pProcessImageIn_1->digitalIn2 = 10;
pProcessImageIn_1->digitalIn3 = 72;
pProcessImageIn_1->digitalIn4 = 216;
```

Figure 3.8 Example of setting values on the input which will be sent to the MN

The values can be viewed from the PLC, although it requires some changes in Automation Studio. This is made in I/O configuration, where the inputs have to be set from *None* to *Read*. This will allow the PLC to print these values for observation.

3.15 PowerLink with B&R PLC MN and BeagleBone Black CN

It is possible to run a BealgeBone Black board as a CN. In this case the built file was imported from USB flashmemory to the board, and the PowerLink network between the PLC and board worked fine. Although no further development was taken with the BeagleBone Black board, as it is much slower than the Raspberry Pi2.

3.16 The OpenCV program

The OpenCV program was downloaded from a github repository [Dahms, 2017]. Here the program, which was written in C++ code, was altered to HSV, so it could be used with a webcamera and be able to detect a circular object depending on its color.

3.17 PowerLink with B&R PLC MN and OpenCV on Raspberry Pi2 CN

Integrating PowerLink and OpenCV requires changes in the PowerLink network. This is because the values that are to be read from OpenCV, which are in this case float numbers, need altering in the CNs *objdicts.h* file. As mentioned earlier, in *objdicts.h* it is possible to create I/O:s with different data types. The *objdicts.h* which came with the PowerLink stack has the restriction that it only has, e.g., four digital inputs and data type UINT8.

The creation of more I/O:s is done inside the file *objdicts.h* and the XDD-file that is used by the MN, PLC in this case. The *objdicts.h* needs to be exactly as the XDD-file and if the XDD-file does not match the other file, then the PLC will not be able to communicate with the new I/O:s, as the XDD-file that the PLC MN uses does not describe the new I/O:s.

The following text and Figure 3.9 and 3.10 describe how the *objdicts.h* and XDD-file are altered for creating new I/O:s are done.

```
// RealInput_00h_AU8
OBD_BEGIN_INDEX_RAM(0x6000, 0x05, FALSE)
    OBD_SUBINDEX_RAM_VAR(0x6000, 0x00, kObdTypeUInt8, kObdAccConst, tObdUnsigned8, NumberOfEntries, 0x04)
    OBD_SUBINDEX_RAM_USERDEF(0x6000, 0x01, kObdTypeReal32, kObdAccVPR, tObdReal32, DigitalInput, 0x00)
    OBD_SUBINDEX_RAM_USERDEF(0x6000, 0x02, kObdTypeReal32, kObdAccVPR, tObdReal32, DigitalInput, 0x00)
OBD_END_INDEX(0x6000)

// DigitalInput_00h_AU8
OBD_BEGIN_INDEX_RAM(0x6010, 0x05, FALSE)
    OBD_SUBINDEX_RAM_VAR(0x6010, 0x00, kObdTypeUInt8, kObdAccConst, tObdUnsigned8, NumberOfEntries, 0x04)
    OBD_SUBINDEX_RAM_USERDEF(0x6010, 0x01, kObdTypeUInt8, kObdAccVPR, tObdUnsigned8, DigitalInput, 0x00)
    OBD_SUBINDEX_RAM_USERDEF(0x6010, 0x02, kObdTypeUInt8, kObdAccVPR, tObdUnsigned8, DigitalInput, 0x00)
    OBD_SUBINDEX_RAM_USERDEF(0x6010, 0x03, kObdTypeUInt8, kObdAccVPR, tObdUnsigned8, DigitalInput, 0x00)
    OBD_SUBINDEX_RAM_USERDEF(0x6010, 0x04, kObdTypeUInt8, kObdAccVPR, tObdUnsigned8, DigitalInput, 0x00)
OBD_END_INDEX(0x6000)
```

Figure 3.9 Creating digital input in *objdicts.h*

```
<Object index="6000" name="DigitalInput_00h_AU8" objectType="8" dataType="0005">
  <SubObject subIndex="00" name="NumberOfEntries" objectType="7" dataType="0005" accessType="const" defaultValue="4" PDOmapping="no"/>
  <SubObject subIndex="01" name="DigitalInput" objectType="7" dataType="0008" accessType="ro" PDOmapping="TPDO"/>
  <SubObject subIndex="02" name="DigitalInput" objectType="7" dataType="0008" accessType="ro" PDOmapping="TPDO"/>
</Object>

<Object index="6010" name="DigitalInput_00h_AU8" objectType="8" dataType="0005">
  <SubObject subIndex="00" name="NumberOfEntries" objectType="7" dataType="0005" accessType="const" defaultValue="4" PDOmapping="no"/>
  <SubObject subIndex="01" name="DigitalInput" objectType="7" dataType="0005" accessType="ro" PDOmapping="TPDO"/>
  <SubObject subIndex="02" name="DigitalInput" objectType="7" dataType="0005" accessType="ro" PDOmapping="TPDO"/>
  <SubObject subIndex="03" name="DigitalInput" objectType="7" dataType="0005" accessType="ro" PDOmapping="TPDO"/>
  <SubObject subIndex="04" name="DigitalInput" objectType="7" dataType="0005" accessType="ro" PDOmapping="TPDO"/>
</Object>
```

Figure 3.10 Creating digital input in XDD-file

As mentioned in the theory part of XDD, different values in the XDD-file have different meaning. The data types can be set according to the Figures 3.9 and 3.10. This will create a digital input with the data type. Notice the similarities and differences from *objdicts.h* to the XDD-file. In *objdicts.h* the *OBD_BEGIN_INDEX_RAM* macro can be found. The macro defines the index entry of the object dictionary [OpenPowerLink, 2017e]. The macro has three sets of parameters. The first one describes the object index of the entry, in this case 6000 - 9FFF indicates that its a standardized device profile area. Next parameter shows how many sub-objects there are in this object, *0x05* indicates that there are five sub-objects. The last parameter is of boolean type and can only be set true or false. This last parameter is a flag which indicates that if the object is accessed, then it will trigger a user event. As there is no need for this, it is set to false [OpenPowerLink, 2017h].

The shell for the digital inputs has now been created. Now the definition of what the object for the variables has for data length, in other words the size of the data. This is done in the macro *OBD_SUBINDEX_RAM_VAR*, which has seven parameters. The first one is the address, which is the same as before, *0x6000*. The second parameter defines the number of sub-indexes, in this case *0x00* is written, which gives the information that it is not used. The third parameter defines the object type, here it is written *kObdTypeUInt8*, which indicates that the data type is UNSIGNED8. [OpenPowerLink, 2017f]. The fourth parameter defines the access right for the object type, here its written as *kObdAccConst*, with this it tells that the object contains a constant value [OpenPowerLink, 2017g]. The fifth parameter indicates which C data type is used, with *tObdUnsigned8* it assigns the data type INTEGER8 according to PowerLink specifications [OpenPowerLink, 2017f]. The sixth parameter is the name of the object, which in this case is just *Number Of Entries*. Lastly, the seventh parameter is the default value for the subindex, with *0x04* it indicates that it is a constant value [OpenPowerLink, 2017e].

The last step is to create the inputs. The *OBD_SUBINDEX_RAM_USERDEF* maps the objects to a PDO. With this macro, objects can be created. The parameters are similar to *OBD_SUBINDEX_RAM_VAR*, the only difference is the type of data type, here a float data type is chosen, which is *kObdTypeReal32*. The access right *kObdAccVPR* gives three rights. First is that the object can be placed in an application, which is in PowerLinks information structure. Second is that it can be mapped to a PDO. The third indicates that the object can be read [OpenPowerLink, 2017g]. The parameter *tObdReal32* is the C definition of the data type that needs to be set in order to get the correct setup. The string name in this case is set to *DigitalInput*. The last parameter is set to *0x00*, which says that the object has not got a default value.

When modifications have been done to the *objdicts.h* and the XDD-file, for the MN and CN, the corresponding file needs to be exactly the same. The result of not doing this will be that the PLC will not find the input or the CN will get an error when building the PowerLink running files. The meaning of "exactly" is that the inputs need to be described in both files. When examining Figures 3.9 and 3.10, the difference is that *objectType* is written in numbers, *0008* indicates thats its an array while *0007* is a single value variable [OpenPowerLink, 2017a]. The data type is defined in *dataType*, where *0005* is UNSIGNED8 and *0008* is data type REAL32. *accessType* is here set to *const*, which says that the object can only be read from the network and node side. *ro* gives the read only from the network [OpenPowerLink, 2017a].

To ensure that the XDD-file is correctly written, either the XDD-file can be compiled and checked for errors at OpenPowerLink's website or in Automation Studio. When all the files are ready, the XDD-file can now be implemented in B&R:s PLC

and a PowerLink network with Raspberry Pi2 can now be established, where the OpenCV can send its values through desired inputs. Note that in this case the written OpenCV code was written in C++ and PowerLink stack in C. To communicate between the programs a server/client base network was established through socketing. Here the OpenCV program will act as a client and send the information to PowerLink, which acts as a server.

3.18 PowerLink with B&R PLC MN and OpenCV on Raspberry Pi3

As mentioned before, Raspberry Pi3 is a bit faster than its predecessor, Pi2. PowerLink and OpenCV was previously installed on a microSD card, which contains all the files. Instead of reinstalling all the necessary files, the microSD card was taken from the Pi2 to Pi3 and this worked without any problems.

3.19 PowerLink with B&R PLC MN and OpenCV on Stationary Computer as CN

PowerLink and OpenCV were also installed on a stationary computer with Ubuntu as the operating system. This was done while investigating reasons to why the PowerLink on the Raspberry restarted itself and was not stable. The reason that Ubuntu was chosen was that with the operating system it made programming much easier as it sets the path, e.g. for cmake, automatically which saves time. All files which were used in the Raspberry Pi3 were imported into the computer through a USB memory. Although all files were needed to be rebuilt as the paths were not correctly set. So the first step is to delete all files and then rebuilt it to make the programs work. After these steps, the files executed as with the Pi3.

3.20 PowerLink with B&R PLC MN and OpenCV on Stationary Computer CN and Threading with Mutex

Some precaution steps were taken when handling PowerLink. During the programming no implementation of threads had been used. This could cause time delays in the system which make the PowerLink restart itself. The protection against this kind of problem, the usage of threading and mutes was sought. The implementation of threads and mutex works as following; the server writes and while PowerLink reads the variables. These variables are *XPosition* and *YPosition*, which correspond to the objects position in the x and y direction. These variables are a shared resource as it is used in both programs. The programs are programmed with mutex which locks/unlocks the resource the current thread is using, so no conflict can appear.

Figure 3.11 illustrates the current setup.

As explained before, the camera will feed the OpenCV program with pictures of the robot workspace. When this information reaches the OpenCV software it will begin to calculate the position of the object. When this is completed, the data will then be sent to the client which in turn sends the data to the server, which is located in the PowerLink software. The server writes the information to variables that are shared with the send data method, which locks the resource, then reads the variables and unlocks. The second thread, PowerLink, will lock the shared resource, read the variables and send the information to the MN, PLC, and then unlock when finished.

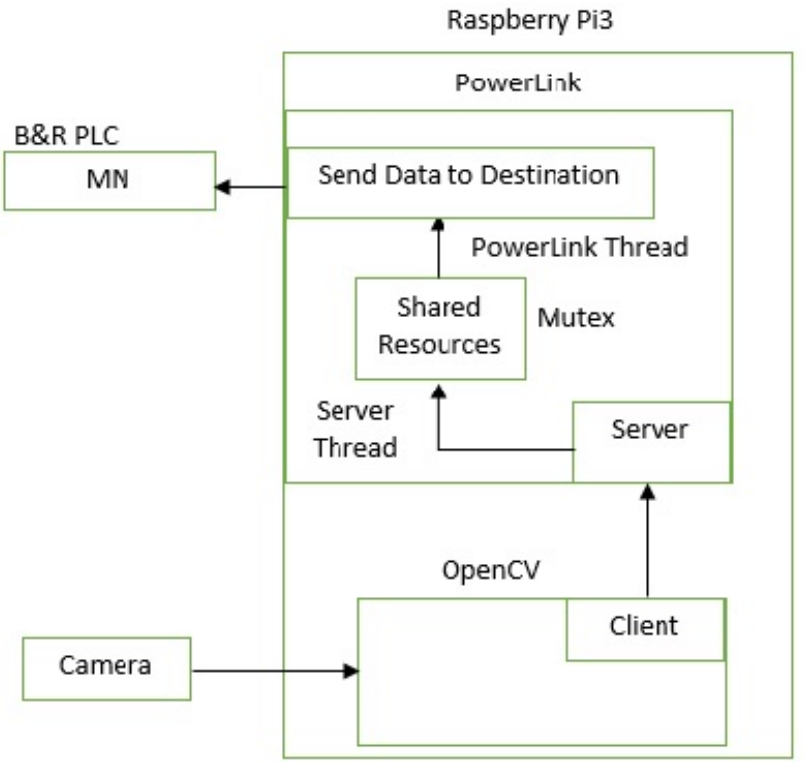

Figure 3.11 The program architecture for PowerLink and OpenCV.

3.21 PowerLink with B&R PLC MN and OpenCV on Raspberry Pi3 CN and Threading

The same code that was used as in Section 3.19. The files were imported from the PC to the Raspberry Pi3 through a USB memory card. The same technique was used as in Section 3.18; deleting the build files and then rebuilding it so it can run properly.

4

Results

4.1 Creating and Controlling Control Nodes

The result showed that creating and maintaining a PowerLink network requires a lot of knowledge of how it works. Setting up the network can be cumbersome for newcomers, as a lot of information how to program is not demonstrated in the document for PowerLink. Directly controlling ACOPOS 1045 was not successful as the XDD-file for the servodrive was not available. Although this could not be achieved, the direct control of B&R:s X20BC1483 showed that it is possible, as long the XDD-file for the device is provided by the manufacturer. With the XDD-file the user can also alter the file to desired configuration like add or remove I/O variable and run it on Raspberry Pi2, Pi3 or Beaglebone. When using B&R products one should not alter their XDD-file since it would not be compatible with the XDD-file inside the device. This made that creating a directly control of ACOPOS 1045 from a PC not possible.

From Section 3.10 the PC could control the I/O on the X20BC1484. The digital module of the bus had a number of outputs that could be lit e.g, if 0x1010 was sent to the module meaning that output 2 and 4 were set. Data on other modules, like digital input, analog input/output can also be read and sent to. Something that was noticeable was that the MN restarted itself continuously, but this was later resolved by expanding the cycle time, as the cause was that the cycle time was too short.

When comparing BeagleBone Black, Raspberry Pi2 and Pi3, the Raspberry Pi3 proved to be the most stable node to handle. For instance the BeagleBone Black took approximately eight hours to just install the software cmake. The BeagleBone Black Board can still be used as a CN, but it is not recommended, as compiling can take long time compared with Raspberry Pi2. It is possible to use crosscompiling, in the sense that one is compiling the program on a PC and then importing to the BeagleBone. However, the problem of hardware limitations is still an issue.

4.2 The OpenCV Software

First the idea was to use OpenCV code from another master thesis, *Alvaro Gomez-Trenor Sobrino's Real-time control of the FlexPicker ABB robot*, but this proved not feasible during the thesis time. The thesis work can be viewed at department of Automatic Control's web page for publications [Control, 2018]. Instead implementation of an object detector was created by using HSV. The program will detect the color and calculate the object's position. In return x and y coordinates will be the output and these values are pixel values. As with the code for the OpenCV, it could still be executed on the Raspberry Pi2, but with bad performance. The OpenCV software was also tested on the newer Raspberry Pi3, which ran faster than its predecessor. Figures 4.1, 4.2 and 4.3 show the OpenCV software in action. Altering the color of the object to be detected can be done in real-time by using the trackbars.

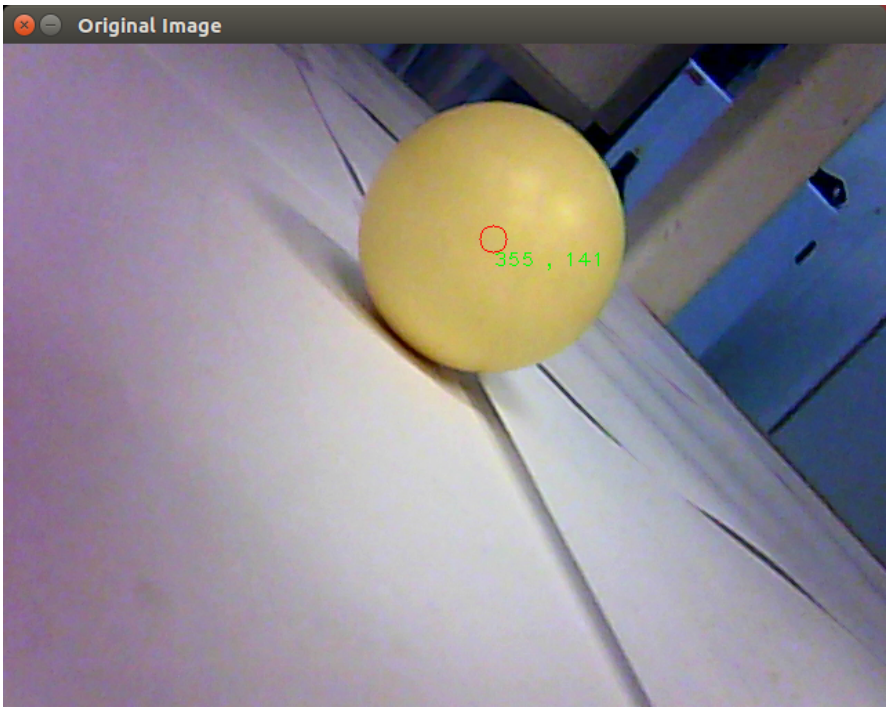

Figure 4.1 The image camera feeds the program.

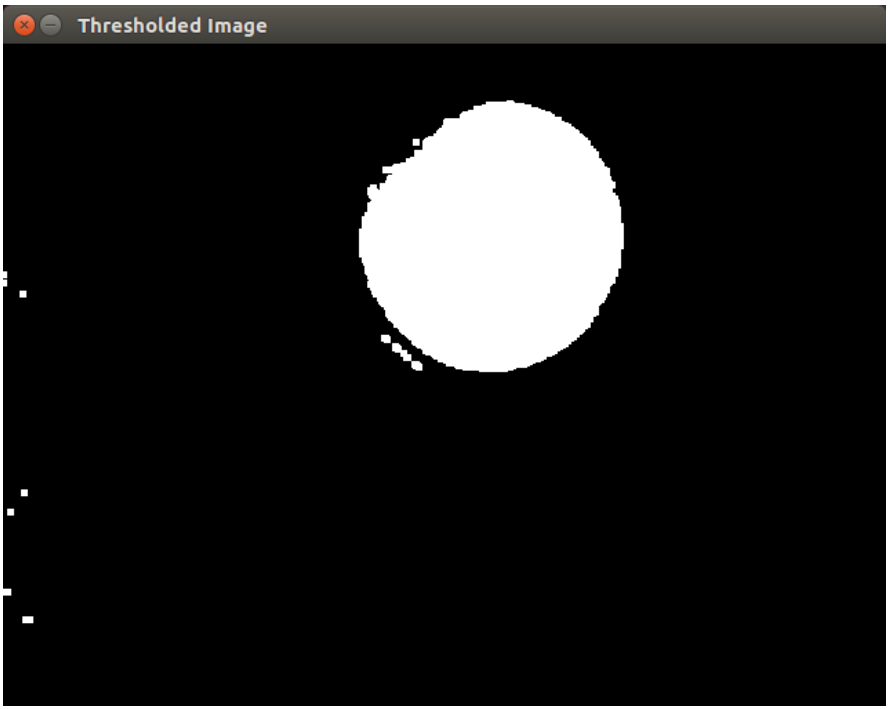

Figure 4.2 The threshold image.

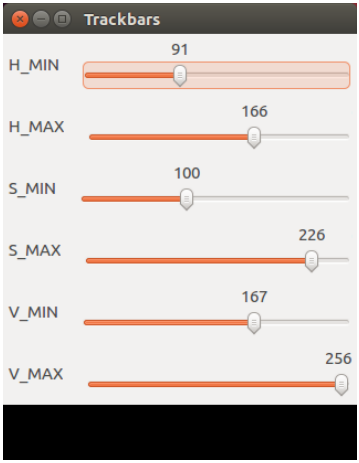

Figure 4.3 The Trackbars.

4.3 PowerLink and OpenCV

One of the main problems was also that the OpenCV code is written in C++ while the PowerLink is mainly written in C. Due to this, rewriting and implementing OpenCV and PowerLink as one program was not an efficient solution. Rewriting the code for OpenCV was considered but this would have taken further time. To solve this problem, creation of a client/server network through socketing was sought and implemented. Thus the functionality of the programs were not compromised and this led to an easier implementation of the two programs.

4.4 Unstable connections and Unfamiliar Errors

Creating a PowerLink network and connecting it to different devices can still be implemented on various devices, but the stability is still an issue. This is due that the CN restarts itself every now and then. The error has been identified as *NMTEvent-NMTCycleError*. The first solution was to expand the cycle time, but this did not solve the problem as it only expanded the time the PowerLink could send information to the MN. The PowerLink and OpenCV program was implemented on Raspberry Pi3 as Pi2 had a very bad performance. This led to a better performance in the OpenCV program with Pi3, but the stability was still an issue. Fortunately the cause has been identified to be the hardware limitation of Raspberry Pi2 and Pi3.

The identification of the problem was made when running Raspberry Pi3 with only the PowerLink side. The PowerLink network sent only some integers to the MN, PLC, and the system did not restart itself. But as soon the OpenCV software was executed the problem reappeared. Even threading and mutex was used to allocate the data power mainly to PowerLink so it can send the information as fast as possible and reassure that the shared resources do not interfere with the programs. If the information takes too long to be send, then the PowerLink communication will restart itself. But the problem still persisted and this led to the conclusion that the Raspberry Pi2 and Pi3 can not be used as sensor nodes due to hardware limitations and not suitable for industrial applications as the stability is a must. Instead, a stationary Ubuntu computer was used, which led to a satisfactory sensor node with good real-time performance.

Performance

When OpenCV was executed on the Raspberry Pi2, measurements showed that the computation time was approximately ten times slower than when running on a PC with i7 processor which make the Raspberry Pi2 not suitable for a high performance system. The Pi3 had a better performance, however the Pi2 and Pi3 had still the same problem of running PowerLink and OpenCV together.

For the stationary Ubuntu computer, the results were good and worked satisfactorily with good real-time performance. Time was measured between the calculation time of the OpenCV and the send data time in PowerLink.
This shows that it has a good real-time performance, which can be easily implemented in terms of automatic control. The sensor node will send the data to the PLC which lets the user use the information for their needs, e.g., a robot movement to pick up a object. This kind of device is both suitable for testing and industrial systems, since it is much easier to install desired programs and software in a Ubuntu computer than a PLC.

The calculation time data was gathered with 65 samples. The median calculation time was 0.8972/65 = 0.0138 seconds, for calculating the coordinates to sending it through to the PowerLink section of the code. While as the time for sending data through PowerLink had the median time of 0.00016 seconds. This shows that the PowerLink node has good real-time performance and is suitable to be integrated in an automatic control device, e.g., determining position for robots or for objects in their workspace.

Operating system

Running PowerLink on Microsoft Windows and Linux is not any problem since both the operating system supports pcap. Linux is using libpcap and Microsoft Windows is using Winpcap. No investigation has been done to macOS since there was no computer available.

5

Discussion and Conclusions

5.1 Conclusions

The PowerLink Network

Creating new I/O:s and datatypes proved to be cumbersome as no direct documentation showed how to create these. The PowerLink stack provides a lot of files, which makes it easy to get lost in the jungle of information. OpenPowerLink's webpage provides a demo of how to build a PC to PC PowerLink connection, though there is no mentioning on how, for example, to send some data from one PC to another. This knowledge is self gained, as it requires the user to go into the stack files and identify where the programming should take place.

Implementing direct control of a CN with X20BC0083 was possible as the XDD-file could be provided from B&R:s webpage. Though it was not possible for implementing direct control of ACOPOS 1045, as these could not be provided from the manufacturer. Creating an own XDD-file is fully possible, but not for a product which is made by a manufacturer. The reason is that the XDD-files require addresses, where information can be sent from and received to desired place. The problem with this is if there is no knowledge where these addresses are. In order to control, e.g., the torque, the datatype has to be correct and the address has to be exactly the same as the drives I/O:s. The probability of making an XDD-file that matches with the drives inside ACOPOS 1045 makes it very unrealistic.

Choosing another set of drives was not possible, as the motor drives, ACOPOS 1045, are only those available and connected to the FlexPicker in the Robotlab. The idea was to compare, later on, the implementation of controlling the ABB FlexPicker with the master thesis as mentioned before, *Alvaro Gomez-Trenor Sobrino's Real-time control of the FlexPicker ABB robot* [Control, 2018], and show that it is easier to just use PowerLink. Nevertheless, due to the missing piece, which is the

XDD-file for ACOPOS 1045, made this not possible. One could ask the question, why was it not investigated beforehand if the XDD-file was available. The reasons are two. The first reason is the lack of experience of building these kinds of systems. The second reason was that reading manuals gave the idea that XDD-file could be provided by just asking the manufacturer. As the XDD-file for X20BC0083 was directly provided from B&R:s webpage, it gave the idea the other XDD-file for ACOPOS 1045 also was available, which later on proved that this was not the case.

Learning how to use and create a PowerLink network was a handful of work, as the demo for implementing a Raspberry Pi2 CN did not work as planned. This was because the files for some reasons could not be installed properly, which required a lot of time before this problem could be solved. Even using OpenConfigurator was a problem. There is a lot of information about it, even tutorials and videos, but no documentation showed which files should be replaced in the PowerLink stack. To know which files to replace was learned through testing.

To mention, the support for PowerLink exists from OpenPowerLink in form of forums. Here questions were asked regarding the files for RaspBerry Pi2 files and the error that makes the PowerLink restarting. It took approximately two days before a response on the first question, the last one has not been responded yet, but will hopefully be soon so others can handle this kind of problem when meeting it. This shows that a project can be significantly delayed if need of support and no work around the problem were available. The problem was later resolved but took a lot of time which significantly delayed the project.

One of the reasons for implementing a BeagleBone Black board as a PowerLink sensor, was that it is faster than Raspberry Pi. Note that it is true for Raspberry Pi but not for Pi2. Due to confusion and the insufficient investigation of data specification prior to implementation, resulted in a waste of time when trying to implement the BeagleBone Black as a sensor node. In fact Raspberry Pi2 is much faster due to that its quad core while BeagleBone Black is a single core. As mentioned earlier, just installing cmake for running the PowerLink stack took approximately eight hours. A lot of effort was put into making this work, but to no avail, due to the long time it took.

Initially when trying to install OpenCV on the Raspberry Pi2 board, it did not go as planned. The board froze during the installation and when rebooting was attempted, the board could not boot up. This led to reinstallation of a newer operating system on the board which, eventually, led to that OpenCV could be successfully installed. As mentioned before, the OpenCV code that was implemented on the Raspberry Pi2 board, was borrowed from *Alvaro Gomez-Trenor Sobrino's Real-time control of the FlexPicker ABB robot* [Control, 2018]. The code was successfully implemented and executed but due to limitation in computer power on the Raspberry Pi2 board,

made it ten times slower than running with an ordinary PC. Conclusion was made when discussing with the thesis writer, Alvaro Gomez-Trenor Sobrino, about the efficiency of the program.

Having OpenCV on a Raspberry Pi2 and Pi3 board proved that camera vision program can be implemented and act as a sensor. The main problem with this was compatibility, where the OpenCV code was written in C++ and PowerLink in C. Rewriting the OpenCV code would have required a lot of time. Knowledge about how OpenCV works would have to be gained and then rewriting the code. All of this would have taken a lot of time. Thus a server/client network was created to solve this problem.

During the implementation part, efficiency was also looked at. If one, e.g., has a system where the dependency for running smoothly lies on a sensor. That is slow and for some unknown reason restarts itself, while the good part is that it is easy to implement. Even if the PowerLink sensor is easy to connect, stability and performance is still the crucial part of making the desired system a working system. For research purposes it can still be used, as the PowerLink network will still send the desired information, but for industrial systems it is not recommended as, e.g., a controlled machine will stop now and then due to that the PowerLink network with the sensor is restarting. Thus its not recommended to use small boards like Raspberry Pi for sensor implementation. It is recommended to use a ordinary Linux computer for building sensors for research and industrial systems.

OpenCV and PowerLink

The implementation of the OpenCV program worked on Raspberry Pi2 and Pi3. The program was written in the programing language C++ by its master thesis author, *Alvaro Gomez Trenor Sobrino*. This is a problem as PowerLink's main function is written in C. In order to solve this problem several solutions were looked into. One solution would be to rewrite the OpenCV code to C and implement it the PowerLink code. But as the OpenCV code is borrowed from another master thesis, it is very hard to understand and requires significant time to understand a master thesis code and implement it onto PowerLink, and functions that exists in C++ cannot be handled in C and needs basically to be rewritten. Another problem is that changes to the PowerLink code must be done with caution. This is because as nearly all files are linked with each other. This was not implemented as a lot of time and priority went to identify the problem of the PowerLink stability, which made the implementation of OpenCV and PowerLink suffer.

A server/client network was implemented to get the two programs to communicate. Even with multi threading in use in favor to the PowerLink code the Raspberry could not handle it. In the end the only conclusion is that the Raspberry hardware

is not powerful enough to handle OpenCV and PowerLink.

5.2 Future Work

One of the criteria of creating a sensor which sends information through PowerLink, is that the connection has to be stable. Stability is as mentioned before, a crucial part of a system. For research purposes, PowerLink has great potentiality to be able to directly connect and test equipment by just connecting an Ethernet cable and run a PowerLink network would be very beneficial, in terms of installing, e.g., firmware-update, OpenCV, etc, on the desired device. But as mentioned before, stability is still an issue and this problem is hardware dependent. Hopefully when newer versions with better hardware of the Raspberry Pi is released, this problem will be resolved.

For future development integrating this kind of sensor into automatic control solutions will be possible. Other implementations of the OpenCV, such as Aruco which ressembles a QR code but has vast information on it, is also very interesting to look into and implement it as a PowerLink sensor node [OpenCV, 2017b]. It will also be much easier to implement as the user does not have to program into Automation Studio, only receiving the information and using it in an automation control solution. This thesis shows that its possible to create a sensor node and send the data through PowerLink with ease. At the same time, it shows that its not necessary to open sockets on, e.g., a PLC in order to send desired information to it.

The development for the future can be summarized into two parts. Ensure that the PowerLink connection is stable and integrate *Alvaro Gomez Trenor Sobrino's thesis*, *Real-time control of the FlexPicker ABB robot* [Control, 2018], and this thesis work together and show that determinism can be achieved in a automatic control network.

Bibliography

BeagleBone (2017). *What is BeagleBone*. http://beagleboard.org/black. [Online; accessed 10-May-2017].

B&R (2008). *B&R*. https://www.br-automation.com/en/perfection-in-automation/. [Online; accessed 19-Feb-2017].

B&R (2015). *TM950 - PowerLink Configuration and Diagnostics (Internal)*. B&R.

B&R (2017a). *Acopos 1045*. https://www.br-automation.com/en/products/motion-control/acopos/. [Online; accessed 18-May-2017].

B&R (2017b). *B&R Homepage*. https://www.br-automation.com/en/perfection-in-automation/. [Online; accessed 15-Aug-2017].

B&R (2017c). *X20BC0083*. https://www.br-automation.com/smc/436939515bf8783498e912c30efc9a4d680e8e04.jpg. [Online; accessed 28-April-2017].

B&R (2017d). *X20CP1483 PLC*. http://pkcontrol.com/th/Production.html/. [Online; accessed 03-Mars-2017].

B&R (2017e). *XDD-package for X20BC0083*. https://www.br-automation.com/en-au/downloads/networks-and-fieldbus-modules/powerlink/bus-controllers/x20cbc0083/dwldwl10000455016/. [Online; accessed 28-April-2017].

Cmake (2017). *About Cmake*. https://cmake.org/. [Online; accessed 18-May-2017].

Control, A. (2018). *Publications*. http://www.control.lth.se/Publications.html. [Online; accessed 23-Jan-18].

Dahms, C. (2017). *OpenCV Github Repository*. https://github.com/MicrocontrollersAndMore/OpenCV_3_Windows_10_Installation_Tutorial. [Online; accessed 21-May-2017].

Eclipse (2017). *About Eclipse*. https://eclipse.org/org/. [Online; accessed 18-May-2017].

Ethernet.PowerLink.org (2017a). *PowerLink Architecture*. http://openpowerlink.sourceforge.net/web/POWERLINK.html. [Online; accessed 01-March-2017].

Ethernet.PowerLink.org (2017b). *PowerLink Cycle*. http://openpowerlink.sourceforge.net/web/POWERLINK/Mechanism.html. [Online; accessed 14-March-2017].

Ethernet.PowerLink.org (2017c). *PowerLink Cycle*. http://openpowerlink.sourceforge.net/web/POWERLINK/PDO.html. [Online; accessed 24-April-2017].

Kalycito (2017a). *Kalycito*. https://www.kalycito.com/index.php/detailedhowto-run-openpowerlink-on-raspberry-pi2. [Online; accessed 15-Dec-17].

Kalycito (2017b). *User Manual for Ethernet POWERLINK openCONFIGURATOR eclipse plugin, Version: 2.1.1*. https://goo.gl/jORhdR. [Online; accessed 08-May-2017].

Microsoft_Visual_studio (2017). *About Visual Studio*. https://blogs.msdn.microsoft.com/developer-tools/. [Online; accessed 18-May-2017].

Notepad++ (2017). *About Notepad++*. https://notepad-plus-plus.org/. [Online; accessed 18-May-2017].

OpenCV (2017a). *About OpenCV*. http://opencv.org/about.html. [Online; accessed 15-May-2017].

OpenCV (2017b). *Detection of ArUco Markers*. http://docs.opencv.org/3.1.0/d5/dae/tutorial_aruco_detection.html. [Online; accessed 07-Sep-17].

OpenCV (2017c). *Installation in Linux*. http://docs.opencv.org/2.4/doc/tutorials/introduction/linux_install/.html\#linux-installation. [Online; accessed 15-May-2017].

OpenPowerLink (2017a). *Determining Cycle Time*. http://openpowerlink.sourceforge.net/web/openPOWERLINK/FAQ.html. [Online; accessed 25-April-2017].

OpenPowerLink (2017b). *Installation in Linux*. http://openpowerlink.sourceforge.net/web/openPOWERLINK/FAQ.html. [Online; accessed 18-May-2017].

OpenPowerLink (2017c). *Installing Libraries*. http://openpowerlink.sourceforge.net/doc/2.5/2.5.1/page_build_stack.html. [Online; accessed 28-April-2017].

OpenPowerLink (2017d). *Node Addressing*. http://openpowerlink.sourceforge.net/web/POWERLINK/Frame\%20Format/Node\%20Addressing.html. [Online; accessed 18-May-2017].

OpenPowerLink (2017e). *obd.h File Reference*. http://openpowerlink.sourceforge.net/doc/2.5/2.5.1/d5/df2/obdmacro_8h.html\#a27797b92cc8e9ccc60690f541380383f. [Online; accessed 21-May-2017].

OpenPowerLink (2017f). *obd.h File Reference*. http://openpowerlink.sourceforge.net/doc/2.5/2.5.1/d9/db5/obd_8h.html\#a4325e838c1353473c2303163d809c731. [Online; accessed 21-May-2017].

OpenPowerLink (2017g). *obd.h File Reference*. http://openpowerlink.sourceforge.net/doc/2.5/2.5.1/d9/db5/obd_8h.html\#sect_obdAccessRights. [Online; accessed 21-May-2017].

OpenPowerLink (2017h). *obdmacro.h File Reference*. http://openpowerlink.sourceforge.net/doc/2.5/2.5.1/d5/df2/obdmacro_8h.html\#a2fb60002068195ebb6045fa7efeb9958. [Online; accessed 19-May-2017].

OpenPowerlink (2017a). *Object and sub-object*. http://openpowerlink.sourceforge.net/web/POWERLINK.html. [Online; accessed 30-Mars-2017].

OpenPowerlink (2017b). *Object and sub-object*. http://openpowerlink.sourceforge.net/doc/2.5/2.5.1/. [Online; accessed 30-Mars-2017].

OpenPowerLink (2017a). *Object and sub-object*. http://openpowerlink.sourceforge.net/web/POWERLINK/Object\%20Dictionary/Object.html. [Online; accessed 21-May-2017].

OpenPowerLink (2017b). *OpenPowerLink Download*. http://openpowerlink.sourceforge.net/web/openPOWERLINK/Download.html. [Online; accessed 21-May-2017].

PowerLink (2017). *PowerLink Technology*. http://www.ethernet-powerlink.org/en/powerlink/technology/. [Online; accessed 15-Aug-2017].

randu.org (2017). *Multithreaded Programming (POSIX pthreads Tutorial)*. https://randu.org/tutorials/threads/. [Online; accessed 21-May-2017].

Raspberrypi (2017a). *Raspberrypi2*. https://www.raspberrypi.org/products/raspberry-pi-2-model-b/. [Online; accessed 24-April-2017].

Raspberrypi (2017b). *Raspberrypi2*. https://www.raspberrypi.org/app/uploads/2016/02/Raspberry-Pi-2-web.jpg. [Online; accessed 24-April-2017].

Rosebrock, A. (2017). *How to install OpenCV on Raspbian Jessie*. http://www.pyimagesearch.com/2015/10/26/how-to-install-opencv-3-on-raspbian-jessie/. [Online; accessed 15-May-2017].

Sourceforge (2017). *Win32 Disk Imager*. https://sourceforge.net/projects/win32diskimager/. [Online; accessed 11-May-2017].

techopedia (2017a). *C++ Programming Language*. https://www.techopedia.com/definition/26184/c-programming-language. [Online; accessed 15-Dec-17].

techopedia (2017b). *C Programming Language (C)*. https://www.techopedia.com/definition/24068/c-programming-language-c. [Online; accessed 15-Dec-17].

ThoughtCo (2017). *HSV*. https://www.thoughtco.com/what-is-hsv-in-design-1078068?i10c. [Online; accessed 03-July-2017].

WhireShark (2017). *WhireShark capture*. http://searchsecurity.techtarget.com/tip/Wireshark-tutorial-How-to-sniff-network-traffic. [Online; accessed 29-Mars-2017].

Wireshark (2017). *Packet capture library (libpcap)*. https://wiki.wireshark.org/libpcap. [Online; accessed 15-May-2017].

Lund University
Department of Automatic Control
Box 118
SE-221 00 Lund Sweden

Document name
MASTER'S THESIS

Date of issue
February 2018

Document Number
TFRT-6048

Author(s)
Lloyd Tran
David Barbulovic

Supervisor
Anders Robertsson, Dept. of Automatic Control, Lund University, Sweden
Rolf Johansson, Dept. of Automatic Control, Lund University, Sweden (examiner)

Title and subtitle
Real-time Control of Industrial Robot Cell with PowerLink

Abstract

This thesis investigated how one can build a PowerLink network and Control Nodes that act as, e.g., sensor, actuators, etc., together with B&R:s products. Establishing different PowerLink networks was also investigated such as PLC connected to Raspberry Pi2. Configuration and step-by-step introduction of what PowerLink is and how it works is also mentioned in this thesis. In addition to creating PowerLink networks and nodes, this thesis also looked into implementing OpenCV for image processing with a camera on a Raspberry Pi2 connected to a B&R PLC.

PowerLink is a deterministic Ethernet-based protocol. It works by having a Managing Node that acts as the moderator for the network. Then there are the Control Nodes, which are slaves nodes in the network. The result showed that it is possible to create control nodes that can be controlled and read through PowerLink on a PC or PLC. The PowerLink camera node that was created was implemented on Raspberry Pi2 with OpenCV. The idea behind this was to show that it is possible to easily setup a PowerLink node. One of the problems that was experienced was that the PowerLink stack and OpenCV code that was to be implemented was written in C/C++, but this was solved by establishing a Server/Client communication between the two programs. The Raspberry Pi2 limited hardware, which made it not suitable for high performance systems.

This thesis showed that it is possible to create sensors, control and directly test them from a PC or implement them on a industrial PLC. For the users it is greatly beneficial due to the fact that a control node can execute code, e.g., filter measurements, before sending data to the PLC.

Keywords

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280-5316

ISBN

Language
English

Number of pages
1-59

Recipient's notes

Security classification

http://www.control.lth.se/publications/