Diploma Thesis

Pick and Place in a Minifactory environment

Cornelius Niemeyer

Prof. Dr. Ralph L. Hollis
The Robotics Institute
Carnegie Mellon University (CMU)
Adviser

Prof. Dr. Bradley J. Nelson
Institute of Robotics and Intelligent Systems
Swiss Federal Institute of Technology Zurich (ETH)

2006-12

Institute of Robotics and Intelligent Systems

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Preface

I would like to thank Prof. Ralph Hollis for his guidance, his support and the opportunity to write this thesis at the Microdynamic Systems Laboratory and Prof. Bradley Nelson for his support and the opportunity to go overseas for the work on this diploma thesis. Many thanks go to Christoph Bergler for his good spirit, invaluable help and the good collaboration as my lab mate as well as Mark Dzmura, Jacob Thomas, Mike Cozza, Jay Gowdy and Bertram Unger for their contribution to the Minifactory project, their explanations and help.

Abstract

This report describes the realization of vision and non-vision based pick and place operations for cooperating low degree of freedom robotic agents in the Minifactory environment developed at the Microdynamic Systems Laboratory (MSL) at Carnegie Mellon University (CMU). The Minifactory is a flexible and agile assembly system consisting of independent robotic modules. Procedures and robot programs were developed to perform a precision assembly of a telescopic sight. Furthermore, as an abstract model for general pick and place application needs, micro scale spheres as model parts have been chosen. Vision based pick and place of these spheres has been successfully demonstrated. For that purpose means to connect and control a preexisting tweezer gripper were conceived. Vision processing has been implemented in an extensible vision server application integrated into the Minifactory software architecture. It is able to provide results to client programs running on real or simulated robots. The recognition and locating of the spheres in images grabbed from a video feed was realized using an ellipse fitting algorithm.

Zusammenfassung

Diese Arbeit beschreibt die Realisierung von “pick and place” Operationen für kooperierende Roboter in der Minifactory. Die Minifactory ist eine flexible und schnell anpassbare Montageanlage, die aus von einander unabhängigen Robotermodulen besteht. Sie wird am Microdynamic Systems Laboratory (MSL) an der Carnegie Mellon University (CMU) entwickelt. Im Rahmen dieser Arbeit wurden Verfahren und Roboterprogramme für die Montage eines Zielfernrohrs erarbeitet. Ausserdem wurden kleine Kugeln mit Durchmessern im Mikrobereich als Modellbauteil für generelle “pick and place” Anwendungsfälle ausgewählt. Das Aufnehmen und gezielte Ablegen dieser Kugeln unter Verwendung von Bilddaten wurde erfolgreich demonstriert. In diesem Zusammenhang wurde die Anbindung und Steuerung für einen schon bestehenden Greifmechanismus erarbeitet. Die Bildverarbeitung wurde dabei in einer erweiterbaren, in die Softwarearchitektur der Minifactory integrierten, Serverapplikation implementiert. Diese stellt Bildverarbeitungsresultate Client-Programmen zur Verfügung, welche sowohl auf realen als auch auf simulierten Robotern laufen können. Die Erkennung und Lokalisierung der kleinen Kugeln auf aus einem Videosignal gewonnenen Bildern wurde mit Hilfe eines “ellipse fitting”-Algorithmus realisiert.

Contents

List of Tables

List of Figures

1 Introduction

Today's high tech industries are confronted with ever shorter product life cycles, rapidly changing market demands, a drive towards more and more miniaturization and integration as well as the demand for more product variants and search for individuality. This concerns fashionable articles and gadgets as well as e.g. MEMS or optoelectronic devices. The *Agile Assembly Architecture (AAA)* philosophy developed at the Microdynamic Systems Laboratory (MSL) at Carnegie Mellon University (CMU) and its physical instantiation in the robotic *Minifactory* aim to meet these requirements.

The Minifactory undergoes a constant process of development. Two projects were carried out to enable pick and place operations which are an integral part of a product assembly. The first task was to implement the precision assembly of a telescopic sight carried out by multiple cooperating robotic agents. Lenses of different sizes and shapes had to be assembled in a collimator housing and then be fixed in place with screwed in retaining rings. Several people worked on the realization of this task. This report covers the author's contribution, namely the implementation of programs for robotic agents and implemented modifications on project associated hardware.
The second task was to perform vision based pick and place of small spheres with a diameter of 300 μm using two cooperating robots. This included finding means to connect and control a previously designed tweezer gripping mechanism. Furthermore image processing to locate spheres had to be implemented and a place for the image processing unit in the complex Minifactory software architecture needed to be determined.

This report is structured in three main chapters. An introduction to the Minifactory environment is given in Chapter 2. The realization of the telescopic sight assembly is presented in Chapter 3. The vision based pick and place of spheres is discussed in Chapter 4. Important source code and technical data can be found in the appendix. Further material is included on an enclosed DVD.

2 The Agile Assembly Architecture

The Agile Assembly Architecture (AAA) [6], [15] is a philosophy conceived at the Microdynamic Systems Laboratory (MSL) at Carnegie Mellon University (CMU) to meet the modern day requirements of the assembly part of product manufacturing. It aims not only to be *flexible* in terms of being capable to cope with variability in the manufacturing process but also for *agility*, which is the ability to adapt to a rapidly changing product market. Thus its goals are to drastically reduce the assembly system's design and changeover times as well as to enable its geographically distributed design and rapid deployment. An AAA assembly system consists of tightly integrated and computationally independent robotic modules knowing about their own capabilities and capable of flexible cooperation with their peers [12], [8]. The modules can provide models of their geometry as well as their behavior for simulation and monitoring in a unified interface tool [3]. Assembly systems can be designed, programmed [4] and tested in a simulated factory environment using the interface tool and working with actual module specifications loaded remotely over the internet. Standardized protocols and procedures as well as structured robotic agent autonomy simplify the design and programming task. Once ordered modules have arrived and have been assembled, the modules ability to self calibrate and explore their environment is exploited to enable a smooth transition from simulated to real assembly.

2.1 Minifactory

The Minifactory developed at MSL is a physical Agile Assembly Architecture instantiation as a tabletop factory [11],[5]. It is shown in Figure 1. It combines low degree of freedom (DOF) robotic modules, referred to as *agents*, to perform a cooperative task consisting of integrated part transportation and up to 5 DOF assembly. The use of cooperative robots with two or three degrees of freedom enables independent high precision movements in any direction as opposed to e.g. the precision loss in robots with long kinematic chains such as a SCARA robot. Low-DOF robots can be designed with low masses and thus allow higher accelerations and speeds to reduce operation times. Furthermore the use of multiple cooperating agents allows parallelization and pipelining of assembly processes.

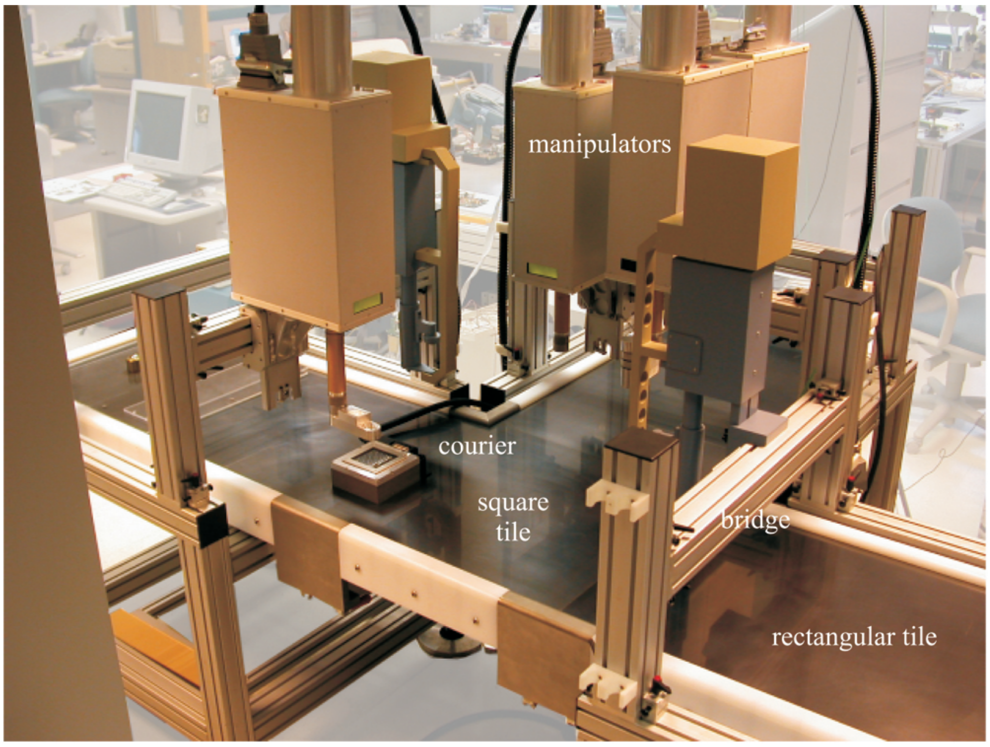


Figure 1: T-shaped Minifactory at MSL.

A Minifactory consists of different modules such as base frames, base units, platen tiles, bridges and robotic agents that enable a multitude of different configurations and swift set-up. An aluminum profile base frame forms the structural support of the factory comprising the so called *base unit*, a service module that can supply connected agents with power, network connection, pressured air and vacuum. At the top of the base frame a *platen* tile is mounted that represents the factory floor (see Figure 2). A platen consists of a grid of ferromagnetic posts on a 1mm pitch, surrounded by epoxy to obtain a flat surface. Polyethylene curbs are attached at the platen edges. Adjustable structural bridges can be clamped to the base frame to support robotic agents mounted above the platen. Further agents can be attached to the sides of the base frame. The two types of robotic agents used in this project are the overhead manipulator and courier agents. Both will be presented in detail in the following sections. A global 100Mbit network using standard IP protocols and a local 100Mbit network, specially adapted for low latency and real-time capabilities referred to as AAA-Net [9] enable high performance agent communication.

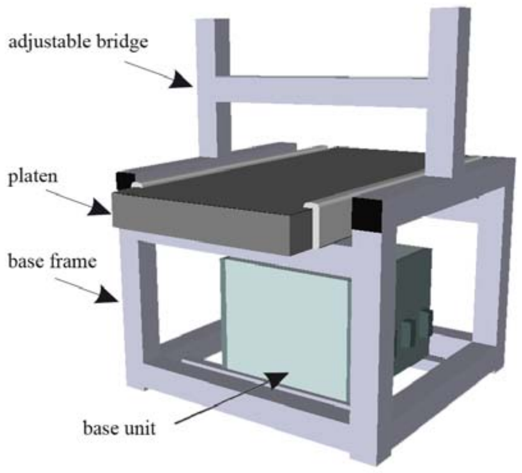


Figure 2: Minifactory core module.

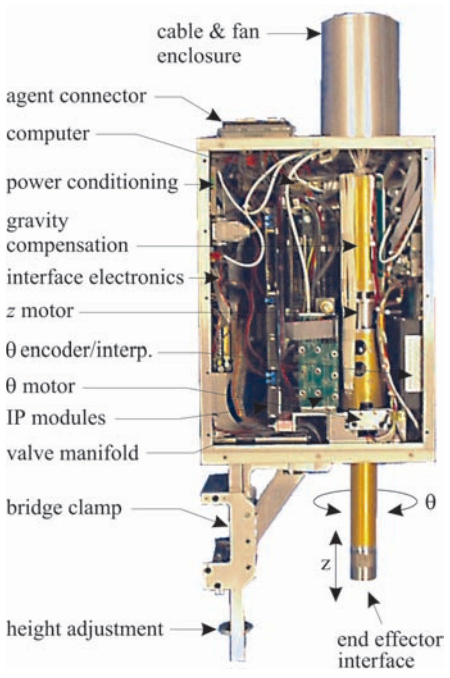
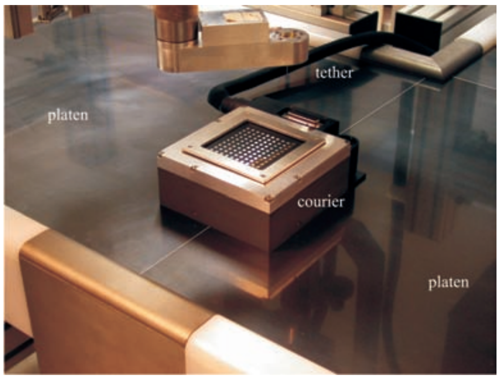


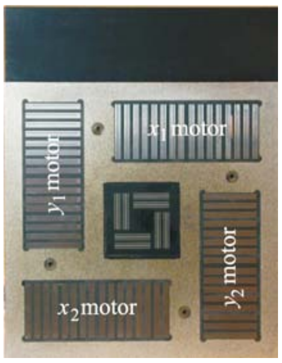
Figure 3: Manipulator agent with removed cover.

2.2 Overhead Manipulator

The overhead manipulator (OHM) agent, shown in Figure 3 is a two DOF robot able to move vertically along a Z axis and to rotate in θ around Z. Manipulators are clamped to bridges above the platen performing their task in a working area in between them and the platen. Figure 1 shows mounted manipulators. The range of motion in Z is about 150 mm with a resolution of 5 µm and 570 ° in θ with a resolution of 0.0002 ° (1σ) [1]. Different end effectors can be connected to an electrical and pneumatic interface, providing pressured air and vacuum as well as signal and video transmission to resp. from the end effector. The manipulators computational unit has recently been updated to an embedded Intel Celeron M system running the QNX 6.3 real time operating system. It is equipped with an analog frame grabber card for vision processing, an Acromag APC-8620 PCI carrier card with Industry Pack (IP) modules that interface to the actuators and sensors and two network connections. Files are stored on a 1 Gb compact flash memory card. All mechanics, electronics and computing hardware are tightly integrated in the so called *brainbox* making the manipulator easy to handle and move around. The manipulator design is explained in detail in [1]



(a) Courier agent on platen



(b) Bottom side of courier agent

Figure 4: Courier agent.

2.3 Courier

Courier agents as shown in Figure 4 glide in two DOF on air bearings at an altitude of $10 - 15\,\mu\text{m}$ [8] over the platen factory floor. They can transport products as well as cooperate with the manipulator in an assembly operation. To move in X and Y the courier is equipped with four planar stepper motors exploiting the Sawyer principle arranged as shown in Figure 4 b). The grid of ferromagnetic teeth in the platen provides the electromagnetic reaction forces for the courier motors. Position references extracted by a *platen sensor* enable closed loop control at a position resolution of $0.2\,\mu\text{m}$ (1σ) at a speed of $1.5\,\text{m/s}$ [13]. Furthermore the courier is equipped with an optical coordination sensor able to detect and measure the relative distance to LED beacons attached to manipulator end effectors with a resolution of $0.15\,\mu\text{m}$ (1σ) [10] at a bandwidth of 100Hz. The courier is connected to its *brainbox* by a tether whose length constitutes the only limit of its area of operation on the platen. The *brainbox* contains all electrical and computing hardware and can be clamped to the side of the Minifactory base frame. The computing hardware is the same as for the manipulator with the exception of the frame grabber. Task specific appliances such as e.g. a fixture to safely transport products can be fixed to the courier's upper side.

2.4 Interface tool

Minifactory's core software is the interface tool described in detail in [3]. Using the interface tool one can design an assembly system in a 3D virtual factory environment as shown in Figure 5. Agent programs and assembly procedures can be elaborated in an iterative process by testing them in simulation. Once connected to real agents, the interface tool can download the programs to them and launch the assembly operations. While the assembly is executed the agents continuously send status information to the interface tool enabling it to visualize what is happening. Thus the interface tool also serves as a monitoring device when the real factory is running. Furthermore the interface tool is capable to combine simulation and monitoring by e.g. simulating some agents that are not connected and monitoring others that are actually running their programs.

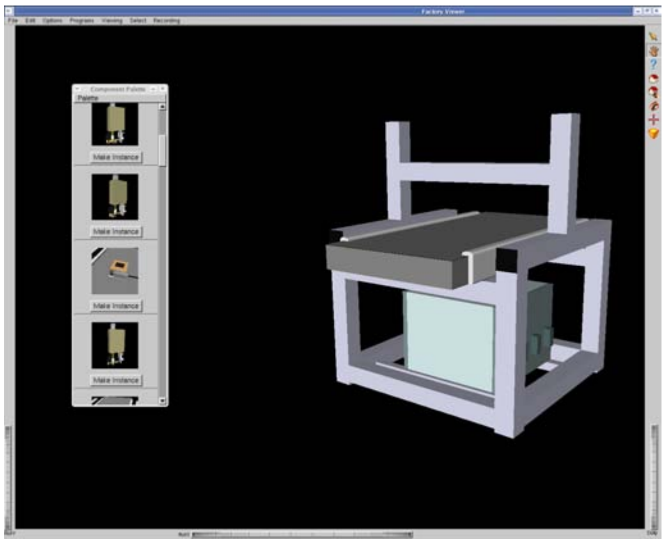


Figure 5: Designing a factory in the interface tool.

2.5 Programming Minifactory agents

There is no central program or control unit running Minifactory. Instead agent based distributed programming is used [16], [4]. Agents run their own program and communicate directly with other agents, providing information about themselves (as e.g. their current position, direction and speed) as well as sending instructions and negotiating actions in order to perform coordinated movements and complete cooperative tasks. High level programming is done in Python scripts defining the agent's behavior in different factory operations and at discrete events. Factory wide standardized procedures involving multiple agents are also implemented on the Python level. The parameterization, sequencing and execution of low level control strategies to e.g. perform movements is implemented in low level C and C++ programs. Thus two C/C++ programs run on every agent. One is a *head* program running and executing the Python script in a special thread while at the same time handling the tasks of communicating with other agents. The second is a *real time executive* handling everything hardware associated such as handling sensor data and controlling movement in real time. Minifactory agents use the hybrid control model elaborated in [14] to adaptively determine the order of execution of different low level control strategies called *controllers*. For the manipulator agent there are e.g. the following important control strategies:

MoveTo Move the end effector to a specific position

StopAt Move the end effector to a specific position close by with very high precision.

GraspAt Move the end effector to a specific position such that an item can be grasped by the end effector

Controllers are associated with a target position and a Boolean function called *predicate*. A predicate can depend on the agent's position or on other different sensor values. For instance a predicate can be defined to return true when the manipulators position on the Z axis is in the region of -10 to -50 and otherwise false. Controllers are inserted at a position in an *actionlist*, a generated list of controllers. The *real time executive* handles this *actionlist* and periodically scans it beginning at the top. It evaluates the predicate functions and the first controller in the list whose predicate is true will be executed. Thus a controller in the list is executed as long as it's predicate remains true and no predicate of a controller higher in the list returns true. The use of the hybrid control method to program Minifactory agents is at times complex and unintuitive. However if one makes proper use of its capabilities very powerful and adaptive cooperative agent behaviors can result.

2.6 Setting up a working system

Prior to the start of this thesis an effort to update and refit the Minifactory agents was begun. Among other things the computational hardware in all Minifactory agents was changed to embedded Intel Celeron M systems and the operating system was switched from LynxOS to QNX which required the porting of all agent software. The interface tool previously running on a Silicon Graphics machine was moved and ported to a new X86 PC computer running on Linux. These modifications took however more time than planned and were not completely finished by the start of this project. Thus a significant amount of time had to be spent to solve several problems, remove bugs and to set up at least a partly working system. Some issues could not be dealt with to the end although a lot of progress was made. For the course of this thesis the manipulators had several hardware problems including e.g. Z axis position errors as large as 1.5 mm. Furthermore the courier platen position sensor as well as the optical coordination sensor were not yet working again. Thus only open loop dead reckoning courier movements and no automated factory calibration could be performed.

3 OSTI-Project: Assembly of a telescopic sight

3.1 Task analysis

The idea behind this project is to reassure and demonstrate the refitted Minifactory system capabilities. The overall task is to implement a precision assembly of a telescopic sight from Optical Systems Technology, Inc., Kittaning, PA, USA (OSTI). Several lenses of different thicknesses and diameters have to be set and glued into a tight-fitting collimator housing. Each lens has to be secured by a retaining ring which is screwed into the housing (see Figure 6). As the glue dispensing agent was still being developed and not operational only a “dry” assembly was attempted. The overall task was split up in different smaller entities such as

1. Mechanical design of an end-effector and tools to pick up and screw in resp. put down items.

2. Generating a virtual Factory simulating and visualizing the assembly pro-

cess

3. Programming the actual pick, place and screwing operations to be performed by the factory's agents

4. Merging all of the above to a working assembly

Several people worked on this project. The author's contribution to task number 4 and the solution to his assigned subtask number 3 will be presented in the following sections.

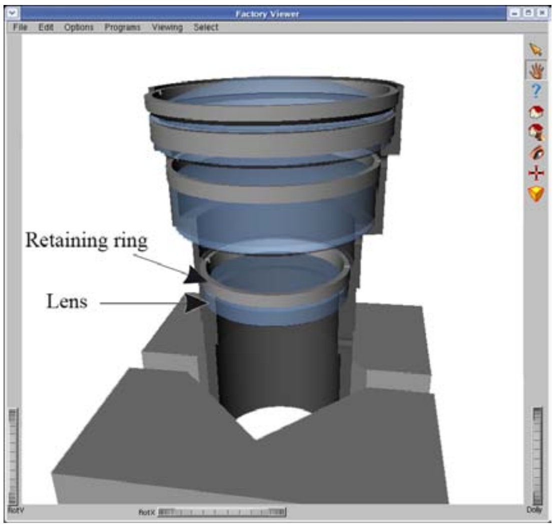


Figure 6: Section of collimator housing with assembled lenses and retaining rings.

3.2 Hardware

3.2.1 End effector and tools

As the items to be assembled have different geometric proportions a special tool was designed for each lens and retaining ring. The tools bottom side is adapted to mate with the picked up and placed item whereas the top is standardized and can be attached to a single pick and place end effector. The designs of the different tools and the end effector were provided and are described in detail in [7]. The end effector is shown in Figure 7 (a). It has two pneumatic channels which the manipulator can connect to pressured air or vacuum. If a tool is mated with the bottom side of the end effector a small chamber is created. The centered channel leads vacuum or air to that chamber and thus sucks at the tool to keep it in place or pushes it away. The second channel is continued through the tool to the part. In the case of a lens tool as shown in Figure 7 (*c*) it leads to a chamber formed by the tool and the lens. Using the same principle as before, the lens can be hold in place or pushed away. In the case of a ring tool, as shown in Figure 7 (*b*), the second channel leads to a ring of very small holes at the connecting surface between ring and ring tool. Thus the sucking vacuum or the air can be applied to the ring. Furthermore the ring tool has two small metal bars fitting into two small slots at the top of each ring. They enable the tools to transmit torque to the rings and thus the screwing of the rings into the collimator housing.

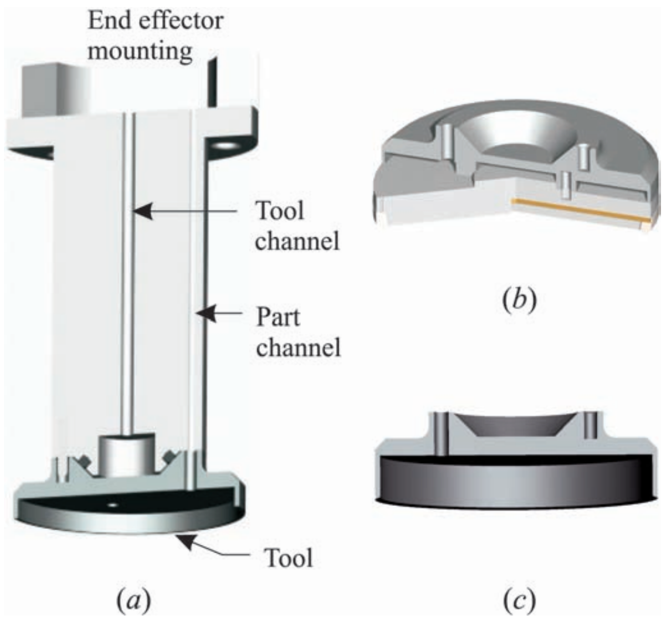
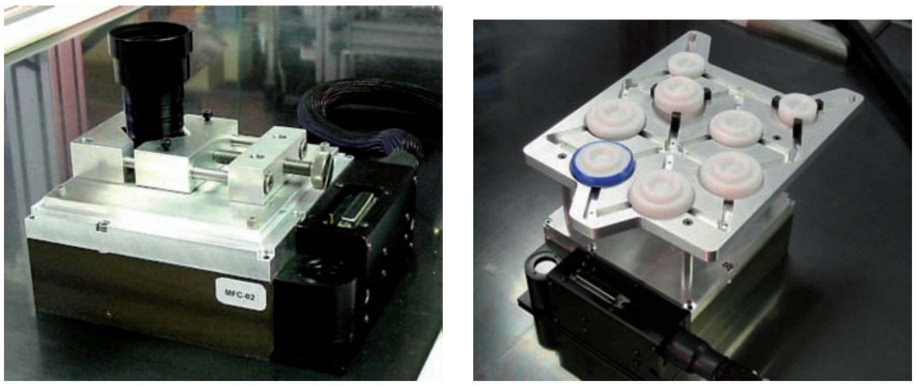


Figure 7: OSTI end effector, ring and lens tools (sections).

3.2.2 Couriers

The collimator housing in which all parts have to be assembled is mounted on the *assembly courier* and the parts are stored in nests on a pallet fixed to the *parts courier* as shown in Figure 8. The tools are placed directly on top of the parts so that the manipulator can pick up both at the same time.



(a) Assembly courier (b) Parts courier

Figure 8: OSTI assembly and parts couriers.

3.3 Factory programming

The overall assembly process can be decomposed in similar subunits for each part to assemble. Each unit consists of the pick up of the part and the corresponding tool from the parts courier, the assembly operation on the assembly courier and the deposing of the tool on the parts courier. Since there is an individual tool for every part that is assembled, the used tool is not needed anymore. It is replaced in the nest where the now assembled part was previously stored (see Figure 8 (*b*)).

In order to perform an interaction between a courier and the manipulator such as placing an item in a defined position on a courier or picking up an item, the courier and the manipulator have to meet in such a configuration that the (target) position of the item on the courier lies directly beneath the Z axis of the manipulator as shown in Figure 9. This is achieved by using provided coordination functions. The courier initiates a so called *RendezVous*, announcing the future cooperation. The manipulator accepts the *RendezVous* if it is currently unoccupied and a “connection” between the two agents is established. Using the command *coordinateTo* the courier “enslaves” the manipulator and tells him to cooperate in a coordinated move towards the desired relative position of the two. Once in position the manipulator can perform a pickup, place or screwing operation. In order to simplify the programming, these operations have been implemented in such a way that no further courier movements are needed until the operation is complete. The different operations have been implemented in individual Python functions running on the manipulator agent. They will be explained in detail in the following sections. The sequence of actions performed by the courier and manipulator agents during the assembly of a lens or a ring is listed in a schematic way in Table 1.

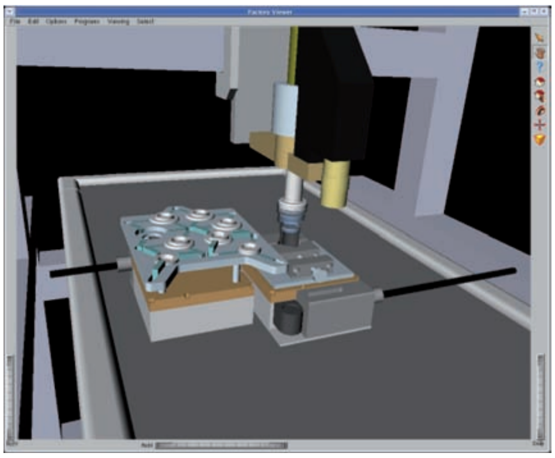


Figure 9: Scene from virtual OSTI factory.

Table 1: Agent actions in assembly operation.

Parts courier (C2)	**Manipulator**	**Assembly courier (C1)**
`initiate RendezVous`		`initiate RendezVous`
	`accept Rendezvous from C2`	
`move in position`	`move in position`	
	`perform pick up`	
`move away`		
`finish RendezVous`	`finish RendezVous`	
	`accept RendezVous from C1`	
	`move in position`	`move in position`
	`perform place/screwing`	
		`move away`
	`finish RendezVous`	`finish RendezVous`
`initiate RendezVous`		
	`accept RendezVous from C2`	
`move in position`	`move in position`	
	`place back tool`	
`move away`		
`finish RendezVous`	`finish RendezVous`	

3.4 Picking up

3.4.1 Pick up software module

The purpose of this function is to make the manipulator end effector move to a specified Z coordinate *target_Z* and pick up a tool or a part and a tool. The lifting is done by applying vacuum to the pneumatic channels in question, thus sucking at the tool and part. The function submits different controllers with different predicates (see Section 2.5) to the manipulator's action list. Since the execution of controllers in the action list is not linear but condition based, the source code looks rather different from the actual sequence of actions resp. movements of the manipulator which makes everything rather complex. To simplify things the generated action list and the resulting actions/movements of the manipulator will be presented in the following. The corresponding Python code can be found in Appendix A.1.

Table 2: Manipulator action lists: pick up function

ID	Description	Predicate	Channel	Flags
3	StopAt *target_Z* + offset	$p \leq graspPressure$	sucking air	"Got part"
4	Move to *target_Z* + offset	$p \leq graspPressure$	sucking air	
1	GraspAt *target_Z*	at *target_Z*	sucking air	
2	Move to *target_Z*		sucking air	

The manipulator agent's action list for the during the execution of the pick up function is shown in Table 2. For timing reasons the order of insertion of the different controllers, which is represented by their ID, is different from their order in the action list. From bottom to top the different controllers have predicates that are only fulfilled if the previous execution of a controller lower in the list has brought the desired results. For instance the predicate condition for the controller with the *ID* 4, moving the end effector upwards from *target_Z* about a given *Offset*, is that the air pressure on the pneumatic lines in question has dropped below 30 kPa. This condition is only met if the controller one down in the list with the *ID* 1 has successfully made the end effector mate with the tool such that the connection is sealing enough to establish a low enough pressure. Thus the different controllers in the action list are executed from bottom to top, as long as everything goes well, going down in the list again when it does not. The behavior of the manipulator for the case where everything goes as desired is listed in Table 3. The function returns if the "Got Part" flag is emitted or when a time limit is reached to avoid infinite unsuccessful trying.

Table 3: Manipulator behavior in successful pickup (top to bottom).

Performed action	**Status of pneumatic channel**
`move to` *target_Z* `(ID 2)`	`starting to suck air`
`grasp item at` *target_Z* `(ID 1)`	`sucking air`
`move to` *target_Z* + `offset (ID 4)`	`close to vacuum`
`stop at` *target_Z* + `offset (ID 3)`	`vacuum`

3.4.2 Hardware modifications

In order to pick a tool and lens up successfully, the connection between the end effector and the tool must be sealing well enough to generate a low enough air pressure in the pneumatic channel. For that seal to be established, the upper face of the tool and the lower face of the end effector (see Figure 7 in Section 3.2.1) must be coplanar. The plane of the face on the tool is mainly defined by the plane of the Minifactory platen with added influences of manufacturing inaccuracies resp. tolerances on parts of the courier, the parts palette and the tool itself. The plane on the end effector is mainly defined by the mounting of the whole manipulator to the mounting bridges above the platen with added influences of manufacturing inaccuracies resp. tolerances of the mounting bridge, the manipulator and the end effector itself. This number of different influences is so great that assuring the right tolerances on all different parts would be costly and intricate. Therefore it was decided to integrate an element of compliance which lets the tool and/or end effector move in the correct respective position. The compliant element could be at the link of the palette and the courier, at the link of courier and part and at the link of end effector and manipulator. For reasons of simplicity and the low alteration effort it was chosen to put some compliant material between the parts and the palette in the nests. Test series of different foams and rubber types returned a ring of ordinary rubber glued into the nests as best solution as shown in Figure 10. In addition to that the tightly designed nests had to be widened to give the part and tool room to move a little bit.

Picking up the different retaining rings by sucking them onto the tool could not be achieved after several alternations to the ring tools, such as e.g. widening

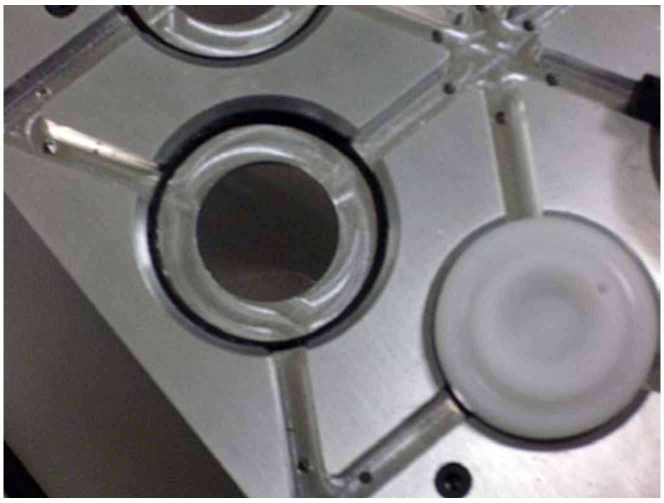


Figure 10: Nest on parts courier with compliant rubber material.

the small air holes on the contact surface of the tool. The contact surface of the ring proved to be too small and not plane enough to generate a sealed connection of tool and ring. In addition to that, the sucking air flow slowed down due to the small diameters of the holes. As a result the force that could be generated on the ring was not large enough to lift it up. The solution developed is based on holding the ring in place on the tool by friction. The diameter of the lower part of the tool shown in Figure 11 which mates with the inner walls of the ring as is slightly increased to tighten the fit such that the ring stays on. The ring is mounted to the ring tool before placing them in the nest on the parts courier. When picking up the ring tool the ring stays on and after one full turn of screwing the thread holds the ring in place on the collimator housing such that it is stripped off when the end effector and tool are lifted up.

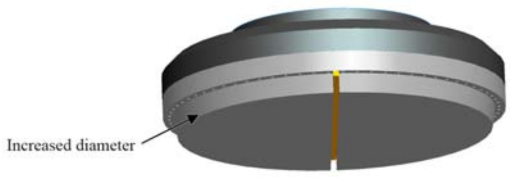


Figure 11: Modified ring tool.

3.5 Placing

3.5.1 Place software module

The purpose of this function is to make the manipulator end effector move to a specified Z coordinate *target_Z* and drop a part or a tool releasing the vacuum in the corresponding pneumatic channel of the end effector (see Section 3.2.1). To overcome any residual adhesion of the item on the end effector additional compressed air can be blown through the channels. As in the previous section the different states of the action list and the actions/movements of the manipulator will be presented in the following. The corresponding Python code can be found in Appendix A.2.

Table 4: Manipulator action lists: drop function

ID	Description	Predicate	Channel	Flags
5	StopAt *target_Z*	"Start dropped" emitted	blow air	
2	StopAt *target_Z*	at *target_Z* and $p > 90kPa$	air off	
1	StopAt *target_Z*	at *target_Z* and $p \leq 90kPa$	air on	"Start dropped"
3	Move to *target_Z*		vacuum	
4	Move close to *target_Z*		vacuum	

Table 5: Manipulator behavior in successful place (top to bottom).

Performed action	Status of pneumatic channel
`move closer to` *`target_Z`* `(ID 4)`	`vacuum`
`move very close to` *`target_Z`* `(ID 3)`	`vacuum`
`stop at` *`target_Z`* `(ID 2 and 1)`	`release vacuum`
`stay at` *`target_Z`* `(ID 5)`	`blowing air`
`flush action list`	
`assure that item has dropped`	

Table 4 shows the action list of the manipulator during the main part of the place function. Due to different timing and execution condition issues, the order of the controllers in the list can not be the same as the sequence of their insertion which is represented by their *ID*. As described in Section 2.5, when the executor moves through the list, it executes the first controller action with fulfilled predicates. What results is a behavior as listed in Table 5 . The manipulator approaches the *target_Z* position, first quickly, then cautiously (controllers with *IDs* 4 and 3). Once arrived it blows a little bit of compressed air through the pneumatic channel to release the vacuum (controller with ID 1) until the obtained pressure is bigger then 90 kPa. Then the controller with the ID 2 is executed and temporarily stops the insertion of air into the pneumatic channel. If it has been specified when evoking the function, controller 5 is executed and blows a further and, because of the lengthier time, stronger stream of compressed air through the channel to blow off a potentially sticking item. After flushing the actionlist, the drop of the part or tool in question is then confirmed by measuring the current pressure in the pneumatic channel. If it is not close to atmospheric pressure the item is assumed not to have dropped and the manipulator will stop. Otherwise it will move up again.

3.6 Screwing

3.6.1 Screwing software module

This manipulator function is used to screw retaining rings into the collimator housing. The amount of total turns, the *delta_theta* of the first turn and the pitch of the thread are given as notable parameters. The whole ring screwing can be decomposed in three phases:

1. First turn to catch the thread (usually about 2 Π)
2. Perform turns about Π
3. Turn the rest $\leq \Pi$ till the ring is screwed in tight.

The individual movements performed by the manipulator are listed in Table 6. To screw, the end effector with the picked up tool is moved down and connects to the ring. The end effector is then turned and transmits the momentum to the ring via the small metal bar fixed to the tool. Due to the applied momentum a tension between tool and ring arises causing a lot of friction at their contact surface. Before moving up the tool to turn back and screw again this tension has to be eased by turning the ring for some degrees in the opposite direction. If this is not done, the arising friction causes the tool to keep sticking on the ring and the end effector to loose it.

Due to the repetitive nature of the screwing motions, after each of the movement controllers inserted into the action list another controller is inserted. It becomes active when the end effector reaches its intermediate goal position and emits a signal. Thus the correct execution of each movement can be assured without any interference by other controllers submitted to the action list at an earlier time. The action list itself becomes quite complex and varies. Therefore it is not presented in detail. The corresponding Python code can be found in Appendix A.3.

Table 6: Manipulator actions in screwing operation.

Performed action

```
move closer to target_Z
move very close to target_Z
stop at target_Z


turn about delta_theta
turn back a little
move up

do n times:
  turn back
  go down
  turn about Π and move down about the pitch/2
  turn back a little
  go up

turn back
go down
turn till ring is stuck
move up
```

3.7 Experimental results

The functions for the individual operations (picking up tools and parts, placing parts or tools and screwing in retaining rings) have been developed in several iterations of programming and extensive testing with a non moving courier. Under the condition that the relative position of the courier and the manipulator

Z axis is exact enough (about 0.1 mm of maximum position error for picking up and placing and 0.05 mm maximum position error for screwing) they have shown to perform robustly.

In general these requirements would be easily met by the Minifactory agents. However due to the refitting and modernizing efforts the sensors measuring the position of the courier on the platen and the relative position of courier and manipulator agents as well as vision were not working by the time of the end of the OSTI project. Small position errors occurred when moving couriers around due to the pull of the courier's tether. With careful handling of the tethers, initializing couriers at known positions and using a dead reckoning approach the relative positions could be made accurate enough for picking and placing one item but not for screwing. In addition to that, the fixture of the manipulator agent at the bridge over the platen proved to have some slackness. The manipulator would be slightly tilted towards any side when a certain force was exerted along the Z axis, occurring e.g. occasionally when picking up parts. This would result in position errors larger then 1 mm at the end effector. A complete assembly could therefore not be performed.

4 Vision based pick and place

4.1 Task analysis

The goal of this project is to pick and place small nylon spheres of a diameter varying around 300 µm using computer vision. If that succeeds pick and place of even smaller metal spheres of a diameter varying around 40 µm should be tried. An end effector including a camera, microscope, light source and an attached tweezer gripper has already been designed. The task can be divided into several subtasks:

Vision hardware The analog camera feed has to be led to a frame grabber in order to digitalize it and use it in image processing algorithms. In the previous Minifactory setup an Imagenation PX610 frame grabber was used. However there is no driver for this card for the QNX 6.3 operating system at this point.

OpenCV It is intended to use the Intel OpenCV open source vision library for image processing. It will have to be ported to the QNX operating system since there is no available distribution at this point.

Tweezer end effector A tweezer gripping mechanism has been designed to be mounted on the current end effector. The design has to be evaluated and made ready for use in this application.

Gripper control So far there is no solution to control the tweezer gripper and it is not known how to send analog or digital signals from the manipulator box to the end effector for that purpose. A control circuit has to be designed to transform these signals to an adequate form for the tweezer actuator. Furthermore the tweezer control has to be implemented in the Minifactory software.

Image processing The image processing has to be integrated in the Minifactory software architecture and the generation of the information for the specific task at hand has to be implemented.

Agent programming Programs running on the Minifactory agents, in this case on a manipulator picking and placing and a courier carrying the spheres have to be elaborated.

Interface tool factory A virtual representation of the pick and place factory that runs in the interface tool has to be created. Thus the execution of coordinated agent movements and the pick and place of the spheres can be performed in simulation and in reality.

The solution to these different subunits will be presented in the following sections.

4.2 Hardware

4.2.1 Vision hardware

The vision capable end effector designed prior to this project is shown in Figure 12. The mechanical, pneumatic and electrical interface connector and a vision unit are mounted on a “box” of anodized aluminum. The vision unit contains a Hitachi KP-M22 monochrome camera and a white LED light source. Different 2X, 5X and 10X magnifying Mitutoyo objectives can be used. Thus the manipulator has its own microscope. The objectives working distance of 34 mm permits the use of a variety of grippers which in turn are mounted on the underside of the end effector.

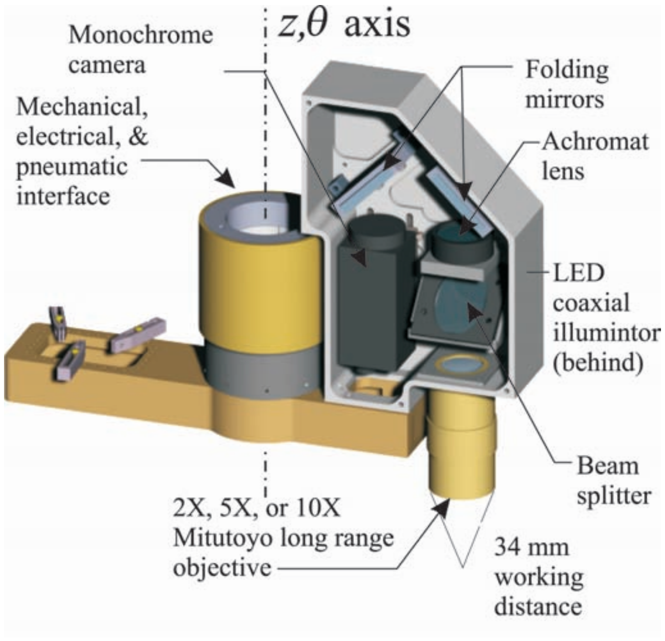

z,θ axis
Monochrome camera
Mechanical, electrical, & pneumatic interface
Folding mirrors
Achromat lens
LED coaxial illumintor (behind)
Beam splitter
2X, 5X, or 10X Mitutoyo long range objective
34 mm working distance

Figure 12: Vision equipped end effector.

In the previous Minifactory setup the feed from the camera is digitalized by an Imagenation PX610 frame grabber card. Further investigation turned up an unofficial driver for this card for QNX4 but not for QNX6.3[1]. As there are major differences for resource managers and device drivers for the two operating system versions the amount of work needed to rewrite the drivers was judged too great. Instead it was decided to use a newer card model, the Imagenation PXC200AL which comes with the needed device drivers for QNX. Unfortunately the new card proved to have some erroneous behavior when combined with the Advantech PCA-6105P5 Backplane and PCI6880F CPU card. It would not work at all in the designated PCI slot number 1. Despite efforts from Advantech the source of the problem could not be found. As the Manipulator Agent is very tightly build there is no mean to plug the PCX200AL in PCI slot number 3, the only other available one, without slight modifications. These modifications were carried out on a part supporting the valve manifold (shown in Figure 3 in Section 2.2) shifting the block of vacuum and pressured air valves about 2.5 mm to the right. A drawing of the

[1]Arnauld Delmas, CEA - CEREM, French atomic energy commission

modified part can be found in Appendix E.1. While at fist the card seemed to perform well, it was not discovered until very late in the course of the project that, when the card is plugged in, the manipulator can not be properly actuated. The manipulator produces a *velocity limit exceeded* error, which means that it is trying to suddenly accelerate strongly and is thereby violating the velocity limits set in the software. However the commands send to the APC-8620 carrier card over the PCI bus are correct. Due to the limited time available for this project the source of this problem could not be exhaustively investigated and the problem could not be solved. However a “work around” was developed enabling the demonstration of the elaborated pick and place operations. It will be described in the Section 4.5.1

4.2.2 Tweezer gripper

The gripper provided for this pick and place application consists of a standard non-magnetic tweezer and is shown in Figure 13. Its legs are opened and closed symmetrically using a H2W technologies NCC05-11-1X frictionless voice coil actuator (see Appendix D.1). The voice coil and the corresponding magnet are mounted on opposite sides of the tweezer with a small counterbalance attached to the coil to equal the higher weight of the magnet. The tweezer is mounted in a clamp enabling rotational adjustment, which is in turn attached to a Newport DS25 XY stage. Thus the tips of the tweezer can be moved in XYZ. The whole gripper mechanism is attached to the bottom of the end effector with the tips of the tweezers below the objective lens. The position of the tips can then be adjusted to move them in focus into the area seen by the camera.

4.3 End effector control

The manipulator uses an Acromag APC-8620 PCI carrier card with four Industry-Pack (IP) modules for the input and output of sensor values, actuator commands and other signals. As presented in detail in [1] only lines for digital I/O signals, analog inputs and the video input are run down to the end effector connector. The two analog outputs generated by the DACs on a SBS Technologies IP-PRECISION module are used to control the Z and θ axis motors. This means that only the output signals on the digital I/O lines can be used for the gripper

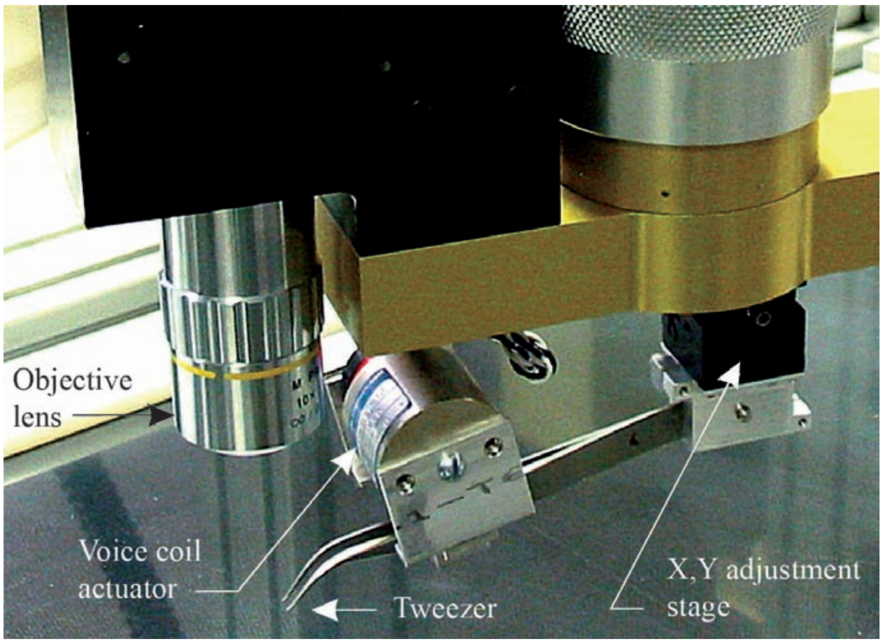


Figure 13: Tweezer gripper.

control. Of six digital I/O lines in total at the end effector, four are available for this task since two are reserved to access an EEPROM device on future versions of this end effector. The digital I/O signals are generated resp. read on a Acromag IP408 high voltage digital I/O module. As no detailed documentation was available for these digital I/O signal lines, reverse engineering of the manipulator cable routing provided the pin numbers on the connectors and cables. The augmented pin description tables can be found in Appendix E.2.

The forces generated in the voice coil actuator and therefore the tweezer displacement are controlled by the input coil current. As experiments showed a high repeatability and task - sufficient accuracy of the tweezer movements it was concluded that an implementation of closed loop control would not be necessary. The needed forces for the gripping task are varying. While the gripping force exerted on a sufficiently rigid gripped object is a linear function of the current, the change of the distance between the two tips before the closure of the tweezer is not linear due to the slightly non linear spring behavior of the tweezers legs. In addition to that, the location of the mounting spots of the voice coil and magnet on the tweezer greatly influences the needed force to close the tweezer as well as the distance between the tips in the *zero position* when no current at all is applied. The current needed to barely close the tweezer can vary from 25 to

63 mA at 12 V.

To transform the 4 bit signal into 16 different currents the control circuit shown in Fig. 14 was designed. The four digital I/O lines are pulled up to 5 V using an LM7805 voltage regulator. The resistors R1 to R4 limit the current when the lines are shorted to ground in the I/O module on the carrier card, which happens when 'sending' a *0*. The central unit of the circuit is the LMD18245 DMOS full-bridge motor driver (see Appendix D.2). It includes a 4 bit DAC controlling an output current via a fixed off-time chopper technique. The gain for the internal current amplifier defines the range of the output current. It can be set using different resistor values for the resistor R_s at the CS OUT pin (labeled *R6* in Fig 14). R_s is calculated using the formula 4.1 which has been obtained from the data sheet.

$$R_s = \frac{(V_{DACRef} * \frac{DIO}{16})}{((250 * 10^{-6}) * I_{out})} \tag{4.1}$$

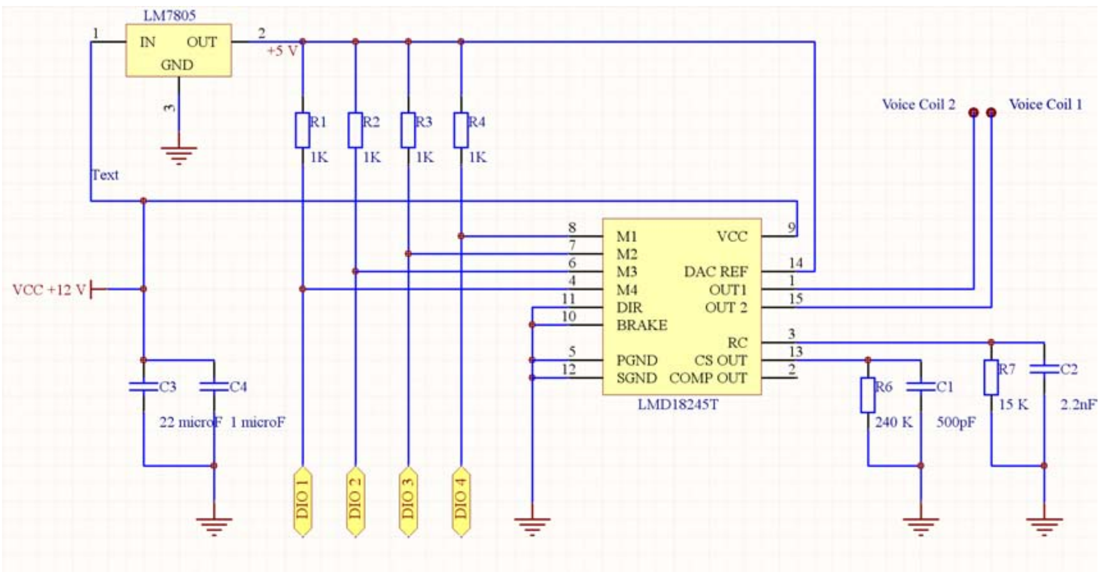


Figure 14: Tweezer control circuit.

An initial calculation returned a value of 280 kΩ for R_s. However measurements showed that the obtained currents were smaller by a constant offset then the expected value. The offset corresponds to two steps in the values 0-15 written on the digital I/O lines. This seems to be caused by the very low current range at which the motor driver is operating. Increasing the current range and letting most of the current flow through a resistor parallel to the voice coil would however return too imprecise results, since the currents would change considerably with the change of resistances due to thermal effects. Since only a fraction of the whole 16 digital I/O settings can be used in any case due to too small or too high currents depending on how the voice coil is mounted on the tweezer and since the behavior of the circuit is stable otherwise, it was chosen to accept the loss of two current settings. The new value determined for R_s taking this into account is 240 kΩ. The resulting currents are listed in Table 7.

Table 7: Voice coil currents.

DIO value	15	14	13	12	11	10	9	8
Current [mA]	69.1	64.0	58.8	53.6	48.3	43.1	37.9	32.7
DIO value	7	6	5	4	3	2	1	0
Current [mA]	27.5	22.3	17.1	11.9	6.7	0	0	0

A reference voltage of 5V for the DAC is connected to pin 14. The off-time for the current chopper is set to 36.3 µs via a parallel resistor capacitor network connected to RC. This proved to be well fast enough to be entirely damped in the voice coil and thus the on and off switching of the current provokes no oscillation in the tweezer. As there is no more output line available at the end effector the BRAKE pin is permanently connected to ground. The flow of current across the voice coil can be disabled by sending *"0000"* on the digital I/O lines. Since there is no need for any change of current direction the DIRECTION pin is connected to ground as well. The whole circuit is supplied by a 12 V source available at the end effector. The supply is bypassed by the aluminum electrolytic capacitor C3 (22 µF) and the high frequency ceramic capacitor C4 (1 µF) to protect the circuit and minimize effects of normal operation on the supply rail. The different circuit components were soldered on a small card as shown in Figure 15 that fits inside the lower aluminum box of the end effector (see Figure 16). The card has two four pin connectors, one for the four digital I/O's and one for VCC, ground and the two outputs going to the tweezer. As shown in Figure 16 (a) the connected cables are directly soldered to the end effector connector. The Pin mapping for the card can be found in Appendix E.2.

To be able to make efficient and manageable manipulator programs one should be able to set the values on the four digital I/O lines from the Python script level just as e.g. when commanding to open a valve and send pressured air down to the

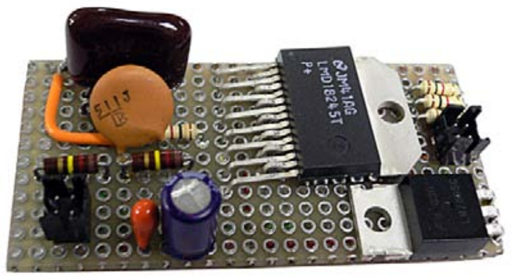
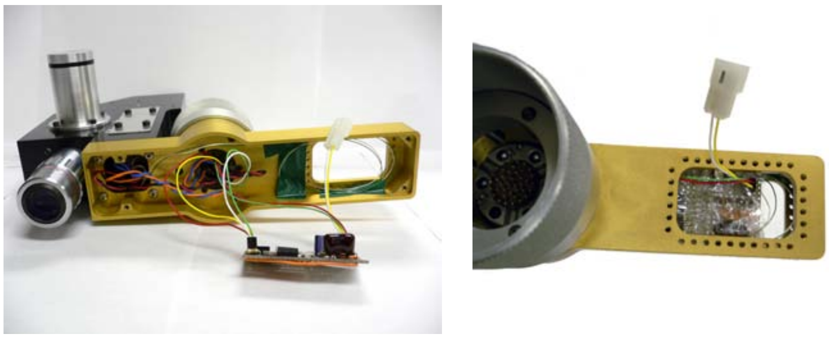


Figure 15: Tweezer control card.



(a) Open end effector with connected tweezer control card

(b) End effector with mounted tweezer control card

Figure 16: End effector and tweezer control card.

end effector. In fact the control of these vales is effectuated using other digital I/O lines originating in the same IP408 module on the APC-8620 carrier card. Thus the propagation of the command to set these lines to a specific value has been taken as example for the end effector digital I/Os. The general architecture of the software running on the manipulator and the flow of information resp. commands within it is very complex and unfortunately mostly undocumented. What seems to be happening is illustrated in Figure 17. The *maniphead* program launches a scripting thread executing the manipulator Python program. Within the manipulator Python environment the global variables *ee_dio1* to *ee_dio4* have been defined. The bits to be sent on the digital I/O lines have to be set here. For convenience the function `setDIO(self, value)` has been included in the manip-

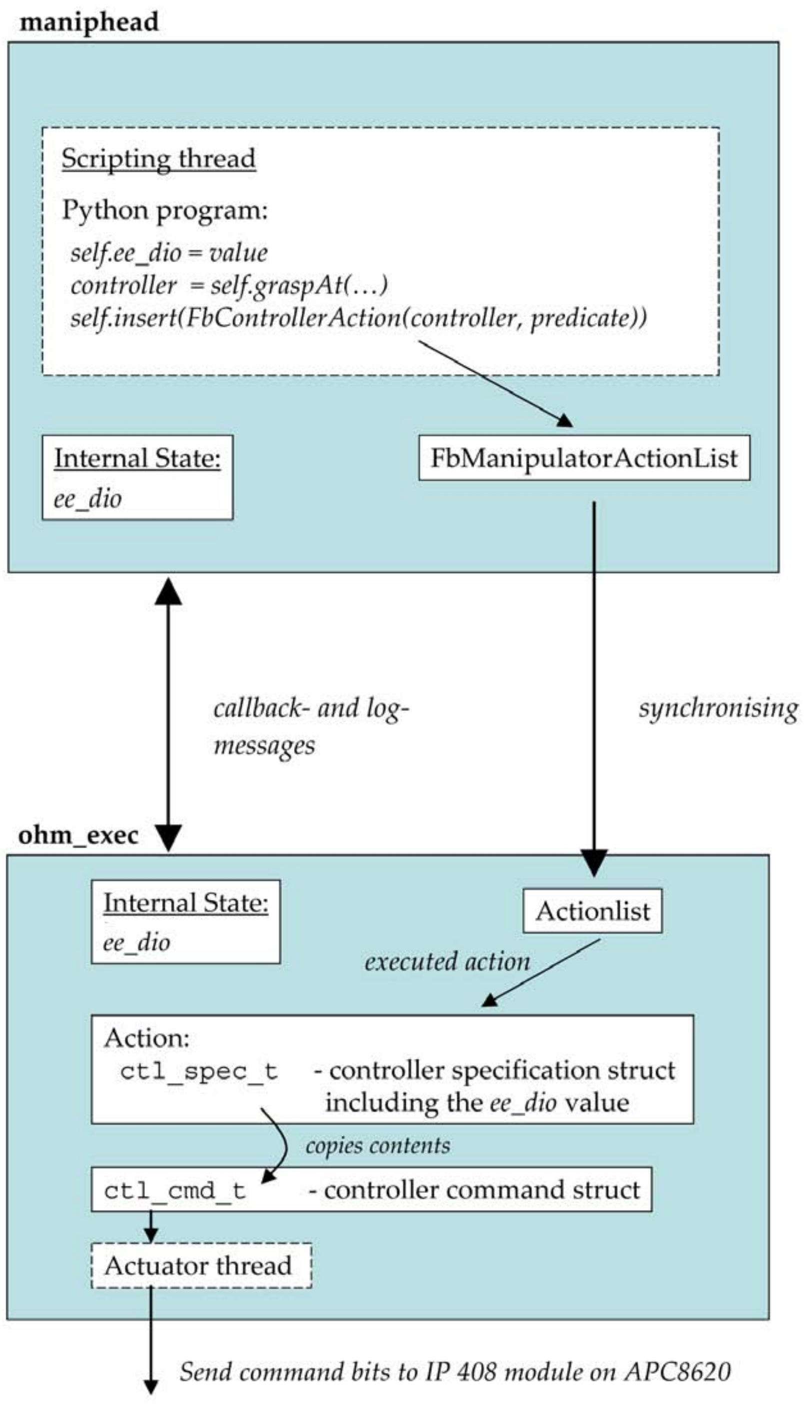

maniphead
Scripting thread
Python program:
self.ee_dio = value
controller = self.graspAt(...)
self.insert(FbControllerAction(controller, predicate))
Internal State:
ee_dio
FbManipulatorActionList
callback- and log-messages
synchronising
ohm_exec
Internal State:
ee_dio
Actionlist
executed action
Action:
ctl_spec_t - controller specification struct including the ee_dio value
copies contents
ctl_cmd_t - controller command struct
Actuator thread
Send command bits to IP 408 module on APC8620

Figure 17: Setting the *ee_dio* value: flow of information in *maniphead* and *ohm_exec* programs.

ulator program class *DemoMainpProgram.py*. When handed a `value` between 0 and 15 the *ee_dio* bits are set accordingly. When an controller is generated as e.g. a *graspAt* controller using the `self.graspAt(...)` function, the function `setAux(self, controller)` defined in *DemoMainpProgram.py* is invoked and copies the values of the global Python variables to the controller parameter (`controller.ee_dio1 = self.ee_dio1`). When the insert command is executed on the Python level, the controller parameters are converted to C++ variables and a controller containing the *ee_dio* values is inserted into the maniphead action list. This list is synchronized with another action list running in the *ohm_exec* real time executive program. There the controller parameters are stored again in a struct called `ohm_spec_t` and, when executing the controller, copied to a command struct `ohm_cmd_t` which is passed to the actuator thread. Here, among other things, the low level read and write operations to the different IP modules are performed. For each digital I/O line the bit stored in *ee_dio* is written to the IP408 digital I/O module. Which bit at the IP408 card corresponds to which line at the end effector had once again to be obtained by reverse engineering. The augmented IP408 bit table can be found in Appendix E.2. The complete list of altered files can be seen in Appendix D.3.

4.4 Vision software module

4.4.1 Vision in the Minifactory software architecture

To keep in line with Minifactory's modular software architecture, a client - server approach was chosen to implement computer vision, structurally based in parts on previously implemented visual tracking code. Image processing is done in an independent vision server application. The server then provides the results and possibly additional data or even complete images to clients. That way there can be multiple clients running anywhere within the factory. Clients may be independent applications as well as Python programs running on Minifactory agents or in an interface tool simulation environment. All communication between server and clients is done via the IPT middle ware [2] already used in the Minifactory environment.

4.4.2 Vision server

The vision server consists of the main server application implemented in *vision_serv.cc*, the `AAAVision` class implemented in *AAAvision.h* and *AAAvision.cc* and the `PXC_Controls` class implemented in *pxc_controls.h* and *pxc_controls.cc.* The source code can be found in the Appendix B.2. The `PXC_Controls` class acts as an interface to the PXC200AL frame grabber card. It loads the driver libraries, initializes the card and provides access to camera feed properties, grabbed frames or to frames converted into other image data structures. All image processing routines are implemented in the `AAAVision` class as e.g. a function that returns the coordinates of a sphere if one is found in the current frame. The main server application sets up the IPT communication containers and managers, registers message types and links them to different message handlers which invoke the corresponding routines in the AAAvision class. The handler for the *VISION_GET_CLOSEST_SPHERE_MSG* message invokes e.g. the function mentioned above and sends the results back to the client in an other message. The whole vision server is set up in such a way that it is easily extensible by specific image processing code and client-server messages for future applications. A detailed instruction for this and for setting up the vision server can be found in Appendix B.4 resp. B.1. So far the vision server only processes direct video feed from the frame grabber and therefore has to be circumvented when simulating a factory in the interface tool. However a vision server in *simulation mode* could be imagined which reads in and processes previously acquired and saved images instead of grabbing them from the frame grabber. File input and output routines are already made available so that the implementation should not require too much effort.

4.4.3 Vision client(s)

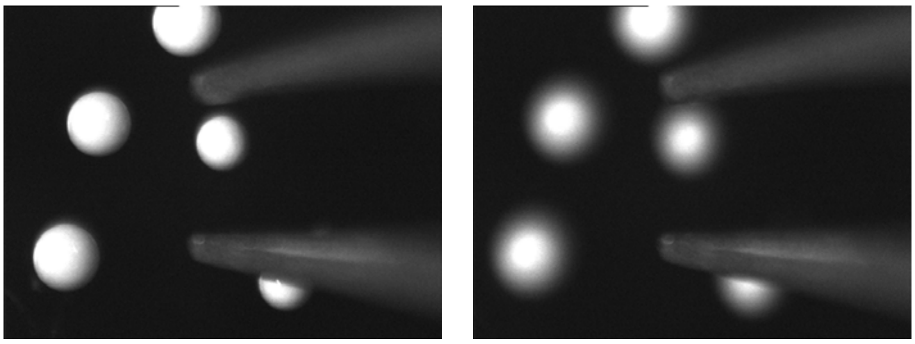
A vision clients sends IPT messages to the client server requesting some data and receives it in another IPT message. Therefore vision clients can have all kinds of forms. So far for lack of time, only Python code running on the manipulator has been enabled to be a client. To be able to set up an IPT connection between client and server the client needs the IPT specification string of the vision server e.g. *"ipt:ocelot|1500,0".* This IPT "address" has to be set in the *VISION_SPEC* environment variable on the agent where the *maniphead* program running the

Python code is launched. When the maniphead application starts it reads the specification and passes it to the manipulator C/C++ - Python interface. Thus the string becomes available in Python and an IPT connection can be initiated from the Python level. What remains to be done is to register the same messages and message contents as defined in the vision server. Example code can be found in Appendix B.3.

4.4.4 Image processing

The Intel OpenCV open source vision library was partially ported to QNX6.3 to enable convenient image processing. OpenCV essentially consists of four main libraries. The libraries *libcxcore, libcv, libcvaux* containing all image processing data structures and algorithms were entirely ported and successfully tested. The library *libhighgui* contains a high level user interface and different file input output functions. Porting the user interface functions of *libhighgui* to QNX requires significant work since QNX is quite different from Unix/Linux in these areas. As the Minifactory concept does not intend for the user to directly access agents while some factory operation is performed but to monitor it from a central station, there is no need for any image processing user interface library on QNX. (The central station, i.e. the computer running the *interface tool* runs on Linux). Therefore only some of the file input output operations were extracted and ported into the library *libcvAAA*. Should the need arise, special image processing algorithms can also be implemented into this library in the future.

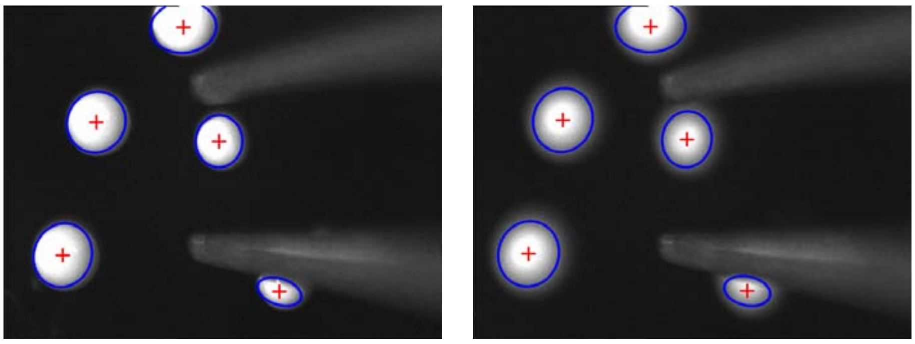
To be able to pick up and place spheres one needs to find their center points. Image processing code is needed to analyze the current camera feed and find spheres, if there are any, but not return points on other items like dust or the tweezer tips. The algorithm to do that should be as robust as possible, unperturbed by changes of the light or by blurring of the camera image because the spheres are out of focus. A normal and a blurred sample image taken with the end effector camera can be seen in Figure 18. Two different approaches to find the spheres were developed and examined. The first one finds circles in thresholded images using the 21HT Hough transform method described in [17], implemented in OpenCv. The second generates a threshold image as well, then extracts contours and fits ellipses to the contours. Only ellipses with diameter sizes above a minimum of 30 pixels (about 110 μm) were retained. Applying a threshold of 130



(a) Spheres in focus (b) Spheres out of focus

Figure 18: Spheres as seen by the end effector camera.

on the gray scale from 0 to 255 proved to be a robust method to avoid recognizing the tweezer tips, since there is a considerable brightness difference between them and the spheres. If properly tuned, the second method returned very promising and robust results as can be seen in Figure 19. The Hough transform method could be tuned to either recognize the spheres in focus as in Figure 20 or the blurred ones with acceptable robustness, but not both. Therefore the ellipse fitting method was implemented in the vision server. For instance the function `getClosestSphereDistance(...)` in the *AAAVision* class extracts the ellipses and returns the distance in X and Y from the ideal pick up point in the middle between the tweezer's tips to the closest sphere found. The Hough transform code can be found in Appendix B.5, the implemented image processing code in Appendix B.2.



(a) Spheres in focus (b) Spheres out of focus

Figure 19: Results of sphere recognition with ellipse fitting.

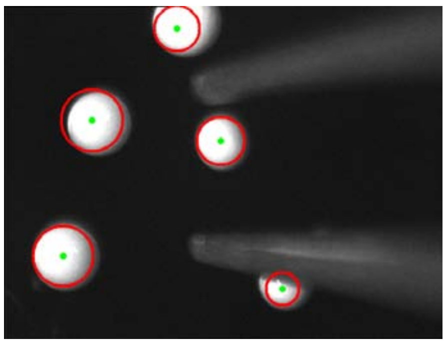


Figure 20: Result of sphere recognition with Hough transforms.

4.5 Experimental results

4.5.1 Test setup

The spheres are stored in a AD-23CT-00X4 GelPak box on a sticking gel surface. For the experiments the GelPak box was glued on top of a courier as seen in Figure 21. As described in section 4.2.1 hardware problems with the frame grabber card made it impossible to plug it into the manipulator used for picking and placing. However due to the modular implementation of the vision software it was possible to plug the card into an other manipulator which is not used otherwise and to run the vision server on it. The camera feed is redirected from a external video output of the first manipulator to the frame grabber card in the second.

4.5.2 Factory programming

The programming of a factory for a pick and place demo consists of the generation of a factory file, a manipulator agent program and a courier program. First a *.fac* factory file has to be generated which is a description of the different modules and objects and their positions in the factory. This is partly done by using the interface tool and partly by manual editing. The *visionDemo* factory consists of one courier with an attached GelPak moving spheres around for a manipulator to find, pick and then to place them at a different place on the GelPak. Furthermore a platen table, bumpers for the courier at the edges of the platen as well as a bridge to mount the manipulator are added. The positions of these objects entered in the *.fac* file have to match the actual positions in the real factory. The generated *visionDemo.fac* file can be seen in Appendix C.1. The resulting virtual representation of the factory which is used for factory simulation as well

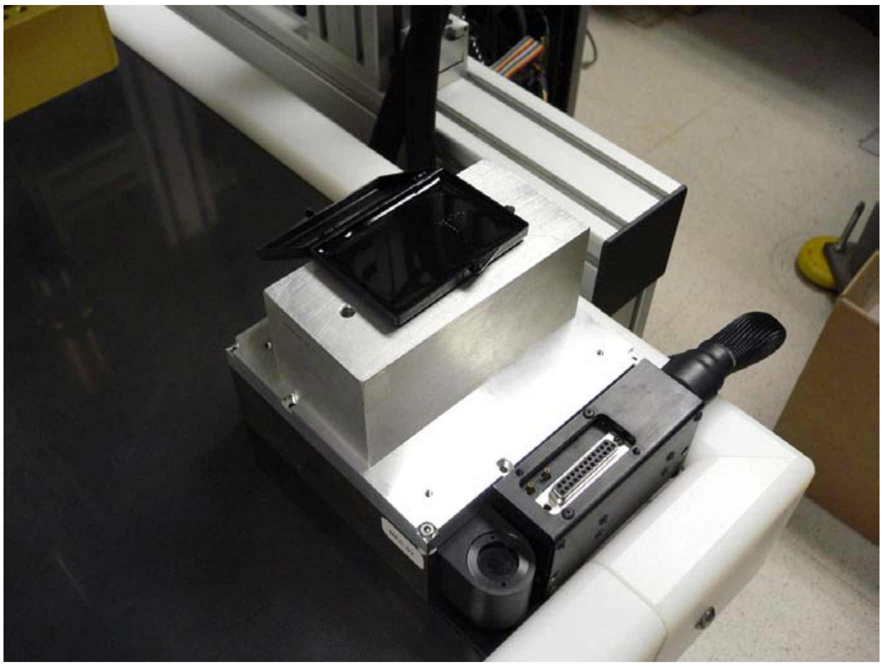


Figure 21: GelPak with spheres on courier.

as visualization of what is happening when the factory is actually running can be seen in Figure 22.

Manipulator and courier have to perform coordinated moves e.g. when a sphere is found and needs to be approached. Since the manipulator program acts as vision client the manipulator knows where the sphere is and therefore takes the leading role in the coordination. The tweezer mechanism showed to be oscillating heavily when a θ axis movement of the manipulator is performed, due to the heavy weight of the voice coil attached to the tweezer acting like a plate spring. Therefore the manipulator should only move up and down along Z when picking and placing while the courier movements take care of the relative X and Y displacements of the sphere. The manipulator therefore sends messages to the courier telling it where to perform a movement to and waiting for a confirming message that the courier has arrived at its destination.

The courier program is implemented in *VisionDemoManipProgram.py* (Appendix C.2) and is very simple. When the courier starts up it first sets its zero position on the platen. Then it initiates a *RendezVous* as described in 3.3 to initiate manipulator courier interaction and simply waits for the manipulator to tell it where to go. Once the pick and place operation is done it terminates the

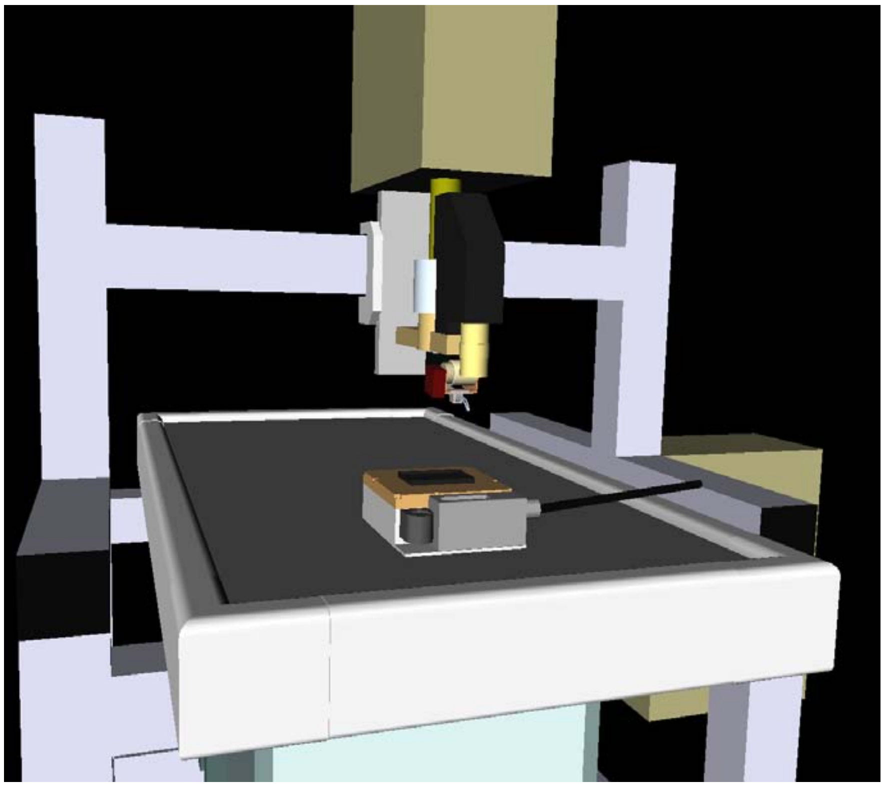


Figure 22: Virtual sphere pick and place factory.

rendezvous, regains it's "will" and moves away.

The manipulator program is implemented in *VisionDemoManipProgram.py* (Appendix C.3). The manipulator waits after initialization for the courier to request a *RendezVous* and accepts it. It then tells the courier to move to a position under its camera. The manipulator moves down to get the surface of the GelPak on the courier in focus and begins to perform a search for spheres. This is implemented in the `searchForSphere(self)` function. The manipulator program tells the vision server to process the current view through the camera and to return the coordinates of any found spheres. If none are sent back the manipulator tells the courier to move. Thus a search pattern as shown in Figure 23 is performed. Experiments returned that to guarantee that spheres lying exactly at the edges of the fields of view are guaranteed to be found if the overlapping area of two searched fields of view is two thirds of the spheres diameter, which is 200 microns. When a sphere is found the vision server returns the distance form the spheres center to the ideal position for a pick up. The courier is told to move in such a way that the sphere's center comes to lie within a small tolerance area around this ideal position. Once the sphere is in place, the manipulator moves slightly down and closes the tweezers one step after another. Closing the tweezer at all once increases the probabilities of the sphere being pushed away. The manipulator moves up, tells the courier to move to the goal position, moves down and places the sphere. After moving up again, the *RendezVous* with the courier is terminated.

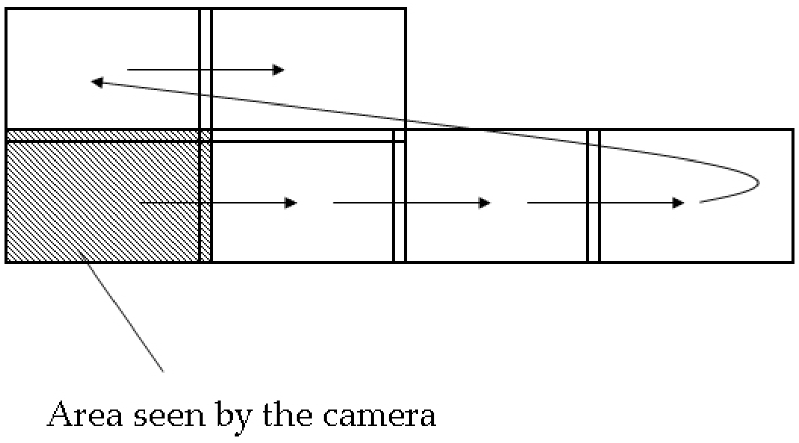


Figure 23: Search pattern performed when looking for spheres.

4.5.3 Results

Picking up and precisely placing spheres could be achieved as is shown in the image sequence in Figure 26. Though the locating and approaching of spheres proved to be very robust from the beginning there were a couple of issues with picking and placing. Sometimes the spheres stuck to one of the tweezer's tips when opening the tweezer for placing as shown in Figure 24. This seems to be caused by static adhesion and could be overcome in some cases by pushing the sphere and tweezer harder onto the GelPak. However this destroys the GelPak surface after some time. As seen in Figure 25 some spheres are not perfect spheres but have an egg like shape. These proved to be very difficult to pick up. An alignment of the long axis of these ellipsoids with the tweezers could bring some improvement but was not tried since some manipulator movement in θ would be necessary. This is made impossible so close to spheres by the described uncontrollable oscillations of the tweezer. As the spheres also vary in size the Z position of the tips of the tweezer would have to be slightly adapted to avoid closing the tweezer below or above the sphere's midpoint while sliding on its surface and pushing it away. Due to lack of time this could however not be implemented anymore.

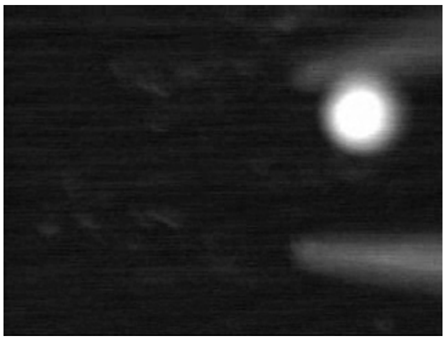


Figure 24: Sphere sticking to tweezer when placing.

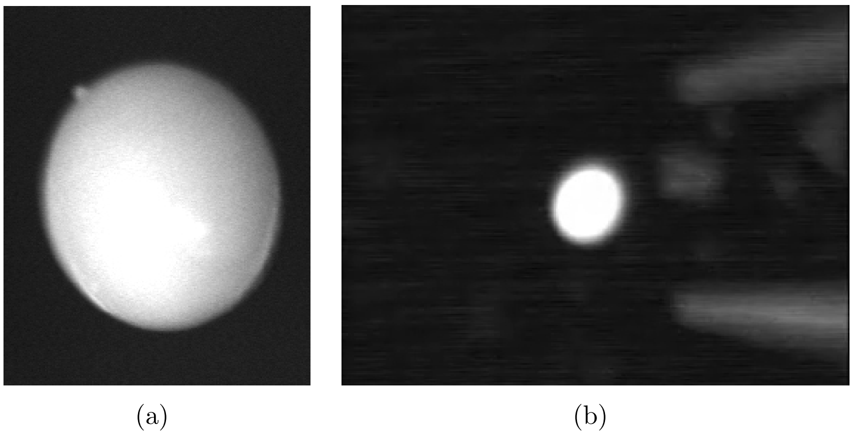
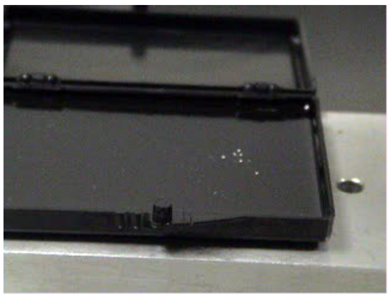
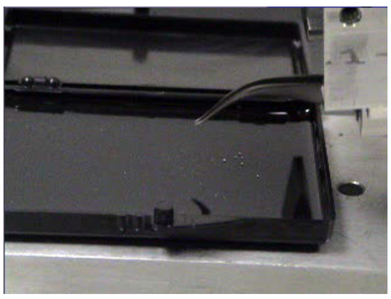
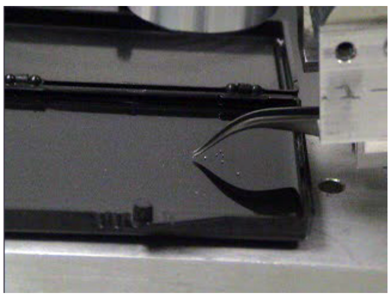
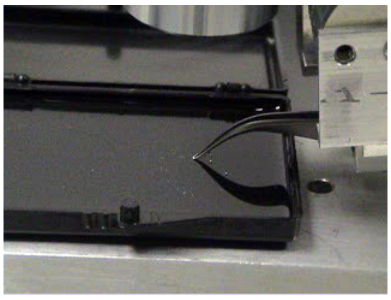


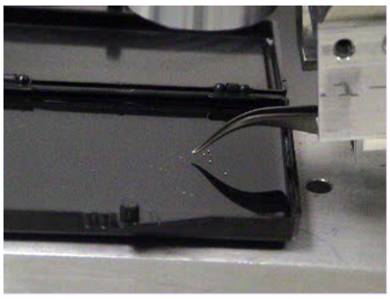
Figure 25: Egg shaped spheres.

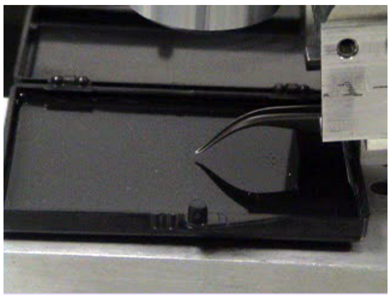

(a) GelPak with spheres

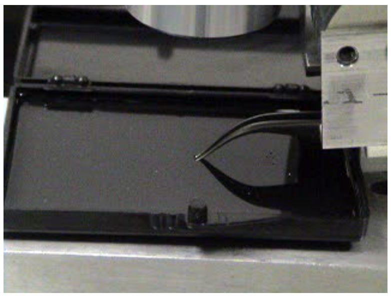

(b) Going down


(c) Searching for spheres


(d) Picking up


(e) Going up with sphere


(f) Moved towards goal


(g) Placing sphere

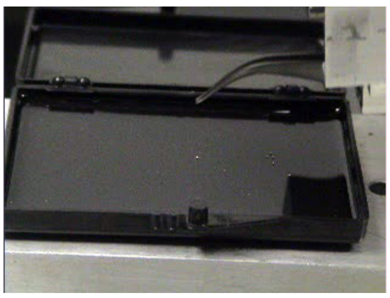

(h) Going up

Figure 26: Pick and place of a sphere.

5 Summary and Contributions

A significant amount of time available for this thesis was spent solving problems and correcting bugs appearing in conjunction with the Minifactory refitting and upgrading effort. Though some issues could not be dealt with in time, at the end stands a Minifactory environment capable again of performing tasks such as the sphere pick and place. Pick up, place and screw agent programs and procedures have been developed that enable the complete assembly of the telescopic sight once the coordination and platen sensors are available again and the manipulator fixation problem has been solved. The tweezer gripper is now controlled and can be actuated. The possibility to send signals over the four digital I/O lines to the end effector has been integrated in the Manipulator software. This could also be put to use in different future grippers utilizing similar control circuits as designed for the tweezer voice coil actuator. An extensible, independent vision processing client server module has been developed and integrated in the Minifactory software architecture. Procedures for a robust sphere recognition have been implemented. Demonstration factory and agent programs for the sphere pick and place have been generated. Finally, spheres of 300 µm diameter have successfully been picked up and placed at a target position.

The presented work can serve as a base for future Minifactory pick and place applications. The pick and place functions should be usable for different items, picked and placed with vacuum and air or different grippers with only minor adaptations.

5.1 Future Work

Further work can be done to enhance the robustness of the sphere pickup. Due to lack of time this could not be optimized. The next step would then be to enable and optimize the pick up of smaller metal spheres with a diameter around 40 µm. The mechanical design of the tweezer gripper should be adapted to solve the described oscillation problems when actuating the manipulators θ axis. A modification of the fixation of voice coil and magnet on the tweezer could probably lead to the possibility to take the gripper apart without having totally different currents/forces needed to close the tweezer after reassembly. A reexamination of the manipulators Z axis PD and PID controller parameters could be a way to solve the position errors appearing after the refitting and the use of heavier end effectors.

The vision server could be extended by a *simulation mode* as described in Section 4.4.2 to tightly integrate it into the AAA concept and to speed up assembly system development processes. Vision clients receiving complete images from the vision server could be imagined on the interface tool system to monitor the manipulator's vision and the vision processing. Likewise, a client acting as an image processing code development environment integrated in the interface tool could be created. Since this would be running on a Linux system the OpenCv user interface library is already available to display image data and could be used.

References

[1] 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)*, volume 2, pages 1030 – 1035, 2001.

[2] J. Gowdy. Ipt: An object oriented toolkit for interprocess communication. Technical Report CMU-RI-TR-96-07, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, March 1996.

[3] J. Gowdy and Z. Butler. An integrated interface tool for the architecture for agile assembly. In *IEEE International Conference on Robotics and Automation*, volume 4, pages 3097 – 3102, May 1999.

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

[5] R. Hollis and J. Gowdy. Miniature factories for precision assembly. In *International Workshop on Microfactories*, pages 9 – 14, December 1998.

[6] R. Hollis and A. Quaid. An architecture for agile assembly. In *American Society of Precision Engineering 10th Annual Mtg*, October 1995.

[7] R. L. Hollis, D. O'Halloran, G. Fedder, N. Sarkar, and J. R. Jones. Vision guided pick and place in a minifactory environment. In *Proc. 5th Int'l. Symp. on Microfactories, Besancon, France*, October 2006.

[8] R. L. Hollis, A. A. Rizzi, H. B. Brown, A. E. Quaid, and Z. J. Butler. Toward a second-generation minifactory for precision assembly. In *Proceedings Int'l Advanced Robotics Program, Moscow, Russia*, April 2003.

[9] 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)*, volume 3, pages 3121 – 3126, May 2001.

[10] W.-C. Ma, A. Rizzi, and R. Hollis. Optical coordination sensor for precision cooperating robots. In *IEEE International Conference on Robotics and Automation 2000*, volume 2, pages 1621 – 1626, April 2000.

[11] P. Muir, A. Rizzi, and J. Gowdy. Minifactory: A precision assembly system that adapts to the product life cycle. In *SPIE Symposium on Intelligent Systems and Advanced Manufacturing*, October 1997.

[12] A. Quaid and R. Hollis. Cooperative 2-dof robots for precision assembly. In *IEEE International Conference on Robotics and Automation*, April 1996.

[13] A. Quaid and R. Hollis. 3-dof closed-loop control for planar linear motors. In *IEEE Int'l. Conf. on Robotics and Automation*, pages 2488 – 2493, May 1998.

[14] A. Rizzi. Hybrid control as a method for robot motion programming. In *IEEE Int'l. Conf. on Robotics and Automation*, volume 1, pages 832 – 837, May 1998.

[15] A. Rizzi, J. Gowdy, and R. Hollis. Agile assembly architecture: An agent-based approach to modular precision assembly systems. In *IEEE Int'l. Conf. on Robotics and Automation*, volume 2, pages 1511 – 1516, April 1997.

[16] A. Rizzi, J. Gowdy, and R. Hollis. Distributed programming and coordination for agent-based modular automation. In *The Ninth International Symposium of Ribotics Research, Snowbird,UT*, October 1999.

[17] H. K. Yuen, J. Princen, J. Illingworth, and J. Kittler. Comparative study of hough transform methods for circle finding. *Image Vision Comput.*, 8(1):71–77, 1990.

A OSTI software modules

A.1 Pick up

```
# osti_pickup(self, dest_z, dest_th, offset, grippers, test_gripper, graspPressure1 =\
#      30000, graspPressure2 = 30000,limit = 5.0 )
#
# Picks up parts and /or tools with OHM endeffector.
#
# Context: OSTI Project, Demo on 9/20/06
# Version 1.00, 9/19/06
# Authors: Cornelius Niemeyer <cornelius.niemeyer@gmail.com>,
#          Mark Dzmura <mdz@cs.cmu.edu
# based on previous version of NN
#
def osti_pickup(self, dest_z, dest_th, offset, grippers, test_gripper, graspPressure1 =\
        30000, graspPressure2 = 30000,limit = 5.0,verbose =1 ):

    #Air and vacuum channels  (IF POSSIBLE DEFINE THIS GLOBALLY !!)
    GRIP_PART_TOOL = 1      # air/vac port #1 is assigned to part tool
    GRIP_PART = 2           # air/vac port #2 is assigned to part
    GRIP_SUCK = -1
    GRIP_OFF = 0
    GRIP_BLOW = 1

    #Movement tolerances (for predicates)
    TH_DIFF_MAX=0.04                #in Radians
    TH_DIFF_MIN=0.04                #in Radians
    TH_DIFF_SINGLESTEP=0.04         #in Radians
    Z_DIFF_MAX=2.0  #1.0            #in mm
    Z_DIFF_MIN=1.0  #0.2            #in mm


    if verbose:
        print "\n#################################################################"
        print "#                    OHM pickup() function                     #"
        print "#                                                              #"
        print "#                Version 1.00 - OSTI Project                   #"
        print "#       (c) Cornelius Niemeyer, Mark Dzmura, MSL 2006          #"
        print "################################################################"
        print "\n# Parameters: "
        print "# graspPressure1 is %6.4f Pa" % (graspPressure1)
        print "# graspPressure2 is %6.4f Pa\n" % (graspPressure2)
        print "# limit is %6.4f sec\n" % (limit)



    #
    # move to courier approach position
    #

    self.singleStep(dest_z + offset, dest_th)
```

```
        if verbose:
            print"_______________________________________\n"
            print"Moving into position, starting pickup."
            print"_______________________________________\n"
        self.sleep(2)

        #
        # drive down SLOWLY to part position (enable grip vac #1 and #2)
        #

#        predicate = self.inBox(min_th = dest_th-0.04, max_th = dest_th+0.04)
#        print "Single stepping"
#        self.singleStep(dest_z, dest_th, move_predicate=predicate)
#        print "Single stepped"
#        self.sleep(0.1)

        #Grasp ctrl
        for gr in grippers:
            self.grip_set(gr, GRIP_SUCK)
        controller = self.graspAt(dest_z, dest_th, grippers)
        action = FbControllerAction(controller,
                                    self.inBox(dest_z-Z_DIFF_MIN, dest_z+Z_DIFF_MAX,
                                               dest_th-TH_DIFF_MIN, dest_th+TH_DIFF_MAX))
        id = self.insert(action)
        print"Grasp ctrl"

        #Move to pistion ctrl
        id2 = self.insert(FbControllerAction(self.moveTo(dest_z, dest_th),
                                             self.inBox()), id)
        print"Move to Position ctrl"

        # shouldn't have to do this!
        for gr in grippers:
            self.grip_set(gr, GRIP_SUCK)

        predicate = self.inBox(dest_z+offset-Z_DIFF_MIN, dest_z+offset+Z_DIFF_MAX, dest_th- \
            TH_DIFF_MIN, dest_th+TH_DIFF_MAX)

        if test_gripper == 1:
            predicate.max_gripper1_pressure = graspPressure1
        elif test_gripper == 2:
            predicate.max_gripper2_pressure = graspPressure2


        controller = self.stopAt(dest_z+offset, dest_th)
        action = FbControllerAction(controller, predicate)
        action.addStartCallback(self.tag("GotPart"))
        tag = self.tags["GotPart"]
        # shouldn't need below statement...?
        tag.fired = 0
        id = self.insert(action)
```

```
    predicate = self.inBox()

    if test_gripper == 1:
        predicate.max_gripper1_pressure = graspPressure1
    elif test_gripper == 2:
        predicate.max_gripper2_pressure = graspPressure2


    self.insert(FbControllerAction(self.moveTo(dest_z+10, dest_th),
                                   predicate), id)

    # shouldn't need to set self.gripperN here....
    for gr in grippers:
        self.grip_set(gr, GRIP_SUCK)

    start_time = FbTime_getTimeOfDay()
    status = 1
    while not tag.fired:
        cur_time = FbTime_getTimeOfDay()

        if test_gripper == 1:
            gripper_pressure = self.interface.gripper1_pressure
        elif test_gripper == 2:
            gripper_pressure = self.interface.gripper2_pressure


        print "Elapsed time %5.2f, %8.0f, %6.4f, %6.4f" % \
              (cur_time.getValue()-start_time.getValue(), \
               gripper_pressure, dest_z, self.interface.pos[0])
        if cur_time.getValue()-start_time.getValue() > limit:
            status = 0
            break
        self.processEvents(1.0)
    tag.fired = 0

    return status
```

A.2 Place

```
# osti_drop(self, dest_z, dest_th, offset, grippers, test_gripper, blow_time)
#
# Drops parts and /or tools hold by OHM endeffector.
#
# Context: OSTI Project, Demo on 9/20/06
# Version 1.00, 9/19/06
# Author: Cornelius Niemeyer <cornelius.niemeyer@gmail.com>
#
#
def osti_drop(self, dest_z, dest_th, offset, grippers, test_gripper,\
        blow_time=2.0,verbose =1):
```

```
#Air and vacuum channels  (IF POSSIBLE DEFINE THIS GLOBALLY !!)
GRIP_PART_TOOL = 1      # air/vac port #1 is assigned to part tool
GRIP_PART = 2           # air/vac port #2 is assigned to part
GRIP_SUCK = -1
GRIP_OFF = 0
GRIP_BLOW = 1

GRIP_PRESSURE_ALMOST_ATM = 90000


#Movement tolerances (for predicates)
TH_DIFF_MAX=0.04                #in Radians
TH_DIFF_MIN=0.04                #in Radians
TH_DIFF_SINGLESTEP=0.04         #in Radians
Z_DIFF_MAX=2.0  #3.0            #in mm
Z_DIFF_MIN=0.3  #0.5            #in mm


if verbose:
    print "\n###################################################################"
    print "#                    OHM drop() function                         #"
    print "#                                                                #"
    print "#                Version 1.00 - OSTI Project                     #"
    print "#        (c) Cornelius Niemeyer, Mark Dzmura, MSL 2006           #"
    print "##################################################################"
    print "\n# Parameters: "
    print "# BLOW_TIME is %6.4f  sec " % (blow_time)
    print "# GRIP_PRESSURE_ALMOST_ATM is %6.4f Pa\n" % (GRIP_PRESSURE_ALMOST_ATM)



#
# move to courier approach position
#

self.singleStep(dest_z + offset, dest_th)
if verbose:
    print"________________________________________\n"
    print"Moving into position, starting to drop."
    print"________________________________________\n"
self.sleep(2)


#Controller , executed 3 - if still vacuum, blow a little air, start drop
predicate = self.inBox(dest_z-Z_DIFF_MIN, dest_z+Z_DIFF_MAX,
                       dest_th-TH_DIFF_MIN,dest_th+TH_DIFF_MAX)

if test_gripper == 1:
    predicate.max_gripper1_pressure = GRIP_PRESSURE_ALMOST_ATM
elif test_gripper == 2:
    predicate.max_gripper2_pressure = GRIP_PRESSURE_ALMOST_ATM
```

```
for gr in grippers:
    self.grip_set(gr, GRIP_BLOW, True)
controller = self.stopAt(dest_z, dest_th)
controller.limit = (40, controller.limit[1])
controller.switch_time = 0.0075

action = FbControllerAction(controller, predicate)
action.addStartCallback(self.tag("StartDropped"))
id = self.insert(action)
print"Start drop ctrl"

#Controller , executed 4

for gr in grippers:
    self.grip_set(gr, GRIP_OFF)

predicate = self.inBox(dest_z-Z_DIFF_MIN, dest_z+Z_DIFF_MAX,
                       dest_th-TH_DIFF_MIN,dest_th+TH_DIFF_MAX)

if test_gripper == 1:
    predicate.min_gripper1_pressure = GRIP_PRESSURE_ALMOST_ATM
elif test_gripper == 2:
    predicate.min_gripper2_pressure = GRIP_PRESSURE_ALMOST_ATM


controller = self.stopAt(dest_z, dest_th)
controller.limit = (40, controller.limit[1])
action = FbControllerAction(controller, predicate)
self.insert(action)
print "air pressure high ctrl"

#Controller , executed 2 - move very close
for gr in grippers:
    self.grip_set(gr, GRIP_SUCK)

controller = self.moveTo(dest_z, dest_th)
predicate = self.inBox(max_z = dest_z+Z_DIFF_MAX+2.0, min_th = dest_th-\
                 TH_DIFF_MIN, max_th = dest_th+TH_DIFF_MAX)
id = self.insert(FbControllerAction(controller, predicate), id)
print "Move very close ctrl"

#Controller , executed 1  - move closer
action = FbControllerAction(self.moveTo(dest_z+2, dest_th),
                            self.inBox())
def_id = self.insert(action, id)
print "Move close ctrl"
self.truncate(def_id)

self.waitFor("StartDropped")

#Controller to blow air for BLOW_TIME seconds
```

```
        for gr in grippers:
            self.grip_set(gr, GRIP_BLOW)

        predicate = self.inBox(dest_z-Z_DIFF_MIN, dest_z+Z_DIFF_MAX,
                               dest_th-TH_DIFF_MIN,dest_th+TH_DIFF_MAX)

        controller = self.stopAt(dest_z, dest_th)
        controller.limit = (40, controller.limit[1])
        action = FbControllerAction(controller, predicate)
        action.addStartCallback(self.tag("Blowing"))
        blow_id=self.insert(action)
        print "Blowing air ctrl"

        self.waitFor("Blowing")
        self.sleep(blow_time)
        self.remove(blow_id)

        for gr in grippers:
            self.grip_set(gr, GRIP_OFF)

        #Controller to assure that item has dropped (== air pressure is up)

        predicate = self.inBox(dest_z-Z_DIFF_MIN, dest_z+Z_DIFF_MAX,
                               dest_th-TH_DIFF_MIN,dest_th+TH_DIFF_MAX)

        if test_gripper == 1:
            predicate.min_gripper1_pressure = GRIP_PRESSURE_ALMOST_ATM
        elif test_gripper == 2:
            predicate.min_gripper2_pressure = GRIP_PRESSURE_ALMOST_ATM

        controller = self.stopAt(dest_z, dest_th)
        controller.limit = (40, controller.limit[1])
        action = FbControllerAction(controller, predicate)
        action.addStartCallback(self.tag("Dropped"))
        id = self.insert(action)
        print "drop assurance ctrl"

        self.waitFor("Dropped")
        if verbose:
            print"____________________________________\n"
            print"           Item dropped."
            print"____________________________________\n"
        self.sleep(0.1)
        self.singleStep(dest_z + offset, dest_th)

        return 1
```

A.3 Screw

```
    # osti_screw(self, dest_z, dest_th, offset, grippers, total_turn, \
              first_turn=math.pi, turn_back=0.1, pitch=0.0, verbose=1)
    #
    # performs movements to screw lens fixation rings into collimeter housing, runs on OHM
    #
    # Context: OSTI Project, Demo on 9/20/06
    # Version 1.01, 9/19/06
    # Author: Cornelius Niemeyer, cornelius.niemeyer@gmail.com
    #
    # Parameters:
    # - total_turn : Total turn in radians; Screwing in is >0
    # - first_turn : Size of first turn when trying to catch thread (in radians); Screwing
    #                in is >0
    # - turn_back  : Angle to rotate back after each turn to ease the tension on the ring
    #                (in radians)
    # - pitch      : Pitch per revolution in mm
    # - blow       : if set to true, blowing will be activated when going up.

    def osti_screw(self, dest_z, dest_th, offset, grippers, total_turn, first_turn=math.pi,\
            turn_back=0.1, pitch=0.0, blow=0, verbose=1):

        #Air and vacuum channels  (IF POSSIBLE DEFINE THIS GLOBALLY !!)
        GRIP_PART_TOOL = 1      # air/vac port #1 is assigned to part tool
        GRIP_PART = 2           # air/vac port #2 is assigned to part
        GRIP_SUCK = -1
        GRIP_OFF = 0
        GRIP_BLOW = 1

        #Movement tolerances (for predicates)
        TH_DIFF_MAX=0.04                #in Radians
        TH_DIFF_MIN=0.04                #in Radians
        TH_DIFF_SINGLESTEP=0.04         #in Radians
        Z_DIFF_MAX=2.5  #3.0            #in mm
        Z_DIFF_MIN=0.3  #0.5            #in mm

        #WORKING TEST VALUES:
        #------------------------
        #dest_z     = 51.0
        #dest_th    = 6*math.pi/4.0
        #
        #Turning angles:
        #TOTAL_TURN = 560.0/180.0*math.pi
        #FIRST_TURN = 180.0/180.0*math.pi
        #TURN_BACK  = 0.1
        #
        #Pitch:
        #PITCH      = 0.6


        if verbose:
            print "\n##################################################################"
            print "#                   OHM Screwdriver function                    #"
            print "#                                                               #"
            print "#                 Version 1.01 - OSTI Project                   #"
            print "#               (c) Cornelius Niemeyer, MSL 2006                #"
            print "################################################################"
            print "\n# Turning angles [radians]: "
            print "# TOTAL_TURN is %6.4f" % (total_turn)
            print "# FIRST_TURN is %6.4f" % (first_turn)
            print "# TURN_BACK is  %6.4f\n" % (turn_back)
            print "\n# Pitch [mm]: "
            print "# PITCH is %6.4f" % (pitch)


        #Decompose Screwing revolutions

        noHT=int((total_turn-first_turn) / math.pi)   #Number of half turns after initial half turn
        resTurn=total_turn-first_turn-(noHT)*math.pi  #Residual radians to turn in last move





        #
        # move to courier approach position
        #

        self.singleStep(dest_z + offset, dest_th)
        self.sleep(2)

        #Controller 1, executed 3
        predicate = self.inBox(dest_z-Z_DIFF_MIN, dest_z+Z_DIFF_MAX,dest_th-TH_DIFF_MIN, \
            dest_th+ TH_DIFF_MAX)

        for gr in grippers:
            self.grip_set(gr, GRIP_SUCK)

        controller = self.stopAt(dest_z, dest_th)
        controller.limit = (40, controller.limit[1])
        controller.switch_time = 0.0075

        action = FbControllerAction(controller, predicate)
        action.addStartCallback(self.tag("MovedToPosition"))
        id = self.insert(action)

        #Controller 2, executed 2
        controller = self.moveTo(dest_z, dest_th)
        predicate = self.inBox(max_z = dest_z+2.0+Z_DIFF_MAX, min_th = dest_th-TH_DIFF_MIN,
                               max_th = dest_th+TH_DIFF_MAX)
        id = self.insert(FbControllerAction(controller, predicate), id)

        #Controller 3, executed 1
        action = FbControllerAction(self.moveTo(dest_z+2.0, dest_th),
                                    self.inBox())
        def_id = self.insert(action, id)
        self.truncate(def_id)
        self.waitFor("MovedToPosition")

        if verbose:
            print"____________________________________\n"
            print"Moved to Position, Beginning to Screw"
            print"____________________________________\n"

        #
        #First screwing motion, Turning first_turn radiants, controller 4 , executed 4
        #

        predicate = self.inBox(dest_z-Z_DIFF_MIN, dest_z+Z_DIFF_MAX)

        self.singleStep(dest_z, dest_th-first_turn-turn_back,th_margin=TH_DIFF_SINGLESTEP,\
            truncate_action_list=0, omega_max=0.5)
        #controller = self.moveTo(dest_z, dest_th-first_turn,omega_max=0.5)
        #controller.limit = (40, controller.limit[1])
        #action = FbControllerAction(controller, predicate)
        #self.insert(action)

        #predicate = self.inBox(dest_z-Z_DIFF_MIN, dest_z+Z_DIFF_MAX,dest_th-first_turn, \
        #      dest_th-first_turn+TH_DIFF_MAX)
        #controller = self.stopAt(dest_z, dest_th-first_turn)
        #controller.limit = (40, controller.limit[1])
        #action = FbControllerAction(controller, predicate)
        #action.addStartCallback(self.tag("Screwed"))
        #self.insert(action)



        #
        #Turning a bit back to ease the tension controller 5 , executed 5
        #

        predicate = self.inBox(dest_z-Z_DIFF_MIN, dest_z+Z_DIFF_MAX,dest_th-first_turn- \
            turn_back- TH_DIFF_MIN, dest_th-first_turn+TH_DIFF_MAX)

        controller = self.moveTo(dest_z, dest_th-first_turn,omega_max=0.3)
        controller.limit = (40, controller.limit[1])
        action = FbControllerAction(controller, predicate)
        action.addStartCallback(self.tag("TurnedBack"))
        turn_id = self.insert(action)
        print "1. Screwing Motion: Turn back Ctrl"

        self.waitFor("TurnedBack")
        self.sleep(2.5)

        #
        #going up after turning, controller 6 executed 6
        #
        if blow:
            for gr in grippers:
                self.grip_set(gr, GRIP_BLOW)

        predicate = self.inBox(min_th = dest_th-first_turn-TH_DIFF_MIN,max_th = dest_th-
            first_turn+TH_DIFF_MAX)

        controller = self.moveTo(dest_z+10, dest_th-first_turn)
        controller.limit = (40, controller.limit[1])
        action = FbControllerAction(controller, predicate)
        self.insert(action)
        print "1. Screwing Motion: Go up Ctrl"

        #
        #Assuring arrival , controller 7, executed 7
        #

        predicate = self.inBox(dest_z+10.0-Z_DIFF_MIN, dest_z+10.0+Z_DIFF_MAX,
                               min_th = dest_th-first_turn-TH_DIFF_MIN,max_th = dest_th- \
            first_turn+ TH_DIFF_MAX)
        controller = self.stopAt(dest_z+10, dest_th-first_turn)
        controller.limit = (40, controller.limit[1])
        action = FbControllerAction(controller, predicate)
        action.addStartCallback(self.tag("Turned"))
        go_up_id=self.insert(action)
        print "1. Screwing Motion: Assuring arrival Ctrl"

        #
        #Setting up for Screwing loop, Ring is assumed to have caught the thread
        #

        self.waitFor("Turned")
        self.truncate(go_up_id)
        not_finished =1
        halfTurnCounter=0
        PITCH_DIFF=pitch*first_turn/(2.0*math.pi)

        if noHT==0:
            SL_TURN=resTurn
            turn_back=0.0
        else:
            SL_TURN=math.pi




        #
        #Screwing loop
        #

        if verbose:
            print"___________________________________\n"
            print"Made first turn, starting screwing loop"
            print"___________________________________\n"
```

```
while not_finished:


    #
    #Turn Back
    #
    for gr in grippers:
        self.grip_set(gr, GRIP_OFF)

    controller = self.moveTo(dest_z+offset, dest_th)
    predicate = self.inBox(dest_z-Z_DIFF_MIN+offset, dest_z+Z_DIFF_MAX+offset)
    tb_id=self.insert(FbControllerAction(controller, predicate))
    print "S_Loop: Turn back Ctrl."
    #self.truncate(tb_id)

    #
    #Go down
    #
    for gr in grippers:
        self.grip_set(gr, GRIP_SUCK)

    #controller = self.moveTo(dest_z, dest_th,vz_max=25.0)
    predicate = self.inBox(min_th = dest_th-TH_DIFF_MIN, max_th = dest_th+TH_DIFF_MAX)
    #self.insert(FbControllerAction(controller, predicate))
    print "S_Loop: Go down Ctrl."

    self.singleStep(dest_z-PITCH_DIFF, dest_th, th_margin=TH_DIFF_SINGLESTEP, \
        truncate_action_list=0, vz_max=25.0, move_predicate=predicate)

    #
    #Turn SL_TURN radians (normally PI)
    #
    predicate = self.inBox(dest_z-PITCH_DIFF-Z_DIFF_MIN, dest_z-PITCH_DIFF+Z_DIFF_MAX)

    self.singleStep(dest_z-PITCH_DIFF, dest_th-SL_TURN-turn_back, th_margin= \
        TH_DIFF_SINGLESTEP, truncate_action_list=0, omega_max=0.5, move_predicate= \
        predicate)


    #
    #Turn Back to ease tension
    #
    predicate = self.inBox(dest_z-PITCH_DIFF-Z_DIFF_MIN, dest_z-PITCH_DIFF+Z_DIFF_MAX, \
        dest_th-SL_TURN-TH_DIFF_MIN-turn_back, dest_th-SL_TURN+TH_DIFF_MAX)

    controller = self.moveTo(dest_z-PITCH_DIFF, dest_th-SL_TURN,omega_max=0.3)
    controller.limit = (40, controller.limit[1])
    action = FbControllerAction(controller, predicate)
    tagString="TurnedBack %i" %(noHT)
    action.addStartCallback(self.tag(tagString))
    self.insert(action)
    print "S_Loop: Turn back to ease tension Ctrl."
```

```
        self.waitFor(tagString)
        self.sleep(2.5)

        #
        #Go up
        #
        if blow:
            for gr in grippers:
                self.grip_set(gr, GRIP_BLOW)

        predicate = self.inBox(min_th = dest_th-SL_TURN-TH_DIFF_MIN,max_th = dest_th- \
            SL_TURN+ TH_DIFF_MAX)
        #self.singleStep(dest_z+10, dest_th-SL_TURN+turn_back, th_margin=0.04, \
             truncate_action_list=0, omega_max=0.5, move_predicate=predicate)
        controller = self.moveTo(dest_z+offset, dest_th-SL_TURN)
        controller.limit = (40, controller.limit[1])
        action = FbControllerAction(controller, predicate)
        #action.addStartCallback(self.tag("Turned_PI"))
        gu_id=self.insert(action)
        print "S_Loop: Go up Ctrl."


        predicate = self.inBox(dest_z+offset-Z_DIFF_MIN, dest_z+offset+Z_DIFF_MAX, \
                           min_th = dest_th-SL_TURN-TH_DIFF_MIN,max_th = dest_th-SL_TURN+ \
                           TH_DIFF_MAX)
        controller = self.stopAt(dest_z+offset, dest_th-SL_TURN)
        controller.limit = (40, controller.limit[1])
        action = FbControllerAction(controller, predicate)
        action.addStartCallback(self.tag("Turned_PI"))
        go_up_id=self.insert(action)
        print "S_Loop: Assuring arrival Ctrl"




        self.waitFor("Turned_PI")
        self.truncate(go_up_id)
        self.remove(go_up_id)
        halfTurnCounter=halfTurnCounter+1
        PITCH_DIFF=PITCH_DIFF+pitch*SL_TURN/(2*math.pi)   #unless doing weird things,
                                                          #this will always be pitch/2

        if noHT==0:                                       #Residual already done.
            not_finished =0
        if halfTurnCounter == (noHT-1):                   #one more turn to go
            if resTurn <= TH_DIFF_MIN:                    #not worth to do residual...
                turn_back=0.0
        if halfTurnCounter == noHT:                       #all HT done
            if resTurn > TH_DIFF_MIN:                     #Is it worth to turn back ?
                SL_TURN = resTurn
                turn_back=0.0                             #don't want to turn back,
                                                          # screwing in tightly
                                                          #Could be used to screw in
                                                          # even tighter when set <0
            else :
                not_finished =0
        if halfTurnCounter > noHT:                        #all HT +Residual done
            not_finished =0


    if verbose:
        print"____________________________________\n"
        print"Ended screwing loop"
        print"____________________________________\n"


    self.sleep(1)
    for gr in grippers:
        self.grip_set(gr, GRIP_OFF)

    self.singleStep(dest_z + offset, dest_th)

    return 1
```

B Vision

B.1 How to set up the vision system

- Connect the camera feed to the frame grabber. Assure that the camera is running.
- Run the vision server executable on the system with the frame grabber card. When the server starts it displays the message *“Portal ID is XXX”* with the IPT “address” instead of XXX.
- Copy this address string and set is as value of an environment variable named *VISION_SPEC* on the manipulator system.

  (e.g. `export VISION_SPEC=ipt:ocelot|1500,0`)
- start *maniphead*. The vision servers IPT specification should be recognized and available in Python.

B.2 Vision server source code

- vision_srv.cc:

```
/************************************************************************
 |                              vision_serv.cc                          |
 |                                                                      |
 |                          AAA Vision package 2.0                      |
 |----------------------------------------------------------------------|
 | Vision server application main function. Initializes AAAVision and   |
 | PXC_Control class. Sets up an IPT container manager. Implement your  |
 | IPT messages and message handlers in here.                           |
 |                                                                      |
 |                                                                      |
 |   Context: Vision based pick and place project,                      |
 |                                                                      |
 |   See also: "Pick and Place in a Minifactory environment", Diploma   |
 |              thesis, Cornelius Niemeyer, ETH Zuerich, 12/2006        |
 |                                                                      |
 |                                                                      |
 |                                                                      |
 | (c) 2006 Cornelius Niemeyer, MSL    |
 |                                                                      |
 |----------------------------------------------------------------------|
 |                                                                      |
 | TARGET PLATFORM: QNX (6.3)                                           |
 |                                                                      |
 ************************************************************************/

#include "iostream.h"
#include "stdio.h"
#include "time.h"
#include "AAAvision.h"

#include <AAA/AAA.h>
#include <AAA/FoContainer.h>
#include <AAA/communications/FbMessage.h>
#include <AAA/communications/FoMessageHandler.h>
#include <AAA/communications/FbContainerManager.h>

#define VERBOSE 1

#define VISION_SAVE_CURRENT_BMP_MSG "VISION_SAVE_CURRENT_BMP_MSG"
#define VISION_SAVE_CURRENT_BMP_FMT "string"

#define VISION_GET_CLOSEST_SPHERE_MSG "VISION_GET_CLOSEST_SPHERE_MSG"
#define VISION_GET_CLOSEST_SPHERE_FMT ""
#define VISION_FOUND_CLOSEST_SPHERE_MSG "VISION_FOUND_CLOSEST_SPHERE_MSG"
#define VISION_FOUND_CLOSEST_SPHERE_FMT "{ float, float }"

#define VISION_GET_CLOSEST_SPHERE_D_MSG "VISION_GET_CLOSEST_SPHERE_D_MSG"
#define VISION_GET_CLOSEST_SPHERE_D_FMT ""
```

```
#define VISION_FOUND_CLOSEST_SPHERE_D_MSG "VISION_FOUND_CLOSEST_SPHERE_D_MSG"
#define VISION_FOUND_CLOSEST_SPHERE_D_FMT "{ float, float, int }"


struct save_current_bmp_param{
//int a;
char* filename;
};


void save_current_bmp_hand(FoContainer* recv, FbMessage* msg, void* data)
{
if (VERBOSE) printf("Entering save_current_bmp_handler.\n");
AAAVision* AV = (AAAVision*) data;
save_current_bmp_param* param= (save_current_bmp_param*)msg->getData();
    printf("Saving image as %s\n", param->filename);
AV->saveCurrentBMPImage( param->filename);
if (VERBOSE) printf("Image saved.\n");
}


void get_closest_sphere_hand(FoContainer* recv, FbMessage* msg, void* data)
{
if (VERBOSE) printf("Entering get_closest_sphere_handler.\n");
AAAVision* AV = (AAAVision*) data;
CvPoint2D32f res = AV->getClosestSphereCenter(15,15,130,VERBOSE);

recv->replyTo(msg,VISION_FOUND_CLOSEST_SPHERE_MSG, &res);
if (VERBOSE) printf("Closest sphere send back.\n");
}


void get_closest_sphere_d_hand(FoContainer* recv, FbMessage* msg, void* data)
{
if (VERBOSE) printf("Entering get_closest_sphere_d_handler.\n");
AAAVision* AV = (AAAVision*) data;
CalcDistanceRes res = AV->getClosestSphereDistance(15,15,130,VERBOSE);

recv->replyTo(msg,VISION_FOUND_CLOSEST_SPHERE_D_MSG, &res);
if (VERBOSE) printf("Distance to closest sphere send back.\n");
}


int main(int argc, char *argv[])
{

    AAA::basicInit();

    // set up the container manager
    char buffer[200];
    int port_num = 1500;   // fixed at 1500 for now
    sprintf(buffer, "ipt: port=%d;", port_num);
    FbContainerManager* manager = AAA::getContainerManager(buffer);
    FoContainer* portal = new FoContainer();
    portal->ref();
    portal->setName("visionPortal");

    printf("Portal ID is %s\n", portal->getSpecString());

    // create the AAA vision  class
    AAAVision AV;

    portal->registerMessage(VISION_SAVE_CURRENT_BMP_MSG,VISION_SAVE_CURRENT_BMP_FMT);
    portal->registerMessage(VISION_GET_CLOSEST_SPHERE_MSG,VISION_GET_CLOSEST_SPHERE_FMT);
    portal->registerMessage(VISION_FOUND_CLOSEST_SPHERE_MSG,VISION_FOUND_CLOSEST_SPHERE_FMT);
    portal->registerMessage(VISION_GET_CLOSEST_SPHERE_D_MSG,VISION_GET_CLOSEST_SPHERE_D_FMT);
    portal->registerMessage(VISION_FOUND_CLOSEST_SPHERE_D_MSG,VISION_FOUND_CLOSEST_SPHERE_D_FMT);

    portal->addMessageHandler(VISION_SAVE_CURRENT_BMP_MSG,
            new FoRoutineMessageHandler(save_current_bmp_hand, &AV));
    portal->addMessageHandler(VISION_GET_CLOSEST_SPHERE_MSG,
            new FoRoutineMessageHandler(get_closest_sphere_hand, &AV));
    portal->addMessageHandler(VISION_GET_CLOSEST_SPHERE_D_MSG,
            new FoRoutineMessageHandler(get_closest_sphere_d_hand, &AV));

    printf("Vision Server is set up.\n");
    manager->mainLoop();
}
```

- AAAVision.h:

```
/***************************************************************************
 |                             AAAvision.cc                                |
 |                                                                         |
 |                         AAA Vision package 2.0                          |
 |-------------------------------------------------------------------------|
 | Headerfile for AAAvision image processing class. Implement all          |
 | your vision code in here.                                               |
 |                                                                         |
 |                                                                         |
 |   Context: Vision based pick and place project,                         |
 |                                                                         |
 |   See also: "Pick and Place in a Minifactory environment", Diploma      |
 |              thesis, Cornelius Niemeyer, ETH Zuerich, 12/2006           |
 |                                                                         |
 |                                                                         |
 |                                                                         |
 | (c) 2006 Cornelius Niemeyer, MSL    |
 |                                                                         |
 |-------------------------------------------------------------------------|
 |                                                                         |
 | TARGET PLATFORM: QNX (6.3)                                              |
 |                                                                         |
 *************************************************************************/




#ifndef _AAAVISION_H_
#define _AAAVISION_H_


#include "pxc_controls.h"
#include "cvAAA.h"
#include "cxcore.h"
#include "cv.h"


struct CalcDistanceRes
{
float x_dist;
float y_dist;
int  success;
};

class AAAVision
{

private:
    PXC_Controls pxc;
    CvPoint tweezer;

public:
AAAVision();
~AAAVision();
int saveCurrentBMPImage(char* filename);
    CvBox2D getClosestSphere(int min_a, int min_b, int threshold , bool verbose );
    CvPoint2D32f getClosestSphereCenter(int min_a, int min_b, int threshold , bool verbose);
CalcDistanceRes getClosestSphereDistance(int min_a, int min_b, int threshold , bool verbose);

};

#endif //_AAAVISION_H_
```

- AAAVision.cc:

```
/*************************************************************************
 |                              AAAvision.cc                             |
 |                                                                       |
 |                          AAA Vision package 2.0                       |
 |-----------------------------------------------------------------------|
 | Implementation of AAAvision image processing class. Implement all     |
 | your vision code in here.                                             |
 |                                                                       |
 | Currently the follwing functions are implemented:                     |
 |   -  SaveCurrentBMPImage : Grabs a frame and writes it into a .bmp     |
 |      file                                                              |
 |   -  GetClosestSphere: returns ellipse box of closest sphere to        |
 |      the specified gripper center point                                |
 |   -  GetClosestSphereCenter: returns center point of closest sphere    |
 |      to the specified gripper center point                             |
 |   -  GetClosestSphereDistance: returns distance of closest sphere      |
 |      to the specified gripper center point                             |
 |                                                                        |
 |   Context: Vision based pick and place project,                        |
 |                                                                        |
 |   See also: "Pick and Place in a Minifactory environment", Diploma     |
 |             thesis, Cornelius Niemeyer, ETH Zuerich, 12/2006           |
 |                                                                        |
 |                                                                        |
 |                                                                        |
 | (c) 2006 Cornelius Niemeyer, MSL    |
 |                                                                        |
 |------------------------------------------------------------------------|
 |                                                                        |
 | TARGET PLATFORM: QNX (6.3)                                             |
 |                                                                        |
 ***************************************************************************/


#include "AAAvision.h"

#include <math.h>
#include <stdio.h>

AAAVision::AAAVision()
{
printf("Initializing Framegrabber...\n");
if (!pxc.pxc_init()) printf("Framegrabber initialisation failed ! \n");
pxc.setCamera(0);
pxc.setComposite();

//gripper center point.
//should be recognised automaticly by image analysis
//replace by automatic algorithm
//coordiantes with respect to image left upper corner.
tweezer.x=495;
tweezer.y=230;


//working estimate in m (5x magnifying objective)
pxc.pixellength=0.00000374;
}

AAAVision::~AAAVision()
{
```

```
pxc.pxc_term();
}

int AAAVision::saveCurrentBMPImage(char* filename)
{
CvMat image;
image=pxc.grabCvImage();
//pxc.flib.WriteBMP( pxc.fgframe,"/root/testbbb.bmp",1);
if(cvSaveImage( filename, &image ))
{
printf ("Image written to %s .\n", filename);
return 1;
}else{
printf ("Writing Image to %s failed !\n", filename);
return 0;
}
}

CvBox2D32f  AAAVision::getClosestSphere(int min_a, int min_b, int threshold, bool verbose )
{

    CvMemStorage* stor;
    CvSeq* cont;
    CvBox2D32f* box;
    CvBox2D32f result ={{0}};
    CvPoint* PointArray;
    CvPoint2D32f* PointArray2D32f;
CvMat *image;
CvMat *image_th;
double min_dist=10000000.0;
double dist;
int nEllipses=0;

CvMat imageRGB = pxc.grabCvImage();


    // convert image to be grayscale

image = cvCreateMat(pxc.height,pxc.width,CV_8UC1);
    cvCvtColor(&imageRGB, image, CV_RGB2GRAY);

    image_th=cvCloneMat(image);

    // Create dynamic structure and sequence.
    stor = cvCreateMemStorage(0);

    cont = cvCreateSeq(CV_SEQ_ELTYPE_POINT, sizeof(CvSeq), sizeof(CvPoint) , stor);

// Threshold the source image.
    cvThreshold( image, image_th, threshold, 255, CV_THRESH_BINARY );

    //detect contours
```

```
cvFindContours( image_th, stor, &cont, sizeof(CvContour),
                    CV_RETR_LIST, CV_CHAIN_APPROX_NONE, cvPoint(0,0));

    // approximate contours by ellipses
    double dist_x;
    double dist_y;
    int count;


    for(;cont;cont = cont->h_next)
    {

        count = cont->total; // number of points in contour


        // Number of points must be more than or equal to 6 (for cvFitEllipse_32f).
        if( count < 6 )
            continue;

        // Alloc memory for contour point set.
        PointArray = (CvPoint*)malloc( count*sizeof(CvPoint) );
        PointArray2D32f= (CvPoint2D32f*)malloc( count*sizeof(CvPoint2D32f) );

        // Alloc memory for ellipse data.
        box = (CvBox2D32f*)malloc(sizeof(CvBox2D32f));

        // Get contour point set.
        cvCvtSeqToArray(cont, PointArray, CV_WHOLE_SEQ);

        // Convert CvPoint set to CvBox2D32f set.
        for(int i=0; i<count; i++)
        {
            PointArray2D32f[i].x = (float)PointArray[i].x;
            PointArray2D32f[i].y = (float)PointArray[i].y;
        }
        // Fits ellipse to current contour.
        cvFitEllipse(PointArray2D32f, count, box);

        // Draw current contour.
        //cvDrawContours(image,cont,CV_RGB(255,255,255),
        //                CV_RGB(255,255,255),0,1,8,cvPoint(0,0));


        if ((box->size.width>min_a)&&(box->size.height>min_b))
        {
         nEllipses++;
         //calculate distance to center
         dist_x=tweezer.x-(double)box->center.x;
         dist_x=dist_x*dist_x;
           dist_y=tweezer.y-(double)box->center.y;
         dist_y=dist_y*dist_y;
         dist=sqrt(dist_x+dist_y);
         //if better, replace the best so far
         if (dist<min_dist){
         //printf("DEBUG: entered here");
         result.center=box->center;
         result.angle=box->angle;
         result.size=box->size;
         }
        }
        // Free memory.
        free(PointArray);
        free(PointArray2D32f);
        free(box);
    }

    if (verbose) {
     if (nEllipses>0) printf("%i sphere(s) found. Returning closest one.\n", nEllipses);
     else printf("No spheres found.\n");
    }

    cvReleaseMat(&image);
    cvReleaseMat(&image_th);
    cvReleaseMemStorage(&stor);
    return result;  //if none was found all values should be == 0
}


CvPoint2D32f AAAVision::getClosestSphereCenter(int min_a, int min_b,
                        int threshold , bool verbose)
{
CvPoint2D32f result;
CvBox2D32f sphere = getClosestSphere(min_a,min_b, threshold,verbose);

if (verbose) printf("Sphere center: %f, %f,", sphere.center.x, sphere.center.y);
if (sphere.size.height==0) {
result.x = -1;
result.y = -1;
}
else {
result= sphere.center;
}

return result;
}

/*
 * Returns relative distance from the center of a found sphere to the set center
 * cordinates of the tweezer in mm. The origin of the used coordinate system lies
 * at the upper left corner of the image
 * -> the coordinates have to be transformed including the current theta value for
 *    later use
 */
```

```
CalcDistanceRes AAAVision::getClosestSphereDistance(int min_a, int min_b,
                           int threshold , bool verbose)
{
    CalcDistanceRes result;
    CvPoint2D32f center= getClosestSphereCenter( min_a, min_b, threshold , verbose);
    if (center.x<0) {
     result.success= 0;
     if(verbose) printf("AAAVision::getClosestSphereDistance  :  No sphere found.\n");
     return result;
    }
    result.success=1;
    result.x_dist=(center.x-tweezer.x)*pxc.pixellength;
    result.y_dist=(center.y-tweezer.y)*pxc.pixellength;

    if(verbose) printf("Distance from tweezer CP to sphere center: %f, %f [m].\n",
                        result.x_dist,result.y_dist);
    return result;

}
```

- PXC_controls.h:

```
/****************************************************************************
 |                                pxc_controls.h                            |
 |                                                                          |
 |                            AAA Vision package 2.0                        |
 |--------------------------------------------------------------------------|
 | Header file of the pxc_controls class acting as interface between the    |
 | vision code and the PXC frame grabber card.                              |
 |                                                                          |
 | Context: Vision based pick and place project                             |
 |                                                                          |
 | See also: "Pick and Place in a Minifactory environment", Diploma         |
 |           thesis, Cornelius Niemeyer, ETH Zuerich, 12/2006               |
 |                                                                          |
 |                                                                          |
 |                                                                          |
 |                                                                          |
 | (c) 2006 Cornelius Niemeyer, MSL                        |
 |                                                                          |
 |--------------------------------------------------------------------------|
 |                                                                          |
 | TARGET PLATFORM: QNX (6.3)                                               |
 |                                                                          |
 ***************************************************************************/

#ifndef _PXC_CONTROLS_H_
#define _PXC_CONTROLS_H_

#include "cxcore.h"

extern "C" {
```

```
#include "pxc.h"
#include "frame.h"
}

class PXC_Controls{

private:

  FRAME __PX_FAR *fgframe;
int SetVideoParameters();

public:

/* structures that hold the function pointers for the libraries */

PXC200 pxc200;
FRAMELIB flib;

//////////////////////////////////////////////////////////////////////
// fgh is the frame grabber handle
//
unsigned long fgh;

//////////////////////////////////////////////////////////////////////
// fgframe holds the 2 frame handles while ImageData holds the 2 pointers
//


//DEBUG: THis could cause errors...
void *ImageData;
//unsigned char *ImageData;
int tagQ;
int bAquire;
int modelnumber;
short width, height;
double pixellength;

int ImageBPP;   //Bits per pixel
int ImageBPL;  //bits per line


PXC_Controls();
~PXC_Controls();
int pxc_init();
int pxc_term();
int setComposite();
int setSVideo();
int PXC_Controls::setCamera(int slot);
void SlantPattern(unsigned char *p, int dx, int dy, int bpp);

FRAME __PX_FAR *grabFrame();
    CvMat grabCvImage();
```

```
};

#endif //_PXC_CONTROLS_H_
```

- PXC_controls.cc:

```
/***************************************************************************
 |                               pxc_controls.cc                          |
 |                                                                        |
 |                           AAA Vision package 2.0                       |
 |------------------------------------------------------------------------|
 | Implementation of pxc_controls class acting as interface between the   |
 | vision code and the PXC frame grabber card.                            |
 |                                                                        |
 | The  PXC initialization procedure implemented below woks as follows:   |
 |   1. Open the PXC and FRAME libraries                                  |
 |   2. Allocate a frame grabber                                          |
 |   3. Determine the video type                                          |
 |        - important for determining frame buffer size                   |
 |   4. Get the model number                                              |
 |   5. Allocate a frame buffer                                           |
 |   6. Get a pointer to the frame buffer                                 |
 |                                                                        |
 |                                                                        |
 |   Context: Vision based pick and place project,                        |
 |                                                                        |
 |   See also: "Pick and Place in a Minifactory environment", Diploma     |
 |              thesis, Cornelius Niemeyer, ETH Zuerich, 12/2006          |
 |                                                                        |
 |                                                                        |
 | Partly based on sample code from Imagenation.                                 |
 |                                                                        |
 | (c) 2006 Cornelius Niemeyer, MSL    |
 |                                                                        |
 |------------------------------------------------------------------------|
 |                                                                        |
 | TARGET PLATFORM: QNX (6.3)                                             |
 |                                                                        |
 ***************************************************************************/


#include <stdio.h>

#include "pxc_controls.h"


#define PIXEL_TYPE PBITS_RGB24

static int videotype;
```

```
PXC_Controls::PXC_Controls()
{
}

PXC_Controls::~PXC_Controls()
{

}

/////////////////////////////////////////////////////////////////////
// Name: SetVideoParameters
/////////////////////////////////////////////////////////////////////
int PXC_Controls::SetVideoParameters()
{
////////////////////////////////////////////////////////////
// Determine the video type
//
videotype = pxc200.VideoType(fgh);
modelnumber = pxc200.GetModelNumber(fgh);
switch(videotype)
{
case 0:     /* no video */
printf("WARNING: Videotype not recognised.\n");
width = 0;
height = 0;
return 0;
break;
case 1:     /* NTSC */
printf("Video: NTSC / RS-170. \n");
width = 640;
height = 486;
break;
case 2:     /* CCIR */
printf("Video: CCIR / PAL. \n ");
width = 768;
height = 576;
break;
}

/*--------------------------------------------------------------------
The following values control scaling and decimation. For normal frames
they can be set to the height and width of the frame buffer.
--------------------------------------------------------------------*/
pxc200.SetWidth(fgh, width);
pxc200.SetHeight(fgh, height);
pxc200.SetXResolution(fgh, width);
pxc200.SetYResolution(fgh, height);

return 1;
}
```

```
int PXC_Controls::pxc_init()
{


////////////////////////////////////////////////////////////
// Open the frame library first
//
if(!FRAMELIB_OpenLibrary(&flib,sizeof(FRAMELIB)))
{
//puts("FrameLib Allocation failed.");
return ( 0 );
}

////////////////////////////////////////////////////////////
// Open the PXC200 library next
//
if(!PXC200_OpenLibrary(&pxc200,sizeof(PXC200)))
{
//puts("PXC200 OpenLibrary Failed. No frame grabber functions found.");
return ( 0 );
}

//width = FRAME_WIDTH;
//height = FRAME_HEIGHT;

////////////////////////////////////////////////////////////
// allocate any PXC200
//
fgh = pxc200.AllocateFG(-1);



////////////////////////////////////////////////////////////
// Get the PXC model number
// And write it to the model number widget
//
modelnumber = pxc200.GetModelNumber(fgh);

switch(modelnumber)
{
case PXC200_LC:
printf("PXC200L Framegrabber recognised.\n");
break;
case PXC200_LC_2:
printf("PXC200AL Framegrabber recognised.\n");
break;
case PXC200_F:
printf("PXC200F Framegrabber recognised.\n");
break;
case PXC200_F_2:
printf("PXC200AF Framegrabber recognised.\n");
break;
default:
printf("WARNING: Framegrabber model not recognised.\n");
break;
}

////////////////////////////////////////////////////////////
// Determine the video type
//   - this is important for determining the frame buffer size
//

if(!SetVideoParameters())
{
//puts("Set Video Parameters failed");
return 0;
}

////////////////////////////////////////////////////////////
// Allocate a frame buffer
// Use PBITS_RGB24 for color and PBITS_Y8 for monochrome
//
fgframe = pxc200.AllocateBuffer(width, height, PIXEL_TYPE);
if(fgframe == NULL)
{
//puts("Allocate buffer failed");
return 0;
}

////////////////////////////////////////////////////////////
// Get a pointer to the frame buffer
//
ImageData = flib.FrameBuffer(fgframe);

    //initialise Image Parameters
    ImageBPP = ((PIXEL_TYPE & 0xFF)+7) >> 3;
ImageBPL = width * ImageBPP;

return( 1 );

}
/////////////////////////////////////////////////////////////////////
// Name: SlantPattern
//
// Put a slant pattern in the buffer. This routine can be used to
// visually verify that a grab occured and that both fields were
// grabbed. This function is not used in this program but it has
// been left here because it can be useful. If you suspect that you
// may be grabbing only a single field, call this function prior to
// each grab. If only one field has been grabbed, you will see a
// ghosting of this slant pattern through your image.
//
/////////////////////////////////////////////////////////////////////
void PXC_Controls::SlantPattern(unsigned char *p, int dx, int dy, int bpp)
```

```
{
int x, y, i;
// unsigned char *p;


// get a buffer pointer from the frame handle
// p = frame.FrameBuffer(frh);

for(y=0; y<dy; ++y)
for(x=0; x<dx; ++x)
for(i=0; i<bpp; ++i)
*p++ = x+y;
}

//////////////////////////////////////////////////////////////////////
// Name: pxc_term
//////////////////////////////////////////////////////////////////////
int PXC_Controls::pxc_term()

{

////////////////////////////////////////////////////////////
// Wait until other stuff has terminated
// It will have always terminated by the time the library
// returns from this Grab by virtue of the order of operations.
//
pxc200.Grab(fgh, fgframe, 0);

////////////////////////////////////////////////////////////
// Deallocate the frame
//
if(fgframe)
flib.FreeFrame(fgframe);

////////////////////////////////////////////////////////////
// Free the FG
//
if(fgh)
pxc200.FreeFG(fgh);

////////////////////////////////////////////////////////////
// Close the libraries
//
FRAMELIB_CloseLibrary(&flib);
PXC200_CloseLibrary(&pxc200);

return(1);

}

int PXC_Controls::setComposite()
{
```

```
pxc200.SetChromaControl(fgh,NOTCH_FILTER);
return 1;
}

int PXC_Controls::setSVideo()
{
//////////////////////////////////////////////////
// The PXC200L and PXC200AL have S-viseo only on channel 1
//
if(modelnumber == PXC200_LC || modelnumber == PXC200_LC_2 )
{
// set the PXC channel to camera 1
pxc200.SetCamera(fgh,1,0);
 }
// turn on s-video mode
pxc200.SetChromaControl(fgh,SVIDEO);
return 1;
}

//Grabs a frame

FRAME __PX_FAR *PXC_Controls::grabFrame()
{
pxc200.Grab(fgh, fgframe, 0);
return fgframe;
}

//grabs a frame and coverts it to an OpenCV image
CvMat PXC_Controls::grabCvImage()
{
pxc200.Grab(fgh, fgframe, 0);

    //check frame type!
    if(PIXEL_TYPE == 0x0218) {
     CvMat *image;
    image = cvCreateMat(flib.FrameHeight(fgframe),flib.FrameWidth(fgframe),CV_8UC3);
     cvInitMatHeader(image,flib.FrameHeight(fgframe),flib.FrameWidth(fgframe),CV_8UC3,
                     flib.FrameBuffer(fgframe),CV_AUTOSTEP );

     return *image;
    }else{
     printf("CvMat PXC_Controls::grabImage()  : Not yet implemented for this type of image.\n");
     CvMat i;
     return i;
    }

}

int PXC_Controls::setCamera(int slot)
{
pxc200.SetCamera(fgh,slot,0);
return 1;
```

```
}
```

B.3 Vision client example source code

In the following exaple source code for a vision client application in Python on the manipulator will be given.

- First the messages to be send have to be defined and registered as e.g. as follows:

```
VISION_GET_CLOSEST_SPHERE_MSG = "VISION_GET_CLOSEST_SPHERE_MSG"
VISION_GET_CLOSEST_SPHERE_FMT = ""
VISION_FOUND_CLOSEST_SPHERE_MSG = "VISION_FOUND_CLOSEST_SPHERE_MSG"
VISION_FOUND_CLOSEST_SPHERE_FMT = "{ float, float }"

def setInterface(self, intf, desc, binder, action_list):

    #...
    self.registerMessage(VISION_GET_CLOSEST_SPHERE_MSG,VISION_GET_CLOSEST_SPHERE_FMT)
    self.registerMessage(VISION_FOUND_CLOSEST_SPHERE_MSG,VISION_FOUND_CLOSEST_SPHERE_FMT)
```

- Then messages can be sent and data extracted from responses as shown in the following code snippet:

```
def get_closest_sphere(self):
    print "get closest sphere begin"
    if self.getVisionProcessor() is None:
        print "no vision processor found"
        return None
    res= self.queryTo(self.visionProcessor,VISION_GET_CLOSEST_SPHERE_MSG, \
                      None, VISION_FOUND_CLOSEST_SPHERE_MSG)
    print "SPHERE:", res[0], res[1]
    if res[0] < 0:
        return None
    return (res[0], res[1])
```

The message definitions have to be identical at client and server.

B.4 How to extend the vision server

Extend the vision server and client:
Include new vision processing functions in the AAAVision class. The following example code illustrates how IPT messages are defined, registered and read and answered in the message handler in the main vision server file *vision_serv.cc*. When receiving, data will arrive as a struct, even if its only one field.

```
#define VISION_GET_CLOSEST_SPHERE_D_MSG "VISION_GET_CLOSEST_SPHERE_D_MSG"
#define VISION_GET_CLOSEST_SPHERE_D_FMT ""
#define VISION_FOUND_CLOSEST_SPHERE_D_MSG "VISION_FOUND_CLOSEST_SPHERE_D_MSG"
#define VISION_FOUND_CLOSEST_SPHERE_D_FMT "{ float, float, int }"


void get_closest_sphere_d_hand(FoContainer* recv, FbMessage* msg, void* data)
{
if (VERBOSE) printf("Entering get_closest_sphere_d_handler.\n");
AAAVision* AV = (AAAVision*) data;
CalcDistanceRes res = AV->getClosestSphereDistance(15,15,130,VERBOSE);

recv->replyTo(msg,VISION_FOUND_CLOSEST_SPHERE_D_MSG, &res);
if (VERBOSE) printf("Distance to closest sphere send back.\n");
}


int main(int argc, char *argv[])
{
    ...

    portal->registerMessage(VISION_GET_CLOSEST_SPHERE_D_MSG,VISION_GET_CLOSEST_SPHERE_D_FMT);
    portal->registerMessage(VISION_FOUND_CLOSEST_SPHERE_D_MSG,\
         VISION_FOUND_CLOSEST_SPHERE_D_FMT);
    portal->addMessageHandler(VISION_GET_CLOSEST_SPHERE_D_MSG,\
         new FoRoutineMessageHandler(get_closest_sphere_d_hand, &AV));

    ...

}
```

B.5 Hough transform algorithm testing code

- houghcircle.c :

```
/******************************************************************************
*
*
*  This is a test program developed to experiment with the hough transformation
*  circle recognition algorithm on specific imgages of small spheres.
*
*  Context: Vision based pick and place project, AAA, Minifactory
*
*  Version 1.00, 9/19/06
*  Authors: Cornelius Niemeyer <cornelius.niemeyer@gmail.com>
*           Loosely based on ellipse fitting OpenCv demo program by Denis Burenkov.
*
*  See also: "Pick and Place in a Minifactory environment", Diploma thesis,
*            Cornelius Niemeyer, ETH Zuerich, 12/2006
*
*  (c) 2006 MSL,CMU
```

```
*
******************************************************************************/
#ifdef _CH_
#pragma package <opencv>
#endif

#ifndef _EiC
#include "cv.h"
#include "highgui.h"
#include <stdio.h>
#endif

int slider_pos1 = 70;
int slider_pos2 = 70;
int slider_pos3 = 100;



// Load the source image. HighGUI use.
IplImage *image02 = 0, *image03 = 0, *image04=0,*image05=0, *image06=0;

void process_image(int h);
void drawCross(CvArr* img, CvPoint center,double length_x, double length_y, \
               CvScalar color, int thickness, int line_type);

int main( int argc, char** argv )
{
    const char* filename = argc == 2 ? argv[1] : (char*)"/4.bmp";
    const char* filename2 =(char*)"/4r.bmp";
    const char* filename3 =(char*)"/4r_b.bmp";

    // load image and force it to be grayscale
    if( (image03 = cvLoadImage(filename, 0)) == 0 )
        return -1;

    // Create the destination images
    image02 = cvCloneImage( image03 );
    //image04 = cvCloneImage( image03 );

if( (image04 = cvLoadImage(filename, -1)) == 0 )
        return -1;
    image05=cvCloneImage( image04);
    image06=cvCloneImage( image04);

    // Create windows.
    cvNamedWindow("Source", 1);
    cvNamedWindow("Result", 1);
    cvNamedWindow("SourRes", 1);
    cvNamedWindow("Threshhold", 1);

    // Show the image.
    cvShowImage("Source", image03);
    // Create toolbars. HighGUI use.
    cvCreateTrackbar( "Param1", "Result", &slider_pos1, 400, process_image );
    cvCreateTrackbar( "Param2", "Result", &slider_pos2, 400, process_image );
    cvCreateTrackbar( "Threshhold", "Threshhold", &slider_pos3, 255, process_image );
    process_image(0);
    // Wait for a key stroke; the same function arranges events processing
    cvWaitKey(0);

    cvSaveImage(filename2, image05);
    cvSaveImage(filename3, image04);


    cvReleaseImage(&image02);
    cvReleaseImage(&image03);
    cvReleaseImage(&image04);
    cvReleaseImage(&image05);
    cvReleaseImage(&image06);

    cvDestroyWindow("Source");
    cvDestroyWindow("Result");
    cvDestroyWindow("SourRes");
    cvDestroyWindow("Threshhold");

    return 0;
}

// Define trackbar callback functon. This function find contours,
// draw it and approximate it by ellipses.
void process_image(int h)
{
    CvMemStorage* stor;

    cvZero(image02);
    // Create dynamic structure and sequence.
    stor = cvCreateMemStorage(0);


    cvThreshold( image03, image02, slider_pos3, 255, CV_THRESH_BINARY );
    cvShowImage( "Threshhold", image02 );


    CvSeq* circles = cvHoughCircles( image02, stor, CV_HOUGH_GRADIENT, 2, \
                        image03->height/10, slider_pos1, slider_pos2 );


    cvZero(image04);
    image05=cvCloneImage( image06);
    printf("%i circles recognised",circles->total);
    int i;
    for( i = 0; i < circles->total; i++ )
    {
         float* p = (float*)cvGetSeqElem( circles, i );
         cvCircle( image04, cvPoint(cvRound(p[0]),cvRound(p[1])), 3, \
                   CV_RGB(0,255,0), -1, 8, 0 );
         cvCircle( image04, cvPoint(cvRound(p[0]),cvRound(p[1])), cvRound(p[2]), \
                   CV_RGB(255,0,0), 3, 8, 0 );
         cvCircle( image05, cvPoint(cvRound(p[0]),cvRound(p[1])), 3, \
                   CV_RGB(0,255,0), -1, 8, 0 );
         cvCircle( image05, cvPoint(cvRound(p[0]),cvRound(p[1])), cvRound(p[2]), \
                   CV_RGB(255,0,0), 3, 8, 0 );
    }



    // Show image. HighGUI use.
    cvShowImage( "Result", image04 );
    cvShowImage( "SourRes", image05 );
}


void drawCross(CvArr* img, CvPoint center,double length_x, double length_y, \
               CvScalar color, int thickness, int line_type)
{
    //horizontal line
    CvPoint pt1=cvPoint(center.x-cvRound(length_x*0.5),center.y);
    CvPoint pt2=cvPoint(center.x+cvRound(length_x*0.5),center.y);
cvLine(img,  pt1, pt2,color,thickness,line_type, 0 );
    //vertical line
    pt1=cvPoint(center.x,center.y-cvRound(length_y*0.5));
    pt2=cvPoint(center.x,center.y+cvRound(length_y*0.5));
cvLine(img,  pt1, pt2,color,thickness,line_type, 0 );

}



#ifdef _EiC
main(1,"houghcircle.c");
#endif
```

C Vision Demo

C.1 .fac file

- visionDemo.fac :

```
file base_frame.aaa {
    children {
        file lg_platen.aaa {
            name P1
            matrix [ 1 0 0 0 0 1 0 0 0 0 1 0 0 0 570 1 ]
        }
        cached vole ipt:vole|1400,interface {
            name Vole
            matrix [ 1 0 0 0 0 1 0 0 0 0 1 0 450 -330 655 1 ]
            program {
from VisionDemoCourierProgram import VisionDemoCourierProgram

program = VisionDemoCourierProgram()
}
            member home {
                matrix [ 1 0 0 0 0 1 0 0 0 0 1 0 -450 330 15 1 ]
                children {
                }
            }
            member motor {
                matrix [ 1 0 0 0 0 1 0 0 0 0 1 0 179.503 -441.91 0 1 ]
                children {
                }
            }
        }

        file bridge.aaa {
            # CHANGE POINT: change the y value below to move the bridge along the frame
            matrix [ 1 0 1.77745e-15 0 0 1 0 0 -1.77745e-15 0 1 0 5.57899e-05 -211.45 \
                    920 1 ]
            member crossbar {
                # CHANGE_POINT: increasing the z value below makes the manipulator go more \
                #               shallowly (moves the bridge up and down)
                matrix [ 1 0 0 0 0 1 1.46125e-15 0 0 -1.46099e-15 1 0 -3.57628e-05 \
                         1.89761e-06 130 1 ]
                children {
#                    cached puma ipt:puma|1400,interface {
                    cached panther ipt:panther|1400,interface {
                        name Panther
                        # CHANGE POINT: change the x value below to move the manipulator
                        #               side to side on bridge
                        matrix [ -0.999994 2.16463e-09 1.54238e-16 0 4.23855e-08 -0.999993 \
                                 6.98763e-22 0 5.03578e-14 -1.82624e-13 1 0 -4.8 -39.9999 \
                                 9.53675e-07 1 ]
                        program {
from VisionDemoManipProgram import VisionDemoManipProgram

program = VisionDemoManipProgram()
}
                        member effectorLink {
                            children {
                            }
                        }
                        member base {
                            # CHANGE POINT: change the z value below to move the base up and
                            #               down on the manipulator (crank it)
                            matrix [ 1 0 0 0 0 1 0 0 0 0 1 0 0 0 60 1 ]
                            children {
                            }
                        }
                    }
                }
            }
        }
    }
}
file outercornercurb.aaa {
    matrix [ -8.84336e-08 1 2.82723e-15 0 -1 -8.83553e-08 3.40823e-08 0 3.40823e-08 \
             -4.51566e-12 1 0 -297 -600 577.25 1 ]
}
file outercornercurb.aaa {
    matrix [ -1 1.5353e-09 4.59664e-17 0 5.53562e-10 -1 7.18114e-15 0 1.56768e-16   \
             -2.12542e-14 1 0 300 -596.5 577.25 1 ]
}
file outercornercurb.aaa {
    matrix [ 1 0 0 0 0 1 0 0 0 0 1 0 -300 596.5 577.25 1 ]
}
file outercornercurb.aaa {
    matrix [ 9.05718e-08 -1 -1.236e-15 0 1 9.57624e-08 3.41116e-08 0 -3.42113e-08   \
             3.87845e-09 1 0 297 600 577.25 1 ]
}
file shortcurb.aaa {
    matrix [ 1 0 0 0 0 1 0 0 0 0 1 0 0 -603.5 577.25 1 ]
}
file shortcurb.aaa {
    matrix [ -1 -2.92803e-07 -6.61493e-09 0 2.14959e-10 -0.999999 -1.14458e-15 0   \
             -1.86728e-08 -2.04577e-08 0.999999 0 -1.34161e-05 603.5 577.25 1 ]
}
```

C.2 Manipulator program

- VisionDemoManipProgram.py :

```
from DemoManipProgram import DemoManipProgram
import math


MAX_SEARCH_STEPS_X = 5
MAX_SEARCH_STEPS_Y = 2
SEARCH_STEPSIZE_X  = 2.07
SEARCH_STEPSIZE_Y  = 1.5
APPROACH_TOL_X=0.00003
APPROACH_TOL_Y=0.00003   #evtl zu klein

DEFAULT_TH = (1-(8.0/180.0))*math.pi
SAFE_Z = -5
```

```
CLOSE_Z=-75.0 #-75.50
TOUCH_Z=-75.90 #-75.70
z_marg=1.0

POS1_Y =  -434.115   # -0.033 #-0.275#in Courier position coordinate system
POS1_X = -130.885 # 0.335 #+0.0275 #in Courier position coordinate system

TARGET_Y =POS1_Y #-27.5 #in Courier position coordinate system
TARGET_X =POS1_X +3.0 #+27.5 #in Courier position coordinate system

TELL_COURIER_TO_COORDINATE_MSG="TELL_COURIER_TO_COORDINATE_MSG"
TELL_COURIER_TO_COORDINATE_FMT="{ float, float }"

TELL_COURIER_MOVE_TO_MSG="TELL_COURIER_MOVE_TO_MSG"
TELL_COURIER_MOVE_TO_FMT="{ float, float }"
COURIER_MOVED_TO_MSG="COURIER_MOVED_TO_MSG"
COURIER_MOVED_TO_FMT="int"

class VisionDemoManipProgram(DemoManipProgram):

    def __init__(self):
        DemoManipProgram.__init__(self)

        self.registerMessage(TELL_COURIER_TO_COORDINATE_MSG, \
                             TELL_COURIER_TO_COORDINATE_FMT)
        self.registerMessage(TELL_COURIER_MOVE_TO_MSG,TELL_COURIER_MOVE_TO_FMT)
        self.registerMessage(COURIER_MOVED_TO_MSG,COURIER_MOVED_TO_FMT)

    def bind(self):
        DemoManipProgram.bind(self)
        self.courier = self.bindAgent("Vole")

    def searchCurrentArea(self):
        res = None
        if self.simulated is not 1:
            res = self.get_closest_sphere_dist()

        if self.simulated is  1:
            res=[-0.000653, 0.000584]
            print "gave false values back"

        if res is None:
            return None
        else:
            return res


    def searchForSphere(self):
        found=0
        #initial_x=self.courier_intf.pos[0]
        #initial_y=self.courier_intf.pos[1]
        initial_x=POS1_X
        initial_y=POS1_Y

        manipdown=0

        target_x=initial_x
        target_y=initial_y
        for steps_y in range(MAX_SEARCH_STEPS_Y):
            for steps_x in range(MAX_SEARCH_STEPS_X-1):
                res=self.searchCurrentArea()
                if res is not None :
                    found =1
                    break
                print "range x",steps_x
                print "range y",steps_y
                target_x=initial_x-(steps_x+1)*SEARCH_STEPSIZE_X
                target_y=initial_y-(steps_y)*SEARCH_STEPSIZE_Y
                #self.sendTo(self.courierObject,TELL_COURIER_TO_COORDINATE_MSG,\
                             (target_x,target_y) )
                moved = self.queryTo(self.courierObject,TELL_COURIER_MOVE_TO_MSG,\
                         (target_x,target_y), COURIER_MOVED_TO_MSG)
                #status = self.acceptCoordination(self.partner, CLOSE_Z, DEFAULT_TH)
                #if not status:
                #    raise Exception(rendezvous_name, 'Failed to coordinate')
                print "Courier response: ", moved
                if moved is not 1:
                    raise Exception(rendezvous_name, 'Error: Courier failed to move.')
                if manipdown is 0:
                    self.singleStep(CLOSE_Z-z_marg, DEFAULT_TH, z_margin=z_marg)
                    manipdown=1
                self.sleep(2)
            if found is 1:
                break

        self.current_x=target_x
        self.current_y=target_y
        if found is 0:
            return None

        return res

    def approachSphere(self, sphereDist):

        #with appropriate DEFAULT_TH (make the job easier) and Courier
        #initialisation
        counter =0
        while abs(sphereDist[0]) >= APPROACH_TOL_X or abs(sphereDist[1]) >= \
                  APPROACH_TOL_Y:
            counter=counter+1

            target_x= self.courier_intf.pos[0] - 1000.0*sphereDist[0]
            target_y= self.courier_intf.pos[1] + 1000.0*sphereDist[1]
            print "current position", self.current_x, self.current_y
            print "approaching target: sending move to:", target_x, target_y

            moved = self.queryTo(self.courierObject,TELL_COURIER_MOVE_TO_MSG, \
                                 (target_x,target_y), COURIER_MOVED_TO_MSG)
            print "Courier response: ", moved
            if moved is not 1:
                raise Exception(rendezvous_name, 'Error: Courier failed to move.')

            sphereDist=self.searchCurrentArea()
            self.sleep(0.2)
            if self.simulated is 1:
                sphereDist=[APPROACH_TOL_X,APPROACH_TOL_Y]

            if sphereDist is None:
                print "Lost sphere!!!!"
                return None
            if counter >5:
                print "Could not get near enough to the sphere"
                return 1

        return 1

    def run(self):
        self.initializeExecutor()
        print "program started"

        self.bind()
        self.courier_intf = self.courier.getInterface()
        self.courierObject=self.getProgramObject(self.courier_intf)
        self.setDIO(0)

        self.singleStep(SAFE_Z-z_marg-10, 0, z_margin=z_marg)


        self.sleep(1)
        self.singleStep(SAFE_Z-z_marg-10, DEFAULT_TH, z_margin=z_marg)
        self.sleep(1)

        rendezvous_name = "test"
        self.partner = self.acceptRendezvous(rendezvous_name)

        sphereDist=self.searchForSphere()
        if sphereDist is None:
            print "No Sphere found in search."
            return

        if self.approachSphere(sphereDist) is 1:

            #move down
            self.singleStep(TOUCH_Z-z_marg, DEFAULT_TH, z_margin=z_marg)
            #pick up
            self.setDIO(4)  #adapt this number
            self.singleStep(TOUCH_Z-z_marg, DEFAULT_TH, z_margin=z_marg)
            self.sleep(0.2)
            self.setDIO(5)  #adapt this number
            self.singleStep(TOUCH_Z-z_marg, DEFAULT_TH, z_margin=z_marg)
            self.sleep(0.1)
            self.setDIO(6)  #adapt this number
            self.singleStep(TOUCH_Z-z_marg, DEFAULT_TH, z_margin=z_marg)
            self.sleep(0.2)
            self.setDIO(7)  #adapt this number
            self.singleStep(TOUCH_Z-z_marg, DEFAULT_TH, z_margin=z_marg)
            #move up
            self.singleStep(CLOSE_Z-z_marg+2.0, DEFAULT_TH, z_margin=z_marg)
            #move courier to target
            self.sleep(4.0)
            moved = self.queryTo(self.courierObject,TELL_COURIER_MOVE_TO_MSG,\
                                 (TARGET_X,TARGET_Y), COURIER_MOVED_TO_MSG)
            print "Courier response: ", moved
            if moved is not 1:
                raise Exception(rendezvous_name, 'Error: Courier failed to move.')
            #move down
            self.sleep(4.0)
            self.singleStep(CLOSE_Z-z_marg, DEFAULT_TH, z_margin=z_marg)
            #place
            self.singleStep(TOUCH_Z-z_marg, DEFAULT_TH, z_margin=z_marg)
            #self.sleep(0.2)
            #self.truncate()
            self.setDIO(4)
            self.singleStep(TOUCH_Z-z_marg+0.3, DEFAULT_TH, z_margin=z_marg)
            self.sleep(0.2)
            #Admire the work for the cameras ;-)
            self.setDIO(0)
            self.singleStep(CLOSE_Z-z_marg, DEFAULT_TH, z_margin=z_marg)
            self.sleep(4)
            #control placement (?)
            #wiggle(?)

        #move up
        self.singleStep(SAFE_Z-z_marg, DEFAULT_TH, z_margin=z_marg)
        #terminate RDVZ
        self.finishRendezvous()


        while 1:
            self.sleep(1.0)
```

C.3 Courier program

- VisionDemoCourierProgram.py :

```
from DemoCourierProgram import DemoCourierProgram
from FoDescription import FoDescriptionPtr
from FbMessage import FbMessagePtr

TELL_COURIER_TO_COORDINATE_MSG="TELL_COURIER_TO_COORDINATE_MSG"
TELL_COURIER_TO_COORDINATE_FMT="{ float, float }"

TELL_COURIER_MOVE_TO_MSG="TELL_COURIER_MOVE_TO_MSG"
TELL_COURIER_MOVE_TO_FMT="{ float, float }"
COURIER_MOVED_TO_MSG="COURIER_MOVED_TO_MSG"
COURIER_MOVED_TO_FMT="int"


class VisionDemoCourierProgram(DemoCourierProgram):

    def __init__(self):
        DemoCourierProgram.__init__(self)

        self.registerMessage(TELL_COURIER_TO_COORDINATE_MSG,TELL_COURIER_TO_COORDINATE_FMT)
        self.addMessageHandler(TELL_COURIER_TO_COORDINATE_MSG, self.tell_courier_to_coord_handler)

        self.registerMessage(TELL_COURIER_MOVE_TO_MSG,TELL_COURIER_MOVE_TO_FMT)
        self.registerMessage(COURIER_MOVED_TO_MSG,COURIER_MOVED_TO_FMT)
        self.addMessageHandler(TELL_COURIER_MOVE_TO_MSG, self.tell_courier_move_to_handler)

    def bind(self):
        DemoCourierProgram.bind(self)
        self.manip = self.bindAgent("Panther")



    def setHome(self, xdir, ydir, xoffset=0, yoffset=0):
        """Set the courier's position to the left/right (-/+), top/bottom (+/-)
        corner of the platen, offset by xoffset and yoffset millimeters.  This
        method will do nothing when not running on a courier head"""
        #Implemented by Jay Gowdy and Christoph Bergler

        # get the point we will be "homing" to.  Presumably the courier
        # is jammed into this corner
        pright, ptop, pleft, pbot, pz = self.platen.getCorners()
        if xdir < 0:
            px = pleft
        else:
            px = pright
        if ydir < 0:
            py = pbot
        else:
            py = ptop
        px = px + xoffset;
        py = py + yoffset;

        # now figure out how to move the courier's reported position to match
        # with us being in this corner
        cur_pos = self.interface.pos
        currentx = cur_pos[0]
        currenty = cur_pos[1]
        home = FoDescriptionPtr(self.description.home)
        plattrans = self.platen.getPlaten().getMatrix(home).getTranslation()
        xoff = (px+plattrans[0])-(currentx-self.description.footprint[2]*xdir)
        print "X position is %f, will move X offset by %f" % (currentx, xoff)
        yoff = (plattrans[1]+py)-(currenty-self.description.footprint[3]*ydir)
        cornery = currenty-self.description.footprint[3]*ydir
        print "Y position is %f, will move X offset by %f" % (currenty, yoff)

        # and set the interface.transform field, which is a field only of real
        # couriers that transforms the rawPos field into the pos field
        t = self.interface.transform
        t[12] = t[12] + xoff
        t[13] = t[13] + yoff
        self.interface.transform = t
        self.interface.homed = 1

    def tell_courier_to_coord_handler(self, container, msg):
        print "Recieved order to initiate coordinated movement."
        res=self.getData(msg)
        print "recieved: ", res[0], res[1]
        self.coordinateTo( self.manip_object, res[0], res[1])
        print "Finishing Coordination"


    def tell_courier_move_to_handler(self, container, msg):
        print "Recieved order to initiate movement."
        res=self.getData(msg)
        print "Target recieved: ", res[0], res[1]
        self.singleStep((res[0], res[1],0))
        print "Finishing move"
        self.replyTo(FbMessagePtr(msg), COURIER_MOVED_TO_MSG, 1)


    def run(self):

        print "program started"

        self.bind()
        self.setHome(1, -1, -27.5, 27.5)

        manip_intf = self.manip.getInterface()
        self.manip_object = self.getProgramObject(manip_intf)

        place_rendezvous=  "test"# to be determined
        print "Initiating"
        self.initiateRendezvous(self.manip, place_rendezvous)

        while 1:
```

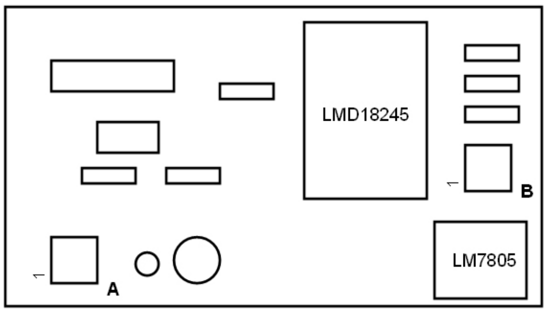
```
    self.processEvents()
    #possibly include event to terminate RDVZ


self.finishRendezvous(place_rendezvous)

self.singleStep((28, 28),max_v=0.05, max_acc=0.05)
while 1:
    self.sleep(0.2)
```

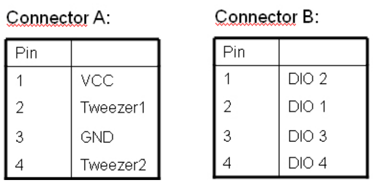
D Tweezer gripper

D.1 Voice coil data

Voice coil data sheets can be found on the included DVD at:
`/Hardware/Gripper/voicecoil/`

D.2 Tweezer control card




Connector A:

Pin	
1	VCC
2	Tweezer1
3	GND
4	Tweezer2

Connector B:

Pin	
1	DIO 2
2	DIO 1
3	DIO 3
4	DIO 4

Figure 27: Schematic tweezer card pin mapping

The LMD18425 datasheet can be found on the included DVD at:
`/Hardware/Gripper/card/LMD18245.pdf`

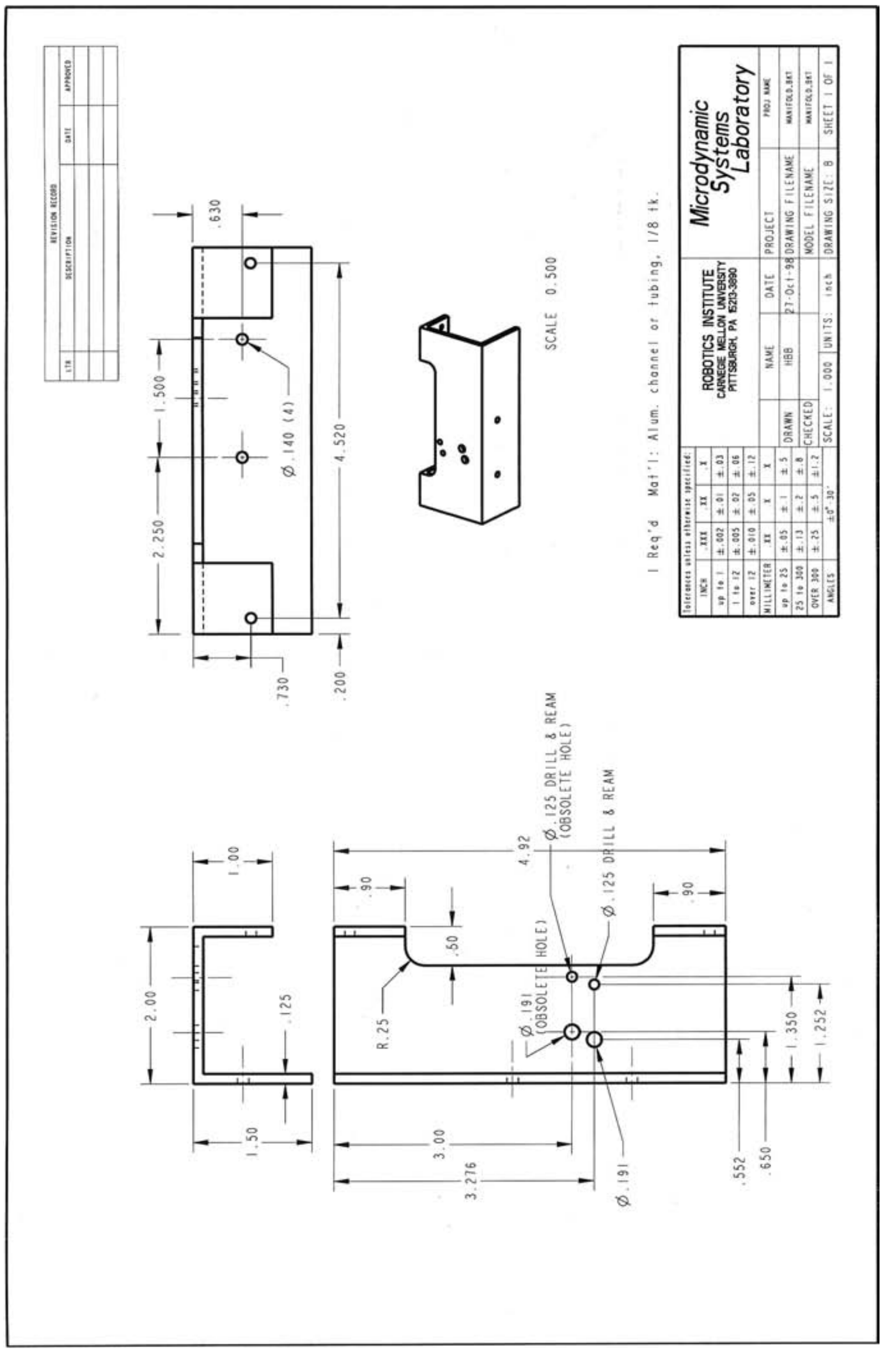
D.3 Changed manipulator code

These files were changed in order to be able to actuate the tweezer gripper. The files can be found on the attached DVD.

```
agent/new_vision/Demo/DemoManipProgram.py
agent/src/AAA/tool/FoSimManipulatorInterface.cc
agent/src/AAA/tool/FoManipMoveToController.cc
agent/src/AAA/tool/Types.py
agent/src/AAA/tool/FoManipMoveToController.h
agent/src/AAA/manip/ManipActions.cc

agent/src/AAA/manip/ExecutorInterface.h
agent/src/AAA/manip/maniphead.cc
agent/src/AAA/manip/ExecutorInterface.cc
agent/src/AAA/manip/ManipControllers.h
agent/src/AAA/manip/FbManipulatorHead.h
agent/src/AAA/include/AAA/interfacing/FoManipulatorInterface.h
agent/src/AAA/agent/FoManipulatorInterface.cc

agent/mini/agent/ohm-exec/actuator/actuator.c

agent/mini/agent/ohm-exec/interface/cbq_srv.
agent/mini/agent/ohm-exec/interface/log_srv.c
agent/mini/agent/ohm-exec/interface/cbq_if.h
agent/mini/agent/ohm-exec/interface/log_if.h
agent/mini/agent/ohm-exec/control/control_if.h
agent/mini/agent/ohm-exec/control/control.c
```

E Manipulator documentation

E.1 Modified valve manifold support



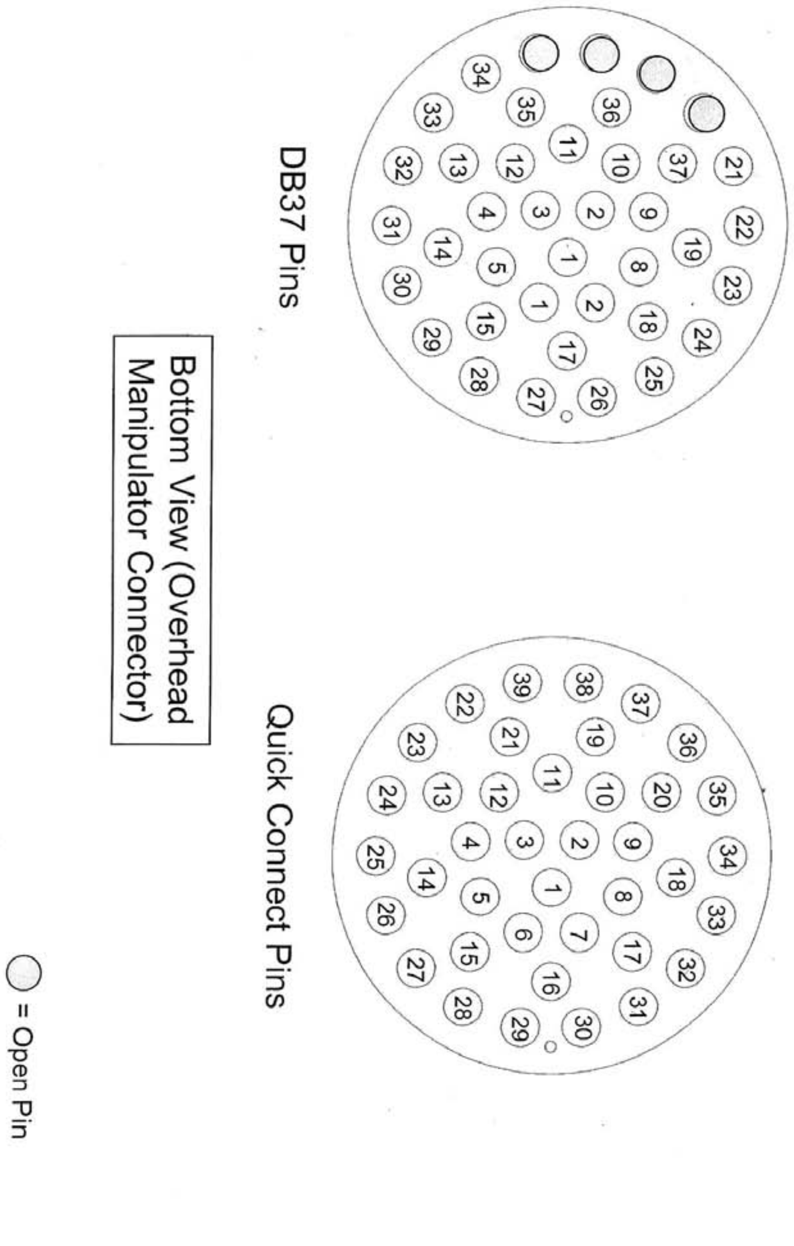
REVISION RECORD
LTR
DESCRIPTION
DATE
APPROVED
.630
1.500
2.250
Ø.140 (4)
4.520
.730
.200
SCALE 0.500
1 Req'd Mat'l: Alum. channel or tubing, 1/8 tk.
Microdynamic Systems Laboratory
ROBOTICS INSTITUTE
CARNEGIE MELLON UNIVERSITY
PITTSBURGH, PA 15213-3890
DRAWN HBB 27-Oct-98
CHECKED
DRAWING FILENAME MANIFOLD.DKT
MODEL FILENAME MANIFOLD.DKT
SCALE: 1.000 UNITS: inch DRAWING SIZE: B SHEET 1 OF 1
1.00
2.00
.125
1.50
4.92
.90
.50
.90
R.25
Ø.125 DRILL & REAM (OBSOLETE HOLE)
Ø.125 DRILL & REAM
Ø.191 (OBSOLETE HOLE)
Ø.191
3.00
3.276
.552
.650
1.350
1.252

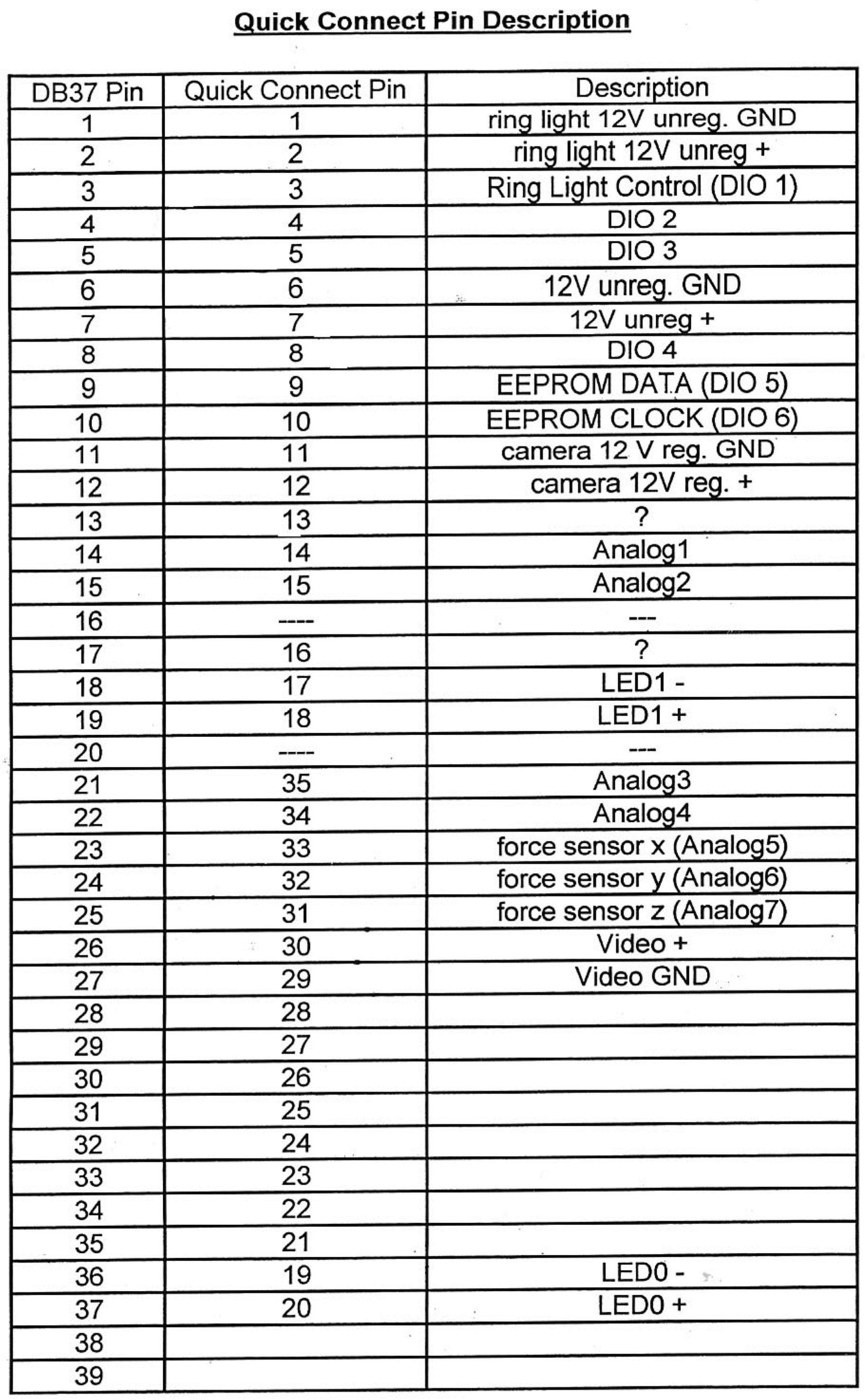
Tolerances unless otherwise specified:			
INCH	.XXX	.XX	.X
up to 1	±.002	±.01	±.03
1 to 12	±.005	±.02	±.06
over 12	±.010	±.05	±.12
MILLIMETER	.XX	.X	X
up to 25	±.05	±.1	±.5
25 to 300	±.13	±.2	±.8
OVER 300	±.25	±.5	±1.2
ANGLES	±0° 30'		

E.2 Pin description tables

Table 8: OHM digital I/O IP408 bit/Pin mapping.

Signal	**Bit number**	**Pin**	**Pin end effector**
Z limit top	0	1	
Z limit bottom	1	2	
beacon 0	2	3	
beacon 1	3	4	
DIO1 (Light)	4	6	3
DIO2	5	7	4
axis 0/Z amp AOK	8	11	
axis 1/th amp AOK	12	16	
axis 0/Z amp enable	16	21	
axis 1/th amp enable	17	22	
DIO3	18	23	5
DIO4	19	24	8
DIO5 (EEPROM DATA)	20	26	9
DIO6 (EEPROM CLOCK)	21	27	10
gripper vacuum 0	24	31	
gripper pressure 0	25	32	
gripper vacuum 1	26	33	
gripper pressure 1	27	34	
gripper vacuum 2	28	36	
gripper pressure 2	29	37	
grav valve	30	38	
brake valve	31	39	


DB37 Pins
Quick Connect Pins
Bottom View (Overhead Manipulator Connector)
= Open Pin



Quick Connect Pin Description

DB37 Pin	Quick Connect Pin	Description
1	1	ring light 12V unreg. GND
2	2	ring light 12V unreg +
3	3	Ring Light Control (DIO 1)
4	4	DIO 2
5	5	DIO 3
6	6	12V unreg. GND
7	7	12V unreg +
8	8	DIO 4
9	9	EEPROM DATA (DIO 5)
10	10	EEPROM CLOCK (DIO 6)
11	11	camera 12 V reg. GND
12	12	camera 12V reg. +
13	13	?
14	14	Analog1
15	15	Analog2
16	----	---
17	16	?
18	17	LED1 -
19	18	LED1 +
20	----	---
21	35	Analog3
22	34	Analog4
23	33	force sensor x (Analog5)
24	32	force sensor y (Analog6)
25	31	force sensor z (Analog7)
26	30	Video +
27	29	Video GND
28	28	
29	27	
30	26	
31	25	
32	24	
33	23	
34	22	
35	21	
36	19	LED0 -
37	20	LED0 +
38		
39		