# MASTER THESIS

**Design and implementation of robotic end-effectors for a prototype precision assembly system**

Prepared for the degree program
Information Technology & Systems Management
at the
University of Applied Sciences Salzburg
&
Carnegie Mellon University, Pittsburgh

submitted by:

**Sebastian Schoendorfer**

Fachhochschule
Salzburg University
of Applied Sciences

| | |
|---|---|
| Program Director: | FH-Prof. DI Dr. Gerhard Jöchtl |
| Supervisors: | FH-Prof. DI Dr. Robert Merz |
| | Prof. Ralph Hollis |

Pittsburgh, August 2016

## Affidavit

I, Sebastian Schoendorfer, born on 15.02.1991 in Freilassing, hereby declare that I have written this master thesis entirely on my own and that I have not used any other sources apart from those mentioned.

Pittsburgh, on the

[illegible]

Sebastian Schoendorfer

1410581005

Matriculation Number

## General Information

| | |
|---|---|
| First Name, Surname: | Sebastian Schoendorfer |
| University: | University of Applied Sciences Salzburg |
| Degree Program: | Information Technology & Systems Management |
| Title of Master Thesis: | Design and implementation of robotic end-effectors for a prototype precision assembly system |
| Keywords: | Smart Factory, Autocalibration, AAA, Multi Agent Factory, High Precision Assembly Line |
| Academic Supervisor: | FH-Prof. DI Dr. Robert Merz<br>Prof. Ralph Hollis |

## Abstract

Manufacturers are facing increasing pressure to reduce the development costs and deployment times for automated assembly systems. Since 1994, the Microdynamic Systems Laboratory at Carnegie Mellon University has been developing an automation framework, called Agile Assembly Architecture (AAA). Additionally to the concept, a prototype instantiation, in the form of a modular tabletop precision assembly system termed Minifactory [1, 2, 3], has been developed. In this thesis various enhancements for a second generation agent-based micro assembly system are designed, implemented, tested and improved. The project includes devising methods for tray feeding of precision high-value parts, micro fastening techniques and additional work on visual- and force-servoing. To help achieving these functions, modular and reconfigurable robot end-effectors for handling millimeter sized parts have been designed and built for the existing robotic agents. New concepts for robot end effectors to grasp and release tiny parts, including image processing and intelligent control software, were required and needed to be implemented in the prototype setup. In order to have a modular system, the factory the main part of this project was the initialization and auto calibration of the different agents. The main focus, of this research, is on improving the design, deployment and reconfiguration capabilities of automated assembly systems for precision mechatronic products. This helps to shorten the development process as well as the assembly of factory systems. A strategic application for this approach is the automated assembly of small sensors, actuators, medical devices and chip-scale atomic systems such as atomic clocks, magnetometers and gyroscopes.

# Acknowledgements

The work described in this thesis was conducted in cooperation with the Microdynamic Systems Laboratory at Carnegie Mellon University. Funding by the Austrian Marshall Plan Foundation is gratefully acknowledged.

The acknowledgment of the author and a word of thanks, is devoted to the people who encouraged this work, to those who have been a source of inspiration and to those who supported this project through their deep scientific expertise, their ongoing effort and their lasting academic commitment.

A special word of thanks is therefore devoted to Prof. Ralph Hollis  from Carnegie Mellon University for his guidance, his support, the opportunity to write this thesis at the Microdynamic Systems Laboratory and for the warm welcome in his team.

Many thanks go to Michael Shomin for the productive discussions, his drive and the good collaboration as lab mate as well as Jun Zhang and Greg Seyfart for their support and valuable counsels.

For establishing the connection to the Microdynamic Systems Laboratory at Carnegie Mellon University, the opportunity to do my thesis abroad and advising me in Austria I would like to thank FH-Prof. DI Dr. Robert Merz. I would also like to thank Dr. Rafael Valencia, PhD  at the University of Halmstad for supervising this thesis.

Lastly, I would also like to thank my family and friends for always supporting me.

# Contents

# Acronyms

| | |
|---|---|
| **AAA** | Agile Assembly Architecture |
| **AC** | Alternating Current |
| **ADC** | Analog Digital Converter |
| **AM** | Amplitude Modulation |
| **API** | Application Programming Interface |
| **ASCII** | American Standard Code for Information Interchange |
| **CAD** | Computer-Aided Design |
| **CMU** | Carnegie Mellon University |
| **DDS** | Data Distribution Service |
| **DNA** | Deoxyribonucleic acid |
| **DoF** | Degrees of Freedom |
| **EEPROM** | Electrically Erasable Programmable Read-Only Memory |
| **FM** | Frequency Modulation |
| **GoF** | Gang of Four |
| **GPIO** | General Purpose Input/Output |
| **GUI** | Graphical User Interface |
| **HMI** | Human Machine Interface |
| **HTTP** | Hypertext Transfer Protocol |
| **ID** | Identifier |
| **IDE** | Integrated Development Environment |
| **IO** | Input/Output |
| **IPT** | InterProcess communication Toolkit |
| **IR** | Infra-Red |

| | |
|---|---|
| **IV** | Inventor file |
| **JSON** | JavaScript Object Notation |
| **LCM** | Lightweight Communications and Marshalling |
| **LED** | Light Emitting Diodes |
| **m2m** | machine-to-machine middleware |
| **MSL** | Microdynamic Systems Laboratory |
| **OMG** | Object Management Group |
| **OS** | Operating System |
| **OSI** | Open System Interconnection |
| **p2p** | Point-to-Point |
| **PLL** | Phase-Locked Loop |
| **PSD** | Position Sensitive Detector |
| **QoS** | Quality of Service |
| **RMS** | Reconfigurable Manufacturing Systems |
| **RTI** | Real-Time Innocations, Inc. |
| **RTOS** | Real-Time Operating System |
| **SCADA** | Supervisory Control and Data Acquisition |
| **SOA** | Service-Oriented Architecture |
| **SPI** | Serial Peripheral Interface |
| **TCP** | Transmission Control Protocol |
| **UDP** | User Datagram Protocol |
| **VCO** | Voltage-Controlled Oscillator |
| **WS** | WebSocket |

# List of Figures

# List of Tables

# Listings

# 1 Introduction

Nowadays a very high demand on new products exists in the high tech industry. The trend is towards smaller and cheaper devices, which are able to perform tasks faster and have lower production costs than their predecessors. This leads to a growing amount of features that have to be integrated into shrinking part sizes. Decreasing the size of the product does not only save weight and transport costs, it is also more convenient for the customer to carry around. The shorter product life cycle [4] is related to the fact that the high tech industry is a fast evolving industry and products have a very fast obsolescence rate. The variety of products increases almost daily which leads to individualization of the products. The customer needs have to be fulfilled, so they can choose between the variations and choose their custom product.

In order to sell products, manufacturers have to decrease their development costs and deployment times for automated assembly systems. Two tasks, which take, among others, the most time to deploy, is the difficult integration and the assembly time to develop and establish the production line. The calibration and programming of the system accounts to a big amount of the setup time of a factory. Each of these trends represents different challenges for production lines. Especially in the high tech industry products are affected by most of these trends, some even by all of them. In this case some restrictions have to be made, since most common assembly systems can not meet all stated requirements and demands.

The Microdynamic Systems Laboratory (MSL) at Carnegie Mellon University (CMU) developed a new philosophy of factories, called the AAA. This philosophy is implemented in the Minifactory (see Figure 1.1) which tries to answer to the stated challenges. The Minifactory and AAA provide a platform that will support and integrate the different precision manufacturing processes, that need to be developed to assemble a large variety of small mechatronic products. With the modular structure of the factory it is easily manageable to manufacture a wide variety of products and test them on a single production line.

The Minifactory is under a constant progress of development. The current version of the system (version 2.0) is still in development. During the advancement process, different research projects to enhance the abilities have been done. Currently only one courier agent (see subsubsection 2.2.1) is fully functional.

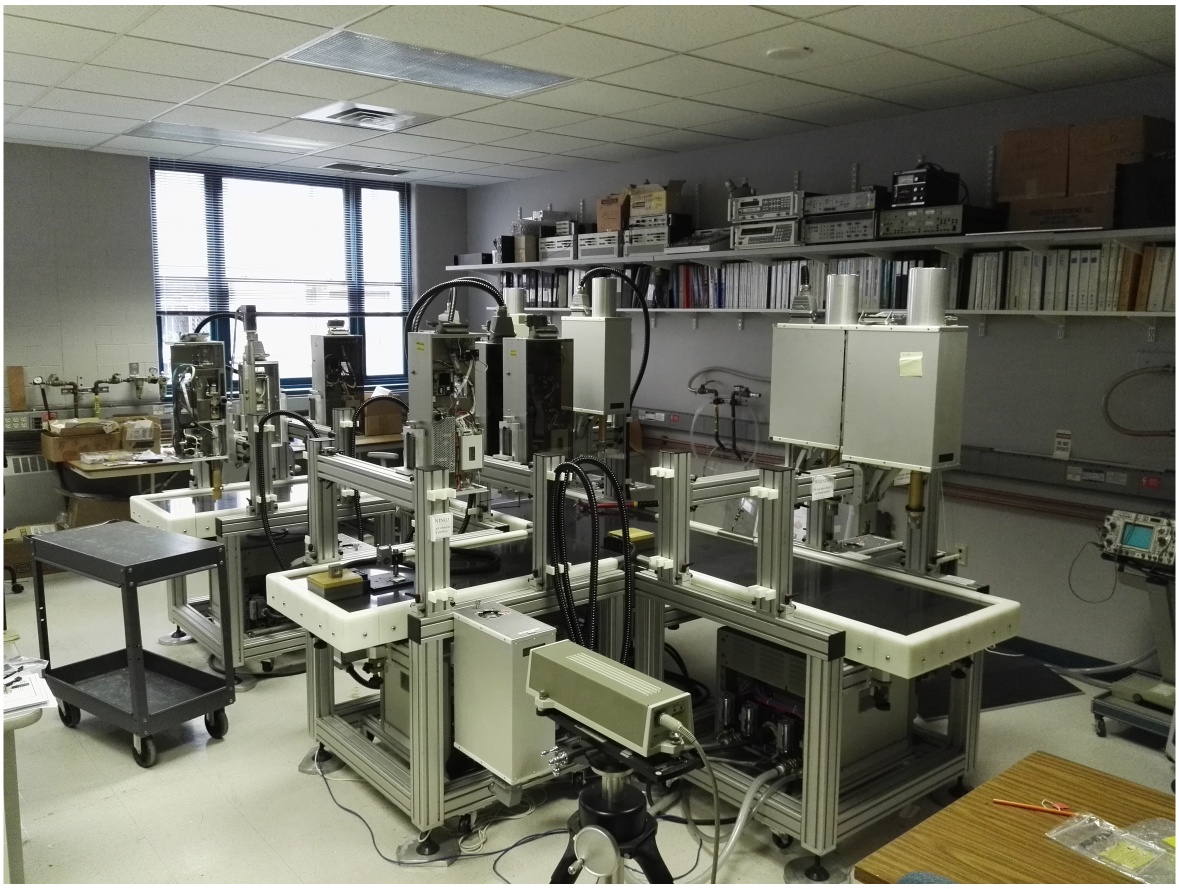


Figure 1.1: Minifactory at MSL

Ten manipulator agents are functional (see subsubsection 2.2.1), but they do not have end-effectors (see subsection 3.1.1) in order to work on a product. In order to get a better test setup, six more couriers as well as end-effectors for the manipulators are currently in production.

## 1.1 Objectives & Requirements

In this thesis, various enhancements for the 2nd generation of agent-based micro assembly system have been designed, implemented, tested and improved. Figure 1.2 shows a modular overview about AAA, the dark blue colored areas correspond to the work done in this thesis.

The AAA is divided into two main parts:

**Software:** The software entails not only the visualization of the factory, it also is part of the system where the whole factory is programmed and simulated.

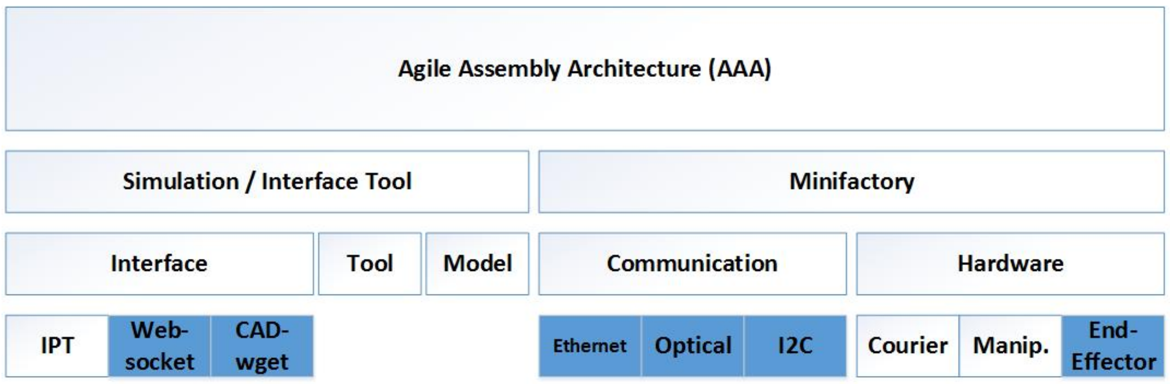


Figure 1.2: AAA modular overview.

A detailed overview about the Interface Tool is given in subsection 2.2.2. In this thesis the **websocket** (see subsection 3.3.1) and the **Computer-Aided Design (CAD)-model** import have been the the two main parts of the software module which were changed in order to accomplish the stated goals in section 4.4. Smaller changes were done in the other modules during the integration of the new software parts into the Interface Tool.

**Minifactory:** The Minifactory is an instantiation of an AAA. There are two main parts: hardware and communication. The hardware is defined by different agents (see subsubsection 2.2.1). One Part of this thesis was the development of an prototype **End-Effector** which is able to show the benefits of distributed computing and the modular structure of AAA, as described in subsection 3.1.1.
The communication part of Minifactory can be divided in several ways. Figure 1.2 describes the division by looking at the hardware on which the communication is happening. In this thesis all three methods of communication are introduced. $I^2C$ (see subsection 3.3.4) is needed in order to connect the end-effector to the correct manipulator. The **optical** communication (see subsection 3.3.3) enables courier agents to detect manipulators (section 4.2) and initiates the data exchange by sending the Identifier (ID) to the courier (section 4.3). Another part of this thesis is the modulation and demodulation of the ID on top of the optical signal, as well as the precision navigation which is needed for finding the manipulator agents.
Finally the main correspondence between agents is done on an **Ethernet** basis. In order to establish a real-time communication between to agents, the name of the agents needs to be known.

In this thesis an Ethernet based multicast framework (see subsubsection 3.3.2) was implemented which resolves the agent in the network and returns information about the connection methods of each agent.
The existing publish/subscribe framework was expanded (see paragraph 2.2.3) in order to offer more different communication methods between agents and also to update the status information and position of agents (see subsection 4.3.3).

This thesis project includes devising methods for tray feeding of precision high-value parts, micro fastening techniques such as adhesive dispensing and additional work on visual- and force-servoing. In order to help to achieve these functions, modular and reconfigurable robot end-effectors for handling parts, which are sized in the millimeter range, have been designed and built for existing robotic agents. In addition, new concepts for robot end-effectors to grasp and release tiny parts were required and needed to be implemented. This includes image processing and intelligent control software. These concepts need to differ largely from traditional handling paradigms in order to solve problems introduced by electrostatic and surface tension forces, which are dominant for manipulating parts that are millimeter and below in size.

The improvement of design, deployment and reconfiguration capabilities of automated assembly systems for precision mechatronic products is one of the main goals of the thesis. A strategic application for this approach is the automated assembly of small sensors, actuators, small medical devices and chip-scale atomic systems such as atomic clocks, magnetometers and gyroscopes.

This thesis aims to achieve the following objectives:

$R_1$ **Decentralized computation during operation**.
The whole factory needs to operate without a centralized knowledge base. Each agent should work independently, only communicating with other agents to participate in joint manufacturing.

$R_2$ **Initialization without prior knowledge**.
In order to initialize the factory each agent needs a calibration routine that enables the robot to determine the environment and other agents in the neighborhood.

$R_3$ **General approach to manipulating parts**.
End-effectors need their own computation unit which stores not only the CAD model of the end-effector, but also is able to do additional computation like image

processing and force sensing.

$R_4$ **Geographically independent control of factories**.
An AAA factory should be able to be controlled independently from the location of the operator.

## 1.2 Innovation

Assembly lines are highly complicated systems which need a great deal of precision in order to work properly. This precision can only be achieved if the system is setup with at least the desired precision which takes a long time. Setup times cost producers money since the place is filled with a new factory but this factory can't produce any products yet.

The Microdynamic Systems Laboratory at CMU developed a new philosophy of factories, called the Agile Assembly Architecture (AAA). With this framework it is possible to develop agile factories which are able to calibrate themselves and produce a variety of products in a small scale rapid prototyping environment. Assembly lines nowadays are designed for a specific task and changing the factory for another product can take a long time and often requires reconstruction of parts of the factory. With AAA the factory can be reconfigured on the fly since the manipulator robots are not configured in a line working at a conveyor belt but instead use carrier robots which transport the product from one step to another. The testing factory which is designed at CMU is called Minifactory.

The Minifactory and AAA provide a platform that will support and integrate the different precision manufacturing processes, that need to be developed to assemble a large variety of small mechatronic products. With the modular structure of the factory it is easily manageable to manufacture a wide variety of products and test them on a single production line. Since the system is designed to produce chip-scale devices with a precision of $200nm$, the location of the manipulator robots needs to be known with at least the same precision. In order to achieve this goal a new initialization method was developed which finds the manipulator robots with help of the carrier robots. Through help of different communication channels the robots are able to find each other and establish a relationship which is able to build the whole factory. This enables user to simply state which manipulator should be used in order to achieve the desired operation. The system will plan all movement between stations and prevent collisions between different courier robots.

This helps to shorten the programming phase of the assembly system. Another step to re-usability is the development of end-effectors which attach to manipulator robots. With help of these end-effectors it is possible to use a manipulator for different task, while only changing the end-effector which includes the part of the robot which does the product modification. The high precision motors which are needed in the factory are in the manipulator robot. This systems saves money since only small parts of the factory needs to be changed in order to enable the assembly line for a new product.

In summary this type of factory and specifically the in this thesis developed auto calibration methods will help to shorten the setup time and thus the overall costs of factory systems in small scale setups. By having an agile and reconfigurable system it is possible to have a faster development phase of new products since many different products can be tested in a short amount of time without the need to rearrange the hardware in the assembly system.

To the best of the authors knowledge there are no smart factories with this high precision which include auto calibration and such ah high degree of agility.

## 1.3 Structure

This report is composed of several chapters. chapter 2 gives an overview about the AAA as well as the instantiation in form of the Minifactory. Furthermore this section explains recent developments in smart factories from different universities. The expansion of the framework and Minifactory during this thesis is described in chapter 3. The process of detecting an agent of the Minifactory, instantiate communication to the agent, as well as updating or generating the agent inside the Interface Tool is described in chapter 4. The validation of these instantiation algorithm is described in chapter 5. chapter 6 describes the scientific contribution of this thesis. Finally chapter 7 summarizes the outcome of this report and gives a conclusion as well as an outlook for future work.

# 2 Smart Factories

In order to reach the in section 1.1 stated requirements a through understanding of smart factories needs to be established first. Thus this chapter introduces first current developments in the area of smart and agile factories. Additionally the proceedings at CMU are described and the current system in form of the Minifactory at the beginning of this thesis is presented. This gives an overview of the needed knowledge by explaining terms and definitions of relevance for the following work.

## 2.1 Existing Approaches to Agile Manufacturing

In recent years there have been different approaches for designing and building smart factories.

The desire to implement a modular and agile environment lead to the development of so called Reconfigurable Manufacturing Systems (RMS). On approach by the University of Applied Scienes Utrecht is a five staged development process where the RMS is indexed by using "'Qualitative Analysis' based on 'Coding' combined with the method of 'Structured Analysis Design Technique's."[5] This design based approach deals with ways to integrate a product into a better development and production factory. Since this approach deals with the early phases during the development the factory needed can be adjusted before the production of the product starts.[5]

Smart factories have to begin during the designing phase of a product. Especially the project management has to adapt to agile factories. The technical university of Prague is researching agile frameworks which enables the user to manage products in an agile way. This transition from software design to physical products enables the usage of well known methods in a new field and thus improves modern product design. [6]

Another way of achieving a smart factory is presented by the University in Clermont. The work describes a formal framework which is able to classify not only physical system by their characteristics but also literature in the field of factories and agile assembly. The main focus is on system characterization which enables a production system to contiguously modify the behavior and and thus fit into changing environments. The concept of a formal framework allows mathematical descriptions of production processes. This enables users to integrate different systems within each other and produce a single description throughout whole companies by combining different factories. [7]

Interoperability and dynamic assembly are two very important keywords for smart factories. Therefore special considerations need to be given to a modular communication framework which connects different parts of a factory. The usage of a Service-Oriented Architecture (SOA) allows the communication of factories based on the offered services which enables factories a fast deployment of different interactions between autonomous devices. By using WebSocket (WS) a standard protocol is implemented which allow Point-to-Point (p2p) between embedded devices. The Loughborough University researches this approach by using standardized messages between parts of the factories. [8]

Smart factories are not only dependent on the programmed software but also on the hardware components. The Technical University Munich researches a framework which allows agile manufacturing processes. The focus relies hereby not only on the software which runs the factory but also on the interchangeability of the hardware and the connection between those two parts of a factory. Another part of this research includes the human as part of the system and thus creates a closed environment. The whole system is divided into subsystems which can run autonomously and only connect at specified exchange points. This allows the whole factory a high level of agility since each subsystem can easily be exchanged with another part of the factory. [9]

The different research projects all have advantages and disadvantages. Neither one offers a whole process which enables an user to stay within one framework during the design and operation phase of a smart factory. Single research projects concentrate on the design process, other at the integration of software and hardware. The MSL at CMU took different approaches and tries to integrate them into a single framework which is explained in the following section.

## 2.2 Agile Assembly Architecture at CMU: an Overview

The CMU developed a new concept about assembly architecture in 1995. The Microdynamic Systems Laboratory presented this forward-looking approach in form of the AAA which is an approach to meet the demands on automated assembly systems from the fast evolving market [10].
These characteristics are achieved by a modular structure of the assembly system with reusable elements, which can be dynamically reassembled to a new system. The idea of AAA resembles building blocks in a construction kit which offers standardized basic modules guaranteeing a high reusability and adaptability to diverse applications easy and quick.

This is achieved by standardizing data protocols as well as mechanical and electrical interfaces. Thus an arbitrary combination of all elements is offered which also allows a possible future extension of the assembly factory [10].

The main feature of an AAA system is the decentralization of the computing power. Each robotic module is independent knowing about the capability of the agent only. This offers the advantage that during operation no central control unit is needed. Each agent communicates to the other agents over network.

Part of the AAA philosophy is the symbiosis between the real assembly system and an identical virtual version for simulation. With the help of the Interface Tool [11], a basis for the virtual environment as well as a platform for displaying the behavior of the robotic modules is provided. The robotic modules get their information from the real hardware that provide geometrical models, their abilities and the communication interface in order to control them. Within the Interface Tool a virtual factory can be designed, programmed and tested with actual module specifications. These specifications can be loaded directly from the existing modules in the factory or even from the Internet. In order to simplify the process of generating a virtual system, a library of routines is provided as well as a standardized communication protocol which helps to develop an assembly system faster.

The modular construction of an AAA system and the standardized interfaces allow an easy setup of a real assembly system. Since there is a close relationship between simulation and real hardware, the tested (simulated) program can be easily changed and uploaded to the robotic modules. This transition is supported by the agents capability of self calibration. Since couriers have the ability to explore the environment, the alignment of the system elements can be determined by the agents as well. This leads to short setup times and the possibility to change the system fast.

The following sections provide an overview of the different aspects of AAA. The instantiation of this concept is explained in form of the Minifactory. In this chapter the different parts of the factory are described. An interface provides a system to program the whole factory in a virtual environment, it also opens the possibility to run simulations in this environment. In order to have an agile system the communication between different agents has to be self-describing and each agent must be able to connect to other agents without the need of a programmer.

### 2.2.1 Minifactory

Minifactory is a tabletop system which was designed by the MSL at CMU. This modular assembly system is an implementation of the AAA philosophy. It is able to produce products, like position sensors and small microphones, in the range of millimeters till up to some centimeters and with the precision of micrometers. In order to get a modular system which can be easily assembled for a multitude of different products the whole factory is divided into different parts. These parts are described in the following sections [12, 1, 10].

#### Base unit

The main operation platform is the *base frame* module which is shown in Figure 2.1. This unit has a platen tile that is mounted on top of this module which serves as the floor for moving agents. The platen tile features a very tight grid (1mm) of ferromagnetic posts which is used by the moving agents (see subsubsection 2.2.1) for navigation and orientation. In order to provide a barrier at the edges of the base module, each unit has curbs made out of polyethylene. This barrier prevent the moving agents from falling off the Minifactory.

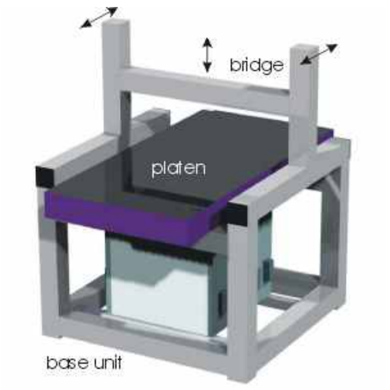


Figure 2.1: Minifactory: Base module provided by [12].

In order to mount manipulator agents (see subsubsection 2.2.1) on top of the module and to connect different modules, aluminum profiles are used for connections and stability. As is shown in Figure 2.1 these aluminum profile bridges are movable and can be placed where a manipulator is needed. A central unit has the ability to serve up to eight agents with power, network services, vacuum and pressured air. The power is already provided with standardized voltages serving the different needs of the agents, which reduces the needed amount of voltage regulation in each agent.

There is a 100Mbit network provided, using standard IP protocols and real time capabilities for communication between the different agents [13].

### Agents

A robot communicating and assembling in the Minifactory is called an agent. Each agent has its own processing unit and works autonomously in an AAA environment. It connects to other agents over a network. There are different types of agents in the Minifactory. Each agent has a low Degrees of Freedom (DoF) factor that is combined with other agents by communicating with them. The DoF can be combined and more complicated assembly tasks can be achieved this way. The different types of agents can be categorized as robot manipulators and mobile robots.

A robot manipulator consists of a sequence of rigid bodies (*links*) and articulations (*joints*) which interconnect the links. This sequence represents the arm of a manipulator. In the Minifactory there are *open kinematic chain* manipulators, which means that there is only one connection between the base of a robot and the end-effector which performs the required task of the robot. The amount of movement is defined by the length of the arm and the DoF of the robot.

Unlike the robot manipulator which has a fixed base, the main feature of a mobile robot is the ability to move its base freely in the environment. The main usage of this type of robot is servicing parts between manipulator agents inside the Minifactory. Generally a mobile robot can be classified in separate categories: wheeled robots, legged robots and planar robots. The most common type of mobile robots are wheeled robots. Minifactory uses planar robots which move around the factory on an air bearing [14, Chap. 1].

### Moving Agents

The moving agents, which are called couriers, have two different tasks in the Minifactory:

The first task is the transportation of the product in the Minifactory from one manipulator agent (see subsubsection 2.2.1) to the next until the product is finished. Conventional assembly systems often use conveyor belts or similar systems.

These systems are unidirectional and it is very hard to insert new production processes into an existing factory street. The Minifactory uses the courier agents to transport the product. Since these agents can move freely in $x$ and $y$ direction there is no need for a serial line up of the assembly stations. The stations can be arranged parallel in order to save space. Also the expansion of a factory with a new manipulator agent does not require any mechanical reconfiguration of the existing factory.

The second task of this type of agents is the positioning of the product under a certain manipulator agent and the following processing for the stated task of the manipulator agent. This cooperation is the solution for the problem that the small number of DoF is insufficient for most assembly tasks. The courier agent offers three DoF [15] which can be added to the DoF of the manipulator agent. So it is possible that by combining a courier with a manipulator agent, complex processes that require more DoF can be performed [14, Chap. 1].

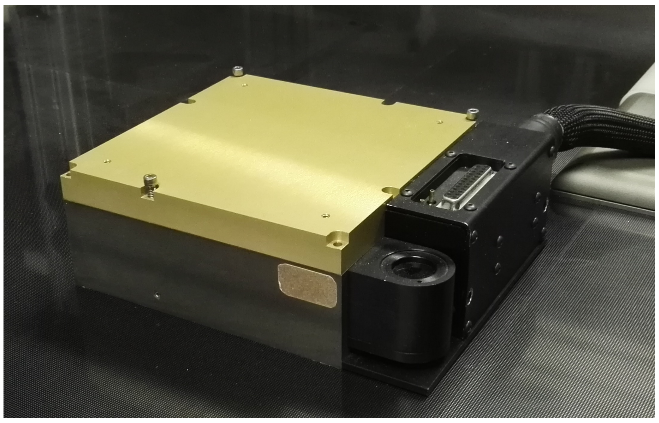


Figure 2.2: Minifactory: Moving agent provided by [12].

The courier agent consists of two parts as shown in Figure 2.2. The brain box is the heart of the agent. It contains most of the electronic and processing hardware which is needed to operate the agent. The brain box is connected to a base module by using a standardized connection cable.

The second part of the agent is the courier unit. This unit is connected to the brain box by a multicore cable which includes vacuum and pressured air. This cable also sets the boundaries of the movement of the courier in its length since it is the maximum distance the courier can move away from its brain box. In order to move the courier on the platen tile an air bearing is generated which raises it to an altitude of $10 - 15 \mu m$ above the factory floor.

The cube is moved by using four planar stepper motors which use the Sawyer principle in combination with the grid of ferromagnetic teeth in the platen in order to shift the agent in $x$ and $y$ directions. The positioning which is able to detect the location of the cube at a precision of $0.2\mu m$ is provided by a platen sensor that uses the magnets in the baseplate. This enables a closed loop system to control the position and speed of the agent with a maximum speed of $1.5\frac{m}{s}$. [11, 1, 15, 2, 16, 17]

In order to be able to establish a network communication between a manipulator and a moving agent, there needs to be a way of discovering each other. An optical coordination sensor mounted on the courier offers the ability to listen for beacons sent out by the manipulator agent. Furthermore it is able to measure the relative distance between manipulator and courier for calibration up to a resolution of $0.15\mu m$. The exact way of establishing a first communication channel is described in subsection 3.3.3 [18].

### Manipulator Agents

The second type of agents are manipulator agents (see Figure 2.3). These agents are mounted above the factory baseplate on bridges which allows them to be very flexible mounted across the Minifactory. The manipulator has two axes which allows two DoF. A vertically mounted Z-axis allows the movement of up to $150mm$ range whereas a rotary axis can move it around $\phi$ with 330°. Since a conventional assembly robot (e.g. SCARA-robot [14, Chap. 1]) has a common DoF-factor of four it is mandatory for the manipulator to communicate and work with the courier agent in order to get this kind of freedom in movement. Since this reduces the number of DoF by half the operation can be executed with a much higher precision. Another advantage of two separate robots working together is the assembly speed. Since the manipulator can start to work even before the courier has reached the final position for assembly, there is an increase in the assembly speed for the assembled item [19, 1].

In order to be modular and flexible in the mounting the manipulator has its own brain box which is inside the manipulator. The same connector which is used to connect the brain box of the courier to the base module is used to connect this agent. The brain box includes its own processor, which allows the function of the agent to be independent from any outside input. The manipulator offers a standardized interface at the end of the vertical axis which allows different kinds of end-effectors (see subsection 3.1.1) to be mounted on the manipulator.

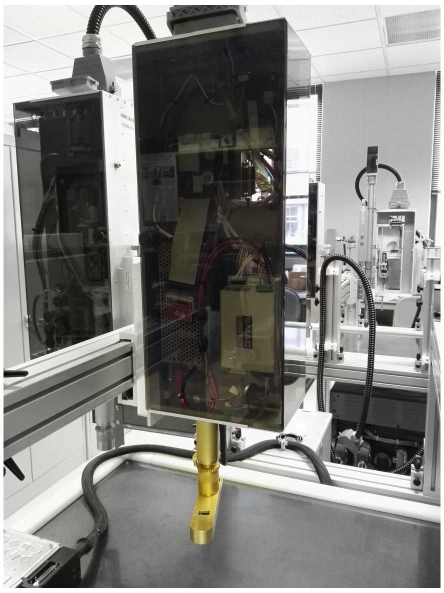


Figure 2.3: Minifactory: Manipulator agent provided by [12].

A quick connector offers a fast and easy way to change the utilization of the manipulator. The interface can be seen in Figure 2.4.

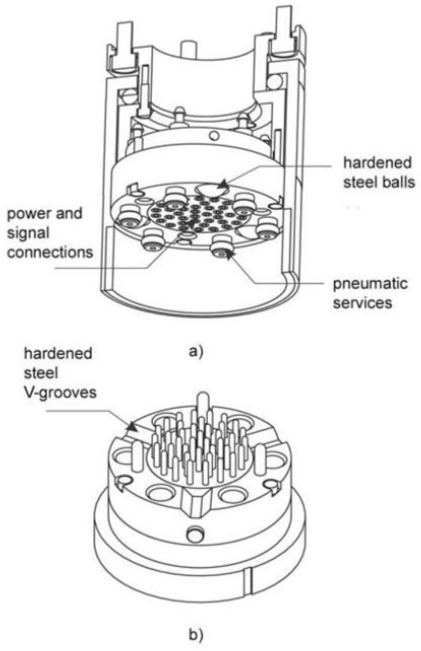


Figure 2.4: Minifactory: Manipulator interface for end-effectors provided by [12].

#### Operating System

In order to have a fast assembly system the Operating System (OS) on each agent needs to be fast, but accurate as well. The only way to guarantee the fixed task plan is by using a Real-Time Operating System (RTOS). There are several different types of computation units in the Minifactory. Since each courier and manipulator agent is a standalone application every single one of them needs a processor. QNX was chosen as RTOS. The Unix derivative offers a micro-kernel design and a modular architecture which makes the whole operating system small and flexible. Especially in the courier agent real-time processing is needed to guarantee that not only the closed loop control, but also the motor control works as desired. [20]

### 2.2.2 Interface Tool

Common development of assembly systems is usually divided into two stages. The development of the software and the hardware usually happens in parallel. Sometimes the software is developed even before the hardware is built. It is normal that a simulated environment is used to program the software offline. After simulation the results are used to integrate software into the physical machines. Traditional online systems like Supervisory Control and Data Acquisition (SCADA) and Human Machine Interface (HMI) do not have the ability to simulate the hardware. As such there is a gap between offline and online systems that often results in two different software environments used during the design, deployment and operation phase. AAA offers a combined solution in the form of the Interface Tool. It is used for planning and creating virtual factories as well as real factories [11].

Having a virtual model of the physical factory can decrease the setup process of an assembly system since the software part can be tested and errors can be fixed even before there is a single piece of hardware. Often the physical factory is only assembled at the customer location. Thus in common proceeding there is a need to fix errors at the customers site which takes time and creates a bad image for the company. Errors during the planning phase can be very expensive when they lead to time- and money consuming modifications of the real system. Simulations can show these errors before the physical factory is built and thus avoid these costs. Another useful feature of virtual assembly systems is the avoidance of downtime during programming of new tasks. If the program can be written offline, the physical system can still run and work on a current task.

There is even the possibility to work on multiple new tasks at the same time since the virtual factory can run parallel on multiple instances. Also testing the new task in a simulation is not dangerous and cannot cause damages to the physical assembly system. This feature is only useful if the virtual factory is nearly identical to the physical one. Otherwise there is a time loss when deploying the code to the physical assembly system since parameters would need to be changed in order to work with the hardware system.

Traditional simulation systems need a high degree of programming in the simulation part. The programmer needs to have a very good knowledge of the physical assembly system in order to build the virtual one. Mechanical and physical proceedings of the system needs to be known and modeled as mathematical equations. A factory is not a simple system and basic simulation software is often pushed to or beyond its limits. As such the timing factor in the development of a factory is a big factor in the success and cost-effectiveness of the assembly system.

AAA Interface Tool offers features which allow both simulation and hardware control. As such the program guarantees the highest possible correlation between these two worlds. There are two possible ways the Interface Tool can be generated. The usual way of implementation is the top down approach. A model is developed and programmed in the Interface Tool which can be used as a base for generating a simulation. After testing the model and programming the agents, the physical system is connected to the Interface Tool. The second possible way is the auto generation of the interface model. In this mode the courier agent generates a map of the factory. After finding a manipulator it is inserted into the Interface Tool. A complete model is generated and can be used without the need of drawing the model itself, after the whole factory is mapped. This option is only available for existing hardware factories.

All physical models are registered in the Interface Tool component palette. This includes information about the body structure, abilities, ways to connect to the interface of an agent and the connection to other components of each module. Each agent has its model stored internally which can be downloaded to the Interface Tool. Changes in the model can therefore be easily implemented in the Interface Tool. It is also an easy way to introduce new models to the Graphical User Interface (GUI) [11].

In order to assure a high correlation between the two systems during the complete lifetime of an assembly factory, the Interface Tool offers bi-directional transitions between simulation and reality.

This means that any change in either one of the two systems can be transferred to the other system. Most common simulation systems allow only the transition from simulation to reality once, during the construction phase of the factory. With Interface Tool the full potential of simulated factories can be utilized.

Since real agents also have the capability of self calibration, it is possible to change the layout of the Interface Tool accordingly. This helps to close small gaps between the simulated and the physical factory. This means it fulfills a requirement of AAA: **a high degree of agility**.

There are two different environments inside the Interface Tool. First, the *tool* environment provides a GUI for the physical interaction with agents and shows the current status of the whole factory. The second one is the *sim* environment. Inside this mode it is possible to program, design and test a virtual factory. Both operation modes offer a similar GUI and have access to the same information about the factory components, products and their description. The *sim* environment offers additionally the ability to take photos and record movies of the simulation.

#### Structure

The Interface Tool is structured in a vertically layered architecture. The GUI is the highest level and therefor the visual part which interacts with the user. It allows the construction of factories in a very convenient way. The models of the required components can be selected in the component palette. These models include the graphical description of them in the form of Open Inventor files[1] as well as a description of the available command interfaces and how to access them. Since the software has a graphical interface it offers all benefits such as menu bars and dialog boxes. The top level also offers the user a three dimensional rendering of the running factory in both operation modes, either in the simulation or real time execution. The engine offers six DoF in the viewing angle. Changing the viewing angle, regulating the execution speed of the simulation as well as zooming in is possible. Since the whole factory is designed as a decentralized environment the Interface Tool is not required for running the assembly process. After programming the factory the parts are uploaded to the agents. The GUI can be used to monitor the progress, as it processes the data in real time and updates the view accordingly to the monitored devices in the factory.

[1]more details: `http://web.mit.edu/ivlib/www/iv/files.html`

There are two different programming languages in the Interface Tool and the agents. The basic and low-level functions as well as the display are written in C++. The logical programs such as the run-time behavior for the simulated and physical factories are written in Python. This programming language is an object oriented programming / scripting language which stores the executable file as byte code. In order to get an “easy to use” programming interface, Python encapsulates the lower level C++ code and offers the user all required methods. The code behind instantiates the needed objects which offers a *bind* method describing all required global factory components. A *run* method states the actual script which defines the runtime behavior of the agents [21].

The GUI already provides the basic layout of a Python program while assembling the virtual factory. The chosen components building the factory are included with their specific position matrices and their parent-child-relations in the Python file by the Interface Tool. The programer has to add the functionality and desired outcome of the agents (the *run* method) to the existing script. These command lines can be put together by using the library which is entailed by the chosen type of robot agent. Each assembly task of a specific robot is included with the required methods in this library. The Python library encapsulates the source code which is written in C++ and establishes the ground functionality of an agent. This offers the benefit that a user only has to know one programming language while using the Minifactory. Since Python is an easy to learn and use language this reduces the programming effort since only short blocks of code have to be written.

### Class Hierarchy

Every component in the Minifactory is based on the description class. This overall class is the base to all products and components as shown in Figure 2.5.

The class *ComponentDescription* is the base class for all components which are needed in the factory. This includes static components such as platens and frames as well as agents which occupy the factory. Each of these components has an own class describing its features and connection properties. Each single agents is derived from the class *AgentDescription*. In this class there exists an interface component which holds all information about the communication between the Interface Tool and the agent. There are different types of agents such as courier, manipulator and end-effector agents. Each type of agents has its own description class.

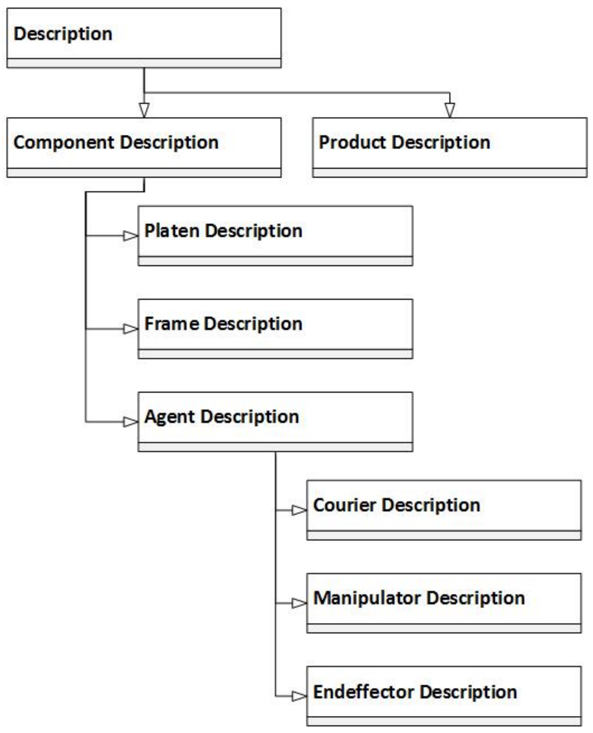


Figure 2.5: Class hierarchy for the description of factories.

Every product that is needed in the assembly process is presented by the *ProductDescription* class [11].

### Design

While designing a factory there are two separate types of hardware which have to be generated. Basic modules, like platen, curbs, bridges and base units are reusable in almost every kind of factory. As such they can be imported into the system from the database. Agent modules on the other side needs to be modified for each task in many instances. In most cases it is enough to extend standard courier and manipulator with the required tools, like specialized end-effectors and mounting kits for the couriers. Since the end-effectors are custom made for the specific task of the assembly system they are often shipped with existing CAD drawings. These drawing can be used in order to get a virtual model integrated into the Interface Tool.
If there is no existing CAD file it has to be created before a model can be inserted into the GUI. Since there is no internal representation of the model needed, a high level of abstraction is allowed in the design process. As such only the outer shell and essential details have to be inserted into the drawing. This procedure has to be fulfilled for each part of the factory.

This includes agents and basic modules as well as all component parts and products which get assembled in the Minifactory.

In order to use a CAD generated model in the Interface Tool the file has to be transformed into a Inventor file (IV) format. The Interface Tool uses the Metric system to display the environment. If the CAD file was not generated in this system the IV file needs to be adjusted to it. During this process there is also the ability to change the material and color in order to have a better representation of the model. Using a transformation matrix enables the programmer to change the size, rotation or position of the model. Since the Interface Tool can only load information from Python files the converted IV files have to be included in Python files which offer the possibility to add additional information like a preview picture or a transformation matrix for the position of the model.

The design of the factory is done inside the GUI. After loading the required components from the component palette the factory designer can link the parts together. This is done by selecting two parts with the cursor and activating the assemble command. Since all possible connection locations are hard coded into their C++ files it is impossible to generate invalid connections. In order to get the right result, the position and orientation of the parts have to be as desired since there are several available connection locations. This alignment can be achieved by switching the GUI from selection into moving mode.

If parts of the assembled product should be displayed at the start of the program on the courier their IV files have to be placed inside the courier's Python file. Another way of integrating parts in the production process is to place the IV files inside the main program of the Minifactory. This is the case for products which should appear during the runtime of the assembly process. These two alternatives are needed if the model is part of the product and not of an agent or basic module.

The final step of the agent design is the generation of a static 3D image of the Minifactory.

### Programming

After designing the Minifactory, a basic structure of the program is already built automatically by the Interface Tool. The relationships between modules are shown as parent-child inheritance inside the factory file.

Those files are stored in the *model* directory of the Minifactory and have the ending *.fac*. This includes the platen to base unit relations as well as that one between manipulator and bridge. The position of each child member is relative to the parent and described as a transformation matrix. Since the programing language for this part of the Interface Tool is Python any text editor is suitable to program the behavior of the factory.

The assembly movements are coded as part of the member fields of an agent. Besides a name for an easier identification of the agent, the whole assembly task is inserted into this Python file. The instruction set for each agents is defined by the implemented interface (see Figure 3.2). In order to generate an "easy to use" language for the AAA a very high level approach was used while designing the command set.

There is a difference in coding for the *sim* and *tool* environment in the Interface Tool. Although the general structure of the Python file is identical, the commands differ for the agents. The real hardware agents have specific commands for controlling the motors and valves. Since the virtual agents do not have to care about the exact behavior of those, the commands are simplified in the *sim* environment. This leads to an easier programming of the virtual factory, which is why the *sim* environment is more convenient and faster in the development of a new factory. Therefore this tool is ideal for testing new agents and implementing fast prototypes. The downside of the separation of commands is that a direct translation from *sim* to *tool* environment is not possible.

### 2.2.3 Communication

This section explains the different mechanism for communication in the Minifactory. The main method of communication is done over Ethernet in a global IPv4 network. The network has different methods as well depending on the communication partner. If it is a user interface an agents provides a websocket. The agent to agent communication is done over publish / subscribe methods. The network is divided into two different subnets.
Each courier agent is equipped with two network cards which support a seamless cooperation between the agents. The first connection is to a standard network which carries non-latency-critical information such as debug and status information as well as commands to and from the Interface Tool and between manipulators and courier. This network utilizes standard IP protocols. In order to have a reliable realtime communication between the courier agents there is a second network connected to them.

The AAA-Net carriers real-time information which are needed in order to coordinate the moving operations between different couriers [22, 23].

### Websocket: GUI-Website

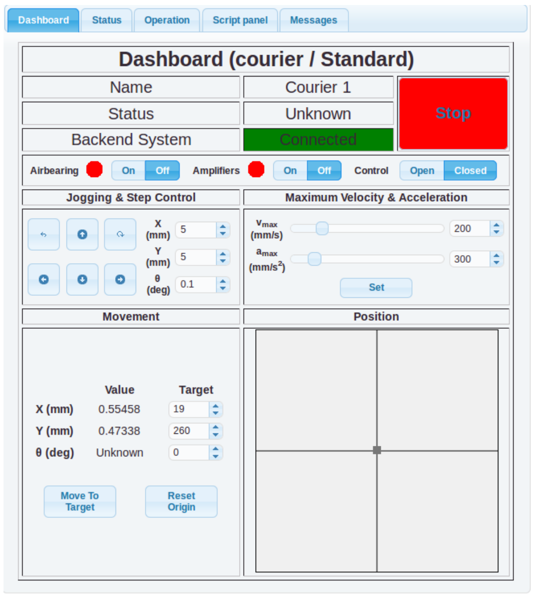
A standard website loads data from a server and displays it to the client. If the clients wants to interact with the server a new request is sent to the server which will result in a new (modified) website being generated and downloaded by the client. This scheme takes time and redundant data that is transmitted over the network. If there is a website which needs frequent update sent to the client as well as giving the client the opportunity to interact with the server there is another method of communication.

A WS is a protocol which offers bidirectional full-duplex communication between server and client. Furthermore this channel can be established by using only a single Transmission Control Protocol (TCP) connection. While the protocol is independent from a standard Hypertext Transfer Protocol (HTTP), its handshake is still interpreted as upgrade request by the server.

In Minifactory a WS uses port 80 for communication but it can be any other port as well [24].

In order to get the actual data from an agent a GUI needs to have a constant exchange between GUI and the program running on the agent. Minifactory offers for each agent a simple GUI in the form of a website display the status and simple control operations. The actual data and the commands are sent over TCP-WSs in both directions. This enables a platform independent survey of the agent state. In a real factory this will enable the user to control the agent while going through the factory with the help of a tablet. Since simple operations, like manually driving an agent and testing its functions, is also possible the GUI represents a simple and efficient communication between user and factory agent.

The GUI of the agents is specific for each type of agent. The dashboard gives a overview about the status, type and name of the agent as well as controls for testing the agent. It is shown in Figure 2.6 for both the courier and manipulator agent. This board is divided into several sections. On the upper part there are general information about the agent such as name, type of the agent, general status informations and the state of the backend system. Furthermore this section offers an emergency break button which stops every movement in cases like collisions or wrong test configurations.



Dashboard (courier / Standard)

| | |
|---|---|
| Name | Courier 1 |
| Status | Unknown |
| Backend System | Connected |

Airbearing ● On Off — Amplifiers ● On Off — Control Open Closed

Jogging & Step Control: X (mm) 5; Y (mm) 5; θ (deg) 0.1

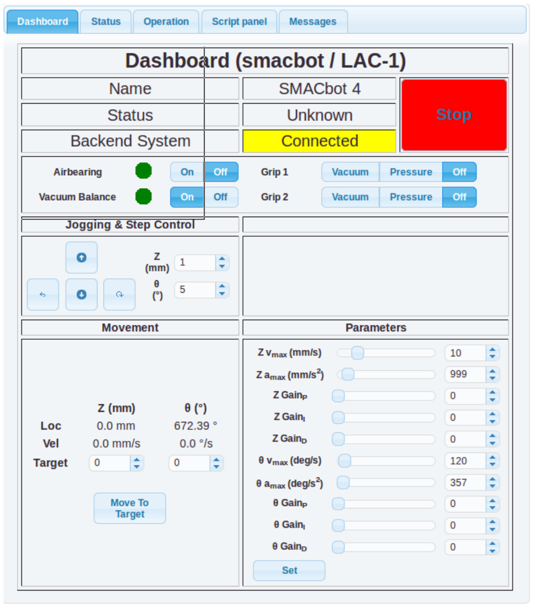
Maximum Velocity & Acceleration: $v_{max}$ (mm/s) 200; $a_{max}$ (mm/s$^2$) 300

Movement

| | Value | Target |
|---|---|---|
| X (mm) | 0.55458 | 19 |
| Y (mm) | 0.47338 | 260 |
| θ (deg) | Unknown | 0 |

Position

(a) Courier GUI



Dashboard (smacbot / LAC-1)

| | |
|---|---|
| Name | SMACbot 4 |
| Status | Unknown |
| Backend System | Connected |

Airbearing ● On Off — Grip 1 Vacuum Pressure Off

Vacuum Balance ● On Off — Grip 2 Vacuum Pressure Off

Jogging & Step Control: Z (mm) 1; θ (°) 5

Movement

| | Z (mm) | θ (°) |
|---|---|---|
| Loc | 0.0 mm | 672.39 ° |
| Vel | 0.0 mm/s | 0.0 °/s |
| Target | 0 | 0 |

Parameters

| | |
|---|---|
| Z $v_{max}$ (mm/s) | 10 |
| Z $a_{max}$ (mm/s$^2$) | 999 |
| Z $Gain_P$ | 0 |
| Z $Gain_I$ | 0 |
| Z $Gain_D$ | 0 |
| θ $v_{max}$ (deg/s) | 120 |
| θ $a_{max}$ (deg/s$^2$) | 357 |
| θ $Gain_P$ | 0 |
| θ $Gain_I$ | 0 |
| θ $Gain_D$ | 0 |

(b) Manipulator GUI

Figure 2.6: GUI of the agents.

The next section offers general settings such as the overall vacuum and pressured air of the agent as well as amplifier circuits depending on which type of agent the GUI is representing. For manual operation and testing of the main functions of an agent, the system offers buttons in the GUI for the basic movements like X and Y for the courier agents. Z and Θ movements are provided for the manipulator agents as shown in Figure 2.6b. In this section the overall parameters such as maximum velocity and acceleration for the different axis and parameters like proportional, integral and differential of the motor controller can be changed here. The last section offers movement to specific coordinates. This section uses the internal controller algorithm in order to reach the stated goal by using the provided controller parameters. If the agent is a courier robot this sections also offers a map showing the couriers position as relative position to the point of the agents origin as is shown in Figure 2.6a.

Another part of the GUI is the status and information center. In this view different information about the agent as well as its sensor status are displayed. For courier agents this also includes the current power which is consumed by all eight coils and the raw readings from the platen sensor. The manipulator agent displays the status of the servos such as the temperature, the error state, the trajectory and the current direction.

Every agent has a script view. In this view different scripts for testing functionality and snippets for productive can be programmed and tested. This scripting can be done without the need of an Integrated Development Environment (IDE) and the need to move the data from the development PC to the agent. The scripts can be directly programmed, stored and tested inside the agent. These snippets can later be used in the actual production code as tested fragments. That way it is easy to program new functionality into the agent without setting up an IDE, which leads to a faster development.

Lastly the GUI offers a message view. Inside this view messages from the agent during operation will be showed. Depending on the debug level of the operating agent the output of the message window can differ in the granularity which represent the operation status.

#### Publish / Subscribe

Publish / Subscribe is a messaging pattern used in software architecture where the receiver is not known during programming of the sender. Therefor the sender of a message, called publisher, characterizes the message in such a way that there is no need for a specific receiver. There can be no, one or many receivers to a particular message. The pattern introduced by the Gang of Four (GoF) is called Observer pattern and is part of the behavioral design patterns. A receiver, called subscriber, does not know the specific sender of the message instead he has a public interface to which he can subscribe to the type of message he wants to receive. This offers a loose interconnection between the two agents. If the messaging is only inside one program the architecture can be handled by different classes inside the same program. If there is a need for interconnection between different programs on the same PC or over the network there are often machine-to-machine middleware (m2m) systems involved. They handle the sending and receiving over the network. This provides the programmer the freedom to concentrate on the programming of the messages instead of taking care of the network part of the transaction [25].

**Data Distribution Service** One standard, which fulfills the properties of a m2m standard for network communication, is the Data Distribution Service (DDS) which was developed by the Object Management Group (OMG).

It is designed for real-time systems with high performance and interoperable data exchanges as well as scalability between publisher and subscriber. There are different implementations of DDS that are open source as well as as commercial. Minifactory implements the DDS solution from Real-Time Innocations, Inc. (RTI)[2] using a commercial license. A Message is marshalled by wrapping the content into a JavaScript Object Notation (JSON) format which allows the exchange of multiple variables in one single message. The middleware handles all connectivity requests and routes the messages through the network by using plain User Datagram Protocol (UDP) multicast standards. By implementing Quality of Service (QoS) specifications the message can be prioritized and thus critical system messages are always delivered in the fastest possible way while info messages can be delayed in order to control the network flow. In order to limit the amount of traffic, a message is only sent if there is at least one subscriber to the message. [26, 27]

Minifactory uses DDS in order to subscribe to agent events. This is mostly done by the interaction between manipulator and other agents, like end-effectors or courier. Certain messages like the position of an agent have their own encapsulation and methods in order to standardize the workflow between the different agent programs and thereby makes it easier for a programmer to get the factory running. After the initial communication is established the corresponding agent can be contacted by the exchanged parameter. The initial communication is established by using other communication paths like $I^2C$ and Infra-Red (IR). The initial communication data is exchanged by using Lightweight Communications and Marshalling (LCM) in order to find the right agent in the network (see subsubsection 3.3.2). Each agent has to have an unique name in the DDS network. This name is used by other agents to subscribe to messages and get updates from the agent. Messages can either be sent as single value messages (*monotone*), by calling a pre-defined encapsulated method like the position of the agent (*position*) or be sending a self generated JSON dictionary (*raw*) to the subscribed agents. Depending on the chosen method of sending the message, the subscriber has to know how to extract the data from the message.

**AAA-Net** AAA-Net is a low latency 100MBit Fast Ethernet based communication protocol developed by MSL in order to fulfill the needs of Minifactory. In order to achieve inter agent coordination of courier agents and synchronize possible configurations there are a number of requirements which AAA-Net needs to satisfy.

[2]more details: `https://www.rti.com/products/dds/`

One of the main features which have to be fulfilled is a low latency. Sharing sensor and actuator data across courier agents requires the support of a real-time network. The longer the transmission is delayed the less information it can provide to other couriers. Since Minifactory is a dynamic system with varying amount of agents connected to it, the network needs to support scalability. The network is able to provide low latency in a small network supporting only a few agents as well as in big networks supporting many agents operating at the same time. The network consists of standard Ethernet hardware. In each base unit a hub connects the agents in a star topology. A relay switch connects the different base units filtering data packets destined for local agents and allows scalability to the Minifactory.

AAA-Net is running on a Fast Ethernet network. In order to compare it to an IP based network the Open System Interconnection (OSI) reference model is used. IP based protocols cover a wide range of applications. Thus the IP implementation is only part of OSI layer 3 and other protocols build on top of this generating different applications. AAA-Net is designed for the usage in Minifactory. This enables AAA-Net to cover a wider range in the OSI reference model covering the layers three to five and implementing the translation of agent software data in layer five down to the conversion into the appropriate format for transmission over Fast Ethernet in layer three. AAA-Net offers two different transport types for communication. *Non-guaranteed transmission protocol* is state and connection-less and aims to provide high network performance. In order to provide a reliable network communication *guaranteed transmission protocol* is used for state-full connections between agents. These two types of protocols are like UDP and TCP but are specifically designed in order to provide high performance and low latency. [13]

# 3 Improvements of Minifactory: Hardware and Infrastructure

In order to implement decentralized computation the modular structure of the AAA systems was expanded and automated initialization procedures the hardware of CMUs Minifactory needed to be improved. In this chapter the changes to the end effectors, manipulator agents and courier, that were made for this thesis, are described. In addition, the existing Interface Tool has been expanded, which allows the system to be programmed and by running the whole factory in a virtual environment, it also opens the possibility of simulations. In order to better integrate this chapter with the whole Minifactory, the layout of the different sections is identical to the layout in section 2.2.

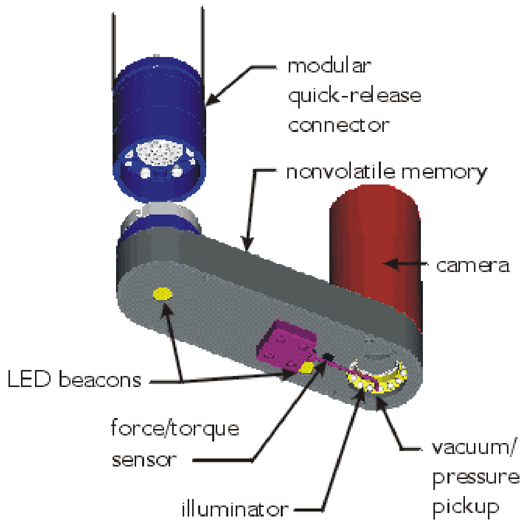
## 3.1 Minifactory

The main focus of this thesis was on extension and initialization of the Minifactory. This instantiation of the AAA framework illustrates the paradigm which are described in section 2.2. As stated in the introduction (see chapter 1) one main requirement of this thesis is decentralized computing. In order to achieve an agile system, each component needs to have its own processing unit. As such an endeffector in an AAA system should contain its own model and environment interaction module. Additionally to the need of a RTOS, an easy integration of optical sensor modules is required for endeffectors. This was achieved by using a standard Linux-based OS which offers an integration of computer vision frameworks.

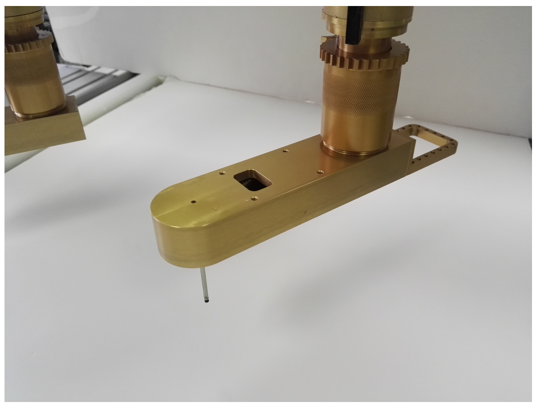
### 3.1.1 End-Effectors

The end-effector is the part of a manipulator which affects the product by manipulating it. There are different kinds of end-effectors like vacuum grippers, welders, gluers, lasers, a japper tip or a tweezer which can be mounted on top of the manipulator. End-effectors can have a force sensor build into it. The implemented interface, that connects the end-effector to the manipulator, offers several different power supply lines, vacuum and pressured air channels. Additionally different Input/Output (IO)-lines as well as communication busses like USB and $I^2C$ for specific hardware like cameras, processing tools and Ethernet for connection to processing power are provided in this standardized connector. Without the connector there would be no quick exchange of different endeffectors between manipulator agents.

Figure 3.1a shows a draft of a typical end-effector, Figure 3.1b the designed prototype [15].



(a) Draft of typical endeffector provided by [12].



(b) Developed prototype endeffector

Figure 3.1: Minifactory: Endeffectors

In order to be self contained the end-effector has a small integrated processing unit. The computer, that was chosen for this task, is a RaspberryPi Zero[1], which is a single-board computer running a Debian based operating system called Raspbian. The OS is stored on a SD-card which offers a fast way to test different configurations for the OS. A 40-pin General Purpose Input/Output (GPIO) offers a wide variety of connection possibilities like $I^2C$ and Serial Peripheral Interface (SPI). Since the RasberryPi Zero does not have a built-in Ethernet connection, a separate Ethernet adapter was connected to the SPI bus. The $I^2C$ bus connects directly to an Analog Digital Converter (ADC) and through the interface to the processing unit of the manipulator. This way the communication can be initially established as described in subsection 3.3.4 [28].

The inventory of an end-effector includes a camera, that enables finding parts which are needed for the assembly and navigating to them. The processing of the camera images is done by the end-effector, using the RaspberryPi. The OpenCV[2] framework, which was compiled for the chosen platform, empowers the end-effector to detect the assembly piece on a courier or storage. A manufactured part can have different shapes and forms with a very small size. Therefore the lens of the camera has to have a magnification factor to capture the millimeter sized parts in a high resolution.

[1]more details: `https://www.raspberrypi.org`
[2]more details: `http://opencv.org/`

One advantage of an integrated image processing in the end-effector is the ability to exchange the end-effectors, since different kinds of effectors need various image processing algorithm. Another advantage is the reduction of the processing load in the manipulator agent. If the image is already processed, then only the relevant information (i.e. movement commands) are relayed to the manipulator. This reduces the processing load of the manipulator and enables the agent to focus on the communication with other agents and the control of the motors.

Achieving a high level of inter-agent coordination is only possible if the sensor of an agent have the needed precision. Contact tasks require accurate sensors to prevent damage which could happen through the usage of to much force. Therefore the possibility to equip end-effectors with force sensors was included in the design. This enables the manipulator to sense the environment. There is a wide selection of commercially available force sensors. In order to fulfill the constraints of the Minifactory a force sensor needs to be able to work in 3 DoF, provide sensitivity of at least $0.1N$ and sense force along the $z$-axis as well as torque in the $x$ and $y$ axes. The used actuator in the current manipulator already offers this force sensing inside the motors by usage of the error signal in the controller and reading the torque of the motors [29].

### 3.1.2 Operating System

As described in subsubsection 2.2.1 the main OS in Minifactory is QNX. The in subsection 3.1.1 described end-effectors have other requirements to an OS than courier or manipulator agents. In order to meet those requirements, the second OS, which was chosen for an AAA factory, is Raspbian. The Debian based OS is used to control the end-effector agents of the Minifactory. Since an end-effector has a camera, the OS needs to be able to process the camera images. Using Raspbian made it possible to use the OpenCV framework which enables feature extraction from an image and positioning of the agent on top of the product. OpenCV is an open source computer vision framework that includes common algorithms for computer vision applications. These algorithms implement object identification, camera movement tracking and object tracking. They make is possible for end-effectors to find the product and interact with the courier while the courier is still moving.

## 3.2 Interface Tool

The existing Interface Tool (see subsection 2.2.2) was designed and programmed in 1995 in order to meet the requirements of version 1.0 of the Minifactory [11].

These requirements changed with the introduction of version 2.0 in 2005. In the course of this change the communication between Interface Tool and hardware agents needed to be redesigned. One major upgrade between the two version was the demand that the Interface Tool should be geographically independent of the hardware factory. This enables programmers and operators to control factories from all over the world without the need to be physically present at the site of the assembly system. The already implemented InterProcess communication Toolkit (IPT) allows message based communication only on local networks. Thus a new communication needed to be established which required the update of the Interface Tool.

The communication to the agents is described in the interface field of an (softare-) agent. This field itself consists of a database which encapsulates the actual implementation for both, the remote communication to the real hardware and the simulated agent running inside the Interface Tool. Inside this database there are state variables which offer monitor functionality of the agent and parameters. This state variables can be used as a way to influence the behavior during the operation mode of an agent. The implementation of an interface field has to be a subclass of the interface class as shown in Figure 3.2.

Each agent description must implement a method called *update*. This function is called by the Interface Tool as often as possible. It handles the movement of rendered parts and the agents description such as the position of the end-effector with respect to the manipulators actions and the position of the courier agents on the platen. Since there is a very strict hierarchy the program needs to be highly nested inside the Interface Tool, more details can be seen in Figure 3.2.

Since the Interface Tool is highly object oriented, the implemented changes fit the requirements of a modular character, which is essential in the Minifactory and AAA. Each component in the factory is now self describing. It offers methods and parameters which describe its appearance and dynamic possibilities. As an example each object offers a method which defines the single parts of the object and how this component is assembled. Thus it is possible to show the process of a correct assembly in the GUI even in the virtual factory by implementing these needed methods, which makes adaption of existing factories to this framework easy.

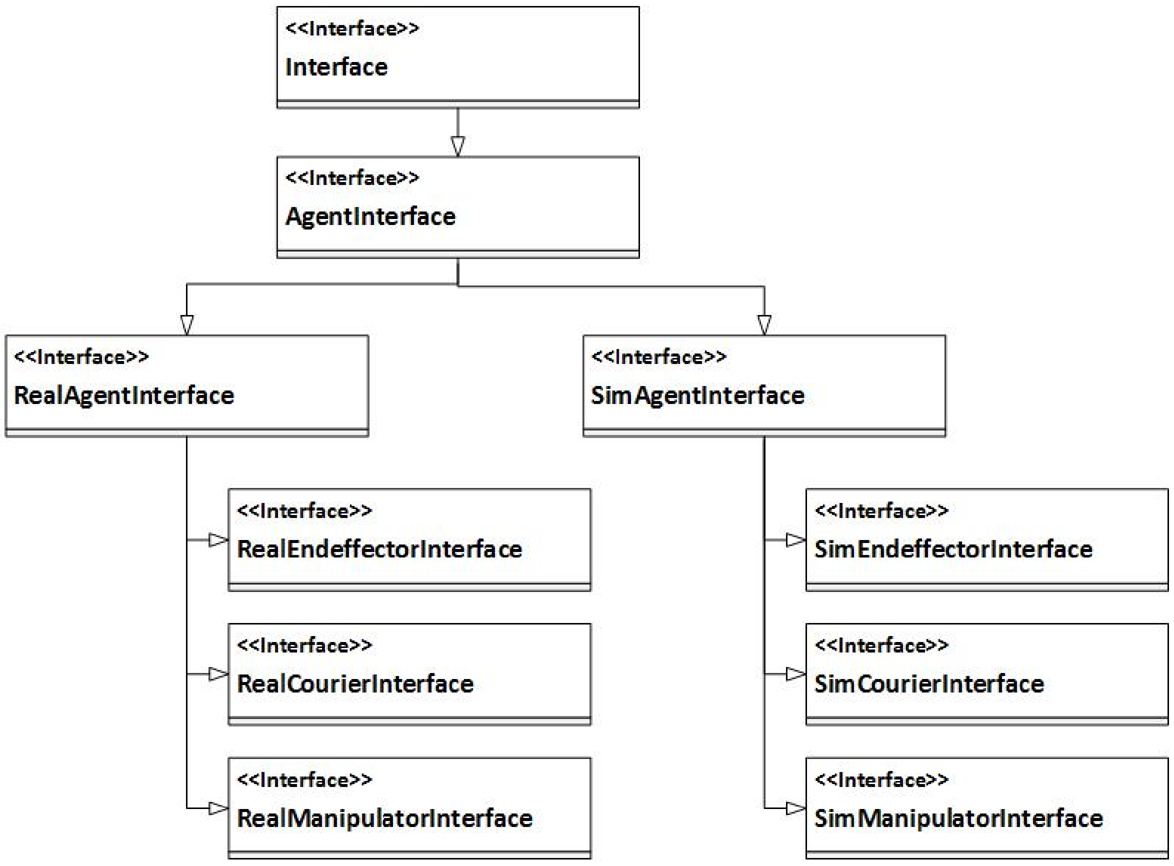


Figure 3.2: Class hierarchy for agents interfaces.

## 3.3 Communication

This section explains the different mechanism for communication in the Minifactory which have been expanded during this project. The main method of communication is done over Ethernet in a global IPv4 network. There are also different methods provided for establishing the first sequence of communication such as $I^2C$ and IR. The network has different methods as well, depending on the communication partner. If it is a user interface or the Interface Tool, an agent provides a websocket. The agent-to-agent communication is done via publish / subscribe methods.

### 3.3.1 Websocket: Interface Tool Interface

The Interface Tool connects to agents in the same way then the GUI. Therefore the same messages as described in subsubsection 2.2.3 were be applied for usage in the Interface Tool. One of the goals of AAA is the decentralization of infrastructure.

Using WSs as communication method between the Interface Tool and the agents made it possible to route the information over the Internet, since most firewalls do not block port 80. This enabled the Interface Tool to be anywhere in the world and still be able to manage and program the factory.

There are additional information, that are not needed by the GUI and therefore ignored by it. All messages which are dedicated to the Interface Tool were marked by a message type *Interface Tool*. This special type includes updates to the position of the agent like $x$, $y$, $z$ and $\theta$. Another type of message, which is dedicated to the Interface Tool, are progress information. Status updates during the operation of the factory are necessary to entail the progress of the production. The Interface Tool collects these information and displays them centrally for all agents. This enables a global reporting of the factory.

Since there are additional messages which are sent to the Interface Tool a higher bandwidth in the network was needed. In order to reduce the load on the network, agents were programmed to send Interface Tool specific messages only to the Interface Tool and not to each connected WS. This is achieved during the connection to the WS. Normal web clients connect to the agent in the root directory of the web server. The Interface Tool connects directly to the sub directory *wsint*, which is an extension of the WS directory *ws*. The difference between those two directories is the storage location of the connection instance. This enabled the agent to send message of the type *Interface Tool* only to connected Interface Tools.

### 3.3.2 Publish / Subscribe

One of the stated requirements of this project is initialization without prior knowledge. In order to achieve this, the existing publish / subscribe frameworks had to be updated and a new framework for broadcasting was implemented. This was needed, since the already existing DDS-framework does not allow multicasts without a specified recipient of the message.

#### Data Distribution Service: Extension

Each type of agent has defined messages which are customized to the special needs of the agent. These messages are partly defined by the abilities of an agent and by the information it needs to provide for the environment of the Minifactory.

A new endpoint for endeffector agents was developed which defines the communication aspects to these devices. This enabled the factory to reach the full agility together with the already existing agent endpoints for manipulator and courier agents.

In order to communicate between agents, messages have to be defined. During this project initialization, messages have been developed which allow the exchange of data like ID, address, model-information and additional information.

Asynchronous communication is the last expansion of the Data Distribution Service-framework which was developed in this thesis. Instead of polling for messages, a subscribed agent registers the desired message as an asynchronous message. This allows an event based communication between sender and receiver. During the registration the receiver defines a callback method which gets called as soon as a message arrives. Instead of waiting for messages the main operation of the agent has priority, which enabled an event based programming of the whole assembly process.

### Lightweight Communications and Marshalling

The third publish/subscribe library, which is used in Minifactory, is LCM. This set of libraries offers the ability to marshall data into defined structures as well as checking the received packages against these structures. LCM offers a platform- and language independent type specification language. In order to send messages over the network, there are some prerequisites which needs to be fulfilled. The first task is to define the type specification of the message.

A **message** is defined as a structure containing multiple variables of a simple datatype like integer, double or string. There is also the possibility to include another defined structure into the message. If the message is defined it can be compiled into byte-code which is specific for the programming language which is used. In order to send a message from sender to one or more receivers, all involved processes needs to agree to the interpretation of the bytes they exchange. If this is not the case the resulting system behavior is undefined.

Therefor each message needs **marshalling** in order to interpret the right bytes of the message. Another feature is the fingerprint which is attached to each message. This additional information is used in order to derive the type definition. Apart from the hash of the member variables and its types, there is also the whole message type itself recursively included.

If this fingerprint is not correct the receiving client reports an type error. In order to send data over a network an UDP multicast channel is used.

Each message needs to be sent over a specified **communication** channel. A receiving client needs to subscribe not only to the UDP multicast channel but also to the named communication channel in order to receive a message. It is typical that for each communication channel that only one defined message is sent over it. There can be multiple communication channel using one multicast address. The maximum size of a single message is 4GB of data. [30, 31]

In Minifactory LCM was implemented in order to be used as initial connection between different agents. As soon as some kind of identifier (see subsection 3.3.3 and subsection 3.3.4) is exchanged between two agents, LCM is used to exchange the rest of the needed information in order to establish a bi-directional communication channel. The courier or end-effector agent sends a request message over LCM to all subscribed agents requesting information about an agent with the delivered ID. Every receiving agent checks this ID against his own and only the affected agents replies with the full set of information needed to communicate with him like the IP address as well as ports for the WS, the DDS channel and additional information like type and position of the agent.

### 3.3.3 Infra-Red-link

In order to get the position and the initial communication between manipulator and courier, an optical interaction system is used in AAA. The manipulator is operating as sender. Mounted on each end-effector are two IR-Light Emitting Diodes (LED) which transmit a static pulse with a frequency of $5kHz$. On top of this base frequency an Identifier is modulated. This ID can be used in order to find the right manipulator with the help of broadcast messages across the Ethernet. Since the signal which is transmitted by the IR-LED needs to be generated there is not enough space inside the end-effector. Thus the generation and modulation is part of the manipulator and the final signal is transmitted through the connection cable down to the end-effector where the LED is mounted.

The courier has a optical coordination sensor mounted on top of it. This sensor is equipped with absorption filters in order to minimize noise, a Position Sensitive Detector (PSD) which is able to determine the relative position of the manipulator in reference to the courier with an accuracy in the sub-micro resolution ($150nm$).

Additionally the needed circuit boards for generating the data are located in the couier as well. After the light is filtered mostly from visible and ultra-violet light coming from the environment is projected onto the PSD. This sensor is able to detect the centroid position of the emitter. These sensor signals are used in order to determine the exact relative position of the manipulator with reference to the courier. After filtering the $5kHz$ base frequency the modulated information can be extracted.[15, 18]

The modulated information is determined by both the IP-Address and the sub-net mask of the manipulator. The ID is build from the client part of the IP-Address. This address is unique inside the sub-net and therefor a good identifier for the agent. After the courier has extracted the information it can contact the manipulator by combining the network-part of its own IP-address with the identification of the manipulator. These information together build the IP of the manipulator agent. After the first contact the courier is able to subscribe to the manipulator with help of the publish/-subscribe network as described in subsubsection 2.2.3. Additionally the courier can set the position data of the manipulator. In the initial stage during the factory mapping this data is only relative positions of the courier map. The courier has to update each position every time its map gets changed when combining the generated map with other maps from other couriers. After the whole factory is mapped each manipulator has the absolute positions with reference to the coordinate system of the biggest mapped courier. This map will be used as reference for the operation of the assembly system.

### 3.3.4 $I^2C$ Bus System

In the initial stage of finding the end-effector connected to a specific manipulator, there is no reliable way to get this information over Ethernet. Since there is usually more than one manipulator and end-effector in a factory, one cannot determine the right pairing by sending out a broadcast message over the network. The easiest way in order to establish a direct connection between the two systems is a wired connection. Since there are a series of data lines connecting an end-effector with the corresponding manipulator the selected initial communication method was established by using the $I^2C$ bus. As it is shown in Figure 3.3 this bus does not only connect the two agents together, it is also used in order to get analog data like the force sensor of the end-effector.

The serial protocol $I^2C$ is a two wire bus system which is used for low-speed devices such as micro controllers, A/D and D/A converters as well as Electrically Erasable Programmable Read-Only Memory (EEPROM).

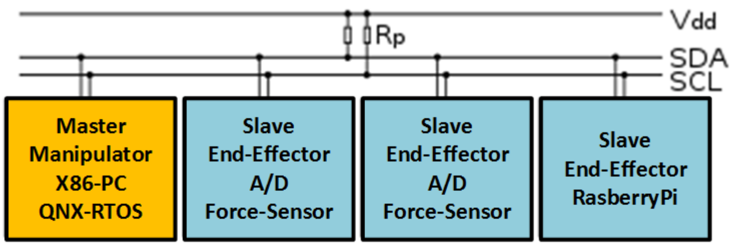


Figure 3.3: Schematic $I^2C$ bus devices, adapted from [32].

Although there are only two wires in the system there are a number of clients which can be used on the same bus depending on the addressing scheme either 127 (7-bit addressing) or in the newer version 1023 (10-bit addressing). Each slave needs to have an unique address in the bus in order to be addressable. There are two different operation modes of the bus.

In **single-master mode** there is only one master which reads and writes to the slaves. The master sets the clock for the whole bus, which also relates to the speed of the transmission.

In **multi-master mode** there is more than one master on the bus. In this operation mode there needs to be backoff mechanism in place, in case there are two masters transmitting at the same time. Also every master node needs to be in multi-mode operation in order to enable the collision detection mechanism [32].

Implementing a **slave** onto a Linux environment was a difficult task since there a strict time requirements in order to fulfill the timing sequences for a slave on the $I^2C$ bus. The processor needs to support $I^2C$ as slave in order to fulfill the timing requirements. Linux does not have an $I^2C$ slave driver built into the kernel by default. Since the manipulator processor does not have $I^2C$ support, an external hardware driver was required, that runs as $I^2C$ master. An additional package was programmed in order to offer an Application Programming Interface (API) for the Python programming environment. Thus the $I^2C$ was included in the high level programming of the manipulator [33].

In order to establish a communication between manipulator and end-effector, the raspberry pi inside the end-effector needed to support $I^2C$ slave mode. The arm processor built into the raspberry pi supports $I^2C$ slave mode since raspberry pi version 2. The Linux kernel was updated in order to enable a $I^2C$ slave driver.

After the kernel was modified the $I^2C$ slave driver could be accessed by writing to the path, specified by the kernel in the device tree. The kernel driver does not support callback methods. Thus a constant reading of the file was implemented in order to process data from an $I^2C$ master on the bus. The driver can be either accessed by an extra program written in order to communicate with the $I^2C$ master or by usage of a python wrapper through the high level programming of the end-effector.

# 4 Initialization of agents

As the AAA is a decentralized architecture there is no central storage holding the position of the agents. Therefore, an initialization routine has been developed which allows each courier to generate a relative map and a coordinate system representing the environment upon starting the factory. During this process there are different aspects which needs to be considered. This chapter describes the process of initialization. It describes the area mapping as well as detecting manipulator agents, communicating to these agents and lastly updating the Interface Tool in order to get an exact representation of the hardware factory.

## 4.1 Calculating relative position information between agents

In order to establish a communication between different agents the courier has to know where the other agents are. Mapping agents to a coordinate system is only useful if the courier has a defined origin. Thus the home position of the courier has to be found first.

### 4.1.1 Initialization of relative coordinate system

In order to find manipulators in the Minifactory the courier has to know its own relative position on top of one of the platen. Thus the courier needs to have a defined coordinate system, to run a self initialization. This must be done before any other agent can be detected by it. After starting the application the courier needs to find its relative home position in the coordinate system. Since the position of the courier is not defined on start-up the courier moves to the next boundary of the platen. The courier runs in a closed-loop control. There are four planar motors on the courier which use the grid in the platen. The same grid is also used by the sensor which measures the Alternating Current (AC) waves and determines the covered distance while moving the courier. Figure 4.1 shows the alignment of the four motors and the platen-sensor in the middle of the courier.

Since the motor driver is controlled in close-loop it is possible to measure the error of the controller. This measurement is used in order to detect the curbs of a platen. When moving the courier across the platen the error is relatively small since there is almost no force preventing the courier from moving.

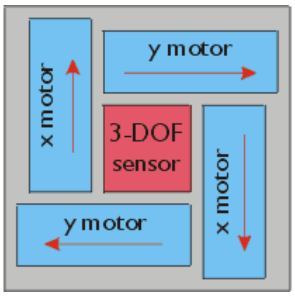


Figure 4.1: bottom side of a courier.

If the courier moves to a curb on the outside of the platen (see Figure 4.2), it runs against a hard barrier which needs more force in order to move on. Since the controller is still in moving mode, the error between the desired and the actual position will increase. This increasing error signal is used to detect the edges of a platen. A standard Minifactory platen which is used to hold manipulators has at least two curbs, trailing on the long side of the platen. Small platen are used to connect the different production phases and plates together. These smaller platen have between three and zero curbs. A courier will always be assigned to a big platen. Thus the initialization routine has at least two curbs in order to reset the relative coordinate system to the home position. If there is a connection to another platen, the platen-sensor is able to detect the small disturbance in the grid and can thereby detect the end of the platen as well.

During the initialization the courier will set the home position to the most negative number in $x$ and $y$ direction, labeled $H$ in Figure 4.2. This is either a curb or an end of the platen on the connection line to another platen. A Minifactory is designed to normally host two courier on a platen. Since each courier is connected by a tether to the brain box, each courier has a primary side on the platen which is the side of the brain box and a secondary side of the platen. If there are two couriers hosted on a platen the brain boxes are mounted on the opposite sides of a platen. The fact that a courier maps the area in $x$ and $y$ while maintaining a theta angle of $\theta = 0°$ the coordinate system of a courier can be transformed between two courier on the same platen by rotating it by $\theta = 180°$ and applying a translation matrix. Each platen can have its own relative coordinate system. The whole factory can be described as different platens connected by small platen forming a grid. The layout of the platen can be discovered by detecting the connections between a platen from each courier and detecting a manipulator on another platen.

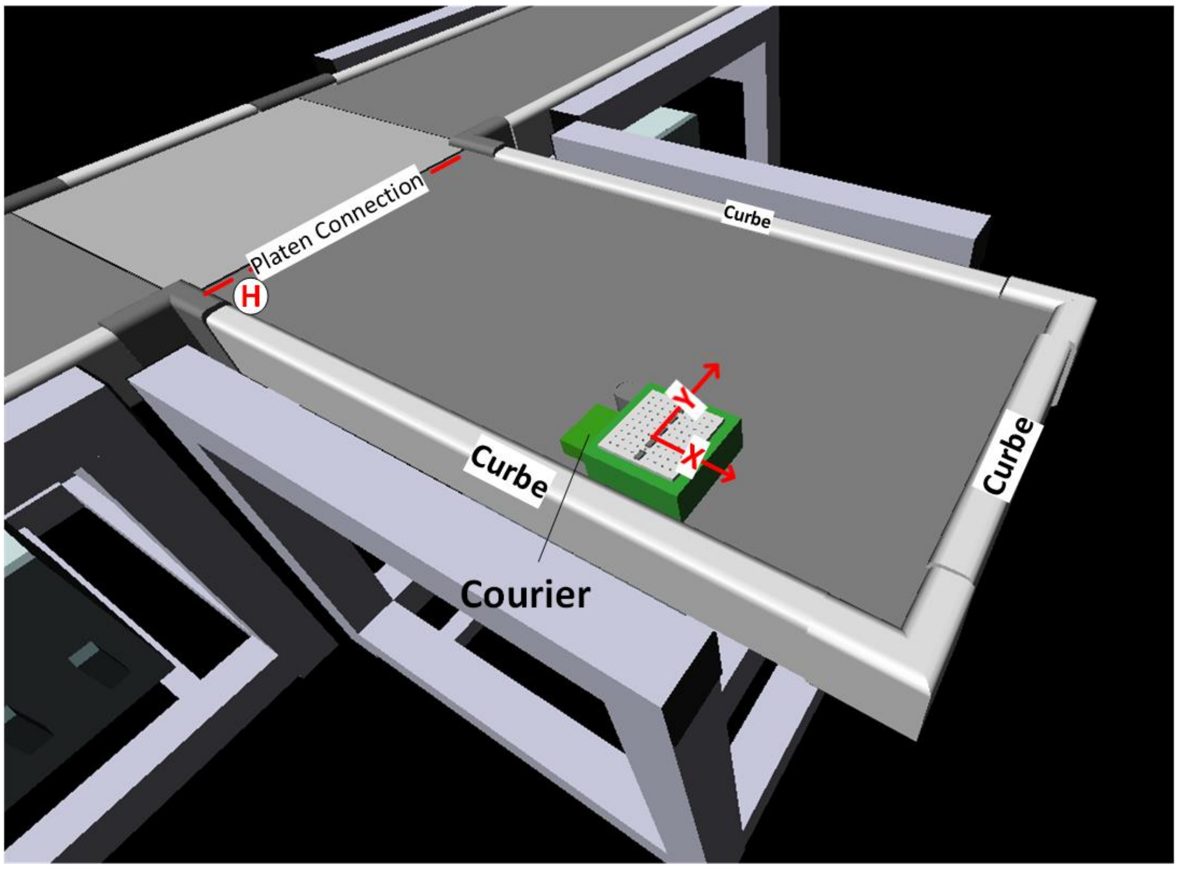


Figure 4.2: Orientation of courier on platen.

### 4.1.2 Area Mapping

After the home position of the courier is found the agent can start to map the area in the neighborhood. Since platen carrying a courier are standardized with a size of 1200 mm by 600 mm the courier can detect its orientation in reference to the platen by driving in either $x$ or $y$ axis until it hits another curb or a connection to another platen. Since the home position of the courier is in one corner of the platen, the covered distance to the next obstacle is the whole length the courier can drive on the platen. Therefor the agent is able to determine which one of its axis ($x$, $y$) are aligned to the long side of the platen. Additionally this initialization helps to find other platen and build a map since after this routine the courier has knowledge about connecting platen and curbs.

Since the courier has no prior knowledge of the factory, the second step of the mapping process is done by driving along the length of the platen while at the same time covering the whole width of the platen. By doing this the courier covers the whole area of the platen at least once.

This enables the courier to determine if there are manipulator agents mounted above the platen since the agent is able to detect there IR-signal as described in subsection 4.1.3.

A platen has up to two courier moving around in initialization mode. In order to prevent collisions, the courier runs a defined path while mapping the area. Since the two courier are mounted at opposite sides of the platen they have a homing position diagonally with reference to each other. As a result the first half of the platen can be mapped without any precaution. The second half of the platen could lead to a collision if both courier keep going on and map the area from their starting position. After the first half is mapped the courier proceeds to the end of the platen while moving along the curb nearest to the brain box. After the end of the platen is detected the courier will map the second half of the platen in reverse order. This procedure helps to prevent collisions. Additionally the gain of the closed loop control is very low during the initialization. This is needed in order to detect curbs without damaging the courier. Another benefit of a low gain is the low energy which helps to prevent damage if two courier collide together.

### 4.1.3 Detecting a Manipulator

In order to find a manipulator agent the courier has to have a way to detect the presence in the vicinity. This is accomplished by reading data of an optical sensor mounted on the agent. The sensor has, as described in subsection 4.2.1, an analog channel which determines the state of the integrated Phase-Locked Loop (PLL). A manipulator has an IR-LED flasher mounted at the bottom of every end-effector which send a signal with a base frequency of 5kHz. The courier uses the PLL in order to filter out other signals and lock into the base frequency of the manipulator. As soon as the optical sensor has found this signal, it is shown in one of the sensor channel and the state of the initialization program gets changed from area mapping to precision navigation as described in the following section.

## 4.2 Precision Navigation

One part of the factory initialization is the systems auto-calibration. In order to calibrate the manipulator, the exact $x$ and $y$ position needs to be known. The first step in this positioning is finding the manipulator as described in section 4.1. The next step is a precision navigation in order to get the exact relative coordinate position of the manipulator.

This navigation uses the optical IR system between manipulator and courier. This section describes the reading of the coordination sensor and the controller needed to find the exact position of the manipulator as shown in Figure 4.3.

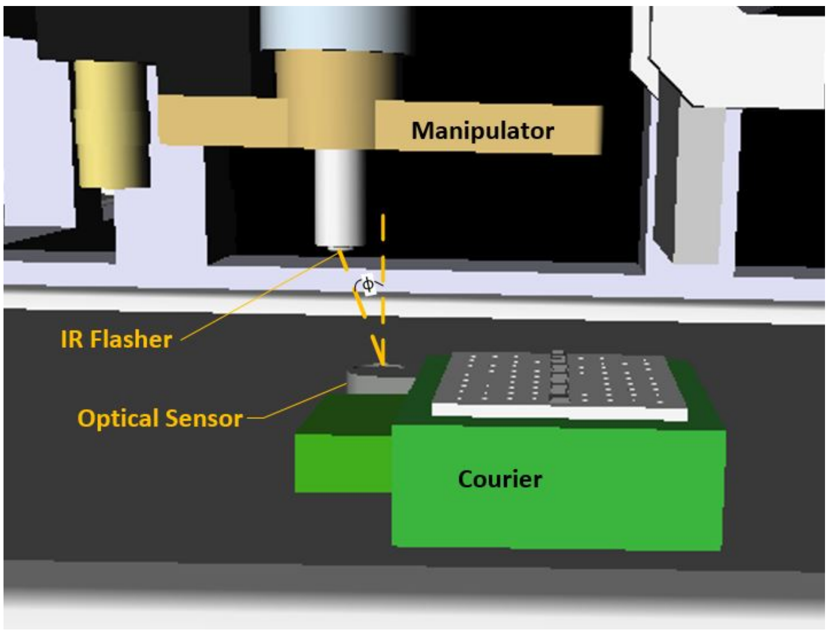


Figure 4.3: Optical Communication

### 4.2.1 Coordination Sensor

The coordination sensor offers 5 ADC channels ( $diff_x$, $sum_x$, $diff_y$, $sum_y$, VCO). The sensors $y$ axis is aligned with the motors $y$ axis. This means that an angle in the IR signal can be mapped directly to the needed motor control in order to correct it independently in both axis. The Voltage-Controlled Oscillator (VCO) signal is the PLL oscillator control output from the phase comparator. The input in the VCO channel reflects whether the PLL in the synchronous detector has locked onto the 5kHz IR-LED signal or not. This is represented by a stabilized signal in a narrow range of values.

The incident ray angle has the form $\arctan(\rho, f)$, where $\rho$ is the distance of the centroid of the LED image from the sensor center and f is the focal length. This can be used to compute the angles of the projection of ray onto the $xz$ and $yz$ planes. For a centroid location (dx, dy), equations Equation 4.1 and Equation 4.2 show the computation of the angle in $x$ and $y$ directions.

$$\phi_x = \arctan(d_x, f) \tag{4.1}$$
$$\phi_y = \arctan(d_y, f) \tag{4.2}$$

The actual centroid location (dx, dy) is reported by the PSD as a non-linear function by the sum and difference of the photocurrents at opposite ends of an axis. Thus a calibrated value can be approximated as shown in Equation 4.3 and Equation 4.4.

$$d_x \cong \frac{diff_x}{sum_x} \tag{4.3}$$
$$d_y \cong \frac{diff_y}{sum_y} \tag{4.4}$$

The sum is always positive. This leads to a rewritten Equation 4.1 as shown in Equation 4.5 and Equation 4.2 as shown in Equation 4.6.

$$\phi_x = \arctan\left(\frac{diff_x}{sum_x}, f\right) = \arctan\left(diff_x, f \cdot sum_x\right) \tag{4.5}$$
$$\phi_y = \arctan\left(\frac{diff_y}{sum_y}, f\right) = \arctan\left(diff_y, f \cdot sum_y\right) \tag{4.6}$$

Since the intrinsic calibration is difficult to recover, this is further simplified to an extrinsic calibration which lumps the focal length and the sensor scaling into a single dimensionless parameter for each channel as shown in Equation 4.7 and Equation 4.8

$$\phi_x = \arctan\left(diff_x, k_x \cdot sum_x\right) \tag{4.7}$$
$$\phi_y = \arctan\left(diff_y, k_y \cdot sum_y\right) \tag{4.8}$$

The shown computation for an angle can only be executed if there is a stabilized reference signal. Therefor the included control mechanism waits until the VCO reference which presents an existing IR-LED signal with a frequency of 5kHz has a stable signal output.

This signal also displays if the calculations are really showing an existing signal or just noise.

### 4.2.2 Controller

As soon as the PLL is locked to the 5kHz signal from the manipulator, the state of the courier program changes into navigation mode, to reach the center of the IR signal. This controller takes the given parameters $\phi_x$ and $\phi_y$, as stated in subsection 4.2.1, and calculates the error to the center of the IR sender. The coordination sensor returns the values as angles in a spherical coordinate system. In order to have an usable distance for the closed-loop control of the motor driver, the controller has to recalculate the point in the Cartesian system. Since the optical sensor already produces separate signals in $x$ and $y$ direction, the conversion from Spherical to Cartesian can be calculated as shown in Equation 4.9 and Equation 4.10.

$$x_{err} = \sin(\phi_x) \tag{4.9}$$

$$y_{err} = \sin(\phi_y) \tag{4.10}$$

The resulting distances $x_{err}$ and $y_{err}$ are the actual values of the two different PID controller while the desired value of both controllers is zero. The courier is controlled by three different controllers. The optical controller is the most outer control. It defines the chosen position of the courier in order to get the angle of the IR-signal to be 0°.

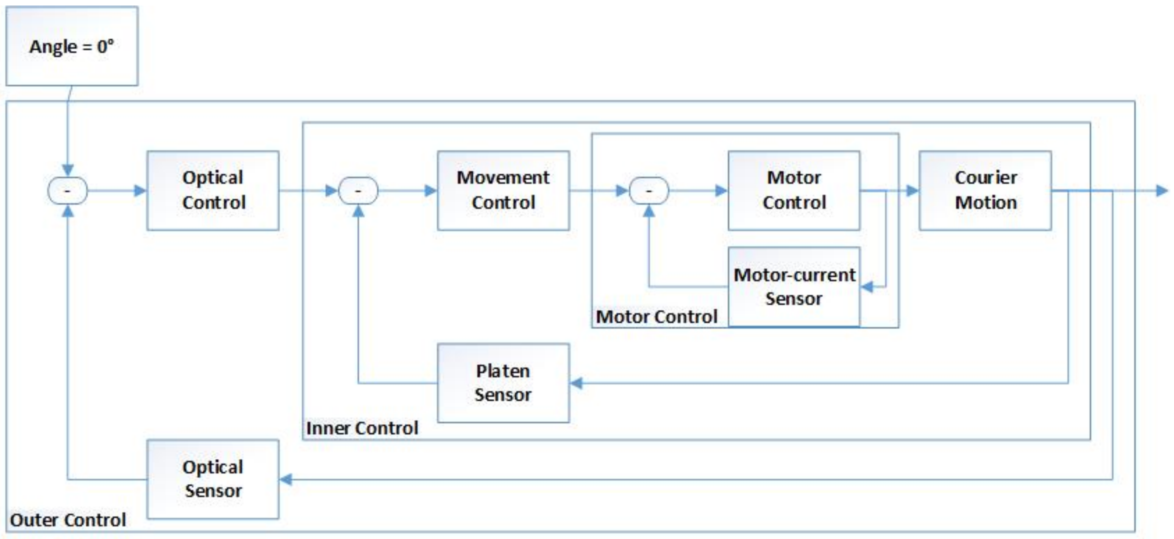


Figure 4.4: block diagram showing the optical control of the courier.

This input is the desired value of the motion control which is represented in Figure 4.4 as inner control. This controller handles the closed loop supervision of the courier motion by using the platen sensor as input. The four motors are directed by a motor control. This controller regulates the power each motor needs in order to fulfill the desired motion.

## 4.3 Communication between courier and manipulator

The initialization of a factory consist of detecting a manipulator and determining the correct position. Furthermore the additional parameter such as ID and network settings of every detected agent needs to be exchanged. Since there is more than one manipulator in a factory, it is not possible to use broadcast as a first communication method, because of the fact that there is no possibility to distinguish between the different manipulator agents on the network.

This section introduces the setup of the communication between courier and manipulator agent by using first an alternative communication path in form of optical methods and second switching to network communication. The full communication scheme, which was developed during this thesis work, is shown in Figure 4.5 and explained in detail in the following paragraphs.

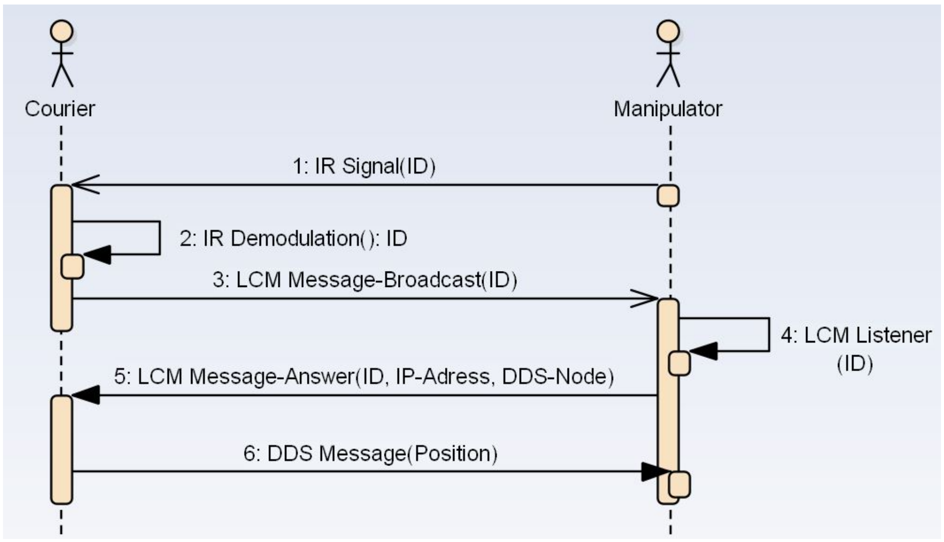


Figure 4.5: Communication flow between manipulator and courier agents[1]

[1]UML sequence diagram. More information http://www.uml-diagrams.org/sequence-diagrams.html

### 4.3.1 Modulation on IR-Signal

Setting up the communication between courier and manipulator requires a different connection method than Ethernet. Since there is more than one manipulator in a factory, it is not possible to use broadcast without any prior knowledge. In the Minifactory an IR based optical communication is already used for precision navigation and determining the position of a manipulator. The same signal can also carry information that enables the courier to identify the manipulator.

The Minifactory uses an IR signal with a base frequency of $5kHz$. The generation of the base frequency is shown in Figure 4.6. In order to add an ID to the signal it has to be modulated. There are different kinds of modulation. Since the receiver in the courier uses a PLL, a Frequency Modulation (FM) is not possible. Manipulator agents in the Minifactory use Amplitude Modulation (AM). This means that the amplitude of the high frequency carrier wave is changed in accordance with the intensity of the signal. The frequency of the carrier (base frequency) remains the same. Since each manipulator has a different ID the modulated signal is generated by the main process of the agent. The generated output is transferred over a serial interface to the IR flasher board and used as modulator on top of the carrier. The whole transmission process from the brain box of a manipulator to the brain box of a courier is shown in Figure 4.7. In the overall communication flow shown in Figure 4.5 the optical transmission of the ID is part of *Message 1*.

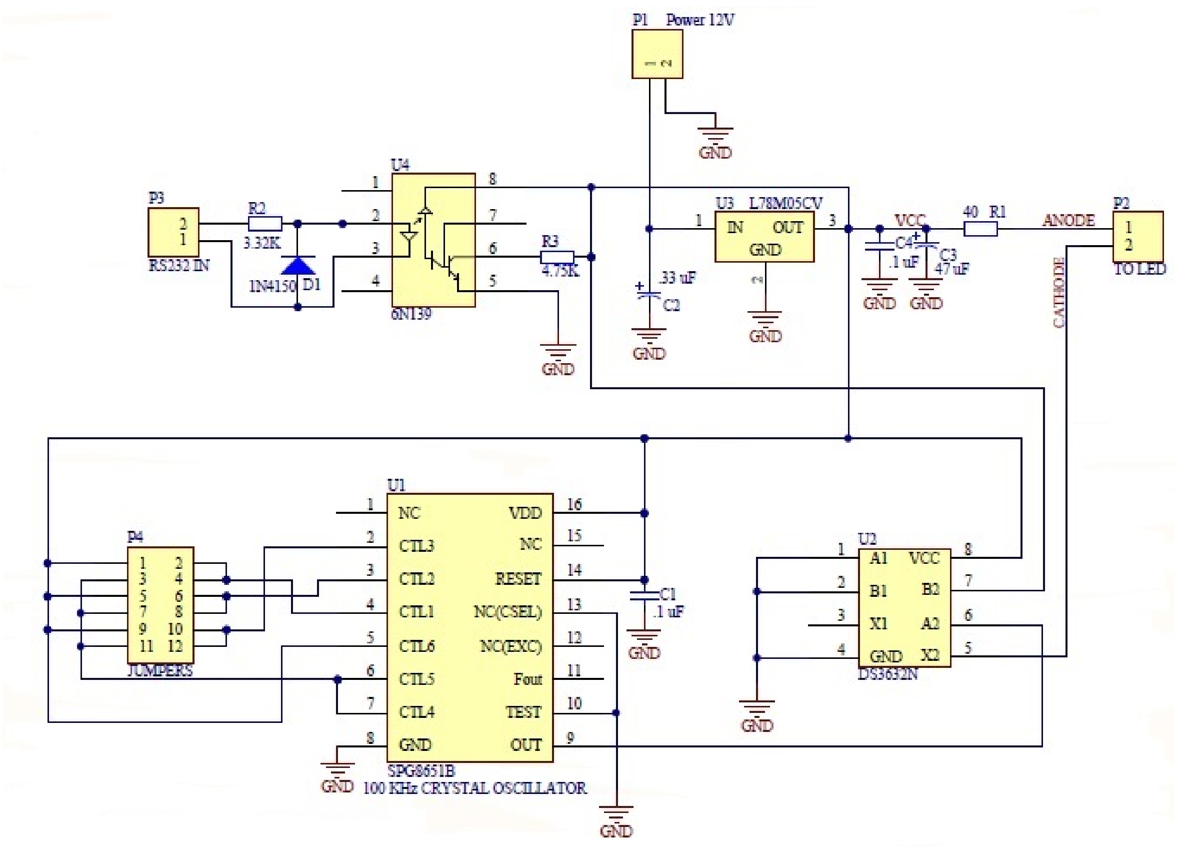


Figure 4.6: IR signal generation provided by [12].

The courier has an optical sensor that is able to sense the IR signal of the manipulator. After the manipulator is detected, the process of demodulation of the signal is started. In order to read a modulated wave , it is necessary to change the nature of the modulated wave. This is accomplished by a circuit called *detector*. A detector circuit performs the following two functions:

**Rectifying the modulated wave**: A modulated wave has a positive and a negative half which are exactly equal. Therefore, the average current is zero and the signal is not readable. Eliminating the negative half of the modulated wave enables the signal to be readable by the system.

**Separating the signal from the carrier**: The rectified modulated wave signal contains the ID signal and the carrier. The recovery of the ID signal is desired. This is achieved by a filter circuit that removes the carrier frequency and allows the signal to reach the input of the courier processor. This process is shown in Figure 4.5 as part of *Method 2: Demodulation IR Signal*.

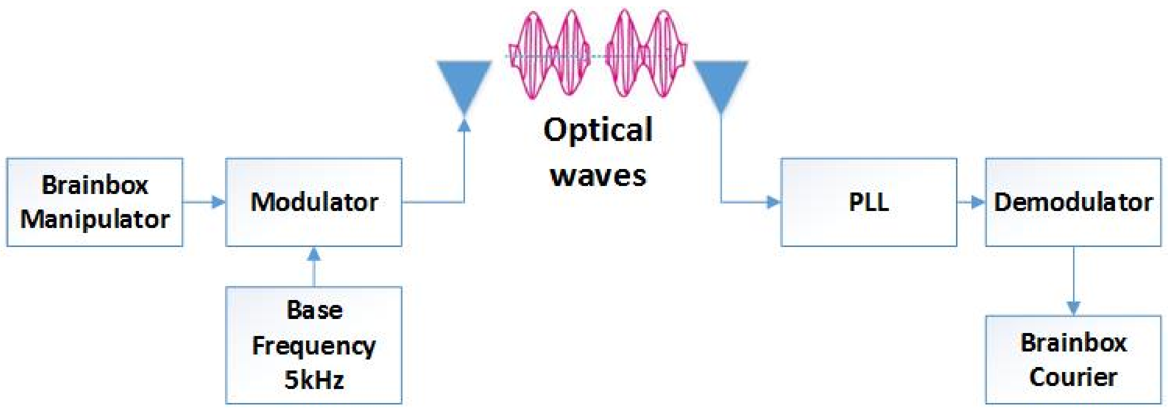


Figure 4.7: Optical Transmission.

### 4.3.2 Finding Network Agent

After the ID is extracted from the IR-signal the courier agent has to determine the network address of the detected manipulator agent. The communication framework used during the operation phase of the production cannot be used since DDS needs to have a specific receiver in order to send messages through the network. Thus the network communication needs to be established with a framework which is able to send to unknown subscribers. This detection is done with help of the LCM-framework. As described in subsubsection 3.3.2 the LCM-framework uses UDP multicast in order to communicate with all subscribers to a certain message.

In order to send defined messages the courier agent generates a LCM message which is described in Listing 4.2. This process is shown as *Message 3: LCM Message-Broadcast* in Figure 4.5. The message can be used for requesting the information from a manipulator and the answer from the manipulator to the courier. After inserting the available information like sender name, sender and receiver type and id the courier proceeds to send the message to the multicast address as stated in Listing 4.1. This address is the default multicast address as well as the default port for LCM messages.

The parameter $ttl = 1$ defines the scope of the message to be the local subnet. By default the messages have $ttl = 0$ which would set the message to be only sent to the local network card and not to the connected subnet. *Ttl* describes the number of nodes the message is allowed to pass in order to reach a receiver. Each network card and router count as one node.

```
udpm://239.255.76.67:7667?ttl=1
```

Listing 4.1: Multicast Address for LCM

Every agent listens to the *DETECT* channel. This channel is used in order to initiate communication between the agents and exchange the data needed for communication. As soon as a message is received (*Method 4: LCM Listener* in Figure 4.5) the agent checks if the message has the same receiver type and ID than the agent. The agent does not do any further processing if it is not the desired recipient of the message. In a factory there is only one agent which has the right ID and type specified in the message. This is one requirement which needs to be fulfilled during the setup of the factory. The agent inserts the missing information like websocket, HTTP links for the interface and the CAD-model as well as the IP-address. Additionally it checks whether the position information of the agent needs to be updated in the local storage and applies the changes if needed. After the message has been modified in order to entail all available information, the manipulator changes the answer flag to indicate that the sender and receiver should now be exchanged. Afterward the agent sends the message through the *DETECT* channel back to the courier, as is labeled as *Message 5: LCM Message-Answer* in Figure 4.5.

```
package lcmMinifactory;
struct agent_t
{
    int64_t  timestamp;
    string   snd_name;
    string   snd_type;
    string   rcv_type;
    int      rcv_id;
    string   rcv_ip_address;
    string   rcv_websocket;
    string   rcv_http_interface;
    string   rcv_3d_model;
    double   rcv_x_pos;
    double   rcv_y_pos;
    boolean  answer;
}
```

Listing 4.2: Message definition of the LCM DETECT channel

The sending courier receives the answer and processes it. After it is established that the courier is the right one (message type and name) the final part of the communication can be established. The provided information allows the courier to switch from LCM to DDS communication. There are additional information like websocket and HTTP links received that are stored in the manipulator database.

Additionally the area mapping (see subsection 4.1.2) is updated with additional information, in order to obtain a complete map of the platen.

### 4.3.3 Updating Agent

During the operation phase of the factory the agents communicate with help of the DDS framework. This framework allows real-time p2p communication between different processes in the Minifactory. In order to establish a connection the name of the remote node is needed. This variable is provided with help of the LCM framework. There are different functions implemented in order to exchange data between agents. Some methods are the same for every agent, others are different depending on the requirements for a specific agent. An example of different implementations would be the publishing of the position of an agent. Courier agents publish their position using $x$ and $y$ coordinates whereas manipulator agents publish their $z$ and $\theta$ position.

A special method is the position update function that allows the courier to update the general position of a manipulator in the coordinate system by sending the, during the precision navigation phase (see section 4.2) detected, $x$ and $y$ values to the manipulator. This update method is labeled *Message 6: DDS Message* in Figure 4.5. The manipulator stores those coordinates in its global database along with the name of the sender as additional information. If there is already an existing entry in the position database the manipulator send the old position/s to the new courier as answer to the update. If there is a difference between the sent coordinates and the stored ones the courier will get the update along with the name of the other courier which has detected this particular manipulator. If there is more than one manipulator detected by the same courier they can update their map and find the rotational and translational matrix between their two relative coordinate systems.

By having two points in a 2 dimensional coordinate system it is possible to determine the vector between these points. Since it is known that the manipulator does not change the position after startup with regards to $x$ and $y$ a courier can calculate the difference between its own vector, between two manipulators and the vector of another courier. After calculating the matrix, the relative coordinate system of one courier can be rewritten as the relative coordinate system of the other courier. The coordinate system which is rewritten is the coordinate system of the courier with the higher ID. Each courier has a list referencing the dependent coordinate systems from other couriers. If a courier meets another courier with a lower ID it needs to update its own coordinate system.

If the courier has dependent couriers it will send a message to the other couriers with the new matrix in order to update their own coordinate systems. This enables the factory to generate a global coordinate system after all couriers have mapped their respecting area and met another courier form a different platen.

Every time a courier updates its relative coordinate system the position of the detected manipulator agents is updated as well. In order to update the database of the manipulator the courier sends a new update message to the manipulator with the new position data. An additional parameter is also provided in order to tell the manipulator the origin of the relative coordinate system in which the position is expressed. If the manipulator receives a position in a certain coordinate system it checks whether it already got a position from this system. If the position is the same in both entries an acknowledgment is sent to the courier. Otherwise an error is generated since there are contradicting position information from different couriers. Additionally the manipulator knows when the last update is received as soon as the reference coordinate system is from the courier with $ID = 1$.

## 4.4 Updating the Interface Tool

The last part of the initialization of the connection between courier and manipulator agent is updating the Interface Tool. The Interface Tool is used as a design and programming platform as well as a visualization tool for the AAA factory. The Interface Tool needs to have a precise model of the assembly system while representing the state of the physical factory. In order to get a precise model, the factory needs to update the model after the connection between courier and manipulator agent is established and a precise position for each manipulator is defined. Since an AAA factory is a decentralized system there is no need for an Interface Tool to be connected directly to the factory. WSs are used as a communication method between the Interface Tool and every single agent in the factory as shown in Figure 4.8. This enables the programming of the assembly system from all over the world.

### 4.4.1 Receiving Updates

The process of building a new factory starts with designing a virtual factory and testing it against the given requirements of the factory. This simulation can be used in order to eliminate errors in the concept before building the hardware and thus lowering the costs for reconstructions or error correction in the hardware factory. Every factory has a plan describing the process of assembling the factory as designed.

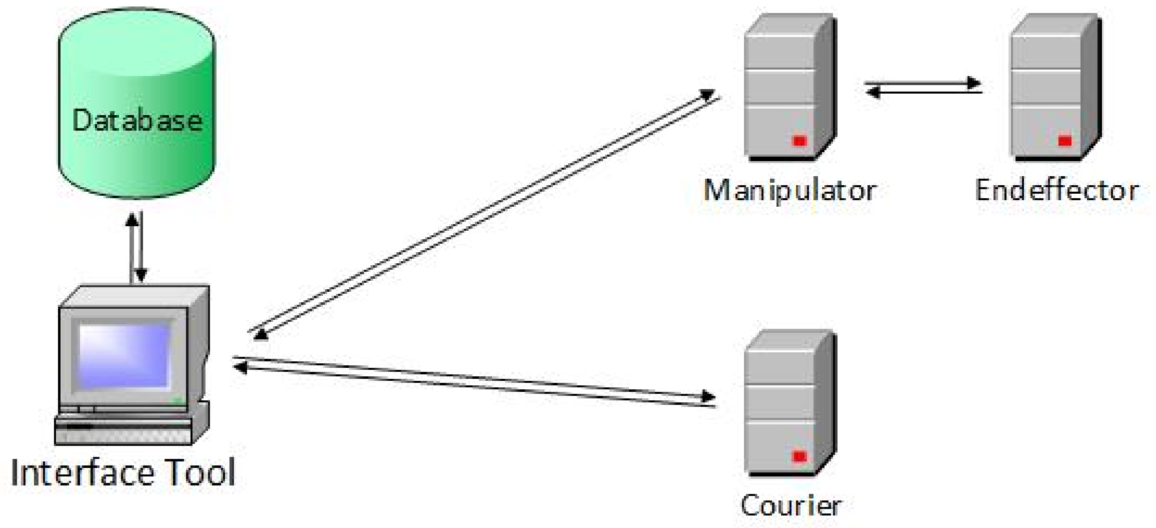


Figure 4.8: Communication flow between agents and Interface Tool

Since there already exists a virtual model of the factory, the assembly of the hardware factory does already have a plan on where to mount the manipulators and where to place the courier on a platen. This prior knowledge is essential while changing the simulation as the base for the later visualization of the hardware factory. The simulation has already the placement of the agents. Connecting the simulation to the real world requires the definition of the WSs to which an agent in the Interface Tool needs to connect. After the connection is defined the Interface Tool is able to communicate with the appropriate hardware agent. As described in subsection 2.2.2, there are two operation modes in the Interface Tool. If the Interface Tool is started in hardware mode it can be chosen by the usage of parameters on start, whether the Interface Tool starts offline or tries to connect to the specified WSs of the agents directly. The Interface Tool also provides menu options to connect to the agents after the program has already been started.

As soon as the Interface Tool is switched to the online mode, the specified WSs try to connect to other agents and once connected, try to receive status updates of them. If no WS is available, the Interface Tool starts a backoff time until it tries to connect again. A WS is an extension of the HTTP protocol. Therefor it enables the Interface Tool to act like a web client while communicating with the different agents of an assembly system. For more details see subsubsection 2.2.3. This fact makes it easy to connect to the factory, as most firewalls allow HTTP traffic while blocking other types of traffic. For every agent in the factory the Interface Tool opens a WS, connecting it to the agent.

Depending on the type of agent there are different interfaces that uses the WS to send and receive data. The specific interface needed for an agent is selected after the WS is connected and the first global information data is downloaded from the agent.

The first information that are received by the Interface Tool after connecting to an agent are name, type, subtype and status of an agent. This information is used by the Interface Tool to determine if the given WS is the correct socket for the specified agent. The selection of the correct interface is needed to communicate with the agent and implement the desired reaction from received messages. The agent is checked against its name specified in the model of the factory.

The Interface Tool generates a message in order to inform the user about the wrong configuration if both agents (model and hardware) do not have the same name. By selecting the desired driver from the internal database, the Interface Tool is able to communicate with the agent and get additional parameters such as the exact position of the agent as well as the relative positions of connected components. In case of a manipulator the Interface Tool gets the current position in $z$ and $\theta$ which are specified as absolute position in the coordinate system of the manipulator. In order to display the position the Interface Tool needs to calculate the position in its own coordinate system, as described in subsection 4.4.2. The state of the motor in an agent can result in a notification of an user or a change in the representation of the model.

### 4.4.2 Position Calculation

The received coordinates from a manipulator are based on a relative coordinate system inside a courier. In order to display the manipulator at the right position the Interface Tool has to transform these coordinate to the desired coordinate system. The model of a factory is described as a child parent relationship between elements of a factory. Each child element has a relative coordinate system describing the position with reference to the parent element of the factory. Transforming the coordinates to the right relative coordinate system enables the position to be converted through every parent and child system that builds the path from courier to manipulator.

The courier agent is a child element of the platen on which it is instantiated. The maximal space a courier can drive on this platen is generated by the courier during the initiation of the relative coordinate system as described in subsection 4.1.1. This maximal position is used by the Interface Tool in order to calculate the relative position of the manipulator to the platen since the courier can not drive to the edge of the platen because of the mounted curbs. Figure 4.9 displays the structure of a factory model with four manipulators and two courier on a single base unit. After the transformation into the relative coordinate system of the platen the position is also considered to be the absolute position since the coordinate system of the platen is the same as the representation of the base unit.

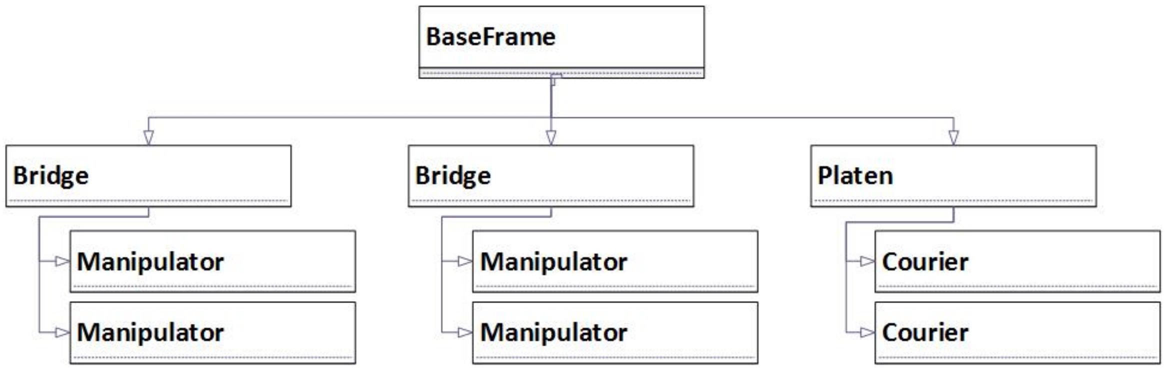


Figure 4.9: Structure of a factory model.

A base unit has bridges that enables the mounting of manipulator agents. These bridges can be moved along the platen to position the manipulator in absolute $x$ position of the base frame. The $x$ position is the axis of the long side of a platen. Since the manipulator has a relative $x$ and $y$ position with reference of the base frame the bridge holding the manipulator is updated with the given $x$ position of the manipulator. Only the bridge is able to move the manipulator in $x$ position. In order to determine the correct position for the bridge further position transformation needs to be completed first. The given point regulates the position of the IR-LED of the manipulator. This position is not the same as the $x$ position of the bridge. Depending on the type of manipulator and the model behind the Interface Tool is able to calculate the difference between IR-LED position and the mounting point of the bridge.

After updating the bridge position and resolving the absolute position in $x$ axis, the $y$ position needs to be calculated. This position is defined by the mounting point of the manipulator on the bridge. The position of the mounting point needs to be calculated the same way from the position of the IR-LED of the manipulator as the $x$ position. Since the Interface Tool models the factory in relative coordinate system the $y$ position needs to be determined with reference to the coordinate system of the bridge.

Since the position of the $z$ and $\theta$ axis are only relative to the manipulator the only calculation needed for the display of these parameters is the transformation from the given parameter to actual position coordinates by usage of scaling matrices. These positions gets updated as soon as the manipulator operates one of the axis during operation mode. The WS sends the new position and the calculation transforms the position to a point in the Interface Tool.

### 4.4.3 Model Update

The model of a virtual factory is stored as a file with the ending *.fac*. Inside this file every component of the assembly system is listed as described in subsection 2.2.2. The file supports instruction commands in form of python programs. During the design of the factory the Interface Tool automatically structures the factory file with preassembled information about the assembly system. The child parent relationships are represented in nesting the children inside the parent objects. Every object entails a position matrix which describes the relative position of the object with reference to the parent object. Additionally the representation as CAD model is linked to the factory file. In order to connect to a hardware agent each virtual agent representing a real agent has a interface defined that entails the corresponding WS for the communication. After the initial assembly of the factory is completed the user can insert the program for each agent into the *run* method of the agent. This program will be transferred to the agent and run inside the agent environment. Available commands are described in the agent library which is hosted inside an agent.

After successfully connecting to an agent there are certain information that are downloaded into the Interface Tool. The information about the location of the CAD model and how to download it is included in the initial message after connecting to the agent. The Interface Tool gets additional information about the CAD model like name and version which can be compared to the stored CAD library inside the Interface Tool enabling the limit of bandwidth usage. If the connected agent is not available in the library, the Interface Tool will download the model from the agent and use it. The Interface Tool is able to process the version of a stored model which enables it to update the model to the newest version as soon as there is a reconnect to the agent. That means the representation is always the most resent and accurate model of the hardware factory.

In order to have a persistent model the changes during initialization like CAD model update or position update need to be written into the factory file. The file is only parsed once during the loading of the factory. It is not needed during the operation phase of the factory. This enables the Interface Tool to write changes of an agent directly to the file without the need of locking mechanism like Mutex, not blocking the operation of the assembly system. The layout of the factory file is very important for the consistent parsing into the Interface Tool. Changes to the model need to be inserted at the right position. During the writing of a file the whole content of the unmodified file is cached in the Interface Tool and updated first. After that the edited file is written as a whole again, instead of updating only the changed sections of a file, reducing the chances of errors in the file. The position of an agent is defined as a matrix existing of a rotational and a translational part. It is written as a single line vector consisting of 16 numbers. This vector represents a $4x4$ matrix in the factory file.

# 5 Validation

In chapter 4 an initialization procedure has been developed. The procedure consists of several individual steps, namely

- Calculating relative position information between agents
- Precision Navigation
- Communication between courier and manipulator
- Updating the Interface Tool

In order to validate each of these components different validation scenarios were designed an executed, to demonstrate the correct functionality of the initialization.

## 5.1 Validation of the relative position calculation

First contact is defined by three different parts:

- initialization of the relative coordinate system
- area mapping
- manipulator detection

As such, three different validation methods were designed to test each parts for correct operation. Due to prior tests the ground truth of the courier movements is given with a precision of $200nm$.

The initialization of the relative coordinate system of a courier was tested with a laser distance measurement system. The test had two different stages. If the courier had detected a curb it would stop until a manual input from the user. After the detection the courier moved back to its origin. The distance between the curb and the origin was captured with help of the laser interferometer. The settings of the validation are shown in Table 5.2. As **??** shows, the validation used absolute positioning of the courier. If the curb is not detected correctly, the distance between origin and curb will not be the same as the real movement. This test was done for all four endpoints of a platen with no distinction between curb and platen joint. Thus the detection can be validated, which is the main process of the initialization of the relative coordinate system of a courier.

| | |
|---:|:---|
| Number of tests | 10 |
| speed of courier | 50 $\frac{mm}{s}$ |
| accel. of courier | 100 $\frac{mm^2}{s}$ |
| max. force of courier | 30 $N$ |

Table 5.1: Rel. coord. system: Settings of validation

Since the area mapping uses the detected curbs and an additional buffer of $5mm$, the laser distance meter was used in order to determine the distance between curb and courier after every movement along the aligned curb. If the used grid is aligned correctly with respect to the surrounding curbs, the distance between curb and courier will stay in the stated tolerance of $200nm$. Figure 5.1 shows the resulting plot of the courier positions during the mapping. Labeled as blue is the initialization of the coordinate system, the area mapping path is drawn in red.

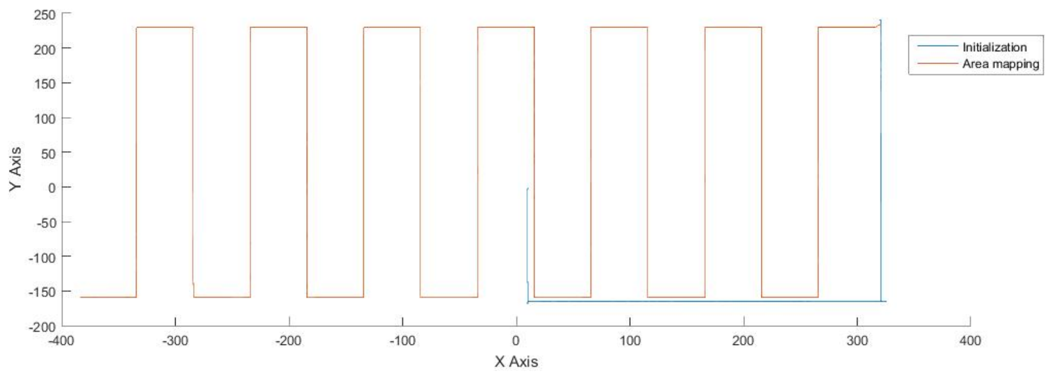


Figure 5.1: Area mapping: Path of courier

The validation of manipulator detection is two folded. Since the optical sensor is mounted at the corner of a courier the range of the detection depends on the angle of the courier. Thus a validation method for the manipulator detection is to position the manipulators at different positions throughout the platen and determine the borders in which a manipulator can be safely detected independent of the angle of a courier.

The second validation is the speed of the area mapping which a courier can have at the maximum in order to detect a manipulator safely. The settings of the validation are shown in Table 5.2.

The faster a courier moves the more uncertain is the detection. Since the PLL takes a few processing cycles to lock into the IR-signal which is transmitted by the manipulator. During the validation, different speeds of the courier were tested, more details can be found in Table 5.3.

| | |
|---|---|
| Number of Manipulator | 4 |
| Number of tests | 10 |
| speed steps | 20 $\frac{mm}{s}$ |
| Lowest speed | 60 $\frac{mm}{s}$ |
| Fastest speed | 400 $\frac{mm}{s}$ |
| max. force of courier | 30 $N$ |

Table 5.2: Manipulator detection: Settings of validation

| Batch Nr. | speed ($\frac{mm}{s}$) | detection rate |
|---|---|---|
| 1 | 60 | 100% |
| 2 | 80 | 100% |
| 3 | 100 | 100% |
| 4 | 120 | 100% |
| 5 | 140 | 100% |
| 6 | 160 | 100% |
| 7 | 180 | 100% |
| 8 | 200 | 100% |
| 9 | 220 | 95% |
| 10 | 240 | 90% |
| 11 | 260 | 80% |
| 12 | 280 | 60% |
| 13 | 300 | 20% |
| 14 | 320 | 5% |
| 15 | 340 | 0% |

Table 5.3: Manipulator detection: Speed and detection rate

As is shown in Table 5.3 the detection rate is very good up to a speed of $200\frac{mm}{s}$. If the courier moves faster, it can no longer be guaranteed that all manipulators will be detected. A slower speed than $200\frac{mm}{s}$ has no negative influence concerning the productivity of the system, as the detection is only done during the initialization phase of the factory.

## 5.2 Validation of the Precision Navigation

The validation of the precision navigation is done with help of a laser interferometer. The manipulator is approached from different directions and the controller is used in order to navigate to the center of the manipulator. The interferometer shows the exact position of the courier. If the optical sensor and the controller work as designed, the courier will always reach the same position, within the stated precision of the courier of $200nm$, independent from the starting point of this test.

## 5.3 Validation of the Communication between agents

The establishment of communication between manipulator and courier agents is done in three different steps. This process was validated by testing each single step itself, in order to prove the whole system.

**Optical Coordination Sensor:** Since the optical coordination sensor was already tested and validated in [18], the validation in this section covers the process of modulation and demodulation as well as sending and receiving the ID from the manipulator to the courier. In order to test the different parts of the system, as shown in Figure 4.7, the system needs to be split in single parts first. The transmitter and receiver were already tested and validated in [18]. Therefore the modulation and demodulation needed to be validated. This validation is done by sending a defined sequence of characters into the modulation and, after the transmission, read the demodulated sequence back. If both sequences are identical, the modulation and demodulation works as defined. The validation was done by using random characters from the visible space of the American Standard Code for Information Interchange (ASCII) set.

**Lightweight Communications and Marshalling:** The main reason of using LCM as communication framework during the initialization is the multicast feature which allows sending messages to multiple subscribers. As such the focus of the validation is on receiving sent messages and to assign them to the correct agent, which have been subscribed to the message. The validation was done by sending the defined message (see Listing 4.2) with random ID parameters. The recorded messages were analyzed according to the transmitted ID, if the corresponding agent responded to the message and if there was any response from agents which received the message but did not have the right ID. The validation was repeated by using different parameters as stated in Table 5.4. The parameters of the network load were achieved by running two Iperf[1] instances which generated the additional network traffic through the hub of the base unit and thus setting a defined network to the overall system. The results show that LCM did send and receive the defined packages even during the high additional network load.

[1]more details: `https://github.com/esnet/iperf`

| | |
|---|---|
| Number of tests | 20 |
| Min. number of agents | 2 |
| Max. number of agents | 12 |
| Min. additional network load | 0 Mbits |
| Max. additional network laod | 50 Mbits |

Table 5.4: Communication LCM: Settings of validation

**Data Distribution Service:** In order to validate the extension to the already existing DDS framework, these specific (new) communication methods were tested in terms of reliability and correctness. Like the tests for the LCMframework, this test also includes Iperf[1], it was used as a network load generator. This was done to test the performance and reliability during different loads on the network. The validation was repeated by using different parameters as stated in Table 5.5. The results show that DDS did send and receive the defined packages even during the high additional network load. Even high network traffic did not change the reliability of 100%.

| | |
|---|---|
| Number of tests | 20 |
| Number of communicating agents | 2 |
| Min. additional network load | 0 Mbits |
| Max. additional network laod | 50 Mbits |

Table 5.5: Communication DDS: Settings of validation

## 5.4 Validation of the Updating of the Interface Tool

The Interface Tool is connected to the agent over websocket. This socket sends periodical update messages to all connected clients. The update methods of the Interface Tool were validated by logging the received messages and comparing them to the sent messages of a single client. Using this validation technique, the whole system with all agents got validated. The results showed that the Interface Tool can process the update messages and display them in the specified way.

The position calculation was validated by sending defined positions to the Interface Tool and evaluating the outcome of the calculation with the predefined values. The system could be verified by using different positions and determining the correct outcome of the calculation.

As with the IR-signal modulation the model update in the Interface Tool was done by storing different versions of the CAD model in the agent and monitor the update process of the Interface Tool.

As is stated in subsection 4.4.3 the Interface Tool checks the current stored version of the model in the internal database against the latest model stored in the agent. If the version number is greater, the model gets updated. This was validated by changing the version number both increasing and decreasing it. If the number was increased, the model got updated during the next initialization of the connection between Interface Tool and the agent. A decremental of the version number did not alter the stored model. The results of the validation are shown in Table 5.6.

| Version Interface Tool | | Version Agent | Model updated |
|---|---|---|---|
| 1.0 | -> | 1.1 | Yes |
| 1.1 | -> | 2.0 | Yes |
| 2.0 | -> | 1.9 | No |
| 2.0 | -> | 2.0.1 | Yes |
| 2.0.1 | -> | 2.1 | Yes |
| 2.1 | -> | 2.0.9 | No |

Table 5.6: Model Update: Results of different versions

# 6 Scientific Contribution

According to the authors of [34], manufacturers are facing increasing pressure to decrease development costs and deployment times for automated assembly systems for a variety of precision mechatronic products. The difficulties of integrating these systems must be significantly reduced in order to meet new and changing market needs. There are many different approaches to solve that problem, among others are these major efforts:

- Design of agile manufacturing workcells [35].
- Rapidly reconfigurable machining systems [36].
- Over-arching frameworks for manufacturing enterprises [37].
- Systems viewed as hierarchical collections of manufacturing "holons" [38].

One goal of this thesis is a contribution to decrease the deployment times for automated assembly systems. By automating the process of calibration and finding the different agents, it enables the user to decrease the setup time, since the manual assembly of the factory needs to have only a rigor of a few centimeters. Prior to that a high precision assembly system was required to have the same exactness in assembling the factory system. This preciseness represents a big part of the assembling process.

Programming an assembly system with planar robots takes a lot of time since the position of each agent needs to be programmed before the operation. Additionally the programmer needs to know the exact position of each agent, considering that the movement between the agents needs to be manually programmed. According to the authors of [34], the most common approaches are "robot-centric". Flexible multi-robot assembly lines tend to be very complicated and take a long time to deploy, because they concentrate on the assembly robots and not on the interaction between robots. This work improves the way of programming an assembly system by discovering the agents automatically. Each agent stores its own discovered position and is able to transmit these information to courier agents. Since this communication is automated between the agents, the programmer only needs to specify which agent should be involved in an assembly operation. The actual trajectory and position definition is included in the agent framework.

This thesis deals with the task of initialization in an AAA factory. As such the following parts of the work are the scientific contribution:

$C_1$ **Initialization of relative coordinate system**:
The courier initializes its own coordinate system on startup. This enables it to have a standardized origin every time the factory is rebooted, which is needed in order to have an automated calibration. Before this thesis the courier could only be controlled manually without any knowledge about its position on the platen.

$C_2$ **Area Mapping**:
A pre-requisite of calibration is the knowledge about the boundaries of the environment. A courier agent can know detect the edges of a platen every time the courier is initialized. This enables collision detection between different couriers since the position of curbs is know.

$C_3$ **Detecting a Manipulator**:
Finding different agents is needed in order to start the communication between them. An AAA factory is able to detect agents by moving the courier along the platen and recognizing an optical signal, sent by manipulator agents.

For a high precision factory the position of the agents needs to be known with the same high accuracy.

$C_4$ **Coordination Sensor**:
Detecting a manipulator as described in $C_3$ is only useful if the exact position of the agent can be determined. The sensor as well as the transformation into a readable position is part of the preparation for $C_5$.

$C_5$ **Controller**:
The courier needs a precise method in order to determine the exact position of a manipulator. This position is needed for the calibration as well as the operation of the factory.

The communication between different types of agents, including the establishment of the communication is presented in $C_6$ and $C_7$.

$C_6$ **Modulation on IR-Signal**:
A factory has more than one manipulator. In order to establish a communication between agents, the courier must determine which manipulator was detected. This is done with help of a modulation on the IR signal.

An ID is modulated onto the carrier frequency using AM. The courier demodulates the signal and extract the ID of the manipulator.

$C_7$ **Finding Network Agent**:
The courier is able to communicate with a manipulator by searching for the interface on the Ethernet. The detected ID from $C_6$ is used during the broadcast. This enables the factory to establish a communication to all detected agents during the initialization.

Another part of this work is finding methods which support the calibration of the agents ($C_8$) and updating the representation in an interface ($C_9 - C_{11}$) in order to simplify the path planing and programming for user.

$C_8$ **Updating Agent**:
An agent needs to know its own position in order to know if another agent is near itself. Since each courier has its own relative coordinate system during the initialization, a common system needs to be established. Communication between agents allows the exchange of transformation matrices, exchanging the relative coordinate system with a global one.

$C_9$ **Receiving Updates**:
The calibration of the factory not only needs to be done for each agent, it is also crucial that the Interface Tool represents the exact position of the agents, providing the correct feedback to an user.

$C_{10}$ **Position Calculation**:
Since the Interface Tool has its own coordinate system based on a parent/child relationship between elements in the factory, the relative position of an agent needs to be calculated after the agent has been updated in the Interface Tool ($C_9$).

$C_{11}$ **Model Update**:
An accurate model is not only needed for representing the state of the hardware factory, it is also useful for future development by using it as a simulation model. As such the updated model ($C_9$ & $C_{10}$) is stored as file, that can be used to develop new products in the factory without the need of the physical system. This fastens the development process, since more then one simulation can run at the same time.

In summary, this work helps to shorten the development process as well as the assembly of a factory system. Contributions $C_1 - C_{11}$ were defined in the requirements $R_2$ & $R_3$ (see section 1.1) which leads to the described outcomes $O_2$ & $O_3$ (see section 7.2). To the best of the authors knowledge there are currently no assembly systems featuring automatic calibration and agent detection helping the setup of an assembly system.

The whole factory is designed for rapid-prototyping of small scale and high precision parts. As such the setup of the assembly system can change quickly. A generic approach to endeffectors enables the factory to be agile. In this work a general approach to endeffectors ($C_{12}$ & $C_{13}$) was described and implemented including the use of dedicated processing units which enables decentralized computation during operation.

$C_{12}$ **Design of a general endeffector-hardware**:
The endeffector needs to have a defined interface which connects it to the manipulator. Pre-defined weight and size restrictions needs to be considered resulting in a modular basic actor, which can be used as base definition of task-specific endeffectors. The included processing unit allows the storage of the CAD model of the endeffector as well as the computation of endeffector specific functions.

$C_{13}$ **Design of a general endeffector-software**:
A dedicated processing unit inside an endeffector allows operators to change endeffectors quickly between manipulators without changing the source code of the operation in itself. Since the parts manipulation procedure is specific to each endeffector the programmer needs to only program each endeffector once. The manipulator inherits the endeffector and can call the specific methods without knowledge about the inner working of an endeffector.

In summary this contributions help to maintain the paradigm of decentralized computation during operating since each agent now has its own processing unit which operates the specified part autonomously. Contributions $C_{12}$ & $C_{13}$ were defined in the requirement $R_1$ (see section 1.1) which leads to the described outcome $O_1$ (see section 7.2).

Globalization is a buzzword in the industry nowadays. It is very important to control factories and get production statistics from around the world. As such the communication between the Interface Tool and the factory should not be dependent on a propriety protocol which only works in a local subnet environment. In this work the communication was exchanged ($C_{14}$) which enables the control of factories from any place in the world where an internet connection exists.

$C_{14}$ **Communication between Interface Tool and agents**:
The need of controlling a factory remotely has certain requirements to the protocol used for communication between agents and the Interface Tool. By changing the communication from the propriety protocol IPT to standardized websockets it is possible to control a factory remotely if there is an internet connection on both ends of the communication.

Contribution $C_{14}$ was defined in the requirement $R_4$ (see section 1.1) which leads to the described outcome $O_4$ (see section 7.2).

# 7 Summary, Conclusion and Outlook

In this chapter the work is summarized and the main achievements and outcomes are discussed. This chapter concludes with an outlook about possible future work for AAA assembly systems.

## 7.1 Summary

In this thesis an approach was defined which allows an automatic calibration in assembly factories. The focus of the calibration is on detecting the different kinds of agents and initialize the factory. Additionally updating the representation of a Graphical User Interface in form of the Interface Tool was discussed.

First, relevant terms and definitions were outlined. This included the concept of Agile Assembly Architecture and the components of the test environment, the Minifactory. Second, an overview about the Interface Tool was provided and the generation of simulation and hardware factory was discussed. The different communication frameworks and approaches, like websockets and publish/subscribe frameworks for network traffic as well as $I^2C$ and optical communication, used in the Minifactory were presented.

Furthermore the design and implementation of the agent initialization and detection was thoroughly discussed. The first contact of a courier agent to a manipulator agent requires the initialization of the built in relative coordinate system of the courier as well as the ability to map the area. A precision navigation using an optical sensor to detect the manipulator was used to determine the general position and a controller enabled the courier to drive to the exact position of the agent.

After the courier got the ID of the manipulator, the agent was able to get additional information like the name and type of the manipulator by using the LCM framework. The main communication was demonstrated by using DDS, a real-time messaging framework. After the position of the manipulator was determined the Interface Tool could be updated and the coordinates for the visualization were calculated.

As a practical proof-of-concept application the detection of a manipulator agent by a courier was presented and used as demonstration throughout the description of the factory initialization.

## 7.2 Conclusion

For this thesis, a set of requirements and objectives has been identified in chapter 1. Each requirement $R_i$ has a corresponding outcome $O_i$. This section outlines first how those requirements are fulfilled by their corresponding outcome. Finally the main achievements are summarized.

$O_1$ **Decentralized computation during operation**.
The design of the AAA is based on decentralized computation. After the design of the factory and the programming is completed, the instructions for operating are uploaded to each agent. Each agent is able to operate without any external instructions.

$O_2$ **Initialization without prior knowledge**.
During the start of the factory, each agent runs an individual initialization routine. For manipulators, this includes driving the axis in the home position and learning the ID of the connected end-effector. Courier agents map the area and detect manipulator agents. A communication is established between the agents.

$O_3$ **General approach to manipulating parts**.
Each end-effector is specific to the task it should handle. A general approach to end-effectors was established, including the recommendation of a small computing unit that fits into the end-effector. Different kind of sensors, like camera and force-sensor, were tested.

$O_4$ **Geographically independent control of factories**.
The introduction of websockets in Interface Tool allows operators to control AAA systems from any point in the world where a connection to the internet exists.

The presented solution in this thesis primarily addresses two of the stated requirements. The decentralized computation and initialization are discussed and explained. Since the end-effector is specific for each task only a generic approach is provided.

A prototype implementation in the form of the Minifactory demonstrated the success of initialization and operation of the newly designed end-effectors. The update of the Interface Tool was shown, which represents the success of the communication establishment.

## 7.3 Future work

Future work in this domain includes the investigation and development of additional end-effectors, couriers, trajectories and the application in future use cases. The application scenarios of these use cases are not limited to high precision factory work, but may also include the processing like chemical synthesis (e.g. dispensing liquids such as Deoxyribonucleic acid (DNA) onto chips) or biological assay (e.g. testing of biological material such as cancer).

Future work in the domain of Agile Assembly Architecture needs to therefore focus on extending the system to automatically determine the end-effector of each manipulator as well as including automated load-balancing between similar manipulators. This needs to be achieved by allowing the handling of trajectory planning and resource reservation to be part of the agent and not having to be programmed manually.

In a first step, the concept of path planning needs to be included in the courier framework and the high level API needs to offer rendezvous between agents. It will then be possible to perform an automated path planning between two agents, by using the data collected during the initialization of the factory.

# Bibliography

[1] R. Hollis and J. Gowdy, "Miniature factories: tabletop assembly systems for mechatronic products," *IARP Micro Robots and Systems Conference*, Oct. 1998.

[2] R. Hollis, A. Rizzi, H. Brown, A. Quaid, and Z. Butler, "Toward a second-generation minifactory for precision assembly," in *Proc. Intl Advanced Robotics Program*, Moscow, Russia, Apr. 2003.

[3] R. Hollis, D. O'Halloran, G. Fedder, N. Sarkar, and J. Jones, "Vision guided pick and place in a minifactory environment," in *5th Intl Symp. on Microfactories*, Besancon, France, Oct. 2006.

[4] P. Muir, A. Rizzi, and J. Gowdy, "Minifactory: A precision assembly system adaptable to the product life cycle," *Architectures, Networks, and Intelligent Systems for Manufacturing Integration*, vol. 3203, pp. 74–80, 1997.

[5] E. Puik, D. Telgen, L. van Moergeste, and D. Ceglarek, "Qualitative product/process modelling for reconfigurable manufacturing systems," in *Assembly and Manufacturing (ISAM), 2013 IEEE International Symposium on*, July 2013, pp. 214–218.

[6] M. Molhanec, "Agile product design - a modern approach to quality improvement," in *Proceedings of the 36th International Spring Seminar on Electronics Technology*, May 2013, pp. 178–182.

[7] L. S. Belisário and H. Pierreval, "A conceptual framework for analyzing adaptable and reconfigurable manufacturing systems," in *Industrial Engineering and Systems Management (IESM), Proceedings of 2013 International Conference on*, Oct 2013, pp. 1–7.

[8] N. Kaur, R. Harrison, and A. A. West, "A service-oriented approach to embedded component-based manufacturing automation," in *Industrial Technology (ICIT), 2015 IEEE International Conference on*, March 2015, pp. 2964–2969.

[9] C. Scheuermann, S. Verclas, and B. Bruegge, "Agile factory - an example of an industry 4.0 manufacturing process," in *Cyber-Physical Systems, Networks, and Applications (CPSNA), 2015 IEEE 3rd International Conference on*, Aug 2015, pp. 43–47.

[10] R. Hollis and A. Quaid, "An Architecture for Agile Assembly," *American Society of Precision Engineering, 10th Annual Mtg.*, Oct. 1995.

[11] J. Gowdy and Z. Butler, "An integrated interface tool for the architecture for agile assembly," *IEEE International Conference on Robotics and Automation*, pp. 3097–3102, May 1999.

[12] R. Hollis, "Minifactory: An Architecture for Agile Assembly," online, Sep. 2013. [Online]. Available: http://www.msl.ri.cmu.edu/projects/minifactory/

[13] S. Kume and A. Rizzi, “A high-performance network infrastructure and protocols for distributed automation,” in *Proceedings of the 2001 IEEE International conference on Robotics & Automation*, vol. 3, Seoul, Korea, May 2001, pp. 3121–3126.

[14] B. Siciliano, L. Sciavicco, L. Villani, and G. Orilo, *Robotics: Modelling, Planning and Control.* Springer, 2009.

[15] R. Hollis and J. Gowdy, “Miniature factories for precision assembly,” in *Intl Workshop on MicroFactories*, Tsukuba, Japan, Dec. 1998, pp. 9 – 14.

[16] A. Quaid, “A miniature mobile parts feeder: operating principles and simulation results,” in *Proceedings of the 1999 IEEE International conference on Robotics & Automation*, Detroit, Michigan, May 1999, pp. 2221–2226.

[17] A. Rizzi and A. Quaid, “Exploiting redundancy for autonomous calibration of a planar robot,” *Experimental Robots VI: the 6th International Symposium*, Mar. 1999.

[18] W. Ma, A. Rizzi, and R. Hollis, “Optical coordination sensor for precision cooperating robots,” *IEEE International Conference on Robotics and Automation*, pp. 1621–1626, Apr. 2000.

[19] H. Brown, P. Muir, A. Rizzi, M. Sensi, and R. Hollis, “A precision manipulator module for assembly in a minifactory environment,” in *Proc. Intl Conf. on Intelligent Robots and Systems*, Maui, Hawaii, Oct. 2001.

[20] BlackBerry, “QNX Operating Systems,” online, 2015. [Online]. Available: http://www.qnx.com/products/neutrino-rtos/index.html

[21] J. Gowdy and A. Rizzi, “Programming in the architecture for agile assembly,” *IEEE International Conference on Robotics and Automation*, pp. 3103–3108, May 1999.

[22] M. Chen, S. Kume, A. Rizzi, and R. Hollis, “visually guided coordination for distributed precision assemlby,” *IEEE International Conference on Robotics and Automation*, pp. 1651–1656, Apr. 2000.

[23] A. Rizzi, J. Gowdy, and R. Hollis, “Distributed coordination in modulare precision assembly systems,” *The international journal of robotics research*, pp. 819–838, Oct. 2001.

[24] I. Fette, G. Inc., A. Melnikov, and I. Ldt. (2011, Dec.) The WebSocket Protocol. online. Internet Engineering Task Force (IETF). [Online]. Available: https://tools.ietf.org/html/rfc6455

[25] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns.* Addison-Wesley, 1994.

[26] Object Management Group, “DDS The proven data connectivity standard for the IoT,” online, Mar. 2016. [Online]. Available: http://portals.omg.org/dds/

[27] Real-Time Innovations, “RTI Connext DDS Professional,” online, Feb. 2016. [Online]. Available: https://www.rti.com/products/dds/

[28] Raspberry Pi Foundation, “Raspberry Pi,” online, Feb. 2016. [Online]. Available: https://www.raspberrypi.org/

[29] R. DeLuca, A. Rizzi, and R. Hollis, “force-based interaction for distributed precision assembly,” *Proc. Int’l Symp. on Experimental Robotics*, Dec. 2000.

[30] “Lightweight Communications and Marshalling (LCM),” online, Mar. 2016. [Online]. Available: https://lcm-proj.github.io/index.html

[31] A. Huang, E. Olson, and D. Moore, “LCM: Leightweight Communications and Marshalling,” p. 6, Oct. 2010.

[32] “I2C Info,” online, 2016. [Online]. Available: http://i2c.info

[33] telos Systementwicklung GmbH, “I2C Bus,” online, Mar. 2016. [Online]. Available: http://www.i2c-bus.org

[34] R. Hollis, J. Gowdy, and A. Rizzi, “Design and development of a tabletop precision assembly system,” *Mechatronics and Robotics*, pp. 1619–1623, Sep. 2004.

[35] R. Quinn, G. Causey, F. Merat, D. Sargent, N. Barendt, W. Newman, J. Virgilio, B. Valesco, A. Podgurski, J. Jo, L. Sterling, and Y. Kim, “Design of an agile manufacturing workcell for light mechanical applications,” in *Proceedigns of the 1996 IEEE International Conference on Robotics and Automation*, vol. 1. IEEE, Apr. 1996, pp. 858 – 863.

[36] M. Mehrabi, A. Ulsoy, and Y. Koren, “Reconfigurable manufacturing systems: Key to future manufacturing,” in *Journal of Intelligent Manufacturing*, vol. 11, no. 4, Aug. 2000, pp. 403 – 419.

[37] *GERAM: Generalised Enterprise Reference Architecture and Methodology*, IPIP-IFAC Task Force Std., Rev. 1.6.2, Mar. 1998.

[38] J. Christensen, “Holonic manufacturing systems: initial architecture and standard directions,” in *Proceedings of First European Conference on Holonic Manufacturing Systems*, 1994.