Diploma Thesis

Teleoperated Microassembly: Combining the Magnetic Levitation Haptic Interface with Minifactory

Michael Kummer

Dr. Ralph Hollis
Carnegie Mellon University
The Robotics Institute
Microdynamics Systems Laboratory
5000 Forbes Ave, Pittsburgh, PA 15213
Adviser

Prof. Dr. Bradley J. Nelson
Institute of Robotics and Intelligent Systems
Swiss Federal Institute of Technology Zurich (ETH)
Tannenstrasse 3
CH-8092 Zurich

2005-05

Preface

I would like to thank Prof. Dr. Bradley Nelson and Dr. Ralph Hollis for giving me the opportunity to write my diploma thesis at the Robotics Institute at Carnegie Mellon University. I would further like to thank Dr. Ralph Hollis and Prof. Alfred Rizzi for their support and guidance during my time working at their lab. Many thanks also to Rob Schlender and Bert Unger for their support with UNIX, programming, the minifactory, and the Magnetic Levitation Haptic Interface.

Contents

List of Tables

List of Figures

Abstract

This thesis describes a technique for teleoperated microassembly. To reach this goal, the minifactory is combined with the Magnetic Levitation Haptic Interface, which were both developed at the Microdynamic Systems Laboratory at Carnegie Mellon University. The minifactory is a system intended for the autonomous assembly of MEMS parts such as hearing aids or magnetic storage devices. The Magnetic Levitation Haptic Interface is a 6-DOF haptic device based on the Lorentz force actuation. The combination of the two technologies shall yield a means to remotely control the minifactory as well as a feasibility estimate to determine to what extent this combination can provide improved alternatives to todays manual microassembly. This thesis illustrates both technologies, describes the communication setup between the involved hardware components, and addresses the control issues underlying this task.

Zusammenfassung

Diese Diplomarbeit beschreibt das Ausarbeiten einer ferngesteuerten Mikromontage-Technik. Um dieses Ziel zu erreichen werden die “Minifactory” und das “Magnetic Levitation Haptic Interface” in einem System integriert. Beide Technologien wurden am Microdynamic Systems Labor an der Carnegie Mellon Universität entwickelt. Die Minifactory ist ein System welches für die automatisierte Montage von MEMS-Teilen, wie z. B. Hörhilfen oder magnetische Speichergeräte entwickelt wurde. Das Magnetic Levitation Haptic Interface ist eine haptische Mensch/Maschine Schnittstelle mit 6 Freiheitsgraden. Es basiert auf dem Prinzip der Lorentz Kraft. Die Kombination dieser beiden Geräte soll einem Benutzer ermöglichen die Minifactory fernzusteuern. Gleichzeitig wird ermittelt in welchem Mass dieses System eine Verbesserung zu herkömmlicher, manueller Mikromontage bietet. In dieser Diplomarbeit werden alle Bestandteile des Aufbaus, die Verbindung zwischen den einzelnen Bestandteilen und die Steuerung des Systems beschrieben.

1 Introduction

After several decades of mass-production, changing marketplace requirements have forced manufacturers to be more adaptable to unknown market requirements. Buzzwords such as *Six Sigma*, *Continuous Improvement*, *Lean Manufacturing*, *Agile Manufacturing* and so forth have started to influence the industry on a strategic as well as a manufacturing level. The premise of these ideologies is to direct companies into a position that will allow them to remain competitive in a fast changing market environment.

Especially in the high-tech industry which increasingly relies on microassembly, the requirements on products drive manufacturers to change their production strategies. Shorter product life cycles [1] and smaller sets of produced parts require production sites to be capable of adapting to changing marketplace requirements faster and in a more flexible fashion. This claim for agility encourages the development of highly modular factories. Furthermore, an increase in automation and autonomy would improve the microassembly production in a domain where scale and environmental issues make it desirable to eliminate humans from the production process.

Increased autonomy and automation, however, cannot eliminate the necessity for human operators to be able to take control of the situation in the occurrence of unpredicted events. Operator interaction must be possible to aid the factory. The human supervisor is supposed to resolve problems that the factory cannot overcome by itself. If such a problem was to occur in a sterile factory environment, it would be beneficial if the operator could fix the problem from outside the factory environment. In case of a larger breakdown, it would be convenient if finished assemblies could at least be remotely recovered before the intact factory environment would be entered.

When a limited set of assemblies does not make the design and programing of a factory profitable, currently the prevailing alternative is manual microassembly. The possibility to manually control a microassembly factory could render a powerful alternative to traditional, manual microassembly. Fatiguing work on a precision microscope could be replaced by a more comfortable and thus productive process, since the same work could be performed while sitting in front of a computer monitor using an ergonomic interface device. Furthermore, the operator could teach the factory a combination of movements, which the factory could learn after few repetitions and eventually could perform on its own. A haptic device is an ideal device to introduce in a factory built to perform microassembly. The reflection of involved forces is a key issue since the manipulated parts are so small and fragile. A haptic interface device enables manual interactions of humans with virtual environments or teleoperated remote systems using sense of touch [2]. Other benefits of using a haptic interface device for teleoperated microassembly are higher resolution and better control of the applied forces to the delicate assembly parts.

This report describes the effort to combine a microassembly factory and a haptic interface device. The goal is to develop a new system to perform teleoperated microassembly.

1.1 Problem Specification

The path to develop means of teleoperated microassembly through the combination of a haptic interface device with a microassembly factory can be structured into three main steps:

- Establishing the capabilities of all involved hardware in order to understand the underlying difficulties in trying to combine the haptic interface device with the microassembly factory.
- Establish communication between the haptic interface device and the microassembly factory.
- Design appropriate controllers for both the haptic interface device and the microassembly factory in order to operate the newly created setup.

2 AAA, Minifactory, and Magnetic Levitation Haptic Interface

Rapidly changing marketplace requirements impose agility demands on microassembly factory systems. Today's factories, however, still lack these abilities and have unsatisfactory changeover times between the manufacturing of different assemblies [3]. In combination with shortened product life cycles, this implies that a major part of new product cost must be allocated to the design and setup of new production sites. It would be highly desirable to be able to switch between the production of different assemblies within days. The *minifactory* at the Micrody-

Figure 1: Minifactory at MSL.

namic System Laboratory* (MSL) at Carnegie Mellon University (CMU) is being developed in order to address these issues (Fig. 1). The minifactory targets the assembly of small electromechanical devices such as storage drives or hearing aids and is the instantiation of a much broader philosophy, namely, the *Architecture for Agile Assembly* (AAA).

Another device developed at MSL is the *Magnetic Levitation Haptic Interface.* The Magnetic Levitation Haptic Interface is a haptic device which features 6 DOF (Fig. 2).

*http://www.msl.ri.cmu.edu

Figure 2: Magnetic Levitation Haptic Interface at the MSL.

The minifactory and the Magnetic Levitation Haptic Interface are the instruments used to develop a teleoperated microassembly system. In this section AAA, minifactory and its components, and the Magnetic Levitation Haptic Interface are described.

2.1 The Architecture for Agile Assembly

The *Architecture for Agile Assembly* (AAA) [3, 4, 5] constitutes a philosophy for the creation of high-precision, modular, miniature assembly systems. Its goals are to reduce assembly system changeover times, facilitate geographically distributed design and deployment of assembly systems, and increase the precision and quality of products. It aims at providing a solution for industries that would benefit from drastically reduced factory design and deployment times. Part of the improvement in factory design, deployment time, and increased portability, results from reducing the size of typical assembly systems from large room size to tabletop size.

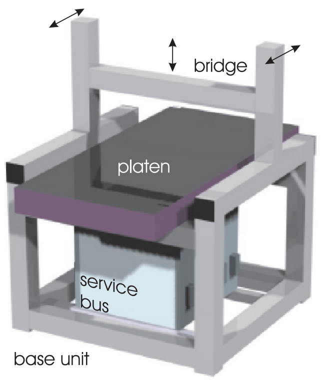
One reason that AAA is so powerful is that a factory built following this philosophy consists of distributed low-DOF robotic modules, called agents. These agents all are mechanically, computationally, and algorithmically modular manufacturing entities. They are capable of interacting with each other and thus performing assembly tasks requiring more DOFs than a single agent provides. More importantly, these modular entities are designed to be part of a larger factory system. This will make it easier to add and remove agents when changes to the factory need to be performed. In order for this distributed behavior to be operative, each agent will have its own powerful computer. Factory-wide standard procedures and protocols will further facilitate the implementation of this behavior and will greatly increase its flexibility.

When short changeover times and high flexibility are a goal, the transition between a model of a factory and the deployment of this model must be short. In order to achieve this goal a unified design, simulation, programming, and monitoring interface tool provides the opportunity to design the factory and its processes on a computer. The real factory then will be programmed by directly interfacing with this automatically synchronized virtual factory [6]. The factory will be self-calibrating to round off this process and speed up the deployment time.

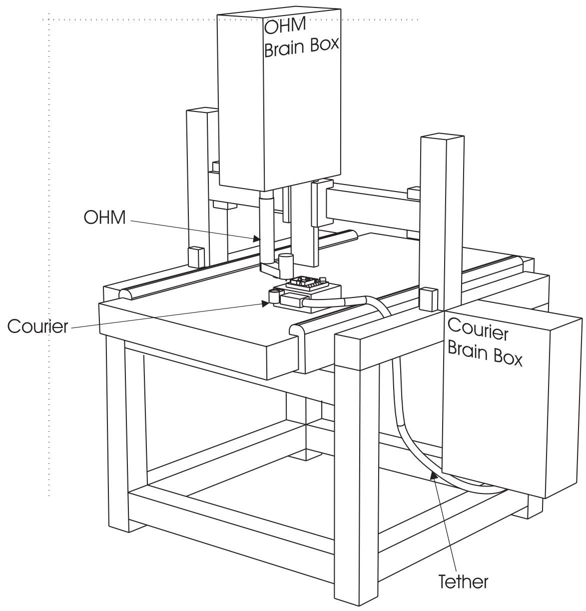
2.2 The Minifactory Environment

One physical instantiation of the AAA philosophy is called the minifactory [1, 4, 5, 7]. The scope of mechanical capabilities of a minifactory has been limited to assembly and processing operations requiring four or fewer DOF. This was done in order to ensure analytic tractability and design practicality. A closer look at present, industrially meaningful, automated microassembly processes reveals that they principally consist of four-DOF vertical insertion tasks. This limitation is due to the fact that no suitable means have yet been developed to efficiently automate assembly processes requiring more than four DOFs at the micron scale. Most microassemblies nowadays are designed with this limitation in mind ("design for assembly"). As long as no efficient high-DOF assembly has been realized, factory throughput for higher DOF assembly will be limited.

The minifactory consists of modular building blocks to ensure high flexibility. The biggest block is the base unit frame. It incorporates a modular service bus that supplies the agents with power, air, vacuum, and network connections. The base unit supports a platen tile that represents the factory floor. The surface of the platen tile consists of a waffle-iron type grid that was planarized with epoxy to yield a flat surface. A base unit and platen can be seen in Figure 3(a). In a departure from assembly robots such as the SCARA robot arm, predominantly used for four-DOF vertical insertion tasks, the minifactory splits its four DOFs



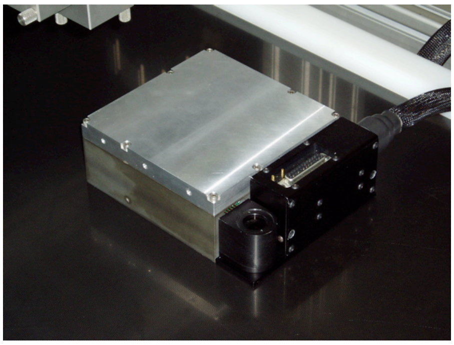
(a)



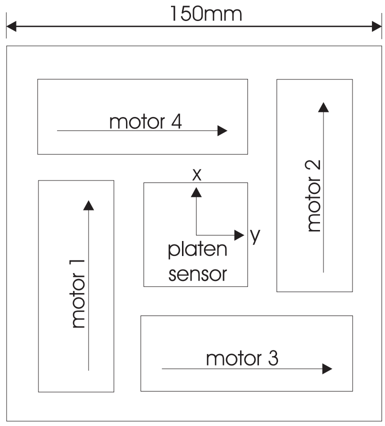
(b)

Figure 3: Figure (a) shows the minifactory base unit comprised of power and network supply, platen, base frame, and the adjustable bridge. In (b) the base unit with an OHM (overhead manipulator) and a courier robot is displayed.

among its various agents. Essentially a coarse distinction can be made between agents that move over the platen, and agents that are mounted to the modular and adjustable bridges spanning the factory floor (Fig. 3(b)). Currently, the minifactory is equipped with the agents displayed in Figure 3(b). These agents emulate the SCARA's four-DOF assembly capabilities by appropriately interacting with each other. The agent that moves over the factory floor is called the courier robot. The courier is a modified two-axis x, y closed loop planar linear servo motor (Fig 4). It glides over the factory floor on a $12 - 15\,\mu\text{m}$ air bearing and incorporates a three-DOF x, y, θ alternating current magnetic platen sensor. This sensor allows for closed loop control and has a linear resolution (1σ) of $0.2\,\mu\text{m}$ and an angular resolution of 0.0014° [8]. A precision optical coordination sensor is also attached to the side of the courier. This sensor detects LED beacons on the overhead manipulators (OHMs) [9]. This sensor is currently operating at a 1σ



(a)

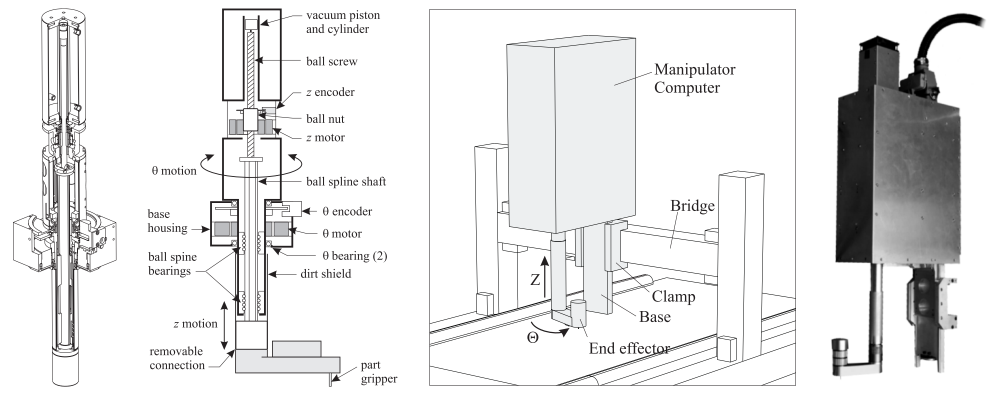


(b)

Figure 4: (a) Actual picture of courier. (b) Schematic bottom view of courier displaying motor arrangement.

resolution of 0.15 μm. The courier is actuated by four motors using a magnetic flux-steering principle. Two motors form a pair along each axis of motion. The motors have teeth that align with the precision position reference grid of the factory floor. The courier robots have a double function in the minifactory. First, they carry the subassemblies through the factory, replacing the commonly used conveyor-belt configuration. Second, courier robots, together with the OHM, transiently form four-DOF manipulators.

The OHM is a two-DOF robotic manipulator. It is clamped to the bridges spanning the factory floor and keeps a fixed base position during operation. Its two DOFs consist of vertical movement along the z-axis and rotation in θ. Its travel in the z-direction is about 150 mm and it can rotate for 570° between stops in θ (Fig. 5). At the end of the OHM a quick-release connection interface allows the attachment of different end effectors. Currently, a 100 mm offset arm is attached which incorporates an end effector/force sensor (Fig. 6). The OHM design facilitates parts picking from potential, future parts feeders that would be clamped to the bridges next to their respective OHMs. The OHM has a vertical resolution of 2 μm and a tangential resolution of 0.9 μm at a 100 mm radius [10]. Sensing and actuation are performed at a servo rate of 2 kHz. Its peak torque is



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

Figure 5: Figure (a) shows a Pro/ENGINEERING drawing of the OHM without the endeffector arm attached to it. In (b) a schematic of the OHM is displayed. Figure (c) depicts an OHM in the minifactory while (d) is a picture of an actual OHM.

$1.4\,\mathrm{N}\cdot\mathrm{m}$. Other forms of *overhead processors* such as screwdrivers, lasers, glue dispensers and the like are possible expansions of the capabilities of minifactory.

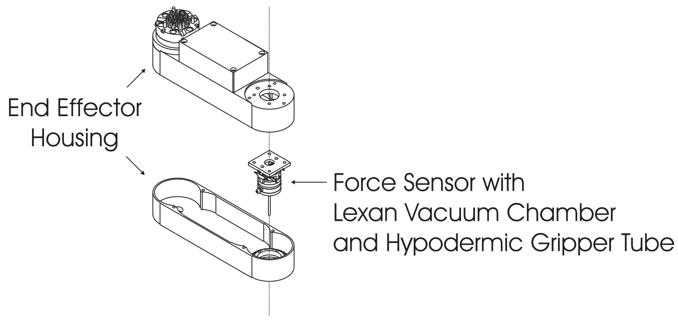


Figure 6: Exploded view of OHM offset arm.

The configuration of OHM and courier forming the four-DOF manipulator has several advantages over the commonly used SCARA arm. Since the sensors and actuators of the agents are collocated in the minifactory, higher precision can be achieved than with a serial linkage based system. Lower masses of the individual agents allow for higher accelerations and, thus, reduced process times. Tasks also can be pipelined between the two robots. While the OHM picks up a part from a parts feeder, the courier can move to the assembly location and position itself appropriately. The mechanical and electrical modularity of the minifactory agents allows for more flexibility, both in adding new elements to the factory and in modifying the current setup.

The minifactory has two disadvantages. First, the limited workspace of the OHM requires parts feeders to be located in close proximity to the OHM. The second disadvantage is the movement restriction resulting from the tether connected to the courier [11]. The tether connects the courier to its *brain box*. Each agent has its own brain box which is the computational unit that controls the agent's representation towards its peers as well as its function in the entire factory. While the OHM is fixed to its brainbox, the courier needs to be able to move around the factory floor. Tether-less couriers are currently still a topic of research and bear the disadvantage of needing to incorporate their own source of power, network, and air pressure supply.

Since there is no central program running in a minifactory, distributed programming is necessary in this environment [12]. Each agent in a minifactory runs its own program and needs to display its capabilities to its peers in order to make cooperation possible. For this reason an agent's execution is divided into two levels. At the high-level, Python[†] scripts control the semantics of factory operations and the associated discrete events. Low-level processes are responsible for sequencing and executing the specific control laws. These control strategies are written in C and C++. While the high-level programs are generated by the user and downloaded to the agents, the low-level programs are hard-coded in the form of a palette of real-time controllers and a manager which executes them. The minifactory relies on two network systems in order to implement its distributed behavior: a global network and a local network.

The global network works over one of two 100 Mbit interfaces in each agent brain box. It utilizes standard IP protocols and controls the global behavior of a factory. It handles factory-wide, non-real time communication and provides the communication line for the high-level programming.

The local network works over a high speed 100 Mbit connection and uses semi-custom communication protocols similar to UDP and TCP, but significantly sim-

[†]An object oriented language which can be interpreted or byte-compiled.

plified to improve performance. The local network, referred to as AAA-Net [13, 14], is responsible for handling the low level interactions of the minifactory and is a real-time, low latency network.

The current minifactory setup at the MSL is displayed in Figure 7. It consists of three complete base units and one base unit that lacks a service bus. Currently there are four OHMs and two couriers. Each service bus can provide power and network for eight agents.

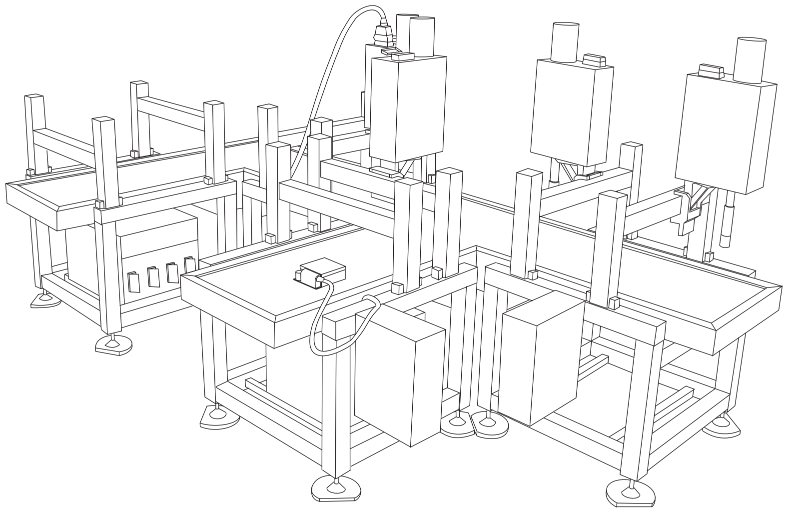


Figure 7: Current Minifactory configuration at MSL

2.3 The Magnetic Levitation Haptic Interface

At MSL, a fine motion, magnetically levitated 6 DOF robot wrist, the IBM Magic Wrist, was adapted for use as a haptic interface device [15]. This early haptic interface device evolved into the Magnetic Levitation Haptic Interface [16]. The physics behind the magic wrist and the Magnetic Levitation Haptic Interface involves the concept of actuation through Lorentz force. Lorentz force is generated when electric currents circulate in a magnetic field. The Magnetic Levitation Haptic Interface consists of a hemispherical bowl, called the flotor, carrying six coils. The flotor bowl is contained in the stator bowl, which is made up of 12 magnet assemblies; six of which are inner, and six of which are outer, magnet assemblies. One inner and one outer magnet assembly, form a pair that provides the magnetic field for the corresponding coil on the flotor bowl (Fig. 8). As the flotor bowl sits in midair when the Magnetic Levitation Haptic Interface is in use, there are no contact forces due to the actuation. This results in high control bandwidth, low inertia and simple dynamics for the levitated flotor. The technical specifications of this device are a motion range between $\pm 7 - 10\,°$ in rotational DOF and $\pm 12\,\text{mm}$ in translational DOF. The maximum stiffness is 25 N/mm at a 1500 Hz servo rate. The maximum force is 55 N and the maximum torque is $6\,\text{N} \cdot \text{m}$. The position sensitivity of the device is between $5 - 10\,\mu\text{m}$. The entire assembly sits in a housing that also contains the amplifiers that provide the coil current and the circuitry needed for position sensing (Fig. 2). The computer controlling the haptic device sits in a separate frame.

2.4 Force Sensor

When trying to perform contact tasks when controlling minifactory with the Magnetic Levitation Haptic Interface, it is necessary to provide means of force sensing. In 2000, DeLuca [17] built and installed a 3-axis force sensor in a minifactory OHM. The force sensor consists of four load cells that are incorporated into a mechanical flexure (Fig. 9). The load cells are arranged in such a way that one pair of load cells is sensitive to only one axis of applied torque. Thus, one pair measures the torque around the x-axis and one pair measures the torque around the y-axis. The axes can be seen in Figure 9(b). Torque in θ can be calculated from the horizontal distance between the OHM's axis of motion and the force sensor location. The load cells are modeled as linear springs and thus a linear relationship exists between the applied forces and torques, and the measured load cell voltages [17]:

$$\begin{bmatrix} V_1 \\ V_2 \\ V_3 \\ V_4 \end{bmatrix} = \underbrace{\begin{bmatrix} \frac{\alpha_1}{\beta_1} & \frac{\epsilon_1}{\beta_1} & 0 \\ \frac{\alpha_2}{\beta_2} & \frac{\epsilon_2}{\beta_2} & 0 \\ \frac{\eta_1}{\delta_1} & 0 & \frac{\zeta_1}{\delta_1} \\ \frac{\eta_2}{\delta_2} & 0 & \frac{\zeta_2}{\delta_2} \end{bmatrix}}_{C} \begin{bmatrix} F \\ M_x \\ M_y \end{bmatrix}, \tag{2.1}$$

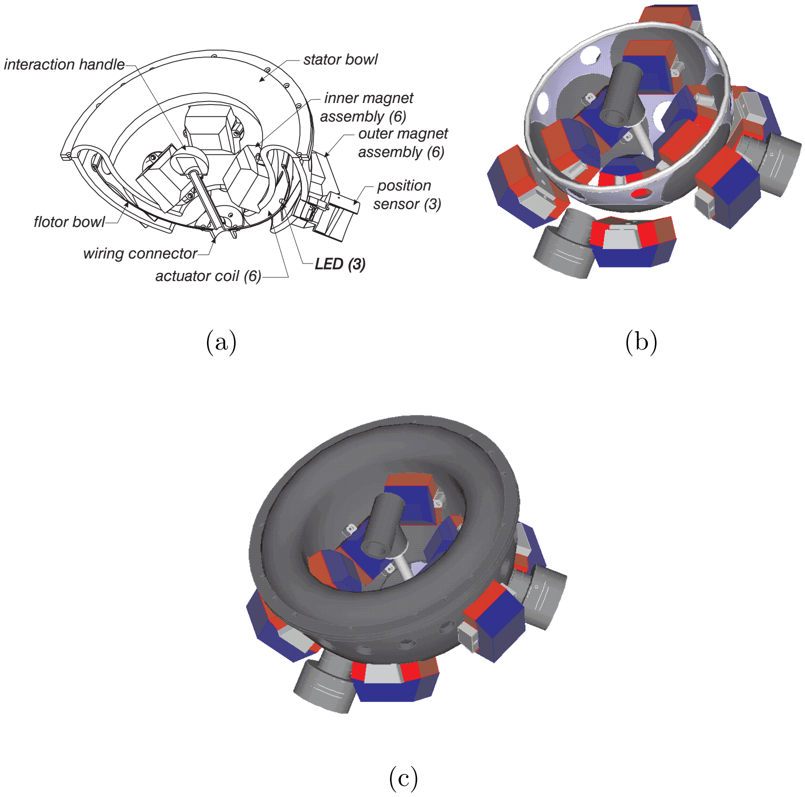
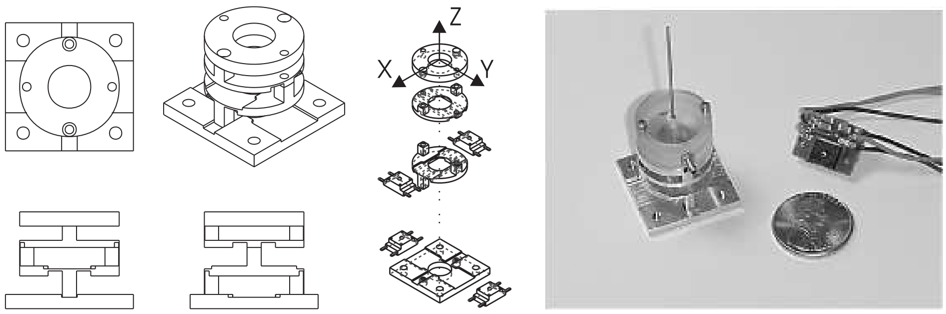


Figure 8: (a) Cut-away drawing of the Magnetic Levitation Haptic Interface assembly. (b) Flotor bowl with magnet assemblies. (c) Finished Magnetic Levitation Haptic Interface with stator bowl.

where the V_i represent the load cell voltages and F, M_x and M_y represent the force along the z-axis and the moments about the x- and y-axis respectively. Due to the fact that the load cells only operate in compression, they are preloaded in the flexure with half the maximum force range.



(a) (b) (c)

Figure 9: 3-axis force sensor: Figure (a) shows a drawing of the mechanical flexure. In (b) an exploded assembly drawing of the flexure and the single axis load cells can be seen. The actual hardware is shown in (c).

The force sensor has a bandwidth of 100 Hz. Once the sensor is incorporated into the end effector housing it achieves a force sensibility of approximately 78 mN (1σ) in the z direction and $0.6\,\text{mN}\cdot\text{m}$ (1σ) in x and y. A Lexan glass with a hypodermic gripper tube is mounted on top of the force sensor (Fig. 9(c)). This end effector can pick and release parts through vacuum.

3 Hardware Setup & Communication

To combine the Magnetic Levitation Haptic Interface and minifactory in a functional way, different hardware components need to interact. This section covers the involved hardware components as well as the communication between them.

3.1 Hardware Setup

In addition to the force sensor describe in Section 2, visual sensing is needed to provide a useful teleoperation setup. It is crucial for the operator of the Magnetic Levitation Haptic Interface to see what motions he introduces into the minifactory. A camera was added, to provide visual feedback of the scene. Besides the camera, an additional four computers were involved in the hardware and communication setup. These computers were the Magnetic Levitation Haptic Interface controller, a graphics workstation, the OHM brain box, and the courier brain box. A schematic drawing of the setup can be seen in Figure 10. What follows is a short description of the hardware components:

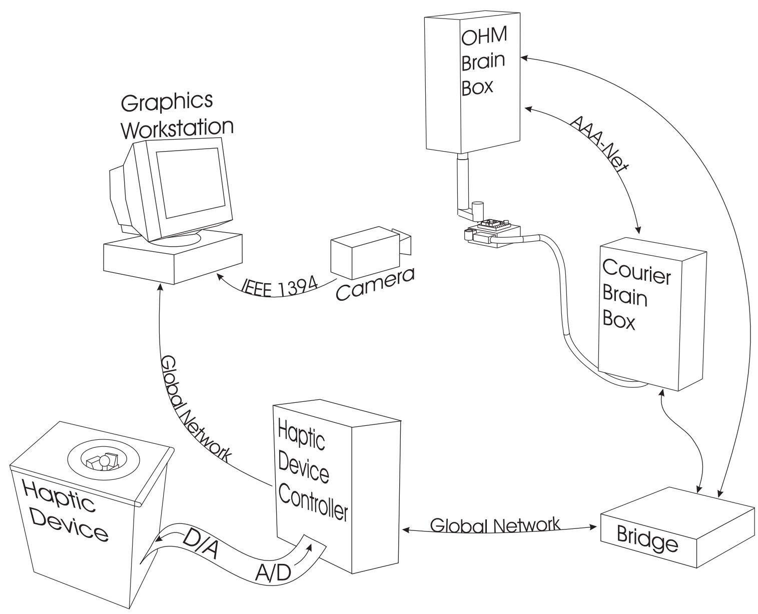


Figure 10: Schematic of the communication and hardware setup.

The Magnetic Levitation Haptic Interface Controller is the computer used to control the Magnetic Levitation Haptic Interface. It runs QNX[‡] version 6.1.0 which is a POSIX-conformant Unix-like real-time operating system. The Magnetic Levitation Haptic Interface controller is equipped with an Acromag® APC8620 PCI bus carrier board. This carrier board holds an IP220 12-bit analog output board, an IP 330 16-bit analog input module and an IP480 16-bit counter/timer module. This PCI card's analog input module will handle the sensor data from the Magnetic Levitation Haptic Interface, while the analog output board will control the Magnetic Levitation Haptic Interface's amplifiers. The processor of this computer is an AMD Athlon XP 2100+ clocked at 1.73 GHz.

The Graphics Workstation, like the Magnetic Levitation Haptic Interface controller, processes computations on an AMD Athlon XP 2100+ processor clocked at 1.73 GHz. In addition, it has a powerful GeForce4 Ti 4600 graphics card and 1 GB of RAM. This computer will be used to display the image of the workscene and for any other graphical process. The operating system on the graphics workstation is Redhat™Linux, kernel version 2.4.25. Furthermore the computer is equipped with an IEEE 1394 PCI adapter used to connect the camera.

The Camera used is a digital camera that complies with the IEEE 1394a standard. It will be attached to the IEEE 1394 bus on the PCI card in the graphics workstation. Video processing software is required in order to display a picture of the workspace on the graphics workstation. Coriander[§] is a GUI for displaying video input from an IEEE 1394 bus. In order for the camera to work with Coriander, it needs to be compliant with the IIDC v 1.04 digital camera specifications. The Coriander version used is version 1.0.0.

The Minifactory Brain Boxes are the OHM computer and the courier computer. The minifactory computers are based on an ATX formfactor Motorola

[‡]A product of QNX Software Systems, Ottawa, Ontario, Canada
[§]http://www.tele.ucl.ac.be/PEOPLE/DOUXCHAMPS/ieee1394/coriander/index.html

MTX-604-003 mother-board with a single 300 MHz Motorola 604e PowerPC [10]. The operating system on the minifactory brain boxes is LynxOS¶ version 3.0.1, which also is a POSIX-conformant real-time operating system. The minifactory brain boxes are also equipped with an APC8620 PCI bus carrier board.

3.2 Communication

All of the above mentioned computers are equipped with 100 Mbit Ethernet interfaces and thus connect with each other through the global network.While the graphics workstation and the Magnetic Levitation Haptic Interface controller both have a single Ethernet interface, the Minifactory brain boxes have two; one interface for the global network and an additional interface for use with the AAA-Network.

The communication interface between the Magnetic Levitation Haptic Interface controller and the minifactory was implemented using the communication libraries of the RHexLib Control Software. The RHexLib Control Software is a collection of software libraries used to control the RHex‖ hexapod robot [18]. The RHexLib communication facilities do not use the TCP/IP messaging protocols. Instead they are based on the UDP protocol, in which messages are sent to a destination with no guarantee of receipt. In RHexLib, some simple, but robust, mechanisms have been implemented on top of UDP to allow some guaranteed message delivery. The user can choose between using streams or mailboxes. While stream sinks maintain a queue of the received messages, mailboxes only hold the most recent message. The RHexLib libraries were used because they provide a simple yet reliable communication infrastructure. Furthermore, the RHexLib libraries were used in a prior project where the Magnetic Levitation Haptic Interface was used to control a PUMA arm. The RHexLib communication libraries were proven to work well for this teleoperated control task which encouraged their reapplication.

One problem inherent to interfacing the Magnetic Levitation Haptic Interface controller with minifactory was the endian issue. While the Motorola PowerPCs,

¶A product of LynuxWorks™, San José, USA

‖http://www.rhex.net

i.e. the minifactory brain boxes, store their most significant byte at the lowest address (big-endian format) the Magnetic Levitation Haptic Interface controller stores its most significant byte at the highest address (little-endian format). In order to work around this problem, byte swapping needed to be done when messages were sent back and forth between the minifactory and the Magnetic Levitation Haptic Interface controller. Since the Magnetic Levitation Haptic Interface controller has a faster CPU than the minifactory brain boxes, the byte swapping during the data transfer was performed on the Magnetic Levitation Haptic Interface controller.

In addition to byte swapping a timing issue had to be resolved in order to achieve acceptable communication bandwidth between the minifactory and the Magnetic Levitation Haptic Interface. The problem was that the `usleep`(μsec) function from the standard gcc library `unistd.h`, did not provide sufficient resolution on the minifactory computers or on the Magnetic Levitation Haptic Interface controller. Thus, the corresponding communication threads could not be reliably suspended and more than 80% of the sent packets were dropped. The problem was resolved by using the built in hardware clock featured on the APC8620 card to time the thread suspension. The byte swapping code as well as the communication script are displayed in Appendix A.

As shown in Appendix A, the main communication code relies on the mailbox paradigm; the reason being that messages were sent over the insecure global network. The loss of few packets could be tolerated whereas waiting on lost packets could cause more severe problems. It was thus necessary to provide a communication bandwidth that would compensate for the loss of a few packets. The minifactory is connected to the Global Network through a Cisco Systems Cisco 3000 bridge. This Bridge turned out to be a bottle neck and limited the total amount of packets that could be sent back and forth between the two agents. A total of roughly 1000 packets is the limit that can pass by the bridge in one second. As mentioned earlier in the text the force sensor has a bandwidth of 100 Hz. Following an engineering rule of thumb the force sensors bandwidth should at least be double to ensure good control performance. For this reason the sending rate from the OHM to the Magnetic Levitation Haptic Interface was chosen to be 250 Hz. The sending rate from the Magnetic Levitation Haptic Interface to the OHM was then chosen to be 500 Hz. This configuration provided sufficient bandwidth to perform the targeted tasks. On average 0.5% of the sent packages between the agents was lost.

The communication was setup so that the OHM initiates the communication by sending a packet to the Magnetic Levitation Haptic Interface, using the stream paradigm (Fig. 11). Then the OHM waits until the reception of the sent packet is acknowledged. The OHM then sends a second message to the Magnetic Levitation Haptic Interface which contains the OHM mailbox ID. Prior to sending this message for the first and only time endian swapping in the OHM communication code is performed. The mailbox ID is converted from big-endian to little-endian format in order for the Magnetic Levitation Haptic Interface to correctly understand the sent mailbox ID. After this initial handshake the stream paradigm will not be used anymore. The Magnetic Levitation Haptic Interface enters a loop where a mailer sends the current position of the Magnetic Levitation Haptic Interface to the OHM. In the same loop a mailbox receives the OHM position information. On the OHM side, a function gets called periodically in which a mailbox with the sent mailbox ID receives the position information of the Magnetic Levitation Haptic Interface. For every other received position information of the Magnetic Levitation Haptic Interface a mailer on the OHM side sends the current OHM position and the z-force and torque to the Magnetic Levitation Haptic Interface. During operation the flotor of the Magnetic Levitation Haptic Interface floats in midair. Initially the priorities of the communication thread and the position control thread for the Magnetic Levitation Haptic Interface were set the same. While the communication thread was waiting for the handshake, however, it starved the position control thread of the flotor and caused it to become unstable which made the flotor crash. This problem was eliminated by giving the communication thread a lower priority than the position control thread.

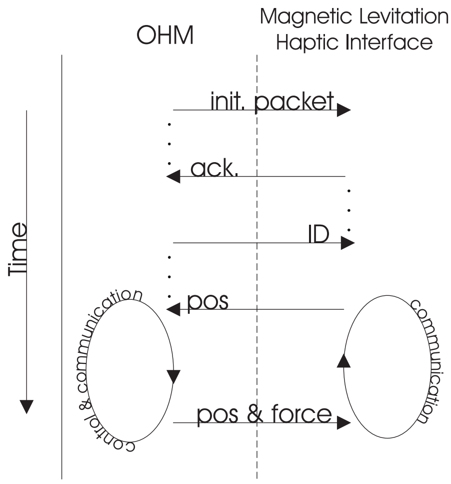


Figure 11: Network timing between minifactory and OHM.

4 Control of the Minifactory with the Magnetic Levitation Haptic Interface

To design a controller for the minifactory/Magnetic Levitation Haptic Interface system, it was necessary to understand the existing software and control architectures of the Magnetic Levitation Haptic Interface and minifactory. In this section, these architectures will be described. Furthermore, the control strategies when the two systems were combined shall be presented. Mechanical problems with the courier reduced the initially intended setup. In this section the control of the OHM with the Magnetic Levitation Haptic interface will be described. Considerations and problems when adding the courier to the setup will be discussed in Section 5.

4.1 Magnetic Levitation Haptic Interface

The Magnetic Levitation Haptic Interface control runs under the QNX operating system as mentioned before. QNX allows all threads in a process to see global variables. This implies that no shared memory structure is needed to make the data received from the communication thread visible to the control thread. The process of the control software starts all required auxiliary threads, specified in an initialization function, prior to starting the flotor position control function. A list of different controllers is specified in the control function, and depending on the handled task, the corresponding controller is chosen. For the operation with the minifactory, two threads, aside from the position control thread, were initialized on the Magnetic Levitation Haptic Interface. One thread handled local communication with the keyboard input and served as an emergency control system. The second thread was the communication thread described in the previous Section.

4.2 Controller in the Minifactory

The minifactory makes use of a hybrid control strategy as described in [19]. A collection of controllers is used where each controller has an associated domain and goal, to describe the motion of a factory agent. For a minifactory task, the free configuration space is decomposed into overlapping polygonal regions with parameterized controllers associated with each region. To achieve the overall goal, the hybrid control system activates the continuous underlying control policies depending on what state the factory currently resides. A predicate is defined as a boolean function associated with a specific controller. This predicate evaluates to true if the system is within the domain of the associated controller. A prioritized list of controllers and predicates is maintained to determine which controller should be activated. Controllers are programmed in C and consist of an initialization function, an execution function, and a terminating function. As mentioned in Section

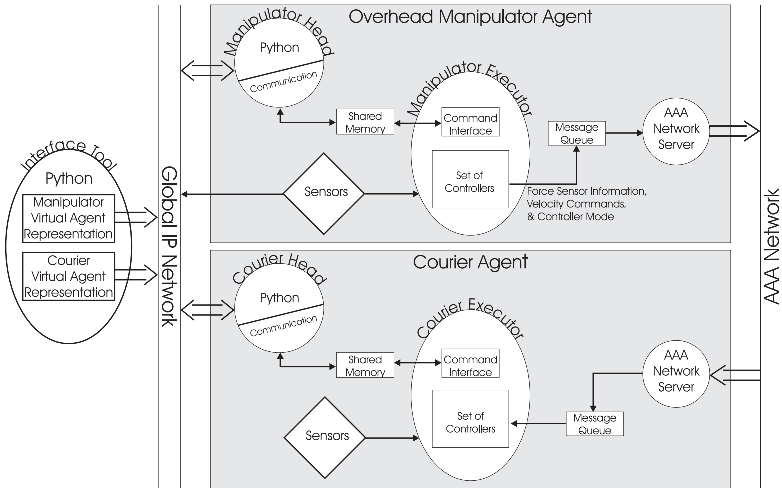


Figure 12: Controllers and communication in the minifactory.

2, the minifactory is programmed by the operator through an interface tool. This interface tool generates the high-level Python scripts that then are downloaded to the agents through the global network. The Python scripts determine the types of controllers running on each agent and the order in which they are instantiated. Figure 12 shows this communication and control setup. Circular shapes denote individual software processes, square shapes stand for hardware resources. The top half of the figure represents the OHM and the bottom half the courier. Heads interpret user-level commands from Python scripts. The executor runs the low level controllers. The internal communication between the processes on the individual agents utilizes shared memory structures, providing fast data sharing.

4.3 General Problems with the Presented Setup

Before addressing the control issue, there are a few points worth mentioning. The minifactory consists of modular agents and thus to represent their motions each agent was assigned its base-frame (Fig. 13). The base-frame of the OHM, c_{ohm}, was located at the zero position of the z-actuator of the OHM. z_{ohm} was aligned with the axis of rotation of the OHM. Its orientation in the $x_{ohm}y_{ohm}$-plane was chosen so that y_{ohm} is aligned with the OHM offset arm when θ_{ohm} equals zero. The courier's base-frame c_c was located at the point where an imaginary line from the end effector hit the factory floor when θ_{ohm} was equal to zero. The orientations of c_{ohm} and c_c differed by a rotation of π about y_{ohm} (Fig. 13). In addition to the base frames of the agents the coordinate frame of the end effector/force sensor c_s was defined as shown in Figure 13. The origin of the Magnetic Levitation Haptic Interface's base frame c_m was located at the center of the handle, after the startup routine of the Magnetic Levitation Haptic Interface was terminated, i.e. when the flotor was sitting in midair with no forces acting on the handle (see Figure 14). Since the flotor, respectively the handle of the Magnetic Levitation Haptic

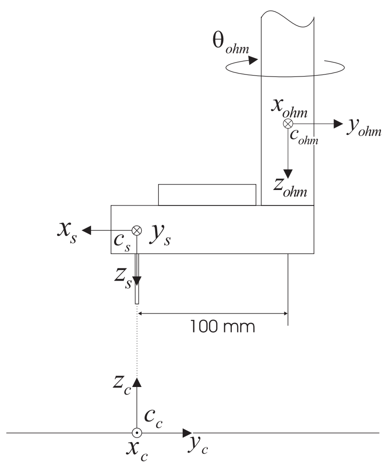


Figure 13: Agents' base-frames and end effector/force sensor coordinate frame.

Interface, can be pushed down or pulled up, an offset of 0.07 m was added to the z motion of the OHM. This point is about half way between the stops of the z motion of the OHM (Figure 17 (c)).

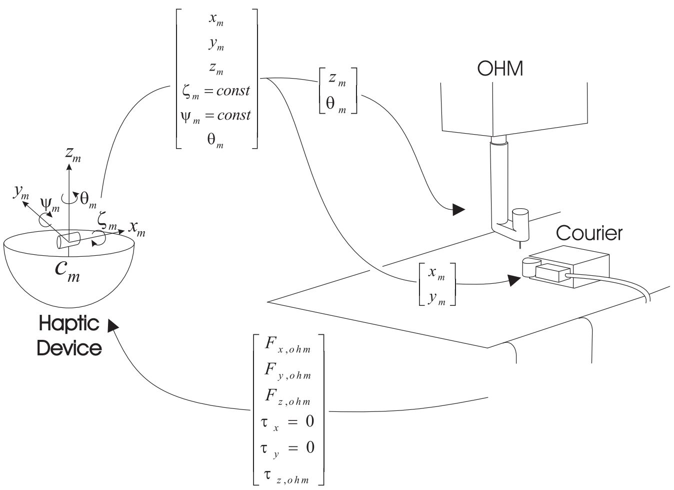


Figure 14: Splitting up the information sent from the Magnetic Levitation Haptic Interface to the minifactory.

Different coordinate frames between the force sensor and the OHM required a transformation of the measured forces between c_s and the agent coordinate frame c_{ohm}. To transform a measured force from c_s to c_{ohm}, a rotation of $\theta - \frac{\pi}{2}$ around the z_{ohm}-axis is necessary (Fig. 15). This leads to the following transformation of the measured forces $F_{x,s}$, $F_{y,s}$, and $F_{z,s}$ by the sensor:

$$\left\{ \begin{array}{c} F_{x,ohm} \\ F_{y,ohm} \\ F_{z,ohm} \end{array} \right\} = \begin{pmatrix} \cos(\theta - \frac{\pi}{2}) & -\sin(\theta - \frac{\pi}{2}) & 0 \\ \sin(\theta - \frac{\pi}{2}) & \cos(\theta - \frac{\pi}{2}) & 0 \\ 0 & 0 & 1 \end{pmatrix} \left\{ \begin{array}{c} F_{x,s} \\ F_{y,s} \\ F_{z,s} \end{array} \right\}$$

At the end of the startup routine of the Magnetic Levitation Haptic Interface, the handle of the Magnetic Levitation Haptic Interface is at the origin of the Magnetic Levitation Haptic Interface's coordinate frame c_m. A deflection of the handle of the Magnetic Levitation Haptic Interface from its position of origin results in motion in the minifactory. To avoid uncontrolled or unwanted motion in the minifactory, the PD position controller running on the Magnetic Levitation Haptic Interface always exerted a restoring force F_r on the handle. By default, the handles position was servoed with moderate gains to the origin of c_m. All other forces that were involved in the control task were superimposed onto F_r.

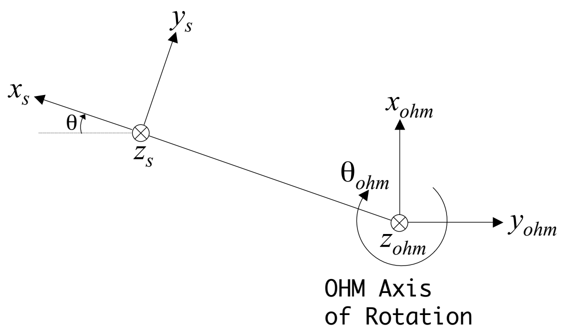


Figure 15: Factory and Force Sensor Coordinate Frames.

4.4 Control

As already mentioned, because of mechanical problems with the courier, the control of minifactory was reduced to controlling the OHM with the Magnetic Levitation Haptic Interface. The control problem was divided into different subtasks which will be described here. In order to avoid complications with the Python interface used to control minifactory the communication setup described earlier was changed according to Figure 16. Instead of initializing the OHM through Python scripts,

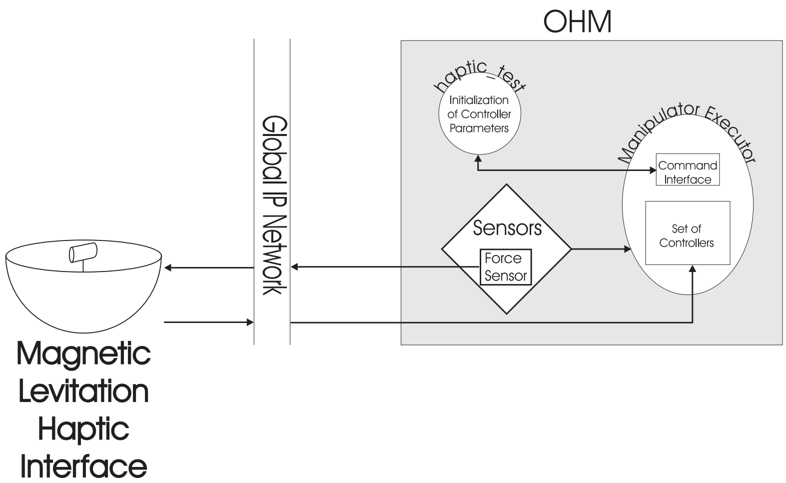


Figure 16: New communication and control architecture

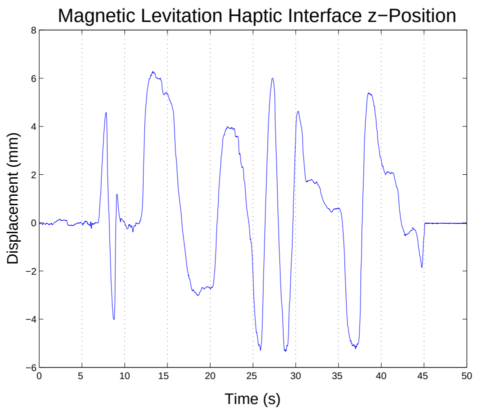
the executable haptic_test was created. This executable initialized all the necessary OHM parameters involved in the process.

4.4.1 Unilateral Control of Minifactory through the Magnetic Levitation Haptic Interface

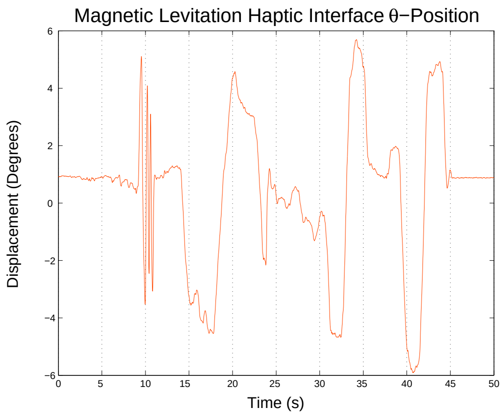
The first step was to gain control of the θ and z motion of the OHM through the Magnetic Levitation Haptic Interface. This unilateral control strategy was implemented to gain insight on how well the communication between the Magnetic Levitation Haptic Interface and minifactory, presented in Section 3, is suited for the control task. The goal for the unilateral control strategy was to servo the OHM to the current position of the Magnetic Levitation Haptic Interface. Both the OHM and the Magnetic Levitation Haptic Interface were controlled by their corresponding PD position controllers. As can be seen in the controller code of the OHM in Appendix A, each DOF was controlled independently. The PD controller of the Magnetic Levitation Haptic Interface was permanently servoing to the origin of c_m.

Because the OHM and the Magnetic Levitation Haptic Interface have such different workspaces, the positions sent to the OHM from the Magnetic Levitation Haptic Interface were scaled. An angular scaling factor k_{ang} was applied to the θ values and a translational scaling factor k_{trans} was applied to the z values sent to the OHM. The angular sensor readings of the Magnetic Levitation Haptic Interface and the OHM are in degrees. The z movement, however, is measured in mm for the Magnetic Levitation Haptic Interface and in m for the OHM. The values of the gains of the OHM's PD controller and the scaling factors can be seen in Table 1. The negative sign of the scaling factors is a result of the different orientations of the frames of reference c_m and c_{ohm}. Through this scaling the small movements of the Magnetic Levitation Haptic Interface were transformed into larger displacements of the OHM.

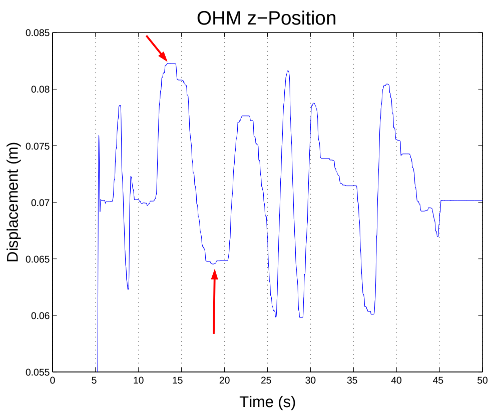
How well the OHM did respond to the motion of the Magnetic Levitation Haptic Interface can be seen in the graphs in Figure 17 where the θ and z positions of the OHM and the Magnetic Levitation Haptic Interface are plotted against time. Due to the different orientations of c_{ohm} and c_m the sensor readings of the Magnetic Levitation Haptic Interface were multiplied by -1 in order to allow an easy comparison of the movements. From Figure 17 it can be seen, that both, the θ and the z position of the OHM followed the general motion of the Magnetic Levitation Haptic Interface. It is apparent, that the angular motion of the OHM followed



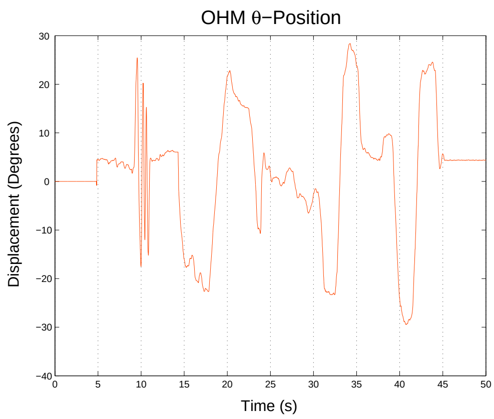
(a)



(b)



(c)



(d)

Figure 17: θ- and z-Position for Unilateral Control: the two red arrows in figure (c) display two examples where the z-motion of the OHM does not exactly match the motion of the Magnetic Levitation Haptic Interface.

Table 1: Gains and scaling factors of the PD position controller of the OHM.

$K_{p,z}$	17500 N/m
$K_{d,z}$	250 N/m/s
$K_{p,\theta}$	60 N m/°
$K_{d,\theta}$	0.65 N m/m/s
k_{trans}	-0.0015 m/mm
k_{ang}	-5

the angular motion of the Magnetic Levitation Haptic Interface more precisely than the translation along the z-axis (Fig. 17 (c)). These small discrepancies in z motion occurred when only very small displacements of the Magnetic Levitation Haptic Interface were executed. These movements must be related to unwanted shaking of the hand more than intended movements. While friction in the θ direction can be neglected for the OHM, friction plays a significant role for movements in the z direction, since the z axis is driven by a ball screw. This is the reason why the OHM did not exactly follow the motion in the z direction. The small position differences in these cases did not generate high enough forces to overcome friction.

4.4.2 Bilateral Control: Transformed Forces

After the unilateral setup was proven to work, the force readings of the OHM's force sensor were integrated to form a bilateral setup. The OHM and the Magnetic Levitation Haptic Interface were still controlled by their individual PD position contollers. Just as in the previous setup, the OHM was servoed to the current position of the Magnetic Levitation Haptic Interface with the same gains and scaling factors as in the previous setup (Tab. 1).

The desired position of the Magnetic Levitation Haptic Interface was initiated to be at the origin of c_m. However, during operation the θ and z values of the desired position for the Magnetic Levitation Haptic Interface were changed according to the forces measured by the force sensor of the OHM. The measured forces were scaled with k_θ and k_z (Tab. 2). When the force sensor of the OHM measured a force a resulting motion was introduced in the Magnetic Levitation Haptic

Table 2: Force scaling factors.

k_z	$-2.6\,\mathrm{mm/N}$
k_θ	$-1.8°/\mathrm{N\ m}$

Interface. This motion caused the handle of the Magnetic Levitation Haptic Interface to move in the direction opposite to the direction of the force. A reduction of the force on the tip of the hypodermic gripper tube was the result since the movement of the handle of the Magnetic Levitation Haptic Interface introduced a new setpoint in the PD position control of the OHM. If an operator was to hold the handle of the Magnetic Levitation Haptic Interface he could feel a force acting on the handle. This was a first form of force feedback between the OHM and minifactory.

In Figure 18 the results of this experiment can be seen. The first set of peaks before 50 s represents position control of the OHM with the Magnetic Levitation Haptic Interface. The second set of peaks after 50 s is a result of force application to the tip of the hypodermic gripper tube. As can be seen in Figure 18 (b), (d) and (e) the initial setup was unstable as exemplified by the green arrows. It can clearly be seen that when forces were applied to the force sensor, the OHM started to oscillate in the θ direction. This problem could be eliminated through reduction of the tangential scaling factor k_{ang} to -0.5. The improvement can be seen in Figure 19. In z the oscillations were not apparent. What perturbed the z signal was crosstalk between the different force sensing axes. However, the values were small enough and did not result in an oscillation in z.

4.4.3 Impedance Control of the OHM

In a third step an impedance control of the OHM was integrated in the setup. In the previous setup the OHM reacted to the measured forces depending on the movement of the handle of the Magnetic Levitation Haptic Interface. This setup inherently bore the risk to break the hypodermic gripper tube connected to the force sensor, because the maximum force and torque exerted by the Magnetic Levitation Haptic Interface are limited. An operator was capable to overpower

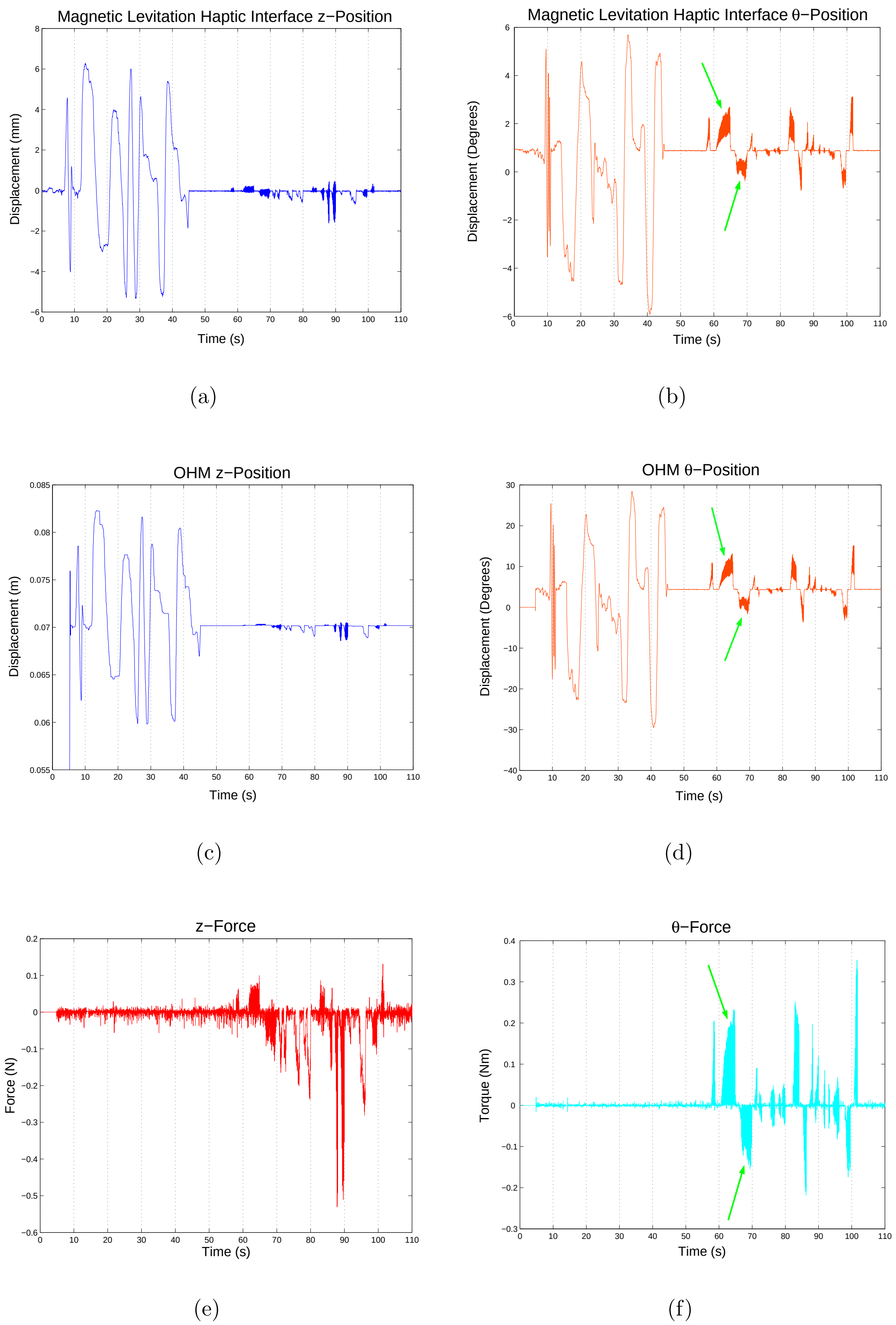
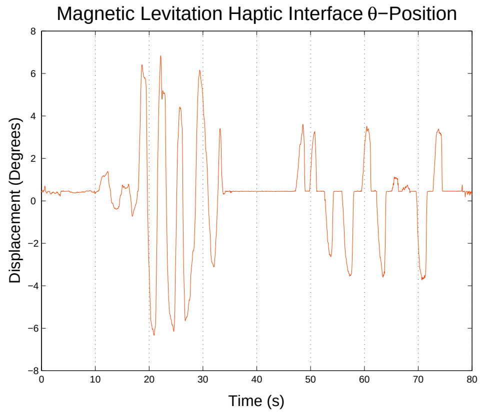
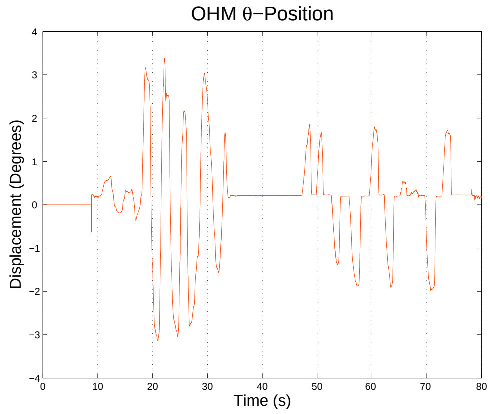


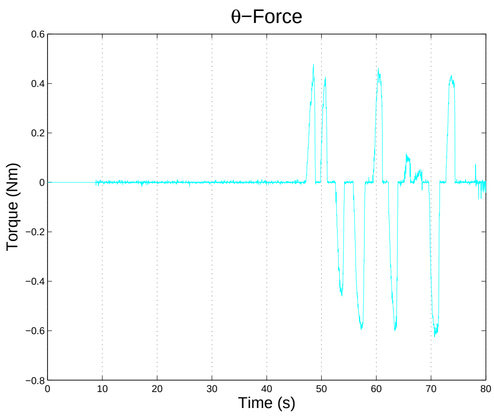
Figure 18: θ- and z-position for bilateral control using PD position controllers: the two green arrows in figures (b), (d), and (f) display two examples where the oscillations occurred in the θ direction when forces were applied to the hypodermic gripper tube.



(a)



(b)



(c)

Figure 19: θ-position for improved bilateral control using PD controllers: it can be seen how reducing the scaling factor for the θ-DOF eliminated the oscillation problem.

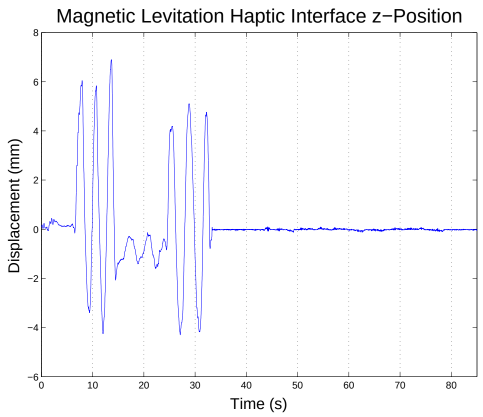
these maximum values fairly easy.

In this new setup the same PD position controller as in the previous two setups was running on the Magnetic Levitation Haptic Interface. On the OHM however, the position was controlled by an impedance controller. The utilized control law was the same as used by DeLuca [17], when he introduced the force sensor into minifactory and was of the form

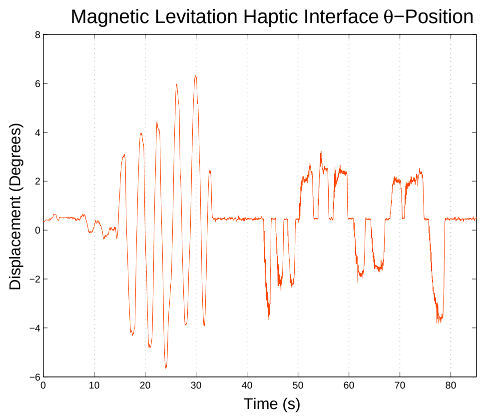
$$\tau_a = M_a M_d^{-1} \left[F_e + K_d(q_0 - q) - B_d \dot{q} + G_i \int_0^t (F_e - F_d) d\tau \right] + B_a \dot{q} + f(\dot{q}) + F_e.$$

For this control law q is the generalized configuration of the system; M_a and B_a are square matrices that describe the mass and damping parameters of the OHM; $f(\dot{q})$ is a vector containing the friction terms; the vector τ_a represents the applied actuator forces; and F_e is the vector of the applied environmental forces. The terms M_d, B_d, K_d, and F_d are respectively the desired mass, the desired damping, the desired stiffness, and the desired force of the system. These terms allowed the user to specify the overall system behavior. In this case the system was made to behave like a mass attached to a spring and damper about a nominal point q_0. The point q_0 was the setpoint prescribed by the positions sent from the Magnetic Levitation Haptic Interface.

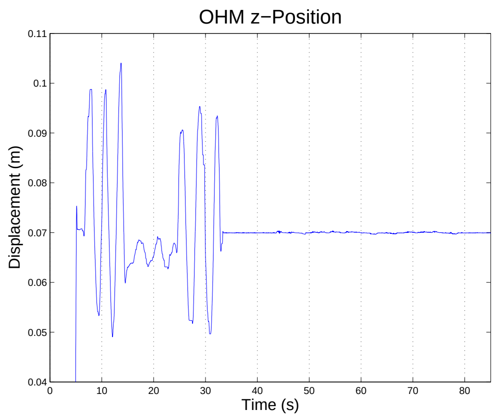
Controlling the OHM with an impedance controller also changed the control strategy used in the first bilateral setup. While the OHM still was servoed to the position of the Magnetic Levitation Haptic Interface, the Magnetic Levitation Haptic Interface now was servoed to the position of the OHM instead of the transformed forces. Whenever a force acted on the hypodermic gripper tube, the new control strategy prescribed the position of the OHM so that a desired force on the sensor was maintained. Forces acting on the hypodermic gripper tube induced a movement of the OHM to the point where this boundary condition held true. This movement in turn induced a movement of the handle of the Magnetic Levitation Haptic Interface. Ultimately this movement induced a force at the hand of the operator of the Magnetic Levitation Haptic Interface. This form of force feedback was saver than the one described before, since the operator was not in control anymore. The results of the impedance control of the OHM can be seen in Figure 20. The friction along the z direction imposed by the ball screw was too high as



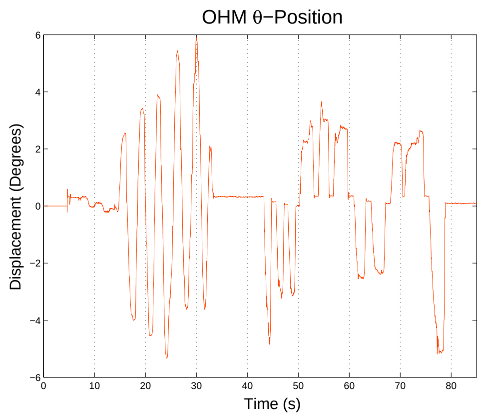
(a)



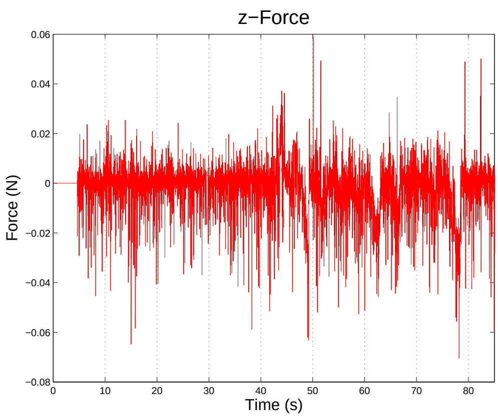
(b)



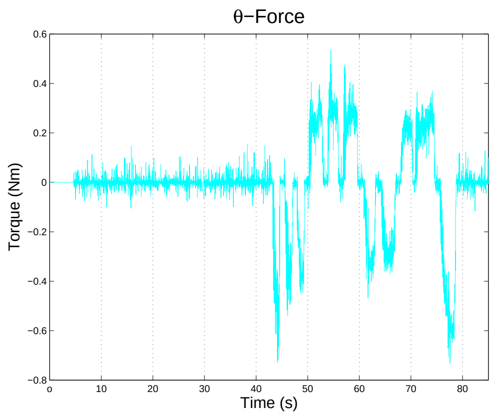
(c)



(d)



(e)



(f)

Figure 20: θ- and z-position for the bilateral control with an impedance controller for the OHM.

Table 3: Parameters for impedance control of the OHM.

$M_{a,z}$	1.361 kg
$M_{a,\theta}$	0.002 kg m^2
$M_{d,z}$	1.361 kg
$M_{d,\theta}$	0.002 kg m^2
$B_{a,z}$	0 N/m/s
$B_{a,\theta}$	0 N m/m/s
$B_{d,z}$	400 N m/m/s
$B_{d,\theta}$	0.002 N m/m/s
f_z	5.0 N
f_θ	0 N
$K_{p,z}$	17500 N/m
$K_{d,z}$	250 N/m/s
$K_{p,\theta}$	4 N m/$^\circ$
$K_{d,\theta}$	0.2 N m/m/s
$F_{d,z}$	0 N
$F_{d,\theta}$	0 N

that the impedance control would have worked in the z direction. With small gains (e.g. $K_p = 100$ and $K_d = 2$), i.e. high compliance, the forces generated in z were too small. As a result the OHM did not move when the hypodermic gripper tube was pushed or pulled in the z direction. When high gains (e.g. $K_p = 17500$ and $K_d = 250$) were applied, the system behaved too stiff. Since high gains allowed better control of the z axis, however, high gains were used in z. Even though the impedance controller was running for both DOF, in z only the PD part was noticeable when the OHM was controlled with the Magnetic Levitation Haptic Interface. The gains and parameters for the last experiment can be read form Table 3.

5 Summary and Contributions

This thesis documents the successful foundation of a platform that can be used to perform teleoperated microassembly. The major accomplishments consist of the combination of the Magnetic Levitation Haptic Interface with minifactory, both developed at CMU; further, the linking of minifactory with the Magnetic Levitation Haptic Interface using the communication libraries of the RHexLib Control Software, developed in part at CMU. Code was developed to enable communication between minifactory and the Magnetic Levitation Haptic interface. Furthermore, control strategies were developed to control the OHM with the Magnetic Levitation Haptic Interface. Force feedback from the OHM to the Magnetic Levitation Haptic Interface was incorporated. The results presented in this thesis are the first experimental confirmation of teleoperation of a minifactory agent. The experimental results convincingly encourage further developments of minifactory to reach teleoperated microassembly.

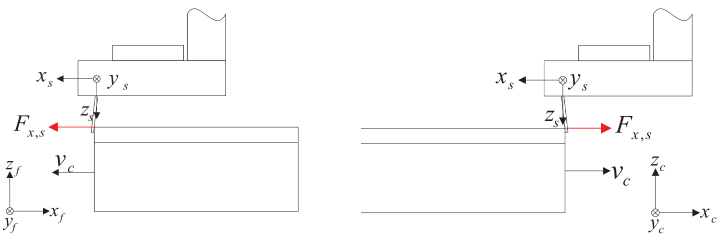
5.1 Future Work

To improve the control of the OHM through the Magnetic Levitation Haptic Interface it would help to lock the “spare” DOFs of the Magnetic Levitation Haptic Interface, that do not introduce any motion in the OHM. Of course once the courier is introduced to the setup the only DOFs that will need to be locked will be ζ_m and ψ_m. The other four DOFs will be needed to control OHM and courier. In order to lock the superfluous DOFs their positions will have to be permanently servoed to their respective starting position. The position controller of the Magnetic Levitation Haptic Interface will need to be set with high gains in these directions to impede deflections and provide high stiffness.

As mentioned earlier there is crosstalk between the different force sensing axes. This problem could be reduced through calibration of the force sensor. Calibration of the force sensor would greatly improve the performance of the OHM when forces act on the hypodermic gripper tube.

The Magnetic Levitation Haptic Interface has a much smaller workspace than the OHM or minifactory in general. A first attempt to account for this difference was to introduce scaling factors. An other possibility to introduce larger motions of the OHM would be to add a velocity controller to the list of controllers with a corresponding domain. Instead of prescribing a desired setpoint, deflecting the Magnetic Levitation Haptic Interface from its origin will induce a velocity in the corresponding direction in the OHM. The induced velocity can be proportional to the deflection of the Magnetic Levitation Haptic Interface handle. This will hold true up to a certain maximum velocity, after which the acceleration will be set to zero, yielding a constant velocity. As mentioned in Section 2, the actuators of the OHM have stops. A safety mechanism would need to be included in order to avoid that the operator pushes the actuators beyond these stops. Like this large motions of the OHM could be controlled with the limited workspace of the Magnetic Levitation Haptic Interface.

Another task for the future would be to directly interface the Magnetic Levitation Haptic Interface with minifactory. Like this the bandwidth limitation imposed by the bridge would be eliminated. This will be especially useful when a new force sensor, with higher sensing bandwidth, is integrated in the OHM.



(a) (b)

Figure 21: Problematic configurations of courier and OHM.

In order to be able to perform teleoperated microassembly with minifactory it is necessary to include the courier into the existing setup. The discrepancy in distribution of DOFs between minifactory and the Magnetic Levitation Haptic Interface presents some problems, however. The DOFs of minifactory are divided between the courier and the OHM. The fact that the minifactory shares its four DOFs among two agents introduces different scenarios, where careful evaluation of the situation needs to take place and influence the controller design, to avoid breaking the force sensor on the OHM. The reason is that the courier does not possess extrinsic means to measure forces, since no force sensor is built into it. During contact tasks where the axis of motion of the courier (either x_c or y_c) aligns with the force sensor's x_s-axis and v_c points in the direction increasing the force on the force sensor, the controller must prevent the courier from breaking the hypodermic gripper tube or the force sensor (Fig. 21). The problem consists of the inability of the end effector to counteract or retract from exposure to the measured force $F_{x,s}$. This problem does not only hold for exactly those points where the axis of motion of the courier and x_s align, but in a region around these points due to frictional effects.

References

[1] P. M. Muir, J. Gowdy, and A. A. Rizzi, "Minifactory: A precision assembly system that adapts to the product life cycle," in *SPIE Symp. on Intelligent Systems and Advanced Manufacturing*, (Pittsburgh, PA), October 1997.

[2] H. Z. Tan, M. A. Srinivasan, B. Eberman, and B. Cheng, *Human Factors For the Design of Force-Reflecting Haptic Interfaces.* DSC-Vol.55-1, Dynamic Systems and Control: The American Society of Mechanical Engineers, 1994.

[3] A. Rizzi and R. Hollis, "Opportunities for increased intelligence and autonomy in robotic systems for manufacturing," in *Robotics Research, the Eighth International Symposium of Robotics Research*, pp. 141 – 151, Springer-Verlag, October 1997.

[4] A. A. Rizzi, J. Gowdy, and R. L. Hollis, "Agile assembly architecture: An agent-based approach to modular precision assembly systems," in *Proc. IEEE Int'l. Conf. on Robotics and Automation*, (Albuquerque, NM), April 1997.

[5] R. L. Hollis and A. Quaid, "An architecture for agile assembly," in *Proc. Am. Soc. of Precision Engineering, 10th Annual Mtg.*, (Austin, TX), October 1995.

[6] J. Gowdy and Z. J. Butler, "An integrated interface tool for the architecture for agile assembly," in *IEEE Int'l. Conf. on Robotics and Automation*, May 1999.

[7] R. L. Hollis and J. Gowdy, "Miniature factories for precision assembly," in *International Workshop on Miniature Factories*, (Tsukuba, Japan), pp. 9 – 14, December 1998.

[8] A. E. Quaid and R. L. Hollis, "3-DOF closed-loop control for planar linear motors," in *Proc. IEEE Int'l Conf. on Robotics and Automation*, (Lueven, Belgium), May 1998.

[9] W.-C. Ma, A. A. Rizzi, and R. L. Hollis, "Optical coordination sensor for precision cooperating robots," in *IEEE International Conference on Robotics and Automation*, vol. 2, (San Francisco, CA), pp. 1621 – 1626, April 2000.

[10] H. B. Brown, P. Muir, A. Rizzi, M. Sensi, and R. Hollis, "A precision manipulator module for assembly in a minifactory environment," in *Proceedings of the 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '01)*, vol. 2, pp. 1030 – 1035, 2001.

[11] A. Quaid and R. L. Hollis, "Cooperative 2-DOF robots for precision assembly," in *Proc. IEEE Int'l Conf. on Robotics and Automation*, (Minneapolis, MN), April 1996.

[12] A. A. Rizzi, J. Gowdy, and R. L. Hollis, "Distributed programming and coordination for agent-based modular automation," in *The Ninth International Symposium of Robotics Research*, (Snowbird, UT), October 1999.

[13] S. Kume and A. Rizzi, "A high-performance network infrastructure and protocols for distributed automation," in *Proceedings of the 2001 IEEE International Conference on Robotics and Automation (ICRA '01)*, vol. 3, pp. 3121 – 3126, May 2001.

[14] J. Gowdy and A. Rizzi, "Programming in the architecture for agile assembly," in *IEEE International Conference on Robotics and Automation*, vol. 4, pp. 3103 – 3108, May 1999.

[15] P. J. Berkelman, Z. J. Butler, and R. L. Hollis, "Design of a hemispherical magnetic levitation haptic interface device," in *ASME International Mechanical Engineering Congress and Exposition*, vol. 58, (Atlanta), pp. 483 – 488, November 1996.

[16] P. Berkelman, R. Hollis, and D. Baraff, "Interaction with a realtime dynamic environment simulation using a magnetic levitation haptic interface device," in *IEEE International Conference on Robotics and Automation*, pp. 3261 – 3266, May 1999.

[17] R. T. DeLuca, "Force-Based Interaction for Distributed Precision Assembly," Master's thesis, Carnegie Mellon University, 2000.

[18] U. Saranli, M. Buehler, and D. E. Koditschek, "Rhex: A simple and highly mobile hexapod robot," in *The International journal of Robotics Research*, vol. 20, pp. 616–631, 2001.

[19] A. Rizzi, "Hybrid control as a method for robot motion programming," in *IEEE Int'l. Conf. on Robotics and Automation*, vol. 1, pp. 832 – 837, May 1998.

A Communication Code

A.1 Endian Swapping Code

The endian swapping code was found on the internet and integrated into the communication setup. The source is shown in the header file as well as the C++ file.

- Header File

```
#ifndef _SWAP_ENDIAN
#define _SWAP_ENDIAN

/***************************************************************
  FUNCTION: SwapEndian
  PURPOSE: Swap the byte order of a structure
  EXAMPLE: float F=123.456;; SWAP_FLOAT(F);
  Source: http://www.gdargaud.net/Hack/Source/SwapEndian.h.txt
****************************************************************/

#define SWAP_SHORT(Var)  Var =  *(short*)\
SwapEndian((void*)&Var, sizeof(short))
#define SWAP_USHORT(Var) Var = *(unsigned short*)\
 SwapEndian((void*)&Var, sizeof(short))
#define SWAP_LONG(Var)   Var =  *(long*)\
         SwapEndian((void*)&Var, sizeof(long))
#define SWAP_ULONG(Var)  Var =  *(unsigned long*)\
SwapEndian((void*)&Var, sizeof(long))
#define SWAP_FLOAT(Var)  Var =  *(float*)\
SwapEndian((void*)&Var, sizeof(float))
#define SWAP_DOUBLE(Var) Var = *(double*)\
 SwapEndian((void*)&Var, sizeof(double))

extern void *SwapEndian(void* Addr, const int Nb);
```

```
#endif
```

- C++ Program

```
#include "swapEndian.h"

/**************************************************************
  FUNCTION: SwapEndian
  PURPOSE: Swap the byte order of a structure
  EXAMPLE: float F=123.456;; SWAP_FLOAT(F);
  SOURCE: http://www.gdargaud.net/Hack/Source/SwapEndian.c.txt
**************************************************************/

void *SwapEndian(void* Addr, const int Nb) {
  static char Swapped[16];
  switch (Nb) {
  case 2: Swapped[0]=*((char*)Addr+1);
    Swapped[1]=*((char*)Addr  );
    break;
  case 4: Swapped[0]=*((char*)Addr+3);
    Swapped[1]=*((char*)Addr+2);
    Swapped[2]=*((char*)Addr+1);
    Swapped[3]=*((char*)Addr  );
    break;
  case 8: Swapped[0]=*((char*)Addr+7);
    Swapped[1]=*((char*)Addr+6);
    Swapped[2]=*((char*)Addr+5);
    Swapped[3]=*((char*)Addr+4);
    Swapped[4]=*((char*)Addr+3);
    Swapped[5]=*((char*)Addr+2);
    Swapped[6]=*((char*)Addr+1);
    Swapped[7]=*((char*)Addr  );
```

```
        break;
      case 16:Swapped[0]=*((char*)Addr+15);
        Swapped[1]=*((char*)Addr+14);
        Swapped[2]=*((char*)Addr+13);
        Swapped[3]=*((char*)Addr+12);
        Swapped[4]=*((char*)Addr+11);
        Swapped[5]=*((char*)Addr+10);
        Swapped[6]=*((char*)Addr+9);
        Swapped[7]=*((char*)Addr+8);
        Swapped[8]=*((char*)Addr+7);
        Swapped[9]=*((char*)Addr+6);
        Swapped[10]=*((char*)Addr+5);
        Swapped[11]=*((char*)Addr+4);
        Swapped[12]=*((char*)Addr+3);
        Swapped[13]=*((char*)Addr+2);
        Swapped[14]=*((char*)Addr+1);
        Swapped[15]=*((char*)Addr  );
        break;
      }
      return (void*)Swapped;
    }
```

A.2 Communication Thread on The Magnetic Levitation Haptic Interface Controller

```
void* mkcom_thread(void* arg)
{
  /*Create a new CommManager that will handle the
    communication between the devices.*/

  CommManager* mkmgr = new CommManager();
  const char* mkspec = "net: int port = 5500";
  if (!mkmgr->initPortal(mkspec)) {
```

```
  fprintf(stderr, "Could not initialize portal %s\n", mkspec);
  exit(-1);
}


/*StreamSink is used to perform a handshake with the minifactory.
  Its sole task is to receive two messages from the minifactory.*/

StreamSink* mksink = mkmgr->createStreamSink(sizeof(com_id_t), 700, 10);
printf("made StreamSink\n");

/*Blocking until the first packet from minifactory was received*/

Message* mkmsg;
mksink->waitData(-1);
mkmsg = mksink->waitData(-1);

/*Waiting for the 2nd message with information of the sender
  of the handshake message*/

if (!mkmsg) {
  perror("Error waiting for box ID\n");
  delete mkmgr;
  exit(-1);
}

com_id_t mkid = mkmsg->getInt();
printf("mailboxID=%d\n",mkid);


/*Create Mailer that will send infromation to the minifactory using the
  received information of the sender*/
```

```
Mailer* mkmailer =
  mkmgr->createMailer(mkmsg->getSource(), sizeof(FloaterPosition), mkid,
  mkmsg->getSource()->getID());


/*Loop that will send the position information to the
  minifactory until the program is terminated.*/


/*getting time for performance test*/
double start_time=getTheTime();
double timeperiode = 30000;


/*MAILBOX TO RECEIVE FORCES FROM PUMA*/
Mailbox* forcebox = mkmgr->createMailbox(sizeof(OHM_Forces), 400);


/*Mailbox that terminates communication between OHM and Haptic Controller*/
Mailbox* comm_kill = mkmgr -> createMailbox(sizeof(int), 300);

/*Message Counter*/
int sent = 0;
int rec_msg = 0;

/*LOOP BEGINS HERE*/

while(1)
  {
    while(1)
{
Message* msg_to_be_sent = mkmailer->createMsg();
FloaterPosition* mkpos = (FloaterPosition*) msg_to_be_sent->getData();
```

```
mkpos->zpos = *(double*)SwapEndian((void*)&sh->fwpos[2], 8);
mkpos->yaw = *(double*)SwapEndian((void*)&sh->fwpos[5], 8);

/*This method returns the maximum size of the message's block of data.*/
msg_to_be_sent->setSize(sizeof(FloaterPosition));

if ( mkmailer->sendMsg(msg_to_be_sent) < 0 ) {
  perror("Failed send\n");
  mkmailer->releaseMsg(msg_to_be_sent);
  break;
}

sent++;

bool new_data = comm_kill->newMail();
if(new_data)
  {
    printf("Sent MSG: %d\n Received Messages: %d\n", sent, rec_msg);
    //exit(1);
    break;
  }

bool new_frc = forcebox->newMail();
if(new_frc)
  {
    Message* rec_frc = forcebox->getData();
    OHM_Forces* frc = (OHM_Forces*) rec_frc->getData();
    //frc->xforce = *(double*)SwapEndian((void*)&frc->xforce, 8);
    //frc->yforce = *(double*)SwapEndian((void*)&frc->yforce, 8);
    frc->zpos = *(double*)SwapEndian((void*)&frc->zpos, 8);
    frc->thetapos = *(double*)SwapEndian((void*)&frc->thetapos, 8);
    frc->zforce = *(double*)SwapEndian((void*)&frc->zforce, 8);
    frc->torque = *(double*)SwapEndian((void*)&frc->torque, 8);
```

```
      sh->des_pos[2]=-(100/6)*(frc->zpos-0.07);
      sh->des_pos[5]=-(frc->thetapos);
      sh->ohm_zpos=frc->zpos;
      sh->ohm_thetapos=frc->thetapos;
      sh->ohm_zforce=frc->zforce;
      sh->ohm_torque=frc->torque;

      forcebox->releaseMsg(rec_frc);
      rec_msg++;
    }
  sem_wait(&haptic_to_mf_Sem);
}

      mksink->waitData(-1);
      mkmsg = mksink->waitData(-1);

      if (!mkmsg)
{
  perror("Error waiting for box ID\n");
  delete mkmgr;
  exit(-1);
}
    }
  //delete mkmgr;
  return NULL;
}

void startmk_com()
{
  pthread_attr_t mkattr;

  sem_init(&haptic_to_mf_Sem,1,0);
```

```
  pthread_attr_init(&mkattr);
  pthread_attr_setdetachstate(&mkattr,PTHREAD_CREATE_DETACHED);
  pthread_attr_setinheritsched(&mkattr,PTHREAD_EXPLICIT_SCHED);
  pthread_attr_setschedpolicy(&mkattr,SCHED_RR);
  mkattr.param.sched_priority = 6; /*Priority of sending thread must always
     be lower than the priority of the haptic
     device position control thread*/
  pthread_create(NULL,&mkattr,mkcom_thread,NULL);
  printf("Started Michael's communication thread\n");
}
```

A.3 Controller Code for the OHM with Impedance Control

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <ohm.h>
#include <ohm_exec.h>
#include <control/control_if.h>
#include <control/coop_force_ctl.h>
#include <aaa_net.h>
#include <util/thread_util.h>
#include <control/control.h>

#include "haptic_ctl.h"
#include "timeUtils.h"
#include "ip_timer.h"
#include "swapEndian.h"
#include "RHexCom.hh"
```

```
#define DEBUG 0

#if DEBUG
#define DPRINT(s) printf s
#else
#define DPRINT(s)
#endif



#ifdef SIGN
#undef SIGN
#endif

#ifdef DEADBAND
#undef DEADBAND
#endif

#define SIGN(a) ((a) >= 0.0 ? 1.0 : -1.0)
#define DEADBAND(a,b) (fabs(a) <= (b) ? (0.0) : (1.0))

static CommManager* mgr;
static Mailbox* box;
static Mailer* forcemailer;
static Mailer* terminate_comm;
static Message* received_msg;
static Message* force_msg;
static StreamSource* source;
static FloaterPosition* dest;
static OHM_Forces* frc;

/*Message Counters*/
int received = 1;
int sent=0;
```

```
void haptic_ctl_init(haptic_spec_t *spec, OHM_state_t *ohm_state)
{
  fprintf(stderr,"Initializing Haptic Control\n");

  if(!mgr)
    {
      mgr = new CommManager();
      box = mgr->createMailbox(sizeof(FloaterPosition), MAILBOX_ID);

      const char* spec_mgr = "net: int port = 5400";
      if (!mgr->initPortal(spec_mgr))
{
  fprintf(stderr, "Could not initialize portal %s\n", spec_mgr);
  exit(-1);
}

      /*Initialize a StreamSource*/
      source =
mgr->createStreamSourceByHost(MASTER, MASTER_PORT,\
sizeof(com_id_t), STREAM_ID);

      /*Create message to be sent with maximum capacity of the mailbox as
specified by CommManager::createStreamSource*/
      Message* msg = source->createMsg();

      /*Send message to remote stream sink*/
      source->sendMsg(msg);

      /*This methode blocks until the sent message msg is no longer on the
queue of messages awaiting acknowledgement*/
      source->ackBlock(0);
```

```
      msg = source->createMsg();
      int mbid =  box->getID();
      printf("boxID = %d\n", mbid);
      int* swap_msg = (int*)msg->getData();
      *swap_msg = *(int*)SwapEndian((void*)&mbid, 4);
      printf("swaped boxID=%d\n", *swap_msg);

      msg->setSize(sizeof(int));
      printf("vlaue of msg = %d", msg->getData());
      source->sendMsg(msg);
      printf("sent msg\n");
      forcemailer = mgr->createMailerByHost(MASTER, MASTER_PORT,\
      sizeof(OHM_Forces), MASTER_FORCE_BOX);
    }
  else
    {
      /*Create message to be sent with maximum capacity of the mailbox as
specified by CommManager::createStreamSource*/
      Message* msg = source->createMsg();

      /*Send message to remote stream sink*/
      source->sendMsg(msg);

      /*This methode blocks until the sent message msg is no longer on the
queue of messages awaiting acknowledgement*/
      source->ackBlock(0);
    }
  spec->axis[0]._integral = 0;
  spec->axis[1]._integral = 0;

  printf("end of hapltic_ctl_init\n");

  return;
```

```
}


void haptic_ctl_eval(haptic_spec_t *spec,
 OHM_state_t *ohm_state, OHM_act_cmd_t *cmd)
{
  int i,r,c;
  static int t=0;
  double dest_pos[2] = {0,0};   /*used to include the force_ctl code*/
  double Chan[4],
    pos,
    vel,
    pos_des,
    kp,
    kd,
    Mact,
    Bact,
    fric,
    Mdes,
    Bdes,
    Fdes,
    new_i,
    f_err,
    command,
    Fe_COUR[3],  /* Z force, X Force, and Y\
           Force in courier coordinates */
    Fe_OHM[3],     /* Z force, Theta Torque, and Radial
            Force in ohm coordinates */
    Fe_GRIP[3],     /* Z force, Tangent Force, and Radial
            Force at gripper tip in ohm coordinates */
    Fe_FS[3] = {0,0,0}; /* Z force, Tangent Moment, and Radial
          Moment in force sensor coordinates */
```

```
  OHM_act_state_t *act_state = &ohm_state->act_state;


  /* Get Load Cell Values */
  for(i=3;i<=6;i++)
    {
      Chan[i-3] = act_state->ad_sens.ad.gripper_ad[i] - spec->no_load[i-3];
    }

  /* Perform Transform to Force/Torques */
  for(r=0;r<3;r++)
    {
      for(c=0;c<4;c++)
{
  Fe_FS[r] += spec->cal[c+4*r]*Chan[c];
}
    }

  Fe_GRIP[0] =  -1.0*Fe_FS[0];
  Fe_GRIP[1] =  -1.0*(1.0/spec->grip_len[0])*Fe_FS[1];
  Fe_GRIP[2] =   1.0*(1.0/spec->grip_len[1])*Fe_FS[2];

  Fe_OHM[0] = Fe_GRIP[0];
  Fe_OHM[1] = Fe_GRIP[1]*spec->grip_loc;
  Fe_OHM[2] = Fe_GRIP[2];

  Fe_COUR[0] =  Fe_GRIP[0];
  Fe_COUR[1] = -Fe_GRIP[2]*cos(act_state->axis[1].pos)+\
      Fe_GRIP[1]*sin(act_state->axis[1].pos);
  Fe_COUR[2] =  Fe_GRIP[1]*cos(act_state->axis[1].pos)+\
      Fe_GRIP[2]*sin(act_state->axis[1].pos);
```

```
  /* poll the mailbox*/
  bool new_data = box->newMail();

  if(new_data)
    { /*new data...update the destination and delete the
       old mail message if there was one*/
      if(received_msg)
{
  box->releaseMsg(received_msg);
}
      received_msg = box->getData();
      dest = (FloaterPosition*) received_msg->getData();
      received++;

      if(received%2==0)
{
  force_msg = forcemailer->createMsg();
  frc = (OHM_Forces*) force_msg->getData();
  frc->zpos = act_state->axis[0].pos;
  frc->thetapos = act_state->axis[1].pos;
  frc->zforce = Fe_OHM[0];
  frc->torque = Fe_OHM[1];
  force_msg->setSize(sizeof(OHM_Forces));
  if (forcemailer->sendMsg(force_msg) == 0)
    {
sent++;
    }
}
    }

  /*if we have a destination from old data, or from newly received
    data, run the control loop*/
```

```
  if(dest)
    {
      dest_pos[0] = 0.07-0.005*(dest->zpos);
      dest_pos[1] = -(dest->yaw);

      for(i=0;i<2;i++)
{
  kp  = spec->axis[i].kp;
  kd  = spec->axis[i].kd;
  pos = act_state->axis[i].pos;
  vel = act_state->axis[i].vel;


  /* Impedance Controller Stuff - Gravity Term Ignored */
  pos_des = dest_pos[i];
  Mact    = spec->axis[i].Mact;
  Bact    = spec->axis[i].Bact;
  fric    = spec->axis[i].fric;
  Mdes    = spec->axis[i].Mdes;
  Bdes    = spec->axis[i].Bdes;
  Fdes    = spec->axis[i].Fdes;

  /* Intergral Term Stuff */
  f_err = (Fe_OHM[i]-Fdes);
  new_i = (spec->axis[i]._integral + CTL_PERf * spec->axis[i].ki * f_err);

  if(new_i > spec->axis[i].i_limit[0])
    new_i = spec->axis[i].i_limit[0];
  else if(new_i < -spec->axis[i].i_limit[1])
    new_i = -spec->axis[i].i_limit[1];
  spec->axis[i]._integral = new_i;

  command = Mact*(1.0/Mdes)*(kp*(pos_des - pos) -\
```

```
         Bdes*vel + new_i - Fdes) + Bact*vel;/*1.5*Fdes) + Bact*vel;*/

  cmd->force[i] = Mact*(1.0/Mdes)*(Fe_OHM[i] + kp*(pos_des - pos) -\
     Bdes*vel + new_i) + Fe_OHM[i] + Bact*vel +\
  fric*SIGN(command);


  if (cmd->force[i] > spec->axis[i].limit)
    cmd->force[i] = spec->axis[i].limit;
  else if (cmd->force[i] < -spec->axis[i].limit)
    cmd->force[i] = -spec->axis[i].limit;
}
    }
  return;
}

void haptic_ctl_finish(haptic_spec_t *spec, OHM_state_t *ohm_state)
{
  /*send 20 message to the haptic device to
     tell it to stop sending its position infromation*/
  terminate_comm = mgr->createMailerByHost(MASTER,\
          MASTER_PORT,sizeof(int), MASTER_BOX);
  int kill=1;
  for(int m=0;m<20;m++)
    {
      if(terminate_comm->sendInt(kill)<0)
{
  printf("Mailbox invalidated\n");
}
      printf("kill = %d\n", kill);
      kill++;
    }
  fprintf(stderr,"finish Haptic Control\n");
```

```
  printf("Received Messages: %d\n Sent Messages: %d\n", received, sent);
}
```

B Running and Terminating the Demos

To run the Demos which allow the control of minifactory with the Magnetic Levitation Haptic Interface the following steps need to be performed:

1. Check that vacuum in minifactory is on and turn on puma.
2. Make sure that the amplifiers on the Magnetic Levitation Haptic Interface are off yet the power is on. Both metal switches must be in on position, the black switch must be off.
3. Open four terminals on a lab computer (Fig. 22)

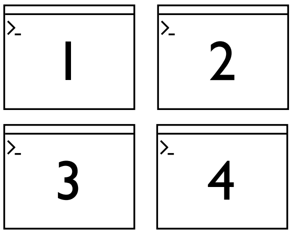


Figure 22: Terminals needed to run Demos.

4. Telnet to puma in terminal 1 and 2:
   `telnet kummermi@puma.msl.ri.cmu.edu` ↩
   password: `____________` ↩
5. If you are not running the demos from Dulcian, ssh into Dulcian in terminal 3 and 4.
   `ssh kummermi@dulcian.msl.ri.cmu.edu` ↩
   password: `____________` ↩
6. In terminals 1 and 2 change directory to:
   `cd ~kummermi/haptic_control/impedance_small_sf/` ↩
7. In terminal 3 change directory to:
   `cd ~kummermi/teleop_src_wdir/maglev_to_minifactory/` ↩

8. In terminal 4 change directory to:
   `cd ~bertram/frozen/drivers/`

9. You will need to run the drivers for the Magnetic Levitation Haptic Interface in terminal 4. In order to be able to do that you will need root access:

   - `su` ↩
   - password: ____________ ↩
   - Run drivers:
     `./apc8620_drv` ↩

10. In terminal 3 run the control and communication software for the Magnetic Levitation Haptic Interface and get the flotor to take off:
    `./Test` ↩
    If `Ok to switch on amps if no further warnings` gets printed on screen, carefully turn on the amplifiers (black switch) of the Magnetic Levitation Haptic Interface. Then hit
    `t` ↩
    and the flotor will take off.

11. In terminal 1 run ohm_exec:
    `./ohm_exec` ↩

12. In terminal 2 run haptic_test with the required parameters for the gains:
    `./haptic_test 17500 250 4 0.2` ↩

You should be able to control the OHM with minifactory now and get some force feedback from the OHM to minifactory. To play around with the demo you can adjust the gains in the last step. This demo is the one with the impedance controller on the OHM. To run the demo with the PD controller with small scaling factors two things need to be changed:

- In step 6 you need to change the directory to:
  `cd ~kummermi/haptic_control/pd_small_sf/` ↩
- haptic_test will need to be run with different gains:
  `./haptic_test 17500 250 60 0.65` ↩

To shut the demo down the following steps need to be performed:

1. Quit the haptic_test application in terminal 2 by hitting ctrl-c.
2. Land the Magnetic Levitation Haptic Interface in terminal 3:
   `l` ↩
3. Turn off the amplifiers of the Magnetic Levitation Haptic Interface once the flotor landed.
4. Quit ohm_exec in terminal 1 by hitting ctrl-c.
5. Exit the application in terminal 3 (`./Test`) by hitting ctrl-c.
6. Run Test again in terminal 3:
   `./Test` ↩
   It will not start up properly. Kill it again (ctrl-c).
7. Finally terminate the drivers in terminal 4 by hitting ctrl-c.

This little inconvenience of killing and restarting the Test application is necessary, because if Test is not restarted once before terminating the drivers, Dulcian will freeze. This will make it necessary to reboot Dulcian.

C Building Your Own Copy

If you want to change the demo and expand it by building stuff on top of the existing setup, it is recommended to build the necessary software directories in your home directories on the different computers. The executables running on the OHM are not compiled on the OHM brain boxes. All the software running on the brain boxes is compiled using a cross compiler on Trombone. In order to be able to build new executables for new demos you will need an account on Dulcian (Magnetic Levitation Haptic Interface Controller), Trombone, and the brain boxes of all involved agents. On the included CD the necessary directories for the different computers are provided as archives. Just untar the different files (`tar -xvf *.tar`) in your home directory on the corresponding computer. The Flute directories are not relevant for the demo. The directory you will need in your Dulcian home directory is:
`~/teleop_src_wdir`
and all of its subdirectories. In the subdirectories:
`~/teleop_src_wdir/maglev_to_minifactory`
`~/teleop_src_wdir/include_michael`
you will find all of the files that allow to modify the behavior of the Magnetic Levitation Haptic Interface in the teleoperation setup. All the important header files are in the `~/teleop_src_wdir/include_michael` directory. All the relevant C/C++ code is in `~/teleop_src_wdir/maglev_to_minifactory`. The most important files here are:

- `maglev.c`
- `masterComm.cc`
- `mailHandler.cc`
- `commHandler.cc`
- `commCommand.cc`

Since the communication of minifactory with the Magnetic Levitation Haptic Interface utilizes the RHexLib communication libraries, it is necessary to have a working copy of the RHexLib software. It should be no problem to build the RHexLib control software on Dulcian. Just get a copy and build it in your home directory. On Trombone, however, building RHexLib is more difficult. For this reason you will need to link against a copy containing the necessary communication components of RHexLib in Al Rizzi's directory. This can be seen in the different Makefiles in `~/mini/agent/ohm-exec/` and its subdirectories. Whatever hard links you will encounter in the different Makefiles should be left as is. To understand best how the compilation process works on Dulcian check the Makefile in `~/teleop_src_wdir/maglev_to_minifactory/`.

To build or modify existing controllers for the OHM the two most important files are `~/mini/agent/ohm-exec/control/control.c` (add controller functions to `ctl_table`) and `~/mini/agent/ohm-exec/control/control_if.h` (add or edit existing controller structure and add it to `ctl_type` and `ctl_spec_t`). Your new controller should contain an initialization function an execution function and a terminating function (example: `~/mini/agent/ohm-exec/control/haptic_ctl.c`).

A lot of other files were changed in order to get everything to compile with g++. All these changes can be found in the files when searching them for `Kummer`.