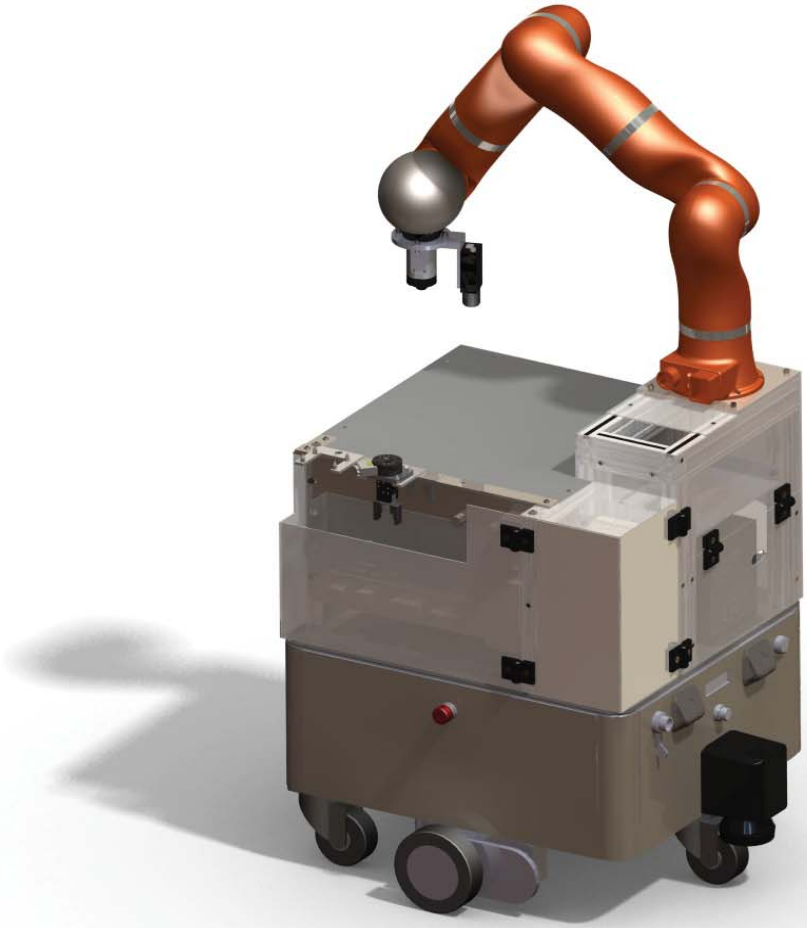
Integration of the KUKA Light Weight Robot in a mobile manipulator



Mikkel Rath Pedersen

Master's Thesis in Manufacturing Technology
Department of Mechanical and Manufacturing Engineering
Aalborg University

Department of Mechanical
& Manufacturing Engineering

Fibigerstræde 16,
DK-9220 Aalborg Ø
Phone: +45 9940 8934
Fax: +45 9815 3040
http://www.m-tech.aau.dk

Title:
Integration of the KUKA Light Weight Robot in a mobile manipulator

Semester themes:
3^{rd}
Integrated product and system design
4^{th}
Manufacturing Technology

Project period:
2010/09/02
- 2011/05/31

Project group:
FIB14,33(b)

Participant:
Mikkel Rath Pedersen

Supervisor:
Ole Madsen

Amount printed: 3

Pages: 90 + Appendix

Ended: 2011/05/31

Synopsis:

The following project describes the integration of the KUKA Light Weight Robot on a new version of the AAU-developed mobile manipulator Little Helper. With the current configuration being analyzed to form a basis of the design, the LHP is designed. However, due to a larger controller, the pneumatic system is removed, resulting in the need for a non-pneumatic tool changer. Therefore this is designed by electrically actuating a commercially available one. The LHP is designed so the capabilities exceed those of the LH, but has yet to be built upon the completion of this project.
After the design is complete, the work on demonstrating the added benefits of the KUKA LWR is carried out. This reveals that some of the added features improve the functionality of the LHP, mainly the use of the Fast Research Interface, which enables realtime control of the LWR from a remote PC contributes to this. Furthermore, the peg in hole task is solved using the cartesian impedance controller of the LWR, and a routine for calibrating a workstation coordinate system by the torque sensors is established.
Conclusively, it is determined that the LWR indeed increases the capabilities of the LHP, compared to the LH.

Abstract

In 2008 the mobile manipulator Little Helper was developed at the Department of Mechanical and Manufacturing Engineering of Aalborg University, in order to create a more flexible and autonomous solution than traditional automation in the industry. Research has been made since then to improve this solution, and within the last year two EU-funded projects have begun, with the Little Helper as a central aspect. Part of these projects is to redesign the Little Helper to accommodate the KUKA Light Weight Robot instead of the currently attached robot arm, to gain increased functionality in the form of greater reach and payload and force feedback, and to utilize this in different scenarios. The project at hand deals partially with both these aspects.

In order to design the new version, the Little Helper Plus, the configuration of the Little Helper has been analyzed, to form a basis for the new design. This analysis reveals the need for a more easily reconfigurable solution, in order to provide a more flexible solution on the hardware side. After this initial analysis, the main aspect of placing the much larger controller for the LWR on the mobile platform is investigated. This results in the choice of removing the pneumatic system on the Little Helper Plus, since there is no room for it. Since the tool changing capability of the Little Helper is to be maintained as part of the flexibility aspect, a non-pneumatic tool changer is designed. An analysis of the different methods of changing tools without a pneumatic system reveals that electrically actuating an existing tool changer is the best option. As such, a fully working electric tool changer is designed for the Little Helper Plus.

After this initial work, the Little Helper Plus is designed from the inside out, by placing the components in CAD software and afterwards designing the chassis, cover plates, brackets, etc. This results in a fully designed Little Helper Plus, with capabilities surpassing those of the Little Helper, though the actual build of the system has not been carried out by the end of this project.

Because of this, the LWR is mounted in a temporary location, in order to investigate the added functionality this provides. The control modes and added functionality of the LWR is described first, to provide an understanding of this. The preliminary investigation of easier programming by moving the robot by hand, instead of using the jog keys on the teach pendant, reveals that this seemingly more elegant programming method in fact takes more time. This is because the position and rotation of the robot tool can

not be ensured with adequate precision when moving the robot by hand, compared to using the jog keys.
The use of the torque sensors in the LWR for calibration purposes is investigated next, since workstation calibration is currently done by machine vision on the Little Helper. This reveals that a calibration adequate for parts handling and vision inspection using the torque sensors is feasible, since a fine repeat accuracy of the calibration is obtainable.
With traditional robots, the task of placing a peg in a hole is quite difficult, due to demands on accurate position and orientation of the peg relative to the hole. It is therefore investigated how the features of the LWR can be utilized to solve this task, by comparing a traditional, position controlled method of insertion with insertion using the cartesian impedance controller. This confirms that the task is indeed hard to accomplish using the position controller, whereas using the cartesian impedance controller allows for greater error in position and rotation of the peg. Furthermore, methods for improving this task is suggested, along with a strategy for inserting sharp-edged pegs in ditto holes.
Along with the LWR, KUKA supplies the Fast Research Interface, enabling realtime control of the LWR through an UDP connection. Since this will be utilized in the Little Helper Plus for sending commands from the main computer on the platform to the robot arm, work has been done on making a full demonstration of this interface. Initially, a console application outputting the measured torque in each joint every second is developed, to gain an understanding of how the interface works. After this a GUI application is developed to demonstrate all of the features of the FRI, including an interface for jogging the robot from the remote PC.
Finally an attempt on measuring the mass and center of mass using the torque sensors has been carried out. This concludes that the precision of the torque sensors is inadequate for these kinds of measurements, since the measurements yield fairly inaccurate results. This of course depends on the specific application, where this would be utilized, since e.g. determining whether or not there are parts in a box is feasible, whereas determining the actual number of parts in the box is not.
It is apparent that the KUKA LWR in most ways contribute to the functionality of the Little Helper Plus, mainly due to the Fast Research Interface and the option to use the built-in compliance control of the robot arm. Though some scenarios have been investigated in this project, there are a lot of possibilities for further work with using the LHP in general and the LWR in particular.

Preface

This 2-semester master's thesis project is composed on the 3rd and 4th semester of Manufacturing Technology at Aalborg University, from September 1st 2010 to May 31st 2011. The semester theme for each semester is integrated product- and system design (3rd) and manufacturing technology (4th).
The project is documented by a main report, with included appendices and an enclosed CD. The main report can be read independently but is supported by the appendices and literature references.
The references in the text are made by the IEEE method and labeled with consecutive numbering in square brackets, e.g. [1] or [2]. Further information about the reference can be seen in bibliography. References to files on the CD are shown as `path\filename.type`.
Figures, equations and tables are numbered by the chapter number and a consecutive number e.g. **Figure 4.1**.
On the enclosed CD can be found:

- Source code for various sub-projects
  - C++ applications
  - KRL programs
- Bibliography
- Manuals regarding the LWR
- SolidWorks 2010 files of the new configuration of Little Helper
- Videos showing the tasks programmed on the LWR

To view the KRL source codes, it is recommended to download and install the open source editor Notepad++, and install the `userDefineLang.xml` file found on the CD (with instructions), to improve readability by ensuring proper highlighting of the code. To view and compile the C++ projects, the free Visual Studio C++ Express (VC++) 2010 is needed for the two first projects, and the 2008 version is needed for the FRI application (only for compilation). Since all of these projects require a connection to

some hardware, compilation is perhaps not required, and thus any version will highlight the code properly. Notepad++ will do this as well, but VC++ will show the relationship between the source files as well as the actual code.
Besides the regular project work in this project, the following activities have been carried out throughout the project period:

- Preparation and instruction a part of a Ph.D. course in Robot Vision
- Preparation and demonstration of the mobile robot Little Helper at a stand at the fair FoodPharmaTech 10 (Nov. 2^{nd} - 4^{th} 2010)
- Participation in a KUKA programming course (Basic Robot Programming, Advanced Robot Programming, some elements of Expert Robot Programming and LWR Programming) at KUKA College in Gersthofen, Germany
- Preparation and conduction of a LWR course for the Automation Group at the Department of Mechanical and Manufacturing Engineering

Mikkel Rath Pedersen

Contents

Abstract i

Preface iii

1 Introduction 1

2 Description 5

2.1 The mobile manipulator Little Helper 5

2.2 The TAPAS project 6

2.3 The KUKA LWR 6

3 Thesis statement 9

I Reconfiguration of the platform 11

4 Configuration of the Little Helper 13

4.1 Components 13

4.2 Construction 15

4.3 Connections 17

5 Hardware changes 19

5.1 Requirements for the Little Helper Plus 21

6 Replaced components 23

6.1 Tool changing 23

6.2 Switching board 31

6.3 Vision system 33

7 Configuration of the Little Helper Plus 35

7.1 Main housing 35

7.2 Tooling 42
7.3 Manufacturing of parts 44
7.4 Power and signal connections 45
7.5 Summary 46

II Capabilities of the KUKA LWR **47**

8 Control of the LWR **49**
8.1 Control strategies 49
8.2 Controlling through the `$STIFFNESS` structure 51
8.3 Built-in LWR functions in KRL 51

9 Programming the LWR by demonstration **53**

10 Workstation calibration using force sensing **55**
10.1 Theoretical solution 56
10.2 Test setup and programming the LWR 58
10.3 Accuracy of the calibration 60

11 Peg in hole **65**
11.1 Using Position control 66
11.2 Using Cartesian Impedance control 67
11.3 Implementation in a production environment 68

12 Demonstration of the Fast Research Interface **71**
12.1 Function of the FRI 71
12.2 Hello FRI 74
12.3 Full demonstration of the FRI 76
12.4 Additional remarks 85

13 Measuring mass and center of mass of parts **87**
13.1 Test setup and method of measurement 87
13.2 Determining the actual values 90
13.3 Accuracy of the measurements 90

13.4 Further work on the measurements 93

14 Conclusion **95**

Bibliography **100**

Appendix **103**

A Case: Vision-controlled robot playing NIM **103**

A.1 Overview 103

A.2 Manipulation 105

A.3 Vision 105

A.4 Overall structure of the game 107

B FoodPharmaTech '10 **111**

B.1 Description of the setup 111

B.2 Modifications during the fair 114

C Setup of the LWR **115**

C.1 Startup and configuration of connectors 116

C.2 Connecting to end-effector equipment 117

C.3 Establishing I/Os on the controller 118

C.4 Temporary tools 120

1 Introduction

The use of automated solutions in manufacturing is increasing, and has been since the first introduction of an automated robot in an industrial environment in the 60's. Sales did take a serious drop of 47% in 2009, but is getting back on track with an annual worldwide sale in 2010 of about 115.000 robots (a slight increase compared to 2008). The estimated sales of 2011 to 2013 also point towards an increase, both in annual sales and in the number of operational robots in all industries. The reason for this is a combination of more intelligent solutions, combined with both a price drop and an increase in wages for human labor.

Although the sales and the number of robots in use are increasing in the next few years, we have yet to see a great boom in these numbers. One reason for this is that industrial robots have not changed that much since the 60's, in terms of general principles. Industrial robots are still stationary and not that smart and flexible, though the introduction of machine vision has enabled robots to "see," picking up arbitrarily placed parts and perform simple decision making regarding the part being processed, as well as performing simple quality control. Compared with the overall need for more flexible production facilities, the traditional robot cells seem outdated, since the number of repetitive tasks, where a robot is particularly useful, is decreasing as the flexibility of the production facility increases.

One way to introduce flexible automation in production facilities is the use of mobile platforms. This principle is known from service robots in particular, and the estimated increase in sales is definitely apparent from Figure 1.1. However, the use of mobile robots in production facilities has yet to see its breakthrough. The many advantages of mobile robots does however look promising, since the versatility of a mobile robot suggests it could replace human labor over time.

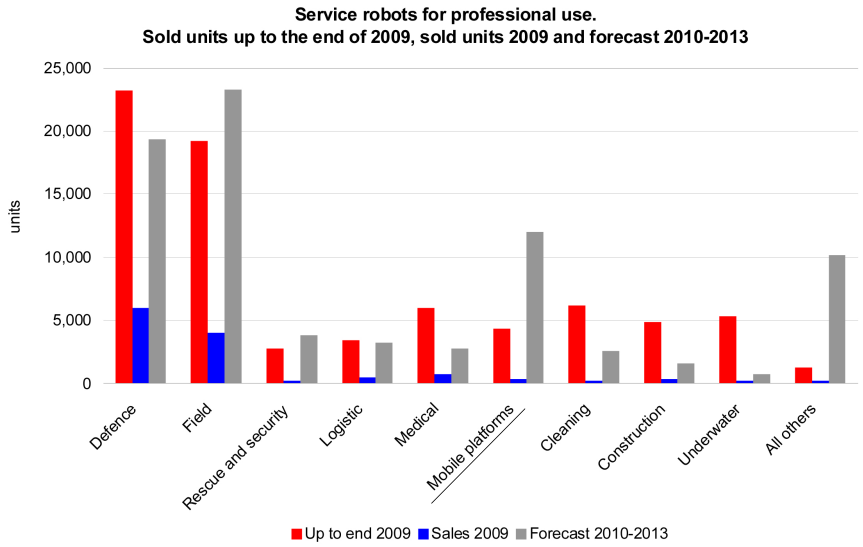


Figure 1.1: Development in the sales of service robots [1]

A fully working mobile manipulator, called Little Helper (LH), has been developed at the Department of Mechanical and Manufacturing Engineering (M-TECH) at Aalborg University, for both proof-of-concept and further research into the use of mobile manipulators in industrial environments. Further research is funded through the TAPAS program, which is an EU-funded research project in collaboration with Grundfos, KUKA and others. The manipulator is illustrated in Figure 2.1.

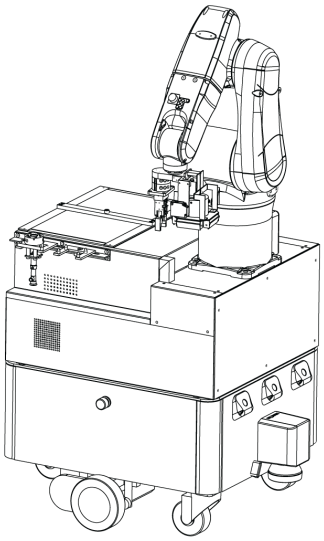


Figure 1.2: The mobile manipulator Little Helper [2]

The goal of the TAPAS program is to create a flexible robot, capable of accomplishing several different tasks at various locations in the production facility, with an on-board scheduling software planning the execution of these tasks. The robot can be programmed like any stationary robot, though an integration of the programming of the platform is also required to integrate the mobility. The programming is however still rather time-consuming and difficult, like in many other robot applications, greatly reducing the flexibility and adjustments. The next step is therefore to introduce a more intuitive method of interacting with the robot, e.g. instructing new tasks, without the need of direct programming.
Several ways to do this is being researched all over the world, not only for mobile robots, but also for faster reprogramming of stationary robots. One way to do this is the established method of online programming, where the user teaches specific coordinates to the robot through a teach pendant. This method is usually very time consuming, and in some cases causes the rest of the production line to be stopped, making the reprogramming cost intensive as well. The cost of reprogramming a mobile robot is however smaller, at least if the robot is not handling a single crucial task, causing the reprogramming process to stop the production line.
The robot company KUKA has a different answer to this, in the form of their lightweight robot (LWR), which has also been showcased attached to a mobile platform. This 7-axis manipulator can sense the torques in each joint, which has a number of beneficial effects. One of the most important, in the mobile robot context, is safety, since the LWR can detect if it is hitting an obstacle, this being both humans and production equipment. Another beneficial effect is the much easier way of teaching new tasks to the robot, since the user can physically move the robot to each coordinate, directly controlling each joint angle and the position of the robot tool. The force sensing capabilities of the robot can also reduce the need for calibration at each workstation, since the robot can "feel" the parts it has to manipulate.
Through both the TAPAS and GISA project, M-TECH has ordered two KUKA LWRs, one of which is to be attached to a mobile platform similar to that of the Little Helper, thus creating another mobile manipulator. The LWRs are delivered in February and March 2011, but has to be implemented on the Little Helper in a minimal amount of time. This of course raises the initial questions:

How can the KUKA LWR be installed on a mobile platform?
How can the potential of the KUKA LWR be fully utilized in a mobile application?

2 Description

This chapter aims to briefly describe parts of this project the reader should be familiar with. The chapter is in no way meant as a full introduction to the three topics presented, but the reader is encouraged to explore the literature for further information on each topic.

2.1 The mobile manipulator Little Helper

This section will briefly describe the foundation of the Little Helper, and the logic behind the development of it.
The whole idea behind the LH is as simple as any other idea regarding automation; to eliminate the use of human labor for mundane, repetitive tasks. With the increasing demand for flexible production, however, the disadvantages of automation becomes more obvious, since the teaching of new routines and the configuration of workcells and production equipment in some cases has to be changed when a new product is introduced in the environment. Thus, there is a need for LH in the future of manufacturing, and it is this need that originally generated the idea to essentially create a production worker that does not require salary [2].
Built around the concept of replacing more flexible human labor, the design requirements were essentially that, i.e. having a high degree of automation that works out of the box, is easily adjustable, user friendly and highly flexible, by incorporating the system on a mobile platform. Furthermore, the quantitative requirements were to be comparable to human labor, especially with regards to reach, payload ($2kg$) and time of operation ($7hr$ shift), with some parameters even exceeding the capabilities of a human, for instance the precision of the manipulator [2].
Due to this being a prototype, the number of tasks the Little Helper should be able to accomplish was limited to transport, pick-and-place, quality control and classification operations. After the initial development, further research has been done on interacting with multiple agents (production equipment, feeders, inspection stations etc.), performing different tasks at each agent. The Little Helper is still characterized by being a prototype, however, and has not been tested in an actual manufacturing environment

before the start of this project.

2.2 The TAPAS project

The TAPAS project is a partly EU-funded research project aiming to bridge the gap between academia and industry, regarding the development of flexible automation. The demand for both high volume and high product variety in manufacturing is directly creating the need for flexible automation to maintain competitiveness, and the industry is lacking both time and finances to carry out this research.
The project is a collaboration between the industry partners *KUKA Roboter GmbH*, *Grundfos A/S* and *Convergent Information Technologies GmbH* and the academic partners *Aalborg University*, *Alberts-Ludwig University* and *DLR*.
The scope of the project is divided into three objectives:

1. Robot logistics and automation, regarding part logistics, extended logistics services and assisting the existing production equipment.

2. R&D in ICT[1] regarding automated mission planning and control, with respect to path planning, navigation, scheduling and communication with the existing production equipment and ERP system.

3. Sustainable solutions for new applications of robots, regarding testing and validation, pilot installations at production facilities and serving both the industry's interest in providing a wide range of products and the incorporation of the academical research in transformable automation solutions.

The project will be based on available robotic technologies, such as the Little Helper and the KUKA LWR, so no development will be made from scratch, making the project use case oriented, rather than technology push oriented. The works will be demonstrated at three demos in month 6, 24 and 39 of the project, respectively. The project kicked off in the beginning of December 2010.

2.3 The KUKA LWR

The LWR was originally developed by DLR, the German Aerospace Center, to investigate the possibility of using robots on space stations, the main aspect being to develop

[1] Information and communication technology

a light weight robot, with a high payload to weight ratio. The first configuration, called LBR I, was developed in 1991, lacking a lot of the functionality that Aalborg University's LWR 4+, which became available for purchase in 2008, has [3].

Figure 2.1: The KUKA LWR4+ shown beside the KCP teach pendant and the KRC2lr controller [3]

Apart from the aspect of creating a light-weight robot, another aspect of the development was the paradigm shift from conventional position control, where the position and motor current is measured, to compliance control, where the torque in each joint is measured instead of the motor current. This has some obvious benefits regarding safety, but also increases the ease of instructing new tasks, as the operator can now physically move the robot to each position, instead of using a traditional teach pendant.
Apart from the aforementioned paradigm shift to torque sensing in joints, the KUKA LWR is quite unlike traditional robots in a number of ways. Traditionally, robots remain in a stationary position in e.g. a factory, securely bolted to the floor and surrounded by fencing to prevent harm to people or damage to equipment. Furthermore, little thought have gone into minimizing the weight of robot arms, and the robot arm as we know it is quite unchanged since its earliest ancestor.
During the development of the KUKA LWR, this line of thought was obviously disregarded. The emphasis was laid on developing a light weight robot with added functionality compared to traditional robot arms.
Apart from the light weight and the effectively added sense of touch, another aspect of the LWR is that it is modeled after a human arm, resulting in the addition of an axis, yielding a total of seven axes. This addition enables the arm to reach the same point and orientation in space in an infinite number of orientations[2], due to the robot having

[2]In reality of course limited by the encoders in each joint

a redundant joint, compared to a traditional six axis robot, which can only reach the same point and orientation in a maximum of eight different ways. This greatly expands the flexibility of the robot, with regards to e.g. interacting with other production equipment, such as CNC milling centers.
The addition of an extra axis and the sense of touch has two great advantages in a production environment. The extra axis offers a much greater flexibility with regards to grasping or interacting with hard-to-reach objects, effectively reducing the need to adapt a current production facility to accommodate robotic solutions. The other advantage is in the same ballpark, since the force sensing capabilities could reduce the safety measures required for using the robot in an environment alongside human labor.

3 Thesis statement

In order to answer the initial questions, it is necessary to look at them separately, thus dividing the project into two coupled parts:

1. The redesign of the Little Helper to accommodate the KUKA LWR, while preserving the current capabilities of the mobile manipulator system
2. An investigation of the added functionality the KUKA LWR provides, with respect to increasing the driving thoughts behind the Little Helper (e.g. flexibility and ease-of-use)

Reconfiguration

In order to design a new mobile manipulator with the LWR as the robot arm, several steps have to be carried out. The main task, however, is to gain an understanding of the current configuration of Little Helper, which parts should be replaced in the new configuration, and which parts should replace them in the design of the new configuration, called Little Helper Plus. It would also be beneficial to investigate the need to add new components for increased functionality in the reconfiguration process, or incorporate an easy method for adding components at a later time.
The goals of the reconfiguration process are therefore:

- Analyze the current configuration
- Analyze the changes in components, and what design demands these changes facilitate
- Establish a requirements specification for the new configuration
- Redesign components to function in the new configuration
- Reconfigure the platform to accommodate the requirements specification, including placement of components, mechanical connections and electrical wiring

Applications of the KUKA LWR

In order to fully gain an understanding of the added functionalities of the KUKA LWR, a number of cases are investigated. The cases are chosen so they have a relevance to the LWR on the Little Helper Plus (LHP), in order to further increase the function of this, especially regarding the flexibility of this. The emphasis will not be placed on creating working tasks, however, but instead on investigating the possibility of using the LWR to solve these tasks, and gain an understanding of which benefits the LWR has compared to traditional robots, and which problems persist in the tasks. In order to do so, however, some configuration and installation of the robot is also required, and will be carried out as well. The cases on the LWR will be:

- Teaching new tasks to the robot by demonstration
- Workstation calibration by touching edges of a worktable
- Utilizing the torque sensors to solve the peg-in-hole task
- Using the Fast Research Interface with the LWR
- Weighing parts using the torque sensors in the robot

Part I

Reconfiguration of the platform

The following part describes the design and configuration of the Little Helper Plus, using the KUKA LWR as a manipulator. Only the hardware design has been carried out in this project, since the software architecture is still being developed as part of the TAPAS project. The goal of this part is therefore to describe the design process of the new mobile manipulator, with respect to installing the KUKA LWR on the mobile platform, based on the design of the Little Helper.

The incorporation of the KUKA LWR on the mobile platform presents some problems, which in turn leads to new design requirements, e.g. the design of an electric tool changer. These problems, along with their solutions, are presented in the following part as well.

4 Configuration of the Little Helper

This chapter describes the mobile manipulator as it was constructed originally, in 2008, and the minor modifications that have been added up to the start of this project period in September 2010. The purpose of the chapter is to describe the former configuration of the mobile manipulator, and as such to provide an understanding of the task at hand. The Little Helper has already been introduced, so the following will purely be a presentation of the hardware used on the Little Helper, and the interfaces between the components that enables them to function as a system.

4.1 Components

The components that make up the Little Helper can be divided into four subsystems that utilize technology that is widely used in the industry. It is the combination of the technologies that is the driving force behind Little Helper. The four subsystems are:

Platform system The system enabling the mobile aspect of the Little Helper. The platform is a commercially available, fully independent system, complete with sensors and control software. This is also seen in AGVs[1] in the industry.

Manipulator system Being a further development of traditional robot solutions, performing manufacturing processes, the platform is equipped with a single manipulator.

Vision system Machine vision is being used more and more in the industry, adding the sense of sight to robots. In order for the Little Helper to be sufficiently flexible, a vision system has been incorporated for both parts detection and calibration.

Tooling system Since a single robot tool is not sufficient to satisfy the flexibility need of the Little Helper, several tools are available for the Little Helper to use. The tooling subsystem is also containing the actuation and tool changing mechanisms.

[1] Automatic guided vehicles

In the following, each subsystem will be described further. A later section will deal with the construction of the system as a whole, thus showing the placement of the various components.

Platform system

The platform system is bought as an independent system from the company *NeoBotix*, who specializes in mobile platforms. The platform comes with two laser scanners and five ultrasonic sensors, used for navigation and path planning, a battery pack and an onboard computer. The battery pack and onboard computer is used as common power supply and control of the rest of the Little Helper as well. Furthermore, the onboard computer is equipped with a touchscreen, though interaction with the robot is usually done from a remotely connected device through VNC.

Manipulator system

More than just the actual manipulator, the manipulator system is everything enabling the function of the manipulator. Thus, the manipulator subsystem is the actual manipulator, an *Adept Viper s650*, the communication module *Adept SmartController CX*, and the power/signal module *Adept Motion Blox R60*. The two latter components are necessary for the function of the manipulator, handling the execution of programmed routines and managing power to motors in each joint. The manipulator, however, requires 230VAC, so part of the manipulator subsystem is also an inverter, converting the 24VDC from the batteries to the 230VAC required by the manipulator.

Vision system

A vision system is much more than a camera, which is obvious when looking at the components that make up the vision system. Apart from the IEEE1394[2] camera, the vision system is composed of an adjustable lens, four bar lights and a distance sensor for calibration. The three latter components all need their separate control, giving a total of seven different components in the vision subsystem. The use of the adjustable lens and the distance sensor is necessary for the calibration of the Little helper, since the position and angular tolerance of the platform is not sufficient for adequate manipulation.

Tooling

Besides the actual tools, the tooling system is everything regarding the changing, actuation and carrying of these tools. As such, the tooling system is composed of a compressor

[2]Popular known as the brand name FireWire

and air reservoir, a pneumatically driven tool changer and three different tools, that are also pneumatically driven. These tools are a suction cup, a parallel gripper and a specialized pallet gripper, the combination enabling the Little Helper to perform the majority of tasks that such a system should.

The components described in this section all have to be mounted on the platform, composing the entire system of the Little Helper. The construction of the entire system is introduced in the following section.

4.2 Construction

The Little Helper was designed on top of the NeoBotix platform, so the footprint of this platform is also the allowed footprint of whatever is mounted on top of it. The reason for this is that the control software of the platform is designed by NeoBotix for the platform's footprint, and as such can not avoid collision of anything outside this footprint. On top of the platform the main housing is built, containing most of the components mounted in a frame, which the manipulator is subsequentially mounted on top of.

Figure 4.1 shows the entire system, and the placement of the main components.

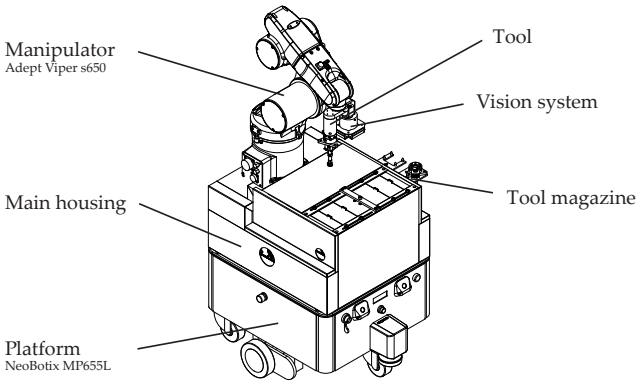


Figure 4.1: Complete system of the Little Helper

The main housing contains all the interior components necessary for the function of the Little Helper. The main housing is shown without cover plates in Figure 4.2, along with indications of the various components. Note that all components requiring surrounding air or direct air intake for cooling (i.e. the compressor, power/signal module, controller and inverter) are positioned adequately, to accommodate these demands.

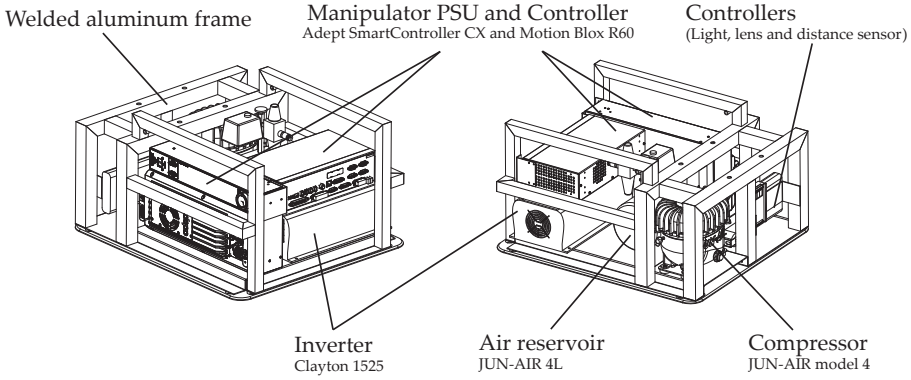


Figure 4.2: The main housing of the Little Helper shown from two different angles without cover plates

A noteworthy aspect of the construction of the Little Helper is the relatively compact design, with regards to the number of components. The compact design and close fit of components, however, decrease the modularity of the solution, since it is nearly impossible to replace or upgrade components, should the need arise.

4.2.1 Tooling

The tool of the manipulator is designed so the camera is attached to this, along with the gripper. The fact that the camera can be moved by the manipulator enables the use of vision for calibration at each workstation, enabling the manipulator to have an accuracy of $\pm 0,1mm$ after the high precision calibration. The use of an adjustable lens and barlights is determined by the need for Little Helper to function in changing environments, where traditional vision systems are more or less isolated from the surroundings, to provide optimal lighting. The complete tool, with the pneumatic suction cup mounted, is shown in Figure 4.3.

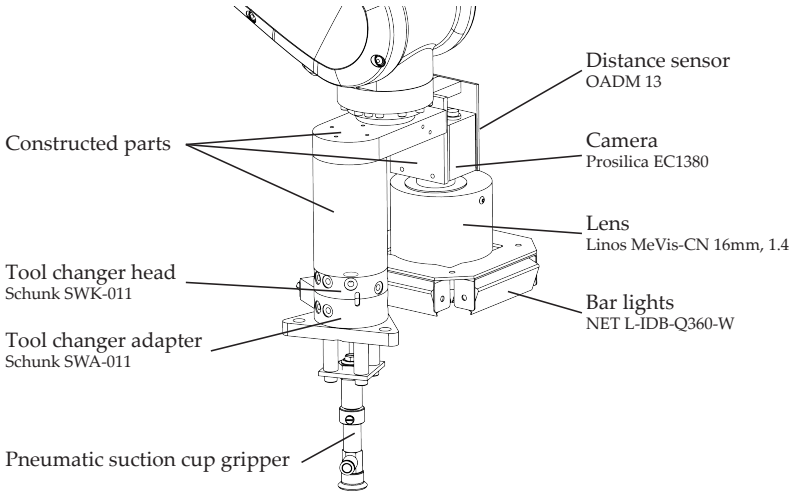


Figure 4.3: Tool consisting of both a gripper and a camera

4.3 Connections

An in-depth discussion and explanation of the connections between the various components is determined to be out of the scope of this report, since the electrical system naturally will be very different from the initial configuration. However, it is necessary to have an overview of the required connections to incorporate this in the redesign, therefore Figure 4.4 shows the connections graphically.

One thing to note from the figure, is the use of two power circuits rated at 24VDC and 230VAC, respectively. It is also apparent from the figure that all components are controlled, directly or indirectly, by the onboard platform computer.

The former configuration of the Little Helper has now been presented, to give the reader an understanding of the aspects of the redesign. Upon redesigning the Little Helper, it is not only necessary to focus on the required components and the placement of them, but also to allow room for mounting, cable connections and room for ventilation.

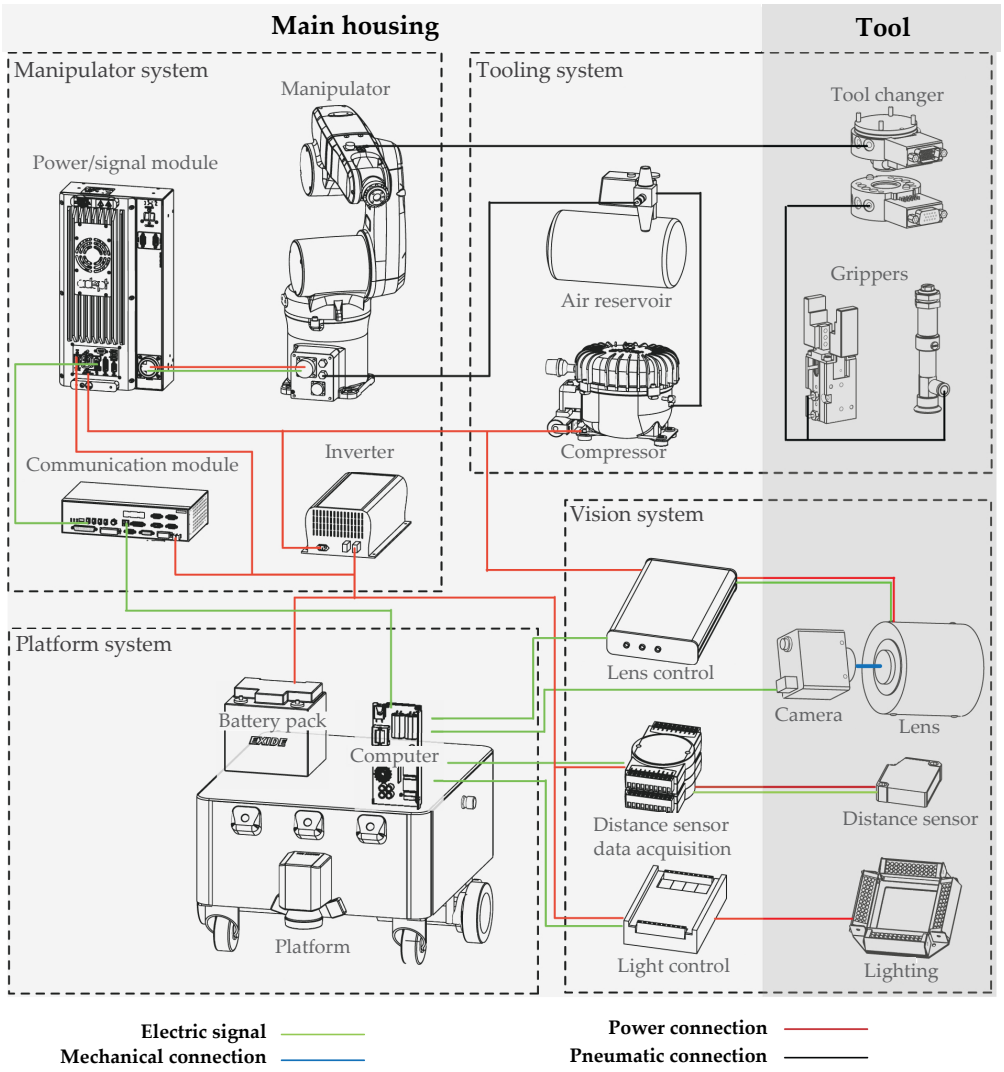
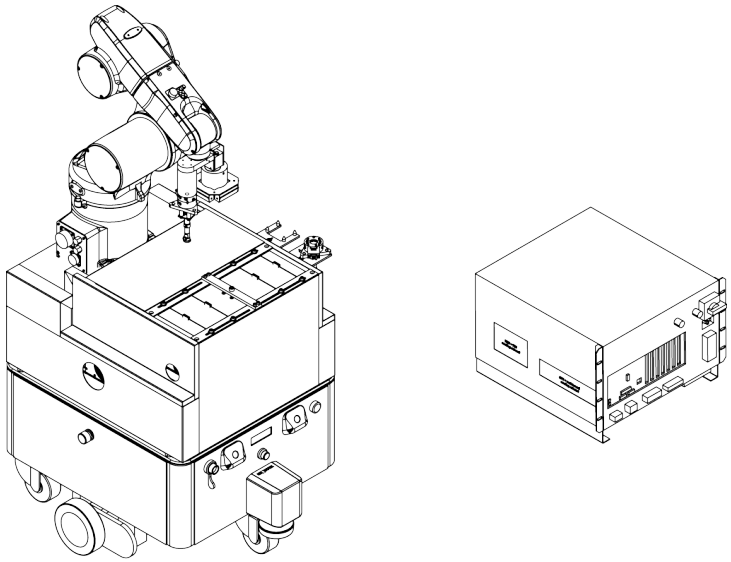


Figure 4.4: Connections between the components and subsystems of the Little Helper [2]

5 Hardware changes

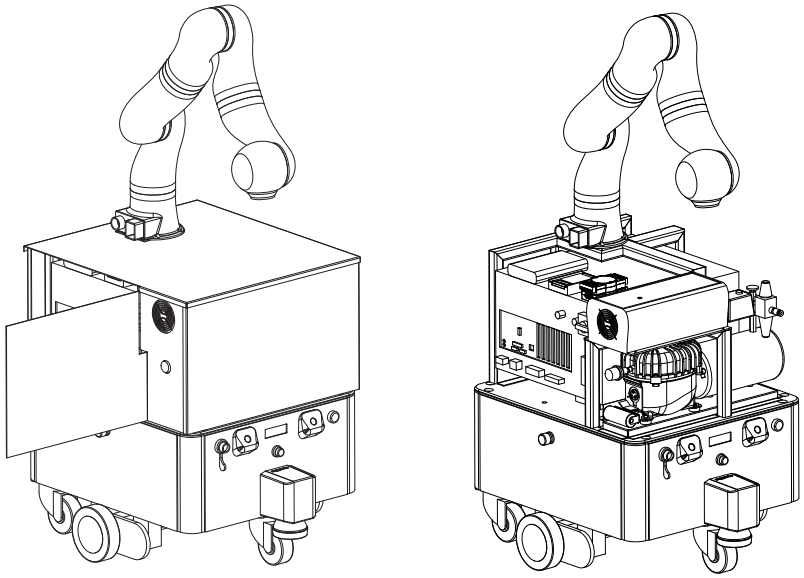
This chapter will provide an overview of the changes of the Little Helper, focusing only on the change in components, and not their mutual connections. Eventually, this will lead to a specification for the Little Helper Plus, after which the further work of placing components, designing the main housing etc. can be carried out. One remark has to be added to the following chapter, namely that the platform will not be changed for the Little Helper Plus, since an identical platform from Neobotix will be bought for this.
Obviously, the most apparent change of components is the replacement of the manipulator from the currently attached Adept Viper s650 to the new KUKA LWR. This, however, necessitates a replacement of the robot controller and power supply as well. The communication and power/signal module from KUKA is built as one unit, the KRC2lr. This controller[1] is designed to fit in a standard 19" rack cabinet, with a height of $313mm$, and as such yields the main problem of the reconfiguration of the Little Helper. The controller is shown besides the Little Helper for reference in Figure 5.1. KUKA is currently working on a new controller model, which is much more compact, to accommodate exactly this problem. This controller is however not available for purchase during the timeframe of this project, in a configuration that incorporates the added features of the LWR. Since the KUKA controller is much larger and bulkier than the two currently used Adept communication and power/signal modules, the room for other components is limited. This results in a series of choices that greatly affects the design of Little Helper Plus.
The mere size of the new controller has an immediate effect on the space left for the larger components mounted on the Little Helper. Several trials have been made on placing all the components of the former configuration in the new configuration, all of them yielding a very crammed design, with little or no room for the mounting frame, in some cases even essential components like the tool magazine. The most successful attempt is shown in Figure 5.2.

[1] The term controller will be used to describe the KRC2lr from this point on



(a) Little Helper (b) KUKA KRC2lr controller

Figure 5.1: Isometric view of the former configuration of the Little Helper shown beside the new KUKA controller



(a) With cover plates (b) Without cover plates

Figure 5.2: The most successful attempt of incorporating all components of the Little Helper in a new design along with the KUKA controller

As a result of the space consumed by the new controller, it is decided that the pneumatic system will be removed, since the compressor and air reservoir take up much of the space on the platform. This decision, however, has some undesirable direct consequences on the rest of the configuration:

1. All tools will then have to be electrically actuated, instead of the currently pneumatic actuation. This is purely an economic problem regarding the parallel gripper, since electrically actuated grippers are generally much more expensive than pneumatic ones.

2. Obviously, there is no way of having a purely electrical suction cup gripper. The suction cup gripper is very useful on the Little Helper, since this gripper is capable of handling nearly all parts with planar surfaces. The suction cup could be maintained, but without the pneumatic system this would be required to be designed from scratch, where one solution could be to use a miniature vacuum pump. This problem is however not pursued further in this project.

3. The tool changer is based on a pneumatic system, so a new tool changing mechanism has to be implemented either by buying or designing a new one.

Apart from the changes in configuration to accommodate the new manipulator, some components are upgraded in the process as well. This is limited to:

- The vision system, where the camera and a fixed focal length and aperture lens replaces the current. The distance sensor is removed as well, since this is rarely used in the former configuration.

- The inverter, which is replaced by a different, more compact, model with similar specifications.

After this initial investigation of the configuration of the Little Helper Plus, a requirements specification can now be established.

5.1 Requirements for the Little Helper Plus

When an overview of the components needed in the configuration of the Little Helper Plus has been made, a basis for the further work of the reconfiguration can be established in the form of a requirements specification. This requirements specification will directly and indirectly determine the design of the new configuration.

Basically, the quantitative requirements established for the Little Helper, stated in [2], should be fulfilled in the new configuration as well. The hardware aspects of these requirements are:

- Maximum weight: $2kg$
- Maximum dimensions: $75mm \times 75mm \times 100mm$
- Possible payload of parts being transported $> 20kg$
- Battery time: $7hr$ (one normal work shift)

Apart from the quantitative demands, it is decided that the Little Helper Plus should have the same capabilities and functions as the Little Helper, i.e. there should be no loss of functionality. An exception to this is of course the case where a feature has been implemented on the Little Helper, but rarely or never used - this is true for e.g. the distance sensor, which is basically never used. The capabilities of the Little Helper will of course not be listed here, but where a function is removed it will be mentioned.

Tool changing

One of the most challenging aspects of the new configuration is the absence of a pneumatic system. Traditionally, robot tools are nearly always pneumatically driven, primarily from an economic point of view. The same is the case for tool changers, since the designers assume a pneumatic system is at hand for actuation of tools. As such, very few electrical tool changers exist, and none in a size that fits the mobile manipulator (all found electrical tool changers incorporate an electric motor on the tool changer, greatly increasing the weight and size). The effect of this is that an electrical solution for tool changing has to be designed.

Range

An investigation of the desired horizontal and vertical range has been made as a preliminary project to [2]. The conclusion of this investigation is that the tool center should be able to reach a point between $900mm$ and $1350mm$ above ground level, and the horizontal stretch from the platform should be between $200mm$ and $450mm$ from the platform.
The range of the manipulator relative to the platform is not considered to be a problem, since the working envelope of the KUKA LWR is greater and more versatile than the Adept s650. The range will however be considered during the rebuild.

6 Replaced components

This chapter will describe the major changes in components for the Little Helper Plus, compared to the Little Helper. The chapter will both describe redesigned solutions and components which is merely replaced with others.

6.1 Tool changing

As previously described, a non-pneumatic automatic solution for tool changing is not commercially available in the desired size, so a new solution has to be designed. Several concepts have been considered, before settling on a final design. The following section will primarily deal with the decision tree leading up to the final design, where a few key concepts will be presented along the way, to demonstrate the line of thought throughout the design, after which the final design is presented.
A quick scan of the market has revealed that there is practically no electrically actuated tool changers, and the few that exists are primarily for larger welding applications, such as the ATI Electric QC [4], making them useless for this application. Therefore, it is necessary to design a non-pneumatic tool changer. Furthermore, it seems that nearly all of the commercially available automatic tool changing systems operate on the same principle, including the Schunk SWS, that is currently used on the Little Helper. This principle will be described in the following.

Principle of the Schunk SWS

The Schunk SWS is a series of tool changers all functioning by the same principle, but available in a wide range of sizes, capable of handling payloads from $8kg$ to $455kg$. In this section, the SWS-011 will be described, since this size is the one currently attached to the Little Helper.
The principle of the Schunk SWS, as well as many other automatic tool changing systems, is a simple one, which most likely is why it is so widely used. The tool changing system is composed of two parts; an adapter and a head, where the head is attached to the robot tool flange, and an adapter is attached to each tool. The locking mechanism between the two parts is a piston being pneumatically driven downwards, pushing a

number of locking balls from the head into the adapter, fixing the adapter to the head. The locking mechanism is shown in Figure 6.1. To release the tool, pressure is simply applied to the other side of the piston, driving this upwards and allowing the locking balls to move into the head.

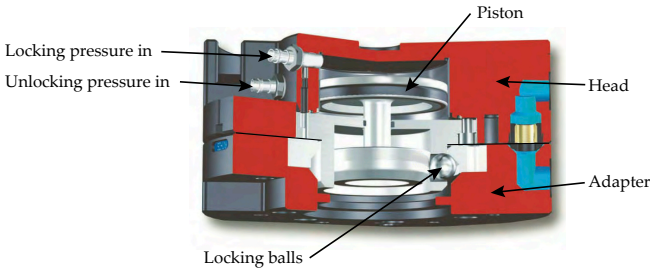
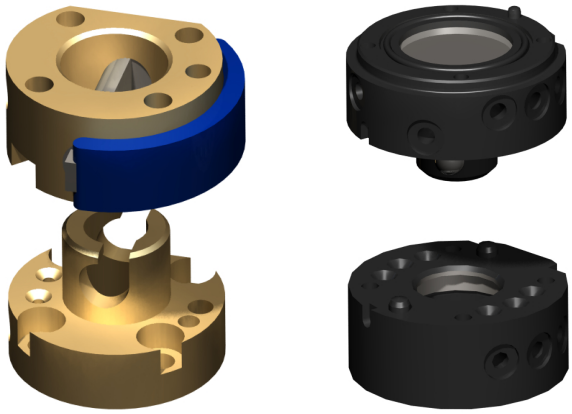


Figure 6.1: Schunk SWS quick change system

6.1.1 Determining basic principle

The automation equipment manufacturer Schunk has a wide range of tool changers available, and have been consulted on the matter of designing a non-pneumatic solution. The reason for this, is the notion that it would be beneficial to design a modification of a commercially available tool changing mechanism, rather than designing and manufacturing one from scratch. Ole Simonsen from Schunk [5] has presented some suggestions, where the most appealing principle is modifying either the Schunk HWS or SWS, both shown in Figure 6.2. The HWS system is a purely manual system, since an operator is required to release the blue arm (in Figure 6.2(a)), turn the pin to release the tool, and turn the pin again to fasten the new tool. It is, however, fairly easy to remove the blue arm and turn the pin by other means.



(a) Schunk HWS-40 (b) Schunk SWS-011

Figure 6.2: Two different Schunk tool changing mechanisms

The suggestions given by Ole Simonsen is considered further, and a map of the possible principles regarding these two tool changing systems is shown in Figure 6.3. The right-most level of this tree is the mechanism that performs the actual locking mechanism in the tool changing, and will be described later.
The pneumatic solution is shown here as well, since one possibility is to have a small pressurized air container on the platform, which is re-pressurized when the batteries on the platform is charging as well. However, this seemingly elegant method is quickly ruled out, for two reasons:

1. There is currently no qualified guess as to how much air is consumed during a normal shift, but it is considered to be too much to be contained in a pressure cylinder of a small size.

2. To refill a pressure cylinder with a sufficient amount of air, a high pressure is needed, which requires a high pressure pump at the charging station to obtain, which is an impractical solution.

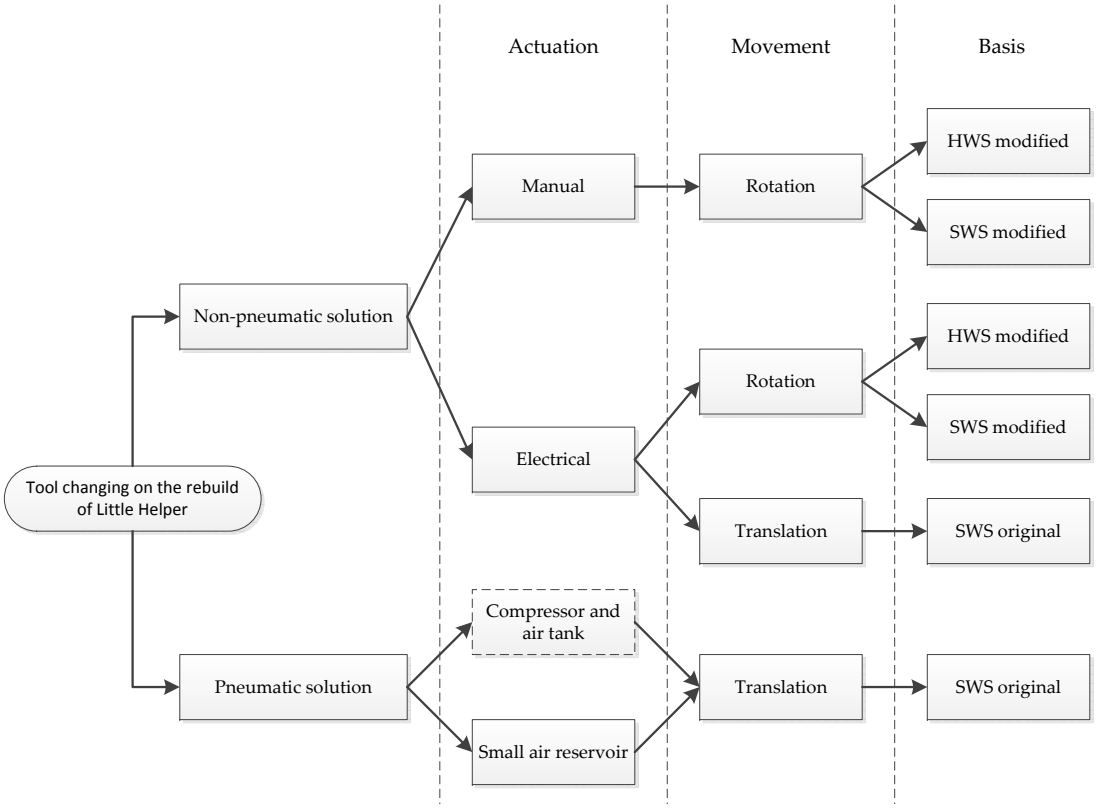
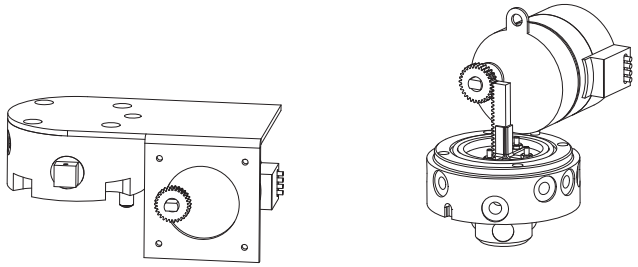


Figure 6.3: Multiple methods for creating a non-pneumatic tool changing system

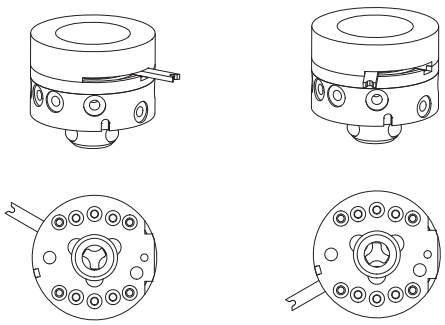
The basic design of each branch of the tree shown in Figure 6.3 has been created in CAD, to better visualize which solution seems the best. The electrically driven rotation of either the pin in the HWS or a modified piston in the SWS is ruled out at this point, due to both the size of an electrical motor with adequate torque, and the complexity of the solution, i.e. the number of moving parts. The CAD model of these concepts is shown in Figure 6.4.



(a) Modified HWS (b) Modified SWS

Figure 6.4: Tool changing mechanisms modified by adding an electric stepper motor to actuate the tool change

A single manual solution has been considered as well, and is shown in Figure 6.5. The term *manual* in this case requires the manipulator to move during the tool change, as opposed to having an operator change the tool. This solution is ruled out as well, also because of the complexity and number of moving parts, and the need to modify or redesign the piston.



(a) Locked (b) Open

Figure 6.5: Modified version of the Schunk SWS, where a rotation of a modified piston is used to actuate the locking balls

The chosen method of tool changing is therefore a linear, electrically driven actuation of an unmodified Schunk SWS-011. This solution has a number of benefits compared to the others, where the most prominent are:

- Simple solution; can be designed with few moving parts
- Can be made compact

- Does not depend on robot movement
- Original equipment is unmodified

Of course, the solution depends on the commercial availability of a small linear actuator, which delivers sufficient force to move the piston of the SWS-011 up and down. This is investigated in the following.

6.1.2 Choice of actuator

In order to choose a linear actuator for this application, two primary quantitative parameters must be known:

1. The required force to be delivered to the piston
2. The required stroke to move the piston between the locked and unlocked state

Apart from this, the qualitative demands of easy control of the actuator and a compact overall solution should be considered.
The required stroke is easy to determine, since this can be measured from the CAD model of the SWS-011. It is determined that a stroke of $7.5mm$ is required, so the stroke of the actuator should be $10mm$ or higher. This is both to allow for tolerances in the manufacturing of the fixture, and to avoid reducing the service life of the actuator, since this is decreased when driving the actuator to its limits.
Schunk specifies a minimum operating pressure of $4.5bar$ to actuate the SWS-011. Given the surface of the piston, that the locking pressure is operating on, the force moving the piston is calculated to be:

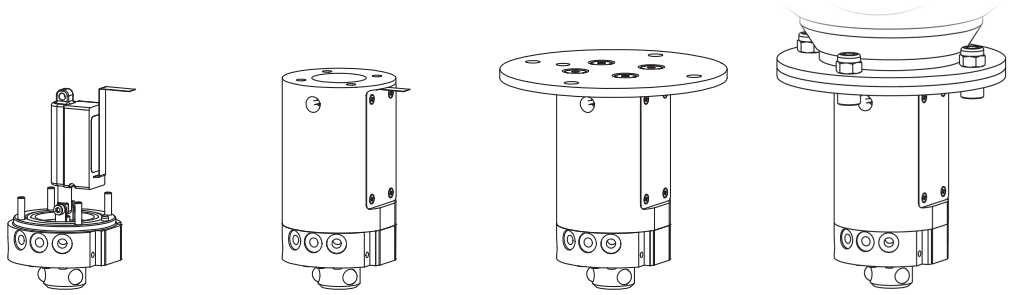
$$F = P \cdot A = 0.45MPa \cdot 531mm^2 = 238.95N \tag{6.1}$$

This result suggest that Schunk has added some overhead to the result of their calculation for required pressure, to ensure correct function of the tool changer. An experiment made by placing various weights on the piston of the SWS confirms this, as a weight of around $2{,}5kg$ ($\approx 25N$) is adequate for moving the piston satisfactory. On the basis of this, the force delivered by the actuator should be at least $30N$.
Miniature linear actuators are not that hard to come by, but generally these actuators are very small, with strokes of a few millimeters and very small tolerances, usually used in medical applications. Very few linear actuators are made with specifications resembling the quantitative demands, which are also compact in design and comes with a controller. Firgelli Automation manufactures the PQ12, which comes in different gear

ratios, where the highest gear ratio of 100:1 yields a maximum force of $35N$. This actuator has a stroke of $20mm$ and is only $48mm$ long in fully retracted state.
Firgelli also supplies a small actuator controller, designated LAC (Linear Actuator Controller). This controller can be programmed via USB, through Firgelli's own software for controlling actuators. Apart from this, Firgelli also supplies a DLL file, for creating an application in e.g. Visual Basic or C++ to control the actuator. This fact, and the fact that both the actuator and controller are rather small components, compared to their specifications, supports the qualitative demands of compact size and ease of use.

6.1.3 Design of the hardware

The hardware implementation of the actuator in the tool changing mechanism is designed, so the actuator is mounted directly on the piston, rather than having the actuator translated from the center of the piston. This is both to save space and make the design as simple as possible. The downside of this, however, is that the tool is mounted away from the tool flange of the robot, to some degree reducing the effective payload of the robot.
Firgelli supplies brackets and bolts for fastening the actuator, and there is a threaded hole in the center of the piston of the SWS, both of which are size M3. The actuator is mounted as shown in Figure 6.6(a).
A housing is designed to function as a common fixture for the Schunk SWS and the actuator. It is designed so the actuator is fixed in at least one direction, to maintain the direction of the piston during movement. Furthermore, a cover plate is mounted to protect the piston from dust and foreign objects, which could reduce the lifetime of this. The housing and cover plate are shown mounted in Figure 6.6(b).
An adapter plate has to be made in order to mount the mechanism on the robot tool flange. KUKA has specified the effective payload of the robot with the gripper positioned in various distances from the center of the robot flange, and this reveals that a displacement of the center of gravity of everything mounted on the tool flange (i.e. gripper, tool changer, handled products, etc.) of only $25mm$ reduces the effective payload to $5.5kg$. This means that the mechanism should be mounted directly on the tool flange, so the center of gravity is only displaced directly outwards from the flange, where the aforementioned effect is not as profound. This is however impractical, due to the physical layout of the robot flange, but is however solved as shown in Figure 6.6(c) and 6.6(d), with adapter plates on both the housing of the actuator and the robot flange.



(a) (b) (c) (d)

Figure 6.6: Mounting of the tool changing mechanism on the robot

6.1.4 Controlling the tool changer

As previously mentioned, Firgelli supplies a DLL file along with the LAC. The DLL file is containing the functions to communicate with the controller via USB, and these functions can easily be imported into applications developed in languages which support the use of DLLs, e.g. Visual Basic, C♯ or C++. The latter is chosen as the programming for the control of the tool change, since a great deal of the future software architecture will be programmed in this language.
Since the actuator and SWS is fixed to each other, the application is developed so the user only has to decide to either detach or attach a tool. Apart from this, it should also be possible to write a configuration file to the controller[1]. These three functions are hard-coded into the tool changing application, so the user will only have to specify which argument to pass when calling the program `ToolChange.exe`.
An in-depth description of the application written in C++ will not be presented here, but rather an overview of the basic principles and functions used, since the application is only calling already specified functions from the DLL file.
When the application is started, it first checks to see if exactly one argument is passed to it. If one argument is passed, the applications continues to establish a connection to the controller, regardless of the argument passed. Two connections have to be created, one to read data from the controller, and one to write to the controller. After the connection is established, the application checks which argument is passed, and proceeds to the appropriate action:

[1]Configurations are automatically saved on the controller, even when the power is cycled, but the possibility to quickly reconfigure the controller should be maintained just in case.

`-config`: Writes a hard-coded configuration to the controller. Though a number of parameters are available for configuration, some of them are irrelevant for this application, and are left out. The configured parameters are the precision, speed, extend and retract limits of the actuator, and that the configuration should be maintained in the memory of the controller (the latter not per se being a parameter, but rather a necessary step in configuring the controller).

`-attach`: Simply writes the position of the actuator in the locked state to the controller, which in turn applies the correct voltage to the actuator to move it to this position. This position is found by trial-and-error after the housing has been made, and the mechanism assembled.

`-detach`: Does the same as the `-attach` command, only for another position.

In order to write the parameters and position to the controller, all values have to be converted to the appropriate format, in this case a very proprietary format, consisting of a character array of size 3, containing a value representing the parameter to set, the value to send and the bit-swapped[2] version of the value. This conversion and the DLL-imported function to write to the controller and read the answer is incorporated in its own C++ function, since this has to be done each time data is sent to the controller. After the desired function is carried out, the application closes the connection to the controller and exits.
A video demonstrating the tool change can be found on the enclosed CD, in `Media\Video\ToolChange.mov`. This is merely a demonstration of the tool changing, and not a real world implementation of it. It is obvious that the actuator takes some time in changing tool, primarily with attaching the tool, since this operation requires the most force. However, tool changing does not occur that often on the mobile manipulator, and as such it is concluded that a slow tool change has practically no effect on the overall efficiency of the mobile manipulator.

6.2 Switching board

When machine vision is used as a method of performing quality control, lighting is everything. In this application, however, where vision is primarily used for pick-and-place operations, lighting does not have to be controlled as strictly. On the Little Helper, a light controller was incorporated to turn on/off the lights for the vision system and

[2]Where the higher order bits are swapped with the lower order bits, i.e. blocks of 8 bits are swapped

the warning lights[3]. The light controller also adds the possibility to strobe the lights. However, since the development of Little Helper, it has become apparent that the added functionality of the light controller is seldom used, since the controller is only used to turn on or off the lights.
In the new configuration of the mobile manipulator, it is determined that this is the only function required for both the lighting for the vision system and the warning lights. Instead of a dedicated light controller, a method of turning any device on/off through e.g. a USB or RS-232 interface is desired. After some investigation, the small controller Mini-BEE from PC Control Ltd. has been bought. The Mini-BEE offers 14 switching outputs in two circuits, which in this case most likely will be one 12VDC circuit and one 24VDC. The drivers used by the Mini-BEE are Darlington DS2003 Drivers, and the data sheet for these drivers state that the maximum capabilities of the drivers are 50VDC at $350mA$ [6]. It is however possible to achieve higher current throughputs, by connecting devices in parallel, and making sure that all switches are turned on/off at the same time. The Darlington drivers consist of 7 NPN transistor pairs each, the NPN meaning that the switching occurs on the common/ground side of the circuit [6], [7]. The connection to the Mini-BEE is simple, since all devices in one circuit connect to a common ground, and each device is connected to the positive side of the power supply and to one of the ports on the Mini-BEE.
PC Control also supplies a DLL file along with their product, and a C++ application has been made to control these outputs. The Mini-BEE controller needs to receive a hexadecimal representation of 14 bits, where each bit represents the state of a channel on the controller, either open (1) or closed (0), which turns out to be quite easy to implement. The state of each switch is not saved when power is cycled, and the Mini-BEE has no function to read which outputs are open and which are closed. This function is implemented as having a simple text file containing the 14 bits, which is read at the start of the program, after which the user specified changes are written both back to the file and to the controller.
Even though the controller will primarily control the lighting for the vision system and the warning lights, other uses could be implemented later, due to the number of channels on the board, e.g. turning off the controller for the tool changer or the camera when these are not needed. One thing to bear in mind with this, however, is the switching transients when turning on an inductive load (e.g. an electric motor or relay). These spikes in voltage has been accommodated in the Mini-BEE by the use of suppressor diodes added to two of the channels. To make use of these, one simply has to connect

[3]Tower light signalling when the platform or manipulator is moving.

the power source positive (+) directly to these channels, and the spikes in voltage are suppressed. A representation of the Mini-BEE is shown in Figure 6.7, with the warning lights and vision lighting connected. The dotted lines represent the transient suppression connection, and is a direct connection from positive to the controller.
The warning lights attached to the controller deserve some extra explanation. The tower light is a Schneider Electric XVC 4B3K, with three lights in colors red, orange and green. The data sheet of the tower light specifies that the common power input to the light (RD wire) should be connected to common positive for switching with NPN transistors, which corresponds with the data sheet of the Mini-BEE and NPN transistors in general. Each of the LEDs in the tower light has its own connection to ground; orange (OR) wire for red light, yellow (YE) wire for orange and green (GR) wire for green.

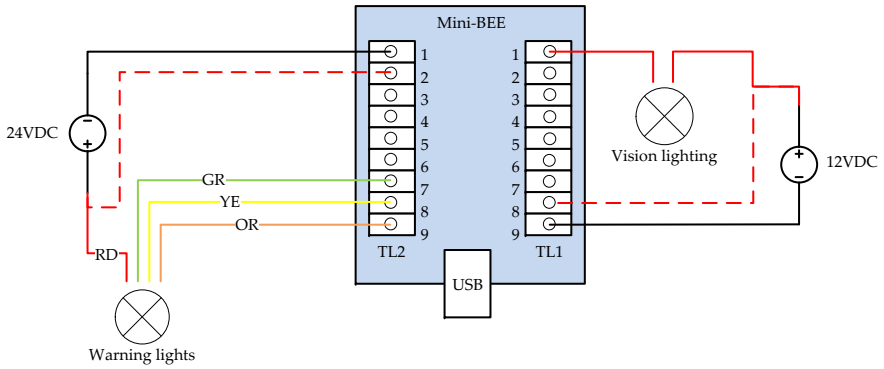


Figure 6.7: Wiring of the lighting to the Mini-BEE.

6.3 Vision system

After the development of Little Helper, it has been used in a number of scenarios, both in the laboratory and in an industrial environment. The use cases has revealed a number of improvements, which is sought implemented in the new configuration. One of the main improvements is regarding the vision system. Originally, this was designed to be very flexible, with an adjustable lens, a distance sensor for calibration and a light controller enabling advanced control of the bar lights. The light controller has been removed in the new configuration, as mentioned in the previous section. The adjustable lens and distance sensor, however, has not. Both of these components add a great deal of flexibility to the overall system, since the vision system can be adjusted to return adequate images of the inspection, given less than perfect conditions. However, in practice these functions are not fully utilized, since *(a)* the distance sensor is rarely, almost never, used

and *(b)* the ability to adjust the lens parameters is only used to make sure the lens is set to the same settings (aperture and focal length) each time an image is acquired.
In the new configuration, the vision system is redesigned, so these functions are not maintained. Instead, a fixed lens is used and the distance sensor is removed. Furthermore, a new, smaller camera is used, since more compact cameras have become available since the development of the Little Helper. The choices of new components for the vision system are however out of the scope of this project, since it requires a greater analysis and experience within the field of machine vision.

7 Configuration of the Little Helper Plus

The following chapter will present the design of the Little Helper Plus, from a hardware point of view, since this project is not concerning the software and control of the solution directly. The chapter is structured in the same way as the work flow, since the order the placement of the various components is described in follows the order these components were actually placed in.

7.1 Main housing

Since the major changes compared to the Little Helper are made in the main housing, this is presented first. The function of the main housing is to fix and protect hardware, and provide a platform for handled parts and the LWR itself. As such, a number of brackets should be designed to mount the components, a chassis to fix everything in place, and cover plates to protect the hardware components. The most challenging part of this task however, is to make room for all components, while still maintaining the mounting requirements for each component.

7.1.1 Component placement

In order to make everything fit on the platform, and thoroughly design a system like this one, the essential task is simple: Where does everything fit? One way of answering this question is modeling components in CAD software, and arranging them in a virtual environment, and since CAD models of nearly all components is already at hand or easily obtainable, this is what has been done.
In order to place all components on the platform, however, one does not have completely free rein, since some components need cooling, some needs to be mounted in level etc. The following aspects for this case needs to be taken into consideration:

Controller: Needs to be mounted horizontally, and needs cooling in the form of air intake in the bottom, and exhaust on the left hand side. Also needs room for cable connections on the front, so some free space should be available here, since the cables protrude some distance from the controller.

Inverter: Has cable connections on the back and front, and air intake/exhaust on the two sides.

Electrical system: Fuses, terminal strips etc. need some cooling, but not much. Easy access has to be implemented, to enhance the possibility to make modifications or add new components, and change blown fuses.

Being by far the largest component, the controller is placed on the platform first. This can almost only be placed in one way, due to cooling exhaust on the left hand side, which has to be placed in free air. Naturally, the placement of the controller almost dictates the placement of the remaining components, which makes the task of placing these components somewhat easier. The inverter seems to present the biggest problem, since this is a tall component, compared to its footprint, and needs some space on all sides for both cable connections and cooling. It seems there is only one way of placing this, given that the air intake is to be placed in free air, and even this placement necessitates leading the exhaust air somewhere else, as it would else lead the heated air onto the side of the controller.
After the removal of the pneumatic system, the two larger components have now been placed, as shown in Figure 7.1. The smaller components are to be placed on a dedicated shelf attached to the chassis above the inverter. The electrical system will be mounted on standard DIN 46277-3 "top hat" rails, as it is the case on the Little Helper, since this is the most widely used standard for electrical installations. These DIN-rails are also to be attached to the chassis. Since the remaining components are all going to be attached to the chassis of the main housing, this is designed next.

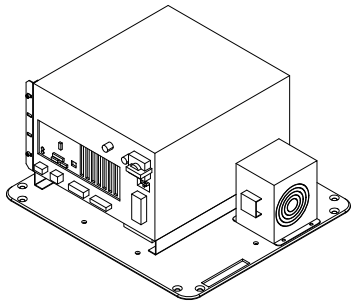


Figure 7.1: The placement of the controller and inverter on the top plate of the platform

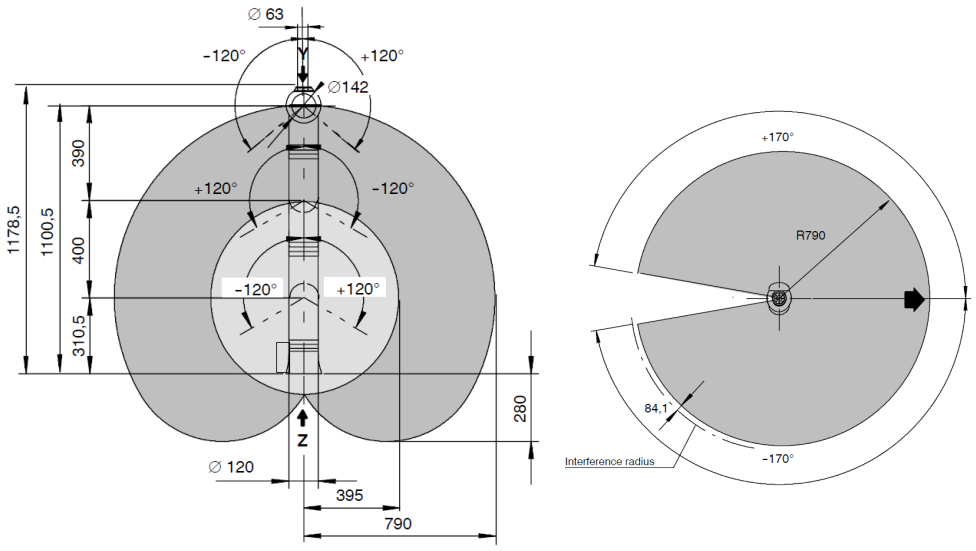
7.1.2 Chassis

Since the chassis is the one part holding everything together, a great deal of consideration has be taken when designing this. Not only does the chassis serve as a common fixture for most components, but it must also withstand the weight of the robot mounted on top, as well as the moment around the base of the robot, when this is handling parts at the maximum of its reach.
The chassis on the Little Helper is made from aluminum profiles that are welded together, which yields a stiff construction compared to a chassis assembled by screws. However, the profiles used have a hollow square cross section, which has presented some problems regarding the fastening of the manipulator and other components requiring very strong mechanical connections. Furthermore, the attachment of new components is complicated by the fact that new threaded holes need to be drilled in the chassis to fasten the components.
In order to enhance the chassis on Little Helper Plus, the choice has been made to use one of the many standardized aluminum profile systems developed explicitly for constructing frames and chassis, in this case the HepcoMotion MCS system [8]. This system, like most others of its kind, consists of extruded aluminum profiles in various sizes and cross sections, along with methods of joining them together and attaching other hardware. The chosen profiles are similar to many others, with a slot on each side of the profile, used to attach hardware or join profiles. The used profiles are shown in Figure 7.2.

Figure 7.2: An example of the aluminum profiles and accessories used for the chassis [8]

The chassis is built around the already added components, i.e. the controller and inverter. For the part of the chassis simply fixing the controller in place, and fastening the electrical system and smaller components, the 30x30mm profiles are used. For the mounting of the robot, however, the 30x60mm profiles are to be used, to increase the rigidity and strength of the chassis here. HepcoMotion has several methods available

for joining profiles together, and a comparison of these methods are given in the manual for the MCS system [8]. This comparison reveals that the simple solution of using M8 bolts for the joining of profiles yields the highest overall stiffness of the chassis, at the cost of flexibility. This method is chosen, since a high rigidity is desired, and the need to redesign the chassis is very small. Given the weight and payload of the LWR, and the instructions for appropriate mounting of it, it is estimated that the stiffness of this chassis is adequate when using the 30x60mm profiles.
One important structural aspect the chassis has a direct influence on, is where the working envelope of the robot is placed, and by that the reach as well. As discussed in Section 5.1, the reach has to be the same or better than the Little Helper. This is not the only aspect controlling the mounting height of the robot, however. Due to the kinematics of the LWR, the manipulator has a spherical volume located around joint 2 that is unreachable. This area has a radius of 400mm, the same as the length of link 2 on the manipulator, and as such poses some problems regarding the picking and placing of parts on the platform table. The working envelope of the LWR is shown in Figure 7.3.



(a) Side view (b) Top view

Figure 7.3: Working envelope of the KUKA LWR [9]

In order for the manipulator to be able to pick and place parts on the platform, it is obvious from the working envelope that the manipulator has to be mounted at a higher level than the platform. However, the manipulator is not to be mounted as high as to

not fulfill the quantitative demands regarding reach listed in Section 5.1.
A mounting height of the robot of 90mm compared to the platform table is considered adequate, since it both fulfills the need for range, and allows pick and place operations to be performed on the platform. The working envelope of the mounted manipulator is shown in Figure 7.4. Measurements in CAD shows that the demands for reach specified in Section 5.1 have been satisfied, the maximum range especially greatly exceeding the required.

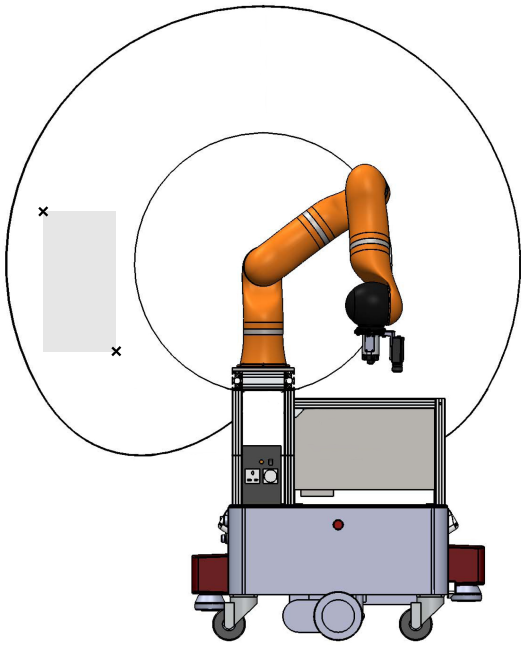


Figure 7.4: Working envelope of the LWR mounted on the chassis, the marks and gray box showing the requirement for reach

Component brackets

The added height of the robot mounting yields the benefit of more space in the compartment immediately beneath the manipulator. This space is used for a shelf containing the smaller components, e.g. the Firelli LAC and the Mini-BEE. If needed, there is spare room for adding an extra shelf for components, and the shelves are cheap to manufacture and easy to attach to the chassis, due to the slots in the profiles.
Immediately next to the component shelves, the electrical system is mounted. Some room is needed for this, due to the many power and signal connections needed, and as such the DIN rails are mounted directly on the 30x60mm profiles. The various connec-

tions are discussed in a later section.
In order to mount the manipulator on the chassis, an adapter plate has to be manufactured, to provide an interface between the chassis and the robot, since it is not possible to attach the robot directly to the chassis. KUKA manufactures an adapter plate for use with the LWR, but this does not fit the chassis, and is instead used as a point of reference regarding the plate thickness and size of fasteners, and a new adapter plate is designed.
As is the case on the Little Helper, a tool magazine is needed so the manipulator is able to carry 2-3 tools with it. Little room is left for this, but it is estimated that there is room for this in front of the controller, the tools suspended from the top and in front of the part of the controller where no connections are attached. If necessary, the tool magazine can instead be mounted on a piece of the aluminum profiles used for the chassis, so the tools are not suspended as low. Regarding the construction of the tool magazine, the exact same principle is used as on the Little Helper, where a number of pivots are used to position the tools and hold them in place.
The mounting of the manipulator, component shelf, electrical system and tool magazine mounted is shown in Figure 7.5.

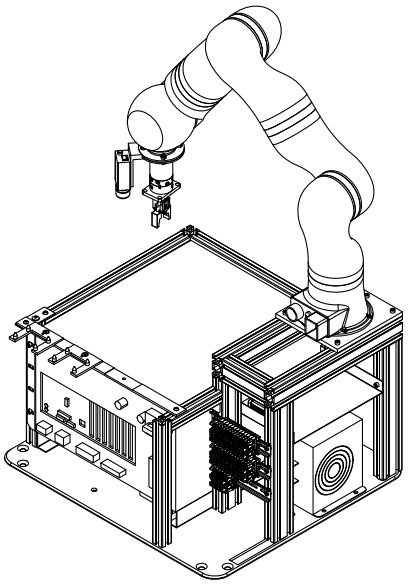
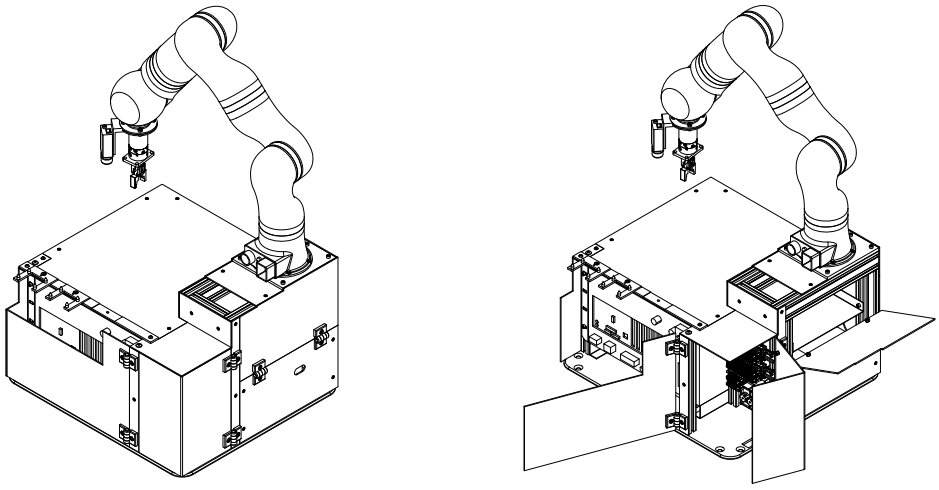


Figure 7.5: The rest of the components are mounted on the platform

7.1.3 Cover plates

In designing the cover plates, only a few aspects have to be taken into account, i.e. which material should be used, and where there should be venting holes. However, when looking at the Little Helper, a few improvements spring to mind, which is implemented in the new design.
The cover plates of the Little Helper are made from $2mm$ aluminum plates, making them rather heavy. Furthermore, when troubleshooting a problem with the Little Helper, it is impossible to see control and warning lights on the various hardware in the main housing without removing the cover plates. A cover plate also has to be removed to turn on the inverter and controller for the Adept manipulator, making the Little Helper ready for operation, which is quite inconvenient.
In designing the cover plates for this configuration, these things are taken into consideration. As such, the cover plates are manufactured in transparent acrylic, so the hardware is visible at all times, and the weight is kept at a low level. The cover plates located on the back and left hand side of the controller are easy to design, since they only need fastening to the chassis and venting holes where the controller has cooling intake and exhaust. The other cover plates are harder to design, though none of them present significant difficulties, apart from the fact that the cover plates has to be split up into smaller, rectangular parts to ease the manufacturing process.
In order to provide easy access to the controller, electrical system and component shelves, it is decided to design the cover plates in these areas as access doors, using hinges available from HepcoMotion, that attach directly to the chassis. This solution gives the user an easy method for gaining access to the controller and various hardware, should the need arise to maintain this.
Though as many cover plates as possible are designed in acrylic plates, it is hard to bend these plates, and as such three plates have to be manufactured in aluminum plates. This is limited to the access to the electrical system, a bracket which is also holding the table of the platform, and said table on the platform. The two latter plates should be able to withstand the weight of the parts carried by the robot during operation, and as such should be stronger than acrylic plates, which is why they are designed in $3mm$ aluminum plates. A rubber mat is glued to the top of the table, to avoid products moving during transport.
The attached cover plates are shown in Figure 7.6, in both the closed and open state.



(a) Closed covers

(b) Open covers

Figure 7.6: Cover plates attached to the chassis

7.2 Tooling

Though already presented to some degree, some further explanation of the tooling should be offered. The previous Section 6.1 mostly deals with the design of the tool changer and housing, though the interface to the LWR tool flange is mentioned as well. In this section, the combined tooling system is presented.

As mentioned in Section 6.1, the displacement of the gripper center axis from the normal of the tool flange results in greatly reduced effective payload of the robot, as shown in Figure 7.7. Due to this fact it is decided to design the tool with the gripper centerline collinear to the normal of the tool flange of the robot, so the tool is only displaced $200mm$ in the z-axis of Figure 7.7, giving a payload of approximately $3kg$, and thus satisfying the quantitative demand regarding payload. It is however quite inconvenient with regard to designing the interface between tool and robot when this method is chosen. The chosen method requires an adapter plate on both the tool and the tool flange, which subsequently should be attached to each other. This method is however considered adequate, given satisfactory tolerances in the manufacturing process of the interface plates.

In order to attach the camera to the tool, this can be done quite easily, utilizing the design of the adapter plates. By making a curved cut on the bracket for the camera which fits the adapter plates, a single bolt is needed to fix the camera in place. The camera is positioned so there is no rotation of the camera axis compared to the common

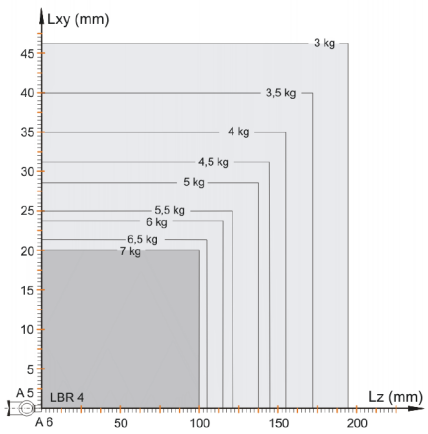


Figure 7.7: The effective payload of the manipulator when the tool center is displaced from the center of the tool flange on the manipulator [9]

tool center and robot flange axis, but merely a displacement, which yields a simpler translation between the camera center and the tool center, and does not require the robot to turn the tool to acquire an image orthogonal to the inspected part. The tooling system is shown in Figure 7.8, although with a different camera, since the camera to be used has yet to be chosen.

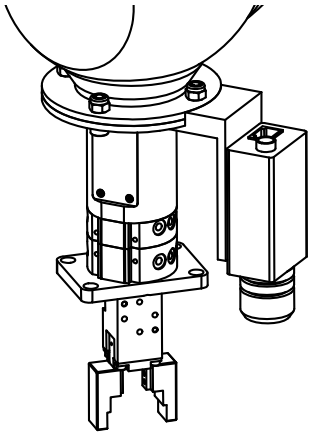


Figure 7.8: The mounting of the camera to the tool flange

The design of the tool magazine on the Little Helper had both the function of fixing the tools in place, and also decreased the need for precision when placing the tools, since the small pivots on the tool magazine made sure the tools where positioned in the same way every time. This design is maintained in the new configuration, and as such there

is no change except the method for fixing the magazine to the platform. As previously mentioned, the tool magazine is placed in front of the controller, at a sufficient height as to not interfere with the cable connections to and from this. As such, a rail needs to be constructed to fix each tool holder in place. An added advantage of the placement of this rail is that it can be utilized to fix the table on the platform in place as well, so this plate is fixed along all four sides. The tool magazine is shown in Figure 7.9.

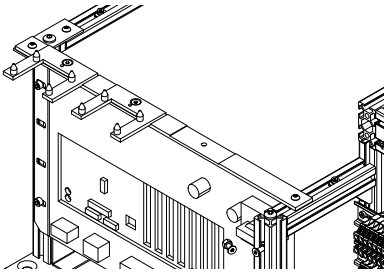


Figure 7.9: The tool magazine mounted on the chassis

7.3 Manufacturing of parts

The design presented in the previous sections requires some time and work to manufacture, since most of it are custom parts. In the following table, an overview of the parts to be manufactures is presented, along with an estimated time frame for the manufacturing. This is mainly done so the custom parts can be manufactured before the manipulator and platform arrives, so the new configuration will be up and running in the least amount of time. The operations and estimated time usage is shown in Table 7.1.

Part	Material	Operations	Time usage
Chassis	Aluminum profiles	Cutting, drilling, assembly	20 hours
Cover plates	Acrylic plates	Drilling, cutting	10 hours
LWR mounting plate	Aluminum	Cutting, drilling	5 hours
Component shelves	Aluminum plates	Cutting, drilling, bending	4 hours
Tool magazine	Aluminum	Cutting, milling, drilling	8 hours
Tool flanges	Aluminum plates	Cutting, drilling	4 hours
Camera bracket	Aluminum	Milling, drilling, assembly	6 hours
Actuator housing	Aluminum	Milling, drilling, assembly	8 hours
Total			**65 hours**

Table 7.1: Time usage and operations needed for the manufacturing of custom components

7.4 Power and signal connections

Figure 7.10 shows the power connections to the various components on the platform. Since the components operate at three different voltages, it is advisable to separate the terminal strip into three different circuits, which correspond fine with the physical layout of the terminal strips, shown in Figure 7.5. In the figure showing the connections, the warning lights and vision lights are both connected directly to power, but in reality these are connected through the Mini-BEE, as shown in Figure 6.7. This has merely been done to simplify the figure showing the connections.
Each of the circuits should be fused to protect the components, the most critical fuse being the one on the positive side of the platform batteries, since a short circuit of the combined $164Ah$ of the batteries could result in damage to the batteries, and even explosion and subsequential damage to personnel and equipment. An electrician has been consulted on the fusing of the electrical system, and recommends a $16A$ fuse for the $24V$DC circuit and $10A$ for both the $230V$AC and $12V$DC circuits.
Since the $230V$AC circuit can not be directly grounded, the grounding on the inverter is used. This grounding is internal and merely connects to the chassis of the inverter. This in turn is connected to the chassis of the platform.
The signal connections for the various components are all leading to the onboard computer controlling the platform, either through USB (the LAC and Mini-BEE) or Ethernet (the KUKA controller and the camera).

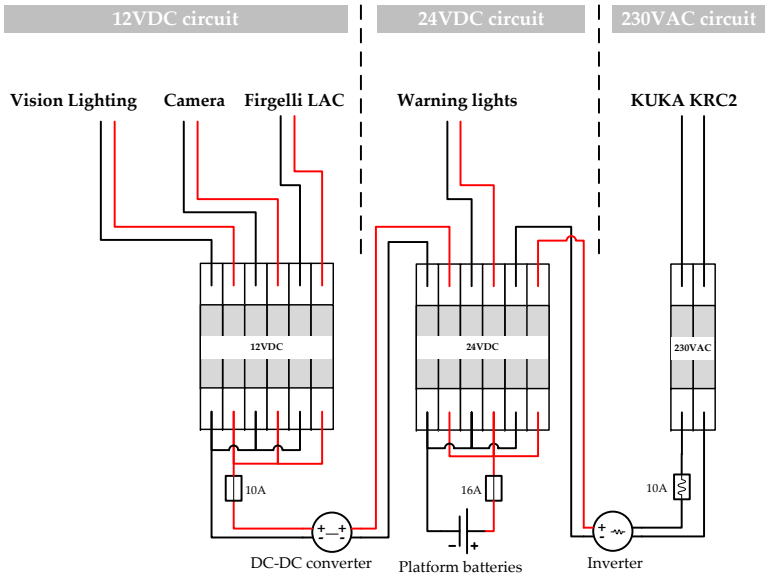


Figure 7.10: Electrical wiring of the various components

7.5 Summary

This concludes the design of the Little Helper Plus. The quantitative demands of the Little Helper are satisfied, and in some cases even surpassed, e.g. for the reach of the robot. However, the design has room for improvements, and improvements might be added once the actual build of the manipulator is begun, since potential errors in the design usually are spotted during this phase. However, the task of designing the Little Helper Plus is complete, and the work on investigating the capabilities of the KUKA LWR can begin.

Part II

Capabilities of the KUKA LWR

This part of the project will deal with investigating the capabilities of the KUKA LWR compared to traditional robots. The majority of these tasks have an immediate use in the mobile manipulator, which is why the following will serve as a preliminary investigation of the added features of the LWR.
A major part of the following is a demonstration of the FRI, since this will most likely be used to sending commands to the robot from the onboard computer handling the execution of tasks. Several other cases, which are useful for the flexibility aspect of the Little Helper, will be described as well.
Before the work on the tasks can begin, an initial setup of the LWR has been made, since the LWR is not mounted on the mobile platform at this time. This setup is described in Appendix C. Furthermore, the controller strategies of the LWR are described, since these somewhat differ from the controller of traditional robots.

8 Control of the LWR

Since the added functionality of the KUKA LWR makes the robot quite different than regular position-controlled robots, the control of the robot is different as well. The following chapter aims to give the reader a basic understanding of the control structure and added features of the LWR, in order to enhance the understanding of the following chapters.

8.1 Control strategies

Where traditional robots only offer a single controller strategy, that handles the position of the end-effector with respect to the robot base, the LWR offers several, which make use of the torque sensors. The following aims to describe the function of these, and as such does not describe the kinematics and control of each control strategy. Each control strategy has its own number, which is used internally in the KUKA Robot Language (KRL).

Position control (no. 10)

Being the usual controller for robots, this controller enables exact positioning of the robot, both in joint angles and cartesian coordinates. This controller is no different than the controller used for regular 6-axis KUKA robots, since the 7^{th} axis of the LWR[1] is fixed during motion, except when explicitly programmed otherwise.

Cartesian impedance (no. 20)

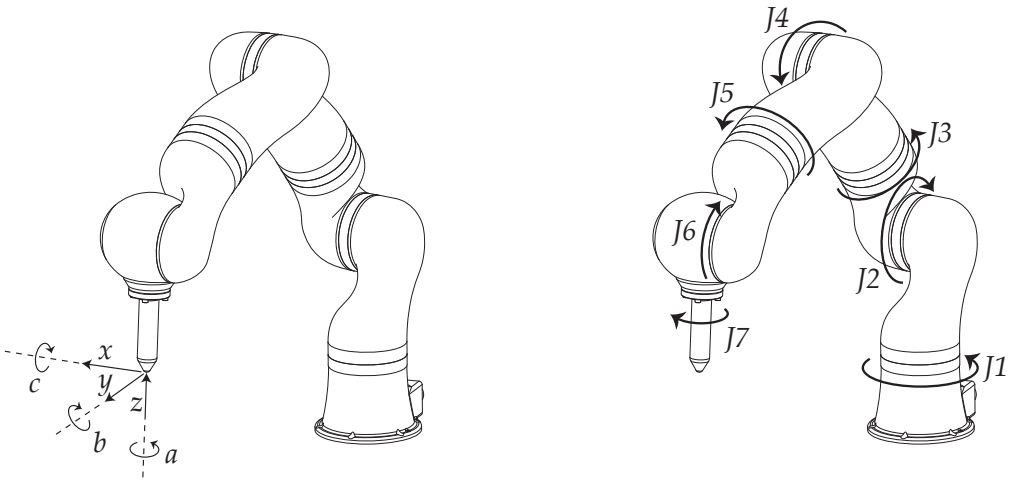
The cartesian version of the impedance controller, this controller enables the robot to function as a virtual spring-damper system in cartesian coordinates, this being either the world or tool coordinate system. The user can specify spring stiffness and damping parameters in each direction and rotation, as well as maximum deviation of the TCP and the maximum force acting on it. One other option in this control mode is to apply a specified force in a particular direction, either a constant force or a sinusoidal force, both in multiple directions. This control mode could be used where e.g. the robot is

[1] Chosen by KUKA to be J3, most likely to yield the most resemblance to traditional robots

required to fit two parts together, and has been used to place complex planar parts in holes [10]. The virtual springs and dampers are shown in Figure 8.1(a).

Joint impedance (no. 30)

The other version of the impedance controller, which instead acts on a joint-level, where a virtual spring and damper is inserted in each joint, instead of in the cartesian directions on the TCP. In this controller it is also possible to set stiffness and damping, along with maximum deviation and torque in each of the 7 joints. This controller is shown in Figure 8.1(b).



(a) Cartesian impedance control

(b) Joint impedance control

Figure 8.1: The virtual spring-damper systems utilized in controller modes 20 and 30, represented here by simple arrows.

Gravity compensation (no. 101)

Definitely being one of the more special control modes, this controller is only reacting on the input of the torque in each joint. The robot maintains its position in this mode, until an external torque is acting on one of the joints. In this case the robot yields, and the robot can be manually guided, effectively making this controller a convenient method for jogging the robot. One important precondition of the correct function of this controller is that the load of the tool must be entered correctly, so the controller can accurately determine the external torques, and not torques from the weight of the tool. This controller can obviously not be used in execution of programs, however, but is mainly for moving the robot by hand.

8.2 Controlling through the $STIFFNESS structure

The active controller and controller parameters are all handled in a single structure[2] in KRL, the system variable $STIFFNESS. This variable contains all data for the controllers, though some are not used depending on which controller is currently active. The active controller is set with the first parameter, $STIFFNESS.STRATEGY, and is an integer specifying the controller to be used. The rest of the structure are the settings for each controller, where only the values relevant for the active strategy needs to be specified.
There are two ways to change the values of $STIFFNESS. One way is to write directly to the system variable, either by writing the whole structure in one, or writing specific parameters, e.g. $STIFFNESS.CPSTIFFNESS for the cartesian stiffness array, where the latter is recommended in order to write more readable code. If this method is used, the assignment of parameters must be written to the system variable by setting $STIFFNESS.COMMIT = TRUE, which will write these values to the controller. Another way to apply parameters, which is recommended by KUKA, is to create a temporary variable with the same structure as $STIFFNESS, and assign values to this temporary variable. When the data is to be written, one simply overwrites the $STIFFNESS system variable with the temporary user variable - in this case there is no need to set $STIFFNESS.COMMIT = TRUE, since the values are stored directly in the system variable.

8.3 Built-in LWR functions in KRL

Two functions has been incorporated in KRL for LWR-specific use, the TRIGBYCONTACT and DESIREDFORCE functions. The following is merely an overview of these functions, and a possible incorporation hereof. TRIGBYCONTACT is used for fast switching between controllers, and this is the only use for this. The function is called with a parameter specifying the direction and magnitude of the measured force to cause the switch between controllers. This function could be used to e.g. move towards a surface in position control to ensure adequate precision, and switch to cartesian stiffness mode when there is contact between the surface and tool, in order to avoid damage to either.
DESIREDFORCE can be used in cartesian impedance mode to specify one or several forces or torques the TCP should apply to its surroundings, this being either physical objects or the virtual spring in the system. It is possible to apply force in all three cartesian

[2] A structure is a variable with custom multiple data types, e.g. three integers and one boolean, stored in one variable

directions and rotations, where the force could be both constant and sinusoidal, where amplitude, duration, rise time and fall time is customizable for each. In the case of a sinusoidal force applied in two directions, the resulting motion of the robot can be described via a Lissajous curve. This feature could be used to ensure prevent jamming of parts or to ensure a correct mounting pressure in joining processes.
This concludes the description of the control of the LWR, which should form the basis for understanding the following chapters.

9 Programming the LWR by demonstration

One of the apparent benefits of the LWR is the ability to move the robot by hand when online programming new tasks, since the time consuming task of this is usually using the keys on the teach pendant to jog the robot. Furthermore, it feels more natural to grab the robot and move it by hand, than using the teach pendant. One aspect of this is however if an adequate precision can be acquired using the available methods for moving the robot by hand. As such, it is to be investigated which method is faster, and if the precision of the motions can be programmed adequately by moving the robot by hand, compared to traditional jogging.
In order to test this, three simple tasks are carried out. All of these tasks are pick and place operations, because of the need for precision and easy programming:

Task 1 is merely picking up a pin from a hole, rotating the pin and pointing the tip to a specific point, and inserting the pin in another hole adjacent to the first.

Task 2 is picking up a hexagonal part, moving this to a position past an obstacle, and performing a slight variation of Task 1 afterwards, with greater distance between the two holes.

Task 3 is a simple stacking task, and as such begins with placing a pin in a hole, after which a hexagonal part is picked up and placed on top of it, and a round part is finally placed on top of this.

The three tasks are increasing in difficulty - task 1 is simple and does not require great precision, task 2 requires additional points to avoid the obstacle, and task 3 finally requires precision in order to stack the three parts without tipping them over. A full demonstration of the programmed tasks can be seen on video on the enclosed CD in `Media\Video\PbD.wmv`.
In order to program these tasks using both methods, first a set of assumptions needs to be established. In order to compare the two methods, it is chosen that all programming will be carried out on the teach pendant of the robot, using the regular KUKA interface. This interface provides easy methods for inserting motion commands, which are the main operations in these programs. Furthermore, only the functions for moving the robot

by hand, which are available through the soft keys on the teach pendant are utilized; these three functions are Gravity Compensation mode, free translation of the tool (no rotation), and free translation and rotation around the tool z axis.
Even though one might think it is easier to program the robot, it is quickly obvious that this method present its problems regarding the precision. Especially when inserting the pin in the hole, the rotation of the tool needs to be very precise, something that is hard to accomplish when moving the robot by hand, and often the angle needs to be corrected in each point by monitoring the values of the robot position. However, when the jog keys are used, the robot motion is very slow, which is less desirable.
The tasks are programmed once each using the two different methods, and the time usage for the programming is noted. The time used to program the various tasks are shown in Table 9.1.

Task	**Method**	**Time (mm:ss)**
1	Hand	09:12
	Jog	08:21
2	Hand	14:09
	Jog	10:35
3	Hand	14:17
	Jog	10:33

Table 9.1: Time usage for programming the separate tasks by different methods.

It is obvious from the time it takes to program the robot by the two different methods, that the regular method of jogging the robot using the jog keys on the teach pendant are faster than moving the robot by hand. A quantitative estimate of the precision is not given, but it is apparent that it is easier to ensure greater precision using the jog keys, since motion then is limited to one axis or rotation, and can be controlled at very low speeds. This is also apparent from the video showing the tasks, since the robot motion programmed by using the jog keys appears more precise than the other method.
One aspect of the above is the choice of using the KUKA interface for programming the robot. This interface has not been developed explicitly for the LWR, but is a standard interface for all variants of the KUKA KRC2 controller. Had the interface been developed for the LWR, methods for jogging would most likely be implemented in a better way. A way to do this could be to enable jog strategies comparable to those of the 6-axis mouse on the teach pendant, e.g. having one dominant axis, where the jog motion takes place. Alternatively, this could be implemented using the Fast Research Interface; these improvements will however not be discussed further in this section.

10

Workstation calibration using force sensing

One of the more time-consuming, though necessary, tasks of the Little Helper is the calibration of the transformation between the TCP and the actual workstation. The calibration is necessary, since the precision of the platform is not very good, with regards to both position and orientation.
The torque sensors in the LWR could be used for this calibration instead, by programming the robot to feel where specific edges of the workstation are, or by having a dedicated calibration tool, that fits in a fixture on the workstation. This exercise aims to investigate the possibility for using the first of these techniques, and give an estimation of the accuracy of this calibration.

Purpose: To investigate the possibility of accurately calibrating the position and orientation of a workstation using the torque sensors in the LWR, i.e. without a vision system.

Objective: To calibrate a workstation coordinate system from a set of assumptions made on the location of certain edges, and measure the accuracy of this calibration.

Since the accuracy of the platform is known ($\pm 10mm$ and $\pm 5^\circ$ [2]), this accuracy forms the basis of the assumptions for the exercise.

Assumptions

Since the calibration is only relevant for the LWR mounted on the mobile platform, some obvious assumptions can be made. The first is that the surface of the workstation is parallel to the surface of the platform, since the platform does not rotate around any other axis than vertical. This means that the rotation of the workstation relative to the *ROBROOT* coordinate system[1] $\{R\}$ of the robot is only around the z axis (vertical). The second assumption is that the calibration takes place near a corner on the workstation, and as such the angle between the three planes forming the corner is 90°. This is a necessary assumption to make, in order to have some initial knowledge of the workstation. The final obvious assumptions is the choice of tool. Which tool is used of

[1] A fixed coordinate system with origin in the base of the robot - common for all KUKA robots

course has to be known, since a transformation between the TCP and the workstation needs to be made. For this exercise, a pin fixed in the 3-finger gripper is used. The assumptions regarding the precision of the platform are mentioned later, as this has an effect on the search area of the robot movements.

10.1 Theoretical solution

Since the rotation of the workstation coordinate system is only around the vertical axis and the searched planes are all orthogonal, only four points are needed to determine the position and orientation of the workstation coordinate system $\{W\}$. The first two search points are along the estimated x axis of the workstation, thus yielding the rotation of the xz plane around the z axis. Points 3 and 4 are used to determine the location of the yz and xy planes, respectively, since the orientation of the xz plane is known. The intersection of all three planes can then be found, in order to determine the origin of the coordinate system. The search strategy is shown in Figure 10.1, along with the coordinate systems $\{W\}$ and $\{R\}$. Note that the rotation in the figure is exactly 180° around the z axis - this is not the case in the real application, however still a point of reference for the further work with this exercise.

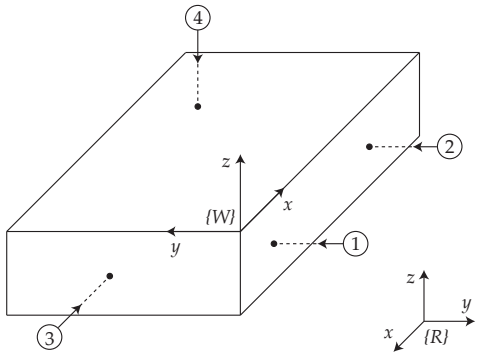


Figure 10.1: Search strategy of the calibration, where the circled numbers specifies the starting points for the search, the arrows depict the search direction, and the dots specify the points measured by contact with the workstation

Since the rotation of the coordinate system is only around the z axis, the rotation is calculated purely by using vectors. This enables the calculation to take place in only the xy plane, since ${}^{R}\{W_z\}$[2] is merely the measured z coordinate of point 4. Hence, the

[2]The z coordinate of the origin of $\{W\}$ expressed in $\{R\}$

following vector is established:

$$\overrightarrow{P_1P_2} = \begin{vmatrix} x_2 - x_1 \\ y_2 - y_1 \end{vmatrix} \tag{10.1}$$

which describes the direction of the x axis of $\{W\}$, $\{W_x\}$, expressed in $\{R\}$. The angle between $\{R_x\}$ and $\{W_x\}$ is the angle between the unit vector in the x direction $\overrightarrow{i}$ and vector $\overrightarrow{P_1P_2}$, which can be calculated by

$$\begin{aligned} \alpha &= \arccos \frac{\overrightarrow{P_1P_2} \cdot \overrightarrow{i}}{|\overrightarrow{P_1P_2}| \, |\overrightarrow{i}|} = \arccos \frac{\begin{vmatrix} x_2 - x_1 \\ y_2 - y_1 \end{vmatrix} \cdot \begin{vmatrix} 1 \\ 0 \end{vmatrix}}{\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \cdot 1} \\ &= \arccos \frac{x_2 - x_1}{\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}} \end{aligned} \tag{10.2}$$

This angle[3] is necessary to know in order to calculate the coordinates of the origin of ${}^R\{W\}$. This calculation will solely take place in the xy plane as well, and also by using vectors. The thought is to project the $\overrightarrow{P_1P_3}$ vector onto the $\overrightarrow{P_1P_2}$ vector, and adding the $\overrightarrow{OP_1}$ vector to this, thus yielding the xy coordinates of the projection, i.e. the two first coordinates of the origin of $\{W\}$. The approach is visualized in Figure 10.2.

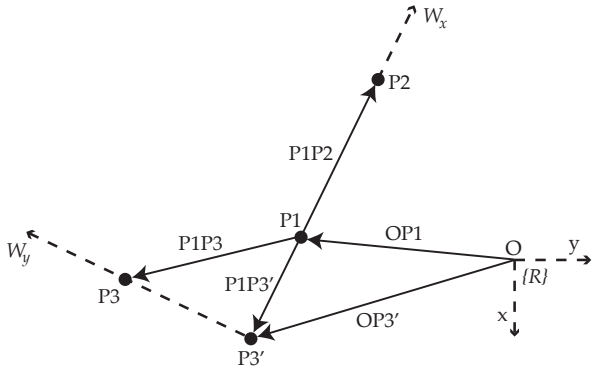


Figure 10.2: 2D-representation of the vectors used for the calculation of the transformation from $\{R\}$ to $\{W\}$

Using vector projection rules, the following calculation of the projection $\overrightarrow{P_1P_3'}$ can be made:

[3]Note that since *arccos* always yields an angle between 0 and 180°, this will be taken into consideration in the implementation in KRL

$$\begin{aligned}\overrightarrow{P_1P_3}' &= \frac{\overrightarrow{P_1P_3}\cdot\overrightarrow{P_1P_2}}{|\overrightarrow{P_1P_2}|^2}\,\overrightarrow{P_1P_2} \\ &= \frac{(x_3-x_1)(x_2-x_1)+(y_3-y_1)(y_2-y_1)}{\sqrt{(x_2-x_1)^2+(y_2-y_1)^2}^2}\begin{vmatrix} x_2-x_1 \\ y_2-y_1 \end{vmatrix} \end{aligned} \tag{10.3}$$

which yields the coordinate of the origin in the xy plane:

$$\begin{aligned}\{W_0\} &= \overrightarrow{OP_3}' = \overrightarrow{P_1P_3}' + \overrightarrow{OP_1} = \frac{\overrightarrow{P_1P_3}\cdot\overrightarrow{P_1P_2}}{|\overrightarrow{P_1P_2}|^2}\,\overrightarrow{P_1P_2} + \overrightarrow{OP_1} \\ &= \frac{(x_3-x_1)(x_2-x_1)+(y_3-y_1)(y_2-y_1)}{\sqrt{(x_2-x_1)^2+(y_2-y_1)^2}^2}\begin{vmatrix} x_2-x_1 \\ y_2-y_1 \end{vmatrix} + \begin{vmatrix} x_1 \\ y_1 \end{vmatrix} \end{aligned} \tag{10.4}$$

Hence, from equations 10.2 and 10.4, the following coordinates express the coordinate system of the workstation with respect to the coordinate system of the robot:

$$x = \frac{(x_3-x_1)(x_2-x_1)+(y_3-y_1)(y_2-y_1)}{\sqrt{(x_2-x_1)^2+(y_2-y_1)^2}^2}\,(x_2-x_1)+x_1 \tag{10.5}$$

$$y = \frac{(x_3-x_1)(x_2-x_1)+(y_3-y_1)(y_2-y_1)}{\sqrt{(x_2-x_1)^2+(y_2-y_1)^2}^2}\,(y_2-y_1)+y_1 \tag{10.6}$$

$$z = z_4 \tag{10.7}$$

$$a = \pm\arccos\frac{x_2-x_1}{\sqrt{(x_2-x_1)^2+(y_2-y_1)^2}} \tag{10.8}$$

$$b = 0 \tag{10.9}$$

$$c = 0 \tag{10.10}$$

This concludes the theoretical solution of the task, whereby the implementation of the task can begin.

10.2 Test setup and programming the LWR

A temporary test setup for this application has been set up at the laboratory, since the LWR is mounted in a stationary position. The test setup is merely three metal blocks sitting on top of each other to provide a sufficient height. The metal blocks are chosen over a constructed fixture, since the only aim of the installation is to demonstrate the task at hand, providing sufficient resistance without moving the blocks, while at the

same time providing an easy method of adjusting the position and angle of the block to be calibrated.
Since the setup is different than an application would be on the Little Helper, one non-essential difference from an actual application is the starting point of the search pattern. In an application on the Little Helper, the starting point, from which all motions regarding the search is made relative to, would be predefined, and perhaps common for all workstations. In this setup, however, the starting point is chosen by the user.
The structure of the program is explained in the following, without going into the actual KRL syntax. Since this is merely a sequential program, it is described by an enumerated list in the order of operations.

1. Basic initialization of variables, including the `$STIFFNESS` structure, the active tool and the reaction force the robot should measure when finding the block, in this case set to $5N$.

2. Declaration of an interrupt[4], to be activated when the search begins.
   This interrupt is declared to be activated when the measured force in the tool z direction is above the desired reaction force. After declaration, the interrupt is immediately switched off, to avoid miscalculation[5].

3. Move to HOME position.
   This is tradtitionally the start of all KRL programs, since it ensures a safe starting point of the robot.

4. Make the robot movable by hand, so the user can move the TCP to the desired starting position.
   This is done by entering cartesian impedance mode, and setting the stiffness of the virtual spring in the x, y and z direction to 0 $^{N}/_{m}$, and the stiffness in the rotation of these axis to 200 $^{Nm}/_{rad}$, thus enabling translation, but disabling rotation of the TCP.

5. Display a dialog message, where the program halts until the user presses OK to continue the program, when the TCP is in an appropriate starting position.

6. Switch back to position control, and move to the first point relative to the starting position. A rotation of 90° is made around the TCP, so the pin points straight

[4]Method to interrupt a program in KRL when a certain condition is met

[5]Since the force/torque acting on the TCP is back estimated from the torque in each joint, and the accuracy is therefore not adequate for continuously having the interrupt enabled

towards the block. The distance from the starting point is chosen to be $40mm$, to overcome the tolerances of the platform in an actual application and to avoid collision with the block.

7. Execute search motion towards the block.
   Since the search motion is the same for all points, it is stored as a function. This function executes the `TRIGBYCONTACT` function, enabling the robot to switch to cartesian impedance mode upon contact with the block. The function also enables the interrupt, and finally moves the robot slowly towards the block.

8. The controller mode is switched and the interrupt is activated when the desired reaction force is measured.
   When the interrupt is activated, it disables itself, saves the position (in $\{R\}$) and moves the tool away from the block.

9. Move to each of the points 2, 3 and 4 and repeat steps 7-8.

10. Calculate the new workstation coordinate system, according to the equations 10.5-10.10.

11. Take the aforementioned *arccos* problem into consideration, by making an `if` structure comparing the measured y coordinates of points 1 and 2 to determine the sign of the angle a; if $y_2 < y_1$ the sign of the calculation is changed.

12. Activate the calculated coordinate system.

One note to the above is the inability of KRL to handle more than basic mathematical calculations (though arccos is a built in function). To accommodate this, a custom function taking the square of a single parameter has been created, and an array of temporary values, obtained through basic mathematical operations, has been created and utilized for intermediate results. The program can be found on the enclosed CD in `Source Codes\LWR Projects\Basecalib.src`.
An initial visual inspection of the accuracy of the calibration reveals promising results, but this of course has to be tested further. This will be dealt with in the following.

10.3 Accuracy of the calibration

In order to test the accuracy of the robot, different tests could be made. The one chosen is a purely visual inspection, by moving the robot to predefined coordinates in

the workstation coordinate system, and checking how the TCP aligns with the points. The repeat accuracy is also tested, by comparing the values of the coordinate system of repeated measurements with a fixed position and orientation of the block. A video showing the measurement of the coordinate system and the test of precision is found on the enclosed CD in `Media\Video\BaseCalib.wmv`.
Since the inspection of the accuracy is purely visual, and the tolerances of the marks made on the block to inspect the accuracy can not be guaranteed, the approach in determining the accuracy should be considered. There is indeed a significant chain of tolerances from the production of the pin used for the calibration, the tool calibration of this pin, and how the pin is attached in the gripper, to how the marks on the block are measured. The following approach is therefore used:

1. Perform measurements on the block several times, with the block fixed in the same position and rotation for each measurement.
   For each measurement, the apparent error in each of the points is inspected and noted, along with the coordinates and rotation of the coordinate system. The latter is used to determine the repeat accuracy of the calibration.

2. Perform several measurements on the block, where the block is rotated and moved each measurement. If the observed error in the points remain the same, the precision of the calibration is deemed satisfactory, though a quantification of the precision of the calibration will be hard.

Initially, ten measurements on the same position of the block is made. The calculated coordinate systems are all within $\pm 0,5mm$ of each other, and within $\pm 0,25°$ for a. The observed errors of the inspection marks are within $0,5mm$, which is estimated to be the least interval the human eye can see with some degree of certainty. The mean of the inspected errors are shown in Table 10.1.

Point 1			**Point 2**		
x	y	z	x	y	z
1,25	0,00	0,10	0,65	1,00	-1,65

Table 10.1: Mean error observed in each of the two test points

The repeat accuracy of the calibration is promising, since it suggests a satisfactory accuracy for e.g. picking up parts. One aspect which the visualization of the measurements reveal, however, is that the block is both turned and moved slightly for each measurement. This is especially apparent from the coordinates of y and a. All the measurements are shown in Figure 10.3.

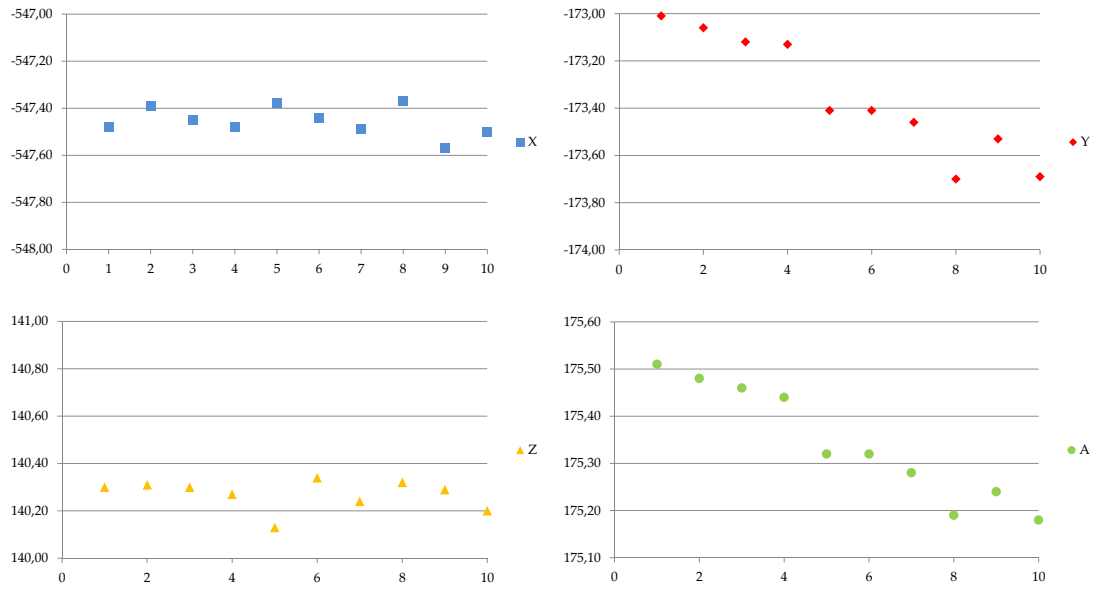


Figure 10.3: Consecutive measurements of the coordinate system; horizontal axis is the measurement no. and vertical axis is mm (for x, y, z) or degrees (for a)

It is obvious from these measurements, that the block is rotated slightly for each measurement. The most likely scenario, when looking at the data, is that the measurement in point 1 has rotated the block slightly, although not enough to change the x coordinate significantly. When looking at the z coordinate, however, which is measured at the one location where the robot could not have influenced the position of the block (since this is on top of it), the measurements are within $\pm 0,1mm$. Compared to a specified repeat accuracy of the robot of $\pm 0,05mm$ [11], the repeat accuracy of the measurement is determined to be adequate for applications on the Little Helper Plus, given of course that the workstation or platform does not yield when touched by the robot.
The test of the actual accuracy of the measurement is then carried out, by rotating and moving the block within the tolerances of the platform. The measurements are shown in Table 10.2.

Error point 1			**Error point 2**			**Measured coordinates**			
x	y	z	x	y	z	x	y	z	a
1,5	0,0	0,0	0,5	1,5	-1,5	-552,29	-166,27	140,20	178,61
1,5	0,0	0,0	1,0	1,5	-1,5	-560,51	-156,12	140,32	-176,37
1,5	0,0	0,5	0,5	1,0	-2,0	-550,88	-162,76	140,33	179,43
2,0	0,0	-0,5	1,5	2,0	-1,5	-570,44	-150,33	140,40	-172,94
1,0	-0,5	0,0	0,0	0,5	-2,0	-542,82	-188,96	139,74	164,82
2,5	0,5	0,5	1,0	2,0	0,0	-591,04	-142,64	140,56	-170,04

Table 10.2: Measurements on various positions and rotations of the block

Note that the two final measurements are made to test the measurements outside the tolerances of the platform, as is obvious from the calculated rotation and position. The remaining measurements yield somewhat promising results, since the observed errors are relatively close to the mean of errors observed in the first 10 measurements. All the results are found on the enclosed CD in `Results\Calibration.xlsx`. The precision of the calibration is determined to be adequate for handling parts or e.g. move the camera to a satisfactory image acquisition location for improved precision of parts handling, or quality control of parts. Since this calibration only takes 45 seconds with the robot moving at 30% speed, the calibration is comparable to the fast version of the vision calibration currently used on Little Helper, though with a slightly better precision. This calibration takes 10-30 seconds and results in an accuracy of $\pm 1,0mm$ [2].
The task of calibrating the coordinate system is now carried out, with respect to the given assumptions. The basis of creating a task on the Little Helper to calibrate using this method is then formed, since this yields an adequate accuracy compared to the high speed calibration currently implemented using vision. For further tests of accuracy, it is recommended to create a better test setup, that is fixed in position and rotation, to avoid moving the block when the measurement is carried out.

11

Peg in hole

A robot task that is traditionally very hard to accomplish for regular, position-controlled robots, is the task of placing a peg in a hole. The reason for this is a very low permissible error, both in position and rotation of the peg relative to the hole. With tight tolerances, the task obviously becomes more difficult. This exercise aims to investigate the possibility of using the torque sensors of the LWR to solve the peg in hole task.
Initially, the specific peg in hole task in this scenario is programmed using only the position control mode of the robot, to determine the permissible errors if this was the only available controller mode. After this, the task is sought solved by entering cartesian impedance mode, and letting the spring-damper system compensate for the error in position and rotation.

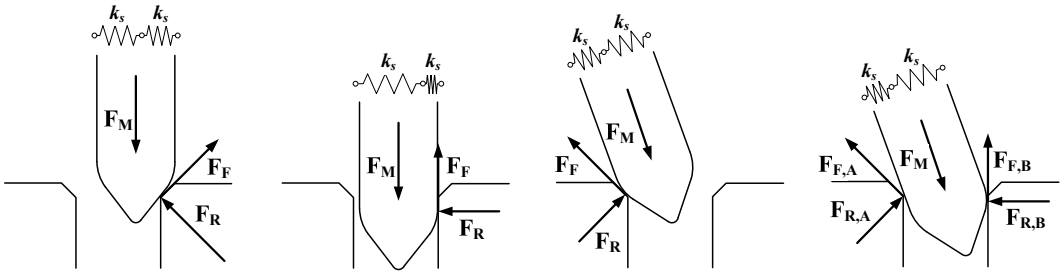
Purpose To investigate the peg in hole task, and how this task can be solved by using the torque sensors on the LWR.

Objective To gain an understanding of the difficulties in the peg in hole task, and to establish a method of solving this using the LWR.

The test setup is using the same pin used for the base calibration task described in Chapter 10, which has a conical tip. The hole is made in a plastic block, which is fastened to the table. The hole diameter is $0,5mm$ larger than the pin diameter, and chamfered at the edge, making the test setup a relatively easy task, compared to peg in hole tasks in assembly operations in a production environment, e.g. fitting a shaft in a bearing.
The following will describe the different scenarios, which has been programmed. Common for the programs, is that the robot is made movable by hand, so the user moves the robot to the starting position for the insertion of the pin, after which the robot moves $40mm$ (which is the height of the block, in which the hole is made all the way through) downward in the tool direction. In order to investigate different scenarios, two different ways to move the robot are implemented; one where the tool is merely movable in the xyz direction, thus keeping the alignment between the hole and peg, and one where the tool rotation is enabled as well.

11.1 Using Position control

When attempting to solve the task using regular position control, the only thing compensating for reaction forces acting on the TCP, is the current applied to the drives in the robot, and how the position is regulated in these joints, along with the fit of the peg and hole. Since the hole is chamfered and the peg is conical, quite large reaction forces can be achieved when attempting to place the peg in the hole; reaction forces which along with friction forces limits the robots ability to place the peg accurately in the hole.
Using the position control mode, only small errors in position are permissible, due to the large reaction forces. In the case of no rotation of the tool, the reaction force is orthogonal to the direction of the motion, since the force causing the reaction force is acting on the internal wall of the hole. This force is due to the inability of the drives to keep the TCP in exactly the same position and orientation at all times, and can be thought of as a spring stiffness k_s of the system, not to be confused with the virtual spring-damper system in the cartesian or joint impedance control modes. The stiffness of these springs is very large[1], and as such a small error in the insertion position results in very large reaction forces F_R, and subsequential friction forces F_F, which makes the robot unable to insert the peg in the hole. The active forces are shown in Figure 11.1, where the force in the motion applied by the robot is denoted F_M.



(a) Initial contact (b) Inserted contact (c) Initial contact (d) Inserted contact

Figure 11.1: Forces acting on the peg during insertion in two scenarios; (a) and (b) without rotation of the peg and (c) and (d) with rotation of the peg. Spring forces are not shown, but are caused by the system stiffness k_s

This figure also shows the case of rotation of the peg relative to the hole. This scenario results in an added reaction force, which subsequentially result in an increase in the

[1]The TCP can only be translated about $5mm$ by hand

overall friction force preventing the insertion of the pin. The reaction forces also result in a moment acting on the peg. This moment contributes very little to the motion of the robot, however, since the tip of the peg is a relatively great distance from the tool flange of the robot.
Trials in each of the above scenarios have been made on the maximum permissible error, which are shown in Table 11.1. The resulting insertions are evaluated by visual inspection, where a good insertion is a full insertion of the peg in the hole, an okay insertion is close to good, and a partial insertion is only some of the way, after which the robot motion stops, since the pin can not be inserted further. The conclusion stands that the

x [mm]	**y** [mm]	**b** [°]	**c** [°]	**Conclusion**
0,0	-2,0	0,0	0,0	Partial insertion
0,0	-1,5	0,0	0,0	Partial insertion
0,0	-1,0	0,0	0,0	OK insertion
0,5	-0,5	0,0	0,0	Good insertion
0,0	0,0	1,0	-1,0	Partial insertion
0,0	0,0	0,0	-1,0	Partial insertion

Table 11.1: Errors in position and angle, and the resulting insertion of the peg, using the position control mode

permissible error in both rotation and position are very small. Taking the difference between the hole diameter and peg diameter into consideration, it is obvious that the peg should be positioned very accurate before attempting to place it in the hole. It is also apparent, that the contribution from the second friction force in the case of rotation of the peg decreases the pegs ability to slide into the hole.
It is obvious from the tests why this task is traditionally hard to accomplish with position controlled robots, as very small errors are permissible, even in this ideal scenario. Therefore, it is investigated how the LWR can be utilized to better accomplish the task.

11.2 Using Cartesian Impedance control

The straightforward way to accomplish the task at hand, is to apply the Cartesian Impedance controller mode, when the peg is near the hole. For this purpose, the `TRIGBYCONTACT` function is utilized, and configured to switch to the Cartesian Impedance controller when a reaction force is registered. The applied parameters for the cartesian impedance controller are a low spring stiffness in the x and y direction, of $200^{N}/_{m}$, for the scenario without rotation. This scenario is similar to that in Figure 11.1(a) and 11.1(b), where the system stiffness k_s now represents the low stiffness of the virtual

spring, thus resulting in much lower reaction and friction forces. This is also obvious from the measurements shown in Table 11.2, for the scenario without rotation of the peg, since the peg is inserted correctly as long as the tip of the peg is within the chamfer of the hole. Even when the pin is the farthest from the center of the hole, while still being within the hole diameter, the maximum force exerted on the hole wall can be calculated from the spring stiffness to be just $1N$.

x [mm]	**y** [mm]	**b** [°]	**c** [°]	**Conclusion**
-4,5	-4,0	0,0	0,0	No insertion (hole out of range)
-3,5	0,0	0,0	0,0	Good insertion
0,0	-4,0	0,0	0,0	Good insertion
4,8	0,0	0,0	0,0	Good insertion
0,0	5,0	0,0	0,0	Good insertion
0,0	0,0	1,0	0,0	Partial insertion
1,0	0,0	-1,0	-0,5	OK insertion
-1,5	-1,0	-1,9	-3,0	Partial insertion
-1,5	-1,0	-2,0	-1,0	Partial insertion

Table 11.2: Errors in position and angle, and the resulting insertion of the peg, using the cartesian impedance control mode.

The rotation of the peg still presents a problem, however. This has most likely to do with the resulting moment in the peg, due to the reaction forces, being too small to overcome the spring stiffness, since the force in the TCP is back estimated from the joint torques. This effectively means that even setting the torsion spring stiffness of rotations a, b and c to 0 $^{Nm}/_{rad}$ does not result in the peg sliding into the hole when it is rotated more than a few angles relative to the hole. A reason for this, apart from poor estimation, is that the wrist of the robot, where the tool flange is located, needs to be moved quite much in order to rotate the peg around its tip; merely specifying low torsion spring stiffness is definitely not enough to result in this motion. The permissable error in rotation is however greater than for position control.
Based on the experiments described above, several implementation suggestions, should the task be solved in a real environment, are presented in the following.

11.3 Implementation in a production environment

It is obvious from the permissible errors in this near ideal test setup, with a conical tip and a chamfered hole, that some additional work is required to solve the peg in hole task in a production environment. This is not to suggest an industrial environment is

less organized and controlled than this setup, but is regarding the shape of the parts - it is a rare situation where e.g. one has a shaft with a conical tip to insert in a chamfered bearing. In an industrial environment however, it is estimated that there is a very small error in the position of the part, and even smaller on the angle, making the shape of the peg and hole the main issue. The following will suggest methods for implementing solutions to both of these problems, however, using the features of the LWR.
A search algorithm could be implemented fairly easy, which searches for the center of the hole if the starting point is not adequate. Similar to the base calibration task, the peg could touch the surface near the hole. If this surface is known relative to the robot, the condition for determining if the hole is found could be the coordinate of the contact on the axis orthogonal to the surface. If the surface is not known, however, the condition could be the force in the tool z direction compared to the x and y directions. The comparison is important, since the position of the hole could be found approximately at the first try, resulting in the scenario in Figure 11.1(a); it is however unlikely that the hole can be found with such great precision as to not yield any reaction force in the x and y directions.
This does not solve the matter of small permissible errors in the angle between the peg and hole. This has to be solved using a different approach than watching the moment acting on the peg, since the moment around the tip of the peg can not be accurately back estimated from the joint torques. One way could be to use the back estimation of the forces acting on the tip of the peg in the xyz directions, as these are more accurate than the moments. It could be checked during runtime, and even during the robot motion downwards, if there was a reaction force in e.g. both the z and x directions, and compensate for this by rotating the peg slightly around the y axis of the tool, to reduce the reaction forces. Obviously, this is not enough, as seen from Figure 11.1(c). Here the reaction force F_R can be decomposed to a reaction force in two axis in the tool coordinate system, in which the force F_M is in the positive z direction. Using the mentioned strategy, this would cause the robot to rotate the peg the wrong direction, increasing the angle between the peg and the hole. This could however be solved by implementing a limit on the reaction force in the z direction, to ensure this reaction force is from the contact with the internal wall of the hole, and not from a sliding contact on the edge of the hole.
In terms of inserting a sharp-edged peg in a sharp-edged hole, this could be made fairly easy if the rotation and approximate position of the hole is known. One way to do this could be to program a robot motion similar to what a human would do when inserting a peg in a hole. In this case, we often rotate the peg, and insert the tip of the peg into

the hole, after which we align the rotation of the peg with the hole. A similar method could be utilized on the robot, since the initial insertion of the tip in the hole is fairly easy. After this, the cartesian impedance controller should be enabled with low spring stiffness, and the peg rotated about its tip, after which the peg should slide into the hole for further insertion.
The methods described above have not been implemented, since the goal of this exercise has not been to program a working peg in hole skill, but rather to investigate the challenges of the task, and provide the basis for further work on establishing a general solution to the task.

12 Demonstration of the Fast Research Interface

KUKA supplies the Fast Research Interface for use with the LWR, to enable real-time control of the LWR from a remote computer, mainly for experiments in research laboratories. One immediate use for this on the mobile manipulator is to facilitate control of the robot from the remote computer on the platform. Since the current software architecture of the Little Helper requires this connection, and there is no support for TCP/IP connections in the KRC[1], the FRI will most likely be used for the connection to the robot. Apart from enabling control of the robot, several monitoring variables are available as well, which could be utilized in certain other scenarios. Work has also been done on creating an open source controller for the robot running on a remote PC [12][2], but it is unsure if this will be utilized in the Little Helper Plus.
The structure of the FRI requires some explanation, and thus will be presented first in this chapter. After this, an initial test program monitoring the torques in each joint is documented in depth, and finally the full program, which enables monitoring of all variables and control of the robot, is described.

12.1 Function of the FRI

In the following, the communication of the FRI and the structure of the packages exchanged between the robot and the remote PC is explained.
There are two main operating modes of the FRI; *Monitor Mode* and *Command Mode*. In Monitor Mode, a packet containing all the available robot data is sent via UDP from the controller (KRC) to the remote PC every cycle, and command parameters are sent from the PC to the KRC. It is only when the FRI is in Command Mode, however, that the command parameters for controlling the robot are utilized. The data exchange is visualized in Figure 12.1.

[1] Although KUKA offers the KUKA.Ethernet package for communicating XML strings through TCP/IP

[2] This article uses the XML interface, but the work has later been expanded to use the FRI for the communication, though this is not described in papers

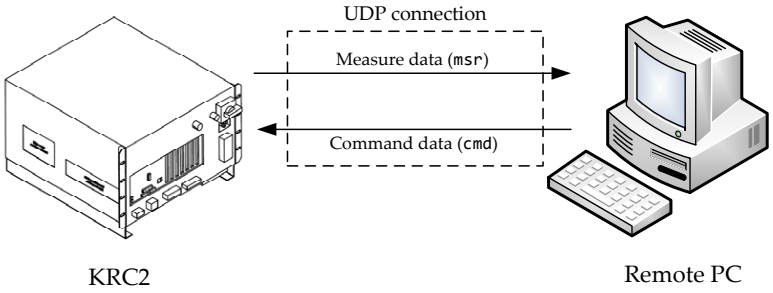


Figure 12.1: Data exchange between the controller and the remote PC through the FRI

The cycle time is user specified upon opening the connection, between 1 and 100ms - in the following implementations the default value of 20ms is used. In order to enter and stay in Command Mode, the quality of the connection should be one of the top two classifications of four, as specified in the FRI manual [13]. This quality is decreased one step every time a packet is lost, and increased when a packet is received. The four states and changes of quality is shown in Figure 12.2.

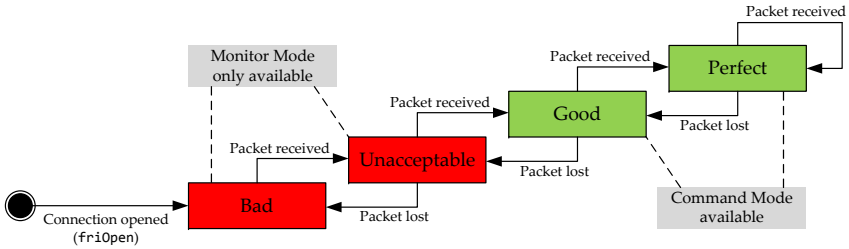


Figure 12.2: The four quality states of the FRI, and how they change

Nearly no configuration is needed of the FRI, since this is a KUKA Technology Package installed from the factory. There are two settings that should be entered in the controller; the IP addresses of the KRC and the remote computer, and the mapping of I/O to the FRI, to enable the direct control of auxiliary equipment. These settings will not be described here, as they are described in the user manual for the FRI [13]. The remote PC should obviously also support the UDP protocol, since this is the protocol used by the FRI [13].

Although the FRI comes preinstalled on the KRC, only the source code of a sample program with a simple GUI is supplied for use on the remote PC. No installation is required on the remote PC, to enable use of the FRI, except installing the free Visual Studio C++ Express to compile the program, but KUKA supplies C++ header files,

which ensure proper communication with the KRC, along with definition of the data structures, when included in a user-developed program.

12.1.1 Data structure of the packages

Data exchange between the KRC and the PC is handled by two data structures, called `msr` (measure) and `cmd` (command). These are defined in the header files as structures, each containing substructures for groups of data. Only the substructures will be described here, along with examples of the data contained in these, since a full description is available in the user manual [13].
One common substructure of both telegrams is the package header `head`, which ensures proper communication between the two parts. The header contains information about the binary size of the package, and the sequence count, which is compared between the two telegrams, to calculate the latency and packet loss. Another common substructure is `krl`, which is a set of 16 integer, 16 floating point and 16 boolean variables, that are all user assignable, and as such can be used to transfer data both to and from the remote PC, depending on which parent structure is used. If `msr` is used, the values are sent from the KRC to the PC, and vice versa (see Figure 12.1). Note that the FRI does not have to be in Command Mode to change the values of `cmd.krl`.
Apart from the two substructures already mentioned, the following substructures are available in the `msr` telegram:

`msr.intf` which contains information about the status of the FRI, such as the time stamp of the connection, state of the FRI (monitor/Command Mode) and quality of the connection. This structure has its own substructure, `msr.intf.stat`, which contains the statistics based on the sequence count, such as latency, package loss etc.

`msr.robot` contains general information about the robot, i.e. if the power for the drives are on, which controller strategy is active and the operating temperature inside each joint.

`msr.data` is the largest of the substructures. In this substructure, the position data of the the robot is sent, in both the joint space and cartesian frame, along with the actual joint torques, and the estimated external joint torques and TCP force/torque values. The Jacobian matrix and the mass matrix are contained in the structure as well.

In the `cmd` telegram, only the `cmd.cmd`[3] substructure is available, apart from the `cmd.head` and `cmd.krl` structures previously described. This substructure is similar to `msr.data`, since it contains the commanded positions and stiffness/damping parameters, also both in joint space and cartesian frame, as well as an additional joint torque or TCP force/torque to be exerted, similar to the `DESIREDFORCE` command in KRL. These data are used to control the robot, and can be specified from the remote PC.

12.1.2 Programming from the remote PC

To facilitate communication through the FRI, KUKA has supplied a number of C++ header and source files. The function of the supplied files is to establish the connection, and define the various package structures, as well as supply easier interaction to the FRI. The supplied header and source files are:

`friComm.h` which is the definition file for the various structures, along with initialization of internal variables to ease the programming further on.

`friUdp.h` is the header file for the UDP connection.

`friUdp.cpp` is the source file for the UDP connection, with functions for sending and receiving the packages.

`friRemote.h` which defines the `friRemote` class, that aims to ease the programming by providing a number of standard functions e.g. for sending data or controlling the robot.

`friRemote.cpp` is the source file for `friRemote.h`, which defines the functions previously mentioned.

Note that the two latter files are not necessarily needed, but greatly assist in programming, since the functions supplied in these files make the communication with the robot much easier.

12.2 Hello FRI

In order to learn the basics of programming an application that communicates with the KRC through FRI, a simple console application has been developed. The sole purpose of the application is to establish a connection to the FRI, and maintain this connection until

[3] Denoted `cmd.data` in the FRI manual [13], which is an error

the program is terminated. When the program is running, it outputs the timestamp, sequence count and the estimated external torque in each joint to the console every second. The following will describe the program in depth, to provide the reader with an understanding of how programming to the FRI is carried out.
At the beginning of the program, a number of variables are declared, mainly the variables for the `msr` and `cmd` telegrams, along with the variable `myFRI` of the `friRemote` class. After the declarations are made, a handshake to the FRI is carried out, by doing an initial data exchange and filling the `msr` and `cmd` variables on the remote side with data. The current timestamp of the KRC is saved in a variable as well, to later determine when to output data to the console. The source code is as follows:

```
1  int main (int argc, char *argv[])
2  {
3    // Declaration of variables
4    int wait = 1;             // Wait 1 second between writing data to console
5    friRemote myFRI;          // The class providing easier communication functions
6    tFriMsrData msr;          // The measure telegram, of type tFriMsrData
7    tFriCmdData cmd;          // The command telegram, of type tFriCmdData
8    FRI_QUALITY myQUAL;       // The quality of the connection
9    // Initial data exchange
10   myFRI.doReceiveData();    // Receive data from the KRC
11   msr=myFRI.getMsrBuf();    // Fill the msr buffer in the friRemote class with the data from KRC
12   cmd=myFRI.getCmdBuf();    // Initially fill cmd buffer in this application with data from the KRC
13   myFRI.doSendData();       // Send data back to the KRC
14   // Save the current timestamp
15   double time = msr.intf.timestamp;
```

After the initial data exchange and declaration of variables, the connection is established and maintained until the program is terminated. This is done through a simple `for` loop, with no end condition, in which certain values are extracted each cycle time, and data exchange is carried out. In the beginning of this loop, it is sometimes necessary to write values to the `cmd` structure, to ensure proper data exchange. Apart from this, the sequence count in the `msr` and `cmd` structures should match, before data exchange is carried out. The source code for the first part of the `for` loop is as follows:

```
16   for(;;)
17   {
18     // Handshaking via the user variables, used to ensure data exchange
19     myFRI.setToKRLInt(1,1);                          // Write value 1 to user variable TO_INT[1]
20     myFRI.setToKRLReal(0,myFRI.getFrmKRLReal(1));    // Write user variable back to controller
21     // Save the current quality of the connection
22     myQUAL = myFRI.getQuality();
23     // Send back the correct sequence count, so the packages match
24     cmd.head.reflSeqCount=msr.head.sendSeqCount;
25     // Data exchange
26     myFRI.doReceiveData();
```

```
27      msr=myFRI.getMsrBuf();
28      myFRI.getCmdBuf()=cmd;
29      myFRI.doSendData();
```

In order to only output the measured torques each second, and not every time a package is received, an `if` structure is made to ensure this. If the saved timestamp when the connection was established is less than or equal to the current timestamp minus the `wait` variable, the timestamp, sequence count, quality and joint torques are printed to the console, after which the `time` variable is update with the current timestamp, and the loop continues. Notice the two different ways of accessing the data; in line 34, the data is accessed directly, and in line 35, the data is accessed by using a function in the `friRemote` class:

```
30      if(time<=msr.intf.timestamp-wait)
31      {
32        // Output data to the console
33        cout << "----------------------------------------------------------------------" << endl;
34        cout << "Timestamp: " << msr.intf.timestamp << endl;
35        cout << "SequenceCounter: " << myFRI.getSequenceCount() << endl;
36        cout << "FRI quality is: " << myQUAL << endl;
37        cout << "Joint torque vector is: " << endl;
38        // For loop to print the names for each joint as well as the external joint torque
39        for(int J=0;J<LBR_MNJ;J++)
40          {
41          cout << "J" << J+1 << "  " << msr.data.estExtJntTrq[J] << endl;
42          }
43        cout << "Waiting for " << wait << " second..." << endl;
44        cout << endl;
45            // Update the timestamp
46        time = msr.intf.timestamp;
47      }
48    } // End of for loop
49    return 0;
50  } // End of program
```

After this initial program has been developed, and an understanding of the header and source files supplied by KUKA is gained, development of the main program to demonstrate all features of the FRI is begun.

12.3 Full demonstration of the FRI

In order to fully demonstrate the capabilities of the FRI, a Windows application has been developed, which aims to cover all functionality of the FRI. The goal of this application is therefore to establish and maintain a connection to the KRC, continuously display

data of the connection and the robot itself, as well as provide means of controlling the robot using the predefined controller strategies. This application is greatly inspired by the user interface supplied as an example by KUKA, since some of the main objects and methods of this application has been incorporated in the same way. The supplied example, however, solely enables joint position control of the robot, and not joint or cartesian stiffness. This has been incorporated in the example at hand.

12.3.1 Description of the GUI

When opening the program, the user is met by a simple GUI with a section, displaying the quality and mode of the connection, as well as the power state of the robot, along with a button to start or stop the connection. Apart from this, a button to write both the `cmd` and `msr` packages to an XML file has been implemented, in order to easily view or save the data of the robot. The main part of the GUI is a number of tabs, which enables the user to switch between various data to view, as well as a tab to control the robot. The GUI is shown in Figure 12.3, with the *Connection Stats* tab selected.

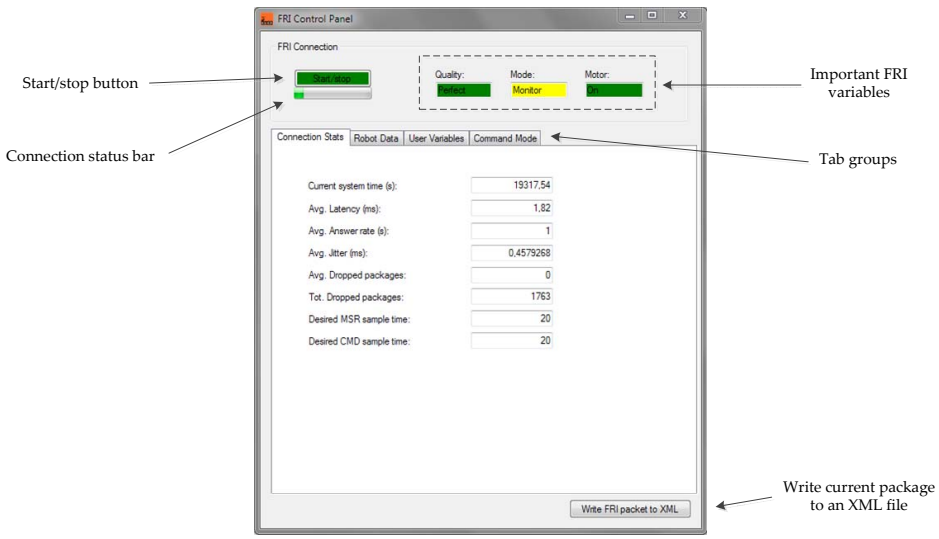


Figure 12.3: The GUI for the FRI interface, in its initial state

The available tabs in the application are:

Connection stats which displays various connection statistics from the `msr.intf` structure

Robot data which displays the data of the robot from the `msr.data` structure. There is a second tab control in this view; one for the joint-specific data, such as temperature of the drives, measured torque, joint positions, etc. and one for the cartesian data, such as measured external force/torque on the TCP and the cartesian position of the TCP.

User variables displays the 6 user variable arrays. The 8 first booleans, both to and from the remote PC is mapped to the 8 inputs and outputs on the KRC respectively, as described in Appendix C.3, and as such are highlighted. This data view is linked to both the `msr.krl` and `cmd.krl` structures.

Command Mode enables control of the robot from the GUI, through the `cmd.cmd` structure, by enabling the switching between controllers, and providing jog keys for joint and cartesian position, as well as an interface to enter stiffness, damping and additional force/torque to be exerted.

Only the data of the currently selected tab is updated continuously, in order to limit the work load on the remote PC. The GUI is updated every cycle, but in order to ensure proper quality of the connection, it is not feasible to have the connection to the KRC and the GUI functions running simultaneously on the same thread. The solution to this is explained in the following.

12.3.2 Exchanging data

In order to implement two separate threads in the application, one for handling the connection to the FRI, and one to handle the GUI, the `BackgroundWorker` .NET class is utilized in the application. This class does in fact enable a process to run on a separate thread, and provides events and properties to easily incorporate another thread in an existing application. In this application, the `BackgroundWorker` is handling the connection and data exchange to the KRC, through the `DoWork` event, which is raised when the user presses the start button. The `DoWork` event continues to loop through the connection sequence, until the user cancels the connection. This loop fills the connection buffers as previously mentioned, and calls an optional command function, for controlling the robot, performs data exchange to the KRC, as well as reporting the progress of the connection.
When using the `BackgroundWorker` to e.g. update a GUI, as is the case here, one should avoid doing this through the regular `DoWork` event, but instead do it through the `ProgressChanged` event, since this event is not executed on the same thread as `DoWork`.

The `ProgressChanged` event is raised every time the `ReportProgress` method is called, which is a method for reporting the percentage completion of the separate thread[4], and is called in the last step in the `DoWork` event. As such the `ProgressChanged` event is effectively executed every cycle time of the FRI, albeit on a separate thread. When the `ProgressChanged` event is raised, it merely updates the GUI, by a custom function. This function will not be described in depth in the following, since a great deal of the function is merely writing variables to text boxes or labels. However, some intricacies in reading certain variables will be described, since these are not obvious from the FRI documentation. The scenario described above is visualized in Figure 12.4.

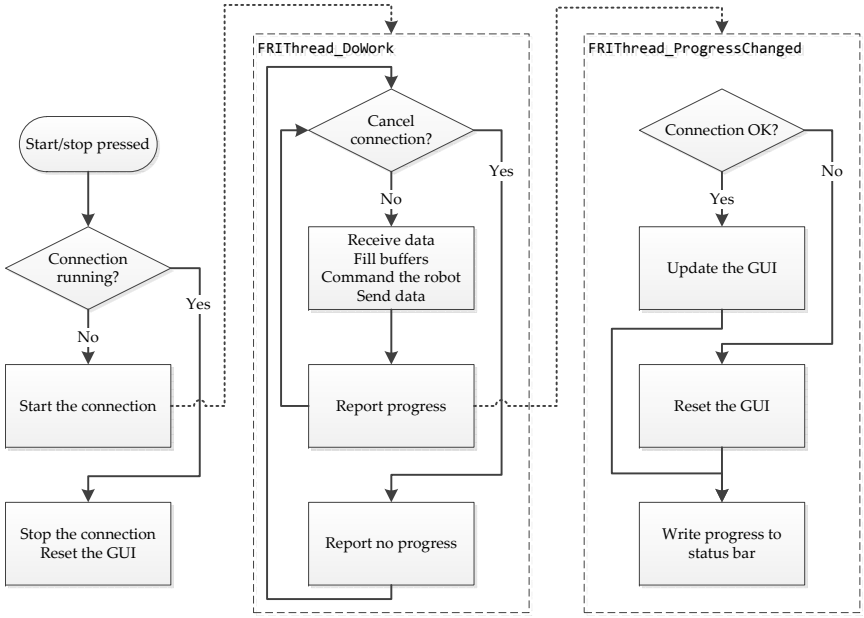


Figure 12.4: The main function of the FRI application. Dashed lines are called automatically, and indicate a change of thread, and dashed boxes indicate a separate function.

As previously mentioned, the `updateGUI` function only updates the current tab, to optimize performance. This is done by a simple switch structure, checking which tab is selected. An exception from this is the boxes in the top of the GUI, showing the quality of the connection etc. These boxes are colored depending on their value, and as such are only updated when e.g. there is a change in the quality, by comparing the current value with the value of the text boxes. There is no problem in reading the connection stats

[4]Although in this application is reporting the sequence count modulo 100, since no defined end of the connection is available

and writing these to the GUI, since these are simple integers or floats, only requiring a conversion to string before being written.

When reading the robot data, which are all specified in arrays of floats, the program should of course loop through these arrays in order to write each value to a specific cell in the robot data data view. This is done by a simple `for` loop, with the end condition being 7, for the number of joints in the LWR. For the cartesian position, however, this is specified in the 3×4 matrix $[\mathbf{R_{3\times3}}\ \mathbf{P_3}]$, specifying rotation and position of the TCP, which is cast as a vector of size 12, and as such requires some extra coding to be represented correctly.

The user data tab containing the user variables to the FRI is also straight-forward, at least for the integer and float variables, thus one `for` loop running from 1 to 16 is handling the update of all the user variables. The boolean variables are stored in a single 16-bit integer, the binary representation specifying which variables are true or false. This is common for each of the `msr` and `cmd` structures, and as such needs to be decoded in order to work with the checkboxes in the columns specifying the boolean data. This is done in two different ways, both using bitwise operators. For the user variables *from* the KRC (`msr.krl.boolData`), a bitwise AND operator (`&`) between the binary representation of the integer, and a binary with 1 on the ith place, created by performing a bitwise left shift on 1 by i bits (`1<<i`), is carried out. If this operation results in a non-zero value, the ith value is 1 (or true), otherwise 0 (or false). This method is also known as bit masking.

The data being sent to the robot is calculated in a similar way. An integer with value 0 is initialized before the loop, and is filled with the values in the `for` loop. The loop investigates whether or not the ith checkbox is checked, and returns 1 or 0 depending on the value (`?1:0`). On the returned value, a bitwise left shift of i bits are made, and this value is appended to the current value of the integer to be sent by a bitwise OR assignment (`|=`). Since any bitwise shift operation on 0 still results in 0, only the values corresponding to a checked box is sent. The code performing both operations is shown below, as part of the loop updating the user variables.

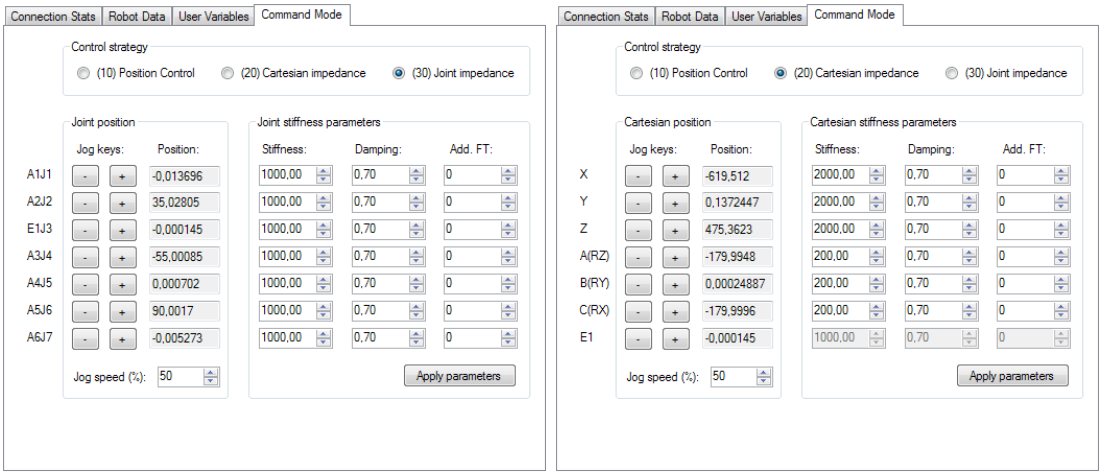
```
1  unsigned short boolData = 0;
2  for (int i=0;i<16;i++)
3  {
4    ...
5    this->dataGridViewUserData->Rows[i]->Cells[2]->Value = (msr.krl.boolData & (1 << i));
6    boolData|= (this->dataGridViewUserData->Rows[i]->Cells[5]->Value ? 1:0 ) << i;
7    ...
8  }
9  cmd.krl.boolData = boolData;
```

The Command Mode tab is always updated, but some additional information regarding the Command Mode is required in order to explain this update function. Instead, the update function is explained as part of the following.

12.3.3 Commanding the robot

In order to control the position of the robot through the FRI, the `cmd.cmd` structure is utilized. This structure contains the same parameters as the `$STIFFNESS` structure in KRL, as well as an additional parameter `cmd.cmd.cmdFlags`. This parameter specifies which settings are sent from the remote PC, and must be specified before sending any of the other parameters of `cmd.cmd`. One problem with this, however, is the value of `cmdFlags`, which is an integer, which in its binary representation specifies the sent settings, similar to the `cmd.krl.boolData` variable. This value cannot be changed in Command Mode, however, so before switching controller or sending additional parameters, the FRI connection should be in Monitor Mode.
Another aspect in controlling the robot, is that the active controller of the robot cannot be changed through the FRI. That is, if the user wishes to command cartesian position and stiffness, the LWR must first be set in cartesian impedance mode, since it is insufficient to just command the cartesian position and stiffness through FRI when the robot is in another controller mode.
These two aspects of controlling the robot through FRI are solved by having a KRL program running on the controller simultaneously. This program loops until terminated by the user, and carries out a switch statement every loop. This switch statement checks if a specific user variable, the 16^{th} integer, is changed from the remote PC, and as such switches the controller strategy accordingly. An example could be if the FRI is in Command Mode, and the cartesian impedance controller is active on the KRC, the user wishes to change to joint position control, while still maintaining the ability to control the robot. The user then writes the value 10 (for joint position control) in the last field of the `FRM_INT` column in the User Variables tab. The program running on the KRC then switches to Monitor Mode, enables a pre-defined `$STIFFNESS` structure for position control and switches the FRI back to Command Mode.
In the application on the remote PC, this change also changes the GUI of the Command Mode tab, according to the selected controller, since different values are permissible in each of these. Examples of the layouts of the command tab is shown in Figure 12.5. For the joint position controller, the layout is similar to that of the joint impedance control mode; stiffness parameters are however not available. The stiffness parameters are also reset to their default values when the controller is changed. When the `updateGUI` func-

tion is called, the measured joint positions or cartesian coordinates are written to the text boxes next to the jog keys.



(a) Joint impedance mode (b) Cartesian impedance mode

Figure 12.5: The difference in the Command Mode tab of the GUI depending on the selected controller

The FRI Command Mode is not developed for sending specific coordinates to the motion planner of the robot, which presents some problems in jogging the robot. The difficulties are different for the joint and cartesian controller mode, which is why these are explained separately in the following. All of the actions described in the following are part of the function `doCommandMode`, that is carried out in the `BackgroundWorker`, before each package is sent to the KRC.

Joint position and stiffness

Since the coordinates sent through the FRI are apparently not sent to the motion planner of the KRC, it is assumed from the function of the FRI that it is instead sent to the joint interpolator in the case of commanding joint positions and stiffness. Due to this, only an increment in position can be sent, and this increment needs to be sufficiently small, so that the LWR can carry out the motion within one cycle time, the default being $20ms$. Since the maximum velocity of all joints without payload is $112°/s$ (except joint 5, which is $180°/s$) [9], the following value for maximum increment applies:

$$i_{max} \quad = \quad 112°/s \cdot 0,020s = 2,24° \approx 0,04rad \tag{12.1}$$

Since this does not take the acceleration of the robot into consideration, it only applies when the robot is in motion. Therefore the maximum increment the user is able to send to the joints of the robot is set to $0,01 rad$, which is chosen to be adequate for jogging the robot.
Before calculating the new joint positions of the robot, the actual commanded position of the robot is first saved in the array, that later is sent to the KRC. The reason the commanded position, and not the measured, is sent to the KRC is that in the case of joint stiffness mode there is a deflection of the virtual springs - if the measured position was used here, the robot would slowly move towards the ground, since the deflected state would then be sent as the new position.
The program then checks if any jog button is pressed. If a jog key is pressed, the new joint position of the specific joint is saved in the array. The increment is calculated as the maximum increment multiplied with -1 or 1 for the direction of the motion and the desired speed of the jog motion, specified in the box below the jog keys.
In the `friRemote.cpp` source file, a function has been added to easier command the robot, and this function is utilized in the application. This function takes two arguments; the joint position array, and a boolean specifying if the function should do data exchange with the KRC. The latter is not the case here, since this is handled by the previously mentioned `BackgroundWorker` class. The function makes sure the interface is in Command Mode and sets the `cmd.cmd.cmdFlags` appropriately, as well as doing data exchange if this is specified.
In the case of the joint impedance controller, there is a separate function for this. This functions also accepts arrays of stiffness, damping and additional torque exerted in each joint. The function checks if any of the parameters are `NULL` (i.e. is not specified), and sets the `cmdFlags` appropriately depending on which parameters are specified. All parameters are sent every cycle time, but only changed if the user presses the *Apply parameters* button in the GUI.
The joint positions and stiffness parameters can now be commanded, when ensuring the change of controller through the user variable previously mentioned. The cartesian position, however, requires some additional work.

Cartesian position and stiffness

Since the cartesian position of the TCP is specified through the FRI as the 3×4 $[\mathbf{R_{3\times3}}\ \mathbf{P_3}]$ matrix, cast as an array of size 12, commanding the position directly presents some problems. First of all, the method for obtaining the rotation matrix is to be determined, and secondly a method for writing increments in angles to this rotation matrix is developed.

The rotation matrix can be calculated in different ways, depending on the order in which the rotations about the axes are carried out, and if the rotations are around the fixed coordinate system (fixed-angle rotation) or the rotated coordinate system (Euler angles). Since the a, b and c angles in KRL are the rotations about the z, y, and x axes, respectively, it is estimated that the rotation matrix is calculated in this order as well. This is confirmed by making test calculations comparing the rotation angles shown on the teach pendant with the rotation matrix sent through FRI, as well as confirming that the rotations are carried out using Euler angles [14]. This conclusion yields the rotation matrix, where $c\alpha = \cos(\alpha)$ and so forth [14]:

$$R_{Z'Y'X'}(\alpha,\beta,\gamma) = R_Z(\alpha)R_Y(\beta)R_X(\gamma) \tag{12.2}$$

$$= \begin{bmatrix} c\alpha & -s\alpha & 0 \\ s\alpha & c\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c\beta & 0 & s\beta \\ 0 & 1 & 0 \\ -s\beta & 0 & c\beta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & c\gamma & -s\gamma \\ 0 & s\gamma & c\gamma \end{bmatrix} \tag{12.3}$$

$$= \begin{bmatrix} c\alpha c\beta & c\alpha s\beta s\gamma - c\gamma s\alpha & s\alpha s\gamma + c\alpha c\gamma s\beta \\ c\beta s\alpha & c\alpha c\gamma + s\alpha s\beta s\gamma & c\gamma s\alpha s\beta - c\alpha s\gamma \\ -s\beta & c\beta s\gamma & c\beta c\gamma \end{bmatrix} \tag{12.4}$$

Although the angles resulting in the rotation matrix as received through `msr.data.cmdCartPos` can be determined from the rotation matrix, two solutions of the set of three angles exist that give the same rotation matrix. Therefore, it is determined to implement a function in the KRL script for changing controllers, that writes the angles to the user variables when switching to the cartesian impedance controller of the robot. Upon switching to this controller, the program on the remote PC saves the current position directly from the `msr.data.cmdCartPos` variable, and the angles from the user variables. The increments in position or angles are then calculated in the same way as for the joint position controller, depending on which button is pressed. The rotation matrix is calculated each time the `doCommandMode` function is carried out, since most entries in this matrix depends on several angles. In the case the user presses one of the position buttons, this is written directly to the array, since no operation is required on this. The maximum increment each cycle for the cartesian position is set to $5mm$.

Similar to the joint impedance mode, there is a function for controlling the robot in cartesian impedance mode. This function takes the same arguments as the one for joint impedance, along with an additional parameter for the extra axis, joint 3, sent through the joint position array. That is, if the user presses the last jog button, controlling the 3^{rd} joint, an increment is made in the joint value for this, resulting in the zero space motion, where the position and rotation of the TCP is maintained as a constraint, but

the rest of the robot is moved since joint 3 is rotated.
The method for applying cartesian stiffness, damping and additional force/torque on the TCP are implemented in the same way as the joint impedance mode.
This concludes the description of the FRI. However, since the development of the application has been marked by a lot of trial and error programming, due to only having a superficial documentation for the FRI, the following will describe which aspects of the application could be made differently, since work with the FRI will most likely continue after the conclusion of this project.

12.4 Additional remarks

The following section will deal with the functions of the FRI, that is not implemented in this application, as well as improvements in the application. The reason for this is to aid the further work with the FRI, when the LWR is mounted on the mobile platform, since the FRI will most likely be utilized for control of the robot from the main computer on the mobile platform.
One aspect that immediately springs to mind is to implement a simple form of motion planning in the application, enabling the user to send the coordinates of a point in space, after which the robot moves to this point regardless of the distance to it. This can be implemented relatively simple, since the distance to the point can be calculated, and a number of points on the path can then be sent to the robot each cycle time. This has not been the aim of this exercise, however, but is estimated to be a simple implementation, depending on the kind of motion carried out (linear, point-to-point etc.).
Another part of the application, that could have been made differently, is the method for commanding the robot in the cartesian space, which has been described in the previous section. Instead of sending the angles through the FRI user variables, one of the two solutions of the angles could be determined from the rotation matrix, and used for the calculation of the new rotation matrix. Although the same increment in each of the two calculated solutions result in different rotation matrices, the orientation of the TCP would be the same, meaning it is up to the cartesian interpolator in the KRC to determine the shortest path, which is why this solution could work. This has not been tried, but could be sought implemented, to overcome some of the current limitations of the interaction between the KRL script and the application running on the remote PC.
This concludes the exercise of demonstrating the features of the FRI, since all the functions of this interface is available in the application. The source code and Visual Studio project files can be found on the enclosed CD in `Source codes\LWR-Projects\FRI-Panel`.

13

Measuring mass and center of mass of parts

In order to move the LWR by hand in Gravity Compensation mode, precise load data has to be entered in the KRC, in order for the controller to accurately estimate the externally applied torque in each joint. Along with the mass of the part, the position of the center of mass, and the moment of inertia around the three axes has to be entered. This makes it cumbersome to enter Gravity Compensation when the robot is handling parts, and more so when the load data is not known beforehand. This exercise aims to overcome this problem, by using the torque sensors to determine the load data. Another likely scenario of using the torque sensors to determine mass, is the case of weighing a box filled with parts, to determine the number of parts in the box. This of course necessitate some knowledge of the dimensions of the box. The following will only deal with the measurement on unknown parts.

Purpose: To determine if an accurate measurement of the mass and center of mass of an unknown part using the torque sensors of the LWR is feasible.

Objective: To measure the mass and center of mass of a known part, and test the precision of these measurements by comparing them to the actual data.

13.1 Test setup and method of measurement

A simple test setup has been created for this exercise, composed of a thick metal plate with three holes, and a pin in the plate used as a handle for the gripper. The aim of the test setup is to provide a method for displacing the center of mass (CM) with respect to the tool flange on the robot. This is effectively done by inserting the pin in one of three holes, where the first hole corresponds to a displacement of the center of mass along one axis (z), the second is along two axes (z, x) and the third is along all three axes. In order to further displace the center of mass in the z direction as well, a spacer can be placed between the pin and the plate, giving a total of 6 scenarios to measure. The test part is shown in Figure 13.1. The pin is hexagonal, since this is estimated to increase the 3-finger gripper's capability to fix the part even when the center of mass is displaced in the x or y direction, as opposed to a cylindrical pin.

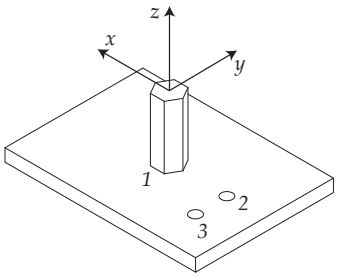


Figure 13.1: The part used for the measurements, shown with the position of the holes and the coordinate system for measurements

In order to determine the mass and center of mass, four measurements have to be carried out. All of these measurements will calculate the value from a specific joint torque. For the measurements of the CM, the mass of the part will be used as well, since the torque divided by the mass results in the distance to the CM. The measurements can be made in only three robot positions, since one position can be used to determine two values for the CM. The measurement strategy and values used for the measurements are shown in Figure 13.2. The joint torques and displacements correspond to equations 13.1 - 13.5 stated below. In the robot position shown in Figure 13.2(a), the weight F_g can be determined by the measured torque in joint 2 divided by the constant distance to the tool flange, after which the mass of the part can be determined:

$$F_g = \frac{T_2}{d} = \frac{T_2}{0,390[m]} \qquad \text{(Figure 13.2(a))} \tag{13.1}$$

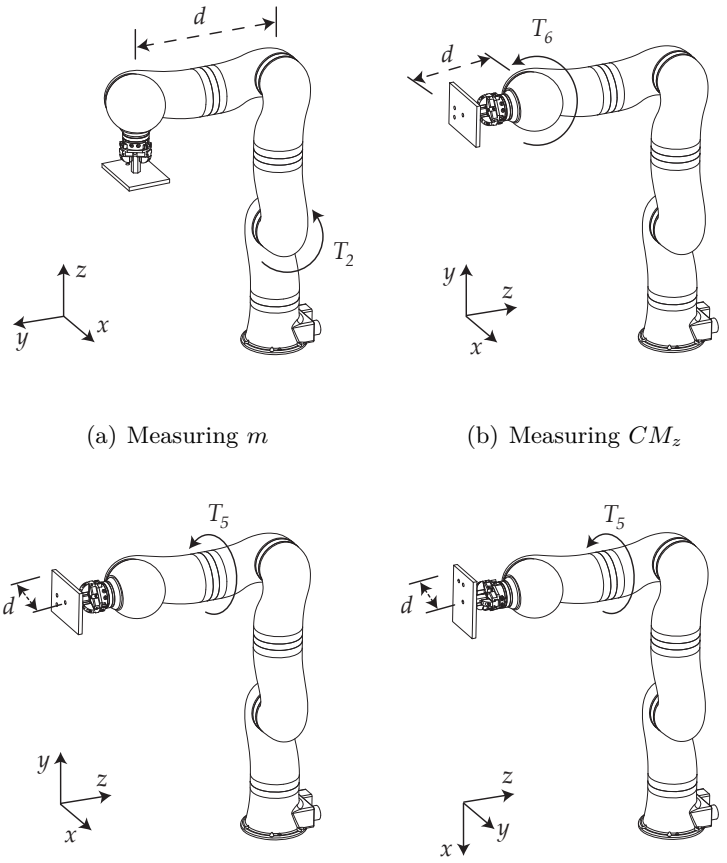
$$\Rightarrow m = \frac{F_g}{9,82^{m/s^2}} \tag{13.2}$$

Note that the torque in joint 4 could be used instead, but tests show that there is an error in the estimated external torque in this joint, after the robot is moved to the measurement position[1]. The error is not present in joint 2, which is why this is used.
In the second position shown in Figure 13.2(b) and 13.2(c), the center of mass in the z and x axis can be determined. The position of CM_z however has to corrected, since the calculated distance is from the center of joint 6 to the center of mass, and not from the origin of the coordinate system shown in Figure 13.1.

$$CM_z = 0,141m + d = 0,141m + \frac{T_6}{F_g} \qquad \text{(Figure 13.2(b))} \tag{13.3}$$

$$CM_x = d = \frac{T_5}{F_g} \qquad \text{(Figure 13.2(c))} \tag{13.4}$$

[1]The error is approximately $-1,2Nm$, and can be removed by briefly entering Gravity Compensation mode, and going back to Position Control mode



(a) Measuring m

(b) Measuring CM_z

(c) Measuring CM_x

(d) Measuring CM_y

Figure 13.2: Strategy for measuring the mass and coordinates of the center of mass in each axis. The shown coordinate system is the tool coordinate system, in which the measurements are made

Likewise, in the last position, the center of mass in the y direction can be determined:

$$CM_y \quad = d = \frac{T_5}{F_g} \qquad \text{(Figure 13.2(d))} \tag{13.5}$$

Since the method for determining the mass and center of mass has now been established, the implementation is made afterwards. This will not be described here, since the robot program is straightforward. The source code is however available on the enclosed CD in `Source codes\LWR projects\WeighPart.src` for further study.
In order to determine the accuracy of the measurements, it is however necessary to determine the actual mass and center of mass of the part in the various scenarios, which is described in the following.

13.2 Determining the actual values

In order to determine the center of mass of the object, the well known formula for the center of mass of a system is used, which is the mean position of each part $\mathbf{r}_i$, weighted by their mass m_i [15]:

$$\overrightarrow{R} = \frac{\sum m_i \overrightarrow{r_i}}{\sum m_i} \tag{13.6}$$

In the calculation, only the contribution from the pin, spacer and plate are used, and as such the bolts used to fasten the pin and spacer, are not calculated, along with the holes in the plate, since neither of these have a great impact on the location of the center of mass. Their weight contribution is however taken into consideration, by weighing these parts along with the pin, spacer and plate.
The calculation has been made in MATLAB, since it is straightforward to program a loop running through the various scenarios in this software. The MATLAB file created for this calculation is available on the CD, in `Source codes\MATLAB\Center-of-mass.m`. The results of the calculation and the measurements of weight is shown in Table 13.1, where the long and short pin refer to the scenarios with and without the spacer, respectively.

Scenario	**Point**	**Pin**	m **[g]**	CM_x **[mm]**	CM_y **[mm]**	CM_z **[mm]**
1S	1	Short	2023	0,0	0,0	-59,2
2S	2	Short	2023	43,0	0,0	-59,2
3S	3	Short	2023	43,0	-23,3	-59,2
1L	1	Long	2089	0,0	0,0	-85,7
2L	2	Long	2089	41,6	0,0	-85,7
3L	3	Long	2089	41,6	-22,6	-85,7

Table 13.1: Actual mass and center of mass of the 6 scenarios

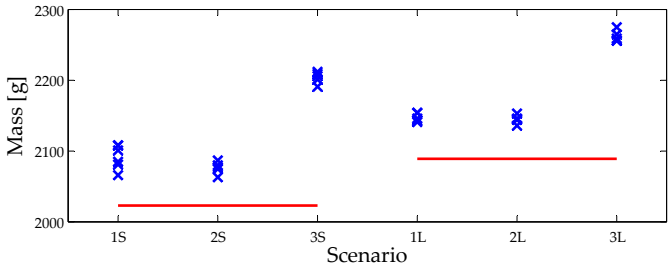
After the calculation of the actual values have been carried out, the measurements are carried out. The measurements and an evaluation of the accuracy of the measurements are described in the following section.

13.3 Accuracy of the measurements

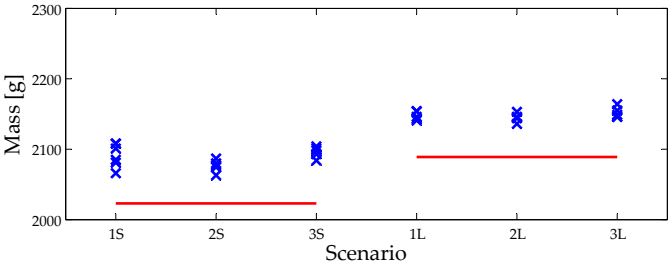
To determine the accuracy of the measurements, 5 measurements have been carried out in each of the scenarios. Furthermore, a zero point measurement without a part is carried out for both the long and short pin[2] giving a total of 40 measure points. The

[2]Since there are different starting positions for the long and short pin

measure data is available on the CD in `Results\WeighPart.xlsx` When looking at the measurements of mass, shown in Figure 13.3(a), it is obvious that there is a significant error in the measured value. This error does not correspond perfectly with the zero point measurements, the mean of which are $510g$ and $549g$ for the short and long pin, respectively. The zero point error does however have an effect, since the measured masses are all greater than the actual. The mass measured in point 3 have a greater error than in the other points. Upon inspecting the test setup and method of measurement, this change in the error is obvious, since the center of mass in this point is shifted in the direction of d in Figure 13.2(a). When adding the distance from point 3 to point 1 in this direction to the calculation of the mass, the error is approximately the same as for the other points. The corrected values are shown in Figure 13.3(b).



(a) Measured values



(b) Corrected values

Figure 13.3: Measurements of mass for the various scenarios, and the corrected values - the line shows the actual mass of the part

A way to compensate for this distance, and a more appropriate way to measure truly unknown parts, would be to set joints 5 and 6 to free motion, by setting the stiffness to 0 $^{Nm}/_{rad}$ in joint impedance mode when measuring the mass. This would result in the center of mass settling on the vertical axis beneath the tool flange, which in turn would make the measurement of mass accurate for all scenarios. It is however hard to say if

this is possible, since the shift in the center of mass in this case should overcome the friction in the joints.
Since the position of the center of mass is calculated from the mass of the object, the aforementioned errors are persistent in these values as well. The measured positions are shown in Figure 13.4. It is obvious that if the mass was determined correctly, the measurements of the center of mass would only suffer from the inaccuracy of the torque sensors in each measurement. When the mass is measured inaccurately, the results for the center of mass becomes the combination of the error from two torque sensors, instead of just one.

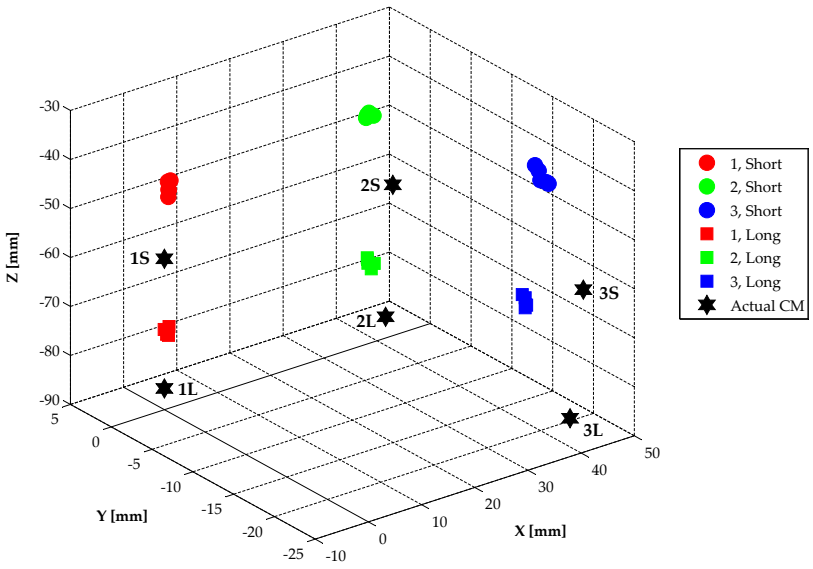


Figure 13.4: Measurements of the centers of mass compared to the actual values

The mean of the errors of the measurements compared to the calculations are given in Table 13.2. The great error in CM_z can be explained by looking at the zero point measurements. This reveal that while the mean of the torques used to determine CM_x and CM_y are less than $0,05Nm$ in the zero position, the measurements of torque T_6, which is used to determine CM_z, averages $0,11Nm$. This error is persistent, even if the torque sensors are calibrated before the measurements. One plausible explanation of this is slightly incorrect payload data of the 3-finger gripper, since this should be entered accurately to enable the controller to neglect the contribution from the tool when performing measurements on the parts.

	CM_x [mm]	CM_y [mm]	CM_z [mm]
1S	-2,8	-2,3	17,8
2S	-3,8	0,5	15,1
3S	-6,5	0,9	24,5
1L	-1,8	-1,5	13,7
2L	-3,1	-0,1	12,3
3L	-5,1	2,2	23,5

Table 13.2: The mean of the errors in measurements of center of mass

Upon inspecting the results described above, it is determined that using the torque sensors to accurately measure the mass and center of mass of unknown objects, can not be done with a satisfactory precision, at least not using the method described above. A zero point calibration has been attempted, by moving the robot to the measurement position before picking up the parts, and saving the measured torques. These torques are then subtracted from the measurements when these are performed on the part. This method does however not yield promising results, since the error in both mass and center of mass is approximately the same.
Apart from this, the moment of inertia is not measured, which is required to move the robot in Gravity Compensation mode[3]. Due to these two facts, suggestions on how to improve the measurements and incorporate a measurement of the moment of inertia are described in the following.

13.4 Further work on the measurements

Due to the precision of the torque sensors, one immediate implementation that springs to mind, is to compensate for the variance in these measurements. One way to do this, is to save the measured external torque in each joint several times, and use the mean of these measurements. The time between the measurements does not have to be great, and as such is not estimated to have a great effect on the overall time of the program.
Another immediate improvement of the method is to make sure the center of mass is centered directly in a neutral position on the vertical axis of the tool flange, when measuring the mass, methods of which are also mentioned above. It is however uncertain if the displacement of the center of mass from the vertical axis is enough to overcome the friction in the joints, thus enabling the part to center itself in the neutral position. Another method could be to incorporate a function in the program, that monitors the

[3]Not exactly required, but entering $0kg \cdot m^2$ for the moment of inertia results in accelerated rotation when even a small external torque is applied to the tool

estimated external torques, and accordingly rotates joints 5 and 6, in order to minimize the external torque, thus resulting in a neutral position.
The method described above could be utilized to determine the mass and center of mass of the tool, in this case the 3-finger gripper. Although the mass of the tool can be determined fairly accurately, to determine the center of mass analytically would be difficult. Therefore, an estimate of the center of mass was entered when the tool was calibrated - it is however uncertain if the method described above would yield a more accurate result than the estimation. The moment of inertia is also still to be determined. One way to do this, is by the definition of torque as the rotational analogue of Newton's second law of motion, $\tau = I\alpha$ [15]. Since the acceleration can be specified for a motion in KRL, the moment of inertia can be determined from the measured external torque. This is however not exactly feasible in KRL, since this is a very basic programming language. Instead the FRI could be used, since the user could write a simple program performing the desired motion and collecting the data for the calculation.
In order to test the effect of the mentioned improvements, some implementations has been carried out, i.e. measuring the torque several times and taking the mean of the measurements, and setting the robot in joint impedance mode before determining the mass. However, tests show that these implementations have nearly no effect on the accuracy of the measurement. It does, however, yield more consistent results, most likely due to the implementation of several measurements. This exercise as such forms the basis of further work with establishing a task for measuring unknown parts, which in turn has to be carried out. It is however estimated that the mentioned implementations will have a significant effect on the precision of the measurements, though a precise measurement is still not feasible.

14 Conclusion

The following chapter describes how the thesis statement has been fulfilled, as well as describing the work carried out in the project. Like the project at hand, the following is divided into the two major parts of the project.

Design of the Little Helper Plus

The design of the original Little Helper has been analyzed, in order to form a basis for the design of the Little Helper Plus, and this analysis has revealed the needs for a more easily reconfigurable solution. Furthermore, it is investigated what major changes in the hardware will be necessary, mainly limited to a new robot, the KUKA LWR, and controller. It is decided to acquire an identical platform to use on the LHP, since there is already experience in using this platform at the department.
The requirements specification for the configuration of the LHP sets up the quantitative demands regarding reach and dimensions of handled parts, which are similar to that of the original Little Helper, to ensure there is no loss of functionality except where this makes sense, e.g. by removing the distance sensor on the tool, since this is seldom used. This has been done to make sure the functionality of the Little Helper Plus is greater than that of the Little Helper. Furthermore, it is decided to remove the pneumatic system, i.e. the air reservoir and compressor, since these are both large components, and it is difficult to fit these in the LHP, due to the larger controller for the LWR. This however requires an electrically actuated tool changer.
Since no compact, electrical tool changer is available, this is designed to ensure the tool changing capability of the Little Helper Plus. A number of options has been investigated, primarily regarding electric actuation of a commercially available tool changer. The choice has been made to use the same pneumatic tool changer of the Little Helper, the Schunk SWS-011, and instead to actuate this using a small, linear actuator on the Little Helper Plus. A housing for this is designed, and the control of the actuator is programmed, resulting in a fully working electric tool changer.
When an overview of the required components on the LHP has been established, the redesign is carried out. This has been done in CAD, by placing the larger components top of the platform and afterwards designing the chassis and fasteners and placing the

smaller components. For the chassis, it is chosen to use standard extruded aluminum profiles, since these are easy to reconfigure and to attach extra hardware to, should the need arise. The LWR is mounted on the chassis in such a way that it can reach the parts being transported on the worktable of the LHP, and the reach of the LWR is determined to exceed the quantitative demands. Furthermore, the cover plates protecting the hardware from dust and foreign objects are designed to provide easy access and visual inspection of the various components, since this was not implemented on the original Little Helper. Finally, a schematic of the power connections for the overall system is established, after the which the design is complete. The Little Helper Plus has not been built by the end of this project, however, but is planned to be completed in the beginning of June 2011.

Capabilities of the KUKA LWR

The LWR is mounted in a temporary location, to investigate the added features of the LWR in the context of the mobile manipulator. Initially, the method of jogging the robot by hand for teaching new routines are investigated. This method seemingly provides an easier method for online programming the robot, thus adding to the flexibility of the LHP. However, tests show that programming the robot in this way, at least through the KUKA HMI, is in fact not faster than programming using the teach pendant to jog the robot. Furthermore, it is also harder to ensure an accurate position and rotation of the TCP when moving the robot by hand. It is however possible that another interface could be developed, which would utilize this function better, for example through the Fast Research Interface.

Since the positioning tolerances of the platform is not adequate for handling parts on a workstation, a calibration has been developed on the LWR, using the torque sensors to touch points on the edges of a workstation. Tests show that this calibration is fairly accurate, with a repeat accuracy of $\pm 0,5mm$ and $0,25°$, tested with the workstation within the tolerances of the platform. This is more accurate than the fast version of the vision calibration currently used on LH. The accuracy is determined to be adequate for handling parts, and to be a supplement to the two forms of vision calibration, since this method both in time consumption and accuracy falls between these two methods.

The classic peg in hole task have been programmed on the LWR, in both the position control mode and the cartesian impedance control mode. It is obvious that when using regular position control, it is hard to insert the peg in the hole, requiring tight tolerances on position ($\pm 0,5mm$) and even tighter on the rotation of the tool relative to the hole ($\pm 0,5°$). This is much easier using the cartesian impedance controller mode since the

peg is conically shaped, however still requiring small tolerances on the angle. Finally, an evaluation on the measures to improve the task is presented. This concludes that, instead of letting the peg slide into the hole, a search algorithm should be implemented to determine the position and rotation of the hole, by measuring the reaction forces acting on the tip of the peg. Furthermore, a method for inserting a sharp-edged peg in a ditto hole is suggested.
In order to provide a method for communicating with the robot through an ethernet connection, the use of the Fast Research Interface is investigated. Since this is a fairly new KUKA Technology Package, some difficulties in using the interface and commanding the robot have presented themselves. Therefore, an initial console application, outputting the joint torques of the robot on the remote PC, has been developed. After this initial exercise, a Windows GUI application has been developed, in order to make a full demonstration of the FRI. This application enables the user to monitor all system variables of the FRI: The connection statistics, robot data (such as position and torque) and the user variables. Furthermore, a method of jogging the robot through the FRI is implemented, by using the *Command Mode* of the interface. This enables realtime control of the robot with a cycle time as low as $1ms$, though only the default cycle time of $20ms$ has been tested. This application should form the basis of further work with the FRI, since it provides methods for overcoming the various pitfalls that have been encountered when using the FRI.
Finally, the precision of the torque sensors is tested, by developing a method for weighing unknown parts and determining the center of mass of these. It is however apparent that the torque sensors are not very accurate, since the precision of these measurements on mass are approximately within $80g$, and the center of mass within $5mm$. Therefore methods of improving the measurements have been suggested, though preliminary tests of these measures show little improvement in the accuracy.
The overall conclusion of this project is that the KUKA LWR in most ways contribute to the function of the Little Helper, mainly by increasing the flexibility of the overall system. This is mostly due to the Fast Research Interface, which enables control of the robot with a high sample rate, and the compliance controller modes, enabling the robot arm to behave like a virtual spring-damper system. Though some scenarios have been investigated in this project, there are a lot of possibilities for further work with using the LHP in general and the LWR in particular.

Bibliography

[1] Statistical Department of IFR, "Press release charts", http://www.worldrobotics.org/downloads/PR_2010-09-14_charts.pdf, September 2010.

[2] Simon Bøgh, Mads Hvilshøj, Christian Myrhøj, and Jakob Stepping, "Fremtidens produktionsmedarbejder - udvikling af mobilrobotten lille hjælper", Master's thesis, Aalborg University, June 2008.

[3] Rainer Bischoff, "From research to products: The development of the kuka lightweight robot", Presentation at the 40th International Symposium on Robotics (ISR), March 2009.

[4] ATI Industrial Automation, "Electrical tool changer for heavy automation", http://www.ati-ia.com/Library/documents/ElectricQC.pdf, October 2010.

[5] Ole Simonsen, "Elektrisk væktøjsskifter", email correspondence, October 10, 2010.

[6] National Semiconductor, *DS2003 High Current/Voltage Darlington Driver*, March 12, 2010, http://www.national.com/ds/DS/DS2003.pdf.

[7] Adel S. Sedra and Kenneth C. Smith, *Microelectronic Circuits*, Oxford University Press, Inc., fifth edition, 2004.

[8] HepcoMotion, *MCS aluminium frame and machine construction system icluding MFS fencing system*, March 8, 2009, http://www.hepcomotion.com/en/literature-mcs-machine-construction-system-pg-16-get-31.

[9] KUKA Roboter GmbH., *Light Weight Robot 4+ Assembly Instructions*, July 6, 2010.

[10] Paolo Robuffo Giordano, Andreas Stemmer, Klaus Arbter, and Alin Albu-Schäffer, "Robotic assembly of complex planar parts: An experimental evaluation", in *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2008.

[11] KUKA Roboter GmbH, *Lightweight Robot 4+ Operating Instructions*, February 25, 2011.

[12] Mathias Schöpfer, Florian Schmidt, Michael Pardowitz, and Helge Ritter, "Open source real-time control software for the kuka light weight robot", in *Proceedings of the 8th World Congress on Intelligent Control and Automation*, 2010.

[13] KUKA Roboter GmbH, *KUKA.FastResearchInterface 1.0*, April 14, 2011.

[14] John J. Craig, *Introduction to Robotics - Mechanics and Control*, Pearson Education, Inc., third edition, 2005.

[15] Raymond A. Serway and John W. Jewett Jr., *Physics for Scientists and Engineers with Modern Physics*, Brooks/Cole - Thomson Learning, sixth edition, 2004.

[16] KUKA Roboter GmbH., *KRC2lr Operating Instructions*, Februrary 2, 2011.

[17] Alexander Rommel, "Fri questions", Email correspondence, March 16, 2011.

[18] Various, "Easiest i/o setup on krc2lr", Forum on the internet, March 28, 2011, `http://www.robot-forum.com/robotforum/kuka_robot_forum/easiest_io_setup_on_krc2lr-t6997.0.html`.

[19] KUKA Robot Group, *DeviceNet for KR C2 edition2005 and KR C2 sr*, December 13, 2007.

Appendix

A

Case: Vision-controlled robot playing NIM

This chapter describes the implementation of machine vision on a robot cell, and the creation of a game of NIM where a user can play against the robot. The case is done as preparation for instructing a Ph.D. course in Robot Vision, where the Ph.D. students are given the task to complete this very case.

Purpose: To gain an understanding of machine vision routines and robotics in general, and to implement this understanding in a real-world scenario, to give satisfactory guidance at the aforementioned Ph.D. course.

Objective: To have a fully working NIM game, where a user can play against a vision-controlled robot.

A.1 Overview

The following provides the reader with an overview of the setup of this case, as well as the game of NIM. The equipment used for this case is an available robot cell at the Department of Mechanical and Manufacturing Engineering at Aalborg University. NIM is chosen as the game, since it has a fairly simple logic to maximize the potential of winning, and is an easy-to-manipulate game, meaning the robot can easily move the pieces.

A.1.1 The game of NIM

The game of NIM is quite simple; it consists of n pieces, lying on a surface. Each of the two players take turn in removing between 1 and a maximum of k pieces. The game can be played in two ways; usually it is played as a *misère* game, where the player who takes the last piece loses, but can also be played as a normal type game, where the person to make the last move (i.e. take the last piece) wins. The game can be played with any number of pins n and any k, with k obviously being much smaller than n.
The strategy of the game is fairly simple, as a good player can make sure to remove

n modulo $k+1$[1] in a normal play game, obviously still making a legal move, i.e. not removing 0 pins. This always leaves one extra piece than a multiple of k for the opponent to remove, effectively making the first player the winner. In a misère game, the optimal strategy is to remove one piece less than n modulo $k+1$.
If both players follow this strategy, the player to start always loses a misère game, and always wins a normal play game, except when n modulo $k+1$ initially is 0 for a normal type game or 1 for a misère game.
In this case, the game is played as a misère game, with $n = 13$ and $k = 3$.

A.1.2 Setup

As mentioned, the setup for this case is an available robot cell at Aalborg University. The cell is purely used for educational purposes like this case. A representation of the robot cell with the equipment used for this case can be seen in Figure A.1.

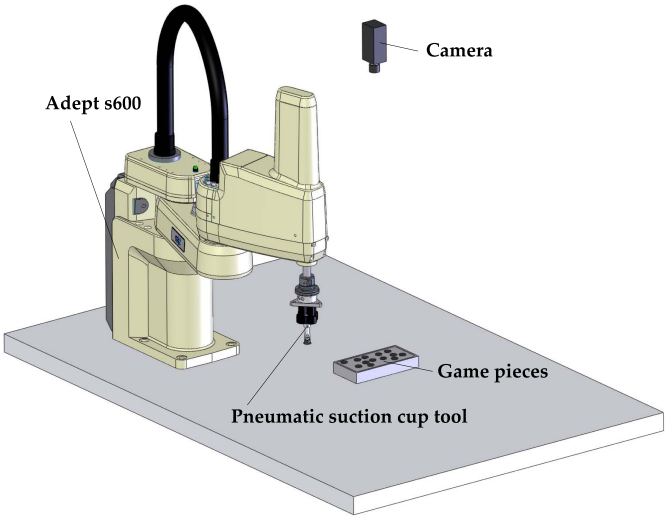


Figure A.1: The robot cell used for the NIM game

The purpose of this case is to establish communication between the camera and the robot, effectively enabling the robot to see the individual pieces, and remove the required amount. For safety purposes, the robot should also remove the desired number of pieces for the user, so there is no risk of harming the human player. In the following, the manipulation of the game and the image processing will be presented, followed by a description of the communication interface and the overall structure of the game.

[1]Modulo: In this case, the remainder on division of n by $k+1$

A.2 Manipulation

The manipulator of the cell is an Adept Cobra s600 SCARA 4-axis robot arm. This is a fast moving robot, usually used for pick-and-place operations and material handling. It is controlled by an Adept SmartController, connected to a remote PC through an ethernet connection. On the computer, the user can write a program in the V+ language, native to Adept products, and run it directly on the controller. The language handles everything related to the controller, e.g. robot movements, communications and I/O operations.
To begin with, an empty script has been made, with nothing but an initialization of a TCP/IP server, running on the controller, followed by an empty loop, running until the program is terminated. This will enable any hardware or software capable of sending TCP/IP commands, to send instructions directly to the controller.

A.3 Vision

The camera is attached directly to the computer, through an IEEE 1394-connection, and image processing is handled by the computer through the Vision Builder for Automated Inspection (VBAI) software by National Instruments. This program can handle image acquisition, enhancement and calibration, parts inspection through built-in algorithms, as well as setting up complex pass/fail criteria for parts inspection.
Before using software for image enhancement, the user should make sure there are optimal conditions for the clearest possible image acquisition, this primarily regarding lighting. Good lighting will result in a clear image being captured, reducing the need for image enhancement, and thereby reducing the processing time. In this case, it is also important to avoid shadows, since the purpose of the camera is to determine the location and amount of the game pieces, which are black on a light surface. For this case, three light bars are used, to provide adequate lighting and avoid shadows cast by the game board.
In the following, the image calibration and enhancement routines will be described.

A.3.1 Calibration

The image calibration is done to precisely obtain a relationship between the dimensions of the game area and the image, usually in terms of $^{mm}/_{pixel}$, and to place the reference coordinate system for the camera $\{C\}$, used to determine the coordinates of each game piece.

The calibration in this case is done with a printed grid of dots with 8 mm spacing, which is placed in the same height as the game surface. In VBAI, the calibration routine is started, and the user can directly select a method based on the grid of dots. Afterwards, the user can adjust the threshold of the image to correctly identify all dots, and enter the physical distance in mm between the dots. The last step is to indicate the axes of the desired $\{C\}$ on the image. In this case, the axes of $\{C\}$ is chosen to correspond with the axes of the manipulator's coordinate system $\{M\}$, with a simple translation in the xy plane. The origin of $\{C\}$ is chosen arbitrarily, and afterwards taught to the robot, making the translation of the coordinate of each game piece as easy as possible.

A.3.2 Image enhancement

Although the image calibration routine has a built-in threshold setting, further enhancement of the image is required to accurately determine the amount and locations of the game pieces. VBAI has a built-in module called Vision Assistant, which runs a sequence of user-specified image enhancement routines on a specified region of interest (ROI) of the image. In the following, only the routines used in this case are described. The steps can be seen in Figure A.2.

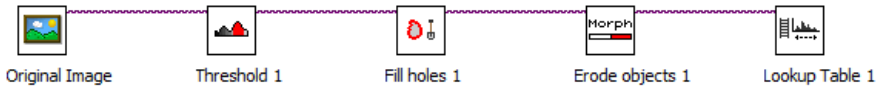
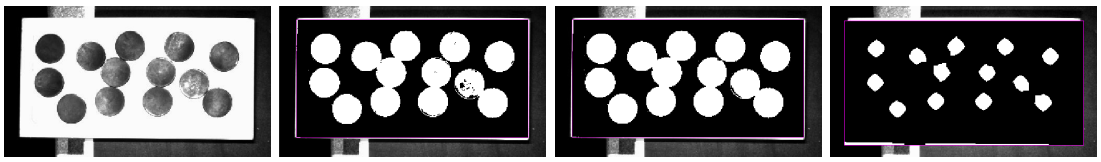


Figure A.2: The image enhancement steps carried out

After the Vision Assistant is run, the game pieces are easily recognized by the computer. Note that the image enhancement is only applied to the specified ROI. The direct effect of each step in the Vision Assistant is shown in Figure A.3.



(a) Acquired image (b) Thresholding (c) Fill holes (d) Erode objects

Figure A.3: Image enhancement of the acquired image

After the image enhancement procedure, it is easy for the Detect Objects algorithm to detect the number of game pieces, and the xy location of each game piece.

A.4 Overall structure of the game

VBAI is used for everything related to the vision system and the game logics. The only thing handled by the SmartController is a continually running while-loop, waiting for input from a TCP/IP client, that tells the robot what to do (in this case VBAI). This input is specified in the V+ script, and in this case is a string. This effectively means that nearly the whole program is made in VBAI, though both programs must be running for the game to work.

A.4.1 Communicating with the robot

Basically, the V+ script can run at all times, even when programming is done in the VBAI program. This is due to the nature of the V+ script. This is basically a TCP/IP server initialization, followed by a while-loop based on the value of a variable that always evaluates as true, i.e. making the while loop run continuously. In this while-loop, a handshake procedure first ensures the connection is established, if it is the first time the client connects.

After the handshake has been made, the server waits for a specific type of command to be sent by the client. The string must however follow the form established in the V+ script. This form is established to be:

$$tag,x,y,z,yaw,pitch,roll,approach,depart$$

where $x, y, z, yaw, pitch, roll$ are the cartesian coordinates and rotational angles the robot should operate in, $approach, depart$ is the distance from the operating point the robot approach from/depart to, respectively. tag is used to initialize a certain set of procedures on the robot, e.g. pick up objects, place objects, go to initial position etc. In this case only one tag that picks up pieces and discards them at a fixed location is used, but more tags have been used for test purposes.

After the SmartController receives the string from the client, the string is decomposed to an array and stored in a set of variables for later use. After this, the case structure is initialized. Only the case that picks up objects is described in the following. The code is presented in the following, with inserted comments after semicolon (;).

```
1  ; Save received world coordinates in a single location variable:
2  SET pickup_vision = TRANS(my_x,my_y,my_z,my_wx,my_wy,my_wz)
3  ; Instruct the robot to move to my_approach mm above the received coordinate:
4  APPRO pickup_vision, my_approach
```

```
5  ; Monitor for errors, and execute program error.trap if errors occur:
6  REACTE error.trap
7  ; Move in a straight line to the received coordinate:
8  MOVES pickup_vision
9  ; Halt until the robot motion is completed:
10 BREAK
11 ; Turn on suction for the tool:
12 SIGNAL 3001
13 ; Move my_depart mm vertically away from the coordinate:
14 DEPART my_depart
15 BREAK
16 ; Move to the specified dropoff location:
17 MOVE #dropoff
18 BREAK
19 ; Turn off suction for the tool:
20 SIGNAL -3001
21 ; Send string back to client with result of operation, and type result to terminal
22 IF error == 0 THEN
23    WRITE (lun, handle) "OK"
24    TYPE "Sent OK to VBAI"
25  ELSE
26    WRITE (lun, handle) "9999"
27    TYPE "::error"
28 END
29 error = 0
30 BREAK
31 REACTE error.trap
```

The use of the BREAK command ensures the robot completes the motion before sending OK back to the client. This effectively ensures the client waits for the robot to complete the case, before the client continues.
With the tag described above, it is therefore possible to send the tag followed by a set of coordinates through the TCP/IP connection. Upon receiving this string, the robot moves to the specified location, picks up one game piece and drops it at a specified location, before sending the "OK" string back to the TCP/IP client.

A.4.2 Picking up several pieces

Picking up a single game piece is of course not sufficient, so a loop is introduced in the VBAI script to pick up the desired amount of pieces. The location of each game piece is determined by the previously mentioned Detect Objects algorithm, and stored individually in temporary variables for each piece. The Index Measurements module is then introduced in the VBAI routine that picks up objects, allowing operations on the ith object detected, in this case extracting the location of the game piece.
The location of the ith piece is then transformed by adding the taught xy coordinates of

the origin of $\{C\}$, relatively to the robot. Finally, the TCP/IP IO module is introduced, and set up to send the previously mentioned string, and wait for the "OK" string in return before proceeding. It is however important to flush the TCP/IP buffer before sending and receiving any commands, but this is easily done in the VBAI module.
The final step is to increment the loop counter, which is reset to 1 before the loop state. This is used as a condition for when the loop should exit, since the counter expresses the number of objects removed in the current turn.

A.4.3 Game structure

In the following, the overall game structure is presented. The sub-structures have been described in the previous sections. The overall structure can be seen in Figure A.4, where c is the counter, n is the total number of pins remaining and p is the number of pins to remove the current turn. Note that the referenced Turn variable updates whenever the desired amount of pieces is removed, either by the system or the user.

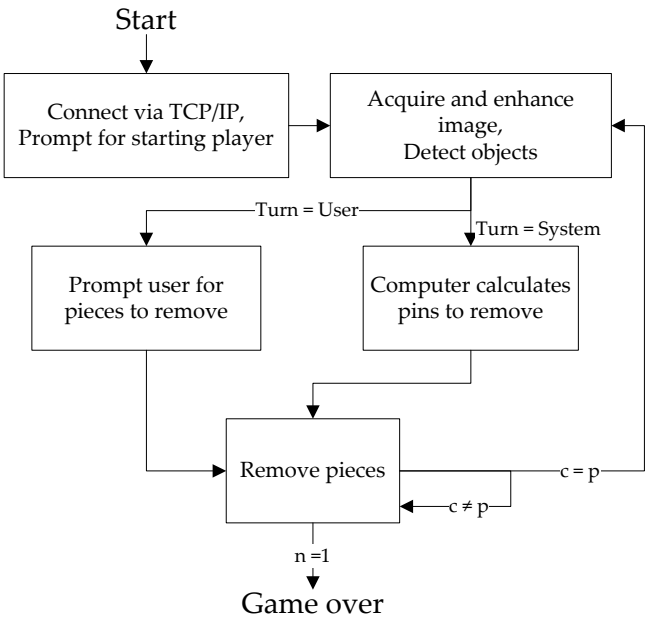


Figure A.4: Structure of the VBAI script; c is the counter, n is the total number of pins remaining and p is the number of pins to remove the current turn

The game logic is implemented as described in Section A.1.1, meaning the computer calculates the number of pins to remove for it to win, so the robot always wins if the user starts.

This concludes the case, since the only thing left is to add interfaces for the user, displaying the state of the game, which will not be covered here.

B

FoodPharmaTech '10

During the 2nd to 4th of November 2010, Aalborg University presented the Little Helper at the fair FoodPharmaTech, regarding in particular manufacturing equipment for the food industry, at Messecenter Herning. This chapter describes the preliminary work of preparing the stand and the fair in general.
RoboCluster is a network of different industrial and academic participants in the field of robotics and automation in Denmark. To profile itself, RoboCluster appeared at a stand at FoodPharmaTech, and Aalborg University was subsequentially contacted by RoboCluster, regarding having LH at their stand. Besides LH, two other installations where presented; a manipulator from Universal Robots, demonstrating the principle of vertical farming, and a PLC-controlled palletizing robot by MRN Robotics. The following will only go into detail about the installation demonstrating Little Helper's capabilities.

B.1 Description of the setup

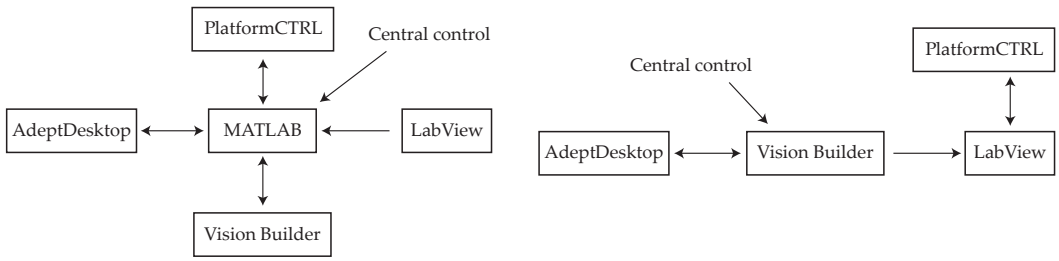
To demonstrate the capabilities of Little Helper as a flexible robot, a setup to demonstrate some of the key features of LH in a single case. To do just that, it was decided to go with the following setup at the stand, focusing on a game of NIM:

1. A visitor starts the game, and selects which player to start, the visitor or Little Helper
2. Little Helper moves to the station, where the NIM game pieces are laid out
3. The visitor and Little Helper take turns in removing pieces, as described in chapter A
4. After the game is over, Little Helper moves to one of two stations, containing a prize for the winning visitor, and one for the losing, respectively
5. Little Helper picks up the prize, and delivers it to the user at a fixed station

This scenario effectively demonstrates the aspects of vision, classification, pick and place, and transportation of parts.

B.1.1 Preparation at AAU

Most of the preparation was done by making a mockup of the stand in the workshop at AAU. This mockup consisted of four stations with separate purposes, as described above. Each workstation is specified in the software PltfGUI, which is used as a frontend for editing maps and roadmaps for the platform, and controlling the platform directly, without third-party applications.
A central problem in preparing the stand was that the GUI for instructing new work routines is developed in MATLAB, which has to be version 2007 to communicate with LabView, which controls all minor hardware components. The site license for the 2007 version of MATLAB, however, has expired before the preparation of the stand was begun. Therefore, the central routine could not be programmed in the MATLAB GUI, and was instead defined in Vision Builder, as LabView VIs[1] can be executed directly from this software. The developed control of Little Helper and the control used in this application can be seen in Figure B.1.



(a) Original control (b) Control used at the fair

Figure B.1: Original and modified control of Little Helper

One benefit of using Vision Builder as the main control is that all of the NIM game described in chapter A can be used again, so the only development is the routine of driving to the stations. One problem, though, is that the calibration routines are developed in MATLAB, so there is no way to calibrate the manipulator when it arrives to a station, which is required to obtain a greater precision than that of the platform.
In order to gain the required precision, a fixed transformation can be made, since the height of the game table is independent of the location of the platform, and the transformation between the camera and tool is constant. When these transformations are

[1]Virtual Instruments, the native file format for LabView files

known, it is fairly easy to pick up the game pieces from the locations calculated from the acquired picture. The location of the prize is taught as a fixed location, since there is only one prize at each station, and the prizes are large enough to be picked up in spite of the poor precision of the platform.
Since control of the platform and the minor hardware components have already been developed in LabView, these VIs are called from Vision Builder when needed. The VIs used in this application do the following:

- Turn on or off warning lights on the platform
- Move platform to specific station
- Turn on or off lights for the camera
- Control the focus and aperture of the camera lens

Each of these VIs is called when needed, and requires different inputs, usually in the form of strings or integers, e.g. the VI that moves the platform to a station requires the station's name as an argument, and the lens control requires the values for focus and aperture of the lens.
The use of LabView VIs greatly reduces the work effort in creating routines, and the control of the game is pretty straight-forward to make in Vision Builder, since it has already been programmed for the Ph.D. course.
The whole routine is then programmed, albeit only in a mock environment. The next step is to create the setup at the fair, which is covered in the following section.

B.1.2 Preparation at the fair

Upon arriving at the fair, a new map and roadmap is created. The map describes the environment the platform operates in by edges and areas of operation, and is used by the platform for localization. The roadmap is used when the platform is moving, and contains information about work stations and the paths between them. Both of these are fairly simple to create in the software PltfGUI. After both maps are created, the vision routine has to be recalibrated, due to the different lighting at the fair, and the height of the tables containing prizes has to be re-teached. Apart from this, no other adjustments has to be done, and the game is up and running.

B.2 Modifications during the fair

During the fair, it became obvious that the chosen setup did not attract the visitors in a satisfactory manner, likely due to the need for user interaction to make LH move. A better way to attract visitors to a stand regarding robots, is to have the robots move continuously. Due to this observation, another routine was created at the fair. The routine was simply to have Little Helper move outside the designated area, and move amongst the visitors at the fair. This routine demonstrated an important aspect of the platform, since it successfully avoided the visitors standing in its way, even in complex situations, where Little Helper was nearly completely surrounded. The routine did not, however, demonstrate any of the aspects demonstrated in the NIM game, since a necessary safety precaution was to disable the power to the robot arm, when the platform was out of the designated area. The evasion of interfering visitors and the appearance of the robot in the visitor's domain, however, did however attract a lot of attention.

C

Setup of the LWR

Since the new platform, which the LWR is to be mounted on, is not delivered by the end of this project, the LWR has to be temporarily mounted in order to use it for Part II of this project. This gives one benefit, however, that the actual installation and initial configuration can take place before the actual construction of the new configuration of the Little Helper.
Due to the lack of the platform, the need for a temporary solution and the ability to move the installation on short notice (due to a major upcoming renovation of the laboratory), the robot is temporarily mounted on a table, so the table functions as both a mounting place and a workstation for the robot. The table is mounted on a pallet, to enable the mobility of the installation, and the controller and various components are placed on the pallet underneath the table, in order to have the largest possible workplace on the table. The installation is shown in Figure C.1.

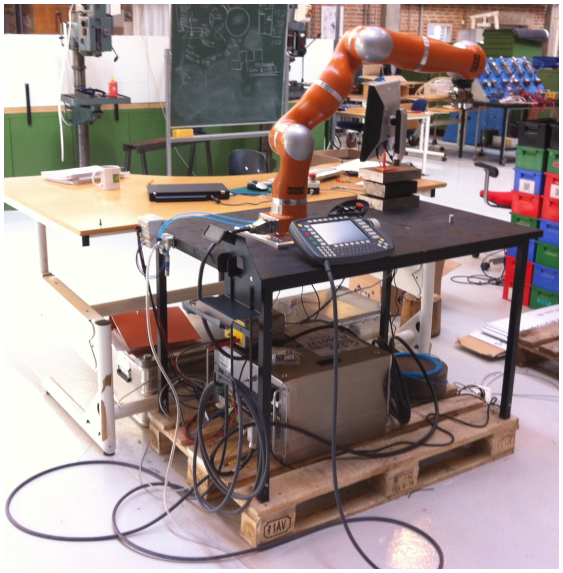


Figure C.1: The temporary installation of the LWR

C.1 Startup and configuration of connectors

The KUKA LWR does not work out of the box without some configuration, which is why a visit from KUKA technicians was included in the purchase of the first robot, and this initial configuration will not be described here. There is some aftermath from this preliminary configuration though, which are some necessary installation steps to have a fully operational robot. Mainly, this is limited to the construction of two plugs, that are to be inserted in the KUKA controller, the X15 and the X11. There are two reasons for this; that KUKA is not allowed to supply these two connectors along with the robot, since both interfaces are used for external safety equipment, and that the configuration of both plugs are almost different for each installation, depending on the safety equipment used in the installation.

The X15 connector is a power limitation connection for the robot, which reduces the power of the robot to $80W$. This connector can be wired to external safety equipment, such as light barriers or gate switches, and can thus be used as a safety measure [16]. For this application, however, the connector is simply wired to a switch, so the operator can disable or enable the power limitation as he sees fit. The wiring of the X15 interface is shown in Figure C.2. Notice that the used switch is dual channel, since both connections has to be switched at the same time.

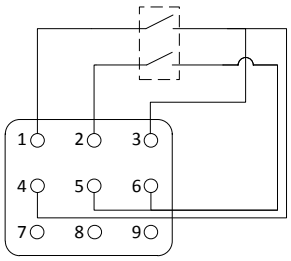


Figure C.2: The wiring of interface X15

The X11 connector is used for external emergency stops, either directly wired to the interface or linked together by a higher-level controller. For this application, a single emergency stop is wired directly to the interface, so the operator can activate the emergency stop from a safe distance of the robot. The emergency stop has to be dual channel, in order to ensure the function in case one of the connections fail [16]. The wiring of interface X11 can be seen in Figure C.3, where only the relevant pins for this particular wiring are designated. Pin numbers 7 and 26 are for channel A and B for the emergency stop, and must be connected to test outputs A and B, respectively pins 1 and 20. The

remaining connections are for various other safety measures, which are not needed in this application, and as such are bridged directly.

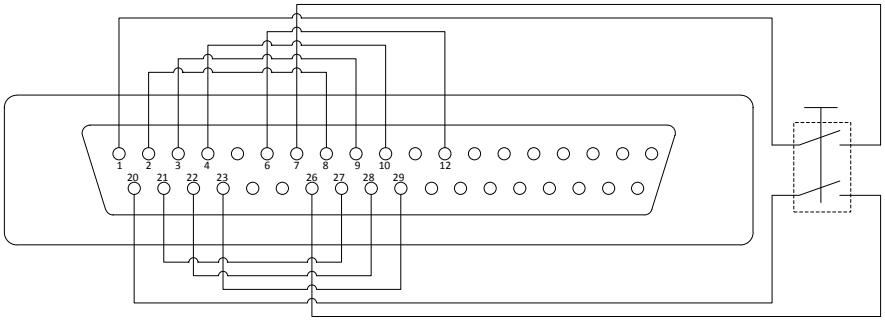


Figure C.3: The wiring of interface X11

After the wiring of these two connectors, the robot arm is fully functional, though lacking tools and I/O possibilities.

C.2 Connecting to end-effector equipment

The LWR, that is to be placed on the mobile platform, is delivered in the *Electrical Energy Supply system* configuration. This configuration has 14 electrical connections through the arm, along with a $100Mb$ ethernet connection, from the base to the end-effector. Connector plugs are delivered with the robot, but have to be soldered to the devices that need a connection through the arm. In order to quickly change which connections are run through the arm, a connector box is made, containing the electrical and ethernet connection. This box has 14 screw terminals and a female ethernet connector, all wired to their respective connectors on the base of the robot. The connector box is shown in Figure C.4.

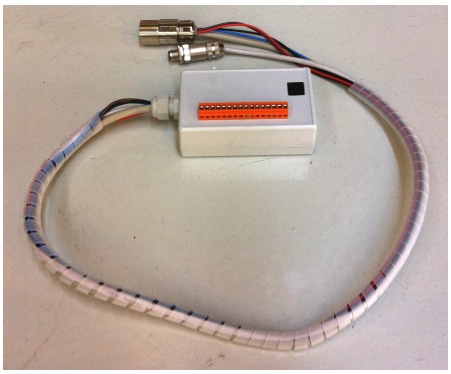


Figure C.4: The connector box used for the electrical connections through the arm

C.3 Establishing I/Os on the controller

The KUKA LWR is delivered without any available input/output possibilities, and as such third-party equipment has to be bought and configured in order to gain the desired I/Os. After consulting KUKA [17] and a robot forum on the internet [18], both sources suggest using the DeviceNet connection on the robot controller, and an external I/O module, where the Beckhoff BK52XX modules are recommended, due to both the low cost and the ease of setup of these modules. This is also a modular solution, which enables an expansion of the I/Os should the need arise. Due to a long delivery time on some products from Beckhoff, the components are chosen to give the least delivery time, enabling a delivery of just three working days. The components received are:

BK5250 Bus Coupler which is the common bus, handling the connection between DeviceNet and the connector blocks attached to the bus coupler

KL2404 digital output 4-channel digital outputs (2 pcs)

KL1404 digital inputs 4-channel digital inputs (2 pcs)

KL9010 bus end terminal closing the connection from the bus coupler to the I/O modules

The connection needs some configuration, since the 5-pin connection of the DeviceNet requires power and signal connections, and the bus coupler and digital outputs require power as well. Since the DeviceNet connection, bus coupler and digital outputs all require 24VDC, a common power supply is used for this. The 2-channel signal part of the DeviceNet connection need to be shielded, where the shield of both connections

needs to be connected to a common pin on the DeviceNet connector. Besides this, each end-point of the DeviceNet connection needs a 121Ω resistor[1] [19]. The complete wiring of the Beckhoff I/O module to the controller is shown in Figure C.5.

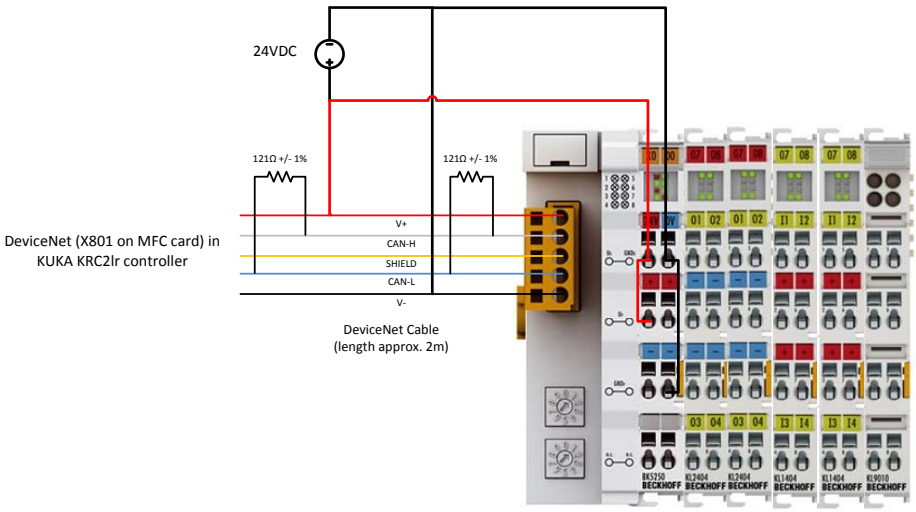


Figure C.5: Wiring of power and DeviceNet connection from the controller to the Beckhoff BK5210

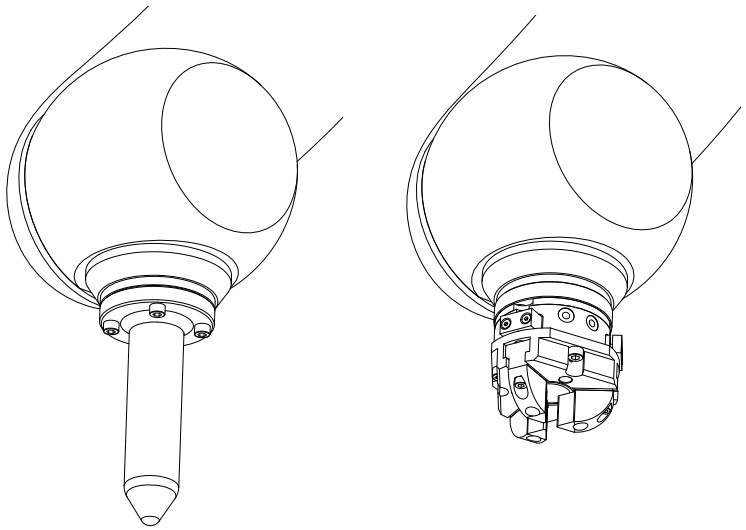
After the wiring of the connector, the KUKA controller needs to be configured to use the Beckhoff module. This is done in two configuration files, `DEVNET.INI` and `IOSYS.INI`. The first configuration file specifies the settings for the DeviceNet connection, such as debug information, log file generation and MACID of the devices attached to the controller[2]. The second file is the configuration for the I/O system, and specifies which interface is used for each input and output. In this case, DeviceNet is enabled in the configuration file, and it is specified how the controller should map the I/Os. This is done by adding the two lines `OUTB0=11,0,x1` and `INB0=11,0,x1` to the file, which specifies input and output (`IN/OUT`), with a byte offset of 0 bytes each (`B0`), on MACID 11, with 0 bytes offset and an address width multiplier of 1, meaning 1 byte total. After this configuration, the Beckhoff device is attached to the controller, and the I/O driver of the controller is restarted, after which the I/O is functional.

[1] DeviceNet is developed so that a lot of devices can be connected in a network, which is the reason the endpoints of the system need to be specified

[2] The MACIDs are unique addresses for each attached device, specified on the BK5250 via two dials

C.4 Temporary tools

Since no electric gripper is available in the laboratory, and the electric gripper for the new configuration of the Little Helper is not chosen and bought yet, a temporary solution has to be made. At KUKA College, Gersthofen, a fixed pin tool was used to demonstrate the capabilities of the LWR, and a copy of this tool has been made in the laboratory as well. This tool is only useful for simple movement operation, while also acting as a handle for moving the robot in Gravity Compensation mode, and is as such not suitable for actual robot tasks, but more for demonstration of the capabilities of the KUKA LWR. The demo tool is shown in Figure C.6(a).
Since a gripper is needed for some of the exercises on the LWR, a pneumatic gripper is chosen, since a gripper is available in the laboratory along with pneumatic valves, actuated by a 24VDC signal. Therefore, the tasks in Part II of the project is carried out on the second LWR, with *pneumatic energy supply system*, consisting of two pneumatic hoses and a 12-pole electrical connection through the manipulator. The gripper chosen is a Schunk JGZ-64 3-finger concentric gripper, and a new set of jaws is designed, with a 60° tip, so the gripper can pick up both planar and round parts, since two of the jaws then provide a plane surface, with the third jaw pressing the part against this plane. The gripper with the new jaws is shown in Figure C.6(b).



(a) Demo tool (b) Schunk 3-finger gripper

Figure C.6: The two temporary tools used on the LWR

After the initial installation, configuration of I/O system and design of temporary tools, the robot is fully functional with the facilities needed for the further work with demonstrating the capabilities of the robot.