ABSTRACT

AUTOMATED LIQUID DISPENSING PIN FOR DNA MICROARRAY APPLICATIONS

by
Suganya Parthasarathy

This thesis describes the research and development of a new liquid dispensing/aspiring system that is capable of producing micro sized spots/droplets for molecular biology research and analysis. In particular, the application is focused on DNA microarray fabrication with the goals of smaller spot size, higher yield, more efficient usage of biological materials, and capability to handle high viscosity liquids. The new system is based on active sensing and control and it is part of a fully integrated robotic microarray system for genomic and proteomic applications. The prototype system handles water as well as thick liquids such as 100% glycerol and generates spots in a contactless manner with controllable spot size ranging from 80 microns to 200 microns. Microarray technology is enhancing many areas of biological research including stem cell, cancer and infectious disease research. This new method of microarray production will afford hospitals and laboratories the system necessary to help detect and study genetic changes in cells in a more efficient and cost effective manner.

AUTOMATED LIQUID DISPENSING PIN FOR DNA MICROARRAY APPLICATIONS

by
Suganya Parthasarathy

A Thesis
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Electrical Engineering

Department of Electrical and Computer Engineering

August 2005

APPROVAL PAGE

AUTOMATED LIQUID DISPENSING PIN FOR DNA MICROARRAY APPLICATIONS

Suganya Parthasarathy

[illegible]

Dr. Timothy N. Chang, Thesis Advisor — Date
Associate Professor of Electrical and Computer Engineering, NJIT

[illegible]

Dr. Patricia Soteropoulos, Committee Member — Date
Managing Director, Center for Applied Genomics, PHRI

Dr. Sui-hoi E Hou, Committee Member — Date
Associate Professor of Electrical and Computer Engineering, NJIT

BIOGRAPHICAL SKETCH

Author:	Suganya Parthasarathy
Degree:	Master of Science
Date:	August 2005

Undergraduate and Graduate Education:

- Master of Science in Electrical Engineering,
  New Jersey Institute of Technology, Newark, NJ, 2005

- Bachelor of Engineering in Electronics and Instrumentation Engineering,
  J.J. College of Engineering and Technology, Trichy, India, 2003

Major: Electrical Engineering

Presentations and Publications:

Chang, T.N., Parthasarathy, S., Wang, T., Gandhi, K., Soteropoulos, P., "Automated Liquid Dispensing Pin for DNA Micro-array Applications", to appear in the IEEE Transactions on Automation Science and Engineering.

To my family for their unconditional love, guidance, encouragement and support,
To JILLU PERI.

ACKNOWLEDGEMENT

I would like to express my deepest gratefulness and respect to Dr. Timothy N. Chang, who not only served as my thesis advisor, providing valuable and countless resources, insight, and intuition, but also constantly gave me support, encouragement, and reassurance. Special thanks are given to Dr. Patricia Soteropoulos and Dr. Sui-hoi E Hou for actively participating in my committee.

I owe much to the members of Public Health Research Institute (PHRI). Sincere thanks go to Dr. Patricia Soteropoulos, Mr. Tongsheng Wang, Dr. Salvatore A.E. Marras and Dr. Sanjay Tyagi for their invaluable help. I appreciate having had the chance to get to know them. I am thankful to the National Science Foundation Grant 0243302, "High resolution, high density microarrayer for genetic research" for partially supporting my work.

Many of my senior colleagues in the Real Time Controls Laboratory are deserving of recognition for their support. I would like to thank my colleagues Biao Chen, Puttiphong Jaroonsiriphan, Yuan Ding, Qiong Shen, Paiboon Sriwilai Jaroen and Kunj Gandhi for their immense help during practical difficulties. Also, many thanks to Ms. Brenda Walker and the entire staff of Electrical and Computer Engineering Department at NJIT. Last, but not the least, I would like to thank Aravind Parthasarathy and Sowmya Vedhartham Sampath – all your understanding and support saw me through difficult times.

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES
(Continued)

LIST OF FIGURES
(Continued)

CHAPTER 1

INTRODUCTION

Genomic research is undergoing an industrial revolution with automation and high speed computation being two of the enabling factors responsible for the explosive growth in sequencing efforts reducing the cost from \$5 per finished base to about \$0.10 in just ten years. The next phase, gene expression profiling, is equally significant, especially in clinical studies of genetic diseases such as cancer. The basic steps of determining gene expression consist of 1. microarray fabrication, 2. hybridization, and 3. detection.

DNA chips, also known as microarrays, present an efficient way to manipulate the huge numbers of reagents that would be required to probe for a large collection of genes and this technology has become a central platform for functional genomics [1]. These bio-devices are dense grids of DNA bound to a solid matrix that can be probed with a complex mixture of labeled DNA or cDNAs. The major advantages of microarrays over other technology are the increase in the number of genes being analyzed, the substantial reduction in sample size requirements, and the use of fluorescence detection schemes for high signal to noise ratios.

Among the many applications for DNA microarrays are:

- Expression profiling of human cancers
- Identification of diagnostic signature transcriptional profiles for host responses to infectious agents
- Gene expression profiling following injury
- Expression profiling following drug treatment

Currently, there are two major approaches for the creating DNA microarrays: photochemical process and printing process. The photochemical process pioneered by Affymetrix [19] has a high inherent degree of reproducibility due to the spatial specificity of the manufacturing process. However, this approach tends to be inflexible and expensive, thus excluding most small/medium scale laboratories from conducting innovative research. The conventional printing technique (impact or inkjet) as shown in Figure 1.1, on the other hand, is flexible and less expensive. For the impact printing method, a hollow pin draws up DNA sample fluid and prints a series of small dots on a glass slide as shown in Figure 1.2.

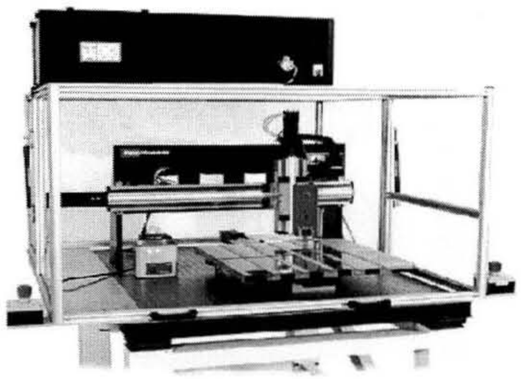
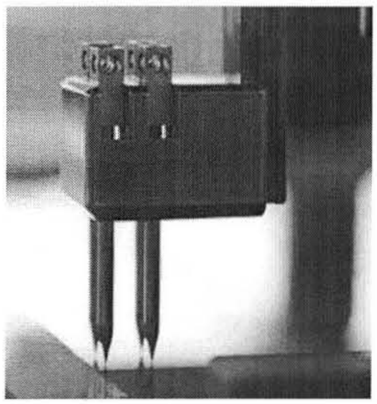



Figure 1.1 Microarrayer with impact printing technology (left) and contact printing pin (right) [18].

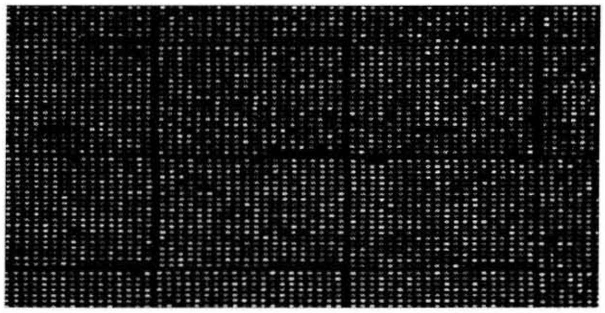


Figure 1.2 Image of a section of a Microarray Experiment using the Center for Applied Genomics (CAG) Human 19K DNA oligo array. The chip was hybridized with Cy3- and Cy5-labeled RNAs prepared from human liver and kidney, respectively.

However, minimum spot size and reliability remain key limiting factors that impede the competitiveness of the printing technique. Presently, a standard microscope slide can hold up to 40,000 spots. To accommodate the entire human transcriptome, this density must be increased to well over 120,000 spots per slide implying that the current spot size of 100 microns must be decreased accordingly. Furthermore, both impact and inkjet printing methods are open loop and require careful calibration for each run where uneven spot formations are not uncommon. A scan of a defective print is shown in Figure 1.3 where blurred and missing spots are present.

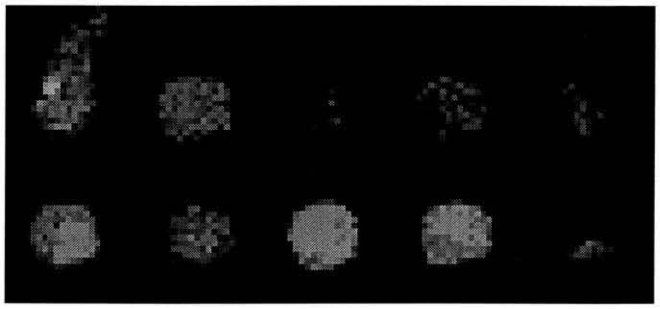


Figure 1.3 Defective spot formation.

Furthermore, the printing technique generally deposits only about 30% of the liquid material that is drawn into the pins by capillary flow during each 100 chip print session. The material that remains in the pins is lost when the pins are washed. This waste of valuable resources significantly increases the cost of chip production. Present fabrication cycle time with a 48-pins and 40x384 well microplates (15360 spots) is about 8 hours which must be significantly reduced. Finally, due to the design of the impact pins, they tend to get clogged by viscous liquids such as those containing glycerol which is useful in proteomics.

Existing drop generation mechanisms can be categorized into contact printing with pins (quill, solid, or pin-and-ring) or non-contact printing with microsolenoid, thermal ink jet, piezoelectric, or acoustic printing heads. Due to the impact at contact, pin structure deformation, and clogging from contaminants collected at contact, pin-based contact printing is prone to suffer from slide-to-slide inconsistency. A novel contact-printing device is the fabrication of a micromachined silicon pin. MEMS technology offers advantages such as freedom of pin and slot design both in size and shape, possibility silicon pin can pick up 1ul of DNA solution. One of the big challenges of droplet size control is preprinting. It has been reported that after loading sample into microchannel, the commercial pins must be spotted a number of times onto a slide to create consistent spots due to the adherence of additional DNA solution to its hydrophilic surface outside the slit. Silicon microarray pin also has a similar problem but to a lesser degree. And it also needs to be packaged in a way that can fit into a pin holder on a commercial arrayer. A molding process, combined with silicon micromaching is needed to manufacture the pin and cap combination [21]. Another contact printing technique would be the quill-pin technique, though it is optimized as a function of substrate wettability and composition of the buffer to improve microarray density it still suffers the other draw backs of the contact process [22].

The commercial non-contact arrayers have lower numbers of print heads per rack than pin-based arrayers. Part of the reason is due to the complexity of plumbing required with their designs and the inherent tendency for clogging [20]. The non-contact piezoelectric systems uses piezo ceramics located adjacent to the fluid near each nozzle. A tiny electric charge is forced into the piezo ceramic, causing it to change shape and displace fluid. When the piezo ceramic displaces in and out, it shoots a droplet of ink out of the nozzle. Piezo ink jet has advantages of small drop sizes and high-ejection rate. In this method though the cell solutions were successfully ejected until reservoir depletion, with no observed adverse effects or clumping, one potential problem was the inhomogeneous cell concentration in the ejected fluids, possibly due to cell settling within the reservoirs [20].

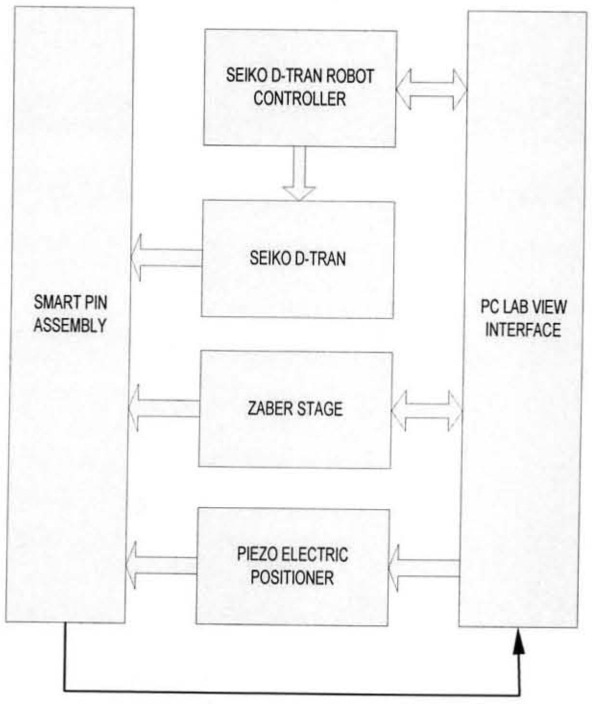
Another printing method using hollow cylindrical ceramic tips improves the morphology of microarray elements, allows higher element density and increases printing tip life over the customary slotted stainless-steel pins. But implementation of ceramic tips requires custom tooling, consisting of a modified tip holder, printing pin assembly, and air reservoir. Moreover the ceramic tips are more sensitive to inaccuracies in leveling of the printing tip holder to the arrayer slide platter. As a result, the tip holder must be carefully aligned to prevent an increase in deposition failure. Another issue with ceramic tips is they are more difficult to dry after cleaning, and if they are not completely dry, the next DNA samples do not aspirate completely [23].

CHAPTER 2

HARDWARE AND SOFTWAR DESCRIPTION

2.1 Overall Hardware Description

In this work, a smart-pin liquid dispensing system is designed and constructed to integrate a number of critical tasks: spot formation via active sensing and control, fine positioning, and spot characterization. The SmartPin combines sensing, actuation, and feedback control: it is capable of regulating spot size and providing robust, non-contact delivery at a high speed.



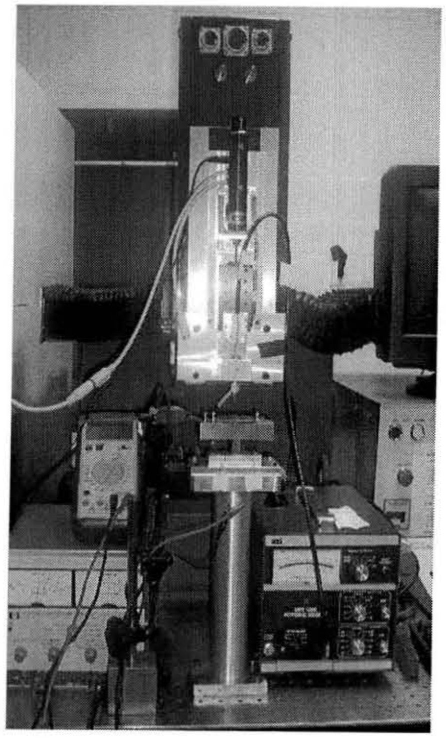


Figure 2.1 Prototype microarrayer with SmartPin.

The prototype system is shown in Figure 2.1 where a Seiko D-Tran robot serves as the coarse platform. The Seiko D-Tran robot communicates with a PC via LabVIEW driver developed in the lab. The product specifications and their definitions are given below:

- **Resolution:** The resolution (or addressability) is the distance equivalent to the smallest incremental move the device can be instructed to make. In other words, it is the linear or rotational displacement corresponding to a single microstep of movement.

- **Repeatability:** The repeatability is the maximum deviation in the position of the device when attempting to return to a position after moving to a different position.

- **Accuracy:** The accuracy is the maximum deviation of the actual position of the device from the requested position over the full range of motion.

- **Backlash:** As was seen in the repeatability section, backlash is the deviation of the final position that results from reversing the direction of approach.

The Seiko D-Tran robot has a +/- 10 micron repeatability on all three axes. The positioning accuracy is augmented by a piezoelectric positioner with a +/- 15 micron range and better than 0.1micron repeatability. The Zaber Stage, with a +/- 8 micron accuracy and +/- 0.3 micron repeatability, drives a fiber probe within the SmartPin assembly.

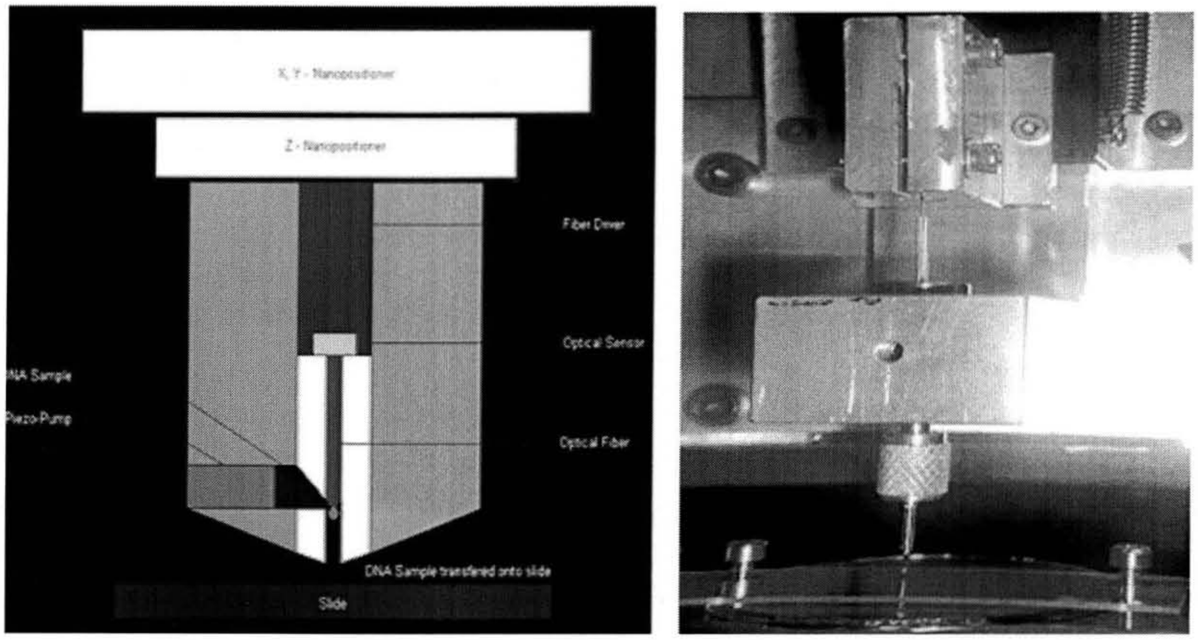


Figure 2.2 SmartPin cross-section view and prototype.

As shown in Figure 2.2, the SmartPin assembly comprises of the following subcomponents:

1. A fluid reservoir containing the sample liquid materials. This reservoir can be the interior of the pin cavity or a pressurized fluid chamber connected to the cavity.

2. A fluid delivery plunger made with an optical fiber probe and a driver. The fiber-driver is a precision lead-screw assembly along the Z-axis as shown in Figure 2.2. In the present version, the optical fiber bundle has a diameter of 150 microns. When the fiber is moved downwards, a metered amount will be dispensed onto the tip of the pin. Similarly, aspiration can be accommodated with upward movement of the probe; the holding volume is about $0.2\,L_z$ nanoliter per micron where L_z is the vertical displacement in microns. For a 2cm vertical cavity, the liquid storage volume is about 4 microliters.

3. The fiber probe also determines the distance between the fiber tip and the slide. The optical sensing scheme utilizes the fiber itself as an optical lever. There are two ways the optical sensors can be realized: active and semi-active. In the active method, a controlled light source generates a collimated beam directed towards the fiber tip and slide. The reflected light is collected through a photodetector. The position information is inferred ratiometrically from the intensity of the reflected light by a digital signal processor. For the semi-active method, the glass slide is uniformly illuminated from below. The optical fiber picks up the light beam and transmits it to a photodetector.

In addition to the optical lever, the SmartPin also utilized the total internal reflection principle: due to the difference in refractive indices of air, glass (pin), sample material, and the shape of the material, contact condition can be derived by continuously monitoring the sensor intensity.

2.2 Description of SEIKO D-Tran Robot

The Seiko D-Tran Robot is four axis Cartesian robot. Other robots in the family of Seiko D-Tran are RT *Cylindrical Coordinates,* TT *Multi Articulated SCARA Robots.* The entire robot assembly consists of three visible parts as seen in Figure 2.3.

1. Seiko Robot Manipulator
2. Controller
3. Key Board Console

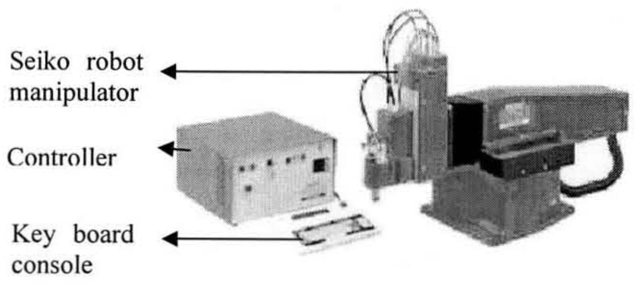


Figure 2.3 SEIKO D-Tran Robot.

The Seiko D-Tran robot offers high precision, repeatability and speed. The Seiko D-Tran robot is modularly constructed and its high precision accuracy, repeatability, and speed make applications with close tolerances possible.

2.2.1 Seiko Robot Manipulator

The Seiko Robot has four manipulators that are based on closed loop DC servo motors, which provide different motion in the robot envelope. The A or the Alpha Axis, allows rotation for the End –Effecter. It can be mounted with the Tool Fang downward or upward. The X-Axis allows front-back motion in the X plane. The Y-Axis allows side ways motion. The up-down motion is provided by the Z-Axis.

2.2.2 Controller

The Robot comes with a well equipped controller which is easy to operate and has an emergency stop button and a time counter that counts the hours of operation. Its basic components consist of a Z80 Microprocessor and DC servo amplifiers. The Seiko Controller supports the DARL robot language which may be programmed via a RS-232 link on a PC or the Tech-Terminal. The Tech-Terminal is ideal for altering the program in an industrial arena and useful for trouble shooting and maintenance. The output unit is a Liquid Crystal Display (LCD), with 40 characters per line, and 4 lines per view. More features of the Tech-Terminal can be found in the user's manual provided by the manufacturer. The Seiko D-Tran controller has two RS–232 ports for serial communication with any device.

2.3 Description of Optical Fiber Displacement Sensor (Fotonic Sensor)

The Fotonic sensor as shown in Figure 2.4 consists of a console and a sensor module. The console processes sensor signal and sensitivity while the probe consists of light-transmitting fibers and light-receiving fibers that are bundled together, realizes the optical lever principle. Displacement measurement is based on the interaction between the field of illumination of the transmitting fibers and the field of view of the receiving fibers. As the probe to target distance decreases, increasing amounts of light are captured by the receiving fibers. This relationship will continue until the entire face of the receiving fiber is illuminated with reflected light. This point is called the optical peak as shown in Figure 2.5. Further decrease in gap distance actually results in reduced intensity measurement corresponding to the "front slope" region as shown in Figure 2.5. At contact, or zero gap, most of the light exiting the transmitting fibers is reflected directly back into those fibers. No light is provided to the receiving fibers and the output signal is "zero" and the primary characteristics are given in the Appendix A section 4.

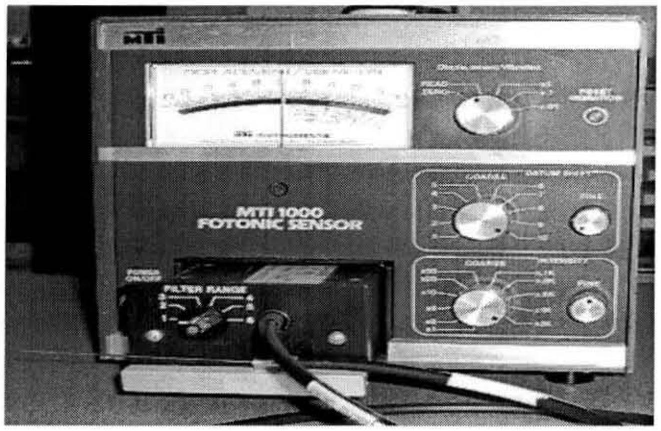


Figure2.4 MTI 1000 Fotonic Sensor with a MTI-3802 sensor module.

According to the manufacturer's specifications [14], a major advantage of the Fotonic sensor is its ability to operate directly with a large variety of surfaces, from specular to diffuse, and materials from conductors to insulators. As shown in Figure 2.5, the gap and displacement range over which the initial rise in signal takes place and at which the maximum occurs is primarily determined by the diameter and the numerical aperture of the fibers and the intensity distribution within the operating field of the fibers. Most commercial devices of this type use multiple transmit and receive fibers in order to obtain the higher levels of intensity at the photo detectors needed to insure acceptable levels of performance.

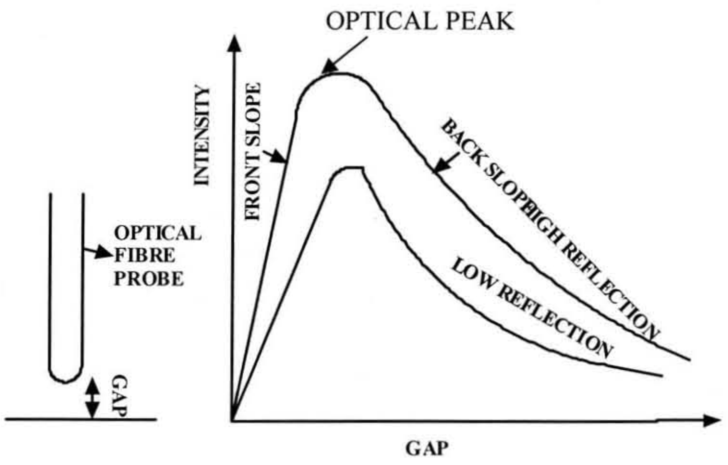


Figure 2.5 Receiver illumination and instrument output.

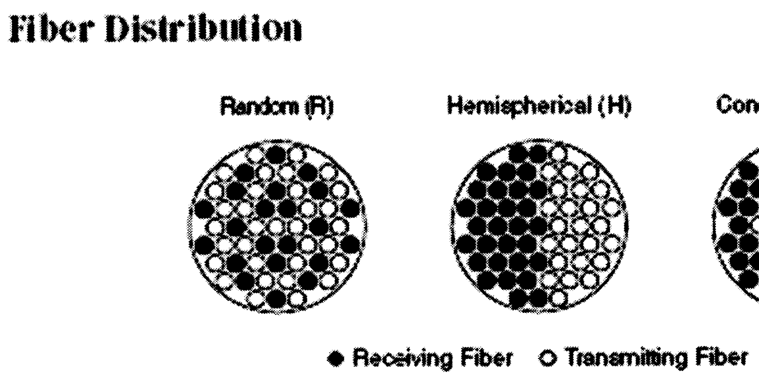


Figure 2.6 Fiber distribution in the sensor probe. [14]

Each MTI-1000 Fotonic probe contains a set of light transmitting and light receiving fibers, which can be arranged in three different configurations (random, hemispherical or concentric) as shown in Figure 2.6. As shown in Figure 2.7 a tungsten halogen lamp acts as the light source and feeds light down the transmit fibers, where it exits the probe tip and hits the target. Light that is reflected from the target is captured by the receive fibers and processed by the console. The light intensity is monitored, which is proportional to the distance between the probe tip and the target being measured.

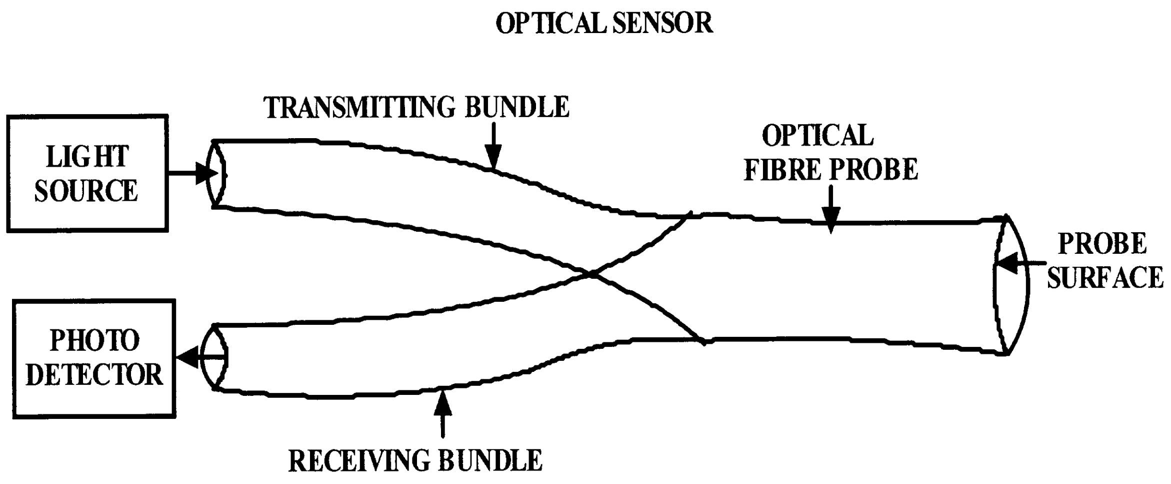


Figure 2.7 The optical lever principle realized on a bundled fiber probe.

Referring to the Figure 2.5, the front side of the curve is defined as the operating area between sensor probe-to-target contact and the optical peak, and it is characterized by a positive slope and higher sensitivity on the response curve. The front side slope is plotted on the most linear rise of the curve's front side and indicates the sensor's front side operating range. The back side of the response curve is the area from the optical peak on out to large gaps. The back side slope is the most linear portion of the back side of the curve and indicates the sensor's back side operating range.

Calibration Procedure of the Fotonic sensor:

1. Perform instrument warm-up for 20 minutes and electrical zeroing.

2. Insert the probe into a notch of the fixture attached to the robot arm. Position the probe by jogging the robot so that the probe tip lightly contacts the target mirror surface and set the zero gap.

3. Tighten the probe clamp evenly and ensure that the probe remains aligned with the target and centered in the notch.

4. Check the panel meter, the indicator should read "0".

5. Turn the displacement/vibration control to the "READ" position. Check the panel indicator; it should remain at "0" when switching to the "READ" position, but actually the panel indicator has a different intensity read out due to the polished surface being at the underside of the target mirror used. Now since the probe does not touch the polished surface of the mirror, the variation from true zero read out is observed.

6. Move the robot upwards and observe the panel indicator. The indicator will move up scale as the target moves away from the sensor probe. The indicator will stop moving when the optical peak is reached.

7. Turn the intensity fine control clockwise to move the indicator to the "set/cal" mark on the scale. If the intensity fine control reaches full stop and the indicator has not reached the "set/cal" mark, then the light intensity of the probe must be increased by changing the intensity coarse control setting.

8. Turn the intensity coarse control to the next higher switch settings. Then turn the intensity fine control clockwise to bring the indicator to the "set/cal" zone.

9. Move the robot downwards to decrease the gap and move into the front slope operating range. Observe the indicator, it will move down scale of the "set/cal" zone as the target moves towards the sensor probe. Continue to move the robot until the indicator is positioned in the "operate" zone on the scale. The gap is now calibrated to the front side slope on the calibration curve. The plot of the gap distance from the mirror to the intensity measure read out on the panel gives the calibration curve for the Fotonic sensor as shown in Figure 2.8.

10. If the back slope area is to be used to obtain more range or standoff distance, then the robot must be moved upwards. This will increase the gap and move the probe into the back side operating area.

11. While calibrating the plug in module MTI-3802 using instrument MTI-1000 the settings used were 2K maximum intensity and datum reference 10. The target being referred to is a polished plane mirror. The front side slope (sensitivity of the Fotonic sensor) obtained from the slope curve as shown in Figure 2.8 is 0.05785um/mv and the back side slope is -0.1034 um/mv.

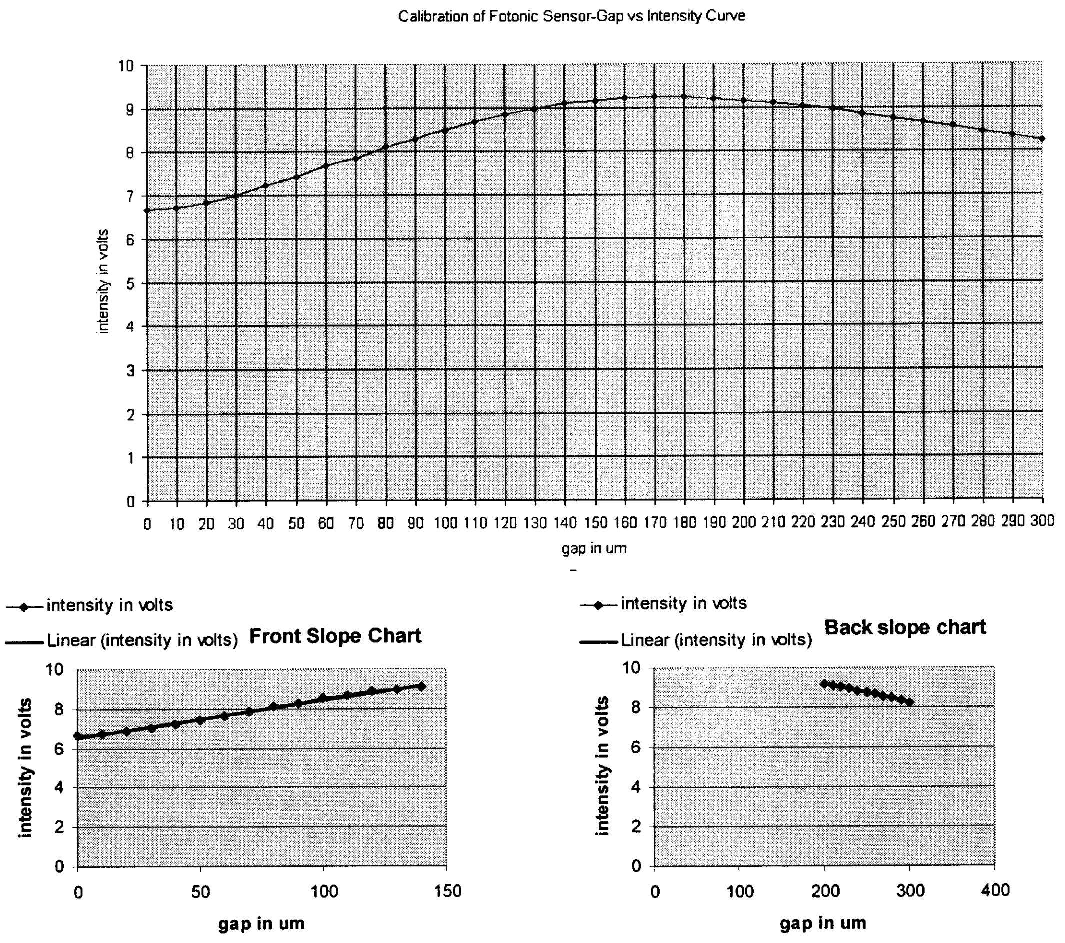


Figure 2.8 Calibration of Fotonic sensor (up) and front side and back side slope charts of Fotonic sensor (down).

To implement this system as a part of the hardware construction light intensity is taken as a reference, the output data will be used to measure gap distances between the sample liquid and the sensor's probe. In turn, this data will allow the user to determine the spot formation and uniformity of the spots.

2.4 Description of Zaber Stage

Zaber Technologies Inc series of computer controlled positioning products as shown in Figure 2.9 use stepper motors and leadscrews to achieve open loop position control. The product specifications are given in Table 2.1. The stepper motor turns by a constant angle called a step for every electrical impulse sent to it. This allows a system to be built without feedback, reducing total system cost. Zaber positioning devices are driven by stepper motors using a micro-stepping drive with 64 microsteps per step. All position data sent to or received from Zaber devices must be in units of microsteps (the software converts position data entered by the user to microsteps before sending it to the device).

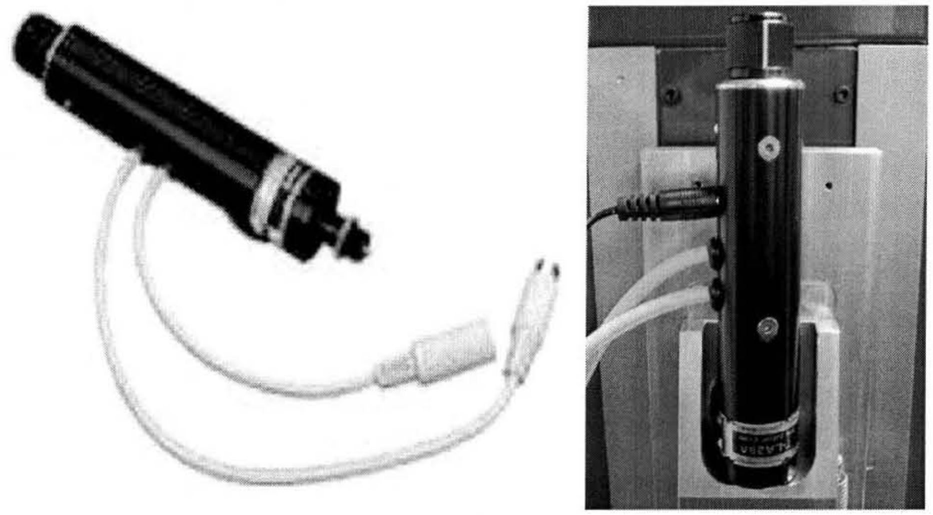


Figure 2.9 T-LA Series – Zaber linear actuators.

Table 2.1 Product Specifications of Zaber Stage

Part Number	**Range**	**Resolution**	**Repeatability**	**Cyclic Accuracy**	**Backlash**
T-LA28A	28mm	0.1um	+/- 0.3um	+/- 8um	2um

The control is through the RS232 port. The communications settings must be: 9600 baud, no hand shaking, no parity, one stop bit. The amber LED will light when there is activity on the RS232 lines. After power-up, the units in the chain will each initialize themselves as unit number 1 and thus they will each execute the same instructions. To assign each unit a unique identifier a renumber instruction as specified in Table 2.3 must be issued after all the units in the chain are powered up and every time an unit is added or removed from the chain. All instructions consist of a group of 6 bytes. They must be transmitted with less than 10 ms between each byte. If the unit has received less than 6 bytes and then a period of more than 10 ms passes, it ignores the bytes already received. It is recommended that the software behaves similarly when receiving data from the devices, especially in a noisy environment. The following Table 2.2 shows the instruction format: The first byte is the unit number in the chain. Unit number 1 is the unit closest to the computer; unit number 2 is next and so forth. If the number 0 is used, all the units in the chain will execute the accompanying command simultaneously. The second byte is the command number. Bytes 3, 4, 5, and 6 are data in long integer, 2's complement format with the least significant byte transmitted first. How the data bytes are interpreted depends on the command. The two "move" commands are tabulated in Table 2.3.

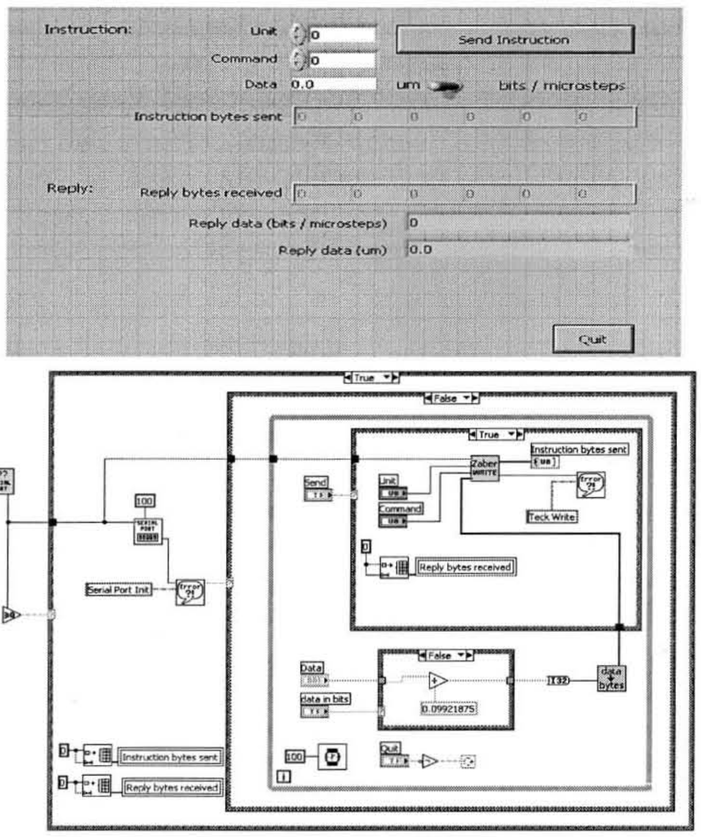
Table 2.2 Instruction Format of Zaber Stage

Byte1	Byte2	Byte3	Byte4	Byte5	Byte6
Unit number	Command number	Data (least significant byte)	Data	Data	Data (most significant byte)

Table 2.3 Command Reference of Zaber Stage [17]

Command	Description	Data bytes	Reply data
20	Move absolute. The device moves to the position given by the data bytes. The position must be within the acceptable range for the device.	Absolute position in micro-steps.	Absolute position. If the data is out of range the device will not move but will return 255 in byte 2 as well as the absolute position.
21	Move relative. The device moves to the position given by its position before the command plus the value in the data bytes. The final position must be within the acceptable range for the device.	Relative position (can be negative) in Micro-steps.	Absolute position. If the data is out of range the device will not move but will return 255 in byte 2 as well as the absolute position.
2	Renumber. This command must always be issued with a 0 in Byte 1 (i.e. it must be issued to all units simultaneously).	Ignored.	Each unit replies with its device ID after it finishes renumbering.

As shown in the front panel of Figure 2.10 the instruction is being transmitted to the Zaber stage. The format of the instruction is a group of 6 bytes as explained before. Command reference is issued as per the required Zaber movement, it can either be relative or absolute. The data field specifies how far the leadscrew driven by the stepper motors should move. For the experiments carried out, the Zaber is moved down to the maximum limit of 20,000um to fit the needle and then to draw the sample the Zaber stage is moved upwards to about 1000um relative from maximum limit making the absolute position as 19,000um to make room for the sample to enter the pin. The reply data in um indicates the present position of the Zaber stage.



Instruction: Unit 0
Command 0
Data 0.0 um bits / microsteps
Send Instruction
Instruction bytes sent 0 0 0 0 0 0

Reply: Reply bytes received 0 0 0 0 0 0
Reply data (bits / microsteps) 0
Reply data (um) 0.0
Quit

Figure 2.10 Front Panel and LabView program for Zaber stage.

The mechanical assembly in the practical model is a sliding bar with springs attached that roles up and connects the Zaber stage, Seiko robot and the smart pin stage together as shown in Figure 2.11.

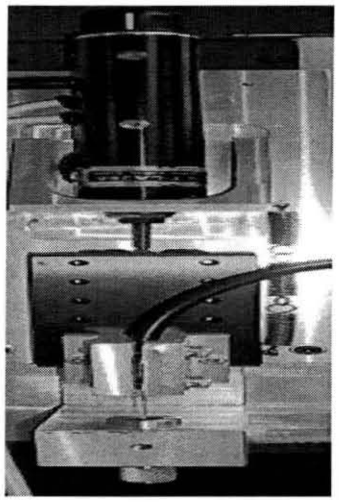


Figure 2.11 Mechanical integration of Zaber.

CHAPTER 3

METHODOLOGY AND CONTROLLER DESCRIPTION

The smart pin consists of a glass tube pointed at the end resembling a pin. The Fotonic sensor of diameter 504um driven by the Zaber stage is placed inside the glass tubing such that it can move freely but compact enough. The robotic arm holds the above arrangement. The robot is commanded to move to a well which contains the test liquid. The pin is immersed into the well and the Zaber stage is withdrawn back making the Fotonic sensor probe act as a plunger and takes in metered quantity of test sample. After cleaning the side walls of the pin, the robot is moved 120um above the glass slide and the Zaber stage is pushed by 20um so that the Fotonic sensor probe plunges a metered quantity of sample at the tip of the pin. Now the robot is jogged 10um each time until a sharp decrease in the intensity of the sensor is noted indicating the formation of spot. This process is a non-contact method as the robot stops jogging further when the liquid on the pin tip contacts the slide surface. In this method as shown in Figure 3.1 there are four stages namely Pull down, Contact, Pull up elongation and Pull up separation. The pin travels down with the sample droplet at the tip and makes a contact with the slide when it is approximately 10um above the slide and then the robot is reversed back during which the droplet tends to stick to the needle and elongates as a column before it detaches fully from the pin. Some part of the droplet is left behind in the pin. Analysis has been done with different liquids with varying viscosities.

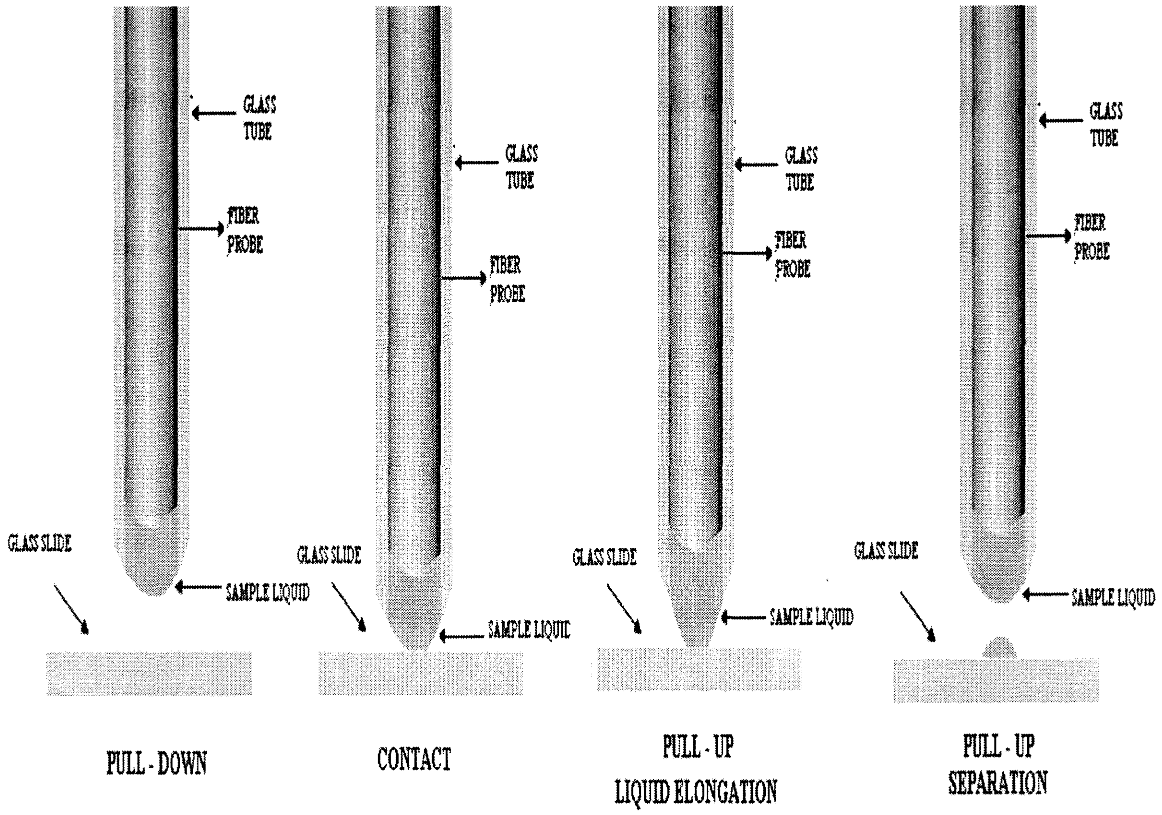


Figure 3.1 Spotting sequence.

The factors that affect the spotting include the following

1. Pin size and shape: The size and the shape of the glass pin are major factors affecting the spot formation. Finely cut and polished borosilicate pin of outer diameter 1.35mm and inner diameter 0.59mm must be chosen to fit around the Fotonic sensor driven by the Zaber stage. The geometry of the pin tip affects the intensity pick up drastically. Uneven ends may cause disturbances in the voltage pick up and affect the spotting. The size and shape of the spots will be irregular and would not be repeatable.

2. Viscosity of samples: The types of sample used have different viscosities and hence the filling of the pin without air bubbles also becomes an issue for flawless spotting.

3. Zaber increment: The spot size drastically depends on the Zaber push given to release a metered quantity of the sample at the pin tip. If the Zaber increment is more, then naturally the spots formed are bigger in size.

4. Air gap control: The size of the spots can be controlled by positioning the pin properly over the air gap distance. Even with larger pin openings smaller size spots can be produced, making spotting independent of pin geometry.

5. Intensity enhancement: The voltage pattern observed can have better resolution with more intensity pick up and this can be facilitated by proper sample level filling in the pin by the Zaber stage. The Zaber has to be pulled just by 1000um above the maximum limit. By doing so there will be sufficient sample for spotting and there will be more intensity pick up by the sensor.

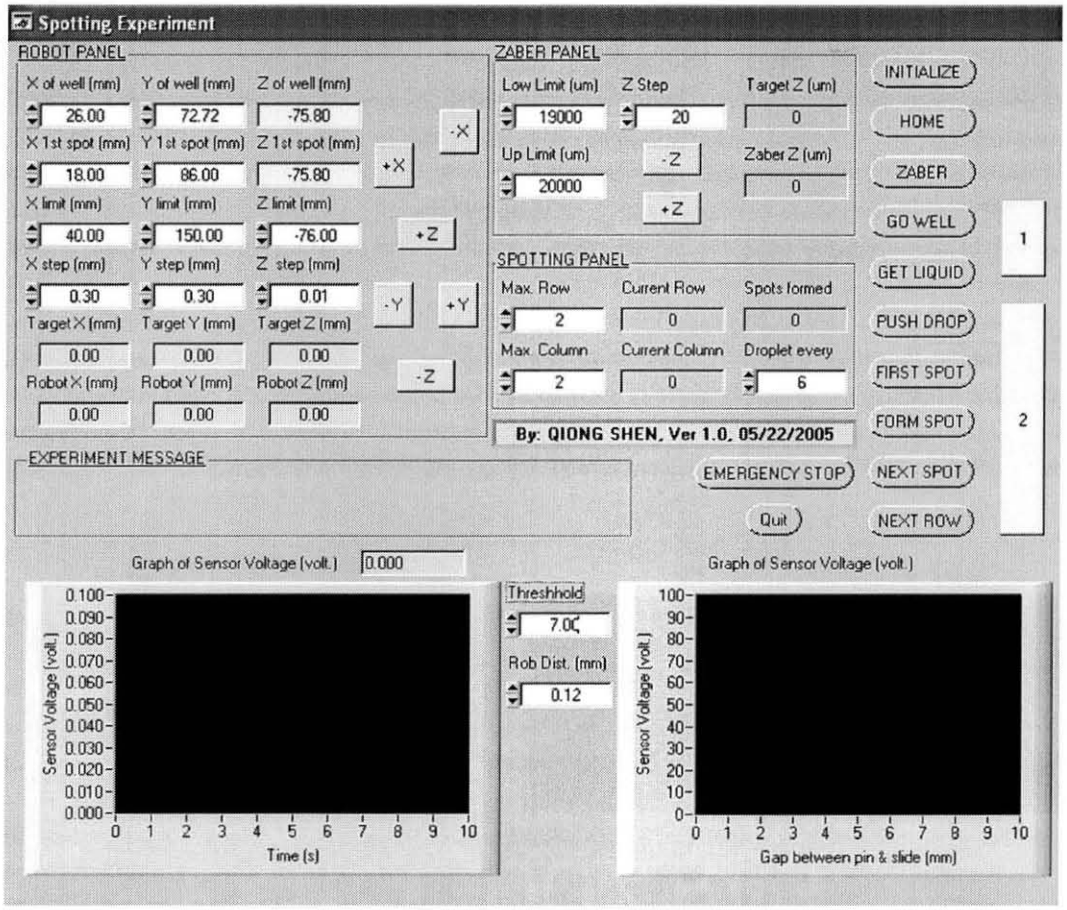
6. Controlling dwell time: After the spotting has been observed the pin has to be pulled back immediately so that the spot size does not grow bigger. Just after the spot formation is observed the pin is reversed, meaning the robot moves back towards home with a maximum speed of 10.1um/ms. Downloading the program to the robot reduces the time delay involved in spotting and pulling up of the pin and hence the spot size will not grow in size.

7. Surface preparation: The slides are coated with PLL (Poly-L-Lysine) so that the entire sample at the tip of the pin could be used in spotting. The coating spreads the spot uniformly and makes it feasible to get regular spots. Uncoated slides will not spread the spot uniformly and hence results in irregular spotting.

3.1 Description of Spotting Experiment

The robot and the Zaber positions are initialized, meaning the robot is taken to the home position and then the Zaber stage is moved to its maximum limit of 20000um. Now a fine cut and polished borosilicate pin of outer diameter 1.35mm and inner diameter 0.59mm is fitted to the end of the sensor driven by the Zaber stage. The maximum safe limit of robot X, Y, Z axis are specified to prevent the needle to come in contact with the experimental glass slide. The center to center distance between the spots can be fixed by the step incremental values of X and Y coordinates of the robot. The position of the well containing the sample liquid is specified by the three coordinates relative to the robot. Also the first spotting position coordinates are fed into the robot panel.

In the Zaber panel the maximum and minimum safe limits are to be mentioned along with the step increment which dictates the metered quantity of the liquid that will cling to the tip of the needle. In the spotting panel the maximum numbers of rows and columns to be spotted are mentioned. All the parameters shown in the front panel, Figure 3.2, are specified before the beginning of spotting process.


Spotting Experiment

ROBOT PANEL

X of well (mm) 26.00 Y of well (mm) 72.72 Z of well (mm) -75.80
X 1st spot (mm) 18.00 Y 1st spot (mm) 86.00 Z 1st spot (mm) -75.80
X limit (mm) 40.00 Y limit (mm) 150.00 Z limit (mm) -76.00
X step (mm) 0.30 Y step (mm) 0.30 Z step (mm) 0.01
Target X (mm) 0.00 Target Y (mm) 0.00 Target Z (mm) 0.00
Robot X (mm) 0.00 Robot Y (mm) 0.00 Robot Z (mm) 0.00

ZABER PANEL

Low Limit (um) 19000 Z Step 20 Target Z (um) 0
Up Limit (um) 20000 Zaber Z (um) 0

SPOTTING PANEL

Max. Row 2 Current Row 0 Spots formed 0
Max. Column 2 Current Column 0 Droplet every 6

By: QIONG SHEN, Ver 1.0, 05/22/2005

EXPERIMENT MESSAGE

Graph of Sensor Voltage (volt.) 0.000
Threshhold 7.00
Rob Dist. (mm) 0.12

Figure 3.2 Front Panel of spotting experiment.

When the spotting process begins, the pin is directed to the well and the Zaber is pulled back 1000um to fill the needle with the sample liquid. Then the robot is taken to the home position and the tip and sides of the needle are cleaned. Now the needle goes to the first spotting location as specified in the panel by the user which is about 120um from the surface of the slide. A step increment of 20um is given in the Zaber to get a metered amount of liquid at the tip of the needle. The present row number is checked with the maximum row number and if the last row has been reached then the present row is at maximum and spotting is stopped else the present column number is checked with the maximum column number and if the last column has been reached then the robot positions to the next row and gets a drop again, else the robot checks if the robot Z limit has been reached or not. If the robot Z limit has been reached then it means the robot has to be pulled back and spotting at the next column has to be executed by again checking for maximum column number occurrence. If the Z limit of the robot has not been reached then the spotting decision is transferred to the decision controller and a YES or NO decision is awaited from the decision controller. A "YES" from the decision controller results from a successful spot formation and then the robot has to back up and has to go to next column in the same row for spotting. A "NO" decision corresponds to unsuccessful spot formation and then the robot Z axis has to be jogged down by another 10 um and again the Z limit is checked and continued in the same fashion as depicted in the below flow chart Figure 3.3.

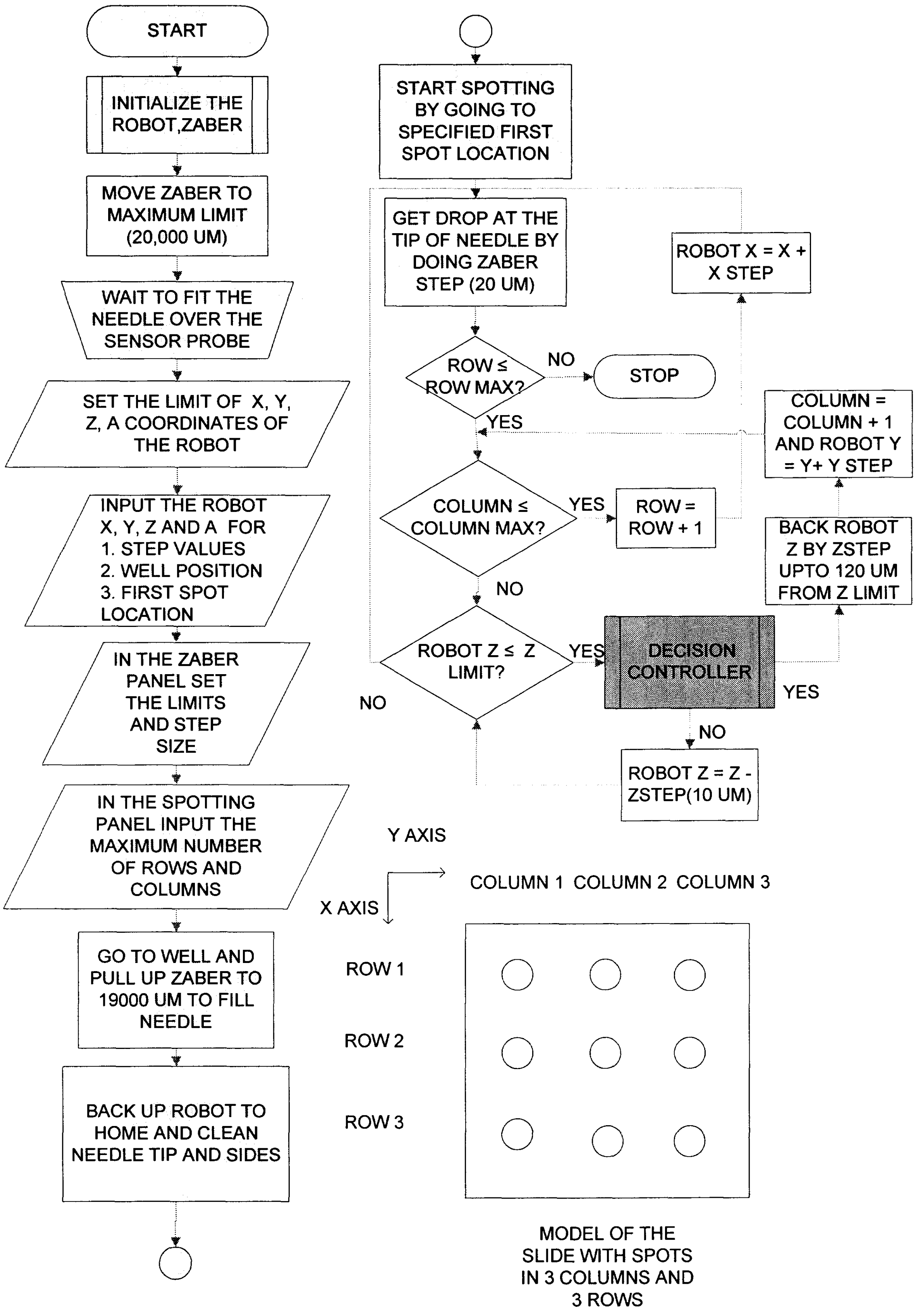

START
INITIALIZE THE ROBOT,ZABER
MOVE ZABER TO MAXIMUM LIMIT (20,000 UM)
WAIT TO FIT THE NEEDLE OVER THE SENSOR PROBE
SET THE LIMIT OF X, Y, Z, A COORDINATES OF THE ROBOT
INPUT THE ROBOT X, Y, Z AND A FOR 1. STEP VALUES 2. WELL POSITION 3. FIRST SPOT LOCATION
IN THE ZABER PANEL SET THE LIMITS AND STEP SIZE
IN THE SPOTTING PANEL INPUT THE MAXIMUM NUMBER OF ROWS AND COLUMNS
GO TO WELL AND PULL UP ZABER TO 19000 UM TO FILL NEEDLE
BACK UP ROBOT TO HOME AND CLEAN NEEDLE TIP AND SIDES
START SPOTTING BY GOING TO SPECIFIED FIRST SPOT LOCATION
GET DROP AT THE TIP OF NEEDLE BY DOING ZABER STEP (20 UM)
ROW ≤ ROW MAX?
NO
STOP
YES
ROBOT X = X + X STEP
COLUMN = COLUMN + 1 AND ROBOT Y = Y+ Y STEP
COLUMN ≤ COLUMN MAX?
YES
ROW = ROW + 1
BACK ROBOT Z BY ZSTEP UPTO 120 UM FROM Z LIMIT
NO
ROBOT Z ≤ Z LIMIT?
YES
DECISION CONTROLLER
YES
NO
NO
ROBOT Z = Z - ZSTEP(10 UM)
Y AXIS
X AXIS
COLUMN 1 COLUMN 2 COLUMN 3
ROW 1
ROW 2
ROW 3
MODEL OF THE SLIDE WITH SPOTS IN 3 COLUMNS AND 3 ROWS

Figure 3.3 Flow chart for the entire automatic spotting sequence.

3.2 Control Methodology used for Spotting

The controller's decision is the most crucial part in the formation of spots. The controller used here is simple look up table rules based. The observation from the initial experimental results for spot formation indicated that each time the spot is formed a particular pattern of graphical display is generated in the plot for Gap vs. Sensor voltage for each sample substance used. These graphs and spot formations are observed to be repeatable. It is observed that when the pin with the sample at the end approaches the slide the sensor voltage keeps increasing as the gap distance becomes closer to the slide. But when the non-contact spotting is done, meaning the droplet contacts the slide then there is a drastic decrease in the voltage readout from the sensor. At this point the needle can be pulled back and during this the droplet spotted on the slide tries to cling to the needle tip and elongates until the gap distance increase to a point were the spot gets detached from the pin. Figure 3.4 shows the graphical pattern observed during spot formation for buffer 3XSSC. The following Table 3.1 shows the regions of the graph and the action to be taken when that region is being plotted in the graph. The slope of the voltage is calculated as [| (Previous data – Present data) | * 1000] / Gap distance. Data refer to the intensity in volts along the Y axis in Figure 3.4.

Table 3.1 Look up Table for the Controller

Signal slope / **Sensor signal output**	*SLOPE OF VOLTAGE ABOVE THRESHOLD*	*SLOPE OF VOLTAGE BELOW THRESHOLD*
HIGH VOLTAGE RANGE	REGION II OF GRAPH (JOG PIN BY 10UM DOWN)	REGION I OF GRAPH (JOG PIN BY 30UM DOWN)
LOW VOLTAGE RANGE	REGION III OF GRAPH (PULL THE PIN UP BY 10UM)	REGION IV OF GRAPH (PULL THE PIN UP BY LARGER STEP OF 30UM)

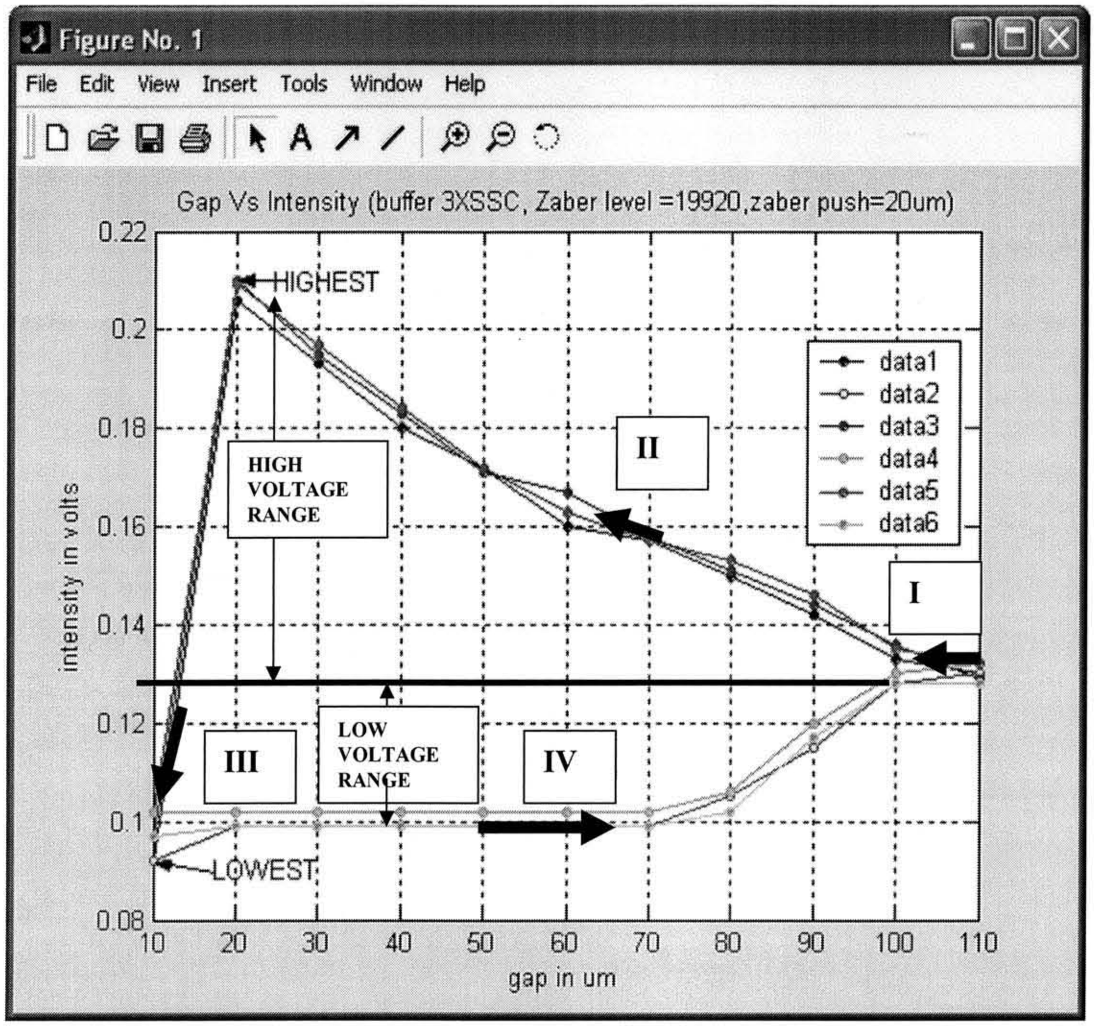


Figure 3.4 Standard spot formation Graph showing the four different regions.

The indication of droplet formation is given by the reach of the highest point. The highest point will be the point after which there is more than the threshold voltage slope and the data points lie in the low voltage range zone. After the highest point is reached the needle is commanded to be pulled back. The controller action is depicted the flow chat given in the Figure 3.5 where DAQ in the flowchart stands for Data Acquisition.

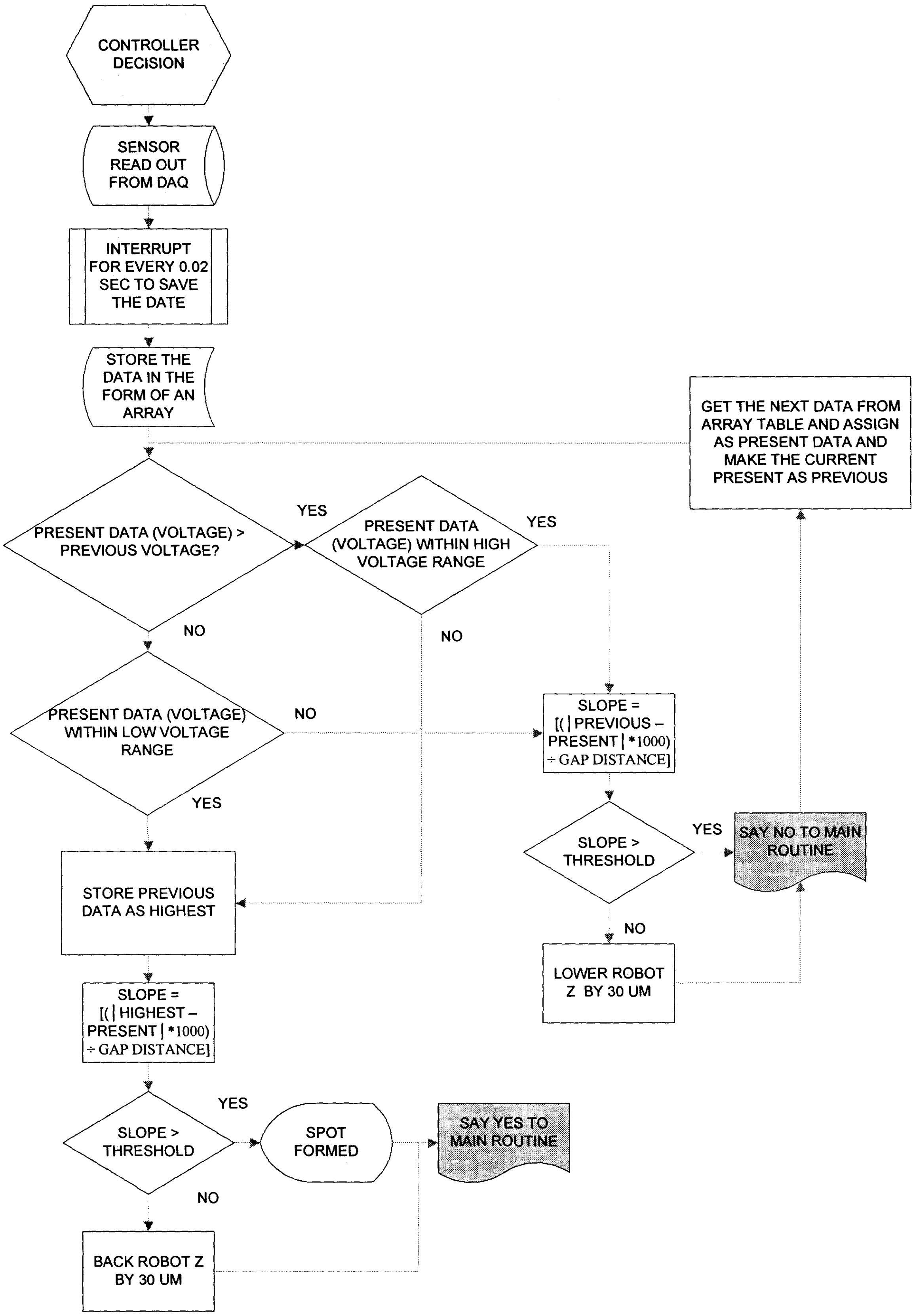

CONTROLLER DECISION
SENSOR READ OUT FROM DAQ
INTERRUPT FOR EVERY 0.02 SEC TO SAVE THE DATE
STORE THE DATA IN THE FORM OF AN ARRAY
GET THE NEXT DATA FROM ARRAY TABLE AND ASSIGN AS PRESENT DATA AND MAKE THE CURRENT PRESENT AS PREVIOUS
PRESENT DATA (VOLTAGE) > PREVIOUS VOLTAGE?
YES
PRESENT DATA (VOLTAGE) WITHIN HIGH VOLTAGE RANGE
YES
NO
NO
PRESENT DATA (VOLTAGE) WITHIN LOW VOLTAGE RANGE
NO
SLOPE = [(| PREVIOUS – PRESENT | *1000) ÷ GAP DISTANCE]
YES
SLOPE > THRESHOLD
YES
SAY NO TO MAIN ROUTINE
STORE PREVIOUS DATA AS HIGHEST
NO
LOWER ROBOT Z BY 30 UM
SLOPE = [(| HIGHEST – PRESENT | *1000) ÷ GAP DISTANCE]
SLOPE > THRESHOLD
YES
SPOT FORMED
SAY YES TO MAIN ROUTINE
NO
BACK ROBOT Z BY 30 UM

Figure 3.5 Controller action flow chart for spotting experiment.

The main routine is interrupted by a subroutine every 0.02 sec ie. sampled at 50 Hz. The data from the sensor is stored in the form of an array and the present data is compared with the previous value and is checked for the range in which it lies. If the value is higher than previous and as well lies in the higher voltage range then the robot has to jog down by another 10um and again the same check is repeated until the present data is leaser than the previous and lies in the lower zone. At this juncture the previous data is stored as the highest point. The forthcoming data values lying in the lower zone can be subtracted from highest point and this difference when divided by the gap value gives the slope in voltage. If this slope in voltage is less than the threshold mentioned in the panel then the spot formation has occurred and its time for the needle to back up and once again the main routine takes over the task. Thus spotting is controlled by the repeated voltage pattern observed.

CHAPTER 4

TEST RESULTS

4.1 Spot Formation

Three types of solutions are considered: 100% glycerol, 50% glycerol, and water. In each case, a 10% Cy3 dye is added to the solution for fluorescence. Results of spotting 100% glycerol are shown in Figure 4.1 where the top traces depict sensor intensity as the pin approaches the slide. The droplet on the pin opening comes into contact with the slide at 20 microns air gap distance at which the sensor intensity drops off by 83% due to the loss of total internal reflection. The disengagement of pin is characterized by elongation of the spot due to the viscosity of glycerol and the binding to the slide surface which has been coated with poly-L-lysine. The total elongation is about 70 microns after which the spot and the pin separate. The results are highly repeatable.

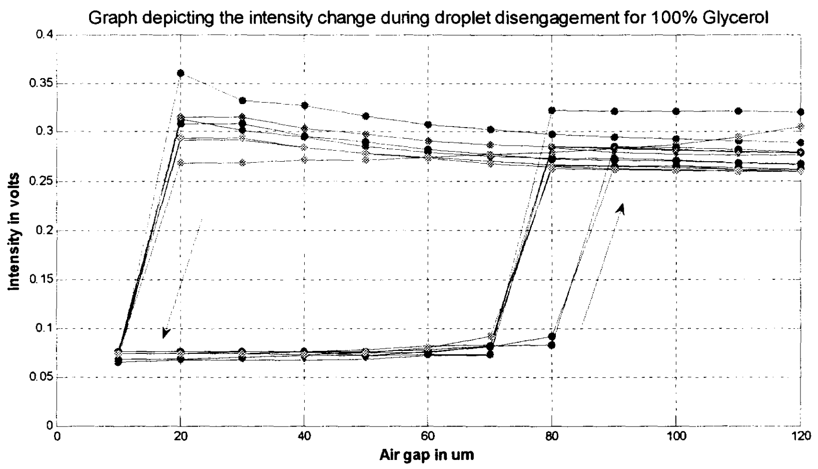
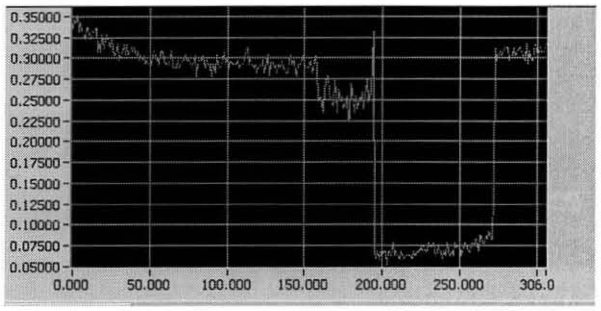


Figure 4.1 Intensity plot for 100% glycerol spot formation.

The corresponding time series plot with X axis scaled as 1 unit = 5 ms and the corresponding spot imaged by a 10 micron GenePix scanner are shown in Figure 4.2. Downward trend of the intensity curve indicates the pin's approaching the slide and the abrupt transition mirrors the changes in the intensity plot in Figure 4.1. It is also observed that the spots are round and uniform.



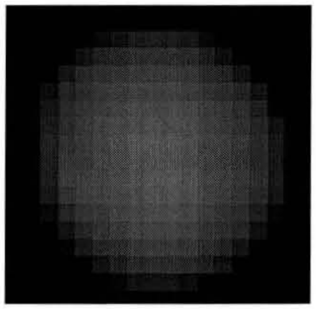


Figure 4.2 Graph depicting the intensity change during 160um droplet formation for 100% glycerol.

The second and third sets of tests dispense 50% glycerol and de-ionized water respectively. Cy3 is added to the sample liquid. The intensity plot, time series plot with X axis scaled as 1 unit = 1 ms and the corresponding spot imaged at 10 micron resolution in a GenePix 4000B scanner for 50% glycerol sample are shown in Figures 4.3 and 4.4. Those for de-ionized water are shown in Figures 4.5 and 4.6. During spot formation, the intensity drops by 56% for 50% glycerol and 30% for water. The intensity change is less pronounced than that for 100% glycerol sample because the effects of total internal reflection decrease with the refractive index. Furthermore, a low viscosity and weaker interaction with poly-L-lysine also result in slightly large elongation and subsequently larger spots.

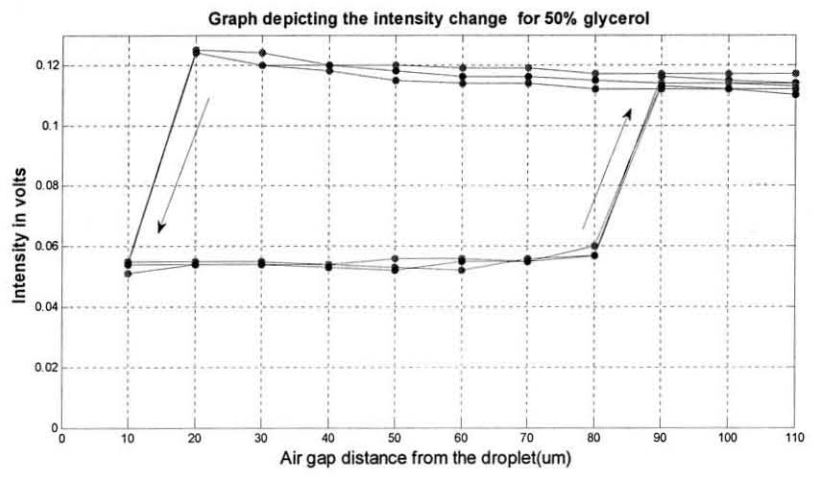


Figure 4.3 Intensity plot for 50% glycerol spot formation.

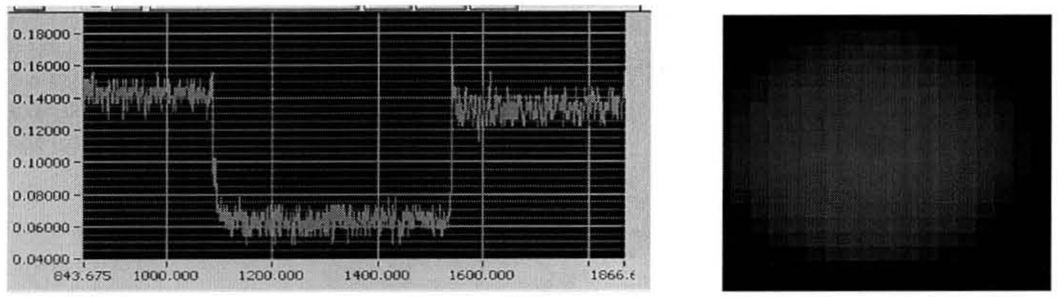


Figure 4.4 Graph depicting the intensity change during 190um droplet formation for 50% glycerol.

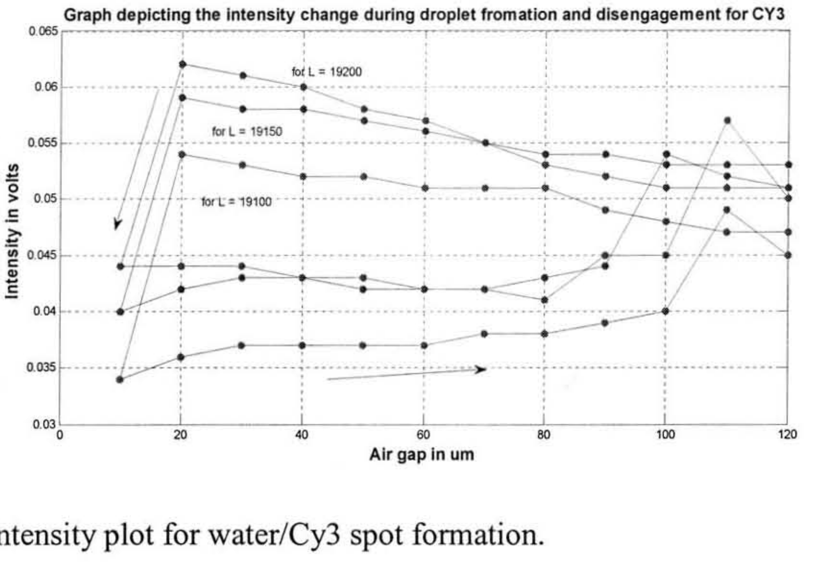
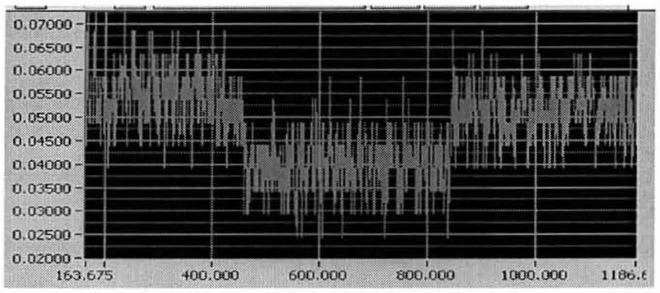


Figure 4.5 Intensity plot for water/Cy3 spot formation.



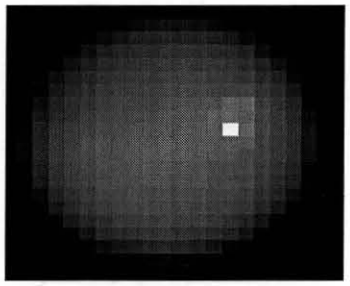


Figure 4.6 Graph depicting the intensity change during 210um droplet formation for water/Cy3.

In addition to the optical lever, the SmartPin also utilizes the total internal reflection principle: due to the difference in refractive indices of air, glass (pin), sample material, and the shape of the material, contact condition can be derived by continuously monitoring the sensor intensity. For example, 100% glycerol has a refractive index of 1.47 while that of water is about 1.33, both of which are sufficiently high to induce total internal reflection at the pin tip due to the liquid convex curvature. Once the liquid contacts the slide, the internal reflection is replaced with transmission due to the change of refractive index from 1 (air) to 1.57 (glass), resulting in a sudden drop of sensor signal intensity as most of the light is transmitted away from the sensor. This effect is shown Figure 4.7 where ray tracing is utilized to show the light transmission patterns. Before the droplet at the pin tip contacts the slide, a major portion of light rays are trapped within the pin and reflected back to the sensor as show in Figure 4.7 (up). Once the contact is made, most of the rays are transmitted into the slide and backscattered, resulting in a lower sensor signal. This signal intensity drop is compared to the threshold stored in the database so that a disengagement signal is synthesized and communicated to the robot and the piezoelectric stack. An internal timer provides the predetermined contact delay time which is one of the dominant influence factors on spot size.

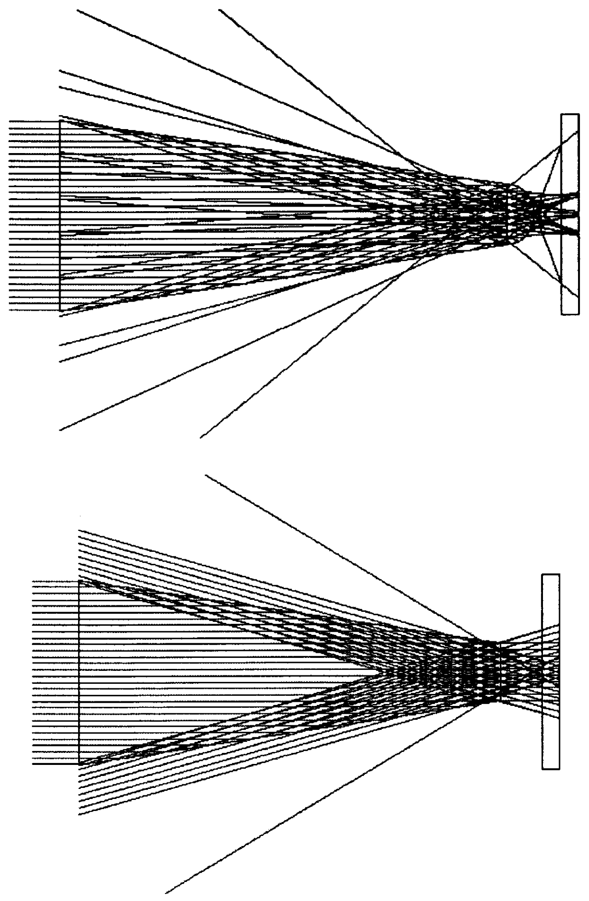


Figure 4.7 Ray Tracing result for SmartPin before (Up) and after (Down) droplet engagement.

A high frequency vibration can be generated by the fiber driver on the piezoelectric positioner to facilitate separation of the droplet from the fiber. This way, extremely small droplets can be formed at high speed. For example, a 50 micron diameter fiber can deliver droplet size as small as 0.1 nL, resulting in a 25 micron spot and a ten-fold reduction of use of the DNA materials. These reduced spots can be reliably detected by scanners with 0.1 fluor/micron^2 and 2.5 –5 micron resolution. Currently, the pin diameter is about 200 microns and spots size can be controlled from 80 microns to 220 microns in diameter. Throughout the process, the SmartPin does not come into contact with the slide and therefore, does not encounter the standard wear out and slide damage problems of the impact pins.

4.2 Spot Detection

The SmartPin not only monitors and controls spot formation, but also doubles as a spot sensor. This function is useful for aspiration of sample as well as spot quality check. A rapid raster scan algorithm is stored in the PC and in turn controls the Seiko D-Tran robot to scan in an x-y plane 60 microns over the slide. An example droplet formed by 100% glycerol is shown in Figure 4.9 where a 3 D view of the spot is imaged (left). Cross sectional view depicting the height of the spot at the time of scanning is shown in the right. Figure 4.10 shows a 3 D view of a 400 micron droplet formed by 100% glycerol. Raster scan is done to find the geometry of the droplet. The 3 D view of the spot can be captured. The pin opening affects the intensity pick up. A pin with a bigger diameter opening admits more light and can therefore detect smaller spots. For this experiment the pin is placed 60 microns above the slide having the sample. The robot Z axis distance is thus fixed and now the robot's X and Y coordinates have to be changed to get the raster scan as shown in Figure 4.8.

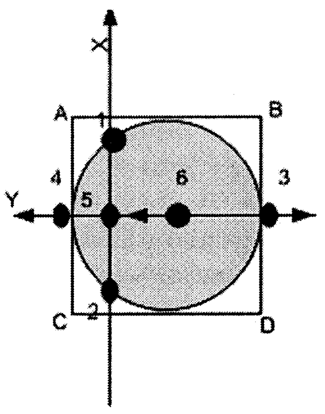


Figure 4.8 Raster scan procedure.

Raster scan procedure:

1. First a column wise scan can be performed by fixing the Y axis value as constant as the Smartpin has to keep moving in the same vertical line path and the X axis is programmed to move by 10um increment. The intensity picked by the sensor is recorded when it moves along a column.

2. The position of the robot is noted as points 1 and 2 as shown in Figure 4.8 when the change in intensity begins and drops during the column scan. The difference in the position value of the start point of intensity change and end point of intensity change will help us fix the midpoint 5 as in Figure 4.8 along the X axis.

3. Similarly by fixing the Z coordinate and fixing X coordinate at point 5 along X axis and programming to move by 10um increment along the Y axis on either sides of point 5 the row scan is performed.

4. The position of the robot is noted as points 3 and 4 at the start and the end point of intensity change during the row scan. The difference between these two positions of the robot is the diameter of the spot in microns. And the mid point of these two positions namely point 6 in Figure 4.8 will give the midpoint of the spot. Now that the size of the droplet is well defined, the raster scan to get the geometry can be carried out.

5. The X direction movement will be column wise scan. The edge of the spot namely point A as in figure can be made as the start point of the scan by just incrementing the position of point 4 by the radius of the spot in the X axis. The Y coordinate value and Z coordinate values are fixed and the pin is moved along the X direction by 10um each time till point C is reached.

6. Then the next column is scanned by incrementing 10um along the Y axis and fixing it. The X coordinate is incremented in steps of 10um for the range of the diameter of the spot and Z value is fixed 60um above the slide as before. In the same fashion all the columns are scanned until points B to D are covered.

7. The intensity values recorded in each column scan is arranged in the form of a matrix. This intensity value will be one of the dimensions of the 3D plot and the other two dimensions will correspond to the diameters of the spot in X and Y directions. Then a mesh plot is done to obtain the geometry of the spot.

A mesh plot gives the 3D view of the droplet as shown in the Figure 4.10. There are three usage of the scan. First, the newly formed spot can be checked for acceptability and if necessary, re-dispense. Second, the pin is capable of locating the spot for hybridization (e.g. "hybridization on the chip"). Third, the pin can locate sample materials in e.g. microwells for aspiration. This is significant if the sample material is limited in volume.

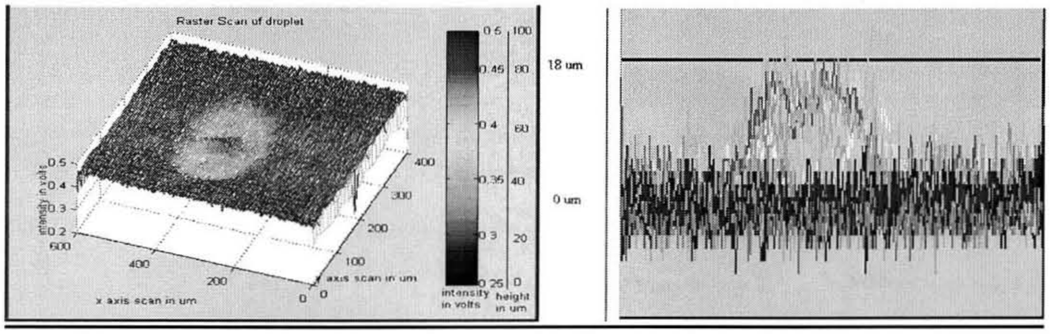


Figure 4.9 Graph depicting the 3D view of a 220micron diameter droplet and a cross sectional measurement of the spot which is about 18 microns high at the time of scan.

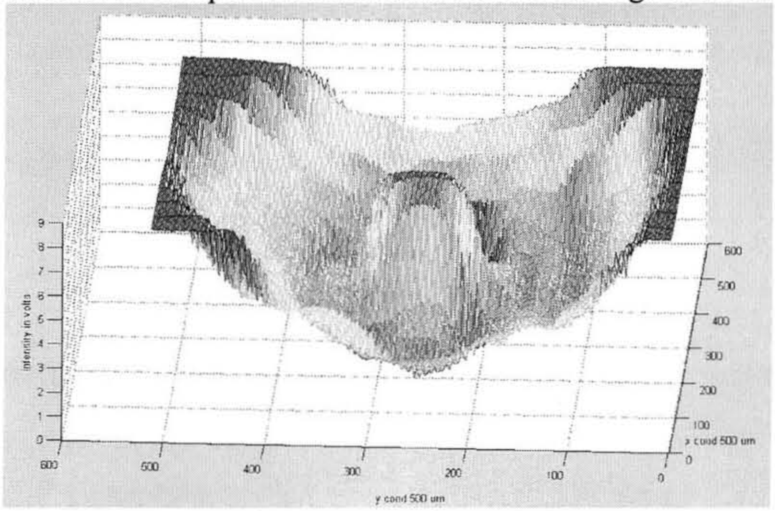


Figure 4.10 Graph depicting the 3D view of a 400micron diameter droplet of 100% glycerol.

4.3 Production of Uniform Size Microarrays

Production of uniformly spaced microarray spots is carried out with the SmartPin for 100%, 50% glycerol and de-ionized water. Sample results are shown in Figure 4.11 and Figure 4.12 where uniform and round spots are clearly visible.

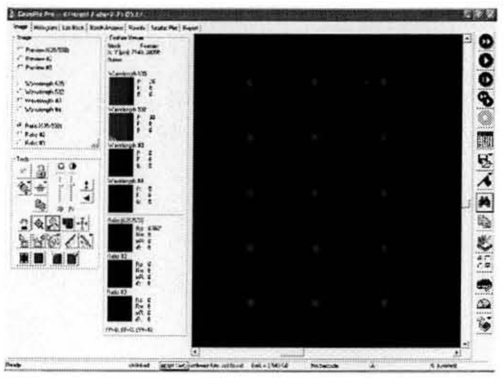
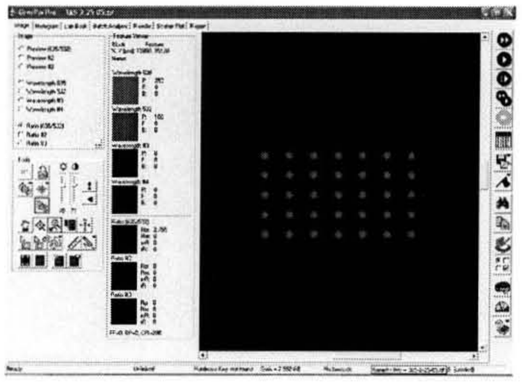
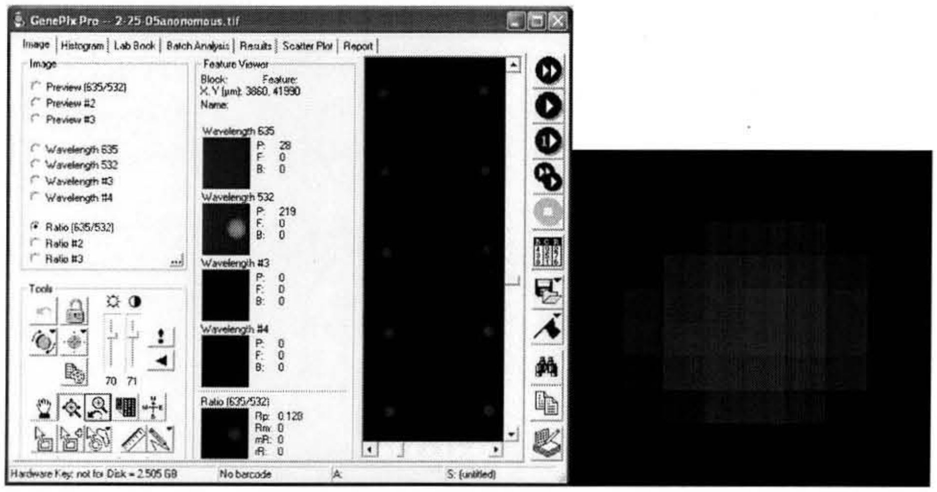



Figure 4.11 Spot matrix formed out of 100% glycerol, each spot measures 160microns (left) and 50% glycerol , each spot measuring 190 microns (right).


GenePix Pro -- 2-25 05anonomous.tif
Image | Histogram | Lab Book | Batch Analysis | Results | Scatter Plot | Report
Image
Tools
70 71

Preview [635/532]
Preview #2
Preview #3
Wavelength 635
Wavelength 532
Wavelength #3
Wavelength #4
[✓ Ratio [635/532]]
Ratio #2
Ratio #3

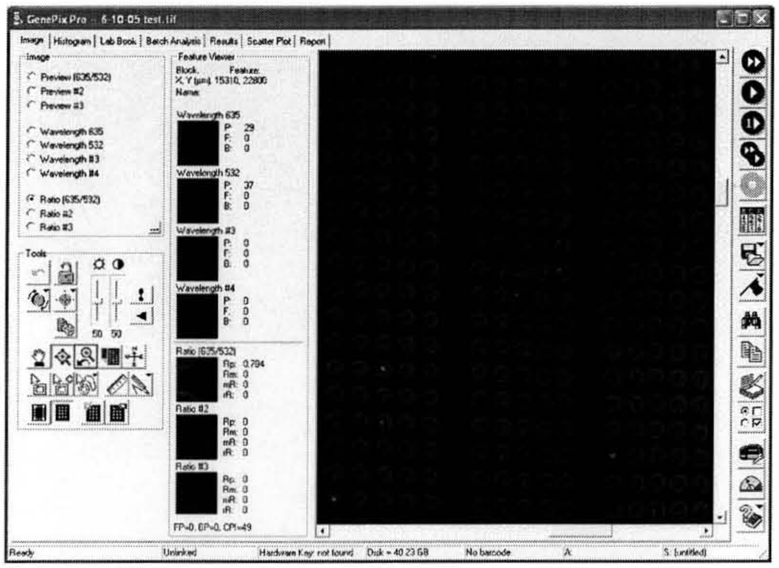
Feature Viewer
Block: Feature:
X, Y [µm]: 3860, 41990
Name:

Wavelength 635 P: 28 F: 0 B: 0
Wavelength 532 P: 219 F: 0 B: 0
Wavelength #3 P: 0 F: 0 B: 0
Wavelength #4 P: 0 F: 0 B: 0
Ratio [635/532] Rp: 0.128 Rm: 0 mR: 0 rR: 0

Hardware Key: not for Disk = 2.505 GB | No barcode | A: | S: (untitled)

Figure 4.12 Microarray with 100% glycerol and 80 microns diameter spots.

Figure 4.13 and Figure 4.14 show that the spots made by the smart pin are uniform in size with uniform spacing in between the spots. Different spot sizes can be achieved by properly positioning the same pin above the slide surface and by controlling the Zaber push.

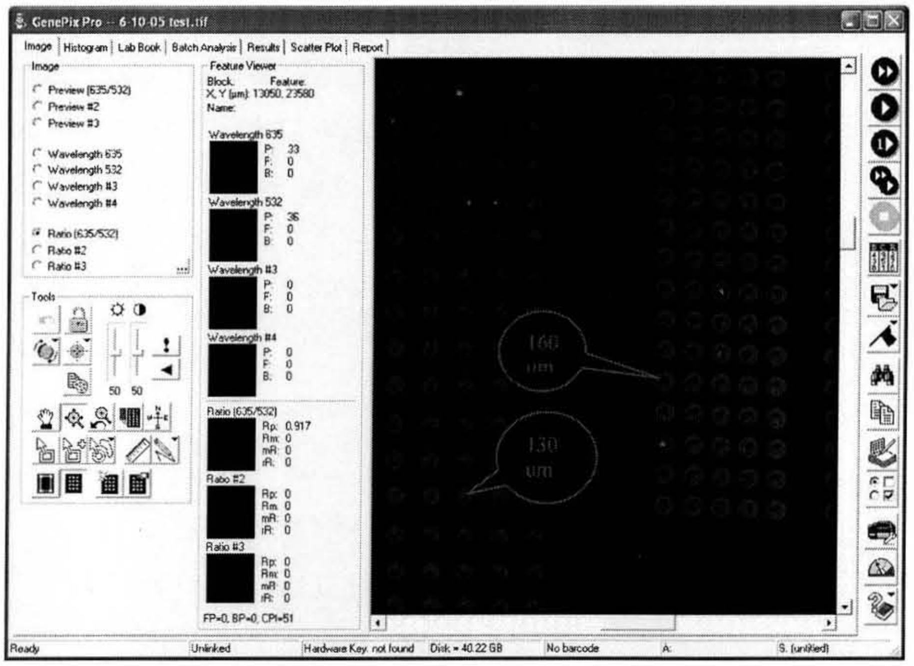


Feature Viewer
Block: Feature:
X, Y (µm): 15310, 22800
Name:

Wavelength 635 P: 29 F: 0 B: 0
Wavelength 532 P: 37 F: 0 B: 0
Wavelength #3 P: 0 F: 0 B: 0
Wavelength #4 P: 0 F: 0 B: 0
Ratio (635/532) Rp: 0.784 Rm: 0 mR: 0 iR: 0
Ratio #2 Rp: 0 Rm: 0 mR: 0 iR: 0
Ratio #3 Rp: 0 Rm: 0 mR: 0 iR: 0
FP=0, BP=0, CPI=49

Ready | Unlinked | Hardware Key not found | Disk = 40.23 GB | No barcode | A: | S: (untitled)

Figure 4.13 Microarray with spotting solution 3XSSC with 160 microns diameter spots.



Feature Viewer
Block: Feature:
X, Y (µm): 13050, 23580
Name:

Wavelength 635 P: 33 F: 0 B: 0
Wavelength 532 P: 36 F: 0 B: 0
Wavelength #3 P: 0 F: 0 B: 0
Wavelength #4 P: 0 F: 0 B: 0
Ratio (635/532) Rp: 0.917 Rm: 0 mR: 0 iR: 0
Ratio #2 Rp: 0 Rm: 0 mR: 0 iR: 0
Ratio #3 Rp: 0 Rm: 0 mR: 0 iR: 0
FP=0, BP=0, CPI=51

Ready | Unlinked | Hardware Key not found | Disk = 40.22 GB | No barcode | A: | S: (untitled)

Figure 4.14 Microarray with spotting solution 3XSSC with 160 microns diameter spots and 130 micron spots.

4.4 Experiments with Molecular Beacons

"Molecular beacons are single-stranded oligonucleotide hybridization probes that form a stem-and-loop structure. The loop contains a probe sequence that is complementary to a target sequence, and the stem is formed by the annealing of complementary arm sequences that are located on either side of the probe sequence. A fluorophore is covalently linked to the end of one arm and a quencher is covalently linked to the end of the other arm. Molecular beacons do not fluoresce when they are free in solution. However, when they hybridize to a nucleic acid strand containing a target sequence they undergo a conformational change that enables them to fluoresce brightly."

"In the absence of targets, the probe is dark, because the stem places the fluorophore so close to the nonfluorescent quencher that they transiently share electrons, eliminating the ability of the fluorophore to fluoresce. When the probe encounters a target molecule, it forms a probe-target hybrid that is longer and more stable than the stem hybrid. The rigidity and length of the probe-target hybrid precludes the simultaneous existence of the stem hybrid. Consequently, the molecular beacon undergoes a spontaneous conformational reorganization that forces the stem hybrid to dissociate and the fluorophore and the quencher to move away from each other, restoring fluorescence" [13] as shown in Figure 4.15.

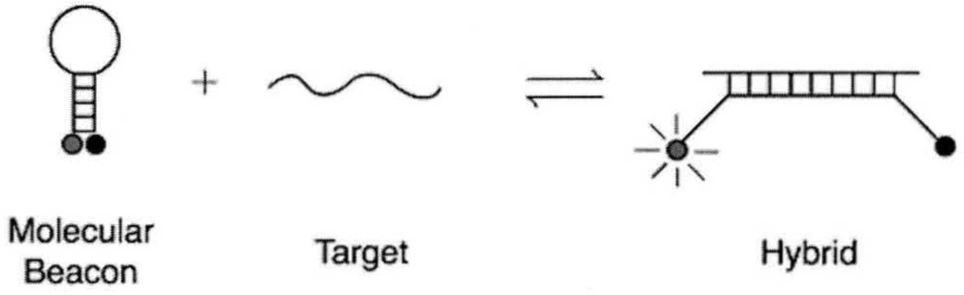


Figure 4.15 Molecular Beacon combining with the target to produce fluorescence. [13]

Determination of Sensitivity of Fluorescence by the Scanner:

The smart pin can handle a metered quantity of the Molecular Beacon (MB) and can be used to deliver the MB to wells. There will not be any waste of material as a metered quantity is taken in the pin and is fully delivered into the wells. 1ul of each of the solutions shown in Table 4.1 is delivered into the wells machined into a plastic surface and the fluorescence of each well is determined with a GenePix microarray scanner. The focus is set to -50um (towards the slide) so that the scanner can look into the wells.

Table 4.1 Sample Solutions of Molecular Beacons
MB- Molecular Beacon, T- Target, nM- nano Molar

	A: BUFFER	B:MB	C:TARGET	D:MB+TARGET
1ul	1X PCR	500 nM	500 nM	500 nM +500 nM
1ul	1X PCR	50 nM	500 nM	50 nM +500 nM
1ul	1X PCR	5 nM	500 nM	5 nM +500 nM

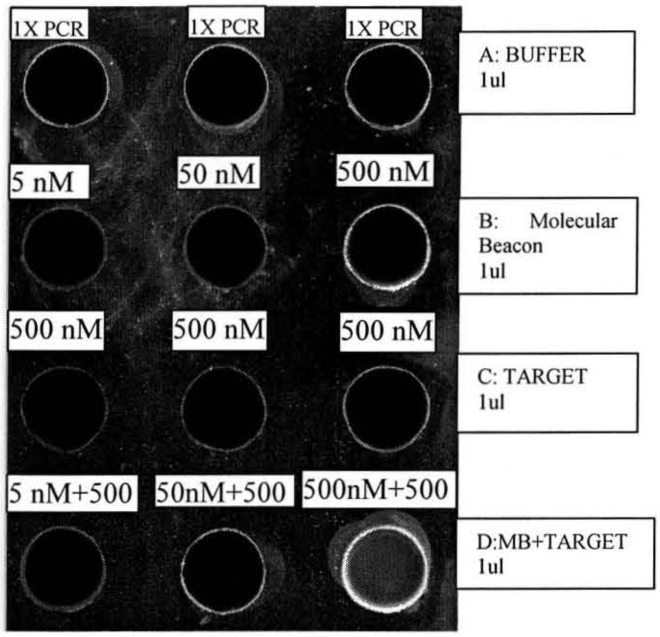


Figure 4.16 Fluorescence exhibited by the buffer, MB, T and MB and T together for varying quantities.

Figure 4.16 shows the fluorescence pattern observed by the scanner. It can be seen that the Molecular Beacon and target combination produce fluorescence as expected.

In the following experiment, one microliter of Molecular Beacon, target and the combination of target and Molecular Beacons are spotted in plastic wells. Each time the concentration of the materials are varied in steps of 100nM from 100 to 500nM as indicated in Figure 4.17 and the fluorescence intensity is measured by the scanner.

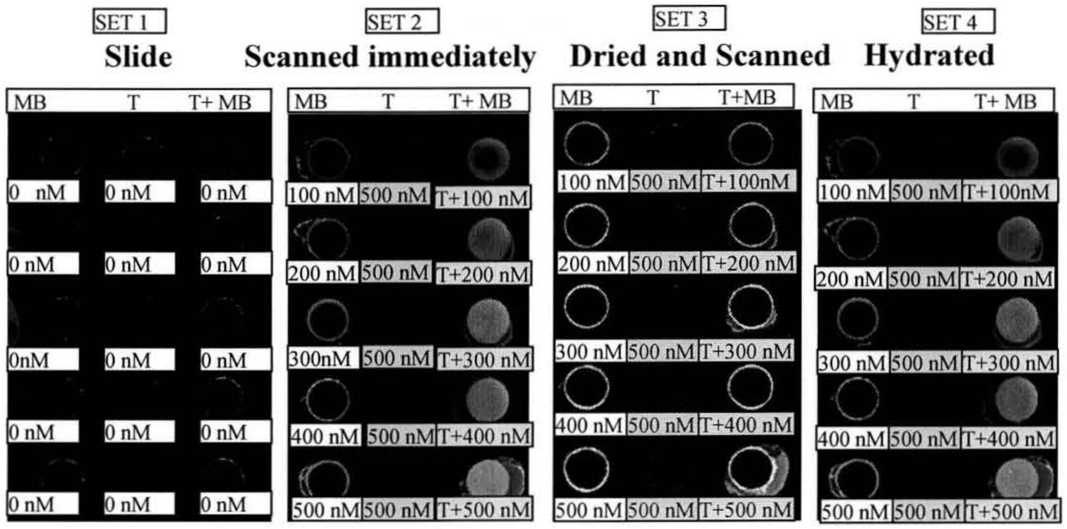


Figure 4.17 Fluorescence observed by the scanner.

MB -Molecular Beacon with the following sequence:
Fluorescein – <u>CGCAG</u> ACC ATG ATC GGC GGC <u>CTGCG</u> – BlackHole Quencher 2. The underlined sequences are the arm-sequences of the Molecular Beacon, and are not related to the target of the MB.

T -Target always 500 Nano Molar (nM) (Complementary Oligonucleotide target with sequence TT GCC GCC GAT CAT GGT TT)

T+MB - Target + Molecular Beacon

The four sets of experiments can be described as follows:

SET 1: The plastic slide was scanned to check for fluorescence effect. There was no fluorescence noticed.

SET 2: The molecular beacon, target and the combination of target and molecular beacon of the specified concentration were spotted by the use the Smartpin metered to deliver 1ul and the slide was scanned immediately before drying. A near background fluorescence is observed in the MB alone or the target alone. But appreciable fluorescence is observed in the combination of the target and the molecular beacon. The lower the concentration of the MB the weaker is the fluorescence. So it can be noticed that the pattern of fluorescence increases as the concentration of MB increases from 100 nM to 500 nM.

SET 3: After the slide was scanned as stated under SET 2, it was left to dry for 30 min and scanned again. It can be noticed that there is edge effect. The molecules gravitate towards the edge of the well and accumulate around it and fluorescence can be seen in the form of a ring.

SET 4: After scanning as in SET 3, the wells are re-hydrated using distilled water. While scanning a homogeneous pattern of fluorescence can be observed as before in SET 2 results unlike the ring effect in SET3. Again the fluorescence increases as the concentration of MB increases from 100 nM to 500 nM.

As the concentration of the Molecular Beacon increase, the Composite Pixel Intensity (CPI) observed under the scanner also increases pronouncing more fluorescence. The CPI is a representative composite intensity calculated for each pixel, based on the intensity values. The slope of the plot in Figure 4.18 is almost linear with 6600 CPI increase with every 100nM increase in concentration of the Molecular Beacon.

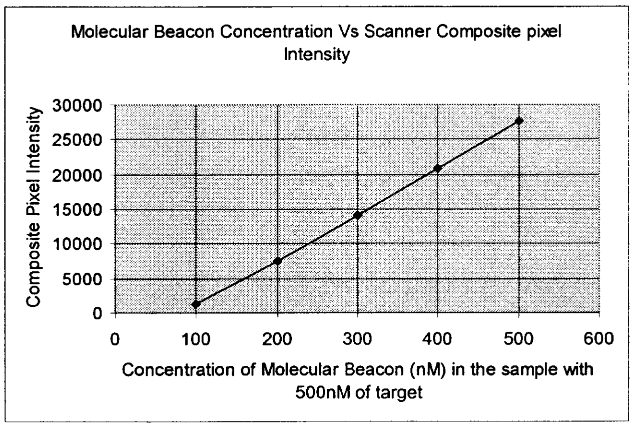


Figure 4.18 Concentration of Molecular Beacon Vs Composite Pixel Intensity.

Apart from efficient material handling and spotting in the wells the smart pin can handle various materials without cross contamination after cleaning with distilled water each time. The sample to be spotted, 3XSSC with 10% Cy3 dye for fluorescence, is picked up by the pin and the spotting is carried out in one column on a glass slide. Six spots are spotted in a column as shown in Figure 4.19. Then the sample is emptied by moving the Zaber to the maximum limit. Now the pin is immersed in distilled water and the Zaber is jogged up and down 4 to 5 times to draw and release distilled water in the pin. Standard ultrasound sonicating is not necessary. Next spotting is carried out with distilled water in the next column and six spots are made. Following this again a fresh supply of the buffer and Cy3 dye is filled in the pin and spotting of the next column is carried out. Thus an alternate column of buffer with Cy3 and distilled water spotting is done. It is evident from the Figure 4.19 that there is no contamination in the pin after cleaning and refilling. The spots in every alternate column (water only) do not produce fluorescence. Only the spots made with the buffer and Cy3 exhibit fluorescence.

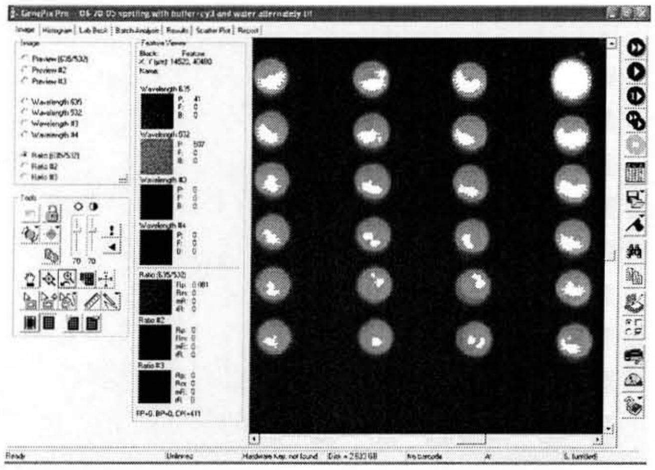


Figure 4.19 Alternate columns of buffer with 10% Cy3 and distilled water spotting showing no cross contamination of material handled by the pin.

CHAPTER 5

CONCLUSION AND FUTURE WORK

In this work, a novel liquid dispensing system termed SmartPin is described. It is based on integrated active sensing and control to produce precise and uniform liquid spots in a contactless manner. Due to active fluid displacement, the pin is capable of handling highly viscous materials such as glycerol. Primary application of this system includes DNA and protein microarray fabrication, and other situations where aspiration of small liquid quantity is required. The current laboratory prototype is capable of spotting arrays of circular dots with uniform dimensions and controlled diameters from 80-200 microns. The features of the pin include not only spotting and spot detection, but it also can locate the missing spots based on the optical sensor intensity information and can redo the spotting.

In the future, the piezoelectric actuator can be incorporated to the model to ensure precise positioning of the slide and to break the elongation of the droplet sooner to get smaller spots. The sensing capability of the smart pin can be exploited to find the nature of the material used. Experimental results show that more elongation occurs in less viscous sample. In the future more efficient control strategy can be employed in the controller to detect the spot formation and the commands can be downloaded to the robot to draw the pin back with the maximum speed of 10.1um/ms so as to achieve smaller dot sizes. The current Fotonic sensor can be replaced by more sensitive sensor and the current Zaber stager with accuracy +/-8 um used can be replaced by a more accurate actuator. A platform with accuracy +/-1um and higher speed will enhance the technique.

APPENDIX A

HARDWARE SPECIFICATIONS

The specifications of the robot, software used, connection establishment between the robot and the computer terminal and the specifications of the Fotonic sensor is discussed below.

1. Specification of SEIKO D-TRAN robot:

Mounting Area

Work envelope: x-axis is 300mm, Y-axis is 200mm and Z-axis is 100mm.

Base plate dimensions: 470mm deep X 420 mm wide.

Seiko D-Tran robot specifications

Repeatability: + or – 0.008mm (0.0003 in.)

Speed: 1500 mm/sec (59 in/sec)

Accuracy: 0.030 mm (0.0012 in)

Payload: 1 kg

Seiko D-Tran dimensions: 554 mm wide X 1084 mm deep

Seiko D-Tran weight: 105 Kg

The Z-axis is located between the A-axis and the X-axis of the Seiko D-Tran robot. The power train of the axis is a rack and pinion assembly.

Z-axis specifications

Stroke: 100 mm (3.94 in)

Resolution: 0.010 mm

Speed: 500 mm/sec

Axis dimensions: 100 mm deep X 150 mm Wide X 441 mm high

2. DARL language overview

The standard language used by the Seiko D-Tran Robot is DARL Version 2.3. It is easy to use and similar to BASIC language.

Steps for running a program using DARL from tech terminal:

- Make sure the TERMINAL/AUTO switch is in TERMINAL position.
- Turn the main power on. Switch the main power to "ON" position.
- Turn on the power on the servo motors by pressing the GREEN button.
  One will see:
  ```
  M DARL XY 3000 VER 2.3
  COPYRIGHT 1984 SEIKO I&E
  ```
  'M' refers to the Monitor mode (it is the default mode of the robot).
- Type 'HOME', which will get the robot in the home position. Home position is necessary before the start of any operation to have the robot know its workspace and initial transitional points.
- Type 'EDIT' to go to EDIT mode, from the Monitor mode.
  One will see:
  ```
  E
  ```

This is EDIT mode, and this mode is used for typing own programs and feeding them to the controller. The following is an example program to move the robot arm in place from home position to a position described by the summation of two translation points T20 and T 30.

```
NEW

10 T20=100.0 20.0 300.0 0.0

20 T30=20.0   30.0 100.0 0.0

30 MOVE T40=T20+T30

40 END
```

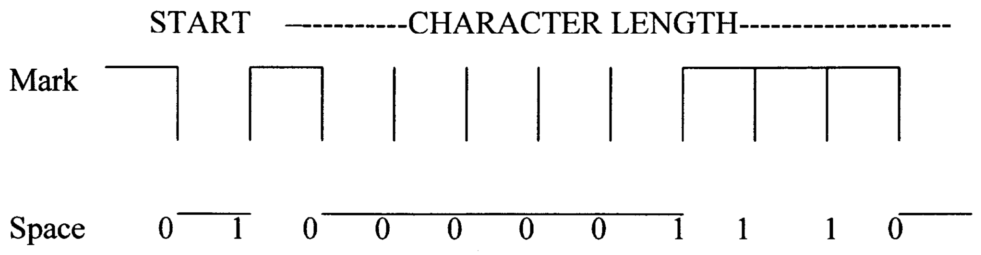
Program execution is done in MONITOR mode. To get back in to MONITOR mode type QUIT in EDIT mode. This will bring back to MONITOR mode from EDIT mode. To run the program and move robot arm to T40 position press START. Pressing START executes the current program in RAM.

3. Serial communication concept

The Seiko D-Tran controller has two RS–232 ports for serial communication with any device. It is a 25 pin female connections. RS-232 is a standard that describes a common method of serial signaling using positive voltage for logic "0" and a negative level for logic "1". When RS-232 line is powered up but idle (not conveying data) it is in its lower voltage stage. When data is sent RS-232 jumps to its higher voltage state momentarily to alert the receiver and this is called the 'start bit'. The communication takes place through the American Standard Code of Information Interchange (ASCII). The Seiko controller's Tech-Terminal or the keyboard's keys are coded in to 7 bits ASCII, which is normally the character length. Parity is generally added for authentication. This is followed by one or two stop bits. Typical ASCII transmission is shown for transmitting a character say "A".

Character:	A
Decimal:	065
Hexadecimal:	041
Binary:	01000001

Format:



The bit time is expressed in samples per second, known as baud. The common baud rate used is generally 9600 baud. A bit time is defined as 1/bps. For example, at 9600 bps the bit time is 1/9600 =104 microseconds.

Protocol and use of DARL RS-232 communication package

Host can have the ability to start and stop the robot. Host could be a Cathode Ray Tube (CRT), Personal or Business Computer, Vision Systems. It uses ASCII for communication and any computer or peripheral device compatible to a UART.

Conditions to Match the Protocol:

- Several Port characteristics must be matched for 2 devices to communicate, using EIA's RS-232. It is Important to have devices to be set as DTE (Data Terminal Equipment) or DCE (Data Communication Equipment). It is not important that which device is DTE or DCE, it is only important that both are not set as same. Speed at which the communication is done should also be same for both the communicating devices. The Robot controller has a UART (Universal Asynchronous Receiver Transmitter) which is Intel 8251A.

- Note that the Baud Rate factor (B1 B0) is never to be changed from 1 0 for current version of communication package as (per the manual). It's according to the clock setting of the processor. Also a Null Modem can be used incase both terminals are either DTE/DCE. The integer in its binary form determines the characteristics of the port. Figure A.1 demonstrates how the desired characteristics of RS-232 port are reduced to the integer in DARL statement.

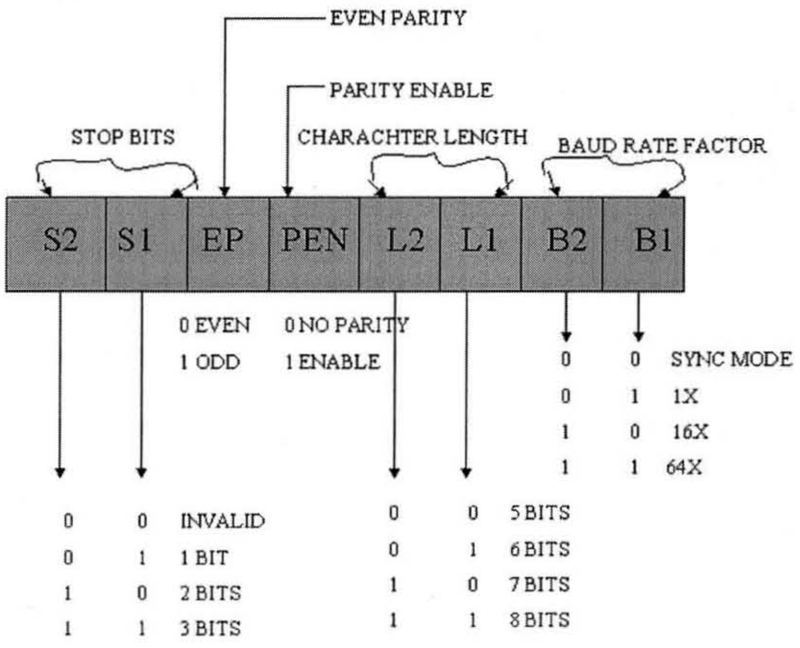


Figure A.1 Reduction of RS-232 characteristics into integers in DARL statement.

Connecting the Robot to the Computer

In the Robot side first of all the physical connection is done using a 25-9 pin cable build using connectors. The wires are shielded to avoid picking stray signals. Also the length of the cable should be approximately 1.5 meter for errorless transmission. The default input and output unit of the robot's controller is the Teach-Terminal. At the robot end the important 8 pins used out of 25 pins of RS 232 Port (Robot Controller) are listed below.

1-Ground	5-Clear To Send
2-Transmit Data	6- Data Send Ready
3-Receive Data	7-Signal Ground
4-Request to Send	20-Data Terminal Ready

DARL statement OUNIT (n) assigns the output unit of the controller and ‘n’ is the unit which will act as the output unit. DARL statement IUNIT (n) assigns the input unit of the controller.

Values of (n):	0 LCD/Key Board on Teach Terminal	
	1 RS-232 Port #1	
	2 RS-232 Port # 2	(there are 2 Ports on the back panel)
	3 Printers	

Example of **MONITOR** Mode

```
DO OUNIT1 {press enter}
DO IUNIT1 {press enter}
```

The ‘Z’ statement holds the key in communication. Its syntax is “Z=integer”. Now this integer value is the DECIMAL equivalent of the 8-Bit Binary number from the UART. Example if Z=**78** DEC ie., **0 1 0 0 1 1 1 0** BIN, it implies that the controller is set to following parameters for communication: 8 Characters

NO Parity

1 Stop Bit

Example of ‘Z’ in **EDIT** Mode:

```
10 Z=78
20 END
```

Example of ‘Z’ in **MONITOR** Mode:

```
DO Z=78 {press enter}
```

In the Computer Side as shown in Figure A.2 the COMM1 port is used.

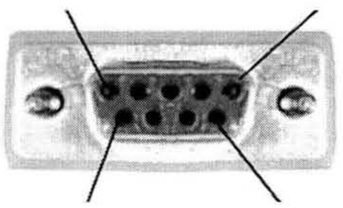
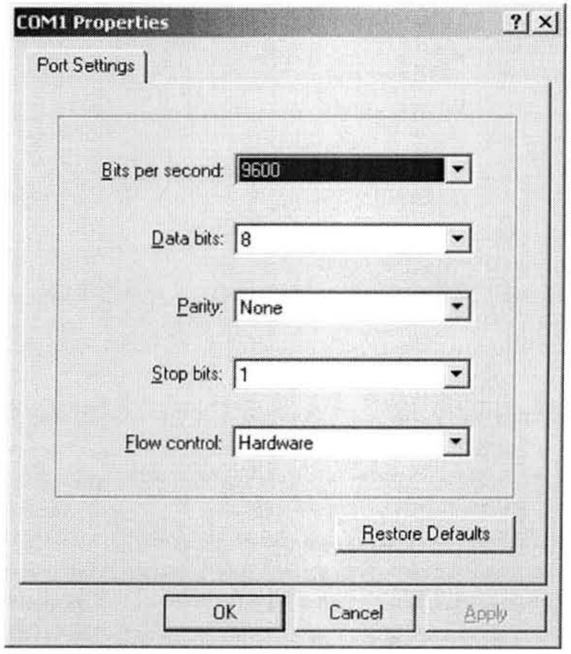


Figure A.2 Communication port pin configuration.

Hyper terminal is a terminal program that will enable a PC to communicate directly with a Communication port (Eg: Comm1). On Windows 95/ 98 /98 SE, Windows 2000, Windows ME, or NT4, the Windows HyperTerminal program can be used as it is included as a part of the operating system. After a new connection is opened, by going to CONFIGURE the parameters on the computer side can be set as per the settings on the robot side as shown in Figure A.3.



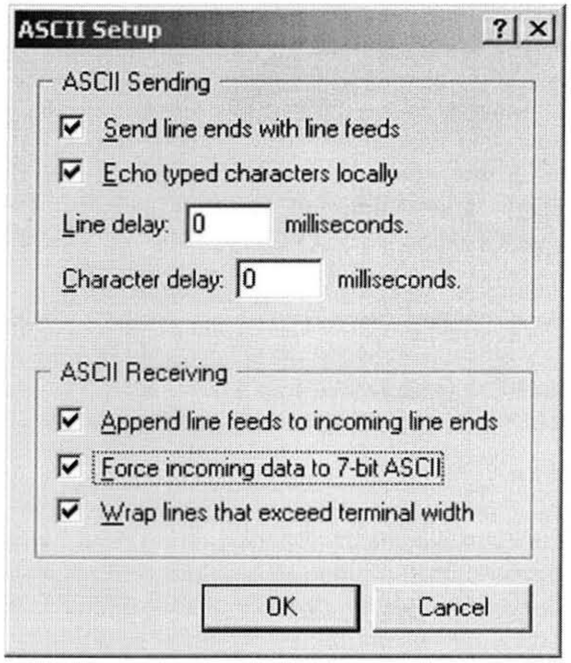
COM1 Properties

Port Settings

Bits per second: 9600
Data bits: 8
Parity: None
Stop bits: 1
Flow control: Hardware

Figure A.3 Communication port 1 settings.

Now to set the other parameters go to PROPERTIES → SETTINGS, leave the EMULATION to auto detect and go into ASCII set up as shown in Figure A.4.



ASCII Setup

ASCII Sending

- [✓] Send line ends with line feeds
- [✓] Echo typed characters locally

Line delay: 0 milliseconds.

Character delay: 0 milliseconds.

ASCII Receiving

- [✓] Append line feeds to incoming line ends
- [✓] Force incoming data to 7-bit ASCII
- [✓] Wrap lines that exceed terminal width

Figure A.4 ASCII settings for communication port.

Important Points about ASCII set up

- The DARL protocol for communication transmits all transmission blocks other than a STOP character with a Carriage Return (CR) and a Line Feed (LF)
- An Enter Key on the Keyboard would automatically do that, if 'Send line with line feed' is enabled.
- 'Echo type characters locally' will display the characters on the screen being typed and outputted.
- 'Enforce the incoming data to 7 –Bit ASCII', This is used because the Robot's Keyboard is fully expressed in 128 bit characters of ASCII so we require $2^7 = 128$ bits
- The Character length is set to 8 bits so that all the bits are '0's and contribute towards a START BIT.

The program is as given below:

In EDIT mode:

```
E
------------------------------------------------
NEW
COMM 1 78: IUNIT 1: OUNIT 1
END
QUIT
M
-----------------------------------------------------
START {Press Start}
```

The LCD is as good as numb and one can see "0M>" on the HyperTerminal Screen. Now the computer is the new I/O terminal.

4. Specifications of the Fotonic sensor:

The plug in module used is MTI-3802 with the probe outer diameter of 504um and inner core diameter of 150um. The Figure A.5 shows the MTI instruments Fotonic sensor gap calibration chart.

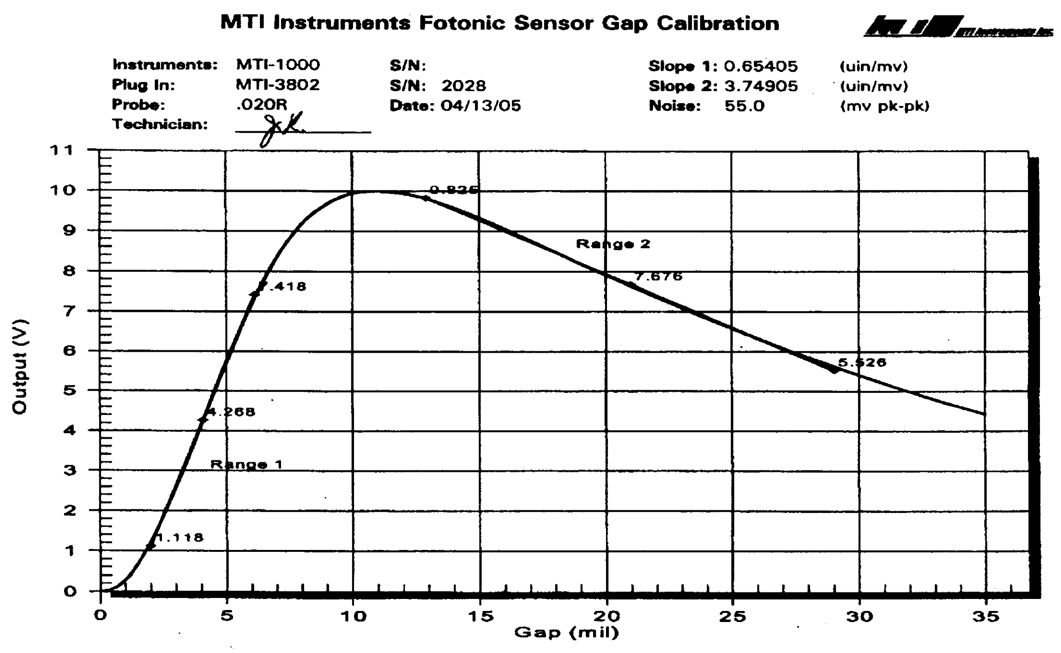


Figure A.5 MTI 1000 Fotonic Sensor characteristics.

APPENDIX B

SOURCE CODE

The automatic spotting sequence method explained in the thesis is implemented using Measurement Studio Lab Windows/ CVI of National Instruments. The following is the source code for the entire process.

```
/*------------------------------------------------------------------------*/
/*                                                                      */
/*  FILE:      spotting.c                                               */
/*                                                                      */
/*  PURPOSE:   Automate the SPOTTING experiment                         */
/*                                                                      */
/*  NOTES:     Make sure all serial cables are connected properly.      */
/*                                                                      */
/*------------------------------------------------------------------------*/

/*------------------------------------------------------------------------*/
/*   Include Files                                                      */
/*------------------------------------------------------------------------*/
#include <windows.h>
#include "daq_num.h"
#include <analysis.h>
#include "daqchart.h"
#include <ansi_c.h>
#include <formatio.h>
#include <rs232.h>
#include <utility.h>
#include <cvirte.h>
#include <userint.h>
#include "spotting.h"

/*------------------------------------------------------------------------*/
/*   Constants                                                          */
/*------------------------------------------------------------------------*/
#define    ROBEOL                   '>'
#define    ROBLIMGAP                  0.2
#define    ROBCMDLEN                  512
#define    ZABUNIT                  (unsigned char)1
#define    ZABCNT                   6
#define    ZABCONVFACT                  0.09921875
#define    ZABCMDLEN                  512
#define    SENSRNGGAP                   0.2

#define    SNESDAQNUMIDXLIM                 2000
#define    SENSDAQNUMRATE                   50.00
#define    SENSDAQNUMINT                  (1.0/SENSDAQNUMRATE)
#define    SENSSPOTTESTTH                 5
```

```
/*-----------------------------------------------------------------------*/
/*  Local Function Prototypes                                            */
/*-----------------------------------------------------------------------*/
void            _MyInitApp (void);
void            _MyExitApp (void);
int             _MyInitExp (void);
void            _DispRS232Err (int _iPortIdx);
void            _SendRobCmd (int _iCmdType, double _dX, double _dY,
                        double _dZ);
DWORD WINAPI    _RobCommMonProc (LPVOID _lpParam);
void CVICALLBACK    _RobRplyEOLFunc (int _iPortNum, int _iEvntMask,
                        void * _pvCallBackData);
void            _RobGoHomeFunc (void);
void            _RobGo2WellFunc (void);
void            _RobGo21stSpotFunc (void);
void            _RobJogXYZFunc (int _iType);
void            _SendZabCmd (unsigned char _cCmdType, long int* _plZ);
DWORD WINAPI    _ZabCommMonProc (LPVOID _lpParam);
void CVICALLBACK    _ZabRplyEOLFunc (int _iPortNum, int _iEvntMask,
                        void * _vpCallBackData);
void            _ZabJogZFunc (int _iType, long int _lStep);
void            _ZabGoHomeFunc (void);
void            _GetCoorVal (char* _strVal, double* _pdX, double* _pdY,
                      double* _pdZ);
DWORD WINAPI    _AutoExpMonProc1 (LPVOID _lpParam);
DWORD WINAPI    _AutoExpMonProc2 (LPVOID _lpParam);
int             _TestSpotForm (void);
void            _EmgyProc (void);

/*-----------------------------------------------------------------------*/
/*  File Static Variables                                                */
/*-----------------------------------------------------------------------*/
static          int             iPnlHdl;

/*-----------------------------------------------------------------------*/
/*  Volatile Variables                                                   */
/*-----------------------------------------------------------------------*/
double          dRobWell[3];
double          dRob1stSpot[3];
double          dRobLim[3];
double          dRobStep[3];
double          dRobTarg[3];
double          dRobCurr[3];
unsigned char     strRobCmd[ROBCMDLEN];
unsigned char     strRobRply[ROBCMDLEN];
unsigned char     strTempRobWrt[ROBCMDLEN];
unsigned char     strTempRobRd[ROBCMDLEN];

long int        lZabLowLim,lZabUppLim;
long int        lZabStep;
long int          lZabTarg;
long int          lZabTargTemp;
long int          lZabTargTemp1;
long int          lZabCurr;
long int          lZabCurrTemp;
char              strZabCmd[ZABCMDLEN];
char          strZabRply[ZABCMDLEN];
char         strTempZabWrt[ZABCMDLEN];
char         strTempZabRd[ZABCMDLEN];

unsigned short int  iCurrRow, iCurrCol;
unsigned short int        iMaxRow, iMaxCol;
unsigned short int        iSpotNum;

double         dCurrV;

char           bEmgSwch;

int          iCmdIdx[2];
int          iPortOpen[2];
int               iCommErr[2];
int               iDevErr[2];
int               iCommPort[2];
int               iBaudRate[2];
int               iPortIdx[2];
int               iParity[2];
int               iDataBits[2];
int               iStopBits[2];
int               iInptQ[2];
int               iOuptQ[2];
int               iXMode[2];
int               iCTSMode[2];
int               iStrSize[2];
int               iBytsSent[2];
int               iBytsRead[2];
int               iBrkStat[2];
int               iCommStat[2];
int               iInQLen[2];
int               iOutQLen[2];

double         dTimeOut[2];

HANDLE            hCommStartEvnt[2];
HANDLE            hCommEOLEvnt[2];
HANDLE            hCommEndEvnt[2];
HANDLE            hCommThrd[2];
DWORD             dwCommThrdID[2];

char         strRobPortName[30];
char         strZabPortName[30];

char         strErrMsg[200];

int          iSensDAQNumID;
int          iSensDAQStartRec;
int          iSensDAQDataIdx;
double         dSensDAQRaw[SNESDAQNUMIDXLIM];
double         dSensDAQData[SNESDAQNUMIDXLIM];
double         dSensDAQTime[SNESDAQNUMIDXLIM];

int          iSensDAQPosDataIdx1, iSensDAQPosDataIdx2;
int          iSensDAQPosDataIdx;
```

```
double          dSensDAQPosData[SNESDAQNUMIDXLIM];
double          dSensDAQPos[SNESDAQNUMIDXLIM];
int             iSensDAQFiltMaxIdx,iSensDAQFiltMinIdx;
double          dSensDAQFiltMax, dSensDAQFiltMin;
int             iSensDAQFiltIdx;
double          dSensDAQFiltData;
double          dSensDAQFilt[10];

HANDLE          hExpStartEvnt[2];
HANDLE          hExpFuncThrd[2];
DWORD           dwExpFuncThrdID[2];

int             iFileHdl;
int             iFileStat;
char            strFileName[64];

int             iWhenToPickRef;
int             iCurvRefIdx;


/*------------------------------------------------------------------------*/
/*   Main procedure                                                       */
/*------------------------------------------------------------------------*/
int main (int _argc, char *_argv[])
{
   /* Initialize variables */
          _MyInitApp ();
          if (InitCVIRTE (0, _argv, 0) == 0)
                 return -1;        /* out of memory */
          if ((iPnlHdl = LoadPanel (0, "spotting.uir", PANEL)) < 0)
                 return -1;
          DisplayPanel (iPnlHdl);
          RunUserInterface ();
          DiscardPanel (iPnlHdl);
          /* Release occupied resources */
          _MyExitApp ();
          return 0;
}

/*------------------------------------------------------------------------*/
/*   Initialize Variables                                                 */
/*------------------------------------------------------------------------*/
void _MyInitApp (void)
{
   int          i_idx;

   for (i_idx = 0; i_idx < 2; i_idx++) {
      iPortOpen[i_idx]        = 0;
      hCommStartEvnt[i_idx]   = NULL;
      hCommEOLEvnt[i_idx]     = NULL;
      hCommEndEvnt[i_idx]     = NULL;
      hCommThrd[i_idx]        = NULL;

      hExpStartEvnt[i_idx]    = NULL;
                 hExpFuncThrd[i_idx]     = NULL;
                 dwExpFuncThrdID[i_idx]  = 0;
   }

   iSensDAQNumID      = 0;
   iSensDAQDataIdx    = 0;
   iSensDAQStartRec   = 0;
   dSensDAQTime[0]    = 0.0;
   iSensDAQFiltIdx    = 0;
}

/*------------------------------------------------------------------------*/
/*   Uninitialize Variables                                               */
/*------------------------------------------------------------------------*/
int CVICALLBACK _QuitApp(int _iPnlHdl, int _iCtrlHdl, int _iEvntHdl,
                    void *_pvCallBackData, int _iEvntData1,
                    int _iEvntData2)
{
        switch (_iEvntHdl)
                {
                case EVENT_COMMIT:
                        QuitUserInterface (0);
                        break;
                }
        return 0;
}

void _MyExitApp (void)
{
   int          i_idx;

   for (i_idx = 0; i_idx < 2; i_idx++) {
                if (hCommStartEvnt[i_idx] != NULL) {
                        CloseHandle (hCommStartEvnt[i_idx]);
                } /* END IF */
                if (iPortOpen[i_idx]) {
        iOutQLen[i_idx] = GetOutQLen (iCommPort[i_idx]);
        if (iOutQLen[i_idx] > 0) {
           MessagePopup ("RS232 Message", "The output queue has\n"
                        "data in it. Wait for device to receive\n"
                        "the data or flush the queue.\n");
           break;
        } /* END IF */
        iCommErr[i_idx] = CloseCom (iCommPort[i_idx]);
        if (iCommErr[i_idx]) {
           _DispRS232Err (i_idx);
        } /* END IF */
      } /* END IF */
   } /* ENF FOR */
}

/*------------------------------------------------------------------------*/
/*   Initialize Experiment                                                */
/*------------------------------------------------------------------------*/
int CVICALLBACK _InitializeExp(int _iPnlHdl, int _iCtrlHdl, int _iEvntHdl,
                        void *_pvCallBackData, int _iEvntData1,
                        int _iEvntData2)
{
```

```
          switch (_iEvntHdl) {
                    case EVENT_COMMIT:
         _MyInitExp ();
                              break;
    }
          return 0;
}

int _MyInitExp (void)
{
   SetCtrlVal (iPnlHdl, PANEL_EXPMSG1, "INITIALIZING ...");
   SetCtrlVal (iPnlHdl, PANEL_EXPMSG2, "");

          if (iPortOpen[0] == 0) {
                    iCommPort[0]        = 1;
                    strRobPortName[0]   = '\0';
                    strcpy (strRobPortName, "COM1");
                    iBaudRate[0]        = 9600;
                    iParity[0]          = 0;
                    iDataBits[0]        = 8;
                    iStopBits[0]        = 1;
                    iInptQ[0]           = 512;
                    iOuptQ[0]           = 512;
                    DisableBreakOnLibraryErrors ();
                    iCommErr[0]         = OpenComConfig (iCommPort[0], strRobPortName,
                                                                                  iBaudRate[0],
iParity[0],
                                                                                  iDataBits[0],
iStopBits[0],
                                                                                  iInptQ[0],
          iOuptQ[0]);
                    EnableBreakOnLibraryErrors ();
                    if (iCommErr[0]) {
                              _DispRS232Err (0);
                              return -1;
                    }
                    iPortOpen[0]        = 1;
                    FlushInQ (iCommPort[0]);
      FlushOutQ (iCommPort[0]);

      /* Allow for only one installation */
      InstallComCallback (iCommPort[0], LWRS_RXFLAG, 0,
                                        (int)ROBEOL,
                                        _RobRplyEOLFunc, 0);
      if (hCommStartEvnt[0] == NULL) {
         hCommStartEvnt[0] = CreateEvent (NULL, TRUE, TRUE, NULL);
         if (hCommStartEvnt[0] == NULL) {
            Fmt (strErrMsg, "%s<CreateEvent failed: (%i)\n", GetLastError());
            MessagePopup ("CreateEvent Message", strErrMsg);
            return -1;
         }
         ResetEvent (hCommStartEvnt[0]);
      }
      if (hCommEOLEvnt[0] == NULL) {
         hCommEOLEvnt[0] = CreateEvent (NULL, TRUE, TRUE, NULL);
         if (hCommEOLEvnt[0] == NULL) {
            Fmt (strErrMsg, "%s<CreateEvent failed: (%i)\n", GetLastError());
            MessagePopup ("CreateEvent Message", strErrMsg);
            return -1;
         }
         ResetEvent (hCommEOLEvnt[0]);
      }
      if (hCommEndEvnt[0] == NULL) {
         hCommEndEvnt[0] = CreateEvent (NULL, TRUE, TRUE, NULL);
         if (hCommEndEvnt[0] == NULL) {
            Fmt (strErrMsg, "%s<CreateEvent failed: (%i)\n", GetLastError());
            MessagePopup ("CreateEvent Message", strErrMsg);
            return -1;
         }
         SetEvent (hCommEndEvnt[0]);
      }

      if (hCommThrd[0] == NULL) {
         hCommThrd[0] = CreateThread (0,0, _RobCommMonProc,
                                                        0,0, &dwCommThrdID[0]);
         if (hCommThrd[0] == NULL) {
            Fmt (strErrMsg, "%s<CreateThread failed: (%i)\n", GetLastError());
            MessagePopup ("CreateThreadt Message", strErrMsg);
            return -1;
         }
      }
           /*SetXMode (iCommPort[0], iXMode[0]);
                 SetCTSMode (iCommPort[0], iCTSMode[0]);
                 SetComTime (iCommPort[0], dTimeOut[0]); */
       }

       if (iPortOpen[1] == 0)
       {
                 iCommPort[1]       = 2;
                 strZabPortName[0]  = '\0';
                 strcpy (strZabPortName, "COM4");
                 iBaudRate[1]       = 9600;
                 iParity[1]         = 0;
                 iDataBits[1]       = 8;
                 iStopBits[1]       = 1;
                 iInptQ[1]          = 512;
                 iOuptQ[1]          = 512;
                 DisableBreakOnLibraryErrors ();
                 iCommErr[1]        = OpenComConfig (iCommPort[1], strZabPortName,
                                                                  iBaudRate[1],
iParity[1],
                                                                  iDataBits[1],
iStopBits[1],
                                                                  iInptQ[1],
       iOuptQ[1]);
                 EnableBreakOnLibraryErrors ();
                 if (iCommErr[1]) {
                         _DispRS232Err (1);
                         return -1;
                 }
                 iPortOpen[1] = 1;
                 FlushInQ (iCommPort[1]);
```

```
    FlushOutQ (iCommPort[1]);

    /* Allow for only one installation */
    InstallComCallback (iCommPort[1], LWRS_RECEIVE, ZABCNT,
                                                  0,
                                                  _ZabRplyEOLFunc, 0);
    if (hCommStartEvnt[1] == NULL) {
       hCommStartEvnt[1] = CreateEvent (NULL, TRUE, TRUE, NULL);
       if (hCommStartEvnt[1] == NULL) {
          Fmt (strErrMsg, "%s<CreateEvent failed: (%i)\n", GetLastError());
          MessagePopup ("CreateEvent Message", strErrMsg);
          return -1;
       }
       ResetEvent (hCommStartEvnt[1]);
    }
    if (hCommEOLEvnt[1] == NULL) {
       hCommEOLEvnt[1] = CreateEvent (NULL, TRUE, TRUE, NULL);
       if (hCommEOLEvnt[1] == NULL) {
          Fmt (strErrMsg, "%s<CreateEvent failed: (%i)\n", GetLastError());
          MessagePopup ("CreateEvent Message", strErrMsg);
          return -1;
       }
       ResetEvent (hCommEOLEvnt[0]);
    }
    if (hCommEndEvnt[1] == NULL) {
       hCommEndEvnt[1] = CreateEvent (NULL, TRUE, TRUE, NULL);
       if (hCommEndEvnt[1] == NULL) {
          Fmt (strErrMsg, "%s<CreateEvent failed: (%i)\n", GetLastError());
          MessagePopup ("CreateEvent Message", strErrMsg);
          return -1;
       }
       SetEvent (hCommEndEvnt[1]);
    }
    if (hCommThrd[1] == NULL) {
       hCommThrd[1] = CreateThread (0,0, _ZabCommMonProc,
                                                      0, 0, &dwCommThrdID[1]);
       if (hCommThrd[1] == NULL) {
          Fmt (strErrMsg, "%s<CreateThread failed: (%i)\n", GetLastError());
          MessagePopup ("CreateThread Message", strErrMsg);
          return -1;
       }
    }

       }
       /* Read Robot Position */
       SetCtrlVal (iPnlHdl, PANEL_ROBMSG, "COMM1 ...");
strRobCmd[0] = '\0';
ResetEvent (hCommStartEvnt[0]);
ResetEvent (hCommEOLEvnt[0]);
_SendRobCmd (3, 0.0, 0.0, 0.0);
SetEvent (hCommStartEvnt[0]);

/* Read Zaber Position */
SetCtrlVal (iPnlHdl, PANEL_ZABMSG, "COMM4 ...");
iCmdIdx[1] = 0;
ResetEvent (hCommStartEvnt[1]);
```

```
   ResetEvent (hCommEOLEvnt[1]);
   _SendZabCmd (60,NULL);
   SetEvent (hCommStartEvnt[1]);

   if (iSensDAQNumID == 0) {
      iSensDAQNumID = DAQ_Numeric_ConvertFromNumeric (iPnlHdl,
                                          PANEL_SENSCURRV,
                                          "Sensor");
      DAQ_Numeric_SetAttribute (iPnlHdl, iSensDAQNumID,
                          ATTR_DAQ_NUMERIC_SCAN_FREQUENCY,
                          SENSDAQNUMRATE);
   }

         return 0;
}

/*-------------------------------------------------------------------------*/
/* Display error information to the user.                              */
/*-------------------------------------------------------------------------*/
void _DispRS232Err (int _iPortIdx)
{
   switch (iCommErr[_iPortIdx])
      {
      default :
         if (iCommErr[_iPortIdx] < 0)
            {
            Fmt (strErrMsg, "%s<RS232 error number %i", iCommErr[_iPortIdx]);
            MessagePopup ("RS232 Message", strErrMsg);
            }
         break;
      case 0  :
         MessagePopup ("RS232 Message", "No errors.");
         break;
      case -2 :
         Fmt (strErrMsg, "%s", "Invalid port number (must be in the "
                         "range 1 to 8).");
         MessagePopup ("RS232 Message", strErrMsg);
         break;
      case -3 :
         Fmt (strErrMsg, "%s", "No port is open.\n"
            "Check COM Port setting in Configure.");
         MessagePopup ("RS232 Message", strErrMsg);
         break;
      case -99 :
         Fmt (strErrMsg, "%s", "Timeout error.\n\n"
            "Either increase timeout value,\n"
            "      check COM Port setting, or\n"
            "      check device.");
         MessagePopup ("RS232 Message", strErrMsg);
         break;
      }
}

/*-------------------------------------------------------------------------*/
/*   Prepare the command string to Robot                              */
/*                                                         */
/* iCmdType = 1 : HOME                                         */
/* iCmdType = 2 : MOVE                                         */
/* iCmdType = 3 : DISP                                        */
/*--------------------------------------------------------------------------*/
void _SendRobCmd (int _iCmdType, double _dX, double _dY, double _dZ)
{
        switch (_iCmdType)
        {
        case 1:
                strcpy (strTempRobWrt, "HOME\r\nDISP\r\nQUIT\r\n");
                strcat (strRobCmd, strTempRobWrt);
                break;
        case 2:
                strcpy (strTempRobWrt, "DO CLEAR\r\n");
                strcat (strRobCmd, strTempRobWrt);
                sprintf(strTempRobWrt,"DO T1=%.2f %.2f %.2f %.2f\r\n"
                                ,_dX, _dY, _dZ, 0.0);
                strcat (strRobCmd, strTempRobWrt);
                strcat (strRobCmd, "DO MOVE T1\r\nDISP\r\nQUIT\r\n");
                break;
        case 3:
           strcpy (strTempRobWrt, "DISP\r\nQUIT\r\n");
                strcat (strRobCmd, strTempRobWrt);
                break;
        default:
           break;
        }
}

/*--------------------------------------------------------------------------*/
/*   Thread for controlling communication with Robot                       */
/*                                                   */
/* hCommStartEvnt[0] control the start of communication                    */
/* hCommEOLEvnt[0]   control the end of communication                      */
/*--------------------------------------------------------------------------*/
DWORD WINAPI _RobCommMonProc (LPVOID _lpParam)
{
        char*       pstr_temp;
        char*       pstr_temp1;
        char        str_temp[50];
        int         i_idx;

        while (iPortOpen[0] == 1)
        {
                WaitForSingleObject (hCommStartEvnt[0], INFINITE);

                pstr_temp = strRobCmd;

                while (*pstr_temp != '\0')
                {
                        FlushInQ (iCommPort[0]);
                        FlushOutQ (iCommPort[0]);
                        ResetEvent (hCommEOLEvnt[0]);
                        i_idx = strcspn (pstr_temp, "\n") + 1;
                        str_temp[0] = '\0';
                        strncat (str_temp, pstr_temp, i_idx);
```

```
                    pstr_temp += i_idx;
                    pstr_temp1 = strstr (str_temp, "T1=");
                    if (pstr_temp1 != NULL) {
                       pstr_temp1 += 3;
                       _GetCoorVal (pstr_temp1, &dRobTarg[0], &dRobTarg[1],
                                   &dRobTarg[2]);
             SetCtrlVal (iPnlHdl, PANEL_ROBTARGX, dRobTarg[0]);
             SetCtrlVal (iPnlHdl, PANEL_ROBTARGY, dRobTarg[1]);
             SetCtrlVal (iPnlHdl, PANEL_ROBTARGZ, dRobTarg[2]);
          }
                    iStrSize[0] = StringLength (str_temp);
                    iBytsSent[0] = ComWrt (iCommPort[0], str_temp, iStrSize[0]);
                    WaitForSingleObject (hCommEOLEvnt[0], INFINITE);
             }

             SetCtrlVal (iPnlHdl, PANEL_EXPMSG2, "DONE!");
             SetCtrlVal (iPnlHdl, PANEL_ROBMSG, "OK.");
             ResetEvent (hCommStartEvnt[0]);
             SetEvent (hCommEndEvnt[0]);
        }
        return 1;
}

/*--------------------------------------------------------------------------*/
/*   Callback function to deal with the ending of communication            */
/*                                                                          */
/*  According to the communication protocol, when '>' letter is received,   */
/*  one communication is over.                                              */
/*--------------------------------------------------------------------------*/
void CVICALLBACK _RobRplyEOLFunc (int _iPortNum, int _iEvntMask,
                          void * _pvCallBackData)
{

        char*       pstr_temp_m;
        char*       pstr_temp_d;
        char*       pstr_temp;
        int         i_idx;

        strTempRobRd[0] = '\0';

        ComRd (iCommPort[0], strTempRobRd, GetInQLen(iCommPort[0]));
        FlushInQ (iCommPort[0]);
        strcat (strRobRply, strTempRobRd);

        pstr_temp_m = strstr (strTempRobRd, "M>\r\n");
        pstr_temp_d = strstr (strTempRobRd, "D>\r\n");

        if (pstr_temp_d != NULL) {
           pstr_temp = strstr (strRobRply, "(XYZA)");
           if (pstr_temp != NULL) {
              i_idx = strcspn (pstr_temp, "\n") + 1;
        pstr_temp += i_idx;
              _GetCoorVal (pstr_temp, &dRobCurr[0], &dRobCurr[1], &dRobCurr[2]);
              SetCtrlVal (iPnlHdl, PANEL_ROBCURRX, dRobCurr[0]);
              SetCtrlVal (iPnlHdl, PANEL_ROBCURRY, dRobCurr[1]);
              SetCtrlVal (iPnlHdl, PANEL_ROBCURRZ, dRobCurr[2]);
```

```
                }
            }

            if ((pstr_temp_m != NULL) || (pstr_temp_d != NULL))
            {
                strRobRply[0] = '\0';
                SetEvent (hCommEOLEvnt[0]);
            }
}

/*----------------------------------------------------------------------*/
/*   Callback functions to HOME the Robot                               */
/*----------------------------------------------------------------------*/
int  CVICALLBACK _RobGoHome(int _iPnlHdl, int _iCtrlHdl, int _iEvntHdl,
                            void *_pvCallBackData, int _iEvntData1,
                            int _iEvntData2)
{
            switch (_iEvntHdl) {
                        case EVENT_COMMIT:
                                    _RobGoHomeFunc ();
                              break;
   }
            return 0;
}

void _RobGoHomeFunc (void)
{
            dRobTarg[0] = 0.0;
            dRobTarg[1] = 0.0;
            dRobTarg[2] = 0.0;
            SetCtrlVal (iPnlHdl, PANEL_ROBTARGX, dRobTarg[0]);
            SetCtrlVal (iPnlHdl, PANEL_ROBTARGY, dRobTarg[1]);
            SetCtrlVal (iPnlHdl, PANEL_ROBTARGZ, dRobTarg[2]);
            SetCtrlVal (iPnlHdl, PANEL_EXPMSG1, "ROBOT GOING HOME...");
            SetCtrlVal (iPnlHdl, PANEL_EXPMSG2, "");
            SetCtrlVal (iPnlHdl, PANEL_ROBMSG, "COMM1 ...");
   strRobCmd[0] = '\0';
            ResetEvent (hCommStartEvnt[0]);
            ResetEvent (hCommEOLEvnt[0]);
            _SendRobCmd (1, 0.0, 0.0, 0.0); /* HOME */
            SetEvent (hCommStartEvnt[0]);
}
/*----------------------------------------------------------------------*/
/*   Callback functions to move the Robot to liquid well                */
/*----------------------------------------------------------------------*/
int  CVICALLBACK _GoToWell(int _iPnlHdl, int _iCtrlHdl, int _iEvntHdl,
                         void *_pvCallBackData, int _iEvntData1,
                         int _iEvntData2)
{
            switch (_iEvntHdl) {
                        case EVENT_COMMIT:
                           _RobGo2WellFunc ();
                                 break;
   }
            return 0;
}
```

```
void _RobGo2WellFunc (void)
{
        SetCtrlVal (iPnlHdl, PANEL_EXPMSG1, "ROBOT GOING TO WELL...");
        SetCtrlVal (iPnlHdl, PANEL_EXPMSG2, "");
        SetCtrlVal (iPnlHdl, PANEL_ROBMSG, "COMM1 ...");
        GetCtrlVal (iPnlHdl, PANEL_WELLX, &dRobWell[0]);
        GetCtrlVal (iPnlHdl, PANEL_WELLY, &dRobWell[1]);
        GetCtrlVal (iPnlHdl, PANEL_ROBLIMITZ, &dRobLim[2]);
        dRobWell[2] = dRobLim[2] + ROBLIMGAP;
        SetCtrlVal (iPnlHdl, PANEL_WELLZ, dRobWell[2]);
        strRobCmd[0] = '\0';
        ResetEvent (hCommStartEvnt[0]);
        ResetEvent (hCommEOLEvnt[0]);
        _SendRobCmd (2, dRobWell[0], dRobWell[1], dRobCurr[2]);
        _SendRobCmd (2, dRobWell[0], dRobWell[1], dRobWell[2]);
        SetEvent (hCommStartEvnt[0]);
}

/*--------------------------------------------------------------------*/
/*   Callback functions to move the Robot to the 1st spot              */
/*--------------------------------------------------------------------*/
int  CVICALLBACK _Go1stSpot(int _iPnlHdl, int _iCtrlHdl, int _iEvntHdl,
                       void *_pvCallBackData, int _iEvntData1,
                       int _iEvntData2)
{
        switch (_iEvntHdl) {
                case EVENT_COMMIT:
                        _RobGo21stSpotFunc ();
        break;
   }
        return 0;
}

void _RobGo21stSpotFunc (void)
{
        SetCtrlVal (iPnlHdl, PANEL_EXPMSG1, "ROBOT GOING TO THE 1ST SPOT...");
        SetCtrlVal (iPnlHdl, PANEL_EXPMSG2, "");
        SetCtrlVal (iPnlHdl, PANEL_ROBMSG, "COMM1 ...");
        GetCtrlVal (iPnlHdl, PANEL_1STSPOTX, &dRob1stSpot[0]);
        GetCtrlVal (iPnlHdl, PANEL_1STSPOTY, &dRob1stSpot[1]);
        GetCtrlVal (iPnlHdl, PANEL_ROBLIMITZ, &dRobLim[2]);
        dRob1stSpot[2] = dRobLim[2] + ROBLIMGAP;
        SetCtrlVal (iPnlHdl, PANEL_1STSPOTZ, dRob1stSpot[2]);
        strRobCmd[0] = '\0';
        ResetEvent (hCommStartEvnt[0]);
        ResetEvent (hCommEOLEvnt[0]);
        _SendRobCmd (2, dRob1stSpot[0], dRob1stSpot[1], dRobCurr[2]);
        _SendRobCmd (2, dRob1stSpot[0], dRob1stSpot[1], dRob1stSpot[2]);
        SetEvent (hCommStartEvnt[0]);
}

/*--------------------------------------------------------------------*/
/*   Callback functions to Form one spot                             */
/*--------------------------------------------------------------------*/
int  CVICALLBACK _FormSpot(int _iPnlHdl, int _iCtrlHdl, int _iEvntHdl,
                        void *_pvCallBackData, int _iEvntData1,
                        int _iEvntData2)
{
           switch (_iEvntHdl) {
                      case EVENT_COMMIT:

                                 break;
   }
           return 0;
}

/*----------------------------------------------------------------------*/
/*   Callback functions to move to next column                        */
/*----------------------------------------------------------------------*/
int  CVICALLBACK _GoNextCol(int _iPnlHdl, int _iCtrlHdl, int _iEvntHdl,
                        void *_pvCallBackData, int _iEvntData1,
                        int _iEvntData2)
{
           switch (_iEvntHdl) {
                      case EVENT_COMMIT:
          _RobJogXYZFunc (5);
                                 break;
   }
           return 0;
}

/*----------------------------------------------------------------------*/
/*   Callback functions to move to next row                          */
/*----------------------------------------------------------------------*/
int  CVICALLBACK _GoNextRow(int _iPnlHdl, int _iCtrlHdl, int _iEvntHdl,
                        void *_pvCallBackData, int _iEvntData1,
                        int _iEvntData2)
{
           switch (_iEvntHdl) {
                      case EVENT_COMMIT:
          _RobJogXYZFunc (7);
                                 break;
   }
           return 0;
}

/*----------------------------------------------------------------------*/
/*   Callback functions to jog the Robot                            */
/*                                                    */
/*  Along X-axis                                           */
/*----------------------------------------------------------------------*/
int  CVICALLBACK _RobJogMinuX(int _iPnlHdl, int _iCtrlHdl, int _iEvntHdl,
                           void *_pvCallBackData, int _iEvntData1,
                           int _iEvntData2)
{
           switch (_iEvntHdl) {
                      case EVENT_COMMIT:
                                 _RobJogXYZFunc (1);
                                 break;
   }
           return 0;
```

```
}

int  CVICALLBACK _RobJogPlusX(int _iPnlHdl, int _iCtrlHdl, int _iEvntHdl,
                             void *_pvCallBackData, int _iEvntData1,
                             int _iEvntData2)
{
        switch (_iEvntHdl) {
                case EVENT_COMMIT:
        _RobJogXYZFunc (4);
                        break;
  }
        return 0;
}

/*----------------------------------------------------------------------*/
/*   Callback functions to jog the Robot                                */
/*                                                         */
/*  Along Y-axis                                                  */
/*----------------------------------------------------------------------*/
int  CVICALLBACK _RobJogMinuY(int _iPnlHdl, int _iCtrlHdl, int _iEvntHdl,
                             void *_pvCallBackData, int _iEvntData1,
                             int _iEvntData2)
{
        switch (_iEvntHdl) {
                case EVENT_COMMIT:
        _RobJogXYZFunc (2);
                        break;
  }
        return 0;
}

int  CVICALLBACK _RobJogPlusY(int _iPnlHdl, int _iCtrlHdl, int _iEvntHdl,
                             void *_pvCallBackData, int _iEvntData1,
                             int _iEvntData2)
{
        switch (_iEvntHdl) {
                case EVENT_COMMIT:
        _RobJogXYZFunc (5);
                        break;
  }
        return 0;
}

/*----------------------------------------------------------------------*/
/*   Callback functions to jog the Robot                                */
/*                                                         */
/*  Along Z-axis                                                  */
/*----------------------------------------------------------------------*/
int  CVICALLBACK _RobJogMinuZ(int _iPnlHdl, int _iCtrlHdl, int _iEvntHdl,
                             void *_pvCallBackData, int _iEvntData1,
                             int _iEvntData2)
{
        switch (_iEvntHdl) {
                case EVENT_COMMIT:
        _RobJogXYZFunc (3);
        break;
```

```
   }
         return 0;
}

int  CVICALLBACK _RobJogPlusZ(int _iPnlHdl, int _iCtrlHdl, int _iEvntHdl,
                        void *_pvCallBackData, int _iEvntData1,
                        int _iEvntData2)
{
         switch (_iEvntHdl) {
                  case EVENT_COMMIT:
         _RobJogXYZFunc (6);
                           break;
   }
         return 0;
}

/*------------------------------------------------------------------------*/
/* iType = 1 : Minus X              = 4 : Plus X                       */
/*      = 2 : Minus Y               = 5 : Plus Y, Next Column                  */
/*      = 3 : Minus Z               = 6 : Plus Z                       */
/*      = 7 : Next Row        = 8 : Get up                       */
/*------------------------------------------------------------------------*/
void _RobJogXYZFunc (int _iType)
{
         GetCtrlVal (iPnlHdl, PANEL_ROBSTEPX, &dRobStep[0]);
         GetCtrlVal (iPnlHdl, PANEL_ROBSTEPY, &dRobStep[1]);
         GetCtrlVal (iPnlHdl, PANEL_ROBSTEPZ, &dRobStep[2]);
         GetCtrlVal (iPnlHdl, PANEL_1STSPOTY, &dRob1stSpot[1]);
         SetCtrlVal (iPnlHdl, PANEL_ROBMSG, "COMM1 ...");

         switch (_iType) {
         case 1: /* Minus X */
                  dRobTarg[0] = dRobCurr[0] - dRobStep[0];
      dRobTarg[1] = dRobCurr[1];
      dRobTarg[2] = dRobCurr[2];
                  break;
         case 2: /* Minus Y */
                  dRobTarg[0] = dRobCurr[0];
      dRobTarg[1] = dRobCurr[1] - dRobStep[1];
      dRobTarg[2] = dRobCurr[2];
                  break;
         case 3: /* Minus Z */
                  dRobTarg[0] = dRobCurr[0];
      dRobTarg[1] = dRobCurr[1];
      dRobTarg[2] = dRobCurr[2] - dRobStep[2];
                  break;
         case 4: /* Plus X */
                  dRobTarg[0] = dRobCurr[0] + dRobStep[0];
      dRobTarg[1] = dRobCurr[1];
      dRobTarg[2] = dRobCurr[2];
                  break;
         case 5: /* Plus Y or Nex Column */
                  dRobTarg[0] = dRobCurr[0];
      dRobTarg[1] = dRobCurr[1] + dRobStep[1];
      dRobTarg[2] = dRobCurr[2];
                  break;
```

```
            case 6: /* Plus Z */
                      dRobTarg[0] = dRobCurr[0];
        dRobTarg[1] = dRobCurr[1];
        dRobTarg[2] = dRobCurr[2] + dRobStep[2];
                      break;
            case 7: /* Next Row */
                      dRobTarg[0] = dRobCurr[0] + dRobStep[0];
        dRobTarg[1] = dRob1stSpot[1];
        dRobTarg[2] = dRobCurr[2];
                      break;
            case 8: /* Get up */
                      dRobTarg[0] = dRobCurr[0];
        dRobTarg[1] = dRobCurr[1];
        dRobTarg[2] = dRobCurr[2] + 50.0;
                      break;
    case 9: /* Elongation */
        GetCtrlVal (iPnlHdl, PANEL_ROBLIMITZ, &dRobLim[2]);
        dRobTarg[0] = dRobCurr[0];
        dRobTarg[1] = dRobCurr[1];
        dRobTarg[2] = dRobLim[2] + ROBLIMGAP;
        break;
            default:
                      break;
            }
    strRobCmd[0] = '\0';
    GetCtrlVal (iPnlHdl, PANEL_ROBSTEPX, &dRobStep[0]);
    if (dRobTarg[2] > 0.0) {
                dRobTarg[2] = 0.0;
            }
    if (dRobTarg[2] < dRobLim[2]) {
                dRobTarg[2] = dRobLim[2];
            }
            ResetEvent (hCommStartEvnt[0]);
            ResetEvent (hCommEOLEvnt[0]);
            _SendRobCmd (2, dRobTarg[0], dRobTarg[1], dRobTarg[2]);
            SetEvent (hCommStartEvnt[0]);
}

/*-----------------------------------------------------------------------*/
/*   Prepare the command string to Zaber                                 */
/*                                                     */
/* iCmdType = 20 : move absolutely                               */
/*          = 21 : move relatively                         */
/*          = 60 : read absolute position                    */
/*-----------------------------------------------------------------------*/
void _SendZabCmd (unsigned char _cCmdType, long int* _plZ)
{
    unsigned char*  pc_temp;

    strZabCmd[iCmdIdx[1]+0] = ZABUNIT;
    strZabCmd[iCmdIdx[1]+1] = _cCmdType;

    switch (_cCmdType) {
    case 20:
        pc_temp = (unsigned char*)_plZ;
        strZabCmd[iCmdIdx[1]+2] = *pc_temp;
```

```
      strZabCmd[iCmdIdx[1]+3] = *(pc_temp + 1);
      strZabCmd[iCmdIdx[1]+4] = *(pc_temp + 2);
      strZabCmd[iCmdIdx[1]+5] = *(pc_temp + 3);
      break;
   case 60:
      strZabCmd[iCmdIdx[1]+2] = 0;
      strZabCmd[iCmdIdx[1]+3] = 0;
      strZabCmd[iCmdIdx[1]+4] = 0;
      strZabCmd[iCmdIdx[1]+5] = 0;
      break;
   default:
      break;
   }

   iCmdIdx[1] += ZABCNT;
}

/*------------------------------------------------------------------------*/
/*   Thread for controlling communication with Zaber                      */
/*                                                                        */
/* hCommStartEvnt[1] control the start of communication                   */
/* hCommEOLEvnt[2]   control the end of communication                     */
/*------------------------------------------------------------------------*/
DWORD WINAPI _ZabCommMonProc (LPVOID _lpParam)
{
        char*       pstr_temp;
        char        str_temp[50];
        int         i_idx;

        while (iPortOpen[1] == 1)
        {
                WaitForSingleObject (hCommStartEvnt[1], INFINITE);

                pstr_temp = strZabCmd;

                i_idx = 0;

                while (i_idx != iCmdIdx[1])
                {
                        FlushInQ (iCommPort[1]);
                        FlushOutQ (iCommPort[1]);
                        ResetEvent (hCommEOLEvnt[1]);
                        iStrSize[1] = ZABCNT;
                        if (*(pstr_temp) != 60) {
                           lZabTargTemp1 = *((long int*)(pstr_temp + 2));
                           lZabTargTemp1 = floor((double)lZabTargTemp1 * ZABCONVFACT);
                           SetCtrlVal (iPnlHdl, PANEL_ZABTARGZ, lZabTargTemp1);

                        }
                        iBytsSent[1] = ComWrt (iCommPort[1], pstr_temp, iStrSize[1]);
                        WaitForSingleObject (hCommEOLEvnt[1], INFINITE);
                        pstr_temp += ZABCNT;
                        i_idx     += ZABCNT;
                }
     SetCtrlVal (iPnlHdl, PANEL_ZABMSG, "OK.");
                ResetEvent (hCommStartEvnt[1]);
                SetEvent (hCommEndEvnt[1]);
        }
        return 1;
}

/*--------------------------------------------------------------------------*/
/*   Callback function to deal with the ending of communication            */
/*                                                                    */
/*  According to the communication protocol,                          */
/*  command and reply have only 6 bytes respectively                      */
/*--------------------------------------------------------------------------*/
void CVICALLBACK _ZabRplyEOLFunc (int _iPortNum, int _iEvntMask,
                     void *_pvCallBackData)
{
        strTempZabRd[0] = '\0';

        ComRd (iCommPort[1], strTempZabRd, GetInQLen(iCommPort[1]));

   lZabCurrTemp = *((long int*)(strTempZabRd + 2));
   lZabCurr = floor((double)lZabCurrTemp * ZABCONVFACT);
   SetCtrlVal (iPnlHdl, PANEL_ZABCURRZ, lZabCurr);
   if (*(strTempZabRd+1) == 60) {
           SetCtrlVal (iPnlHdl, PANEL_ZABTARGZ, lZabCurr);
        }
        SetEvent (hCommEOLEvnt[1]);
}

/*--------------------------------------------------------------------------*/
/* Get liquid from the well                                           */
/*--------------------------------------------------------------------------*/
int  CVICALLBACK _GetLiquid(int _iPnlHdl, int _iCtrlHdl, int _iEvntHdl,
                     void *_pvCallBackData, int _iEvntData1,
                     int _iEvntData2)
{
        switch (_iEvntHdl) {
                case EVENT_COMMIT:
        _ZabJogZFunc (3,0);
                        break;
   }
        return 0;
}

/*--------------------------------------------------------------------------*/
/* Push a drop out of the needle                                          */
/*--------------------------------------------------------------------------*/
int CVICALLBACK _ZabPushDrop (int _iPnlHdl, int _iCtrlHdl, int _iEvntHdl,
                     void *_pvCallBackData, int _iEvntData1,
                     int _iEvntData2)
{
        switch (_iEvntHdl) {
                case EVENT_COMMIT:
                        _ZabJogZFunc (2,0);
                        break;
   }
        return 0;
}
```

```
/*------------------------------------------------------------------------*/
/*   Callback functions to jog the Zaber                          */
/*                                                    */
/*  Along Z-axis                                              */
/*------------------------------------------------------------------------*/
int CVICALLBACK _ZabJogMinuZ (int _iPnlHdl, int _iCtrlHdl, int _iEvntHdl,
                         void *_pvCallBackData, int _iEvntData1,
                         int _iEvntData2)
{
        switch (_iEvntHdl) {
                case EVENT_COMMIT:
                   _ZabJogZFunc (1,0);
                        break;
   }
        return 0;
}

int CVICALLBACK _ZabJogPlusZ (int _iPnlHdl, int _iCtrlHdl, int _iEvntHdl,
                         void *_pvCallBackData, int _iEvntData1,
                         int _iEvntData2)
{
        switch (_iEvntHdl) {
      case EVENT_COMMIT:
                        _ZabJogZFunc (2,0);
                        break;
   }
        return 0;
}

/*------------------------------------------------------------------------*/
/*                                                    */
/*         iType = 1 : Minus Z                                 */
/*                  = 2 : Plus  Z                                  */
/*      = 3 : Ready for getting liquid                        */
/*                  = 4 : Minus Z big step                               */
/*      = 5 : Plus  Z big step                            */
/*------------------------------------------------------------------------*/
void _ZabJogZFunc (int _iType, long int _lStep)
{
        GetCtrlVal (iPnlHdl, PANEL_ZABSTEP, &lZabStep);
        GetCtrlVal (iPnlHdl, PANEL_ZABCURRZ, &lZabCurr);
        GetCtrlVal (iPnlHdl, PANEL_ZABLOWZLIM, &lZabLowLim);
        GetCtrlVal (iPnlHdl, PANEL_ZABUPPZLIM, &lZabUppLim);

        switch (_iType) {
        case 1:
                lZabTargTemp = lZabCurr - lZabStep;
                break;
        case 2:
                lZabTargTemp = lZabCurr + lZabStep;
                break;
        case 3:
                lZabTargTemp = lZabUppLim - 4*lZabStep;
                break;
        case 4:
                lZabTargTemp = lZabCurr - 4*lZabStep;
                break;
        case 5:
                lZabTargTemp = lZabCurr + 4*lZabStep;
                break;
        }
    if (lZabTargTemp < lZabLowLim) {
        lZabTargTemp = lZabLowLim;
    }
    if (lZabTargTemp > lZabUppLim) {
        lZabTargTemp = lZabUppLim;
    }
    lZabTarg = ceil((double)lZabTargTemp / ZABCONVFACT);
    SetCtrlVal (iPnlHdl, PANEL_ZABMSG, "COMM4 ...");
    iCmdIdx[1] = 0;
    ResetEvent (hCommStartEvnt[1]);
    ResetEvent (hCommEOLEvnt[1]);
    _SendZabCmd (20,&lZabTarg);
    SetEvent (hCommStartEvnt[1]);
}

/*----------------------------------------------------------------------------*/
/*  Extract the X, Y, Z values from the Robot communication string           */
/*----------------------------------------------------------------------------*/
void _GetCoorVal (char* _strVal, double* _pdX, double* _pdY, double* _pdZ)
{
    char*       pstr_temp;
    int         i_idx;

    pstr_temp = _strVal;

    while (*pstr_temp == ' ') {
            pstr_temp++;
        }

        *_pdX = atof(pstr_temp);

        while (*pstr_temp != ' ') {
            pstr_temp++;
        }

        while (*pstr_temp == ' ') {
            pstr_temp++;
        }

        *_pdY = atof(pstr_temp);

        while (*pstr_temp != ' ') {
            pstr_temp++;
        }

        *_pdZ = atof(pstr_temp);
}

/*----------------------------------------------------------------------------*/
/*  Callback function to handle the emergency button                         */
```

```
/*------------------------------------------------------------------------*/
int CVICALLBACK _EmergencySwitch (int _iPnlHdl, int _iCtrlHdl, int _iEvntHdl,
                        void *_pvCallBackData, int _iEvntData1,
                        int _iEvntData2)
{
        switch (_iEvntHdl) {
                case EVENT_COMMIT:
        _EmgyProc ();
                        break;
                }
        return 0;
}

void _EmgyProc (void)
{
  int i_bttn_stat;
  /*iSensDAQStartRec = 0;*/
  GetCtrlVal (iPnlHdl, PANEL_EMERGENCYSWITCH, &i_bttn_stat);
  /* 0: released; 1: pressed */
  switch (i_bttn_stat) {
  case 0:
     break;
  case 1:
     break;
  default:
     break;
  }
}
/*------------------------------------------------------------------------*/
/*  Callback function to handle the receive of DAQ data                   */
/*------------------------------------------------------------------------*/
int CVICALLBACK _SensDAQReadData (int _iPnlHdl, int _iCtrlHdl, int _iEvntHdl,
                        void *_pvCallBackData, int _iEvntData1,
                        int _iEvntData2)
{
        switch (_iEvntHdl) {
                case EVENT_DAQ_NUMERIC_DATA_READY:
                  if (iSensDAQStartRec == 1
          && iSensDAQDataIdx < (SNESDAQNUMIDXLIM -1)) {

          /* Get the raw data */
          dSensDAQRaw[iSensDAQDataIdx] = *(double*)_iEvntData1;
          /* Use Butterworth lowpass filter */
          Bw_LPF (dSensDAQRaw, iSensDAQDataIdx, 50.00, 1.00, 2,
                  dSensDAQData);

          iSensDAQDataIdx ++;
          dSensDAQTime[iSensDAQDataIdx] = dSensDAQTime[iSensDAQDataIdx - 1]
                                  + SENSDAQNUMINT;
          DeleteGraphPlot (iPnlHdl, PANEL_SENSGRAPH, -1, VAL_DELAYED_DRAW);

          PlotXY (iPnlHdl, PANEL_SENSGRAPH, dSensDAQTime, dSensDAQData,
                  iSensDAQDataIdx - 1, VAL_DOUBLE, VAL_DOUBLE,
                  VAL_THIN_LINE, VAL_EMPTY_SQUARE, VAL_SOLID,
                  1, VAL_RED);
        }
        break;
                }
         return 0;
}

/*----------------------------------------------------------------------*/
/*  Callback function to handle the change of Robot limit              */
/*----------------------------------------------------------------------*/
int CVICALLBACK _RobLimZChng (int _iPnlHdl, int _iCtrlHdl, int _iEvntHdl,
                        void *_pvCallBackData, int _iEvntData1,
                        int _iEvntData2)
{
        switch (_iEvntHdl) {
                case EVENT_VAL_CHANGED:
                        GetCtrlVal (iPnlHdl, PANEL_ROBLIMITZ, &dRobLim[2]);
                        SetCtrlVal (iPnlHdl, PANEL_WELLZ, dRobLim[2] + ROBLIMGAP);
                        SetCtrlVal (iPnlHdl, PANEL_1STSPOTZ, dRobLim[2] + ROBLIMGAP);
                        break;
                }
        return 0;
}

/*----------------------------------------------------------------------*/
/*  Callback function to automate the 1st part of the experiment        */
/*----------------------------------------------------------------------*/
int CVICALLBACK _AutoExpStep1 (int _iPnlHdl, int _iCtrlHdl, int _iEvntHdl,
                        void *_pvCallBackData, int _iEvntData1,
                        int _iEvntData2)
{
        switch (_iEvntHdl) {
        case EVENT_COMMIT:
                if (hExpStartEvnt[0] == NULL) {
        hExpStartEvnt[0] = CreateEvent (NULL, TRUE, TRUE, NULL);
        if (hExpStartEvnt[0] == NULL) {
          Fmt (strErrMsg, "%s<CreateEvent failed: (%i)\n", GetLastError());
          MessagePopup ("CreateEvent Message", strErrMsg);
          return -1;
        }
        ResetEvent (hExpStartEvnt[0]);
      }
      SetEvent (hExpStartEvnt[0]);
      if (hExpFuncThrd[0] == NULL) {
        hExpFuncThrd[0] = CreateThread (0,0, _AutoExpMonProc1,
                                                        0, 0, &dwExpFuncThrdID[0]);
        if (hExpFuncThrd[0] == NULL) {
          Fmt (strErrMsg, "%s<CreateThread failed: (%i)\n", GetLastError());
          MessagePopup ("CreateThread Message", strErrMsg);
          return -1;
        }
      }
                break;
        }
        return 0;
}

/*----------------------------------------------------------------------*/
```

```
/*  Thread for the 1st part of the experiment                    */
/*-----------------------------------------------------------------------*/
DWORD WINAPI _AutoExpMonProc1 (LPVOID _lpParam)
{
        while (1) {
                /* Trigger the process */
                WaitForSingleObject (hExpStartEvnt[0], INFINITE);

                /* Get ready for getting liquid */
                ResetEvent (hCommEndEvnt[1]);
                _ZabJogZFunc (3,0);
                WaitForSingleObject (hCommEndEvnt[1], INFINITE);

                /* Go to liquid well */
                ResetEvent (hCommEndEvnt[0]);
                _RobGo2WellFunc ();
                WaitForSingleObject (hCommEndEvnt[0], INFINITE);

                /* Wait for the capillary attraction */
                Sleep (3000);

                /* Suct liquid into needle */
                GetCtrlVal (iPnlHdl, PANEL_ZABCURRZ, &lZabCurr);
                GetCtrlVal (iPnlHdl, PANEL_ZABLOWZLIM, &lZabLowLim);
                while (lZabCurr > lZabLowLim) {
                   ResetEvent (hCommEndEvnt[1]);
                   _ZabJogZFunc (1,0); /* Get liquid */
                   WaitForSingleObject (hCommEndEvnt[1], INFINITE);
                }

                /* Move Robot up */
                ResetEvent (hCommEndEvnt[0]);
                _RobJogXYZFunc (8);
                WaitForSingleObject (hCommEndEvnt[0], INFINITE);
                ResetEvent (hExpStartEvnt[0]);
        }
}

/*-----------------------------------------------------------------------*/
/*  Callback function to automate the 2nd part of the experiment         */
/*-----------------------------------------------------------------------*/
int CVICALLBACK _AutoExpStep2 (int _iPnlHdl, int _iCtrlHdl, int _iEvntHdl,
                        void *_pvCallBackData, int _iEvntData1,
                        int _iEvntData2)
{
        switch (_iEvntHdl) {
        case EVENT_COMMIT:
                if (hExpStartEvnt[1] == NULL) {
        hExpStartEvnt[1] = CreateEvent (NULL, TRUE, TRUE, NULL);
        if (hExpStartEvnt[1] == NULL) {
           Fmt (strErrMsg, "%s<CreateEvent failed: (%i)\n", GetLastError());
           MessagePopup ("CreateEvent Message", strErrMsg);
           return -1;
        }
        ResetEvent (hExpStartEvnt[1]);
     }
```

```
      SetEvent (hExpStartEvnt[1]);
      if (hExpFuncThrd[1] == NULL) {
         hExpFuncThrd[1] = CreateThread (0,0, _AutoExpMonProc2,
                                                        0, 0, &dwExpFuncThrdID[1]);
         if (hExpFuncThrd[1] == NULL) {
            Fmt (strErrMsg, "%s<CreateThread failed: (%i)\n", GetLastError());
            MessagePopup ("CreateThread Message", strErrMsg);
            return -1;
         }
      }
                  break;
         }
         return 0;
}

/*-------------------------------------------------------------------------*/
/*  Thread for the 2nd part of the experiment                             */
/*-------------------------------------------------------------------------*/
DWORD WINAPI _AutoExpMonProc2 (LPVOID _lpParam)
{
   while (1) {
      /* Trigger the process */
         WaitForSingleObject (hExpStartEvnt[1], INFINITE);

      /* Initialize variables */
      iCurrRow = 1;
      iCurrCol = 1;
      iSpotNum = 0;
      SetCtrlVal (iPnlHdl, PANEL_CURRENTROW,iCurrRow);
                  SetCtrlVal (iPnlHdl, PANEL_CURRENTCOL,iCurrCol);
                  SetCtrlVal (iPnlHdl, PANEL_SPOTNUM,iSpotNum);


                  /* Go to the first spot */
                  ResetEvent (hCommEndEvnt[0]);
                  _RobGo21stSpotFunc ();
                  WaitForSingleObject (hCommEndEvnt[0], INFINITE);

                  GetCtrlVal (iPnlHdl, PANEL_MAXROW, &iMaxRow);
                  GetCtrlVal (iPnlHdl, PANEL_MAXCOL, &iMaxCol);
                  for (iCurrRow = 1; iCurrRow <= iMaxRow; iCurrRow ++) {
                     for (iCurrCol = 1; iCurrCol <= iMaxCol; iCurrCol ++) {
                        /* Push drop */
                        ResetEvent (hCommEndEvnt[1]);
                        _ZabJogZFunc (5,0);
                        WaitForSingleObject (hCommEndEvnt, INFINITE);
                        /* Wait for stable condition of sensor */
                        Sleep (2000);
                        /* Trigger sensor data recording */
                        iSensDAQDataIdx = 0;
                        iSensDAQStartRec = 1;
                        iSensDAQPosDataIdx1 = 0;
                        iSensDAQPosDataIdx2 = 0;
                        iWhenToPickRef = 0;
                        iCurvRefIdx = 0;
                        /* Robot touches the Z limit? */
```

```
                while (dRobCurr[2] > dRobLim[2]) {
                    /* Spot formed? */
                    if (_TestSpotForm() != 1){
                        /* Remember the index of start time */
                        iSensDAQPosDataIdx1 = iSensDAQDataIdx;
                        /* Move robot one step down */
                        ResetEvent (hCommEndEvnt[0]);
                        _RobJogXYZFunc(3);
                        WaitForSingleObject (hCommEndEvnt[0], INFINITE);
                        /* Get the index of end time */
                        iSensDAQPosDataIdx2 = iSensDAQDataIdx;
                        iWhenToPickRef ++;
                        if (iWhenToPickRef > 4) {
                            iCurvRefIdx = iSensDAQPosDataIdx1;
                        }
                    }
                    else {
                        break;
                    }
                }
                /* Elongation, i.e. move robot up */
                ResetEvent (hCommEndEvnt[0]);
                _RobJogXYZFunc(9);
                WaitForSingleObject (hCommEndEvnt[0], INFINITE);

                SetCtrlVal (iPnlHdl, PANEL_CURRENTROW,iCurrRow);
                SetCtrlVal (iPnlHdl, PANEL_CURRENTCOL,iCurrCol);
                iSpotNum = (iCurrRow - 1)*iMaxCol + iCurrCol;
                SetCtrlVal (iPnlHdl, PANEL_SPOTNUM,iSpotNum);

                /* Stop sensor data recording */
                iSensDAQStartRec = 0;
                /* Save data to file*/
                sprintf(strFileName, "row%dcol%d.dat", iCurrRow, iCurrCol);
                iFileHdl = OpenFile (strFileName, VAL_WRITE_ONLY, VAL_OPEN_AS_IS,
                                VAL_BINARY);
WriteFile (iFileHdl, (char*)dSensDAQData,
            sizeof(dSensDAQData[0])*iSensDAQDataIdx);
CloseFile (iFileHdl);

                if ((iSpotNum % 6) == 0 ) {
                    /* Push more droplet */
                    ResetEvent (hCommEndEvnt[1]);
                    _ZabJogZFunc (5,0);
                    WaitForSingleObject (hCommEndEvnt, INFINITE);
                }

                /* Move robot to next Column */
                ResetEvent (hCommEndEvnt[0]);
                _RobJogXYZFunc(5);
                WaitForSingleObject (hCommEndEvnt[0], INFINITE);
            }
            /* Move robot to next Row */
            ResetEvent (hCommEndEvnt[0]);
            _RobJogXYZFunc(7);
            WaitForSingleObject (hCommEndEvnt[0], INFINITE);
```

```
        }
                ResetEvent (hExpStartEvnt[1]);
            }
}

/*----------------------------------------------------------------------*/
/* Test the formation of a spot                                        */
/*----------------------------------------------------------------------*/
int _TestSpotForm(void)
{
  int          i_idx,       i_idx1,      i_idx2;
  double        d_temp, d_max, d_min;
  double*       pd_temp_data;
  double                          d_int1, d_int2;

  if (iSensDAQPosDataIdx1 != 0
    && iSensDAQPosDataIdx2 != 0) {

    pd_temp_data = dSensDAQData;
    i_idx1 = iSensDAQPosDataIdx1;
    i_idx2 = iSensDAQPosDataIdx1;
    d_int1 = double (iSensDAQPosDataIdx2 - iSensDAQPosDataIdx1);

    for (i_idx = iSensDAQPosDataIdx1 + 1;
        i_idx < iSensDAQPosDataIdx2-1; i_idx++) {
      if (pd_temp_data[i_idx] <= pd_temp_data[i_idx-1]) {
        i_idx2 = i_idx;
        continue;
      }
      else {
        i_idx1 = i_idx;
        i_idx2 = i_idx;
        continue;
      }
    }
    if (i_idx2 != i_idx1) {
       d_int2 = double (i_idx2 - i_idx1);
       d_temp = (pd_temp_data[i_idx1] - pd_temp_data[i_idx2])/
                         pd_temp_data[i_idx1];
      d_temp = (d_temp * 1000.0) * (d_int1 / d_int2) / 10.0;
      if (d_temp > SENSSPOTTESTTH) {
        return 1;
      }
    }
  }

  return 0;
}


int CVICALLBACK _ZabGoHome (int _iPnlHdl, int _iCtrlHdl, int _iEvntHdl,
                        void *_pvCallBackData, int _iEvntData1,
                        int _iEvntData2)
{
        switch (_iEvntHdl) {
                case EVENT_COMMIT:
```

```
        _ZabGoHomeFunc ();
                        break;
  }
        return 0;
}

void _ZabGoHomeFunc (void)
{
  SetCtrlVal (iPnlHdl, PANEL_ZABMSG, "COMM4 ...");
  GetCtrlVal (iPnlHdl, PANEL_ZABUPPZLIM, &lZabTargTemp);
  lZabTarg = ceil((double)lZabTargTemp / ZABCONVFACT);
  iCmdIdx[1] = 0;
  ResetEvent (hCommStartEvnt[1]);
  ResetEvent (hCommEOLEvnt[1]);
  _SendZabCmd (20,&lZabTarg);
  SetEvent (hCommStartEvnt[1]);
}
```

REFERENCES

[1] Dangond, F., "Chips around the world," Physiological Genomics (Online), Volume 2, Issue 2, March 13, 2000, Pages 53-58.

[2] Cheung, V. G., Morley, M., Aguilar, F., Massimi, A., Kucherlapati, R. and Childs, G., "Making and reading microarrays," Nature Genetics, Volume 21, Issue 1 Supplement, January 1999, Pages 15-19.

[3] Duggan, D. J., Bittner, M., Chen, Y., Meltzer, P. and Trent, J. M., "Expression profiling using cDNA microarrays", Nature Genetics, Volume 21, Issue 1 Supplement, January 1999, Pages 10-14.

[4] Chang, T.N., Hou, E. and Godbole, K., "Optimal Input Shaper Design For High Speed Robotic Workcells," to appear in the Journal of Vibrational and Control, Sage Science Press.

[5] Chang, T.N., Kwadzogah, R., and Caudill, R., "Vibration Control On Linear Robots With Digital Servocompensator," IEEE/ASME Transactions on Mechatronics, Volume 8, Issue 4, December 2003, Pages 439- 445.

[6] Chang, T.N., Dani, B., Ji, Z., and Caudill, R., "Contactless Magnetic Leadscrew: Vibration control and resonance compensation," IEEE/ASME Transactions on Mechatronics, Volume 9, Issue 2, June 2004, Pages 458-461.

[7] Chang, T.N., and Sun, X., "Control of hysteresis in a monolithic nanoactuator," Proceeding to the 2001 American Control Conference, Arlington, VA, June 2001.

[8] Chang, T.N and Sun, X., "Analysis and Control of Monolithic Piezoelectric Nano-actuator," IEEE Transaction on Control Systems Technology, Special Issue on Smart Materials, January 2001.

[9] Chang, T.N., Jaroonsiriphan, P., and Sun, X., "Integrating Nanotechnology Into Undergraduate Experience, A Web-Based Approach", International Journal of Engineering Education, Volume 18-5, August 2002.

[10] Chang, T.N., and Jaroonsiriphan, P., "Web-Based Distance Experiments for Real Time Control and Signal Processing," Proceedings of the 2002 ASEE Annual Conference.

[11] Chang, T.N., "Servo Control Design", Encyclopedia of Life Support Systems, United Nations Educational, Scientific, and Cultural Organization (UNESCO), 2004.

[12] Chang, T.N., and Tolias, P.P., "Delivery of metered amounts of liquid materials," US and International Patents pending.

[13] Introduction to Molecular Beacons, Public Health Research Institute. (n.d). Retrieved July 14, 2005, from http://www.molecular-beacons.org/Introduction.html.

[14] Brochure for MTI Fotonic Sensor, MTI Instruments Inc. (n.d). Retrieved July 14, 2005, from http://www.mtiinstruments.com/pdf/mti2100.pdf.

[15] Instruction Manual of MTI 1000 Fotonic Sensor, MTI Instruments Inc.

[16] Instruction Manual of SEIKO D-TRAN Robot.

[17] Zaber Technologies Inc., T-Series Positioning Products User's Manual.

[18] Products microarray hardware, TeleChem International, Inc. (n.d). Retrieved July 21, 2005, from http://arrayit.com/Products/Printing/946/946.html.

[19] Affymetrix GeneChip technology, Affymetrix. (n.d). Retrieved July 26, 2005, from http://www.affymetrix.com/index.affx.

[20] Hsieh, H.B., et al., "Ultra-High-Throughput Microarray Generation and Liquid Dispensing UsingMultiple Disposable Piezoelectric Ejectors", Journal of Biomolecular Screening, 2004.

[21] Jane, T., Chang, J. and Kim, C.J., "A Silicon Micromachined Pin for Contact Printing", Proceedings of the IEEE Micro Electro Mechanical Systems (MEMS), 2003, Pages 295-298.

[22] Smith, J.T. and Reichert, W.M., "The optimization of quill-pin printed protein and DNA microarrays", Annual International Conference of the IEEE Engineering in Medicine and Biology Proceedings, 2002, Pages1630-1631.

[23] George, R.A., Woolley, J.P. and Spellman, P.T., "Ceramic capillaries for use in microarray fabrication", Genome Research, Volume 11, Issue 10, 2001, Pages 1780-1783.