Visually Guided Coordination for Distributed Precision Assembly

Michael Ling Chen

CMU-RI-TR-99-45

Submitted to the Department of Electrical and Computer Engineering
In Partial Fulfillment of the Requirements for the Degree of
Master of Science in Electrical Engineering

The Robotics Institute
Carnegie Mellon University
Pittsburgh, PA

December, 1999

This work was supported in part by NSF grant DMI-9527190.

Abstract

This report documents the integration of machine vision into the minifactory environment. Minifactory is an automated factory system being developed at the Microdynamic Systems Laboratory at Carnegie Mellon University and is aimed toward the assembly of small mechatronic devices. Machine vision can aid assembly systems by eliminating the need for complicated part holding jigs, end effectors, and part feeding equipment. In addition, the position accuracy of a visually servoed robot can be made relatively insensitive to robot calibration errors. By using the visual control and communications systems described in this report, two high-precision 2-DOF robotic agents were visually guided to precisely align parts of a sub-assembly. Test results show that open-loop look-and-move alignment of parts took an average of 3.7 iterations and 3.6 seconds while visual servoing alignment took an average of 1.3 seconds. The standard deviation of the visual servoing alignment errors for the u and v axes were 4.6 μm and 5 μm respectively. However, they were limited by tracker noise which was approximately 10 μm peak to peak with a standard deviation of 2 μm in the u axis of the image plane and 20 μm peak to peak with a standard deviation of 6 μm in the v axis. Overall, the vision project's greatest success was making two 2-DOF agents transiently cooperate to function as a single 4-DOF robot to perform visual servoing tasks.

Thesis supervisor:	Dr. Ralph Hollis Principal Research Scientist, Robotics Institute
First reader:	Dr. Robert M. White Professor of Electrical and Computer Engineering and Engineering and Public Policy Director of the Data Storage Systems Center
Second reader:	Dr. Gary K. Fedder Associate Professor of Electrical and Computer Engineering and the Robotics Institute

Acknowledgments

I would like to thank all the people in the Microdynamic Systems Laboratory who have provided me with their support. I'd first like to thank Dr. Ralph Hollis for giving me the opportunity to contribute to the minifactory project and for providing advice, insight, and guidance along the way. I'd also like to thank Dr. Alfred Rizzi for sharing his technical expertise, encouragement, and endless patience. I also offer many thanks to Jay Gowdy, Patrick Muir, Ben Brown, Zack Butler, Arthur Quaid, Rich Deluca, and everyone else who has participated in the project for their friendship, advice, and support.

I would also like to acknowledge Knowles Electronics and Dave Swanson for providing hearing aid parts, Dr. Gregory D. Hager for his assistance with X Vision, and the National Science Foundation for their support under grant DMI-9527190.

Finally, I'd like to thank my family and all my other friends for always being there when I needed them.

Contents

1 Introduction **1**

1.1 Motivation for the Agile Assembly Architecture and Minifactory 1

1.2 Minifactory Components 2

1.3 Motivation for Computer Vision in the Minifactory 3

1.4 Background Information On Visual Control 5

1.4.1 Open-Loop Look-and-Move 5

1.4.2 Visual Servoing Architectures 5

2 Objectives **6**

2.1 Hardware and Software Goals 6

2.2 Vision Integration and Tests 7

3 Vision Related Hardware **8**

3.1 Frame Grabber Hardware 8

3.2 Camera and Lens System 10

4 Vision Related Software **10**

4.1 Frame Grabber Software 10

4.2 Camera Calibration 12

4.2.1 Tsai's Camera Calibration Technique 13

4.2.2 Camera Calibration and the Minifactory 15

4.3 Vision Library 17

4.3.1 Look-and-Move Related Functions 17

4.3.2 Electret Analysis 18

4.3.3 Diaphragm Analysis 19

4.3.4 Trackers 22

4.4 Look-and-Move Communications Infrastructure 23

4.5 Visual Servoing Communications Infrastructure 26

4.6 Visual Servoing Control 27

5 Experimental Tests and Results **30**

5.1 Open-Loop Look-and-Move 30
5.2 X-Y Visual Servoing Test 32
5.3 Corner Tracker Noise Measurements 35
5.4 X-Y-Theta Visual Servoing Test 36

6 Summary and Conclusions 39

7 Future Work 40

References 41

List of Figures

1 The major components of a minifactory system. . . . 1

2 A courier agent. . . . 2

3 An overhead manipulator agent. . . . 3

4 A minifactory development and monitoring tool 4

5 Parts of a hearing aid device 7

6 The location of the camera and lens parts in the end effector. . . . 10

7 Illustration of how a user application accesses the device driver. . . . 11

8 Interrupt service routine flowchart. . . . 12

9 Graphical User Interface to test some of the frame grabber's functions. . . . 13

10 Perspective Projection using a pin hole camera model. . . . 14

11 Relationship between a point on the image plane and its corresponding pixel location. . . . 14

12 Various measurements needed for camera calibration. . . . 15

13 Coordinate transforms in the minifactory. . . . 15

14 Micrometer scale target used in camera calibration 16

15 The vision library layout. . . . 17

16 An electret being held by a vacuum pickup device which causes an occlusion. . . . 19

17 Magnified view of the left and right corner templates used with the SSD tracker. . . . 19

18 An SSD tracker converging to an electret corner 20

19 Subtracting the electret image from the diaphragm image 21

20 The diaphragm edge finding method 22

21 Mapping of image plane points into parameter space. . . . 22

22 Elimination of outliers from each side of the diaphragm 23

23 Robustness of the new diaphragm finding algorithm 24

24 The communications infrastructure used for open-loop look-and-move. . . . 24

25 Passing solid angle information between processes 25

26 The communications infrastructure used for visual servoing. . . . 26

27 A block diagram of the visual servo controller. . . . 30

28 Open-loop look-and-move times 31

29 Open-loop look-and-move errors 32

30 Open-loop look-and-move qualitative performance 32
31 x-y visual servoing model. 33
32 x-y visual servoing results 33
33 Discrete model of the x-y visual servo control system. 34
34 Root locus of the x-y visual servo control system 35
35 The tracker positions for both non-moving corners. 36
36 Data collected for one x-y-θ visual servoing alignment run 37
37 Histograms of errors from the x-y-θ visual servoing experiment 38
38 Alignment before and after the x-y-θ visual servoing. 38

List of Tables

1 Imagenation PX610 specifications 9
2 Steady state error for the x-y visual servoing experiment 34
3 Standard deviation and peak-to-peak error of the corner trackers 36
4 Error measurements from the x-y-θ visual servoing experiment 38

1 Introduction

1.1 Motivation for the Agile Assembly Architecture and Minifactory

Advancements in technology have recently caused a massive growth of new electro-mechanical products. For example, every few months companies that manufacture hard disk drives develop new models with larger capacities and faster access times. However, as the product life cycle decreases, the cost of setting up an assembly line becomes a more significant portion of the product cost [1]. While many production facilities today use a complex assortment of robots, conveyors, and computers to assemble products, they are often difficult to configure, modify, and update. Robots will often require special cabling or calibration making them expensive to integrate. Many production facilities also face problems when handling small and fragile parts which often means they must still rely on manual labor. Even if their robotic systems are able to handle delicate parts, the precision alignment of small parts often remains a difficult problem.

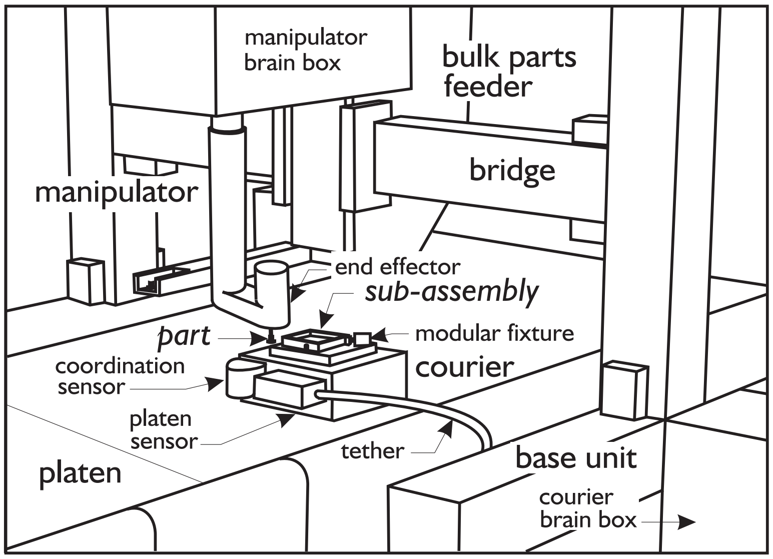


Figure 1: The major components of a minifactory system.

A promising solution to the complex problem of precision automated assembly for small mechatronic devices involves creating a flexible, modular, and reconfigurable system. At the Microdynamic Systems Laboratory at Carnegie Mellon University, we are developing an Architecture for Agile Assembly (AAA) which fulfills these roles, and it is being implemented in the minifactory project shown in Fig. 1. By making the minifactory physically, computationally, and algorithmically modular, it can be designed and deployed in a fraction of the time needed by traditional factory systems [2]. By giving the robots high-quality sensing capabilities, production lines which were previously hard to automate due to the small part sizes and tolerances, can now be automated. In addition to being more efficient, reducing the number of people working in an assembly area reduces contamination which becomes more important with smaller devices [3].

1.2 Minifactory Components

Minifactory is composed of robotic entities known as agents which are physically, computationally, and algorithmically modular. Agents are two-degree-of-freedom (DOF) robots which can cooperatively work together to perform 4-DOF tasks. Currently, each agent has a motherboard with a PowerPC 604r 300 MHz microprocessor and Industry Pack modules which provide mechanisms for analog-to-digital conversion, digital-to-analog conversion, and digital input and output. In addition, agents may have other specialized components such as sensors or frame grabbers.

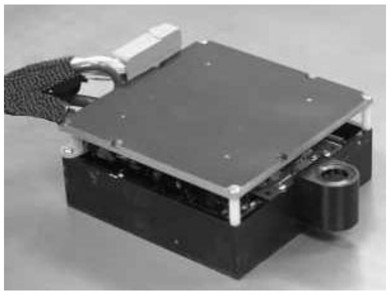


Figure 2: A courier agent.

For the experiments conducted in this paper, two types of agents were used: the courier agent and the overhead manipulator agent. The courier, as shown in Fig. 2, is based on a planar linear motor which has 3 DOF: x, y translational and θ rotational movement. It moves on an air bearing over specially designed metal sheets, known as platens, which comprise the factory floor. A specially designed 3-DOF magnetic sensor has previously been designed to provide position and angle measurements, with a 1σ motion resolution of 200 nm [4]. The sensor allows the courier to be controlled in closed-loop mode [5]. The courier also contains an optical sensor which is used for calibration when the courier explores its workspace [6], [7]. This sensor detects beacon LEDs located on overhead manipulators agents hanging on bridges above the couriers.

The second type of agent used in the experiments described in this paper is the overhead manipulator designed by Ben Brown, as shown in Fig. 3. The manipulator has 2 DOF: z translational and θ rotational movement . The z range of motion is approximately 125 mm with a resolution of 2 μm, and the θ range of motion is $\pm 270°$ with a resolution of 0.0005°. The manipulator also contains a frame grabber, as described in Sec. 3.1, for image processing. A modular quick connection mechanism has also been developed by Patrick Muir which allows different types of end effectors to quickly attach to the manipulator.

In addition to the modular agents described above, the minifactory also contains a simulation and monitoring software program, the *interface tool*, which greatly speeds the development and deployment of a minifactory [8]. An example simulation screen is shown in Fig. 4. The same program is used to set up and simulate a factory as well as monitor the real instantiation of the factory. However, the real factory does not need to exactly match the simulation since the factory self-calibrates during its initialization [9]. The calibration information allows the behavior of the real factory to match the behavior of the simulation even though the agents may not be placed at exactly the same positions. This aids in the quick design and

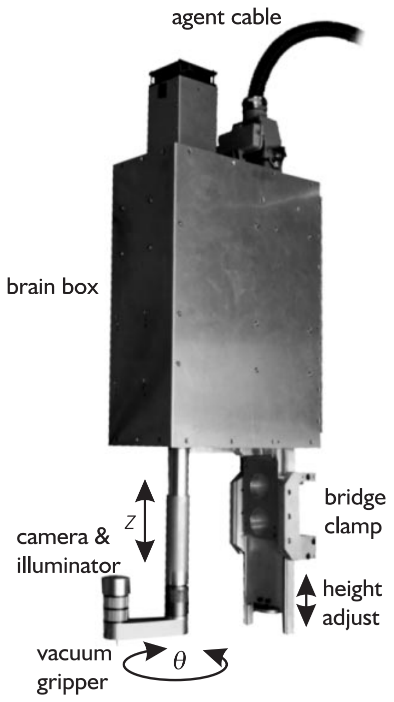


Figure 3: An overhead manipulator agent.

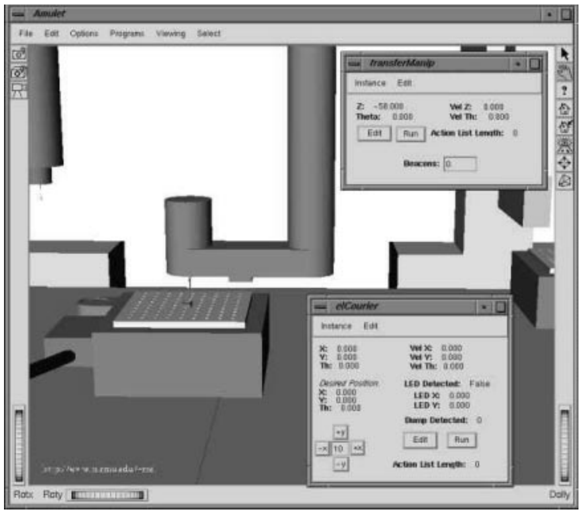
set up of the minifactory.

Another essential component of the minifactory is the communication infrastructure between agents. There are two network structures located in the minifactory. A global network allows factory wide communication, such as between the monitoring program and the agents, while the local network provides high speed communications between local agents working cooperatively to perform a task. The local network's protocols can provide guaranteed and non-guaranteed data transmission to facilitate a wide variety of interactions between local agents [10]. Section 4.5 describes how this network was used in the cooperative task performed by a courier and manipulator.

An agent has computer processes running at two levels. The higher-level processes are currently composed of Python scripts which are used to manage high level factory decisions, such as resource negotiation and product flow decisions [11]. Python is a portable, object-oriented scripting language with the ability to be extended by C and C++ code [12]. Lower-level factory code is written in C and C++ and is used to execute real-time control laws, such as the visual servo controller.

1.3 Motivation for Computer Vision in the Minifactory

Robots can be made fast and precise, but they are still at a disadvantage without computer vision in certain situations compared with people performing the same task. A common problem encountered in systems without computer vision is the need to precisely know the location of the robot's end effector in a robotic assembly system. While this can be determined through kinematics, these equations are usually based on



Z: -58.000 Vel Z: 0.000
Theta: 0.000 Vel Th: 0.000
Action List Length: 0
Beacons: 0

X: 0.000 Vel X: 0.000
Y: 0.000 Vel Y: 0.000
Th: 0.000 Vel Th: 0.000
Desired Position
X: 0.000
Y: 0.000
Th: 0.000
LED Detected: False
LED X: 0.000
LED Y: 0.000
Bump Detected: 0
10
Action List Length: 0

Figure 4: A minifactory development and monitoring tool [8].

robot models which may not match the real robot very well [13]. One solution to this problem includes using tools such as teaching pendants, but teaching a robot where to move can be an expensive and time consuming operation.

Another problem robotic assembly systems without vision must contend with is the imprecise presentation of parts. Without vision, the system must assume that the parts will always appear at the same position within a certain tolerance. This often requires the creation of complicated part holding jigs, end effectors, and part feeding equipment. These solutions can be expensive and time consuming and are not considered agile solutions in the minifactory system, where the goal is to set up a factory in the span of a few weeks.

Once computer vision is incorporated into a robotic assembly system, the above problems become more manageable in terms of the time required to set up a system. Parts no longer need to be precisely presented to the robot since the vision system can determine where the robot needs to move in order to correctly pick or place a part. For example, parts feeding equipment can present rotated parts while the vision system determines the position and rotation before appropriately moving to grasp the part. In addition, special jig and end effectors would not need to be developed, decreasing cost and setup time. Another gain from computer vision is the ability to use it as a sensor in closed-loop mode. The positioning accuracy of a visually servoed robot can be made relatively insensitive to calibration errors of the robots [14].

While there are many benefits to using computer vision, there are also additional costs. Factors such as special image processing hardware and software must be taken into account. Additional hardware components include frame grabbers, cameras, lenses, and special lighting components. Until recent years, using a standard desktop PC to perform all the software chores necessary for a vision system was not very practical. Proprietary hardware and software vision systems could be purchased from companies such as Adept and Cognex, but these were generally expensive. However, with the incredible growth in computing power during the last few years, desktop computer speeds have reached a point where they can perform the same vision tasks previously only possible on proprietary vision systems.

Although there are additional costs related to vision hardware and software, the competitive nature of the vision industry helps to continually improve the technologies while lowering the costs. The recent growth in the computer vision industry can also be attributed to a trend toward the adoption of common hardware and software for vision processing [15]. Vision integrators and end users have been pushing for vision systems which can run on desktop PC's and provide a Windows-like graphical user interface (GUI). This has lead to the development of many new vision software packages, such as the Matrox Imaging Library, as well as PC frame grabbers with various levels of sophistication. However, real-time vision tasks are difficult to implement with the standard frame grabber used with a Windows operating system due to the lack of determinism. Instead of switching to a real-time operating system, many choose to buy a frame grabber with an on-board processor to perform much of the vision computation. As discussed later in this report, the minifactory system has a real-time operating system running at the core of every agent. The challenge therefore becomes finding a frame grabber and software vision library supported by the operating system.

1.4 Background Information On Visual Control

1.4.1 Open-Loop Look-and-Move

Before discussing various visual servo architectures, it will be helpful to first distinguish between open-loop look-and-move procedures and visual servoing. Open-loop look-and-move is an iterative procedure first involving image acquisition and analysis. Next, the robotic system is adjusted to correct for the error information provided by the image analysis. Once the robot has moved and settled, another image is taken and analyzed. This procedure iterates until the error is acceptable. When compared to visual servoing control, open-loop look-and-move accuracy is more dependent on calibration and accuracy of the robot movement [13].

Within the minifactory environment, open-loop look-and-move may be fine for placing one part into another while there is not much disturbance in the system. However, at higher magnifactions, the movement of other couriers or manipulators produce vibrations in the system which cannot be accounted for by the look-and-move method. In these cases, visual servoing will be more effective to compensate for the noise in real-time.

1.4.2 Visual Servoing Architectures

In contrast to open-loop look-and-move techniques, visual servoing uses vision as a sensor in a feedback loop and suffers less from calibration errors and mechanical inaccuracies [13]. However, feedback at field/frame rate may require significant computing power and system stability also becomes a factor.

As described in [14], a visual servo system can be classified based on two major features:

1. Does the system use dynamic look-and-move control or direct visual servo control?

2. Is the feedback error of the system considered to be position-based or image-based?

Dynamic look-and-move control (not to be confused with open-loop look-and-move defined in Sec. 1.4.1) is defined as a controller which provides set point inputs into a joint level controller. For instance, the visual servo controller described in this report sends a velocity command into the manipulator's velocity controller which then actuates the manipulator. In contrast, the output of a direct visual servo controller would be used to actuate the manipulator without using its velocity controller. This would be a challenging task since the vision feedback rate (60 Hz or less) is much slower than the internal feedback rate of the velocity controller (2 kHz). Robots generally have an interface for velocity or position input commands, making dynamic look-and-move a more attractive control scheme [14].

Position-based visual servoing uses features from the image to estimate the pose of the target with respect to the camera. This allows tasks to be written in terms of Cartesian pose, making it somewhat more straightforward than image-based methods. However, the feedback control is based on values which are a function of calibration parameters, making it more sensitive to calibration errors [14]. Another disadvantage of pose-based methods is the computation needed for image interpretation.

Image-based (also known as feature-based) visual servo control uses an error signal computed directly from features in the image plane. To relate image plane corrections with manipulator and courier adjustments, the image Jacobian is used. As described in Sec. 4.6, the image Jacobian relates motion of features in the image plane to motion of the courier and manipulator. The main advantage of image-based visual servoing is that the system is less sensitive to errors in calibration. Comparisons performed by Hashimoto et al. [16] have shown that image-based servoing is also more robust in the presence of quantization error and noise. However, one disadvantage is that the control law may have more unstable points than the equivalent position-based controller [14].

In the remainder of this report, visual servoing will refer to an image-based, dynamic look-and-move control system.

2 Objectives

2.1 Hardware and Software Goals

Since the vision software development depended on having the hardware infrastructure in place, the first objective of this project was to obtain and configure the hardware needed to capture images using the overhead manipulator described in Sec. 1.2. A precision frame grabber which could plug into a PCI slot on the PowerPC motherboard was required. As described in Sec. 3.1, specifications included having pixel jitter less than ± 5 ns, ability to perform direct memory access (DMA) data transfers, and gray-scale noise less than 0.7 times the least significant bit (using an 8-bit digitizer).

The next hardware component required was the camera and lens system, which needed the ability to properly view small components up to 3.7×3.7 mm in size. The parts being analyzed belonged to a hearing aid device supplied by Knowles Electronics and are shown in Fig. 5.

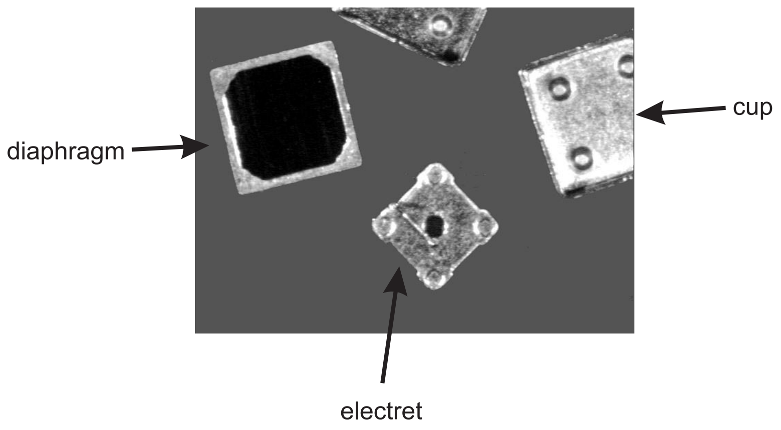


Figure 5: Parts of a hearing aid device with a maximum size of 3.7×3.7 mm.

The first software goal of this vision project was to develop a device driver for the frame grabber. At the time the board was chosen, there were no frame grabbers with LynxOS drivers. The second software goal was to investigate camera and lens calibration methods so that points on the image plane could be related to real world coordinates. The next objective was to develop a vision library in C++ which was accessible to the Python code providing upper-level management of the minifactory. Specifically, the code should be able to identify the center of the diaphragm and electret parts to subpixel accuracy and up to a $20°$ rotation. In addition, the library should include functions which would allow tracking of the diaphragm at field rate (60 Hz) for the visual servoing system. Another software objective would involve developing the minifactory infrastructure to support the transfer of vision data at 60 Hz between various processes and agents. Finally, the controller software used for visual servoing would require development.

2.2 Vision Integration and Tests

The remaining objectives of the project included using the frame grabber, camera and lens setup, and vision library to perform visually guided coordination between a courier and manipulator to precisely align an electret with a diaphragm. First, vision would be used to align an electret 2.5×2.5 mm in size with a diaphragm 3.3×3.3 mm in size using the open-loop look-and-move method discussed in Sec. 1.4.1. The next goal would be to use vision in a feedback loop and to visually servo the diaphragm with the electret's position and orientation. Tests would include first servoing only the x and y position of a diaphragm and then servoing its x, y, and θ position with an electret. Finally, comparisons would be made between the open-loop look-and-move alignment and the visual servoing alignment.

3 Vision Related Hardware

3.1 Frame Grabber Hardware

There are many varieties of frame grabbers available, some which are more suitable for precision image acquisition. Listed below are a few of the criteria used to pick a frame grabber and their importance to this project.

1. Color vs. gray-scale: Gray-scale cameras have higher resolutions, better signal-to-noise ratios, increased light sensitivity, and greater contrast than color cameras in the same price range [17]. In minifactory applications, gray-scale images will generally contain all the image features needed (edges, shapes, textures) in a more compact form than color.

2. Pixel jitter: A frame grabber samples an analog waveform produced by a camera at uniform intervals in time to recover each pixel value. If the timing accuracy of the frame grabber's pixel clock is not accurate with respect to the horizontal sync pulse, pixels for a particular line of the image may be digitized a little early or late. For instance, pixels are sampled about every 80 ns in an image with 640 pixels per line. A pixel jitter specification of 5 ns means a 12.5% error in pixel position. Good frame grabbers generally have $< \pm 5$ ns pixel jitter [18].

3. Gray-scale noise: When constantly sampling the same analog signal representing an image, frame grabber circuitry causes a pixel to have a slightly different digitized value from image to image, even though the value should actually remain constant. The gray-scale noise level is measured as the standard deviation of these different values. If a specification says the noise level is .7 LSB, that means the standard deviation for the noise is .7 times the least significant bit. For example, an 8-bit binary number provides 256 gray-scale values, and .7 LSB means the noise level varies over $\frac{.7}{256}$ of the entire gray-scale range. Low gray-scale noise is important for images with low contrast. It is also needed when searching an image for surface irregularities or edges. A precision frame grabber generally has a noise level of .7 LSB or less [18].

4. Progressive scan support: Progressive scan is a non-standard video format in which all the pixels in a picture are exposed simultaneously, instead of one field at a time. In the past, most video cameras were interlaced, meaning that first the elements on the even lines of the image were exposed and sent out followed by the elements on the odd lines. This is normally acceptable unless an object is moving quickly relative to the video camera since it would cause the object to be shifted in one field relative to the other. This can cause problems when performing image processing because the picture would be blurred. One solution is to use a progressive scan camera. Another solution is to simply process one field at a time at 60 Hz, as long as it is acceptable to use only half the y resolution (one field) during each processing cycle.

5. Onboard memory: Onboard memory refers to extra memory placed on the frame grabber board to hold full size images before transferring them through the PCI bus to the main memory. Frame grabbers without onboard memory use a small FIFO buffer to store one line of an image, and when

the frame grabber has access to the PCI bus, the buffer is emptied. If a frame grabber has DMA abilities, then data can also be sent to main memory without using CPU cycles. For the overhead manipulator agents, a frame grabber without onboard memory but with DMA abilities will suffice.

6. RS-170 is the North American monochrome video standard using a 525 interlaced line system and a frame rate of 30 Hz. Pictures are 640×480 in size, since some of the horizontal lines are used during the vertical blanking interval. CCIR is the monochrome standard used in Europe with a 25 Hz frame rate. Most gray-scale frame grabbers support both formats.

7. Bus design. There are frame grabbers for many types of buses including ISA, embedded PC/104-Plus, and standard PCI. For our purposes, the manipulator computer has an ATX motherboard with standard PCI slots.

8. Device drivers. Most frame grabbers used today support various Windows and Unix platforms. During the time vision hardware was being sought, a frame grabber could not be found which would support LynxOS. Therefore, it was decided that device drivers would have to be written for a frame grabber.

The precision frame grabber chosen in the minifactory project was the Imagenation PX610[1]. Table 1 lists its specifications.

Video format	RS-170 or CCIR
Pixel jitter	$\leq \pm 3$ nm
Gray-scale noise	≤ 0.7 LSB
Progressive scan support	yes
Onboard memory	no
Bus type	PCI
Cropping and scaling support	yes
Gain and offset adjustment	yes
Lookup table	yes

Table 1: Imagenation PX610 specifications

The actual location of the frame grabber board and the manipulator's motherboard is in the brain box as shown in Fig. 3.

[1] Imagenation Corp., 10220 SW Nimbus Ave., Suite K-5, Portland, OR 97223

3.2 Camera and Lens System

A small board level camera[2] was purchased from Edmund Scientific[3] with a CCD size format of $\frac{1}{3}$ inches, an active sensor area of 4.8×3.6 mm, and 512×494 pixels. The video format is RS-170, properly matching the monochrome frame grabber. Automatic gain control as well as automatic shutter control was also built into the camera.

The lens system, end effector casing, and quick connector were designed by Patrick F. Muir. A simple compound lens setup was used to achieve approximately 0.75 magnification with one pixel equalling approximately 10 μm at a working distance of 33 mm. Figure 6 shows the locations of the camera, lens, lighting, gripper, and quick connector on the end effector.

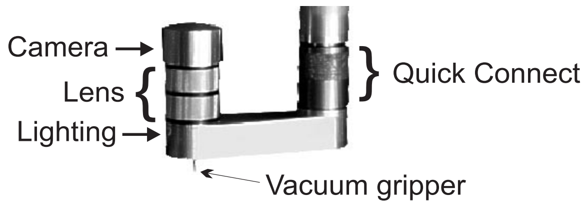


Figure 6: The location of the camera and lens parts in the end effector.

4 Vision Related Software

4.1 Frame Grabber Software

Due to the lack of frame grabber support for LynxOS, software had to be developed in order to use the hardware. As shown in Fig. 7, the software development was split into three main levels. The top level includes the user program which needs access to the frame grabber. The second level is the application programming interface (API) which allows the user application easy access to the frame grabber functions. The bottom level is the frame grabber device driver, which is actually a part of the operating system kernel.

The application programming interface provides the user application with a simple interface to the device driver and frame grabber. It is a library of functions, some of which communicate directly to configuration registers on the frame grabber through memory mapping. These functions include setting the frame grabber gains and offset, setting the image size to acquire, setting the acquisition to grab in field or frame mode, and setting the acquisition to grab in continuous or single image mode. Other API functions used to start and stop the image acquisition required interactions with the device driver because the interrupt service routine was needed.

A device driver is an interface between an operating system and a piece of hardware. In LynxOS, drivers are a part of the kernel and are composed of functions, known as entry point routines [19], which provide an interface between the kernel and the application using the device driver. Entry point routine

[2] Edmund Scientific, Part #H53306

[3] Edmund Scientific Company, Industrial Optics Division, 101 East Gloucester Pike, Barrington, NJ 08007

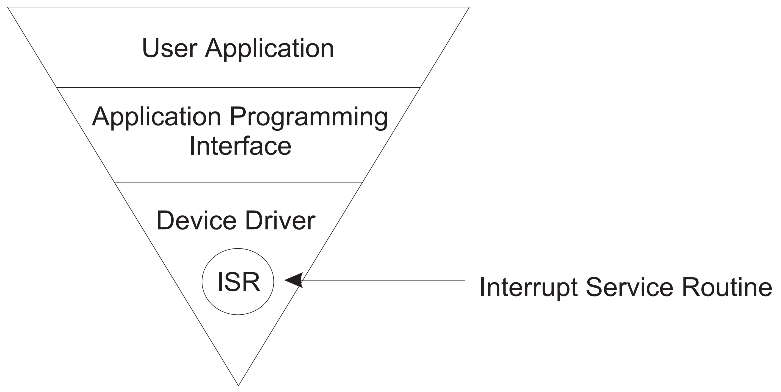


Figure 7: Illustration of how a user application accesses the device driver.

types include *install, uninstall, open, close, read, write*, and *ioctl*. For example, when an application wishes to use the frame grabber device driver, the C code involves using an open entry point:

```
fd=open("px610",READ);
```

This statement calls the open entry point which allows this application to use the frame grabber through various *read* and *ioctl* calls.

Another important part of a device driver is the interrupt service routine (ISR). Interrupts are generated by external hardware to indicate the occurrence of an event, such as data availability. For example, the PX610 frame grabber generates an interrupt every time a field has been acquired and sent to the computer's main memory. In LynxOS, an ISR runs at the highest priority and all other interrupts are disabled while it is running [19]. This means the length of an ISR must be short so other real-time processes are not starved of CPU cycles.

The frame grabber interrupt service routine was developed and trimmed to be as efficient as possible while providing enough functionality to support continuous and single frame/field mode pictures. After each interrupt, the frame grabber must look at the current state of the system to determine which DMA addresses need to be reset and whether or not any processes waiting for an image need to be awaken. Depending on whether a single snapshot or continuous images are being acquired and whether or not images are being captured as fields or frames, the ISR can become difficult to manage. After putting much of the overhead management into the API about whether the image acquisition was in field or frame mode and continuous or snapshot mode, the ISR operated more efficiently. Figure 8 shows the flowchart for the final frame grabber ISR implementation.

In order to test the device driver and API functions, a command line test program written in C was developed. To make the command line program more user friendly, a Tcl/Tk GUI wrapper program was also created. As seen in Fig. 9, the program can be used to quickly test the frame grabber or to simply take an image. The parameters `X Start` and `Y Start` allow the user to choose how soon the frame grabber should start digitizing after the vsync and hsync signals since not all cameras start sending out a valid picture signal at the same times. In addition, the `X Size` and `Y Size` parameters allow the user to

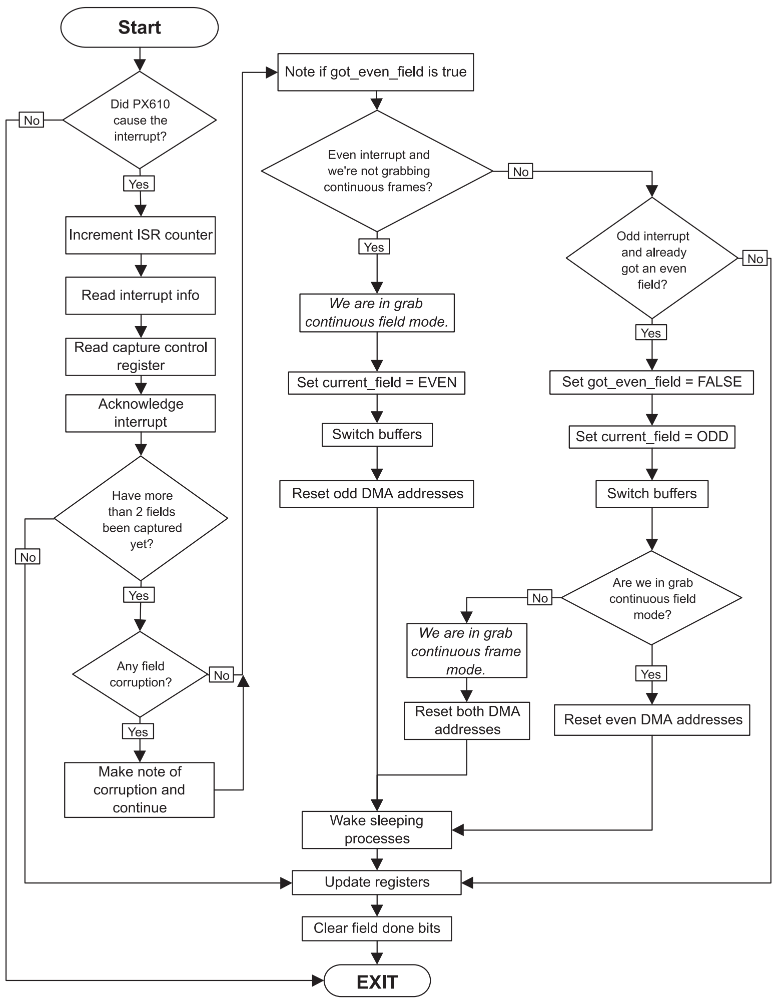
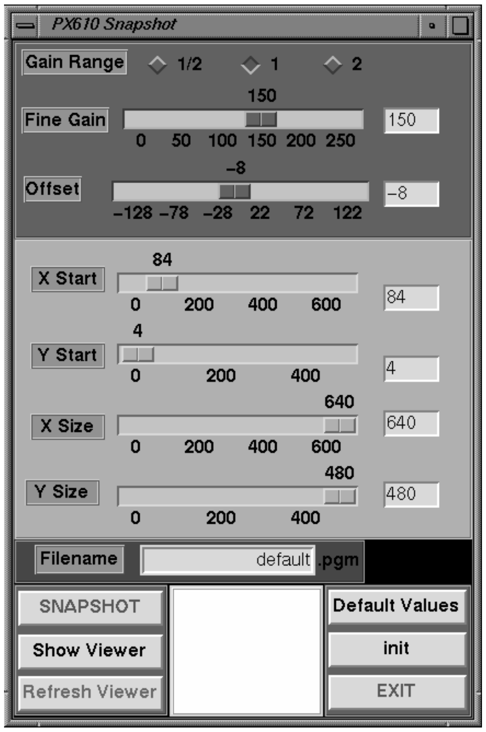

Start
Did PX610 cause the interrupt?
No
Yes
Increment ISR counter
Read interrupt info
Read capture control register
Acknowledge interrupt
Have more than 2 fields been captured yet?
No
Yes
Any field corruption?
No
Yes
Make note of corruption and continue
Note if got_even_field is true
Even interrupt and we're not grabbing continuous frames?
No
Yes
We are in grab continuous field mode.
Set current_field = EVEN
Switch buffers
Reset odd DMA addresses
Odd interrupt and already got an even field?
No
Yes
Set got_even_field = FALSE
Set current_field = ODD
Switch buffers
Are we in grab continuous field mode?
No
Yes
We are in grab continuous frame mode.
Reset both DMA addresses
Reset even DMA addresses
Wake sleeping processes
Update registers
Clear field done bits
EXIT

Figure 8: Interrupt service routine flowchart.

choose the size of the image. The test program is currently limited to testing single image acquisitions.

4.2 Camera Calibration

Camera calibration is a procedure used to obtain camera system parameters which allow us to relate image points, as seen by the camera, to real world coordinates. Parameters which are normally determined include intrinsic parameters such as focal length and lens distortion, and extrinsic parameters which relate the camera coordinate frame to the world coordinate frame. Without these data, a robot would not be able to properly determine the distance it would need to move in order to properly align its end effector with a target. The experiments performed in this report required camera calibration to determine where to move the end effector so that the manipulator and courier could align an electret, held by the gripper, with a diaphragm sitting on the courier.


PX610 Snapshot

Gain Range ◇ 1/2 ◇ 1 ◇ 2

Fine Gain 150 — 150

Offset -8 — -8

X Start 84 — 84

Y Start 4 — 4

X Size 640 — 640

Y Size 480 — 480

Filename default .pgm

Figure 9: Graphical User Interface to test some of the frame grabber's functions.

4.2.1 Tsai's Camera Calibration Technique

Tsai has developed an efficient and widely used camera calibration technique [20]. To use his camera calibration method, an understanding of the relationship between various camera, lens, and frame grabber parameters is required. Tsai determined what parameters would be calibrated and what parameters were needed to perform the calibration by describing a four-step transformation of a point in 3D world coordinates into pixel coordinates [20]. The first step transforms a point in world coordinates to camera coordinates. This involves the finding the associated rotation matrix R and translation vector T.

The second step of the transformation uses perspective projection to map the point's 3D camera coordinates (x, y, z) onto the image plane (X_u, Y_u) of a simple pin hole camera model as seen in Fig. 10. This requires finding the effective focal length of the camera.

The third step of the transformation accounts for radial lens distortion components D_x and D_y. As seen in Fig. 11, (X_u,Y_u) are the undistorted image plane points and (X_d,Y_d) are the distorted image plane points after accounting for lens distortion. The lens distortion model requires two coefficients, k_1 and k_2.

The fourth step of the transformation maps the distorted point (X_d,Y_d) into pixel coordinates (X_f,Y_f). This requires knowledge about the distance between adjacent sensor elements in the camera (d_x and d_y), the number of sensor elements in the x direction of the camera (N_{cx}), the number of pixels sampled in the x direction by the frame grabber (N_{fx}), and the computer image coordinate of the image plane origin (C_x,C_y). Figure 12 illustrates some of these parameters.

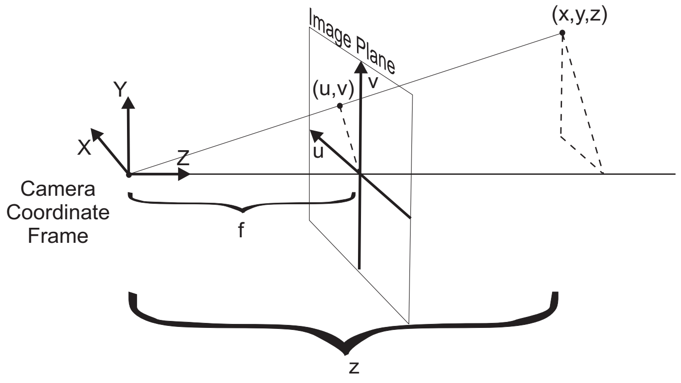
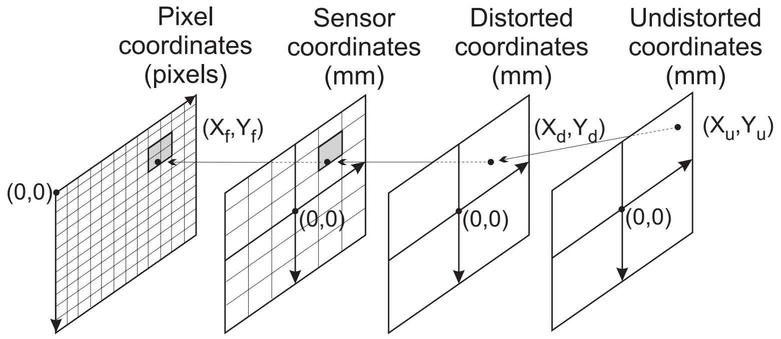


Figure 10: Perspective Projection using a pin hole camera model.



Pixel coordinates (pixels)
Sensor coordinates (mm)
Distorted coordinates (mm)
Undistorted coordinates (mm)
(Xf,Yf)
(Xd,Yd)
(Xu,Yu)
(0,0)
(0,0)
(0,0)
(0,0)

Figure 11: Relationship between a point on the image plane and its corresponding pixel location.

The pixel coordinates (X_f,Y_f) are given by

$$X_f = s_x \cdot \frac{X_d}{d_x} \cdot \frac{N_{fx}}{N_{cx}} + C_x, \tag{1}$$

$$\text{and} \quad Y_f = \frac{Y_d}{d_y} + C_y. \tag{2}$$

Equations (1) and (2) represent the final pixel location after accounting for the offset of the image plane's origin in pixel coordinates.

From the above equations, Tsai determined that the parameters needed in advance to perform camera calibration were the distance between sensor elements in the CCD (d_x,d_y), the number of pixels sampled by the frame grabber per line (N_{fx}), and the number of sensor elements in the x direction (N_{cx}). The values calculated by calibration include the transform from the camera coordinate frame to the world coordinate frame (R and T), the focal length of the camera system (f), radial lens distortion coefficients (k_1,k_2), and the timing inaccuracies in the camera and frame grabber (s_x).

Tsai described his calibration method as a two-stage calibration technique. This first stage involved computing the 3D orientation and x and y position of the camera coordinate frame with respect to world coordinates. The second stage involves computing the effective focal length, lens distortion coefficients,

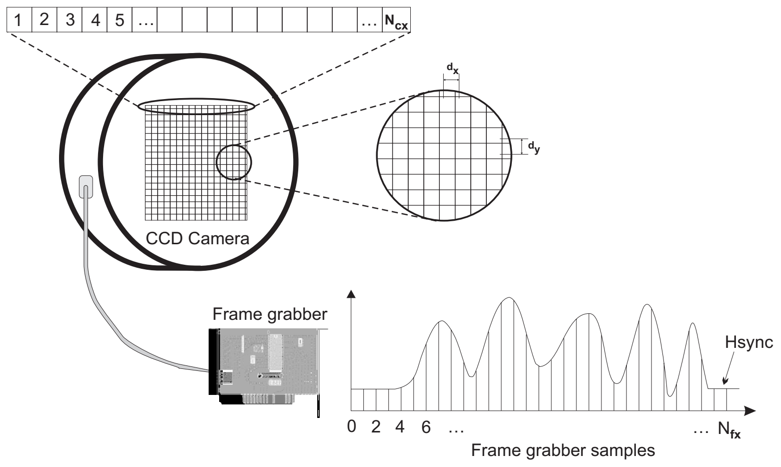


Figure 12: Various measurements needed for camera calibration.

and the z position of the camera coordinate frame with respect to the world coordinates frame using an optimization scheme such as steepest descent. The full derivation of the algorithm can be found in [20].

4.2.2 Camera Calibration and the Minifactory

Camera calibration is required in the minifactory to provide information which will ultimately allow 2D image plane points to be transformed into 3D world coordinates.

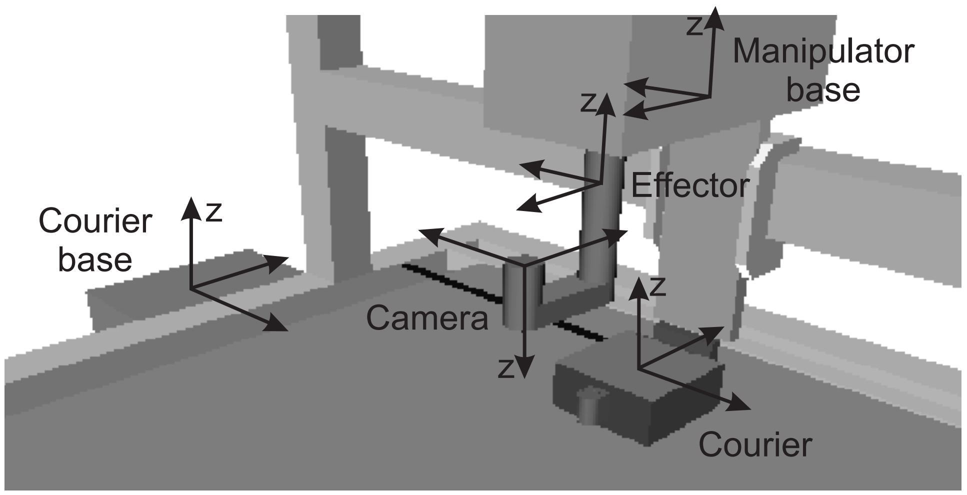


Figure 13: Coordinate transforms in the minifactory.

Figure 13 illustrates the various coordinate frames needed by a minifactory with an overhead manipulator and courier. During initialization of the minifactory, self-calibration is performed providing transformations between various coordinate frames such as the the courier in the courier base frame of reference ${}^{cb}H_{cour}$, the courier base in the manipulator base frame of reference ${}^{mb}H_{cb}$, and the manipulator base in the end effector frame of reference ${}^{eff}H_{mb}$. However, the camera in the end effector frame of reference

${}^{eff}H_{cam}$ is also needed so that ultimately we can transform points from the camera coordinate frame to points in the courier coordinate frame. By performing camera calibration, the extrinsic parameters of the camera are calculated providing the camera coordinate frame in the courier frame of reference ${}^{cour}H_{cam}$. We can therefore derive ${}^{eff}H_{cam}$ from

$$ {}^{eff}H_{cam} = {}^{eff}H_{mb} \cdot {}^{mb}H_{cb} \cdot {}^{cb}H_{cour} \cdot {}^{cour}H_{cam}. \tag{3}$$

After obtaining all the above transforms, image plane coordinates can be transformed into camera coordinates. Once the camera coordinates for a feature are found, they can be transformed into any of the coordinate frames shown in Fig. 13.

To actually perform Tsai's camera calibration in the minifactory environment, three major steps were taken. In the first step, the end effector camera took multiple images of a target sitting at a known location on the courier at different heights. The calibration algorithm required multiple images to differentiate between the effects of focal length and z distance to the target. The target used in our experiments was a micrometer scale target[4] with 200×200 μm squares. A Python script was written to automate the image acquisition.

In the second step, Matlab$^{\text{TM}}$ routines were developed to automatically extract the centroids of the square features from each image, as shown in Fig. 14. For each image, the pixel locations of the features were matched with their respective positions in courier coordinates and the data was saved to a file.

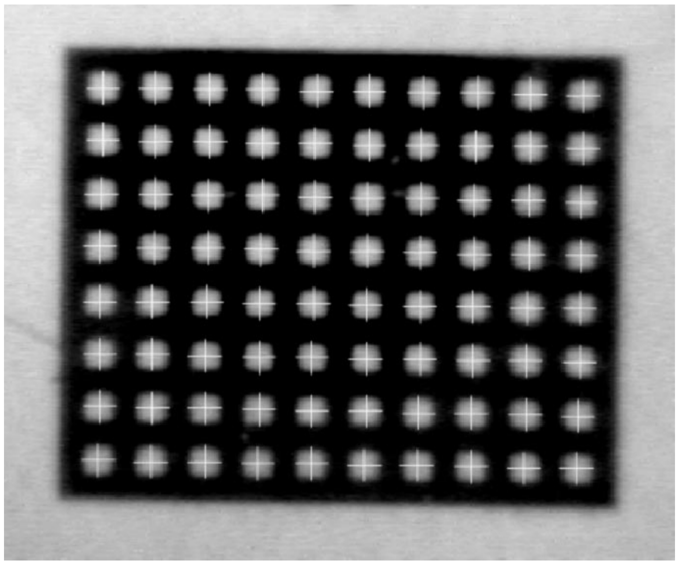


Figure 14: Micrometer scale target used in camera calibration. Centroids found on the target are marked with crosshairs.

The third step used Wilson's [21] C implementation of Tsai's camera calibration routine. The feature locations from step two as well as camera and frame grabber parameters, as previously discussed in Sec. 4.2.1, were passed into this routine. The camera's intrinsic and extrinsic parameters were returned.

[4] Edmund Scientific, Part #H53713

${}^{cour}H_{cam}$ could then be determined from the extrinsic parameters of step three, and implementation of Eq.(3) in Matlab$^{\mathrm{TM}}$ allowed ${}^{eff}H_{cam}$ to be quickly calculated afterwards.

4.3 Vision Library

The vision library was designed in C++ to be accessible by Python code which provides the upper-level management of an agent. For the experiments described in this paper, the library also required the ability to find the corners of electrets and diaphragms which were shown in Fig. 5. The parts were not expected to be rotated more that $20°$ on the image plane. It should also be noted that much of this code was being written because no commercial computer vision packages were available for LynxOS at the time.

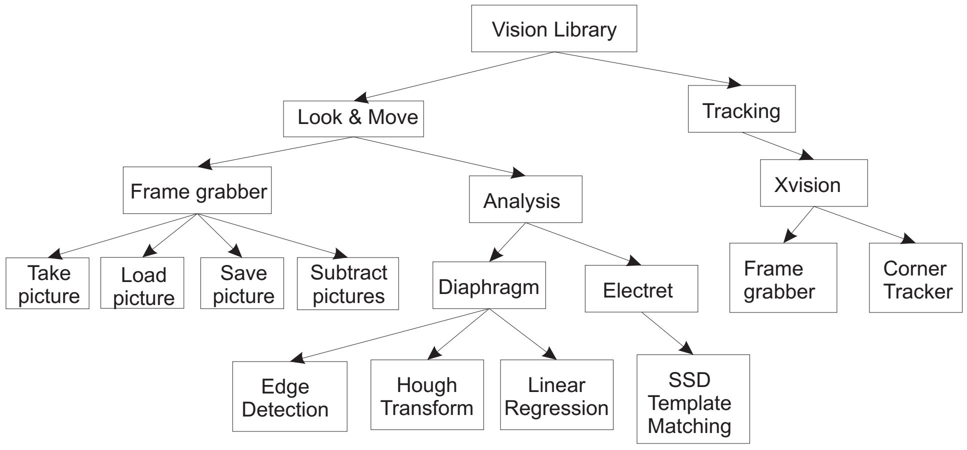


Figure 15: The vision library layout.

4.3.1 Look-and-Move Related Functions

Figure 15 shows the vision library divided into two main sections: look-and-move and tracking. The look-and-move section of the tree is composed of functions which do not need to operate in real time. Using these functions, an image can be acquired and analyzed, and then the Python script can decide where to move the courier and manipulator. Note that this type of look-and-move was referred to as open-loop look-and-move in Sec. 1.4.1.

The frame grabber section of the look-and-move branch incorporates the application programming interface as described in Sec. 4.1, allowing images to be taken, saved, and loaded. The vision library provides three buffers where images may be stored and analyzed. For example, one buffer can contain an image recently taken by the frame grabber, another buffer can contain a picture read from disk, and the third buffer can contain the subtraction of one picture from the other.

Once a picture is in a buffer, it can be analyzed by the functions in the analysis branch of the look-and-move section as shown in Fig. 15. This branch currently has two main types of analyses, one to find the position of an electret, and the other to find the position of a diaphragm. Before discussing how both analyses work, a brief description will be given about a software program called X Vision, which has been incorporated into the vision library,

X Vision was designed to perform real-time tracking using a desktop computer with a frame grabber [22]. Other software tracking systems have been developed previously, but they were all specific to a task and generally not easily portable to a new application. X Vision aims at providing a fast, portable, and reconfigurable software platform for real-time vision. The philosophy behind the software is that although there are many types of tracking applications, the features used in any single application are variations on a small set of primitives such as lines and textures. X Vision provides speed by looking for these primitives in a small region of interest, also known as a window.

Objects which are tracked using X Vision are actually composed of less complex objects. This hierarchical form continues until the object has been broken down entirely into primitives. An object is described by a state vector which may impose constraints on the features that compose it. For example, a square tracker could be considered a complex object composed of four corner features, and each of these corners is composed of two line primitives. Corner trackers may be constrained so the corners they find represent a square shape, while the line trackers composing a corner are constrained so they always intersect to form a corner. X Vision provides line trackers, corner trackers, and sum of the squared differences (SSD) trackers. The SSD trackers perform a template matching function similar to normalized correlation with rotation [22].

4.3.2 Electret Analysis

When analyzing an electret to find its center and orientation, one general solution would include finding all four corners of the object. A line connecting the midpoints of the top and bottom sides would then intersect a line connecting the midpoints of the left and right sides, and this intersection could be used as the centroid of the part. However, as shown in Fig. 16, one of the problems of the eye-in-hand robotic configuration is that the gripper, in this case a vacuum suction device, occludes a significant portion of the part as seen by the camera. Therefore, an alternative approach of finding only the top two corners was devised. This approach uses the fact that the lengths of each side of the electret part are equal. By finding the top two corners, the length of the top side could be found, and the angle made by the line connecting the corners would give the orientation of the part. The length, angle, and geometry of the electret could then be used to locate the center of the part.

The first step in locating the corners of an electret was to approximately find the centroid of the image. Since the electret is the only object in the image, the centroid will belong to it. However, it should be noted that the centroid will be biased slightly towards the upper right since the gripper blackens out part of the electret.

The second step in locating the corners involved using the approximate centroid as a starting point to guess the location of the two corners of the electret. Finding a corner on the electret is not trivial since the sides cannot be approximated by a single line. Unlike a diaphragm, the corners of an electret extend slightly outside the main body. If enough prior information was given, such as the approximate orientation and position of the electret, then local searches around each corner could be performed and the proper lines extracted. However, the goal of this routine was to find the electret in orientations up to $\pm 20^\circ$ without any initial approximations of its orientation. Therefore, normalized correlation with rotation, which is essentially pattern matching, was chosen to find the top two corners of the electret.

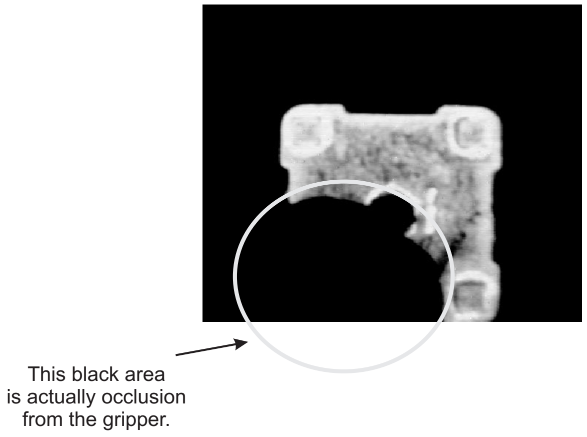


Figure 16: An electret being held by a vacuum pickup device which causes an occlusion.

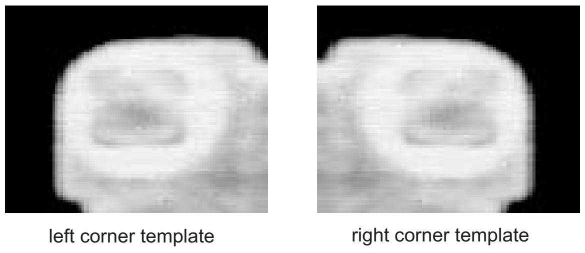


Figure 17: Magnified view of the left and right corner templates used with the SSD tracker.

SSD trackers, provided by the X Vision package, were used to find each corner. However, for this electret finding routine, the SSD trackers were applied to a static image (instead of tracking a moving object through multiple images). As seen in Fig. 17, templates were developed from real parts beforehand. As depicted in Fig. 18, four SSD trackers were applied to each corner and allowed to converge to the corner for a specified number of iterations. Depending on where a tracker was initially located, it may or may not converge at all. However, if the electret is not severely rotated, at least one of the trackers would converge reasonably well. Therefore, for each corner, the tracker with the lowest residual after the iterations was considered the best guess for the location of that corner.

Once both corners were found, the orientation and position of the electret could be determined given the geometry of the part.

4.3.3 Diaphragm Analysis

Determining the center of a diaphragm has the same problems as determining the center of an electret since the lower two corners may be occluded. The problem is actually worse for a diaphragm because both

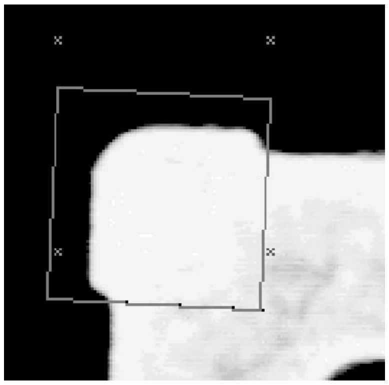


Figure 18: An SSD tracker converging to an electret corner. The square represents the current tracker position while the four surrounding dots represent the initial locations for 4 trackers.

the gripper and the electret being held by the gripper are occluding it. Significant amounts of diaphragm edge information may be lost if the initial alignment, before using vision, is poor. This will prove to be a problem as described later in this section.

The first step in finding a diaphragm requires finding its approximate center in an image. When the gripper is lowered toward the diaphragm, only one diaphragm will be in view. Therefore, one could simply find the centroid of the image. However, if the gripper is already carrying an electret, then the electret will interfere with finding the centroid of the diaphragm. One way to eliminate the effects of the electret is to subtract the electret image from the combined electret and diaphragm image as shown in Fig. 19. The remaining pixels are only related to the diaphragm and can be used to find its approximate centroid. Note that the centroid will be biased in a certain direction depending on where and how much of the diaphragm was occluded.

Once the approximate centroid of the diaphragm is found, edge finding is required to find the left, right, and top sides of the diaphragm. Therefore, code was written which analyzes sections of the diaphragm at a time, as seen by the white grids in Fig. 20. Within each grid, the edge is calculated by averaging the columns or rows of the grid and then using linear interpolation to find its location to sub-pixel accuracy.

After points are found on the left, right, and top sides of the diaphragm, linear regression is used to find a line representing each side of the diaphragm. The intersection of the three lines provides the locations of the top two corners of the part. Then, using the same method described in the electret analysis section, the corner positions can be used to determine the center and orientation of the diaphragm as marked by the crosshairs in Fig. 19.

After using the above diaphragm finding routine in the minifactory experiments, it was seen that the pallets holding the diaphragm pieces were often scratched, appearing as bright lines or patches in the image. This often interfered with the edge finding routine. In addition, depending on the amount of occlusion of the diaphragm by the gripper and electret, too much of one side of the diaphragm may have been missing for the routine to succeed. Therefore, a new method involving the use of the Hough transform to eliminate outliers was developed. While the old edge finding routine tried to find only a minimal number

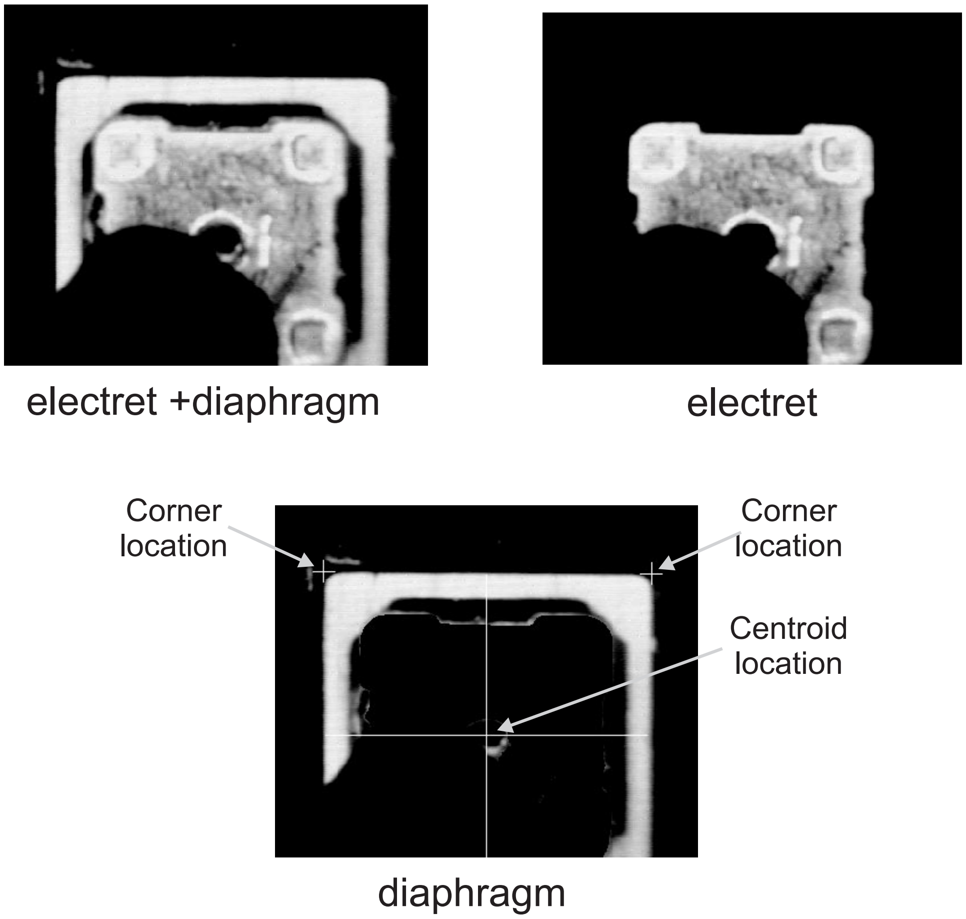


Figure 19: Subtracting the electret image from the diaphragm image. The small crosshairs on the resulting image are the corners which were found. The large crosshair is the centroid.

of points on each side of the diaphragm to minimize the probability of finding a bad point, the new edge finding routine searches for as many points on a side as possible and then uses the Hough transform to find the line that fits the most points.

The Hough transform is a parameter estimation technique which uses a voting mechanism [23]. Using the equation for a straight line $y = mx + b$, the Hough transform works by mapping each point in x-y space to it's equivalent representation in m-b parameter space, as seen in Fig. 21. The parameter space plot shows all the possible slope (m) and y-intercept (b) values for lines which pass through a given (x, y) coordinate. The parameter space must be quantized into accumulators, and then an accumulator must be incremented by one for each line that passes through it. The accumulator with the largest value corresponds to the slope and y-intercept value of the line which fits the most points.

Since the parameter space is quantized, the resulting best fit line is actually not very precise. While it is possible to make the quantization very small, this increases the number of accumulators which must be accounted for, thereby increasing the computational load. Therefore, the next step of the diaphragm edge finding algorithm determines which points in x-y space contributed to the winning accumulator. As seen

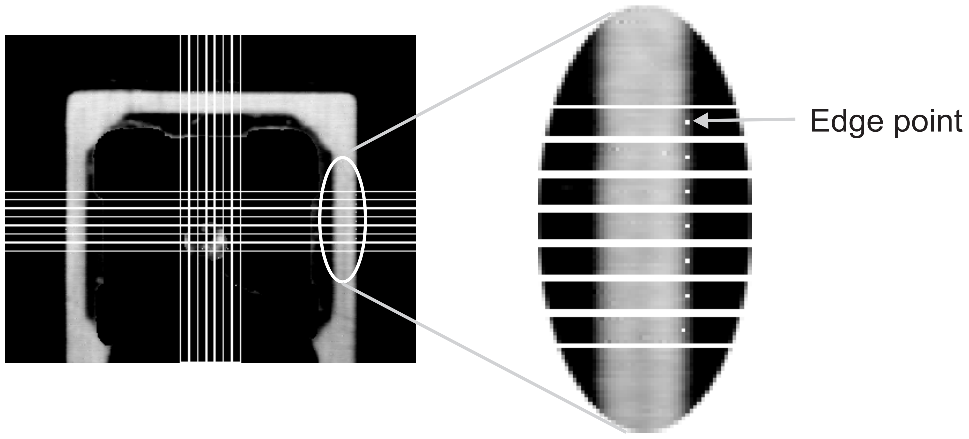


Figure 20: The diaphragm edge finding method first subtracts out the electret image and then finds points along the left, right, and top edges of the device.

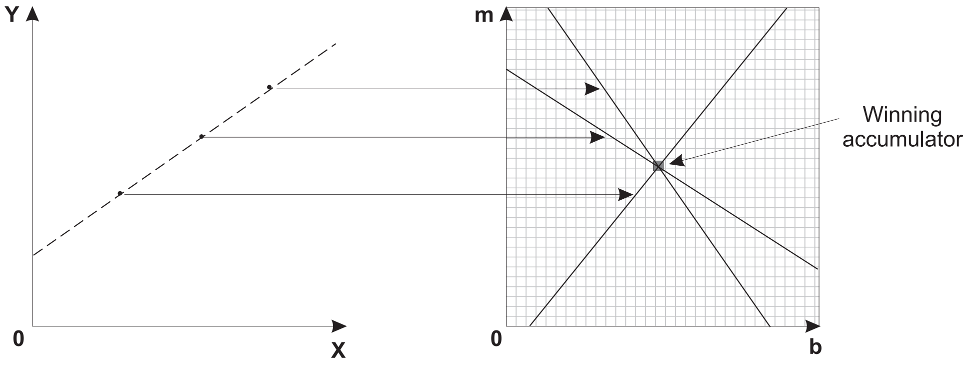


Figure 21: Mapping of image plane points into parameter space.

in Fig. 22, this effectively eliminates most outliers and allows linear regression to be successfully applied to the remaining points.

By applying this new edge finding algorithm to the left, right, and top sides of the diaphragm, the corners and the centroid of the diaphragm can be found as previously discussed. Figure 23 shows an example of how robust the technique is to noise. The electret image and the combination electret and diaphragm image were previously taken, and then the combination image was manually modified (lines were drawn in and parts of the image were blackened out). The images were then processed by the new diaphragm finding routine, and as can be seen by the location of the crosshairs, the diaphragm was successfully located.

4.3.4 Trackers

The second main branch of the vision library, as seen in Fig. 15 is the tracking branch. This branch currently uses X Vision corner trackers (composed of line primitives) to track the corners of the diaphragm during visual servoing. Since trackers work within small windows of the image, they initially need refer-

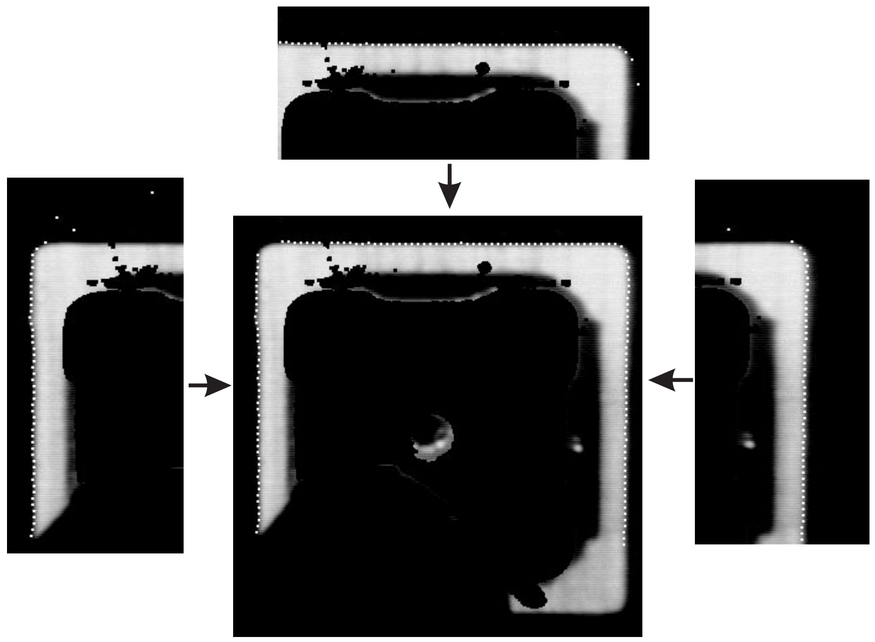


Figure 22: Elimination of outliers from each side of the diaphragm. The outer three pictures have false edges detected while the center image shows that the outliers where rejected.

ence points to find the corners. This can be done using the diaphragm finding methods described above. Once the global search for the corners are complete, they are used as the initial tracking positions for the corner trackers. Since the bottom two corners might be occluded by the electret and gripper, only the top two corners are tracked. Given the diaphragm's geometry and the orientation of one corner tracker, the position of the other corner tracker can be estimated in case that tracker becomes lost.

The diaphragm trackers operate on image fields rather than frames because of the blurring seen in frames when capturing images of moving targets, as described in Sec. 3.1. Besides eliminating the blur, the positive side of analyzing fields includes a faster sampling frequency for closed-loop visual servo control (60 Hz instead of 30 Hz). The downside is that the vertical resolution of the resulting images is only 240 pixels instead of 480 pixels. This decrease in resolution makes it more difficult to accurately determine where a point is in world coordinates given the pixel coordinates.

4.4 Look-and-Move Communications Infrastructure

Figure 24 shows the communications infrastructure needed for an open-loop look-and-move task. The top portion of the figure contains all the processes running on the overhead manipulator agent while the bottom portion shows the processes running on the courier agent. Circular and oval shapes represent processes.

As discussed in Sec. 1.2, Python is the scripting language used to manage high-level factory operations. The manipulator head and a courier head processes (created by Jay Gowdy) run the Python code on the agents. Each head process communicates directly with the executor program running the low-level controllers for the given agent (the courier executor was created by Arthur Quaid and the manipulator

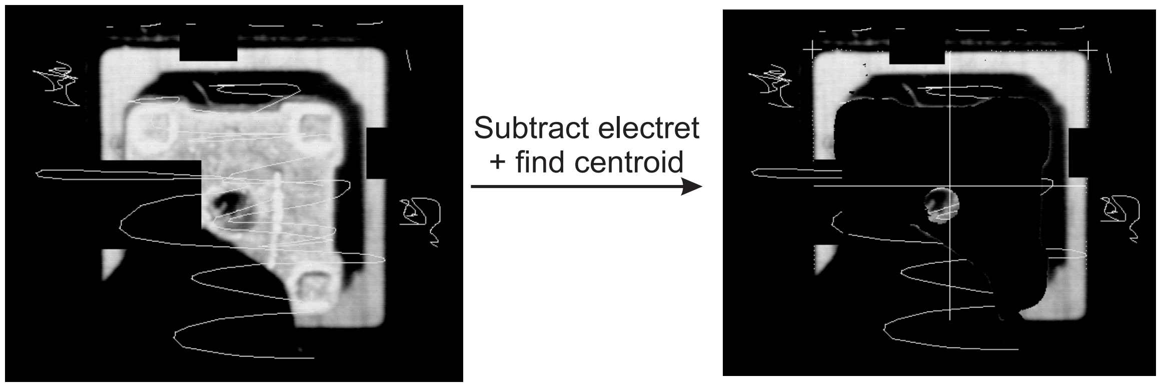


Figure 23: Robustness of the new diaphragm finding algorithm. A previous set of data images was manually modified and then processed by the function.

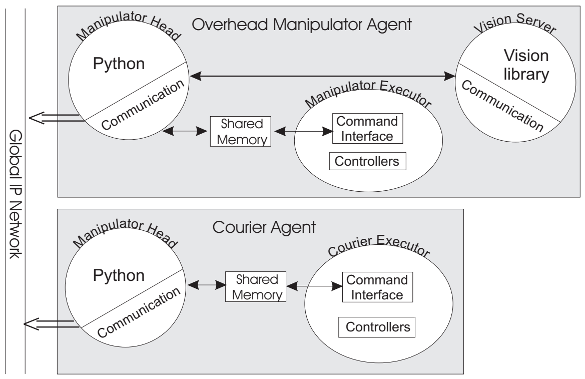


Figure 24: The communications infrastructure used for open-loop look-and-move.

executor was created by Alfred A. Rizzi). Whenever the manipulator head wishes to have an image analyzed, it contacts the vision server which then processes the image and returns the results. The vision server process contains the vision library and has wrappers which allow Python to call its C++ member functions.

There are essentially two levels of communication infrastructure needed, inter-agent and intra-agent. Intra-agent communications occurs between processes within an agent. Inter-agent communications occurs between the manipulator and courier heads using the global IP network. Both levels of communication are more complicated for visual servoing, as described in Sec. 4.5.

For the open-loop look-and-move setup, information related to vision processing and the camera setup has been isolated as much as possible from the manipulator head process to enforce modularity. For example, the vision server is only concerned with what is occurring on the image plane. It does not handle real world 3D coordinates since its primary function is to analyze images. However, the manipulator head process overseas the manipulator by using the appropriate 3D coordinate frames, not by using 2D image plane information. If the manipulator head process needed to determine how a point on the image plane mapped into 3D coordinates, it would require the focal length of the camera as well as the z distance from the camera coordinate frame to the object being measured (using similar triangles). However, the focal length is stored on the vision server while the z distance from the camera coordinate frame to the object is stored in the manipulator head process. Therefore, the 3D coordinate can be recovered by either sending the focal length from the vision server to the manipulator head process, or sending the z distance from the manipulator head process to the vision server. However, it would be preferable not to pass either piece of information between processes, and to keep information related to the vision system cleanly separated from the rest of the system.

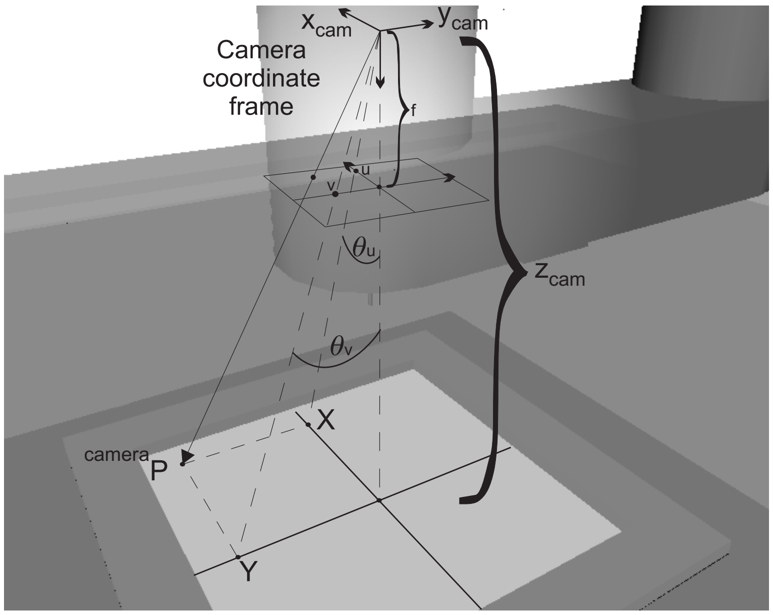


Figure 25: Information is passed between the vision server and the manipulator head process through the use of solid angles.

One solution to the above problem is depicted in Fig. 25 which shows the relationship between a point on the courier in camera coordinates ${}^{cam}P$, the same point in image coordinates (u, v), the z distance from the camera coordinate frame to the point on the courier, and the focal length of the camera. When information about the image plane point is transferred from the vision server to the manipulator head process, it is first changed to solid angle representations:

$$\theta_u = \tan^{-1}(\frac{u}{f}) \; , \tag{4}$$

$$\text{and} \quad \theta_v = \tan^{-1}(\frac{v}{f}) \; . \tag{5}$$

The manipulator head process receives the position of the point as solid angles. Now it can use its knowledge about the z distance between the part on the courier and the camera coordinate frame's origin:

$$X = Z_{cam} \tan \theta_u \; , \tag{6}$$

$$\text{and} \quad Y = Z_{cam} \tan \theta_v \; . \tag{7}$$

Therefore, ${}^{cam}P = [X, Y, Z_{cam}]^T$ and using the transforms described in Sec. 4.2, ${}^{cam}P$ can be transformed into any minifactory coordinate system. Using solid angles to transfer information between the vision server and the manipulator head provides a clean interface, allowing camera calibration information to stay within the vision server and factory state information to stay inside the manipulator head process.

4.5 Visual Servoing Communications Infrastructure

To coordinate the overhead manipulator and courier movements during a visual servoing task, a communications infrastructure had to be properly designed and implemented in addition to what was already implemented for the open-loop look-and-move tasks. Figure 26 shows the additional structures needed for visual servoing tasks.

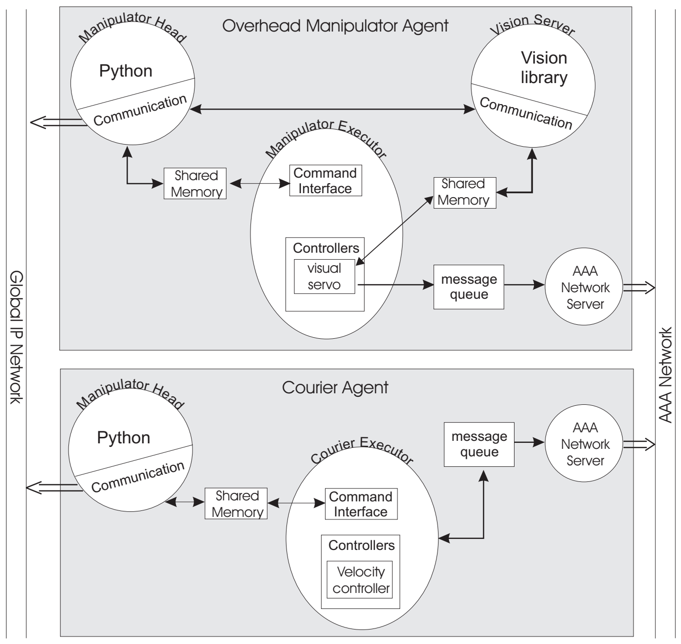


Figure 26: The communications infrastructure used for visual servoing.

The inter-agent communications between the the courier and manipulator occur using two networks: the global IP network and the AAA network. Both networks use 100 Mb Ethernet hardware, but the global IP network is used for non-critical information such as monitoring information being passed to the factory interface tool shown in Fig. 4. Using standard protocols such as TCP and UDP, this network is useful for non-real time information. However, the AAA network (designed and implemented by Shinji Kume) is a semi-custom network designed to carry real-time information in the minifactory environment. Using a low-overhead protocol called AAA-net, low-latency and high bandwidth communication is achieved even though the visual servoing routines only transmit data at 60 Hz. The AAA network provides the inter-agent communications needed for the manipulator and courier to behave in a master-slave fashion during a servoing task. It should be noted that while the visual servoing requires the AAA-net, the open-loop look-and-move does not.

Comparisons between Fig. 24 and Fig. 26 show that the new infrastructure in the manipulator includes communications between the vision server and a visual servo controller running in the manipulator executor. This was required in order to transfer 60 Hz field rate data since the communication between the manipulator head and the manipulator executor was limited to about 10 Hz. The communication of data between the vision server and executor utilized shared memory structures, providing faster data sharing than possible with alternative methods such as pipes or sockets. Another new communications structure in the manipulator is a connection between the visual servo controller and a message queue used by the AAA network server. The AAA network server manages the flow of information to and from processes on the manipulator and the AAA network.

The intra-agent communications are slightly different between the courier and manipulator. In the visual servo system, the courier has only one new structure because it does not have a vision server process. The new structure is a communications channel between the controller, running in the courier executor program, and the courier's AAA network server.

To understand how the various communications facilities within an agent and between agents are used, the general flow of information in a minifactory visual servoing program will now be described. The first step involves the self-calibration of the factory where the manipulator head and the courier head communicate with their executor programs to coordinate the agents' movements. The global IP network allows the user to monitor the individual states of each agent while this is occurring. After calibration, the courier and the manipulator heads continue to coordinate activities until the gripper has picked up an electret and is holding it 1 mm above a diaphragm on the courier. The manipulator head instructs the vision server to determine the electret's position and orientation and then to begin tracking the diaphragm's position and orientation. It also instructs the manipulator executor to run the visual controller and the courier executor to run a closed-loop velocity controller. The vision server then directly updates position information at 60 Hz through shared memory to the visual servo controller in the manipulator executor. As the controller processes the information, it issues commands to update the overhead manipulator z and θ positions. It also updates the courier position by sending velocity information through the AAA network. The courier executor, which is running the velocity controller, is constantly monitoring the AAA network for new courier velocity information.

4.6 Visual Servoing Control

As described in Sec. 1.4.2, the visual servoing controller is an image-based controller, relating movement of features on the image plane to movement of the courier and manipulator. The control strategy will rely on the image Jacobian to determine how the courier and manipulator should be moved to correct for image plane errors. To develop the image Jacobian, the forward kinematic relationship between the camera frame of reference and the courier frame of reference ${}^{cam}H_{cour}$ must first be determined:

$$ {}^{cam}H_{cour} = {}^{cam}H_{eff}\; {}^{eff}H_{mb}\; {}^{mb}H_{cour} \;. \tag{8}$$

Therefore, the equation relating a point in the camera coordinate frame to a point in the courier coordinate frame is:

$$ {}^{cam}P = {}^{cam}H_{cour} \, {}^{cour}P \ . \tag{9} $$

However, we are interested in points on the image plane, not in the camera frame of reference, and we can switch from 3D camera coordinates into 2D image plane coordinates using perspective projection:

$$ {}^{cam}P = \begin{pmatrix} \frac{{}^{cam}P_z}{f} u \\ \frac{{}^{cam}P_z}{f} v \\ {}^{cam}P_z \\ 1 \end{pmatrix} , \tag{10} $$

where f is the focal length of the camera system.

After substituting Eq.(10) into Eq.(9) and solving for u and v, we can represent the mapping of a point in the courier frame of reference onto the image plane:

$$ r = g(x) \ , \tag{11} $$

where $r = [u, v]^T$ and $x = [{}^{mb}tx_{cour}, {}^{mb}ty_{cour}, \theta]^T$, where ${}^{mb}tx_{cour}$ and ${}^{mb}ty_{cour}$ are the x and y translations of the courier frame of reference to the manipulator base frame of reference and θ is the manipulator's θ axis. Now the image Jacobian can be found by taking the derivative of Eq.(11)

$$ \dot{r} = J(x)\dot{x} \ , \tag{12} $$

where $\dot{r} = [\dot{u}, \dot{v}]^T$, $\dot{x} = [{}^{mp}\dot{tx}_{cour}, {}^{mp}\dot{ty}_{cour}, \omega]^T$, and

$$ J(x) = \begin{pmatrix} \frac{\partial u}{\partial\, {}^{mb}tx_{cour}} & \frac{\partial u}{\partial\, {}^{mb}ty_{cour}} & \frac{\partial u}{\partial \theta} \\ \frac{\partial v}{\partial\, {}^{mb}tx_{cour}} & \frac{\partial v}{\partial\, {}^{mb}ty_{cour}} & \frac{\partial v}{\partial \theta} \end{pmatrix} \ . \tag{13} $$

Using the above derivation, some of the Jacobian elements will be in terms of ${}^{mb}tx_{cour}$ and ${}^{mb}ty_{cour}$. Updating these values during every iteration of the control loop would require the courier executor, the only process which knows the exact courier position during the servoing loop, to communicate back to the manipulator executor. However, it would be more convenient if the courier executor only needed to receive information and not send it. The ${}^{mb}tx_{cour}$ and ${}^{mb}ty_{cour}$ terms can be replaced by the image plane u and v terms which are already available from the vision server. These substitutions can be found by first obtaining ${}^{mb}P$ from ${}^{cam}P$:

$$ {}^{mb}P = {}^{mb}H_{eff} \, {}^{eff}H_{cam} \, {}^{cam}P \ . \tag{14} $$

Using ${}^{mb}P$, we can find the translational components between the courier coordinate frame and the manipulator coordinate frame ${}^{mb}t_{cour}$:

$$ {}^{mb}t_{cour} = \begin{pmatrix} {}^{mb}tx_{cour} \\ {}^{mb}ty_{cour} \\ {}^{mb}tz_{cour} \end{pmatrix} = {}^{mb}P - {}^{mb}R_{cour}\,{}^{cour}P \ . \tag{15} $$

Substituting the x and y components of ${}^{mb}t_{cour}$ back into Eq.(13), all the elements of the Jacobian are now in terms of ${}^{cam}P_z$, ${}^{cam}t_{eff}$, u, v, f, and the rotational values between coordinate frames which we have. Most importantly, the image Jacobian can now be reevaluated every cycle by only providing updated values for ${}^{cam}P_z$, u, and v.

The above derivation of the image Jacobian actually only uses one (u, v) coordinate. However, in order to determine how the manipulator's θ axis needs to be adjusted, at least one more image-plane coordinate is required. A new Jacobian can therefore be constructed by stacking two single feature Jacobians together:

$$ J(x) = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \end{pmatrix} \ . \tag{16} $$

Relating back to Eq.(12), we now have $\dot{r} = [\dot{u_0}, \dot{v_0}, \dot{u_1}, \dot{v_1}]^T$ and $\dot{x} = [{}^{mp}t\dot{x}_{cour}, {}^{mp}t\dot{y}_{cour}, \omega]^T$. While this equation calculates image plane velocities given courier and manipulator velocities, visual servoing requires finding courier and manipulator velocities given the desired image plane velocities. Therefore, we take the pseudo-inverse of Eq.(12) and obtain:

$$ \begin{pmatrix} {}^{mp}t\dot{x}_{cour} \\ {}^{mp}t\dot{y}_{cour} \\ \omega \end{pmatrix} = J^{+} \begin{pmatrix} \dot{u_0} \\ \dot{v_0} \\ \dot{u_1} \\ \dot{v_1} \end{pmatrix} \ . \tag{17} $$

Equation 17 is the basic control law used in the implementation of the visual servo controller. Note the desired image plane velocity is generated in the controller code by subtracting the current image plane positions from the desired positions and then multiplying by a gain. The basic block diagram of the visual servo controller used to control the courier's x and y axes and the manipulator's θ axis is shown in Fig. 27.

G_c represents a proportional and integral gain for the image plane errors, J^{+} is the inverse Jacobian derived above, and G_p represents the courier and manipulator agents which take velocity input commands.

After $\dot{r}$ is calculated, it is multiplied by the inverse Jacobian and then, in order to change the velocity commands to bias one axis more than the other axes, it is multiplied by another gain G_b. This additional flexibility allows the controller to try to correct for the θ error slightly faster or slower than u and v errors. However, G_b must be a diagonal matrix with positive values to guarantee convergence. These final velocity commands are then sent into the velocity controllers for the courier and manipulator. After the agents move, the vision system H captures the movement on the image plane by finding the diaphragm's current corner positions.

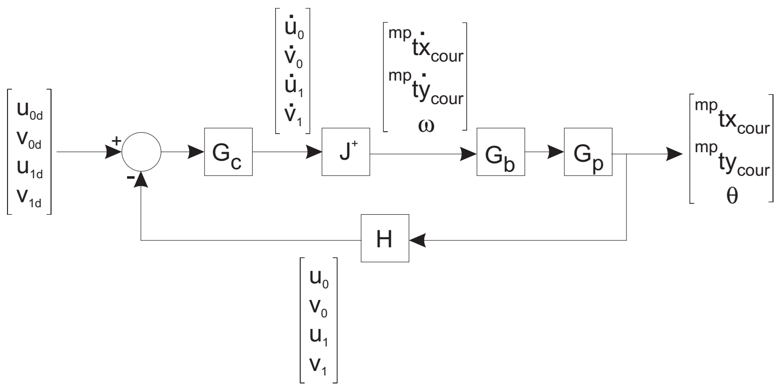

$[u_{0d}, v_{0d}, u_{1d}, v_{1d}]^T$
+
-
G_c
$[\dot{u}_0, \dot{v}_0, \dot{u}_1, \dot{v}_1]^T$
J^+
$[{}^{mp}\dot{tx}_{cour}, {}^{mp}\dot{ty}_{cour}, \omega]^T$
G_b
G_p
$[{}^{mp}tx_{cour}, {}^{mp}ty_{cour}, \theta]^T$
H
$[u_0, v_0, u_1, v_1]^T$

Figure 27: A block diagram of the visual servo controller.

5 Experimental Tests and Results

5.1 Open-Loop Look-and-Move

The open-loop look-and-move experiment was composed of 4 major steps: automatic calibration of the factory, acquisition of an electret by the gripper, presentation of a diaphragm on the courier to 1 mm below the electret, and the look-and-move alignment of the parts. Calibrating the factory provided precise coordinate transforms between various parts of the minifactory as shown in Fig. 13. These transforms were needed to properly move the agents to align the parts as described in Sec. 4.2.2.

After self-calibration, the courier moved to present an electret (2.5×2.5 mm), sitting in a 3.3×3.3 mm cavity, to the manipulator's end effector. The electrets were not firmly seated and were found in different positions and orientations, although generally the orientation was less than $20°$. In a true assembly system, electrets would be seated in cavities which more closely matched their size, but for this experiment we wanted to test the capabilities of the vision system. Therefore, after the courier presented the electret, the vision server took a picture of the electret and analyzed its position and orientation. This information was used to adjust the agents' positions so that the vacuum suction gripper could pick up the electret by its lower left corner. Another picture was taken to determine the electret's position again since the electret may have shifted when it came into contact with the gripper.

The next step involved the courier moving to present a diaphragm to the manipulator. The manipulator head process had enough knowledge about the positions and orientations of the electret and diaphragm so it could instruct the manipulator and courier to move and provide an initial alignment of the parts. The electret was not placed into the diaphragm but was held 1 mm above it.

The fourth step of the open-loop look-and-move task utilized an iterative method to align the two parts. After the initial alignment, a picture was taken containing both the electret and diaphragm, as shown in Fig. 19. Using the diaphragm finding routine described in Sec. 4.3.3, the position and orientation of the diaphragm was found. Note that the position of the electret never moved in the image since it was being held by the gripper, and therefore its position and orientation were the same for every iteration. After finding the differences between the diaphragm's and the electret's center and orientation, the manipulator head process calculated where the agents needed to move to correct for the error. After the agents moved, the vision server waited a short period of time for the agents to stop oscillating[5] before acquiring another image. The system iterated until the manipulator head process decided that the errors were acceptable. Note that the reason the parts were not initially perfectly aligned is due to the slight miscalibration of various factory parameters.

Figure 28 shows a histogram for 100 runs of the open-loop look-and-move experiment. The time measured for each run started at the beginning of the first iteration and ended once the error was considered acceptable. Depending on the initial alignment of the diaphragm and electret, each run took a certain number of iterations before reaching an acceptable error. Error was measured on the image plane and was considered acceptable if both the u and v errors for the centroid positions were less than 20 μm and the angular error was less than 1°.

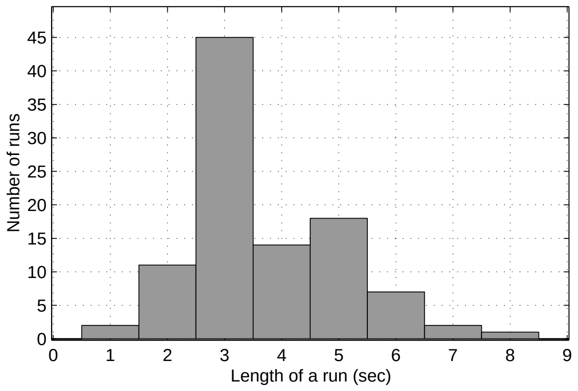


Figure 28: 100 runs categorized by the length of time for a run.

The average iteration time needed for the 100 runs was 3.62 s, and the average number of iterations was 3.69. Figure 29 shows the error between the centroid of the electret and diaphragm as well as the orientation error. The u and v errors are biased around 10 to 20 μm instead of showing a symmetric distribution. However, given the nature of the experimental setup, the majority of the electrets were picked up slightly rotated clockwise. This caused the initial alignment of the electret to be slightly biased to the lower right corner of the diaphragm (which may also be evidence that the initial alignment algorithm needs further development). Therefore, during the iteration processes, the agents move in the same manner for the majority of the runs. This caused most of the u and v errors to be similar, as the histograms show.

It should be noted that the above results measure the performance of the iterative look-and-move procedure, not the performance of the electret and diaphragm finding functions. For example, the iteration routine may believe that the error has reached an acceptable level, yet there might still be real absolute error. This could occur if the electret was not properly detected, since the diaphragm would then be

[5]The manipulator has a small amount of structural vibration caused by the high acceleration and deceleration of the θ axis. Improved control of the acceleration and deceleration will allow the θ axis to reach its goal more smoothly, and it will help alleviate the oscillations. The settling time for the manipulator has already been reduced by utilizing a more rigid manipulator brain box. Shortening and lowering the bridge the manipulator is mounted on will also decrease the oscillations. The settling time for the courier has been improved through closed-loop control [5], but these experiments were still using open-loop control.

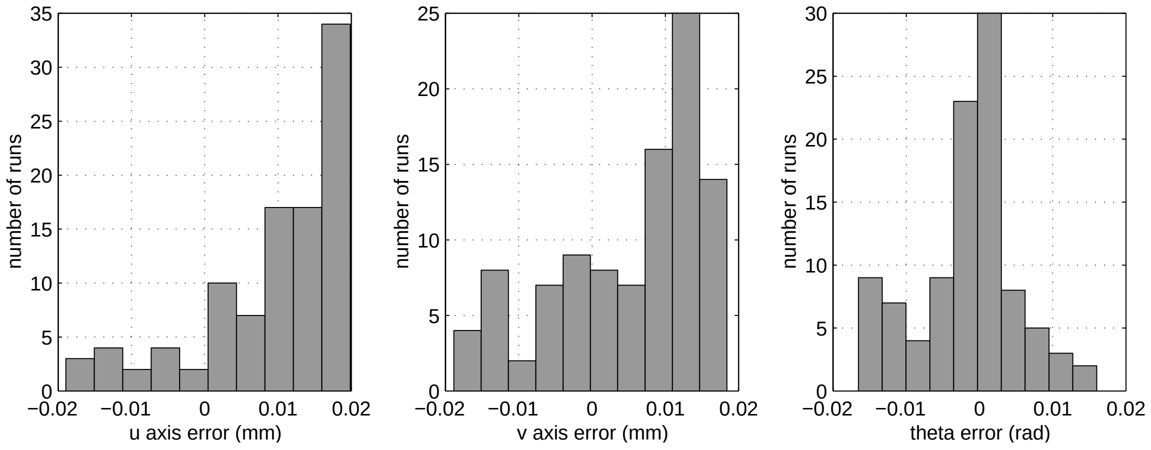


Figure 29: The u and v centroid position error and the orientation error for 100 runs.

aligned to an incorrect location. For these experiments, only qualitative measurements were done in terms of how well the parts were truly aligned. For example, Fig. 30 shows the parts for one run after the initial alignment and after the iterative correction process. The parts seem reasonably aligned.

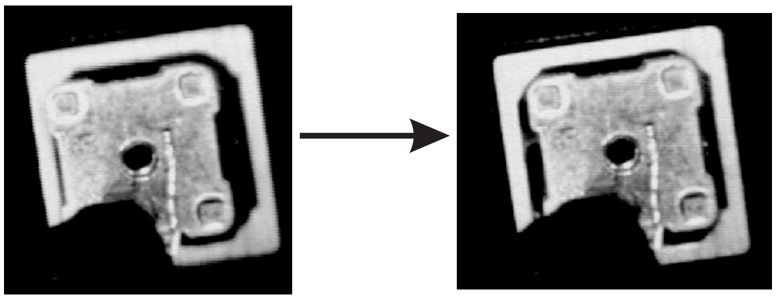


Figure 30: Alignment before the first iteration and after the last iteration of a run.

Note that various factors may make a final alignment appear unreasonable in an image. For example, if the vacuum suction gripper on the end effector becomes slightly bent, it will pick the electret up at an angle with respect to the image plane making a precise alignment difficult. Parts themselves may also be slightly deformed, giving the impression of a poor alignment.

5.2 X-Y Visual Servoing Test

The x-y visual servoing experiment was based on the model of the system shown in Fig. 31. Note that this model differs from the one shown in Fig. 27 since this does not include θ axis control, and instead of explicitly using the diaphragm corner positions in the feedback control, the center of the diaphragm (calculated from the corner positions) is used. In addition, G_p only represents the courier and its velocity controller interface, and there is no G_b gain since the manipulator's θ axis is not being controlled by the servo loop but is instead acting as a disturbance input. The Jacobian used in this experiment was formed by taking the partial derivative of the inverse kinematics of the robot instead of the forward kinematics.

Therefore, this Jacobian did not need to be inverted because it was already in a form suitable for finding the courier and manipulator velocities given image plane velocities. While the x-y visual servoing experiment was performed using this form of the Jacobian, the x-y-θ visual servoing experiment will use the form derived in Sec. 4.6, which is actually the standard derivation for an image Jacobian.

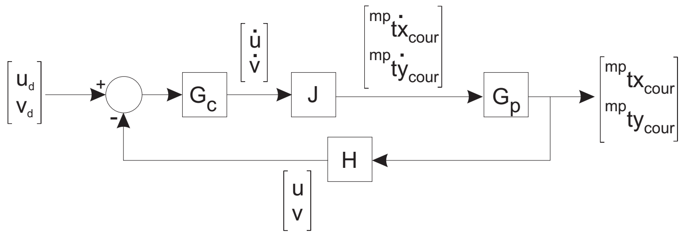


Figure 31: x-y visual servoing model.

The x-y visual servoing experiment was composed of four major steps: automatic calibration of the factory, presentation of a diaphragm to the manipulator, initialization of the visual servoing system, and movement of the manipulator's θ axis as a disturbance input. The factory self-calibration was performed in the same manner as for the look-and-move experiment. After the diaphragm was presented to the manipulator, the vision server acquired an image and searched for the top two corners of the diaphragm to initialize a pair of corner trackers. The third step of the experiment was the initialization of the visual servoing system. To guarantee that the trackers had time to settle, both agents were prevented from moving for 1 s after the vision system began tracking. Figure 32 shows that the desired and measured position of the diaphragm's centroid were different at this point, as measured by the vision system during a typical experiment. While keeping the manipulator fixed, the courier was allowed to move at $t = 1$ s, and as can be seen, the initial error in the position of the diaphragm was quickly accommodated.

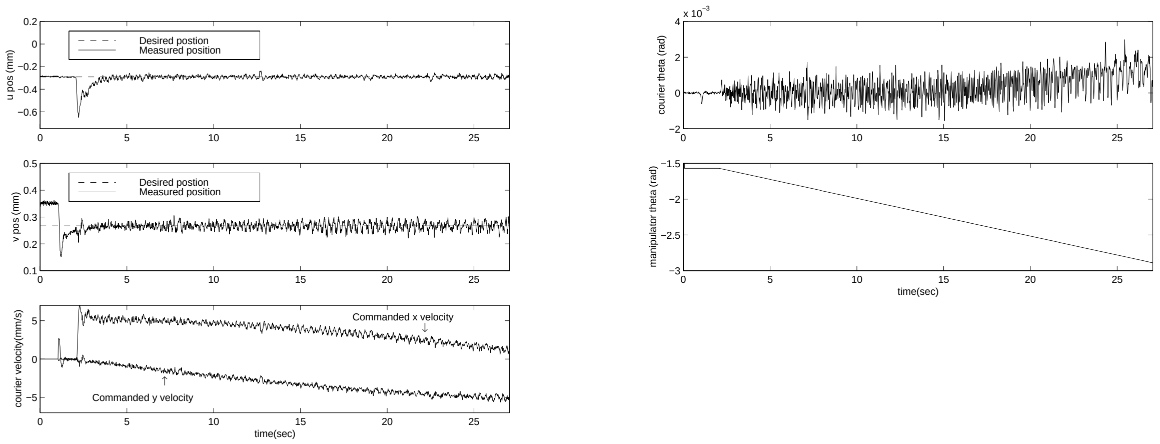


Figure 32: Visual servoing results for $K_p = 7.0$ and $K_i = 0.005$, including measured image plane location of the diaphragm (u and v), commanded courier velocities ($\dot{x}$ and $\dot{y}$), and measured angular position of the courier and manipulator.

After the vision system was initialized, the manipulator's θ axis was rotated clockwise at a constant rate of 0.052 rad/s at $t = 2$ s, while the servoing system attempted to keep the diaphragm's center at the same location in the image. Table 2 shows mean and standard deviation of the visual error signals (both u and v directions) for three different settings of the proportional and integral gains. Note that with the integral term disabled ($K_i = 0$), the resulting u axis error was much greater than the v axis error (note that 1 pixel corresponds to approximately 7 μm on the image plane and 10 μm in 3D camera coordinates through perspective projection). The difference in error magnitude between the axes is a direct result of the camera configuration in the end effector, where the θ axis of the manipulator maps directly into a u-directed motion on the image plane. Setting $K_i = 0.005$ significantly reduced the error during motion, as shown in Fig. 32 and Table 2, at the cost of slightly slower transient performance.

Gains G_c		u error (mm)		v error (mm)	
K_p	K_i	mean	σ	mean	σ
7.0	0.0	-0.286	0.012	0.007	0.013
7.0	0.005	0.000	0.013	0.001	0.014
12.0	0.0	-0.167	0.017	0.005	0.026

Table 2: Steady state image plane position error for the given visual servo controller proportional and integral gains.

The overall results shown in Fig. 32 were encouraging. The manipulator's θ axis rotation caused the courier's tangential velocity to be approximately 5 mm/s, while image plane measurements showed a standard deviation of less than 15 μm in both axes of the vision system. However, the peak-to-peak error was approximately 0.04 mm in the u axis and 0.05 mm in the v axis, corresponding to image movements of nearly 5 pixels. The courier angle plot in Fig. 32 provides one possible cause for this problem. As can be seen, the recorded peak-to-peak motion was approximately 0.002 rad, even though the courier was commanded to hold its orientation throughout this experiment. This "wobbling" is believed to result from miscalibration of the courier position sensor [4] and contributed to most of the error. However, another source of error was from tracker noise, as discussed Sec. 5.3.

A simulation of the x-y visual servoing experiment was performed using the simplified, discrete model of the system shown in Fig. 33.

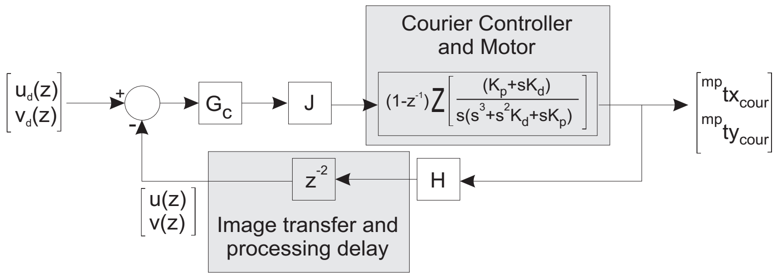

$\begin{bmatrix} u_d(z) \\ v_d(z) \end{bmatrix}$
+
-
G_c
J
Courier Controller and Motor
$(1-z^{-1})Z\left[\frac{(K_p+sK_d)}{s(s^3+s^2K_d+sK_p)}\right]$
$\begin{bmatrix} ^{mp}tx_{cour} \\ ^{mp}ty_{cour} \end{bmatrix}$
H
z^{-2}
Image transfer and processing delay
$\begin{bmatrix} u(z) \\ v(z) \end{bmatrix}$

Figure 33: Discrete model of the x-y visual servo control system.

Figure 34 shows the root-locus plot based on the model in Fig. 33. It shows that instability occurs when the proportional gain in G_c is approximately $K_p = 19$ while the actual experiment showed that a proportional gain of $K_p = 13$ caused marginal stability. The discrepancy can be linked to many simplifications in the system model, but it shows that the important characteristics of the system have been identified (such as vision information delay).

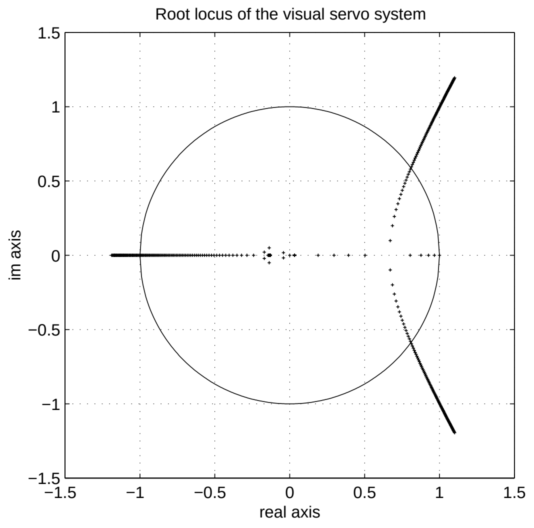


Figure 34: The root locus of the discrete model of the x-y visual servo control system showing marginal stability around $K_p = 19$. The root locus analysis was performed using the Matlab$^{\text{TM}}$ functions $rlocus$ and $rlocfind$.

5.3 Corner Tracker Noise Measurements

The next experiment involved measuring the noise produced by the diaphragm corner trackers, taken from the X Vision library, described in Sec. 4.3.4. The goal was to determine how much noise seen during a visual servoing experiment was actually due to the movement of the agents as opposed to fluctuations in the trackers themselves. To perform the measurement, a similar visual servoing setup as described in the x-y servo experiment was used. However, after the trackers initialized to the diaphragm corners, the agents did not move as the trackers continued to track the corners (which were therefore not moving). Figure 35 shows the u and v positions for both corners.

As seen in Table 3, the noise in both u axes are about 10 μm peak to peak, whereas the v axes have about 20 μm noise peak to peak. One explanation for why the v axes are noisier is because the images are being analyzed in field mode. A field contains either the even lines or odd lines of an image which can cause the vertical starting point for an edge in one field to be different by a pixel in the other field. Given the camera magnification for this setup, a pixel is approximately equal to 7 μm in image plane coordinates (or 10 μm in 3D camera coordinates through perspective projection), which can partially explain the 10 μm noise level difference between the axes.

Another likely source of tracker noise is the pulse width modulation of the LED light source to control the brightness level. This results in some images being more or less illuminated than others, which can affect the edge detection scheme used by X Vision. One possible solution to this problem is to integrate

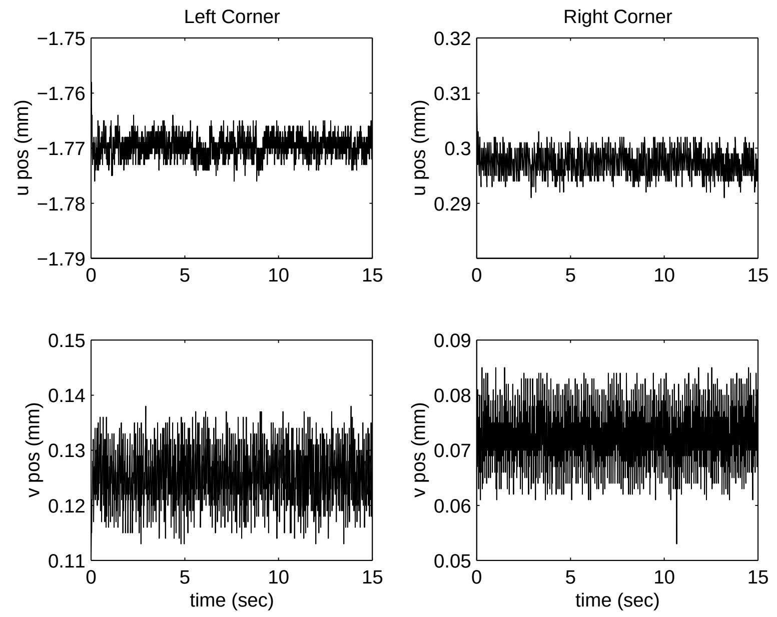


Figure 35: The tracker positions for both non-moving corners.

SSD corner trackers for the diaphragm, which would be more immune to the lighting effects.

5.4 X-Y-Theta Visual Servoing Test

The x-y-θ servoing experiment, like the open-loop look-and-move experiment, attempted to align the position and orientation of an electret with a diaphragm. The experiment was composed of five major steps: automatic calibration of the factory, picking up an electret with the vacuum gripper, presentation of a diaphragm on the courier to 1 mm below the electret, initialization of the visual servoing system, and movement of the manipulator's θ axis and the courier's x and y axes to align the electret and diaphragm. The first three steps are identical to the open-loop look-and-move experiment, but instead of iterative look-and-move corrections, the system uses the visual servo controller presented in Sec. 4.6.

corner	axis	Image plane (mm)		3D Camera Coordinate Frame (mm)	
		σ	peak-to-peak error	σ	peak-to-peak error
left	u	0.002	0.01	0.0028	0.014
left	v	0.006	0.02	0.0084	0.028
right	u	0.002	0.01	0.0028	0.014
right	v	0.006	0.02	0.0084	0.028

Table 3: Standard deviation and peak-to-peak error of the corner trackers when measuring a non-moving target. Image plane measurements are related to 3D camera coordinate measurements through perspective projection.

Figure 36 shows the corner positions, the commanded courier velocities, the commanded manipulator θ velocity, and the manipulator's θ orientation for a typical run. During the first 0.5 s, both the courier and manipulator are prevented from moving to allow the corner trackers time to settle on the diaphragm. After the servoing starts, the controller decides to stop servoing when the u and v image plane error for each corner is less than 5 μm. For the run shown in Fig. 36, the alignment was acceptable after 1.4 s. However, the program was instructed to servo for approximately 5 s more so the additional data could be used to filter out tracker noise while examining the final position errors after the experiment. The final error of a corner was calculated as being the desired position minus the average position for 1 s after the controller deemed the position to be acceptable.

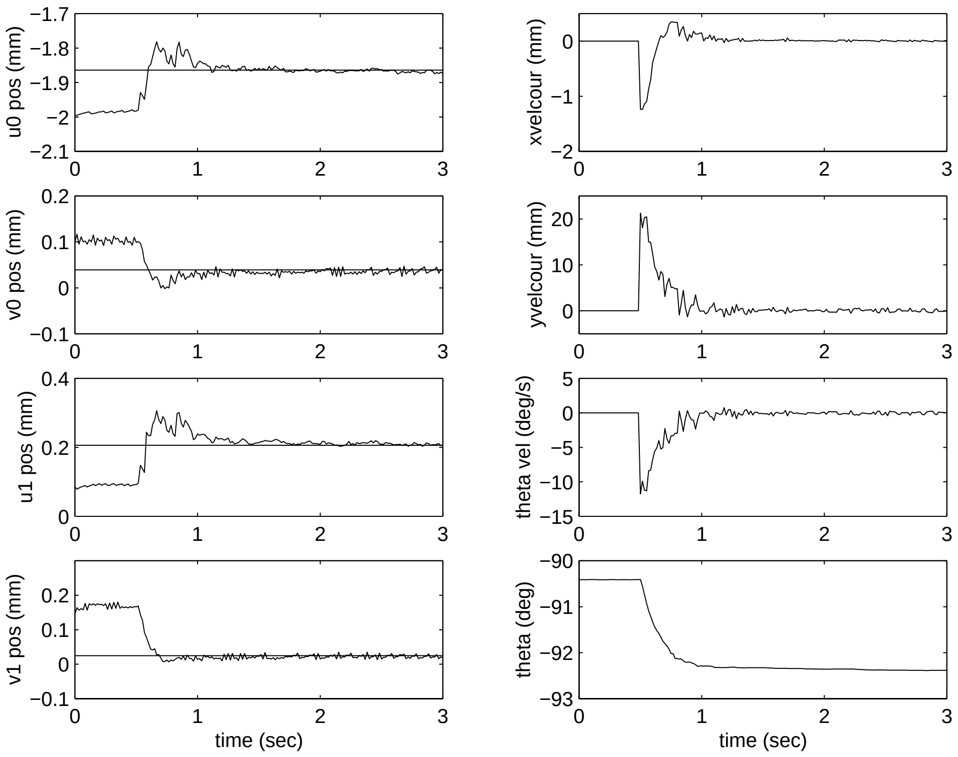


Figure 36: Tracker positions, courier and manipulator velocities, and manipulator θ position during one x-y-θ visual servoing alignment run.

Figure 37 shows histograms of the resulting errors for 100 runs. Note that the u and v errors shown do not represent the corners, but rather the centroid error found when combining information from both corners. This facilitates easier comparison with the look-and-move experiments which were based on centroid errors, not individual corner errors. Table 4 shows the mean and standard deviation of the errors in Fig. 37.

The overall results of the x-y-θ visual servoing experiment were encouraging. However, the system initially tended to oscillate around its goal position when using higher proportional gains in G_c due to the fact that the tracker noise was being amplified. Therefore, a gain scheduler, which linearly adjusted the proportional gain in G_c depending on the size of the corner position errors, was implemented. This limited

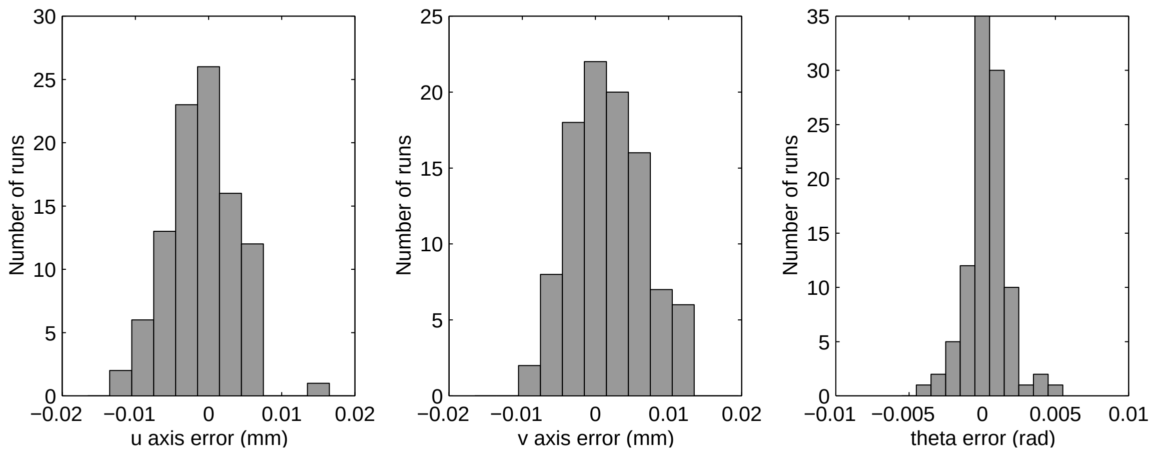


Figure 37: The u and v centroid position error and the orientation error for 100 runs of the x-y-θ visual servoing experiment.

error (mm)	u	v	θ
mean	-0.001	0.0017	0.0003
σ	0.0046	0.005	.0014

Table 4: Mean and standard deviation of the u, v, and θ visual servoing alignment errors.

the effects of the tracker noise when the system was near its goal while still allowing the system to move quickly when far from its goal. Using the gain scheduler with a proportional gain range from 0.3 to 5.0, the average time for the 100 runs was 1.3 s, which is almost three times faster than the average time in the open-loop look-and-move experiment.

It should be noted that the above results measure the performance of the visual servo controller and trackers, not the performance of the electret and diaphragm finding routines. As discussed in the open-loop look-and-move section, there might still be some real absolute error if the electret was not properly detected, since the diaphragm would then be servoed to an incorrect location. For these experiments, only qualitative measurements were done in terms of how well the parts were truly aligned. Figure 38 shows a reasonable alignment of the parts using x-y-θ visual servoing for one run.

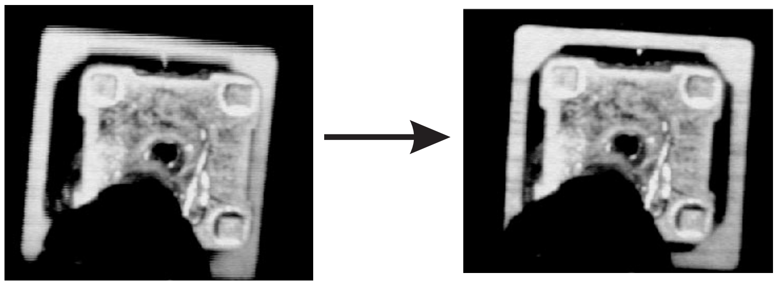


Figure 38: Alignment before and after the x-y-θ visual servoing.

6 Summary and Conclusions

This report summarizes the details of integrating machine vision into the minifactory for use in open-loop look-and-move and visual servoing applications. The major accomplishments are listed below.

- Software was developed to support the Imagenation PX610 frame grabber running on the LynxOS. This included an application programming interface library and the device driver.
- Camera calibration was implemented using Tsai's calibration routine. This required using the minifactory to determine various transforms between coordinate frames as well as acquiring images of a calibration slide sitting on the courier. Routines were written to extract features from the images to use in the Tsai algorithm.
- A vision library was written in C++ to support the analysis of electret and diaphragm parts. The electret analysis used SSD trackers from the X Vision software library. Diaphragm analysis was performed by detecting the edges of the part and using the Hough transform to filter out outlier points. Diaphragm tracking was also supported by using X Vision corner trackers.
- The infrastructure required for visual servoing was implemented by adding additional inter-agent and intra-agent communication channels. Intra-agents communications required setting up shared memory structures between the vision server and manipulator executor. Fast and reliable inter-agent communications were instantiated by accessing the semi-custom AAA network.
- The visual servoing controller was developed to align the position and orientation of a diaphragm with an electret.
- Experiments were performed which utilized the vision library to perform look-and-move iterations to align a diaphragm with an electret. The average time needed was 3.6 s and the average number of iterations was 3.7.
- Visual servoing experiments were performed to align a diaphragm with an electret. Tracker noise was discovered to be approximately 10 μm peak to peak with a standard deviation of 2 μm in the u axis of the image plane and 20 μm peak to peak with a standard deviation of 6 μm in the v axis. The average alignment time was approximately 1.3 s.

The vision project's greatest success was making two 2-DOF agents, each with their own actuators, sensing capabilities, and controllers, transiently cooperate to function as a single 4-DOF robot to perform visual servoing tasks. These were also the first tasks to utilize the AAA network for cooperative behavior. Although only utilizing non-guaranteed data transport, the success of these experiments provided a positive first step toward proving the AAA network's viability for high speed data communications between agents.

Another noteworthy success was the ability of the manipulator to simultaneously acquire and process images at 60 Hz while running a manipulator head process, a vision server process, and a manipulator executor process. This validated our belief that desktop computing power had reached a point where a single computer could effectively run all of these processes at once.

One slightly disappointing finding of this project was the amount of corner tracker noise measured. This unanticipated behavior limited the accuracy of the visual servoing system. However, instead of lowering the overall gain of the system and consequently slowing the error correction, implementing a gain scheduler allowed the alignment task to perform quickly while limiting the effects of the noise.

Although the open-loop look-and-move experiments yielded slower alignment times than the visual servoing experiments, open-loop look-and-move still has its merits including its simplicity of implementation. For example, there is no complicated controller development, the communications infrastructure is simpler, and closed-loop stability is not a concern. However, the open-loop look-and-move iteration times are limited by the agent settling times as well as the efficiency of the diaphragm finding routine. Although it may be possible to decrease the iteration time, movement of other agents may still produce visible vibrations in the image, especially at higher magnifications. In this case, visual servoing will have the advantage. Visual servoing also provides additional benefits over open-loop look-and-move such as being more robust to calibration errors and mechanical inaccuracies.

7 Future Work

The immediate future goals for the integration of vision into the minifactory include decreasing the amount of time needed for open-loop look-and-move iterations by reducing vibrations seen on an image immediately after the agents move. Manipulator vibrations will be reduced by improving the control of its acceleration and deceleration. To reduce courier vibrations, closed-loop control, which has already been integrated into the courier executor process, will soon be accessible to the courier head process. It can then be used by the open-loop look-and-move routines to decrease courier settling time.

Another near-term goal includes decreasing the diaphragm tracker noise. One option is to include an SSD tracker into the current corner tracker routine. The current corner tracker (composed of line primitives) has the advantage of computational speed while the SSD tracker has the advantage of being less susceptible to changes in lighting intensity. When the current corner tracker finds a corner in an image, it can pass its position as a starting point to an SSD tracker which could then quickly refine the corner position. This would provide less of a computational burden on the system than using a purely SSD tracker. A second option to decrease tracker noise is to find an alternative method of controlling the intensity of the LEDs. Instead of PWM, the intensity could be controlled by only turning on a select number of LEDs while being careful of potential shadowing effects.

Short-term experiments will also include tests for the real absolute placement error for both the open-loop look-and-move and the x-y-θ visual servoing experiments. One option for testing the electret and diaphragm alignment tasks would be to first align the parts, and then make the manipulator move its z axis down to place an electret into a diaphragm and hold it there. A small drop of glue could be manually applied to the sub-assembly to bond the parts. After the glue has dried, the manipulator would be raised and the sub-assembly moved to a microscope for measurements. Another option for measuring the accuracy of the system would be to place a diaphragm into a cup, similar to a peg-in-hole task.

Further tests should be performed on the visual servoing controller's ability to cancel vibrations seen by the camera. For example, two manipulators could be mounted on a bridge with one manipulator exercising its z and θ axes while the other is engaged in visual servoing.

Long-term goals for the vision system must include research to eventually allow the vision server to automatically determine which features of an object to analyze or track, freeing the factory programmer from the chores. However, to smoothen the transition into this programming-free environment, it may first be desirable to implement GUI solutions which allow the user to program image analysis tasks using a more intuitive graphical approach.

References

[1] P. M. Muir, J. Gowdy, and A. A. Rizzi, "Minifactory: A precision assembly system that adapts to the product life cycle," in *SPIE Symp. on Intelligent Systems and Advanced Manufacturing*, vol. 3203, (Pittsburgh), pp. 74–80, October 1997.

[2] A. A. Rizzi, J. Gowdy, and R. L. Hollis, "Agile assembly architecture: An agent based approach to modular precision assembly systems," in *Proc. IEEE Int'l Conf. on Robotics and Automation*, pp. 1511–1516, April 1997.

[3] R. L. Hollis and J. Gowdy, "Miniature factories: Tabletop assembly systems for mechatronic products," in *IARP Micro Robots and Systems Conference*, (Beijing), October 20-22 1998.

[4] Z. J. Butler, A. A. Rizzi, and R. L. Hollis, "Precision integrated 3-DOF position sensor for planar linear motors," in *IEEE Int'l. Conf. on Robotics and Automation*, pp. 3109–3114, 1998.

[5] A. E. Quaid and R. L. Hollis, "3-DOF closed-loop control for planar linear motors," in *Proc. IEEE Int'l Conf. on Robotics and Automation*, pp. 2488–2493, May 1998.

[6] W.-C. Ma, "Precision Optical Coordination Sensor for Cooperative 2-DOF Robots," Master's thesis, Carnegie Mellon University, 1998.

[7] W.-C. Ma, A. A. Rizzi, and R. L. Hollis, "Optical coordination sensor for precision cooperating robots," in *IEEE Int'l. Conf. on Robotics and Automation*, May 2000. (accepted for publication).

[8] J. Gowdy and Z. J. Butler, "An integrated interface tool for the architecture for agile assembly," in *IEEE Int'l. Conf. on Robotics and Automation*, pp. 3097–3102, May 1999.

[9] Z. J. Butler, A. A. Rizzi, and R. L. Hollis, "Contact sensor-based coverage of rectilinear environments," in *IEEE Int'l Symposium on Intelligent Control*, 1999.

[10] M. L. Chen, S. Kume, A. A. Rizzi and R. L. Hollis, "Visually guided coordination for distributed precision assembly," in *IEEE Int'l. Conf. on Robotics and Automation*, May 2000. (accepted for publication).

[11] J. Gowdy and A. A. Rizzi, "Programming in the architecture for agile assembly," in *IEEE Int'l. Conf. on Robotics and Automation*, pp. 3103–3108, May 1999.

[12] M. Lutz and D. Ascher, *Learning Python*. O'Reilly & Associates, Inc., 1999.

[13] P. Corke, *Visual Control of Robots: high-performance visual servoing*. John Wiley & Sons Inc., 1996.

[14] S. Hutchinson, G. D. Hager, and P. I. Corke, "A tutorial on visual servo control," *IEEE Transactions on Robotics and Automation*, vol. 12, pp. 651–70, Oct. 1996.

[15] T. Barron, "Eye on machine vision," *Robotics World*, vol. 16, pp. 30–34, summer 1998.

[16] K. Hashimoto, T. Kimoto, T. Ebine, and H. Kimura, "Manipulator control with image-based visual servo," in *Proc. IEEE Int'l Conf. on Robotics and Automation*, pp. 2267–2272, 1991.

[17] Edmund Industrial Optics, "1999 optics and optical instruments catalog." pp. 170.

[18] G. Marshall, "Choosing a frame grabber for performance and profitability." Imagenation Corporation, Available at http://www.imagenation.com/pxc/frame.htm.

[19] *Writing Device Drivers for LynxOS*. LynxOS Version 2.5.

[20] R. Y. Tsai, "An efficient and accurate camera calibration technique for 3D machine vision," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 364–374, 1986.

[21] R. Wilson, "Implementation of Tsai's camera calibration method." Available at http://www.cs.cmu.edu/∼rgw/Tsai-method-v3.0b3.tar.Z.

[22] G. D. Hager and K. Toyama, "X Vision: a portable substrate for real-time vision applications," *Computer Vision and Image Understanding*, vol. 69, pp. 23–37, Jan. 1998.

[23] R. Jain, R. Kasturi, and B. G. Schunck, *Machine Vision*. McGraw-Hill, Inc., 1995.