Université d'Ottawa · University of Ottawa

Université d'Ottawa · University of Ottawa

FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTDOCTORALES

FACULTY OF GRADUATE AND POSTDOCTORAL STUDIES

Codrin PASCA

AUTEUR DE LA THÈSE - AUTHOR OF THESIS

M. A. Sc. (Electrical Engineering)

GRADE - DEGREE

Department of Electrical Engineering

FACULTÉ, ÉCOLE, DÉPARTEMENT - FACULTY, SCHOOL, DEPARTMENT

TITRE DE LA THÈSE - TITLE OF THE THESIS

Smart Tactile Sensor

E. Petriu

DIRECTEUR DE LA THÈSE - THESIS SUPERVISOR

CO-DIRECTEUR DE LA THÈSE - THESIS CO-SUPERVISOR

EXAMINATEURS DE LA THÈSE - THESIS EXAMINERS

N. Geerganas

H. Schwarts

J.-M. De Koninck, Ph.D.

LE DOYEN DE LA FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTDOCTORALES

DEAN OF THE FACULTY OF GRADUATE AND POSTDOCTORAL STUDIES

Smart Tactile Sensor

by

Codrin Pasca

A thesis submitted to the

Faculty of Graduate and Postdoctoral Studies

in partial fulfillment of the requirements for the degree of

Master of Applied Science

Ottawa-Carleton Institute for Electrical and Computer Engineering

School of Information Technology and Engineering

Faculty of Engineering

University of Ottawa

September 2004

Your file Votre référence
ISBN: 0-494-01576-4
Our file Notre référence
ISBN: 0-494-01576-4

To my family

Abstract

This thesis presents an experimental smart tactile sensor system based on a 16-by-16 array of Force Sensing Resistor (FSR) elements. The tactile image data are acquired, memorized, and then transferred to a PC by an electronic interface using a PICmicro microcontroller. A user-friendly graphical user interface, integrates all the functions needed to acquire, process, display and save the tactile measurement data. Experiments on a set of 3D geometric symbols illustrate the functionality of the tactile sensor system. Finally, an application of the smart tactile sensor for robotic tactile object recognition is presented.

Acknowledgment

First of all, I would like to thank my thesis supervisor Dr. Emil M. Petriu for his continuous support and encouragement during the whole period of my studies.

Special thanks go to Dr. Voicu Groza and Dr. Pierre Payeur for sharing their experience and information and for their valuable suggestions and help during this period.

I would like to express my deep gratitude and appreciations to my mother Marta for her unfailing love and encouragement during my whole life, but especially since my family and me have started a new life.

My warmest thanks go to my wife Isabela and my son Ionut, for their love, support, encouragement, and understanding; for the sacrifices they had to endure because of this thesis; and especially their patience during my work. Even if my son is just a little kid, he was able to understand that his father had to spend a great deal of time, even in week ends, at the university. I do hope their patience will be rewarded.

I am grateful to my brother Sever and his family who were always there for me providing support, encouragement, and, many times, valuable technical suggestions and advises.

I would also like to thank my brother Vasile for his support, guidance, and encouragement he always was ready to give me.

Finally, I would like to thank my colleagues in the Sensing and Modeling Research Laboratory, especially Ana-Maria Cretu and Rami Abielmona, for their help and support during the last two years.

Table of Contents

List of Figures

List of Acronyms

3-D	Three dimensions
A/D	Analog to Digital
CCD	Charge Coupled Device
FSR	Force Sensing Resistor
TS GUI	Force Sensing Resistor Tactile Sensor Graphical User Interface
GUI	Graphical User Interface
PRA	Pseudo-Random Array

Trademarks

- **FSR** and **Force Sensing Resistor** are registered trademarks of Interlink Electronics Inc.
- **Microchip, PIC,** and **PICmicro MCU** are registered trademarks of Microchip Technology Inc.
- **EPIC Plus PICmicro MCU Programmer** is registered trademark of microEngineering Labs, Inc.
- **MATLAB** is registered trademark of The MathWorks, Inc.

Chapter 1

Introduction

1.1. Motivation

Tactile sensing technologies have been developed for more than thirty years and the transduction methods and configurations have been continuously improved [1]. Haptic perception essentially emulates biological haptic perception mechanisms, which integrates two distinct sensing modalities [2, 3]:

- *Cutaneous* tactile sensing, which provides information about contact force, contact geometric profile, and the temperature of the touched object [4]; and
- *Kinesthetic* sensing, which provides information about the position and velocity of the structure carrying the tactile sensor [5].

While at the beginning researchers were chiefly investigating new transduction techniques, recently their focus was shifted on using tactile sensors in a variety of applications [1]. Early tactile sensors were based on force-sensing devices. The new trend is to measure not only the shape and size of an object, but also, if possible, one or more of its following properties: texture, hardness, consistency, temperature, and even moisture content, just like the human hand is able to do. Thus, studies of human tactile performance have been conducted

and new soft materials have been considered to use as an elastic compliant overlay for a large variety of tactile sensors: elastomers, fluids, powders, and even gels that present electrorheological effects (application of a strong electric field change the gel from a fluid to a plastic solid), etc.

Various application areas already do, whereas other will very soon integrate tactile sensing technologies.

In robotics, tactile sensors are used to enhance the visual capability of robots in telerobotic and object recognition applications and to perform human like dexterous object manipulations. Robots equipped with tactile sensors are also used in exploration, hazardous environment, and aerospace applications.

In medicine, tactile sensors increase the safety and reliability of minimally invasive surgeries, external palpation procedures, and orthodontic diagnosis. Also, user friendly robots aid the disabled people in completing a variety of tasks.

In agriculture and food processing, special designed robot grippers for soft products like donuts, asparagus, tomatoes, etc. could increase the manufacturing productivity, reduce hygiene risks, and eliminate human errors.

Tactile sensors are becoming popular in virtual environment human-interface technology. Haptic interfaces provide the tactile and kinesthetic feedback resulted from user contact with objects in the virtual environment.

Other areas include: archaeology, in which non-destructive methodologies are used to analyse finds; recycling industry, where tactile sensors can be used to automate the selection and sorting of various waste materials; service industry, where robot floor cleaners that sense the shape of furniture and other objects in order to clean around them or even to disseminate between the objects that are to be cleaned or picked-up are considered.

The motivation of the work that follows was to provide a basic understanding of how various tactile sensors, and in particular the Force Sensing Resistor (FSR), work and how tactile images can be acquired, processed, displayed, and saved for future use.

1.2. Objective

The main objectives of this thesis are:

- Designing and implementing a smart microcontroller-based interface to a force-sensing tactile sensor that is able to gather, memorize, and then transfer the tactile sensor data to a personal computer.
- Designing a graphical user interface that helps to acquire, process, display, and save the tactile data received from the tactile sensor interface.
- Performing experiments using a set of geometrical symbols in order to demonstrate the performance of the smart tactile sensor.

1.3. Thesis Organization

The thesis has the following layout:

Chapter 2 begins with a basic introduction of tactile sensors and later on describes a few examples of basic tactile sensing technologies found in literature.

Chapter 3 focuses on the Smart Tactile Sensor system. First, the Force Sensing Resistor Transducer specifications and a description of its functionality are presented. Then the block diagram of the tactile sensor interface is introduced. Later on, the hardware and software functional blocks are presented in detail.

Chapter 4 describes the tactile image acquisition, processing, and display functions performed by TS GUI, a custom designed graphical user interface.

Using special handcrafted wooden characters, a mechanical test set up, and the graphical user interface, tactile images are obtained and depicted in *Chapter 5*. First, raw data is

acquired. Then filters are applied and the results are displayed. Also, for a particular randomly selected pixel of the acquired image, the moving average for each possible window size is calculated and the standard deviation as a function of window size is plotted.

Using the smart tactile sensor, an application for robotic tactile object recognition is presented in *Chapter 6*. A commercial manipulator equipped with a passive-compliant wrist and the smart tactile sensor explores geometric symbols embossed on object surfaces. The shape and orientation of the geometric symbols and eventually the object face and exact position of the geometric symbols on the face are recognized by a neural network.

Chapter 7 summarizes the work done in this thesis and briefly suggests some directions for further research developments on the smart tactile sensor.

Appendix A contains the schematic diagrams of the electronic interface, whereas *Appendix B* and *Appendix C* contain the listing of the assembly code for the PIC microcontroller and the MATLAB code for the TS GUI graphical user interface, respectively.

1.4. Main Contributions

The specific contributions of this work are as follows:

- Development and implementation of a tactile sensor electronic interface, based on a PICmicro microcontroller.
- Development of a PICmicro microcontroller assembly software interface, which performs the following tasks: selects each taxel of the tactile sensor; performs analog-to-digital conversion of the currently selected taxel value; memorizes the tactile image data; and transmits the data to a personal computer through the serial communication port.
- Design and implementation of a MATLAB PC-based software interface, which integrates all the functions that are needed in order to acquire, process, display, and save the tactile image data from the PICmicro microcontroller.

- Development of a mechanical test set up for the tactile exploration of object surfaces that holds both the tactile sensor and the geometric symbol to be sensed and which was used to perform the experiments.

Chapter 2

Tactile Sensing Technologies and Applications - Review

The American Heritage Dictionary of the English Language [6] defines **tactile** as:

1. *perceptible to the sense of touch, tangible [...];*
2. *used for feeling; ex.: a tactile organ;*
3. *of, relating to, or proceeding from the sense of touch; tactual; ex.: a tactile reflex.*

A **tactile sensor** is defined in [1] as *"a device or system that can measure a given property of an object or contact event through physical contact between the sensor and the object"*. Human tactile perception is a complex process that consists of two distinct sensing modalities [7]:

- *Passive sensing* produced by the "cutaneous" sensory network;
- *Active sensing,* which integrates the cutaneous sensory information with "kinesthetic" data.

Through direct contact, a tactile sensor can measure various properties of the object it touches, such as its shape, texture, temperature, hardness, and moisture content. Of course, there is not such a tactile sensor system that can measure all of these properties. Researchers have been focusing on those properties to be measured that best suit a specific application.

Based on the principle of operation, tactile sensors can be grouped into two categories:

- *Force-sensitive* tactile sensors (conductive elastomer, strain gauge, and piezoelectric), which measure the contact force profile; and
- *Displacement-sensitive* tactile sensors (capacitive and optoelectronic), which measure the mechanical deformation of a sensing element.

A great variety of tactile sensor technologies have been developed over the years [5]:

- conductive elastomer;
- strain gauge;
- piezoelectric;
- piezoresistive;
- capacitive;
- optoelectronics.

A few examples of tactile sensor technologies are presented in the following sections.

2.1. Displacement-Sensitive Tactile Sensors

2.1.1. Displacement-Sensitive Silicon Tactile Sensor

It is known that silicon is a material that presents very little hysteresis, that is, it is almost perfectly elastic. Moreover, silicon micro-machining is very easy to accomplish with today's technology [8].

Displacement-sensitive tactile sensors convert the force applied to a material, in this case a silicon membrane, into a microscopic displacement. There are three possible techniques to measure the deformation of the membrane: capacitively, optically, or with integrated strain gauges. Figure 2.1 shows the principle of the capacitive type displacement-sensitive silicon tactile sensor. A glass substrate is attached face-to-face to a silicon wafer by anodic bonding. One part of the capacitors consists of a matrix of square electrodes deposited on the glass substrate. On the other hand, a matrix of membranes in line with the electrodes on the glass

substrate is etched out on the silicon wafer. The second part of each flat capacitor consists of the electrode deposited on the membrane. The air gap between the two electrodes, thus the capacitance, will vary with the applied force.

The advantages of the capacitive displacement-sensitive silicon tactile sensors are:

- No hysteresis, due to their silicon based construction;
- High sensitivity, due to the very thin membrane and air gap;
- High spatial resolution.

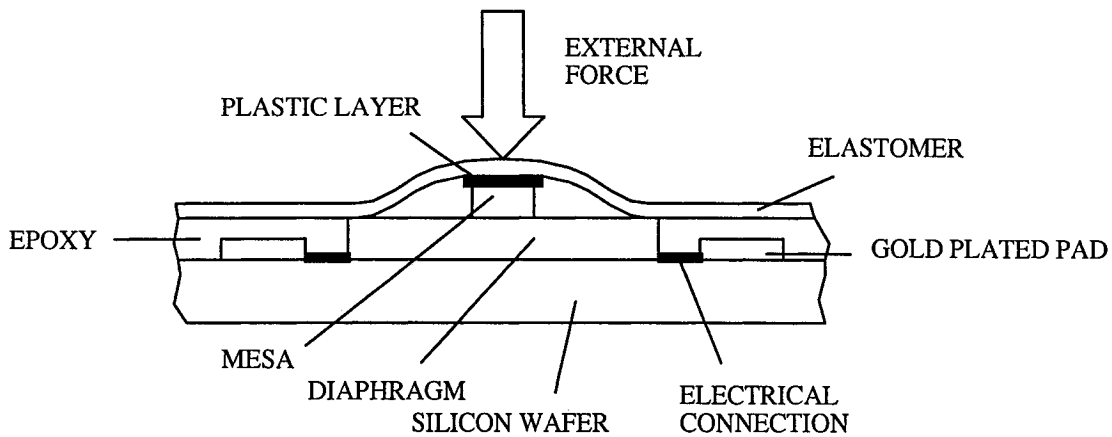


Figure 2.1. *Displacement-sensitive silicon tactile sensor (adapted from [8]).*

2.1.2. Tactile Sensing Using an Optical Transduction Method

This type of sensor, introduced in [9], uses a pressure intensity to light intensity transduction method. It uses the properties of optical reflection between media of different refractive index. Mounted on the finger of a robotic arm, the sensor outputs a video signal suitable for feeding into a conventional image processing system producing very detailed images of the probed object. The design of the sensor is based on the following phenomenon:

Consider light travelling from a medium A of refractive index n_A to a medium B of refractive index n_B. If $n_A > n_B$ then the light is totally reflected, whereas if $n_A < n_B$ diffuse reflection occurs.

The transducing structure is shown in figure 2.2. It consists of a light source and a compliant membrane stretched above, but not in close contact, with a clear plate (made of acrylic, glass or perspex).

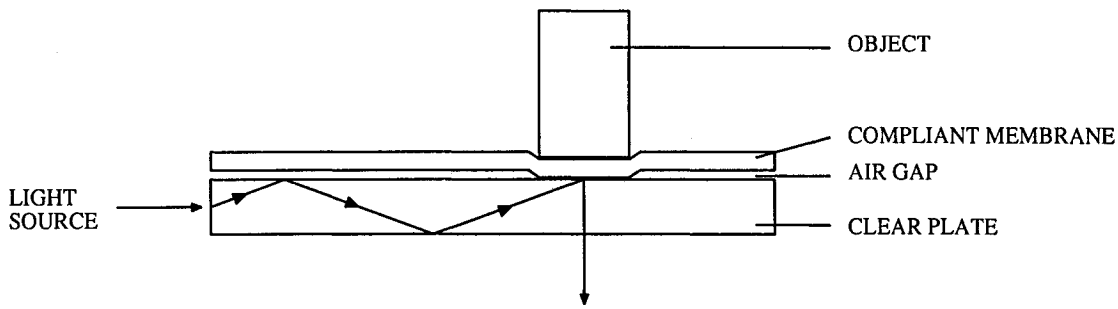


Figure 2.2. *Tactile sensor using an optical transduction method (adapted from [9]).*

Light directed along the left edge of the plate shines into the plate body. The relationship between the refractive indexes of air (n_0), plate (n_1), and the membrane (n_2) is: $n_0 < n_1 < n_2$. If no object indents the membrane of the sensor, the plate is surrounded by air and the light is totally internally reflected inside the plate because $n_1 > n_0$. Thus, the light entering along the edge of the plate does not leave it. As a consequence, the lower surface of the clear plate, which is the imaging area, appears dark.

If an object is placed on the sensor surface, the membrane is brought into contact with the clear plate. In this situation, diffuse reflection occurs on the plate-membrane boundary because $n_1 < n_2$. Light passes out through the lower surface of the plate at the location where the membrane has contact with the plate and is then directed onto an imaging device, such as a CCD (Charge Coupled Device) array. The obtained image has a bright patch whose position

and size corresponds to the part of the object in contact with the surface (membrane) of the sensor. As the pressure applied on the plate increases so does the intensity of the bright patch. The explanation is very simple:

Consider a single pixel of the image. It corresponds to a small area of the plate. The amount of light that reaches the pixel is proportional with the number of elementary contact points between the membrane and the clear plate. But the number of contact points increases as the pressure increases. As a consequence, the pixel intensity increases with pressure. In conclusion, the whole image is brighter as the pressure on the sensor increases.

The resolution and sensitivity of the sensor are influenced by the colour of the compliant membrane. Experiments made with red, vibrant-red, and green latex showed that membranes made of these materials are translucent. Therefore, the obtained image has a dark 'shadow' area in the background before an object makes contact with the sensor, which is a reverse effect to that of proper contact that shows as bright patches. But this is not a major inconvenience as the two effects can be conveniently separated by the image processing system. As expected, it was determined that the background intensity of the image decreases as the latex thickness increases (because the translucency of latex decreases). A resolution of 0.3 mm has been obtained using red and green 0.18 mm thick membranes.

Another important parameter is the illumination of the clear plate. It was determined that the image intensity decreases as the illumination level decreases, and vice versa.

Finally, the membrane texture also affects the sensor's behaviour. A suitable membrane material should meet a number of criteria: the membrane must respond quickly to surface impulses; it must recover quickly after the release of pressure on it; the ambient temperature must not affect the behaviour of the membrane; it must be opaque, with the surface in contact with the clear plate being reflective; the membrane should be robust and durable.

The advantages of the optical transduction method based sensors are:

- High resolution;
- Design simplicity;
- Compatibility with vision sensing and processing technology;
- Electrical interference free.

The limitations of the sensor come from the fact that it depends on an elastomer, which may not be robust and durable, can cause hysteresis and creep, and might present an intrinsic pattern or texture that would be imposed on the image.

2.2. Force-Sensitive Tactile Sensors

2.2.1. Piezoresistive Silicon Tactile Sensor

The principle of piezoresistive sensors is based on the property of a specific elastic material of changing its conductivity when an applied force causes a geometrical deformation of it. Materials that have this property are elastomers, which are intrinsically non-conductive but they are made conductive by impregnation with metal or carbon particles.

Figure 2.3 shows a piezoresistive sensor realized using integrated-circuit technology [8]. Active circuitry, like A/D converters, amplifiers, and signal processors, are integrated with the sensor components on the same silicon wafer.

The pressure-sensitive elastomer changes its resistivity as the force is applied on it. Sensing electrodes connect the active circuitry to the elastomer, so that its resistivity changes can easily be measured and converted into images.

The advantages of this type of sensor are:

- Good resolution;

- Large sensitivity, due to the several order of magnitude changes in the conductivity of piezoresistive materials.

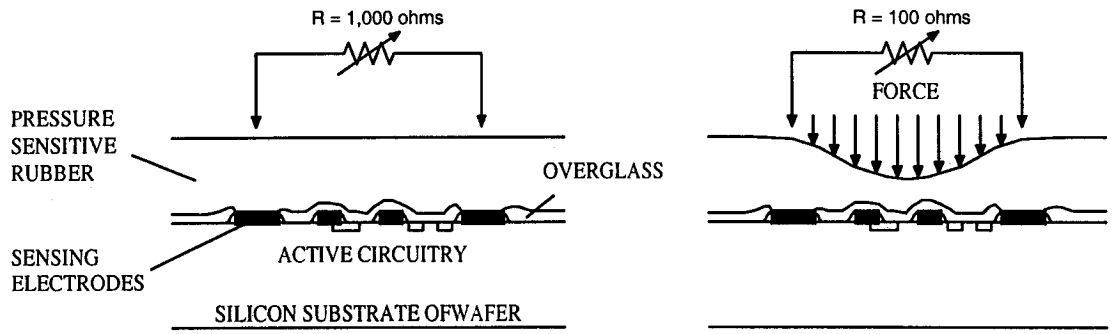


Figure 2.3. *Piezoresistive silicon tactile sensor (adapted from [8]).*

The major overhead is that such a sensor has a quite large hysteresis due to the elastomer membrane.

2.2.2. Piezoelectric Silicon Tactile Sensor

There are different materials that exhibit piezolelectric properties, such as ceramics, silicon, thin and thick polymeric films. A silicon based piezoelectric tactile sensor is presented in [8].

A force applied to a piezoelectric material induces the generation of a surface charge (a voltage difference), while the deformation of the material is negligible.

As with the piezoresistive sensors, piezoelectric sensors incorporate polymers. But, in this case, polymers are not impregnated but rather polarised.

Using integrated-circuit technology, charge sensitive MOSFET transistors are integrated onto a silicon wafer. A piezoelectric material is connected on top of the silicon so that a force applied to it generates electric charges that are then transformed by the MOSFET transistors in electric voltage.

Advantages of the piezoelectric sensors:

- High resolution;
- Very high sensitivity.

The major disadvantage is that piezoelectric materials are pyroelectric, i.e. temperature has a major influence on their properties, so that such a tactile sensor will respond to temperature changes and gradients.

It is worth mentioning that silicon is a very thin material that can integrate local signal processing circuitry for each tactile element while allowing a very high density, properties that make it a very good candidate for tactile sensors. High resolution is achieved with all the above sensors. Silicon is used either as the sensor material itself, or as a substrate for tactile sensor sheets.

The major disadvantages of silicon are:

- It is a quite fragile, inflexible material, thus it brakes easily;
- It is only flat in shape;
- It has a quite low temperature range (about 150 °C).

2.2.3. A Strain Gauge Tactile Sensor

Human finger force can be measured using the metallic strain gauge-based sensor described in [10] and depicted in figure 2.4.

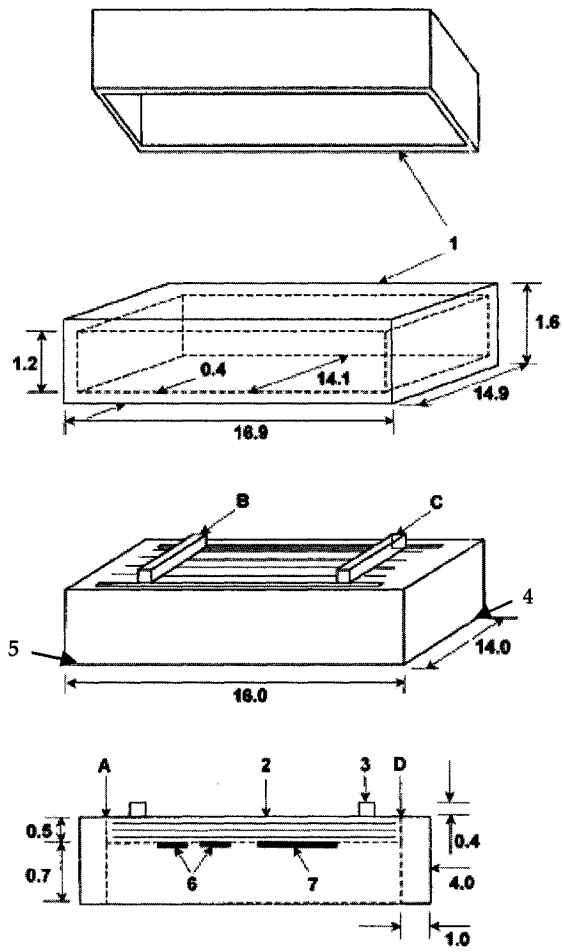


Figure 2.4. *Strain gauge tactile sensor for finger-mounted applications: 1- Metallic cover; 2 - Metallic web; 3 - Force concentrators; 4 - Transversal walls; 5 - Longitudinal walls; 6,7 - Strain gauges; (dimensions are in mm) (reprinted from [10]).*

The external forces are applied to the hard protective metallic cover of the sensor, which has also a role in directing all the applied forces through the force concentrators to the metallic web. This arrangement assures that the sensor does not vary its response with the point where the external force is applied on the cover. The web is then deformed both

longitudinally and transversally between B and C, which is actually the area of interest. Under the web, there are the strain gauges, which are located between B and C. Two strain gauges are parallel and two are transverse to the longitudinal axis of the sensor. They are placed into a full Wheatstone bridge, the output of which is connected to an electronic interface and a PC in order to display the data gathered by the sensor.

The advantages of this sensor are:

- Robust design;
- Linear characteristic;
- Good repeatability;
- Quite high resolution of 0.3 N;
- High sensitivity of 0.12 V/N;
- Very low hysteresis.

On the other hand, the sensor has a quite large size, which can be a drawback for some applications.

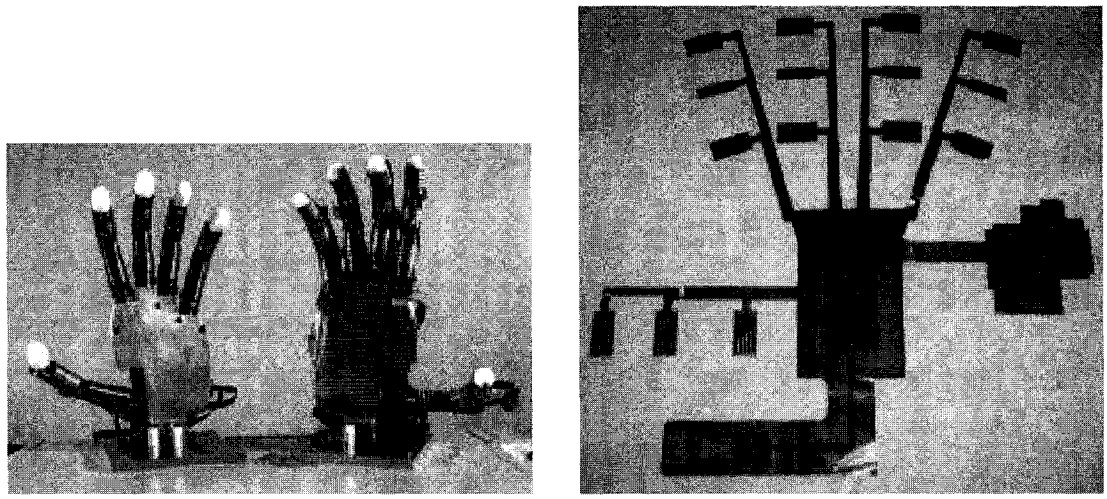
2.3. Applications

2.3.1. Dexterous Anthropomorphic Robot Hand with Distributed Tactile Sensor

"GIFU hand II" [11] is an anthropomorphic robot hand that simulates the human hand and can be used to perform dexterous object manipulations. As part of a humanoid robot, the hand would be able to execute tasks in dangerous or remote environments.

The hand (figure 2.5.a) consists of a thumb and four fingers linked by servomotor-driven joints built into the fingers and the palm. Each finger has four joints with three degrees-of-freedom, whereas the thumb has four joints also but four degrees-of-freedom to better simulate the human hand. Each fingertip has a six-axes force sensor attached to it. The surface of the entire hand is covered with a distributed tactile sensor which has 624

force-detecting points made of conductive ink (figure 2.5.b). The resistance of the ink changes according to the pressure applied on it and through a special interface board is transmitted to a PC where signal processing is performed. Finally, a tactile image of the object grasped by the hand is displayed on the computer screen.



a) Robot hand. *b) Distributed tactile sensor.*

Figure 2.5. *Dexterous anthropomorphic robot hand (reprinted from [11]).*

2.3.2. Remote Palpation Instruments for Minimally Invasive Surgery

A remote palpation system used in minimally invasive surgery [12] is presented in figure 2.6. The surgical instrument uses miniature tactile sensors that measure the pressure applied on the human internal organ. The surgeon operates a handle outside the patient's body. The motions of the surgeon's fingers are transmitted to the tip of the instrument, where the tactile sensors are located. A dedicated computer system gathers the tactile information and applies specific signal processing algorithms. Finally, the tactile feedback device – the so called "tactile display" – recreates the remote pressure distribution at the surgeon's fingertips, so that the surgeon can "feel" like he or she is actually touching the tissues inside the patient's body.

The "tactile display" consists of a pad on which a line of 10 pins are raised against the surgeon's fingertips. The pins can be individually actuated: an electric impulse pushes a pin up by 3 mm and produces over 1 N of force, which is quite enough to provide the feeling of touch.

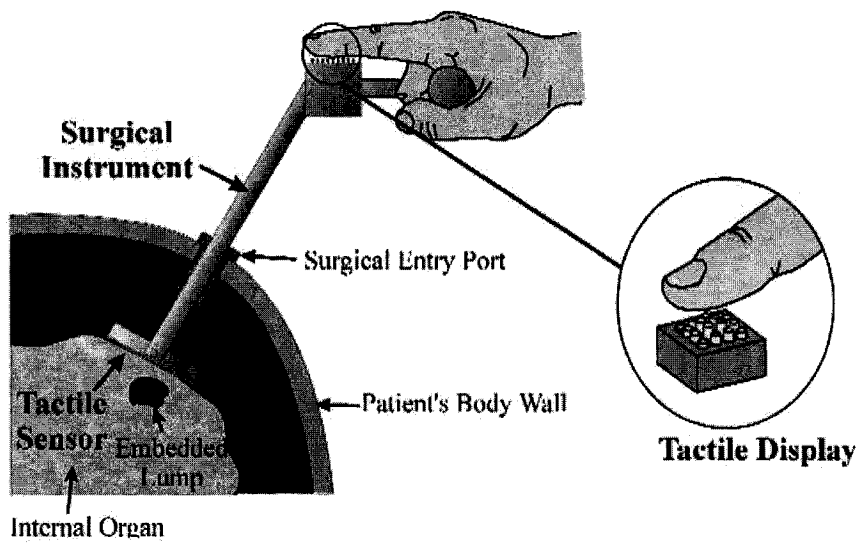


Figure 2.6. *Remote palpation instrument (reprinted from [12]).*

Remote palpation techniques will increase the safety and reliability of the common minimally invasive procedures used these days.

2.3.3. Telehaptics

Researchers at Massachusetts Institute of Technology (MIT) and University College London completed a task in a long distance collaborative virtual environment, not only by seeing and hearing, but also by "feeling" each other manipulations of a box that each team was able to see on their computer screen [13].

The experimental set up on each side of the Atlantic consists of a computer and a miniature robotic arm used instead of a mouse. On the computer screen, each user enters a 3-D virtual room in which a small box is located. Each user is represented in the room by a small

pointer. Manipulating their robotic arms, the users collaborate to lift the box in the virtual room because they can "feel" each other's actions like they were in the same real room. As the pointer is touched to the virtual box, the users can "feel" the texture of the box. The users attempt to lift the small virtual box together by applying force on its sides. After one user lifts a side of the box, the other one lifts the opposite side. Finally, the users end up holding the box together and feeling each other further manipulations.

2.3.4. Tactile Sensor for the Material Properties Recognition

The smart tactile sensor presented in [14] is able not to detect the presence or recognize the shape of the object it senses, but rather to identify the material the object is made of. The object is first touched, and then stimulated with known signals. Finally, the signals gathered by the probe are analyzed and compared to a model in order to classify the unknown material.

Two different architectures were developed. Figure 2.7 shows the "twin device 'faced' sensor and actuator", in which the actuator and sensor are placed opposite each other. The actuator stimulates the surface of the unknown material with an acoustic signal, whereas the sensor receives the response signal that travels through the surface of the object. The material properties are then extracted from the received signal. Both actuator and sensor are bimorph piezoceramic elements. This architecture is sensitive to longitudinal and transverse waves.

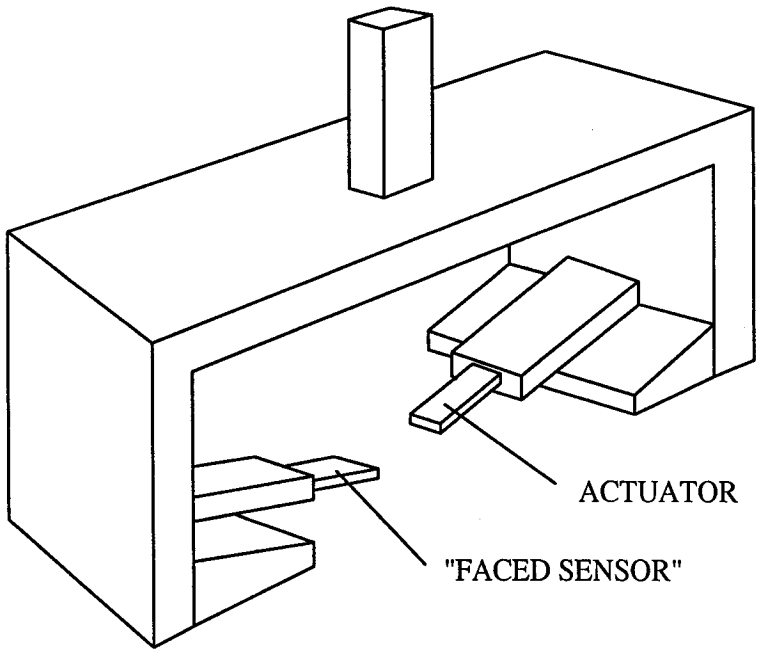


Figure 2.7. *Faced structure of the tactile sensor for the material properties recognition (adapted from [14]).*

The faced structure is driven with bursts of an amplitude-modulated square wave signal.

Figure 2.8 presents the second architecture, the so called "three fingered bivalent probe", which is more sensitive to the viscoelasticity characteristic of the unknown material. In this case, the internal piezoelectric element is the sensor, whereas the two external elements act like actuators.

In this case the actuator uses a long impulse to transfer a single pressure pulse to the surface of the material and examine its elastic reaction.

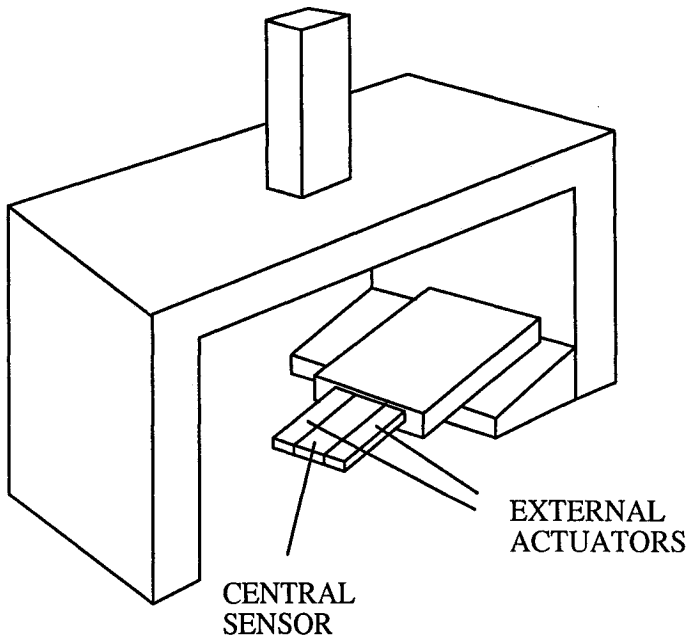


Figure 2.8. *Three-finger structure of the tactile sensor for the material properties recognition (adapted from [14]).*

Linear models of a set of different materials have been obtained. Fuzzy logic is used with both actuators to perform experiments in order to recognize the material of the explored surfaces. The materials were classified by taking into account their time and frequency response, based on their elastic properties.

The "three-finger" structured sensor distinguishes among only four main classes of materials: stone, metal, wood, and plastic, whereas the "faced" structure can classify all the six main categories considered in the experiments: stones, aluminium, iron, plastic, pieces of wood, and glass.

Applications for this type of sensors range from robotics to the field of recycling, from archaeology to humanitarian demining, and in all other fields where non-destructive technologies must be used to analyse materials.

2.4. Discussion

There have been numerous studies on developing tactile sensors. Each method has advantages and disadvantages, which make them more or less suitable for a specific application. When choosing one method over another, a system designer has to take into consideration the characteristics of the tactile sensor involved. Some of them are:

- Spatial resolution;
- Repeatability;
- Accuracy;
- Precision;
- Linearity;
- Response time;
- Hysteresis;
- Size;
- Mass;
- Force range;
- Presence of over force protection;
- Ability to bend;
- Reliability.

Chapter 3

Tactile Sensor System

This chapter presents the structure and working principle of the developed tactile sensor system based on a Force Sensing Resistor (FSR) transducer technology.

3.1. Force Sensing Resistor Transducer

A basic component of the Smart Tactile Sensor is a Force Sensing Resistor Array transducer [15], which is a piezoresistive sensor.

3.1.1. Specifications

The Force Sensing Resistor Array is composed of 256 pressure-sensing elements – Force Sensing Resistor transducers – arranged in a 16 by 16 matrix occupying an active area of one square inch [15]. A sandwich of two flexible polymer thick films, the FSR is a device that uses the electrical property of resistance to measure the normal force applied to its surface. Each FSR in the tactile transducer array is a variable resistor. The resistance of the sensor decreases (the conductance increases) as the force applied to the sensor is increased. The

power-law relationship between the applied force and the resistance (conductance) of the FSR is of the form [16]:

$$F \propto \frac{1}{R^{\beta}} \quad or \quad F \propto G^{\beta}$$

where:

- β is sensor design dependent;
- R is the resistance;
- G is the conductance of the sensor (the reciprocal of resistance, measured in Siemens - S).

A typical plot of resistance versus force is shown in figure 3.1 [17].

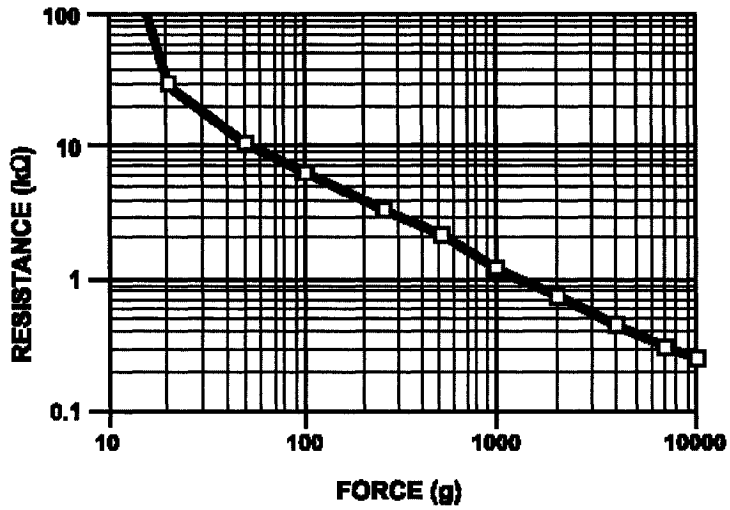


Figure 3.1. *Resistance versus force for a typical FSR (reprinted from [17]).*

We can see that the FSR acts like a switch at the low force end of the resistance versus force characteristic. The turn-on threshold that swings the resistance from 100 kΩ to 10 kΩ is design dependent (β). On the other hand, at the high force end, the response does not follow the power-law behaviour anymore, but rather saturates to a point where increases in force do not yield a decrease in resistance [17].

Figure 3.2 shows a plot of conductance as a function of force for a typical FSR sensor [17].

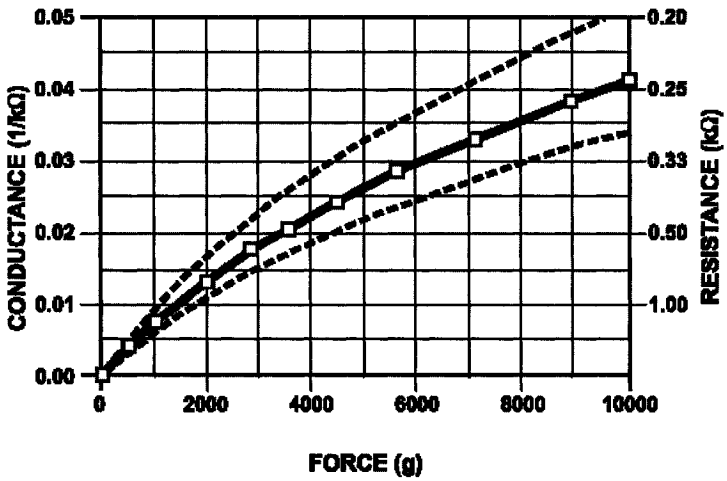


Figure 3.2. *Conductance versus force for a typical FSR (reprinted from [17]).*

This format allows the analysis on a linear scale. The shape of the curve also depends on the value of β. If β is close to 1, above the break point, the conductance is approximately linear with force, a fact that eases the design of the electronic interface.

Some of the advantages of the Force Sensing Resistor Array transducer that are making it suitable for a large range of applications are [15, 16, 17, 18, 19]:

- High sensitivity: for a pressure range of 1 N/cm^2 – 100 N/cm^2, such an element will change its resistance by three orders of magnitude;
- The force resolution is better than ±0.5% of full used force;
- Its impedance is nearly purely resistive;
- It is very thin (0.25 mm);
- It is a rugged and durable device (10,000,000 actuations);
- It is relatively insensitive to humidity;
- Its behaviour is not affected by mechanical vibrations and cross talk, thermal changes, and acoustic noise;

- It is passive with respect to Electromagnetic Interference (EMI) and Electro Static Discharge (ESD);
- It is environmentally resistant;
- It is relatively inexpensive.

As a disadvantage, Interlink admits that the FSR is not suited for precision measurements since its accuracy ranges from approximately ±5% to ±25%.

Measurements made during the testing of the tactile transducer array detected no electrical cross talk between FSRs. The components chosen during the design of the electronic interface were in part determined by the minimum and maximum resistance values of the FSRs. The measurements yielded to a stand-off resistance in the MΩ range and a minimum resistance of approximately 200 Ω.

3.1.2. Functional Description

As stated earlier, the tactile transducer array is composed of 256 Force Sensing Resistors, which are acting like variable resistors. For simplicity, we will consider the electrical representation of only a 4x4 section of the tactile transducer array (shown in figure 3.3). This simplified sensor has 4 rows (1 to 4) and 4 columns (A to D) of FSRs, which are named accordingly. For instance, FSR 2B is the FSR located on row 2 and column B. There are only 8 outputs (4 rows and 4 columns) for a total of 16 sensors. Thus, the 16x16 tactile transducer array has 32 outputs (16 rows and 16 columns) for a total of 256 sensors [16].

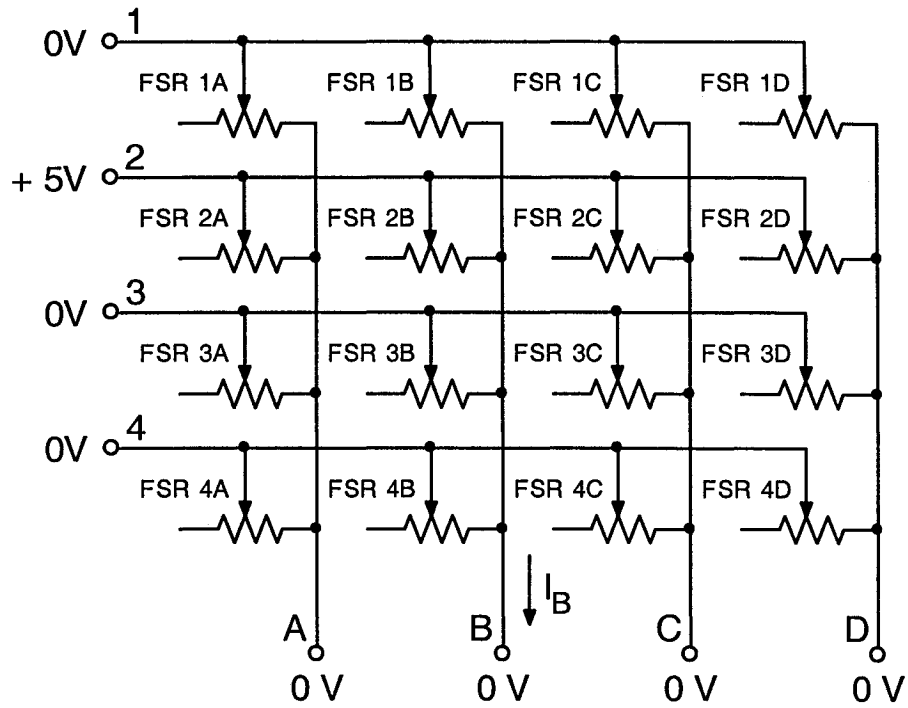


Figure 3.3. The electrical representation of a 4 x 4 section of the tactile transducer array.

The goal is to find out the value of the resistance of a force sensing resistor, for example FSR 2B. One method to doing that is to simply measure the resistance between points 2 and B, leaving all the other nodes floating. This method gives an accurate value of FSR 2B if a pressure is applied to the measured sensor only, and not to any other sensors. But, if a pressure is applied to more than one FSR, then the measured resistance will not be the real one.

Suppose the pressure applied to the tactile transducer array is over FSR 2B, FSR 2C, FSR 3B, and FSR 3C, we try to measure FSR 2B by connecting a multimetre between points 2 and B. Because all the other points are floating, it is easy to see that we actually measure a combination of the four resistors: FSR 2B is in parallel with the serial combination of FSR 2C, FSR 3C, and FSR 3B. If these four resistors are equally pressed so that they have all an equal resistance of 10 kΩ, then the effective resistance measured between nodes 2 and B would be 7.5 kΩ. Since the actual value of FSR 2B is 10 kΩ, the measured resistance has an error of 25%, which is not acceptable.

Therefore, if more than one resistance have pressure applied to them, the reading will not represent the real value of the measured resistor. In conclusion, this method is not applicable because in reality we will always have force applied to more than one FSR simultaneously.

The following method, used in the design of the electronic interface to the tactile transducer array, eliminates the problem of effective resistance so that only the real resistance is measured. It consists in isolating the FSR that is to be measured by applying 5 V to one node and 0 V to all the other nodes. In figure 3.3 the force sensing resistance to be measured is FSR 2B. The current I_B flowing through column B is only the current flowing through FSR 2B because of the 5 V drop across it. There is no current through any other resistor since all other FSRs have no voltage drop across them. The idea of this method is to measure the current rather than directly the resistance. Therefore, the resistance of FSR 2B can easily be calculated:

$$R_{FSR\,2B} = \frac{5V}{I_B}$$

In the same way, all other FSRs can be determined by applying 5 V to the proper row and measuring the current that flows out from the corresponding column. Section 3.3 will show how this method was implemented.

3.2. Tactile Sensor Using a Force Sensing Resistor Transducer

The developed tactile sensor system is based on a 16-by-16 Force Sensing Resistor transducer. An elastic overlay is placed on top of the probing side of the FSR transducer [5]. Under the pressure from an external bias force the elastic overlay deforms, "moulding" the geometric profile of the explored object surface [20]. The induced contact forces are transmitted through the elastic overlay to the FSR transducers.

Figure 3.4 shows the functional steps occurring in the measurement sequence when this type of tactile sensor is used as a geometric probe. The measurement is identified as the

local 3-D geometric profile (shape) of the touched object surface. The exerted force provides the external source of energy needed to impose the necessary contact between the tactile probe and the explored object surface. The elastic pad acts as o low-pass filter and displacement-to-force converter, while the force sensor array accomplishes a 2-D sampling and a 1-D force sensing [5].

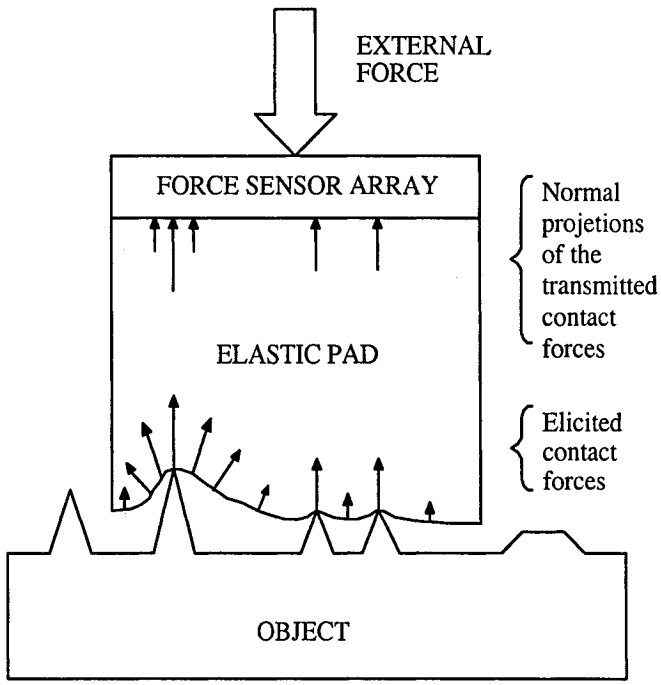


Figure 3.4. *Operating principle of the tactile sensor (adapted from [5]).*

There is a complex and difficult to control relation between the displacement and the effectively measured strain in the elastic pad [21, 22, 23] (figure 3.5).

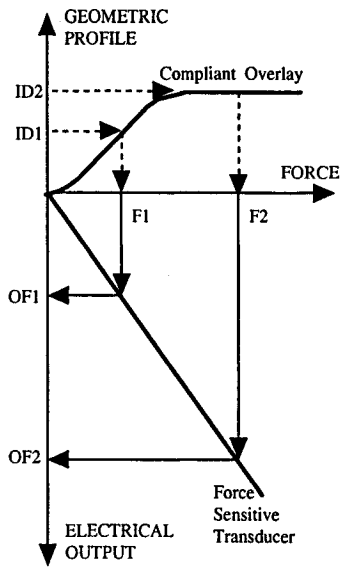


Figure 3.5. I/O characteristics of a force sensitive transducer and its elastic/compliant overlay (reprinted from [23]).

The elastic overlay has a protective damping effect against impulsive contact forces [24, 25], and its elasticity resets the measuring system when the sensor ceases to touch the object.

It is demonstrated [7] that the fully geometrically constrained one-piece elastic pad cannot be compressed in the normal z direction, and thus it cannot take on the shape of the touched object surface.

In order to avoid the inherent blurring effect of the one-piece elastic pads, we are using a custom-designed elastic overlay consisting of a thin membrane with protruding round tabs sitting on top of each node of the tactile sensor array providing a *de facto* spatial sampling as shown in figure 3.6.

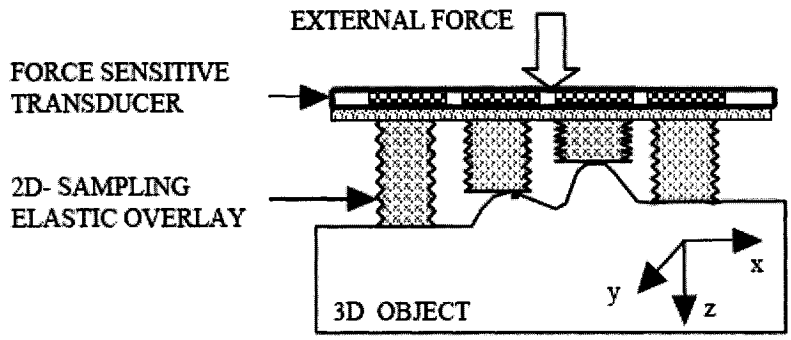


Figure 3.6. *Protruding round tabs of the elastic overlay provide spatial sampling (adapted from [7]).*

Based on recommendations made in [26, 27], we are using circular tabs, which occupy 50% of each 2D sampling area. This allows each tab to expand laterally in the x and y directions without any stress allowing for a proportional relationship between the displacement in the normal z direction and the resulting stress component in each tab. Figure 3.7 shows the experimental strain/stress average response of a single tab of this elastic overlay [7].

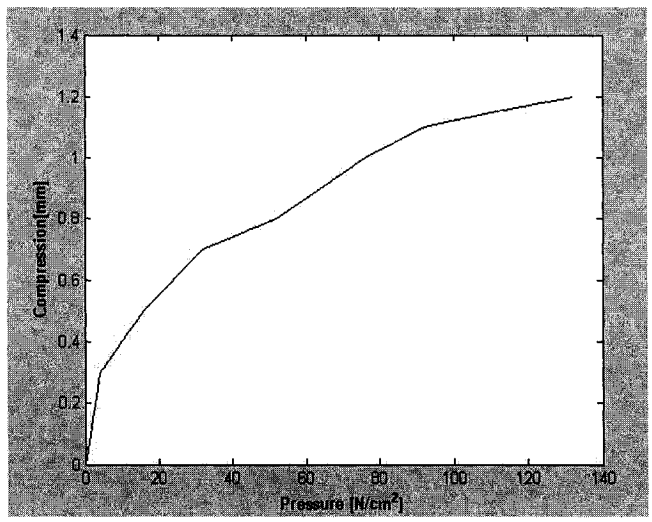


Figure 3.7. *Strain/stress single-tab response of the elastic overlay (adapted from [7]).*

As a result, the tactile probe output is a 16x16 array of data that represent normal displacement components of the 3D geometric profile of the investigated object surface $[z(i,j) \mid i=1,\ldots,16; j=1,\ldots,16]$, where i and j are the row and column coordinates of the tactile sensor array.

The Force Sensing Resistors that compose the tactile sensor sense compression forces and thus should be placed on a rigid backing. For this reason, the tactile sensor is placed in an aluminium protective case. The resulting design of the Force Sensing Resistor Tactile Sensor is shown in figure 3.8.

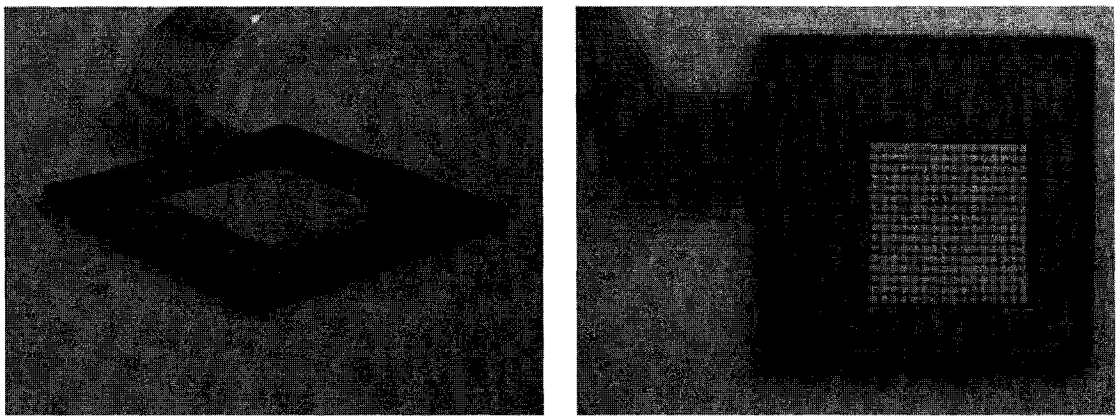


Figure 3.8. *Force Sensing Resistor Tactile Sensor case and elastic overlay.*

3.3. Hardware and Software Interface Functions

The basic design goals of the electronic interface are the ability to deal with the tactile sensor's large resistance range and to avoid electrical cross talk between taxels.

Following the recommendations in [15], I designed the tactile sensor interface circuit whose block diagram is presented in figure 3.9. Two analog multiplexers, one for the selection of the row address (applying 5 V) and another for the selection of the column address (measuring the current exiting the column), allow random access to any individual force sensitive resistor within the 16x16 Force Sensing Resistor Tactile Sensor. The resistance of each selected FSR element is measured by an A/D converter onboard a 16-bit PIC microcontroller. The microcontroller also provides the following sensor control functions: FSR address selection and serial data communication interface.

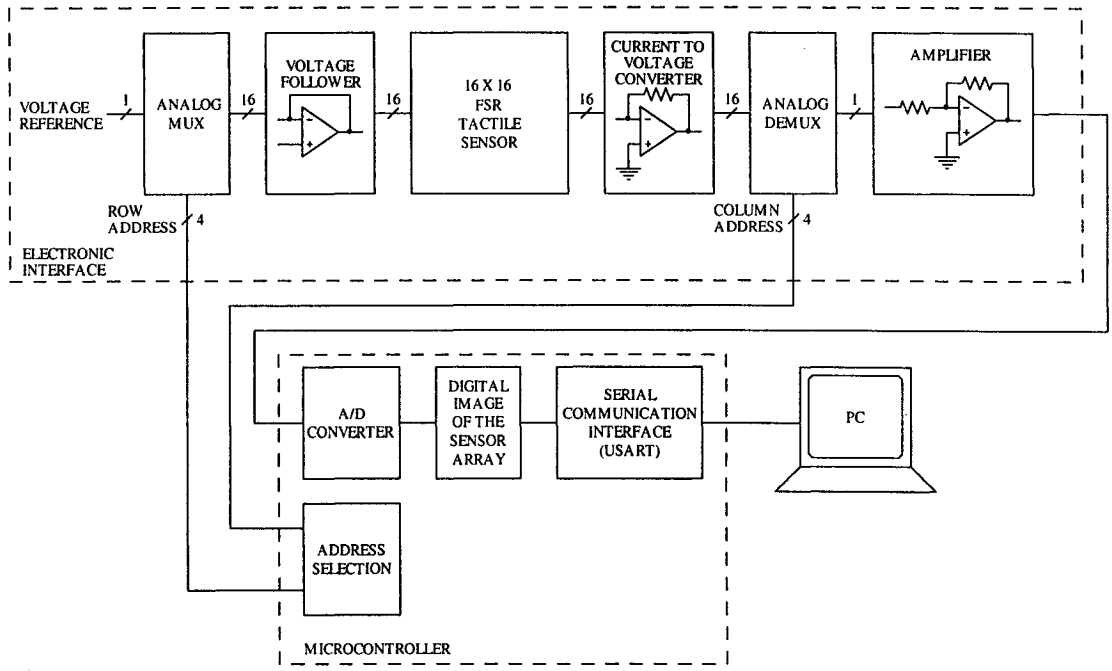


Figure 3.9. Block diagram of the tactile sensor interface.

3.3.1. Analog and Digital Electronics

3.3.1.1. Current-to-Voltage Converter

We already know that the force sensing resistor (FSR) is acting like a variable resistor, whose resistance (or conductance) has to be measured to determine the force applied to it. For dynamic FSR measurement applications, the current-to-voltage converter circuit is used [17] (figure 3.10), which is basically just an inverting amplifier with a feedback resistor (R68). The output voltage of this circuit is inversely proportional to the resistance of the force sensing resistor. But, as previously stated, the FSR resistance is inversely proportional to the applied force. Thus, the output voltage is actually directly proportional to the force applied to the FSR. In other words, the current-to-voltage converter linearly increases the output voltage for increases in applied force. In conclusion, the current-to-voltage converter circuit performs two tasks: first, it applies 0 V to the column node because of the virtual ground created by the

operational amplifier; and second, it converts the measured current into a voltage so that the A/D converter can measure it.

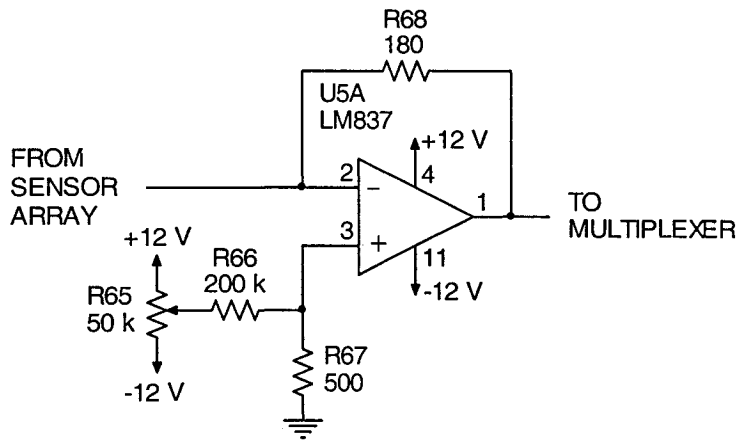


Figure 3.10. *Current-to-voltage converter.*

The operational amplifier was chosen so that it can sink a current above the maximum current that results when FSR resistance has the smallest value (200 Ω):

$$I_{\max} = \frac{5V}{200\Omega} = 25mA$$

The resistors connected to the positive input 3 of the operational amplifier – R65, R 66, and R67 – are used for the offset adjustment of the current-to-voltage converter circuit.

3.3.1.2. Voltage Follower

As mentioned in section 3.1.2, the method of measuring the resistance of an FSR of the tactile sensor consists of applying 5 V to the row node in which the FSR is located and 0 V to all other nodes. As mentioned in section 3.3.1.1., at the column nodes the current-to-voltage converters provide the 0 V needed because of the virtual ground effect. In order to obtain accurate measurements of the FSR resistance, the 5 V applied to one of the row nodes needs to be constant even if the current requirement changes. The current change as pressure is applied to the sensors: low pressure means low current, whereas high pressure means high current (to

a maximum of 25 mA). In order to keep the voltage constant, a voltage follower is implemented. Figure 3.11 shows the schematic diagram of a voltage follower with transistors added in the feedback loop to provide extra current drive.

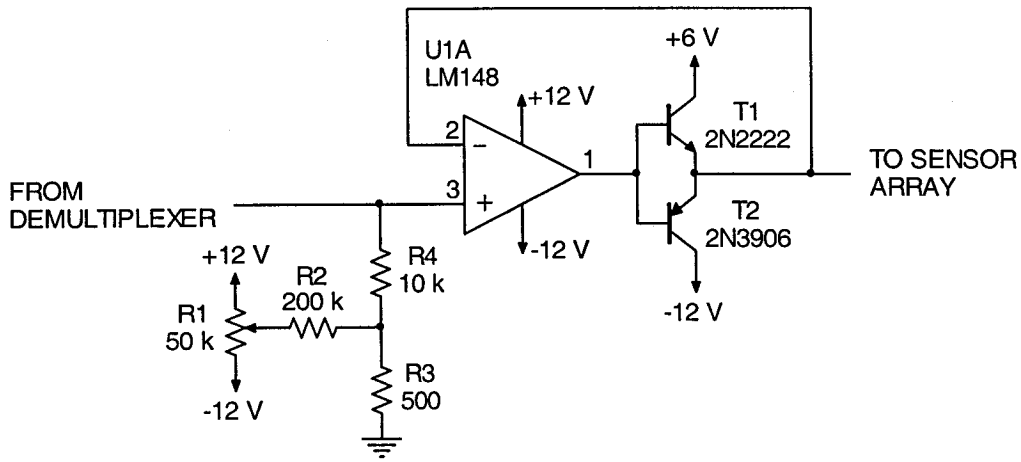


Figure 3.11. *Voltage follower.*

Without using the voltage follower the following situation would occur: if the current is increased, due to the source resistance, the output voltage would decrease.

Resistors R4 and R3 let the input fall to ground when the input is left floating, providing the 0 V input when 5 V is applied to another node.

Suppose all 16 FSRs of a row where the 5 V is applied are pressed, so that all the resistors have the minimum value of 200 Ω. Each FSR is traversed by a current of 25 mA; therefore the voltage follower circuit would have to supply a total current of 16 x 25 mA = 400 mA. Because the maximum output current of the operational amplifier is around 20 mA, transistors T1 and T2 were added in the feedback loop to increase the current that can be supplied. The NPN transistor needs to be able to supply a maximum current of at least 400 mA. In order to reduce the power consumption of transistor T1, 6 V (instead of 12 V as in the case of T2) are applied to its collector.

The resistors connected to the positive input 3 of the operational amplifier – R1, R 2, R3, and R4 – are used for the offset adjustment of the voltage follower circuit.

3.3.1.3. Variable Gain Amplifier

The output stage is placed between the multiplexer and the A/D Converter of the microcontroller. It consists of two voltage followers (buffers) and an inverting variable gain amplifier (figure 3.12).

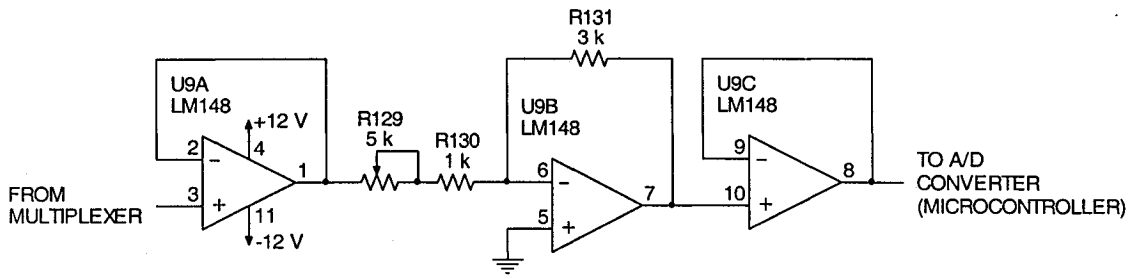


Figure 3.12. Variable gain amplifier.

The first voltage follower acts as an impedance converter, eliminating the effect of the resistance in the multiplexer. Since the current-to-voltage converter gives a negative output voltage, an inverter is needed in order to obtain a positive voltage value. The gain of the inverter can be adjusted to effectively use the dynamic range of the A/D converter.

3.3.1.4. Analog Multiplexer/Demultiplexer

Analog multiplexers/demultiplexers are needed in order to select the nodes that are to be measured. A 1 to 16 analog demultiplexer applies the 5 V to the desired voltage follower. The input is a 5 V regulated voltage. A 16 to 1 analog multiplexer selects the desired signal to be read by the A/D Converter. The switching is controlled by the microcontroller via the external output lines.

3.3.2. PICmicro Microcontroller

The main functions of the microcontroller in figure 3.9 are to obtain a digital image of the tactile sensor and to send it to a PC. To achieve these goals, the microcontroller has to be able to:

- Select the rows and columns of the tactile sensor so that each FSR is evaluated;
- Convert the measured analog voltages into digital values;
- Store the 256 value array into memory;
- Transmit the array through a serial communication port to a PC.

I chose a cheap, but at the same time, reliable microcontroller that not only meets but also exceeds these expectations: the PIC18F452 from Microchip Technology. The key features of this microcontroller are [28]:

- High performance RISC CPU;
- Operating speed: up to 40 MHz clock input;
- 32K FLASH Program Memory;
- 1536 bytes of Data Memory (RAM);
- 256 bytes of EEPROM Data Memory;
- In-Circuit Serial Programming via two pins;
- 8 channels of 10-bit Analog to Digital Converter;
- Universal Synchronous Asynchronous Receiver Transmitter (USART/SCI) with 9-bit address detection.

3.3.2.1. Hardware

The schematic diagram is shown in figure 3.13. It consists of a PIC18F452 microcontroller (U12) connected to a Maxim MAX232 level-translator chip (U13).

The decoupling capacitor (C1) filters the voltages within the PIC microcontroller. During the transition of the electronic circuits from low-to-high current requirements and from

one state to another, the voltages inside the chip will fluctuate, which could cause the chip to behave unpredictably, to reset or lock up [29]. The decoupling capacitor is placed as close as possible to the chip.

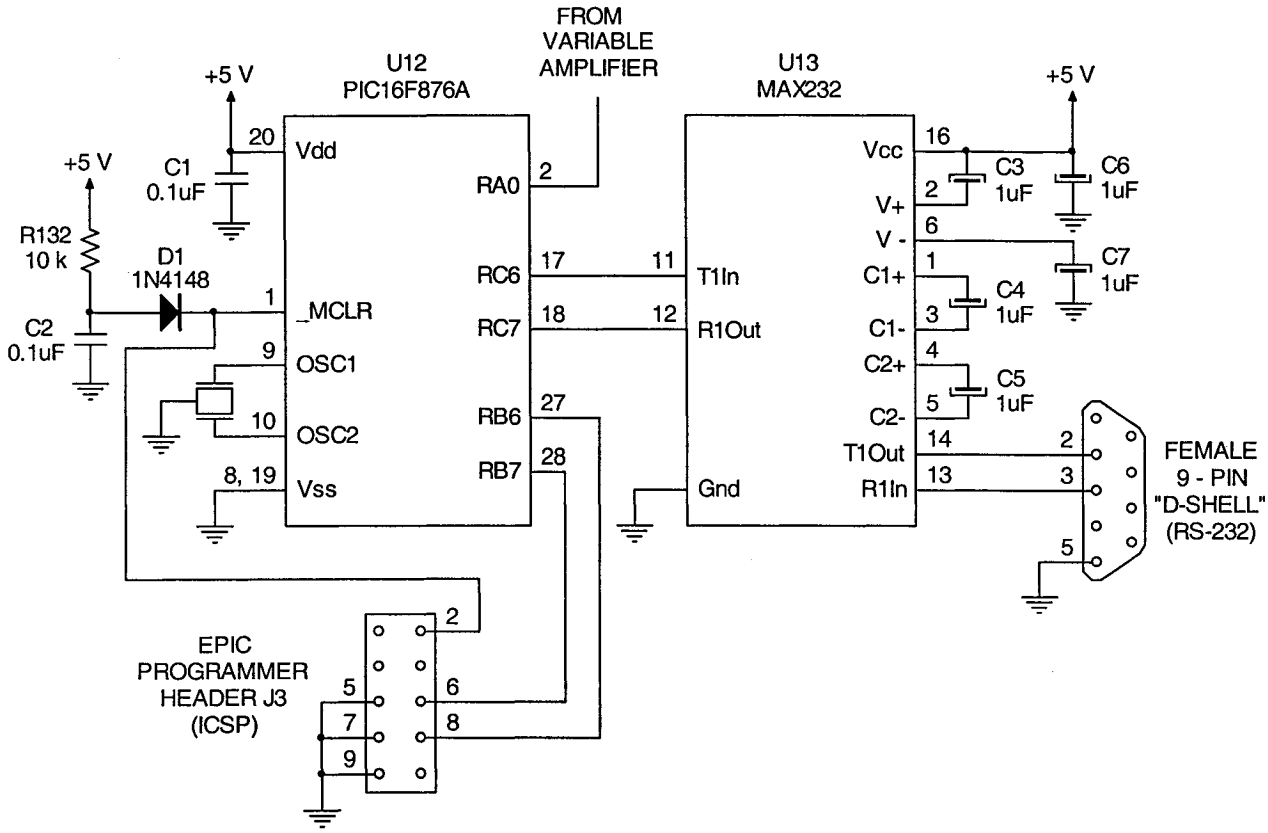


Figure 3.13. *PICmicro microcontroller schematic diagram.*

The reset circuit is very simple: it consists of a resistor (R132) and a capacitor (C2).

The microcontroller can be operated in eight different oscillator modes [28]. I chose the XT Crystal/Resonator mode and used a 4 MHz ceramic oscillator connected to pins 9 and 10 of the PIC18F452 chip. Ceramic resonators are much more robust and cheaper than crystal resonators. Their accuracy is not as good as crystals (usually accurate to 0.5 percent versus a crystal accuracy of 0.02 to 0.1 percent) but for the purpose of this thesis this accuracy is enough.

The MAX232 chip converts TTL/CMOS data levels into RS-232 data levels with negative voltage for the mark (1) and a positive voltage for the space (0). MAX232 requires five 1.0 μF capacitors (C3 to C7). Pin 14 is the output (connected to pin 2 of the 9-pin "D-Shell" connector), whereas pin 13 is the input (connected to pin 3 of the connector). Connection between this connector and the PC is done using a straight serial cable.

3.3.2.2. Software

The easiest way to write the assembly code for the PICmicro microcontroller was to use the MPLAB Integrated Development Environment (IDE) from Microchip, which is an integrated toolset for the development of embedded applications employing Microchip's microcontrollers. Its editor let me write the assembly code. Then, with the integrated assembler I built and assembled my project, and with the built-in simulator and debugger I tested my code.

A very useful feature of PICmicro microcontrollers is the capability of being serially programmed. Using a simple programmer (the EPIC Plus PICmicro MCU Programmer from microEngineering Labs, Inc), the chip can be programmed after being wired into the application. The EPIC Plus Programmer connects to a PC compatible parallel printer port and includes an assembler and programming software. The programmer uses two lines for clock and data and two other lines for ground and the programming voltage. Diode D1 isolates the In Circuit Serial Programming circuit from the reset circuit.

The main tasks of the microcontroller software are to read the raw data from the Force Sensing Resistor Tactile Sensor, convert it into a binary sequence, and send the results to the PC via the communication link.

The following steps are performed:

- PORTD that contains the address of the tactile sensor node is initialized by setting all its bits to output.

- The A/D converter is set up to 10-bit conversion. Only the most significant 8 bits will be retained.
- The USART (Universal Synchronous Asynchronous Receiver Transmitter) is set up to 8-bit, 19.200 bps, and asynchronous serial communication.
- The communication port is polled for the "start acquiring" signal. Upon receiving it, the next address of the tactile sensor is supplied on PORTD and the analog value of the FSR is read.
- Next, the A/D conversion is completed and the value written to a memory location.
- After all 256 FSRs are read and memorized, their digital values are sent to the PC via the communication port.
- Once all of them are sent, the routine jump back to the polling for the "start acquiring" signal line.

The flowchart shown in figure 3.14 describes the actions of the microcontroller assembly code.

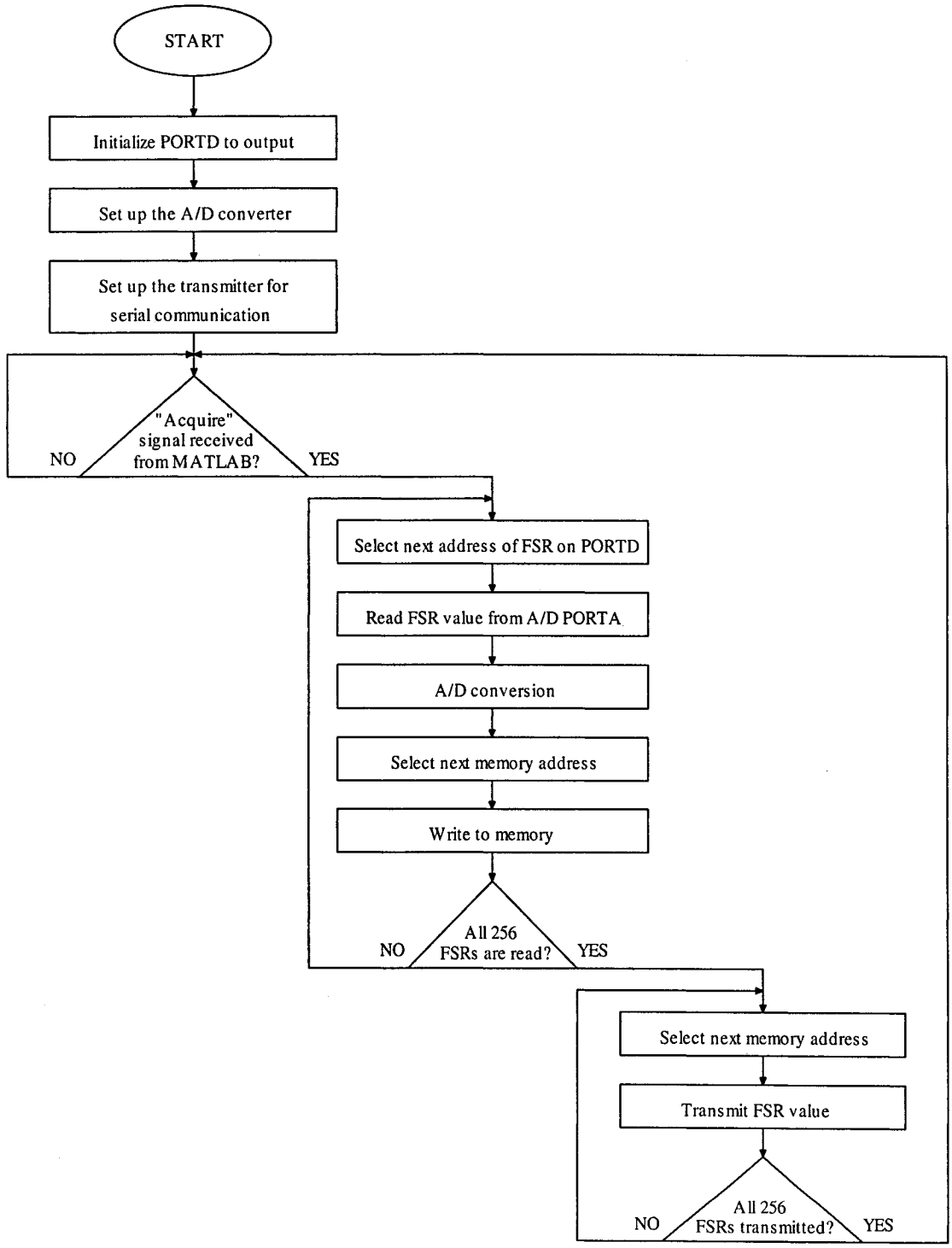


Figure 3.14. Microcontroller assembly code flowchart.

The complete assembly code listing is shown in Appendix B.

3.3.3. Hardware Implementation

The schematic diagram of the electronic interface is shown in figure 3.15. Appendix A contains the complete set of schematic diagrams including the voltage follower, the current-to-voltage converter, and the regulated power supplies.

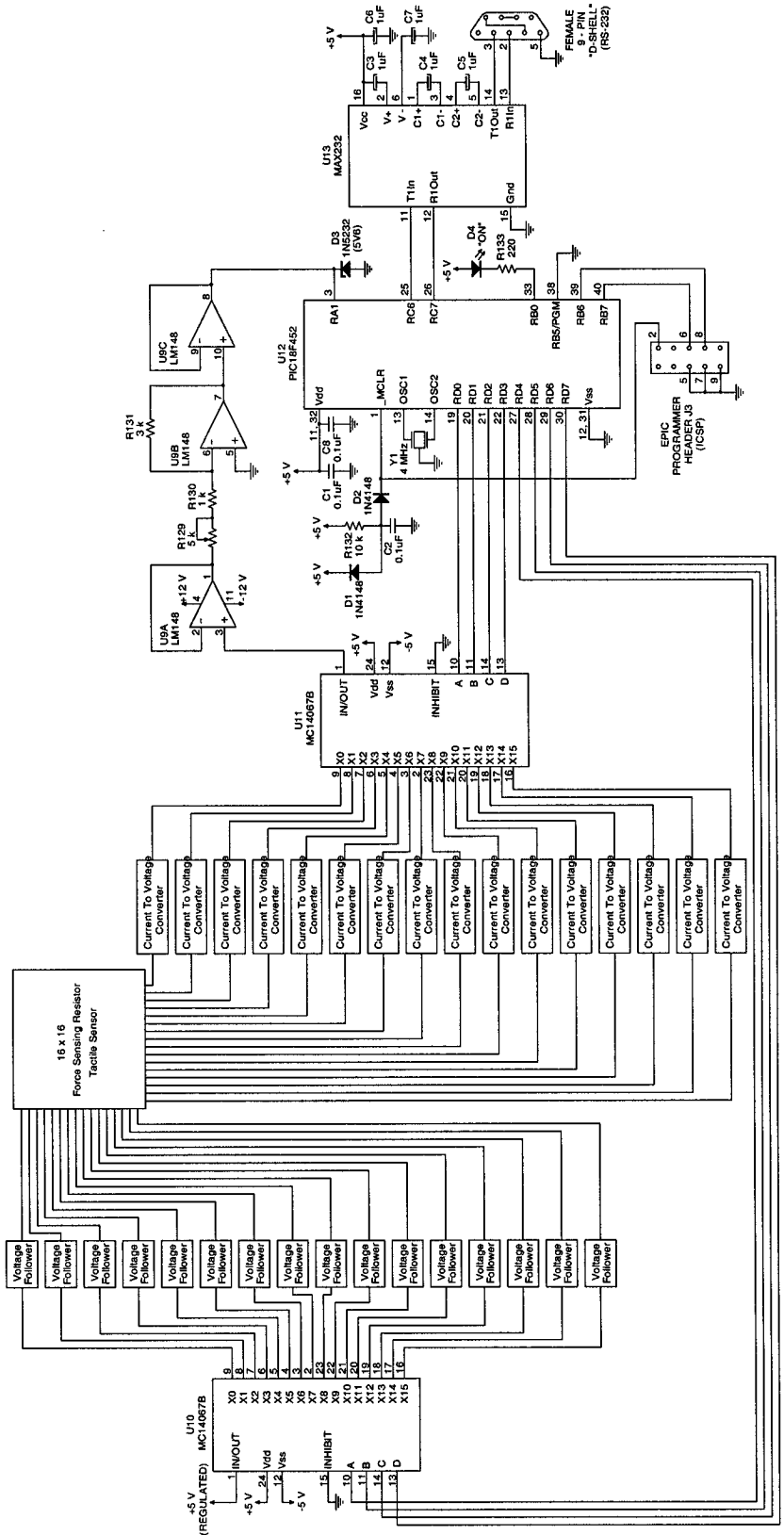


Figure 3.15. *Schematic diagram of the Tactile Sensor Electronic Interface.*

I built the electronic circuits on a breadboard that let me perform many tests. I fitted the circuit into a commercial aluminium box. Figure 3.16 contains a picture of the implementation of the electronic interface circuit, whereas figure 3.17 shows a picture of the electronic box and the external power supplies.

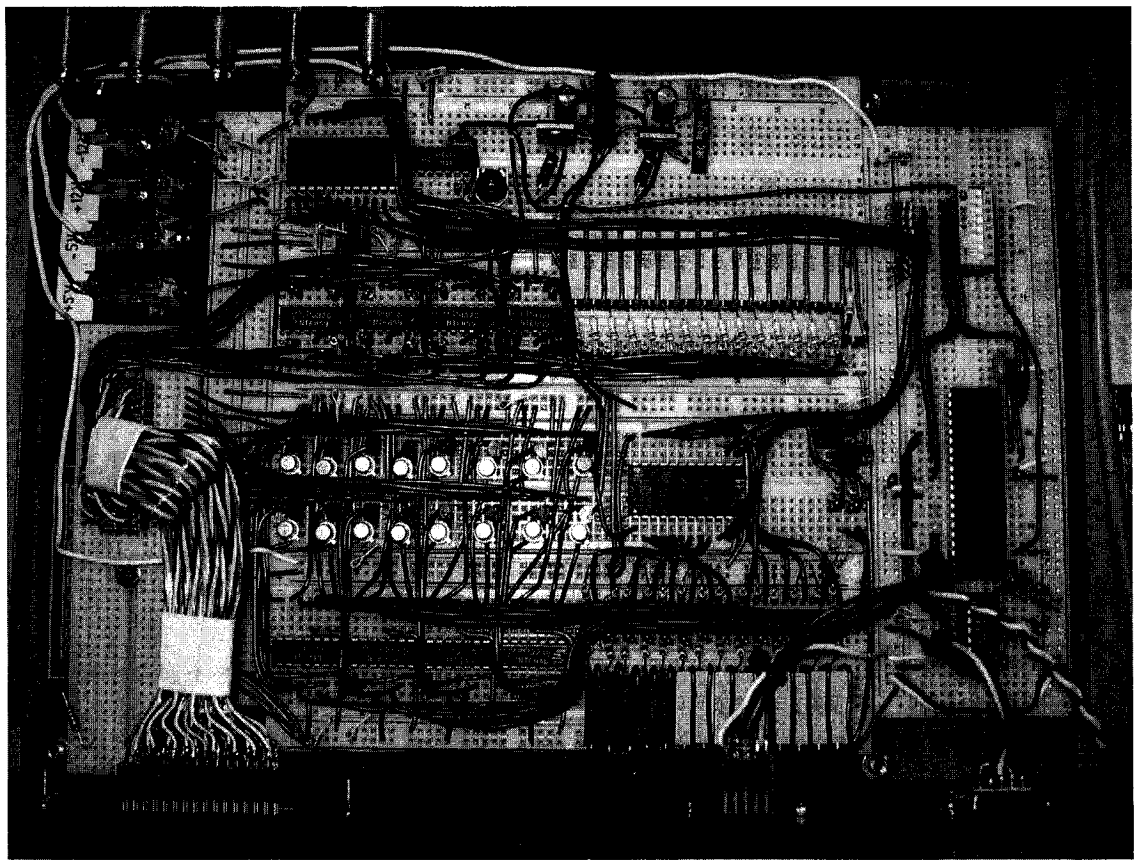
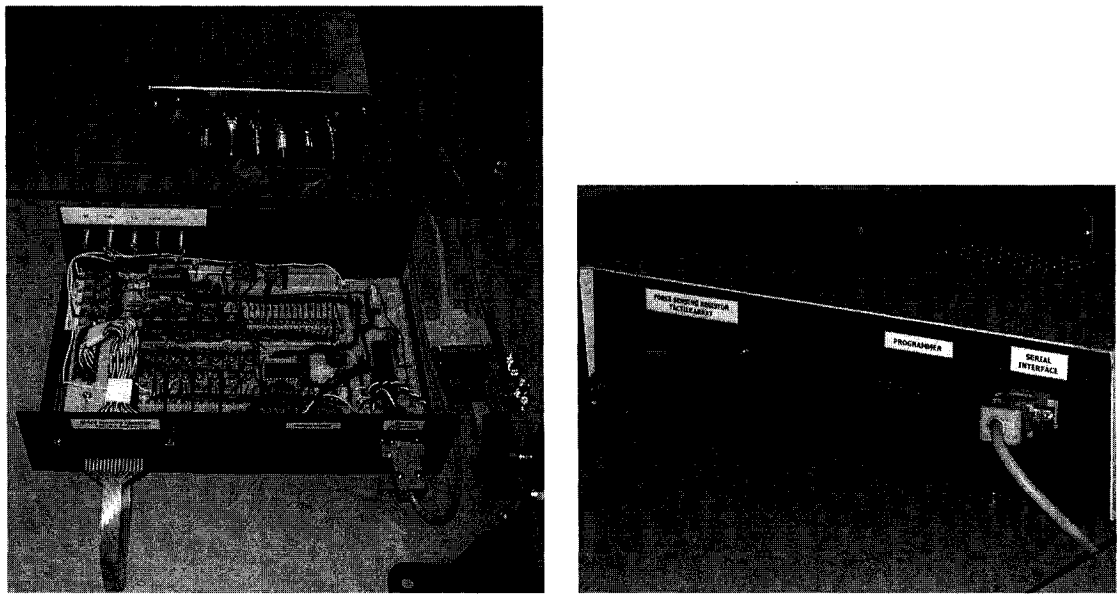


Figure 3.16. *Tactile sensor electronic interface built on a breadboard.*



a) Electronic box (open) and power supply. *b) Electronic box (front view).*

Figure 3.17. *Electronic box.*

The electronic interface was tested under extreme power supply conditions: the PICmicro microcontroller and the level-translator chip were tested over the 4.2 to 5.5 VDC operating range specified in [28], whereas the operational amplifiers were subjected to an ecquivalent range of 10.08 to 13.2 VDC. Practically, the output signals were insensitive to the above mentioned power supply variations.

Chapter 4

Tactile Image Acquisition, Processing, and Display Functions – PC Application Software

As previously stated, the Tactile Sensor Electronic Interface (microcontroller) is able to send, through the serial data communication port, the raw data read from the Force Sensing Resistor Tactile Sensor. The analog resistance of each FSR element is converted into an 8-bit digital value – a byte. As a result, the 3-D tactile image consists of a 16-by-16 array of bytes, which are sent to a PC as a 256-byte stream of data.

A PC application should receive, process, and display the tactile image from the Tactile Sensor Electronic Interface. To achieve this goal, the application software needs to be written in a language that, at least, supports serial communication and 3-D drawing. The tool of choice to write the PC application software is MATLAB, which not only has serial communication and 3-D drawing capabilities, but it also provides GUIDE (Graphical User Interface Development Environment), a quick and efficient way to build a Graphical User Interface. The advantage of using MATLAB also resides in the fact that it contains an image processing library of functions, some of which I used to process the raw data.

I named the application software TS GUI, an abbreviation for "Force Sensing Resistor Tactile Sensor Graphical User Interface". TS GUI is not only a Graphical User Interface that displays the raw tactile image, but it also contains all the software needed to acquire, process, and save both the initial and processed images.

Appendix C contains the source code for TS GUI.

Because I wanted to be able to demonstrate the functionality of TS GUI even without having the Tactile Sensor Electronic Interface available, I modified it into TS GUI Demo, which does not require any data-acquisition hardware. In TS GUI Demo, the raw images acquired by the tactile sensor are simulated by previously saved MATLAB workspace variables (MAT-files), whereas in TS GUI these images are actually acquired through the Tactile Sensor Electronic Interface. TS GUI fully contains the features of TS GUI Demo.

The following sections describe the functions performed by TS GUI.

4.1. The Force Sensing Resistor Tactile Sensor Graphical User Interface (TS GUI) – General Look

After successfully installing TS GUI, the user is able to run the software by entering "fsr" at the MATLAB "Command Window" prompt. The TS GUI window opens (figure 4.1). It contains all the controls the user needs for:

- Acquiring a new raw image;
- Processing (filtering) the image;
- Displaying both the raw and the filtered images;
- Importing previously saved workspace variables, exporting the results, changing the colours of the graphical interface, and getting help.

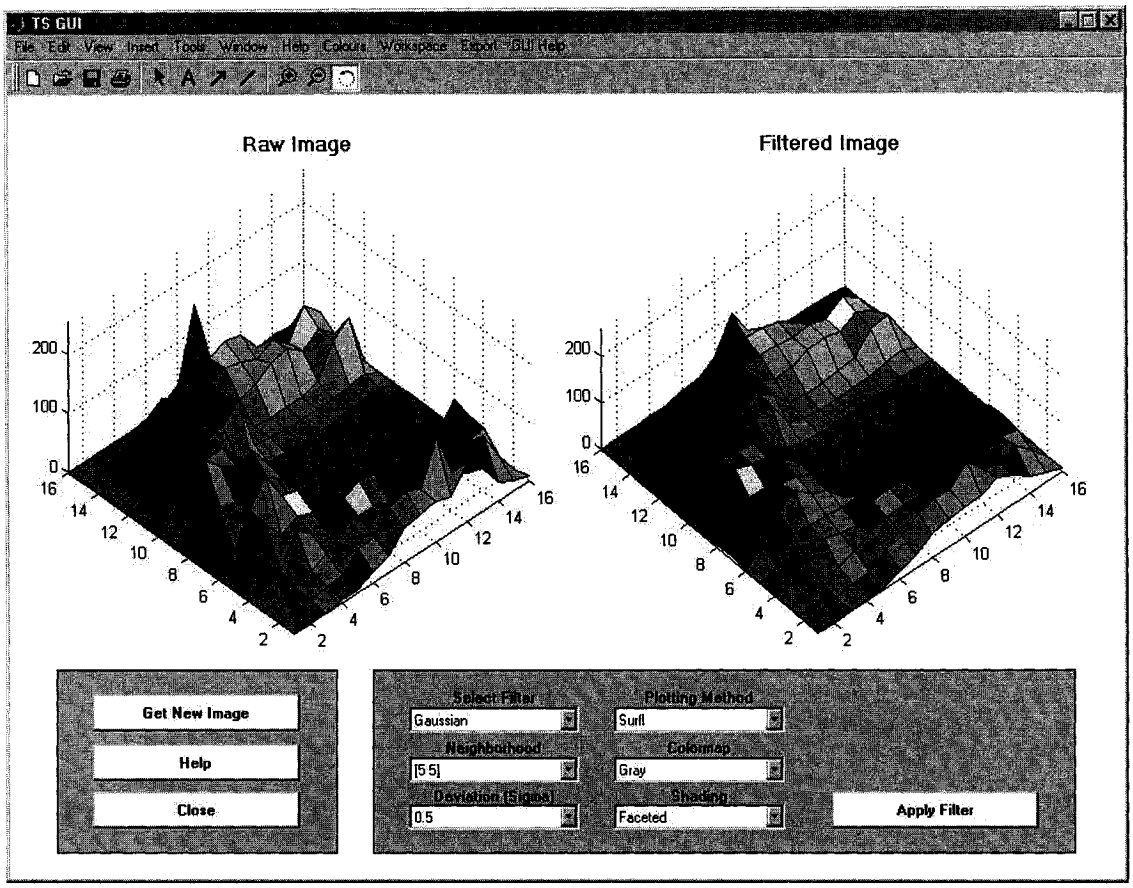


Figure 4.1. *The TS GUI window.*

Located at the top of the TS GUI window, the *Menu Bar* contains, in addition to the MATLAB built-in menus, four custom made pull-down menus: *Colours* (used for changing the colours of the graphical interface), *Workspace* (used for importing raw images and saving workspace variables), *Export* (used for saving graphical representations of the results), and *GUI Help* (used for getting help about TS GUI).

Under the *Menu Bar*, there is the MATLAB built-in *Tool Bar*, which contains commands that let the user open, save, print, and edit the plots (like inserting text, arrows, or lines); other useful commands let the user change the view of axes by either zooming in and out or rotating the 3-D axes.

The middle area of the TS GUI window contains the *Raw Image Axes*, in which the acquired or imported raw image is drawn, and the *Filtered Image Axes*, the place where the filtered image is drawn.

The *Left Frame* is a box that encloses three user interface controls: the *Get New Image* push button (acquires a new raw image), the *Help* push button (opens the Help file), and the *Close* push button (closes the application software).

The *Right Frame* is a box that encloses the *Select Filter Area* pop-up menus, the *Welcome Text* (or the *Apply Filter Text*), and the *Apply Filter* push button.

The *Select Filter Area* contains the filtering process related user interface controls: the *Select Filter*, the *Neighborhood*, the *Deviation (Sigma)*, the *Plotting Method*, the *Colormap*, and the *Shading* pop-up menus. Filtering is detailed in section 3.3.3., whereas displaying is presented in section 3.3.4.

Two different texts could appear in the upper right corner of the *Right Frame*. The blue *Welcome Text* only appears after opening the *TS GUI Demo* window and shows the user the next step to take, i.e. acquiring a new image. The red *Apply Filter Text* appears immediately after acquiring or importing a raw image into the *Raw Image Axes*. It shows the user the next step to take, i.e. filtering the raw image.

4.2. Tactile Image Acquisition - Principle and Examples

4.2.1. Principle

In TS GUI, when the user clicks the *Get New Image* push button, the acquisition of a new raw image is initiated. The following steps are performed:

- A serial port is created, its properties are configured, and then it is connected to the serial port device [30].

- A "start acquiring" signal is sent to the Tactile Sensor Electronic Interface. The start signal triggers the process of scanning the tactile sensor by the built in microcontroller.
- Next, the raw image is sent through the serial communication port to TS GUI.
- The input buffer is continuously checked. When it is full the serial port is disconnected and deleted [30].
- Finally, the raw image is plotted in the *Raw Image Axes* and the *"Apply Message"* is displayed in the *Right Frame*, inviting the user to apply a filter to the raw image.

The flowchart in figure 4.2 presents the actions performed in order to acquire and display a new raw image. For the MATLAB code, see the function pushbutton_get_callback in Appendix C (Source Code for the Graphical User Interface).

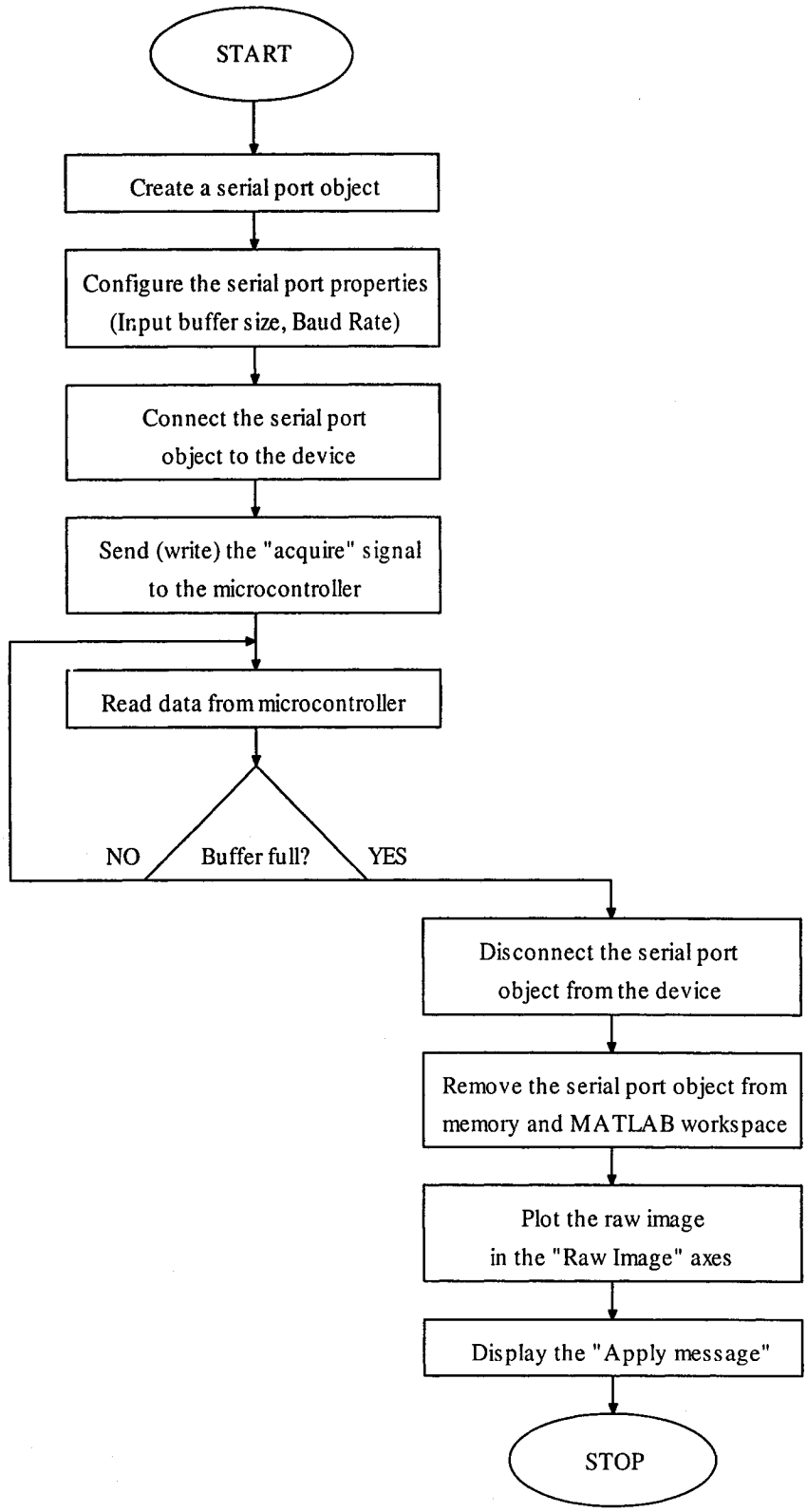


Figure 4.2. *The MATLAB acquisition flowchart.*

4.2.2. Example: Acquiring a New Image

To acquire a new image in TS GUI the user simply clicks the *Get New Image* push button.

In TS GUI, when the user clicks the *Get New Image* push button a new raw image is acquired and then plotted in the *Raw Image Axes*.

Since TS GUI Demo does not require a Tactile Sensor Electronic Interface, the *Get New Image* push button opens the *Workspace: Raw Data: Import ...* menu item, i.e. opens the *Select File* dialog box in which the user can select a previously saved workspace variable file (MAT-file). The selected file is then plotted in the *Raw Image Axes*.

An example of an acquired raw image (letter "C" from the supplied *Library*) plotted in the *Raw Image Axes* is shown in figure 4.3.

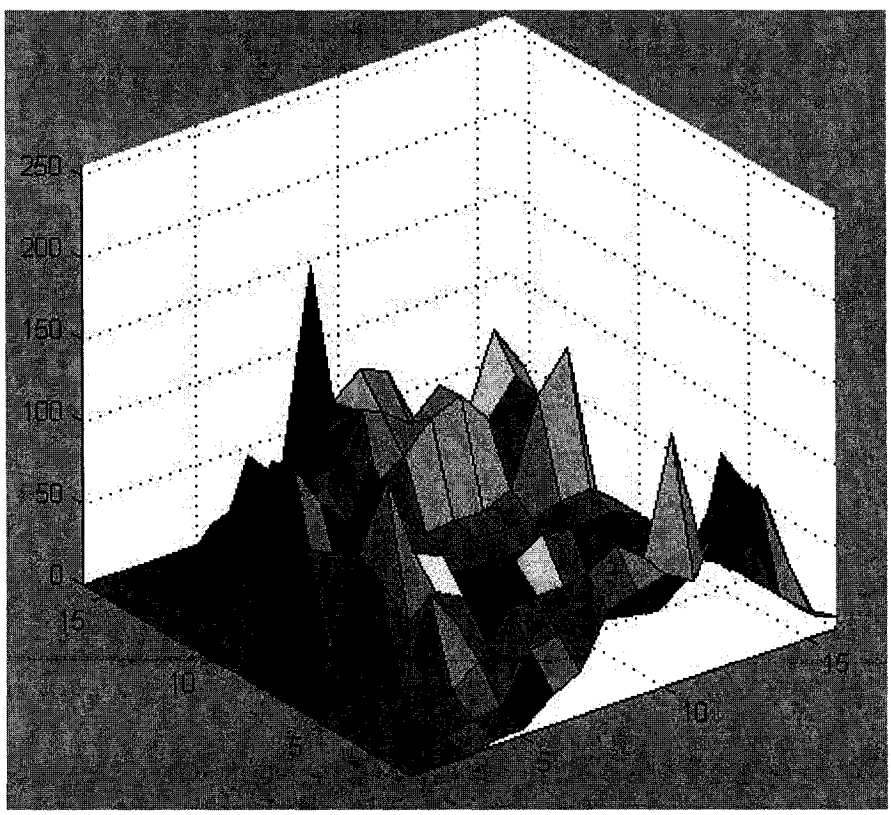


Figure 4.3. *An example of an acquired raw image.*

4.2.3. Example: Importing a Raw Image

The user can also import a previously saved workspace variable file (MAT-file) into the *Raw Image Axes*.

4.2.3.1. The Import Sub-Menu

To import a Raw Data MAT-file, the user points to the *Menu Bar* and selects *Workspace -> Raw Data -> Import ...* (figure 4.4).

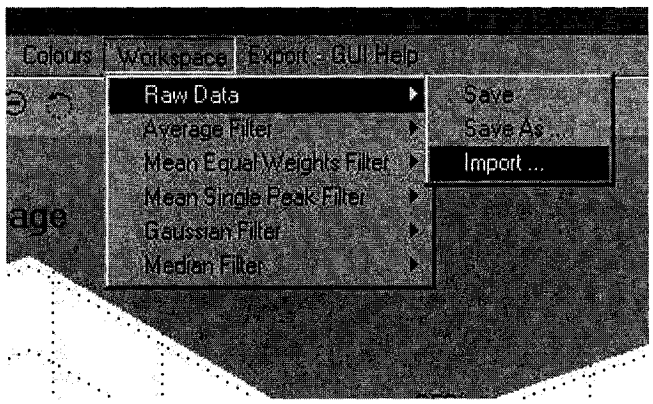


Figure 4.4. *The "**Workspace -> Raw Data -> Import ...**" menu item.*

The *Select File* dialog box appears (figure 4.5).

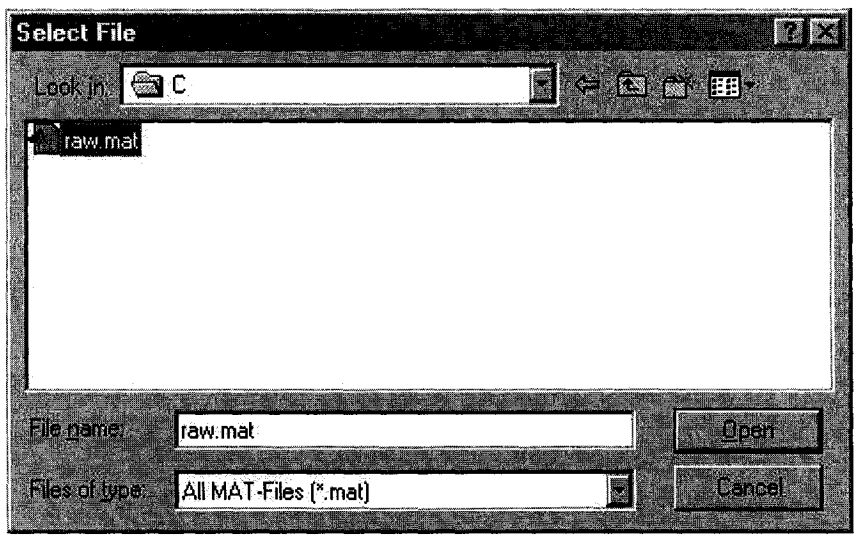


Figure 4.5. *The **Select File** dialog box.*

The user selects the desired MAT-file and click *Open*.

The above selected *raw.mat* file (geometric symbol "C" from the supplied *Library*) plotted in the *Raw Image Axes* is shown in figure 4.6.

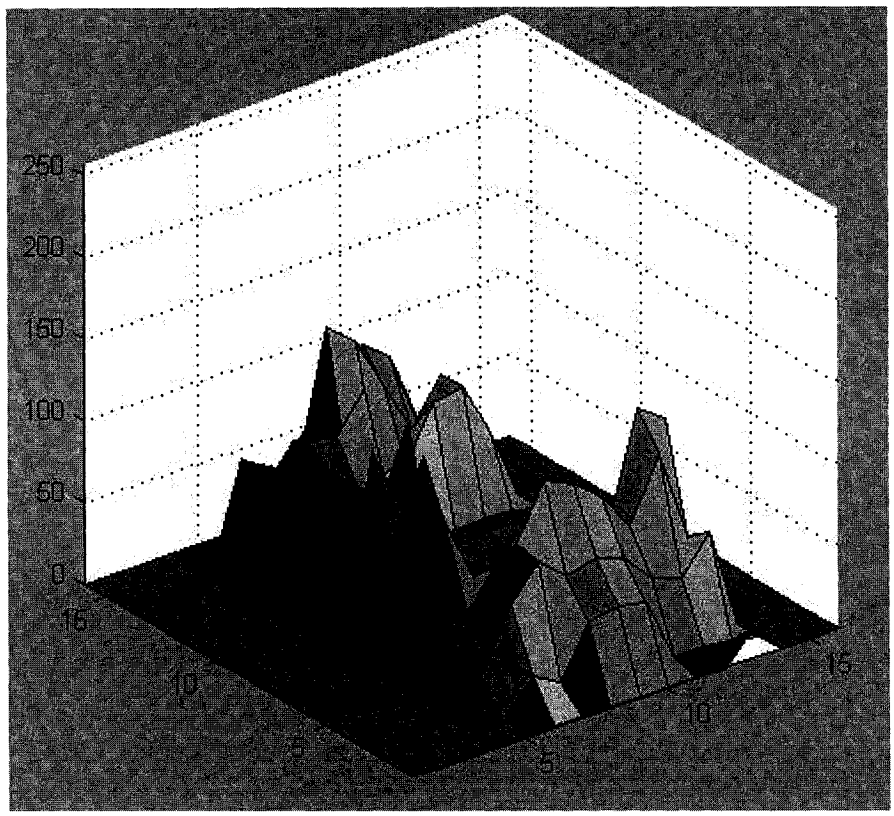


Figure 4.6. *The raw.mat file (geometric symbol "C" from the supplied* ***Library****) plotted in the* ***Raw Image Axes.***

NOTE: The user can import either a file from the supplied *Library* or a file that have been saved with TS GUI.

4.2.3.2. The Supplied Libraries

The TS GUI folder contains four *Library* subfolders. Each of them is a collection of MATLAB MAT-files saved with TS GUI. The MAT-files were obtained by acquiring raw data, through the Tactile Sensor Electronic Interface, from a tactile sensor impressed by different shapes.

The *TS GUI Libraries* are:

- Characters: library of four geometric symbols (h, C, L Y) representing the terms of a pseudo-random array;
- Letters: library of letters (A, B, C, ..., Z);
- Numbers: library of numbers (0, 1, 2, 3, 4, 5, 6, 7, 8, 9);
- Shapes: library of different shapes.

4.3. Tactile Image Processing - Principle and Examples

4.3.1. Image Noise

The acquired image displayed in the *Raw Image Axes* contains some amount of noise caused by:

- The Force Sensing Resistor Tactile Sensor, since it is not suited for precision measurements [19];
- The operational amplifiers and other electronic components of the Tactile Sensor Electronic Interface;
- The analog-to-digital conversion process;
- Transmission errors.

Noise refers to any entity, data or intermediate results that are irrelevant to a certain image [31]. The real image is composed of a spurious, random signal, $n(i,j)$, which needs to be removed or at least attenuated, added to the ideal taxel values $I(i,j)$ [31, 32]:

$$\bar{I}(i,j) = I(i,j) + n(i,j)$$

There are three types of noise:

- Gaussian noise, when for each location (i,j), the noise $n(i,j)$ is a random variable distributed according to a zero mean Gaussian distribution function of fixed standard deviations;

- Impulsive noise, which is added to the noise introduced by acquisition and makes the value of randomly distributed taxels very different from their real values and from those of neighbouring taxels [31];
- Salt and pepper noise, which contains random occurrences of both maximum and minimum intensity values [33].

The white Gaussian noise is only an approximation of the additive real noise [31], thus for the purpose of this thesis I will only consider some smoothing operations for attenuating the impulsive noise. Linear and nonlinear noise filtering are presented in the following sections.

4.3.2. Linear Noise Filtering

Linear filtering is a widely used technique for noise smoothing. The filtered image is a convolution of the image with a constant matrix, called mask or kernel [31]:

$$I_A(i,j) = I * A = \sum_{h=-\frac{m}{2}}^{\frac{m}{2}} \sum_{k=-\frac{m}{2}}^{\frac{m}{2}} A(h,k) I(i-h, j-k)$$

where:

- I is an N x M image;
- A is the kernel of a linear filter, that is an m x m mask;
- I_A is the filtered version of I at each pixel (i,j);
- m is an odd number smaller than both N and M;
- * indicates discrete convolution, and $m/2$ integer division.

In a linear filtered image, the value of each taxel $I(i,j)$ is replaced with a weighted sum of I values in the neighbourhood of (i,j). The weights are the entries of the mask. The output of this procedure is shift independent [34], that is, the filtered image depends on the pattern of the neighbourhood, rather than the position of the neighbourhood.

4.3.2.1. Average (Mean) Filtering

One of the simplest linear filters is the average filter, in which the value of each taxel is replaced by the average of all values in its local neighbourhood [33]:

$$h(i,j) = \frac{1}{M} \sum_{(k,l) \in N} f(k,l)$$

where M is the total number of taxels in the neighbourhood N. An average filter can be implemented as a convolution operation with equal weights entries in the convolution mask.

The filtering results depend on the size of the neighbourhood N (the size of the convolution mask). The larger the neighbourhood, the greater the degree of filtering is achieved.

There are some disadvantages of average filtering [31, 33]:

- High-frequency components shared with noise are removed. As a consequence, the average filtered images are blurred and the sharp details are lost.
- Average filtering dilates some details in the image.
- Impulsive noise is not completely eliminated but only attenuated and diffused.

In TS GUI the *Average* filter is implemented with the MATLAB *fspecial* and *filter2* functions [30]. As different weighting schemes can be used, I also implemented a *Mean Equal Weights* filter, which uses a fixed 3-by-3 constant-weight mask, (1/9)[1 1 1; 1 1 1; 1 1 1], and a *Mean Single Peak* filter, which puts more weight on the initial intensity value of the taxel. It also uses a fixed 3-by-3 mask: (1/16)[1 2 1; 2 4 2; 1 2 1].

4.3.2.2. Gaussian Filtering

As an alternate approach to using equal weights for the mask, it is possible to reduce the weight of the input taxels with increasing distance from the centre taxel. In other words the taxels in the centre of the mask weight more strongly than those at its boundaries. This is the

case for the Gaussian filter, a particular case of averaging, for which the weights of the mask are chosen according to the shape of a standard 2-D Gaussian function. [31, 32, 33, 34]:

$$g(i,j) = e^{\frac{(i^2+j^2)}{2\sigma^2}}$$

where σ is the Gaussian spread parameter, or the *standard deviation* of the Gaussian (also known as its *"sigma"*);

The properties of Gaussian filters make them particularly useful for image processing [32, 33, 34]:

- Gaussian functions are rotationally symmetric, that is, the amount of filtering is the same in all directions.
- The Fourier transform of Gaussian filters have a single lobe in frequency spectrum. This means that the unwanted high frequency signals are removed, while most of the desirable signals are retained.
- The Gaussian function has a single lobe, thus a Gaussian filter smooths by replacing each pixel with a weighted average of the neighbouring pixels. The weight given to a neighbour decreases proportionally with distance from the central pixel. This is important because an edge is a local feature in an image, and a smoothing operation that gives less significance to pixels farther away will not distort the features.
- The degree of smoothing of a Gaussian filter depends on the value of σ. It can take a value between 0.1 and 1.0. A larger *Deviation* results in a greater smoothing but also in a greater blurring of the image.

The *Gaussian* filter is implemented with the MATLAB *fspecial* and *filter2* functions [30].

4.3.3. Nonlinear Noise Filtering

Some of the disadvantages of linear filters are [31, 32, 33, 35]:

- Tend to blur edges and other sharp discontinuities in intensity values in an image;
- Cannot correctly localize image features;
- Do not completely suppress peak noise;
- Present secondary lobes in frequency domain, which let noise into the filtered image.

As the Fourier transform of a Gaussian has no secondary lobes, Gaussian filters solve the secondary lobe problem. The rest of them are more or less solved by nonlinear filters, that is, filters that cannot be modeled by convolution.

The most used nonlinear filter is the median filter, which replaces each taxel value of the image with the median of the values found in a predefined neighbourhood of that taxel [31, 32, 33, 34, 35]. In a sorted set of n values, the median M is found such that $n/2$ values are less than M and $n/2$ values are greater than M. Thus, the algorithm for applying a median filter to a taxel simply consists in sorting the values of the taxel and its neighbours, determining the median M, and replacing the value of the taxel with the median M.

The advantages of median filtering are:

- As the filtered intensity values do not depend on values which are significantly different from typical values in the neighbourhood, the details of the image are better preserved than in the case of linear filters, while the uniform regions are very well smoothed;
- Creates less blur in the image by better preserving the structure of boundaries between noisy regions;
- Efficiently removes salt and pepper and impulsive noise.

In TS GUI the non-linear *Median* filter is implemented with the MATLAB *medfilt2* function [30]. Like with averaging, the larger the neighbourhood, the smoother the result.

4.3.4. Example: Filtering the Raw Image

4.3.4.1. Selecting a Filter

The *Select Filter Area* contains the pop-up menus the user needs for choosing a filter and its parameters.

To select a filter, the user clicks the arrow to open the *Select Filter* pop-up menu (figure 4.7) and selects one of the following filtering functions: *Average, Mean Equal Weights, Mean Single Peak, Gaussian,* and *Median.*

Figure 4.7. *The* ***Select Filter*** *pop-up menu.*

Four linear filters (the *Average,* the *Mean Equal Weights,* the *Mean Single Peak,* and the *Gaussian* filters) and one non-linear filter (the *Median* filter) can be used for smoothing the noise that accompanies the *Raw Image.*

4.3.4.2. Selecting a Neighbourhood

For each filter operation, the degree of filtering depends on the chosen *Neighborhood* (mask). To chose a *Neighborhood* for the selected filter, the user clicks the arrow to open the *Neighborhood* pop-up menu (figure 4.8) and select one of the following values: *[1 1], [2 2], [3 3], [4 4], [5 5], [6 6], [7 7], [8 8],* and *[9 9].*

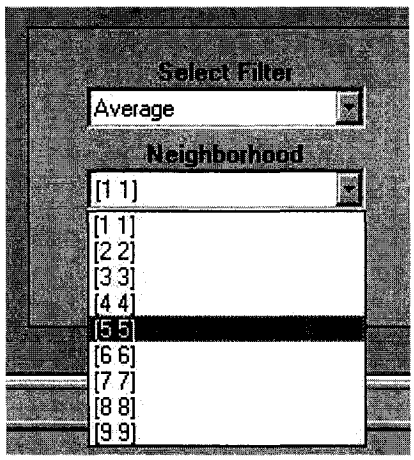


Figure 4.8. *The* ***Neighborhood*** *pop-up menu.*

4.3.4.3. Selecting a Deviation (Gaussian Filter Only)

To select the standard *Deviation (Sigma)* of the Gaussian function, the user clicks the arrow to open the *Deviation (Sigma)* pop-up menu (figure 4.9) and select one of the following values: *0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9,* and *1.0.*

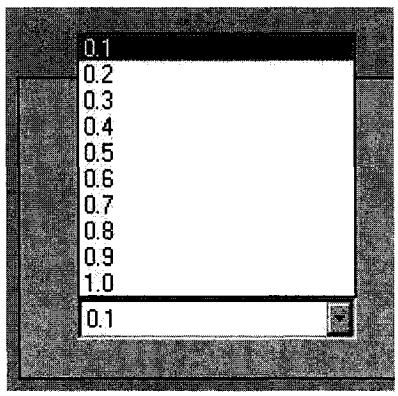


Figure 4.9. *The* ***Deviation (Sigma)*** *pop-up menu.*

4.3.4.4. Applying a Filter

Now the user can click the *Apply Filter* push button to apply the filter and its parameters to the raw image. An example of a filtered image is shown in figure 4.10.

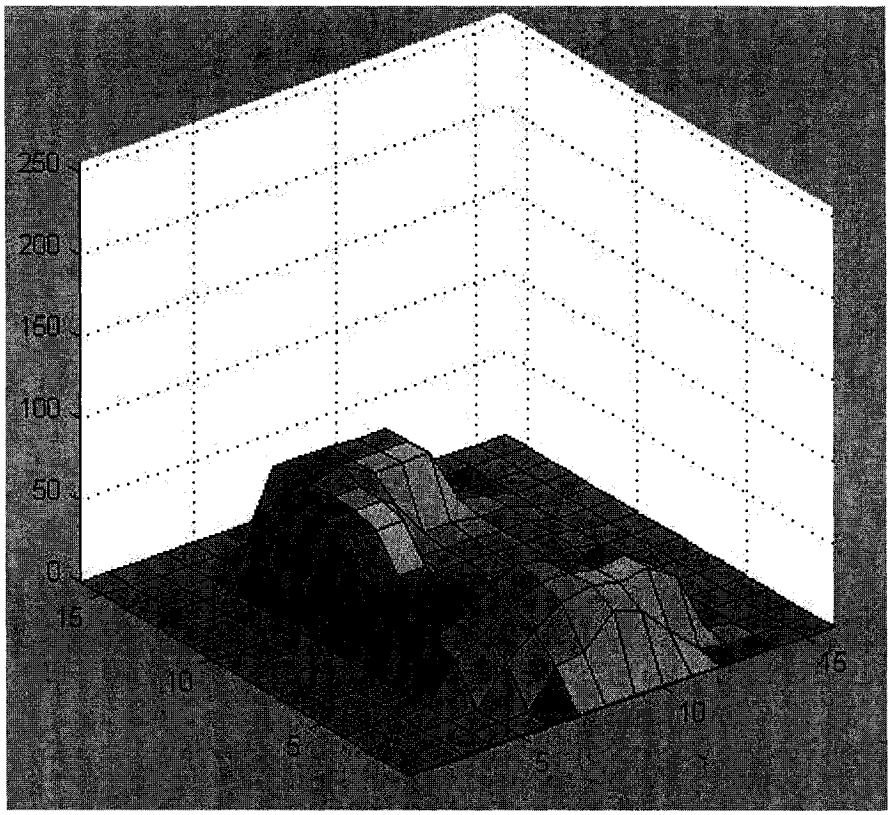


Figure 4.10. *An example of a filtered image*

4.4. Tactile Image Display

As stated earlier, the raw and filtered images are displayed in the middle area of the TS GUI window, i.e. in the *Raw Image Axes* and the *Filtered Image Axes*, respectively. The filtered image can be altered by choosing the plotting method, the colormap and the shading method. The user can also use the *Tool Bar* to change the view of both the raw and filtered images by either zooming in for a closer look at a portion of an axes or rotating the 3-D axes.

4.4.1.1. Selecting a Plotting Method

Two different methods of plotting are available: *surf* and *surfl*. The user clicks the arrow to open the *Plotting Method* pop-up menu (figure 4.11) and selects the one that best suites his or her colouring needs.

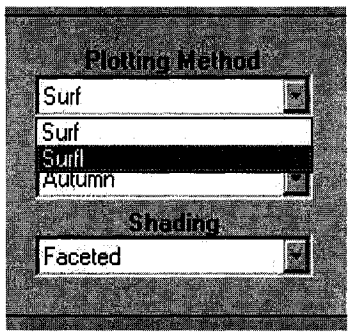


Figure 4.11. *The* ***Plotting Method*** *pop-up menu.*

Surf displays a 3-D shaded surface graph.

Surfl displays a 3-D shaded surface based on a combination of ambient, diffuse, and specular lighting models.

4.4.1.2. Selecting a Colormap

The user can change the colouring of the plots by selecting a colormap. The user clicks the arrow to open the *Colormap* pop-up menu (figure 4.12) and selects one of the following colormaps that are available in MATLAB, as well as in TS GUI Demo: *autumn, bone, colorcube, cool, copper, flag, gray, hot, hsv, jet, lines, pink, prism, spring, summer, white,* and *winter*.

The default *colormap* for the acquired and imported images is *gray*. If the user changes the colormap of the filtered image, the colour of the raw image changes accordingly.

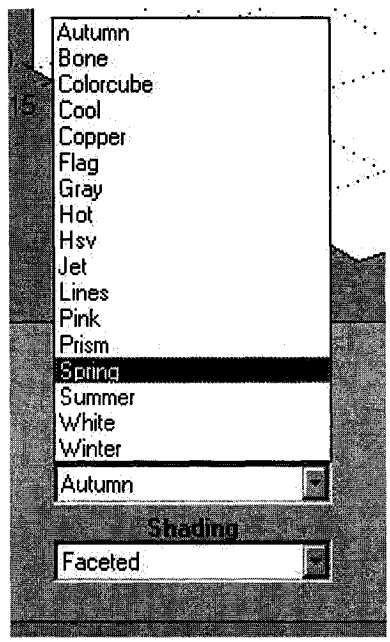


Figure 4.12. *The* ***Colormap*** *pop-up menu.*

4.4.1.3. Selecting a Shading

Three colour *shading* of the plots are available. The user clicks the arrow to open the *Shading* pop-up menu (figure 4.13) and selects one of the following shading functions: *faceted, flat,* and *interp*(olated).

Upon applying *flat* shading, each mesh line segment and face has a constant colour.

Faceted shading is flat shading with superimposed black mesh lines.

Interp(olated) shading varies the colour in each line segment and face by interpolating the colormap index or true colour value across the line or face.

The default shading mode is *faceted.*

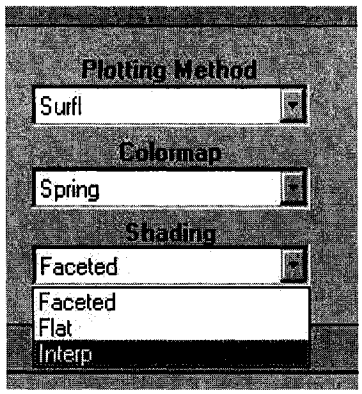


Figure 4.13. *The* ***Shading*** *pop-up menu.*

4.4.1.4. Applying a Filter

The user clicks the *Apply Filter* push button to apply a filter with the above selected properties. Then, the user can use the *Tool Bar* to zoom in and out and to rotate the image.

An example of a TS GUI window that contains a raw image and the corresponding filtered and rotated image is shown in figure 4.14.

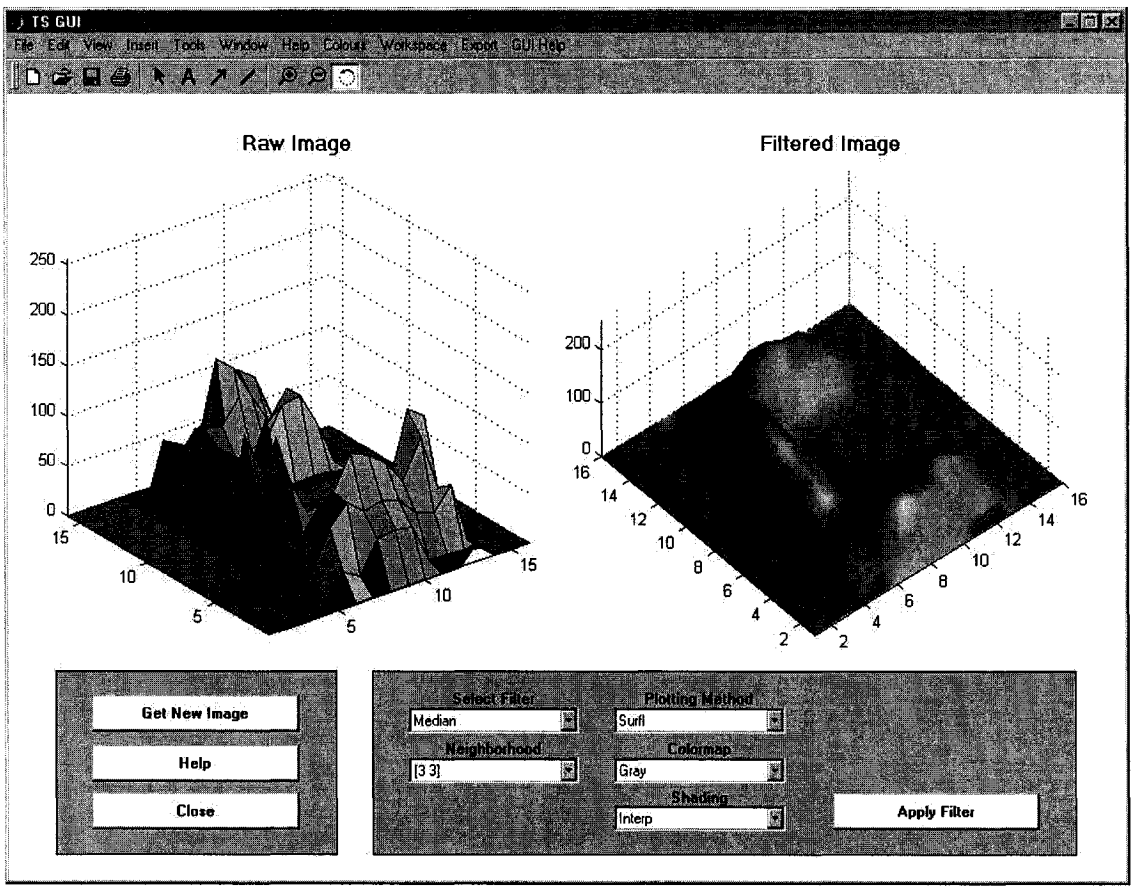


Figure 4.14. *An example of a filtered and rotated image.*

4.5. Miscellaneous Functions

4.5.1. Changing the Colours

The user can change the colours of virtually all the controls and components of TS GUI. To change the colours, the user clicks on the *Colours* pull-down menu (figure 4.15), then chooses the item of which colour he or she wants to change. Finally, the user selects the desired colour.

The following colours can be used for all of the graphical user interface components: *blue, cyan, gray, green, light gray, magenta, pink, red, teal, white,* and *yellow.* Black can only be used for the *"Raw Image" Text* and the *"Filtered Image" Text.*

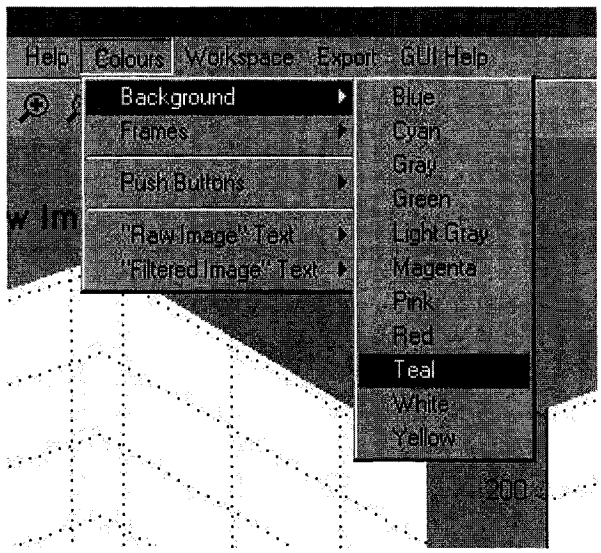


Figure 4.15. *The "Colours" pull-down menu.*

4.5.2. Saving the Workspace Variables

After plotting to the *Raw Axes* or *Filtered Axes,* the user can save his or her work in a format that can be opened during another TS GUI Demo session, or within the current session.

The MATLAB *workspace* consists of a set of *variables* (named arrays) built up during a MATLAB session and stored in memory. Thus, when quitting MATLAB, the workspace is cleared. During a MATLAB session, the user can save any or all of the variables in the current workspace to a MATLAB specific binary file – a MAT-file (a file with a .mat extension). The user can then reuse the *workspace variables* by loading the MAT-file at a later time.

In TS GUI Demo the user can only save the variables one at a time, not all together. After acquiring an image, the user can save the raw data workspace variable, whereas after applying a filter, he or she can save the correspondent filter workspace variable. A workspace

variable of a filter that has not been applied yet cannot be saved. Since the workspace variables are maintained in memory, the user can save them after applying all the filters. If a new image is acquired though, the variables are cleared.

To save a *Workspace Variable*, the user points to the *Menu Bar*, opens the *Workspace* menu, selects *Raw Data* or the desired *Filter*, and clicks *Save* or *Save As ...* (figure 4.16).

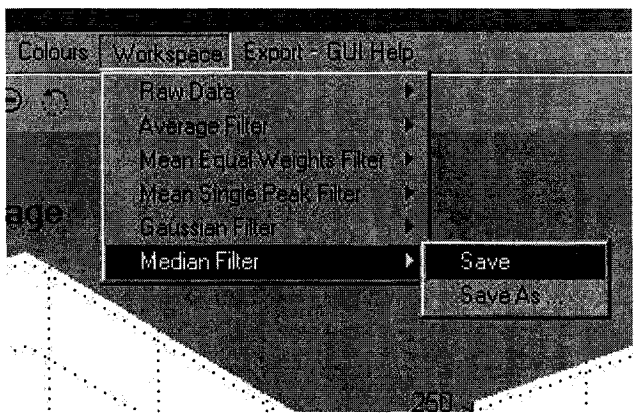


Figure 4.16. *The* ***Workspace*** *pull-down menu.*

4.5.3. Exporting Images

After plotting to the *Raw Axes* or *Filtered Axes*, the user can save his or her work in a format that can be used by other applications. TS GUI Demo allows saving (*Exporting*) images in *JPEG* and *Bitmap* formats.

To *Export* an image, the user points to the *Menu Bar* and clicks the *Export* menu. One of the following images can then be saved: the *Raw Image*, the *Filtered Image*, both the *Raw and Filter Images*, and the *Entire Figure* (figure 4.17).

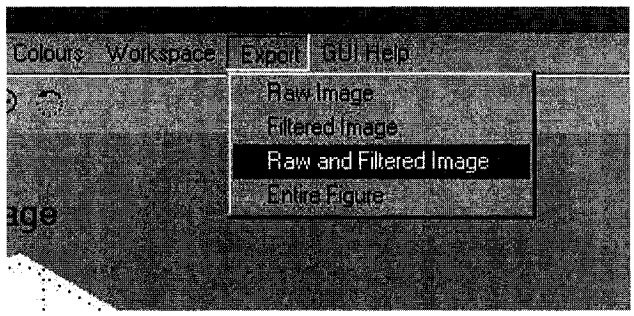


Figure 4.17. *The* ***Export*** *pull-down menu.*

The following dialog box appears (figure 4.18):

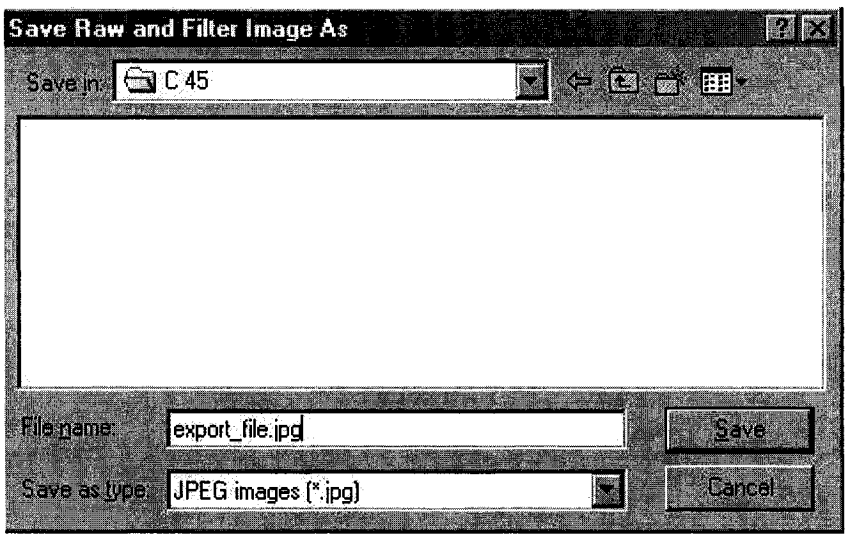


Figure 4.18. *The* ***Save Raw and Filter Image As*** *dialog box.*

The user selects *JPEG images* or *Bitmap files* in the *Save as type* menu and types the name of the file in the *File name* box.

The results of the above export situation are shown in figure 4.19.

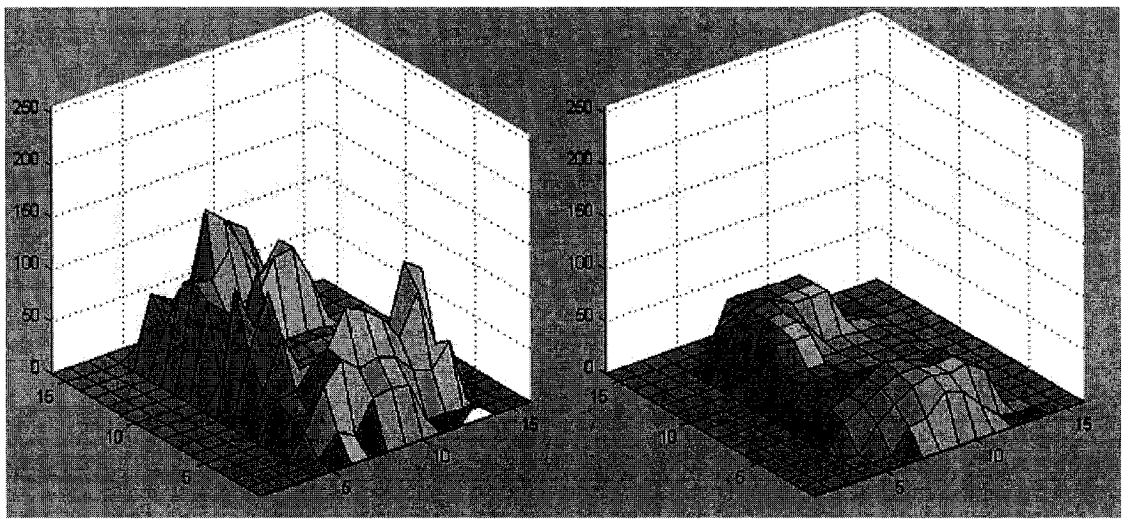


Figure 4.19. An example of an exported image.

4.5.4. Warning Messages

There are four types of *Warning Messages* that can occur while using TS GUI if the user attempts to:

- Save a workspace variable (*Workspace* menu) before acquiring (*Get New Image* push button) or importing (*Workspace -> Raw Data -> Import ...* menu item) an image;
- Apply a filter (*Apply Filter* push button) before acquiring (*Get New Image* push button) or importing (*Workspace -> Raw Data -> Import ...* menu item) an image;
- Save a filter workspace variable (e.g. *Workspace -> Average Filter -> Save*) before applying the filter (*Apply Filter* push button);
- Import (*Workspace -> Raw Data -> Import ...* menu item) a MAT-file that was not previously saved by TS GUI or TS GUI Demo.

Every message contains a short description of the steps to follow in order to avoid the Warning Message.

For instance, if the user attempts to save a workspace variable before acquiring or importing an image, the message in figure 4.20 appears.

Figure 4.20. *An example of a warning message.*

To avoid this message, the user should acquire (*Get New Image* push button) or import (*Workspace -> Raw Data -> Import ...* menu item) an image before saving (*Workspace* menu).

4.5.5. Getting Help

The user can open the *Help* files by either clicking the *Help* push button or pointing to the *Menu Bar* and selecting *GUI Help -> Operating the GUI.*

For instructions on installing TS GUI the user can open the *Read Me* file by pointing to the *Menu Bar* and selecting *GUI Help -> Read Me* (figure 4.21).

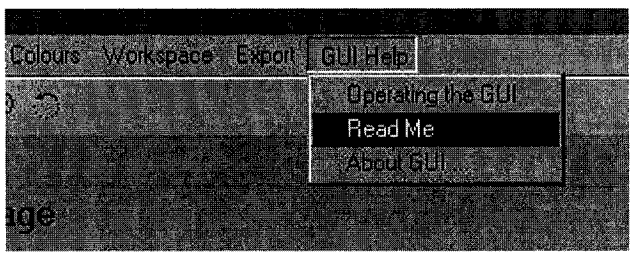


Figure 4.21. *The **GUI Help -> Read Me** menu item.*

The *Read Me* file opens in the MATLAB Help window (figure 4.22).

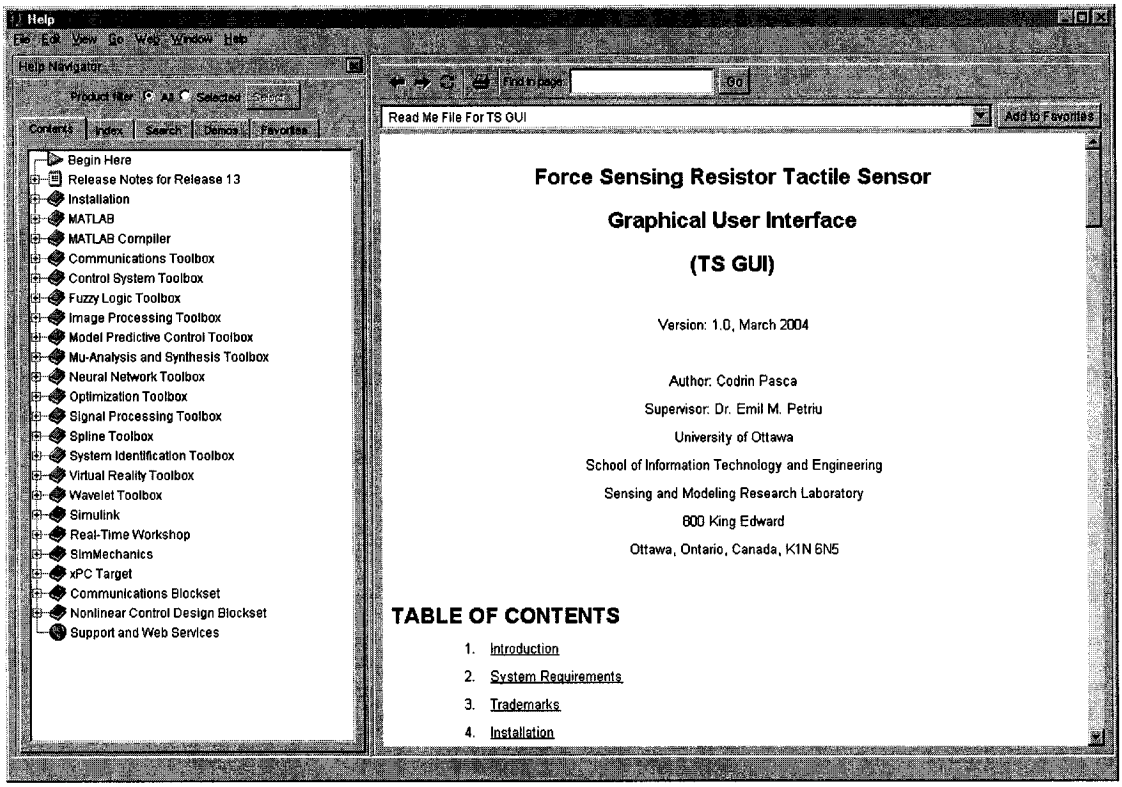


***Figure 4.22.** The **Read Me** file.*

To open the *About TS GUI* window (figure 4.23) the user points to the *Menu Bar* and selects *GUI Help -> About GUI ....*

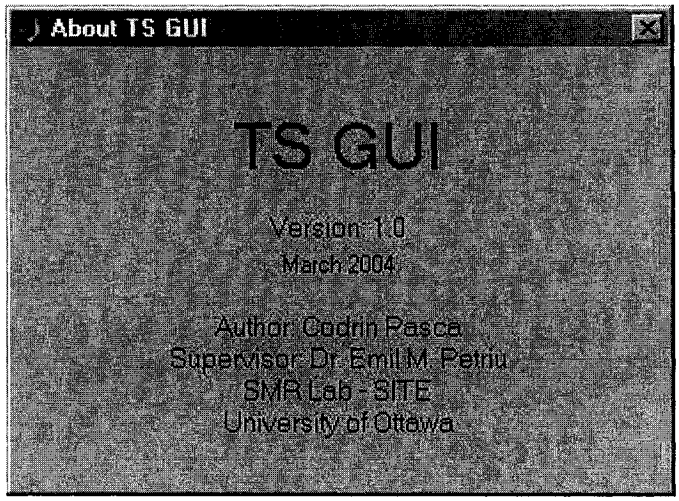


*Figure 4.23. The **About TS GUI** window.*

4.5.6. Closing the GUI

The easiest way to close TS GUI is to click the *Close* push button. The following close dialog window appears (figure 4.24):

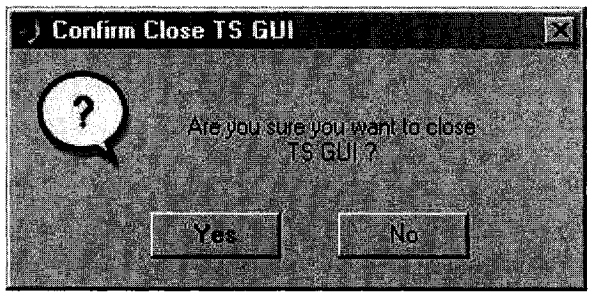


*Figure 4.24. The **Close TS GUI** dialog window.*

The user can chose *Yes* to close, or *No* to return to TS GUI Demo.

TS GUI Demo can be closed in three other ways: selecting *File -> Close*; pressing the *Ctrl+W* keyboard shortcut; or clicking the *Close Window* box in the upper right corner of the *TS GUI Demo* window.

Chapter 5

Experimental Results

This chapter contains the experiments that I performed with the tactile sensor interface and TS GUI. First, a short description of the experimental set up is presented. Then, a set of results are shown.

5.1. Experimental set up

Basically, the experimental set up should contain the Force Sensing Resistor Tactile Sensor connected to the input of the Tactile Sensor Electronic Interface, which is then connected to a Personal Computer (PC) via the serial communication port (figure 5.1).

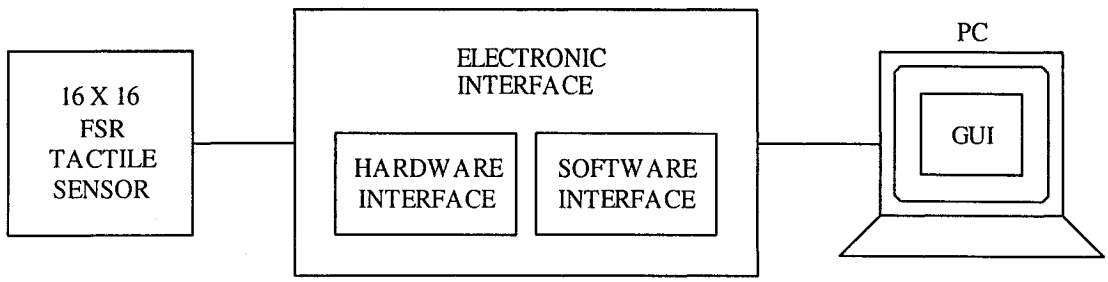


Figure 5.1. Experimental setup (principle).

The tactile sensor is scanned; the tactile sensor image is memorized and then transmitted to the serial communication port of the PC. The PC application – TS GUI – receives the raw data image. Using TS GUI, the user is then able to filter, display, and save the resulted image.

In order to perform the experiments we also needed the following:

- A set of objects, geometric symbols or characters, to be sensed;
- A mechanical test set up that would fit both the sensor and the character to be sensed and let the user press the character over the sensor.

5.1.1. Sample Characters

Figure 5.2 shows the sample geometric symbols (characters) used in the experiments and figure 5.3 is a close-up picture of one of the characters compared to the tactile sensor case. Having in mind the possibility of future research, I handcrafted a set of wooden geometric symbols representing quaternary terms of a *pseudo-random array* defined over the Galois field $GF(4) = \{0, 1, A, A^2\}$ [36]. The upper four characters in figure 5.2 are positioned orthogonally relative to the edges, whereas the lower four are the same characters rotated at 45 degrees.

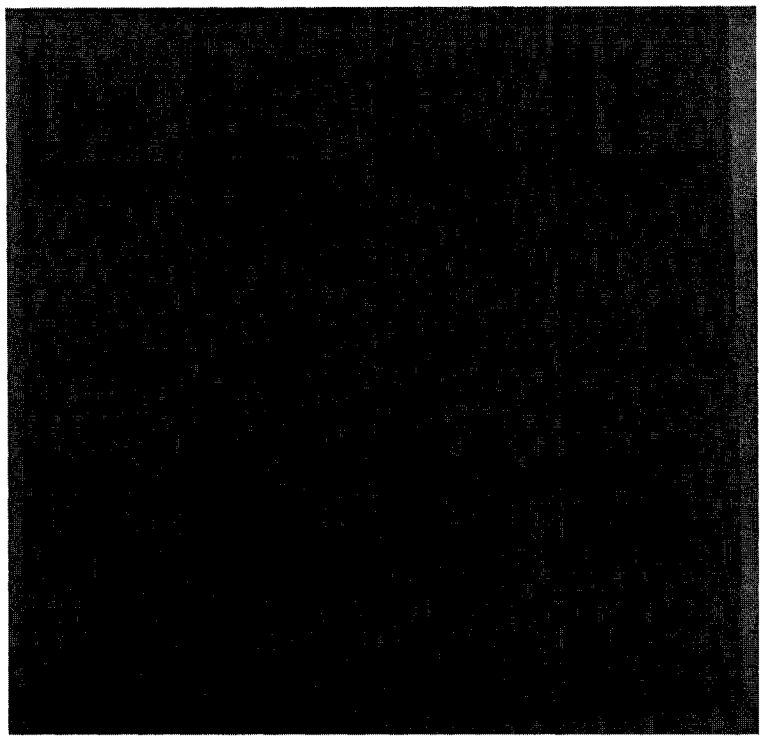


Figure 5.2. Sample characters.

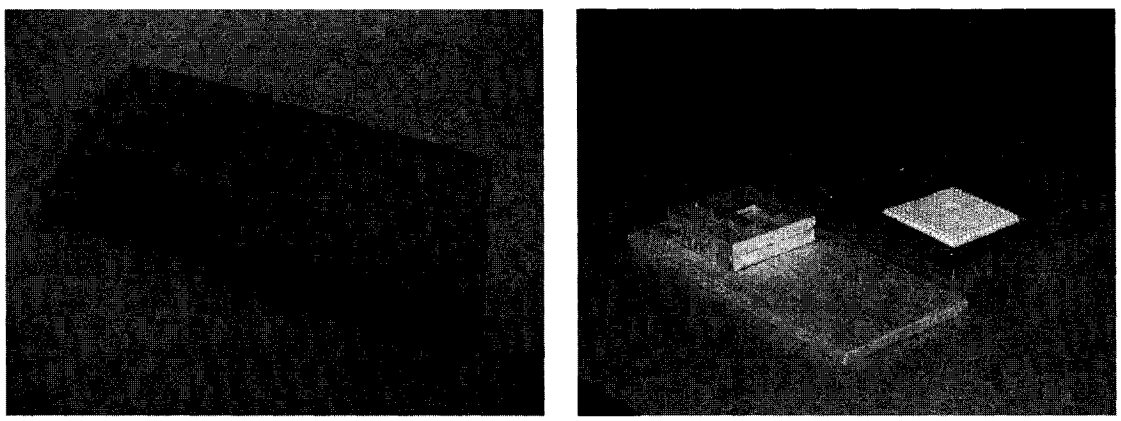


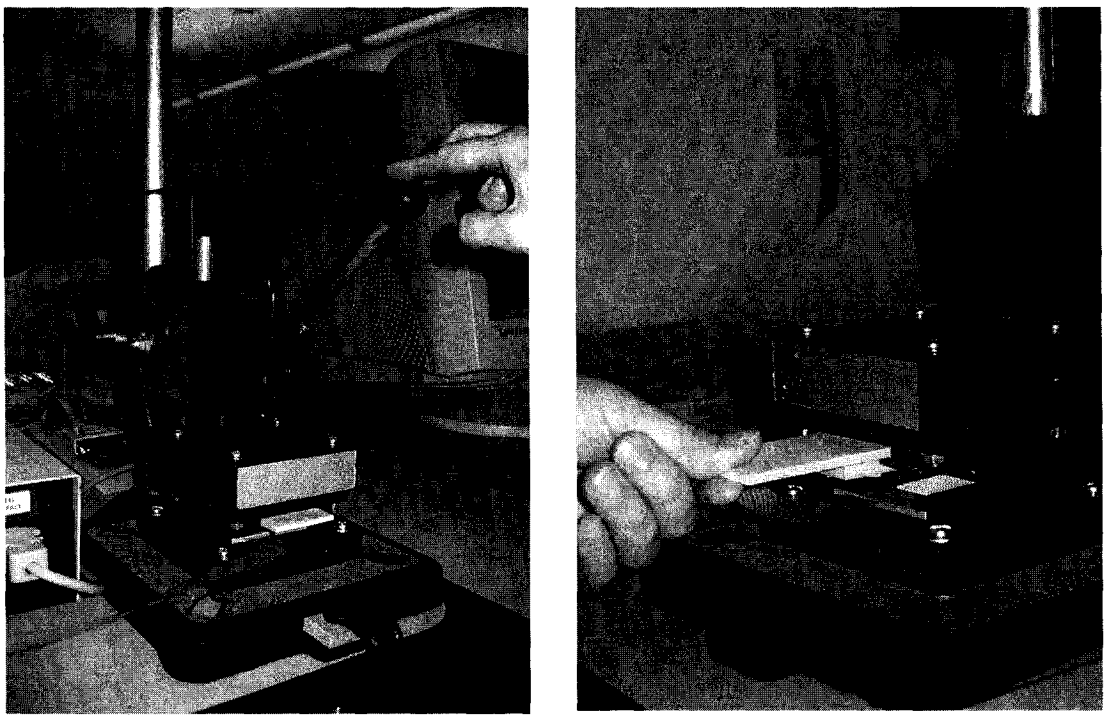
Figure 5.3. Character "h" (close-up).

5.1.2. Mechanical Test Set Up for Tactile Object Exploration

Now that we have the tactile sensor case and the characters to be sensed, we need a mechanical set up that allows reproducing measurements in a controlled test environment. We started from a commercial drill press stand and adapted it so that it can:

- Hold both the tactile sensor and the character to be sensed;
- Let the user press the character over the tactile sensor.

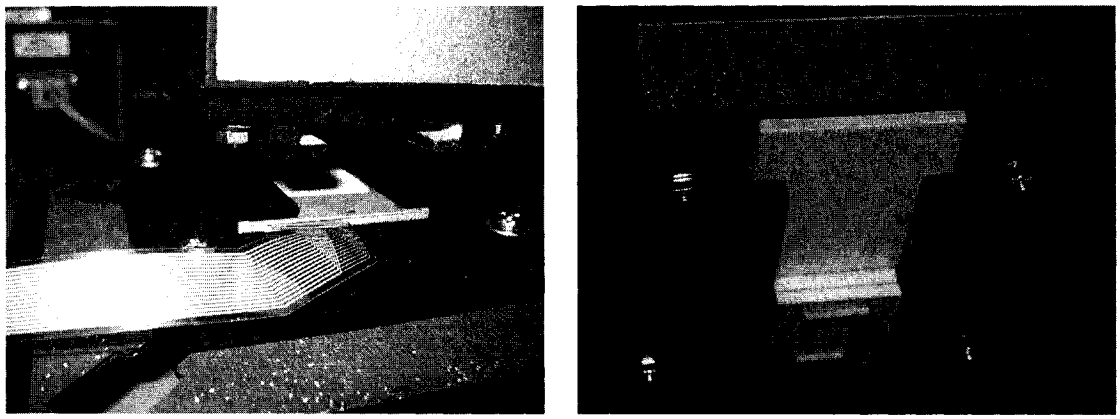
Figure 5.4.a shows the user pulling the handle so that the character presses the tactile sensor, and figure 5.4.b shows the user introducing the character into the upper slot of the mechanical test set up.



a) *b)*

Figure 5.4. *Mechanical test set up: a) pressing the character over the tactile sensor; b) introducing the character into the upper slot.*

Detailed pictures of the lower and upper slots are presented in figure 5.5.



a) *b)*

Figure 5.5. *Mechanical test set up (details): a) force sensing resistor tactile sensor in lower slot; b) character in upper slot.*

The entire experimental set up is pictured in figure 5.6.

Figure 5.6. *Experimental set up.*

5.2. Filtering

Using the experimental set up shown in figure 5.6, I gathered, in TS GUI, the raw data for each character in figure 5.2 and displayed the raw image in the *Raw Image Axes*. Then I filtered the raw image using the TS GUI filters.

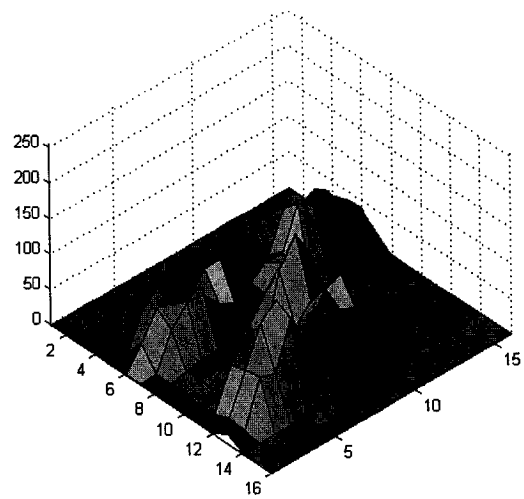
For each character I applied the following filters:

- Average filter, with a 3-by-3 square neighbourhood;
- Gaussian filter, with a 3-by-3 square neighbourhood and a value of 1.0 for the standard deviation;
- Two median filters, with a 3-by-3 square neighbourhood, both in 3-D and 2-D representations.

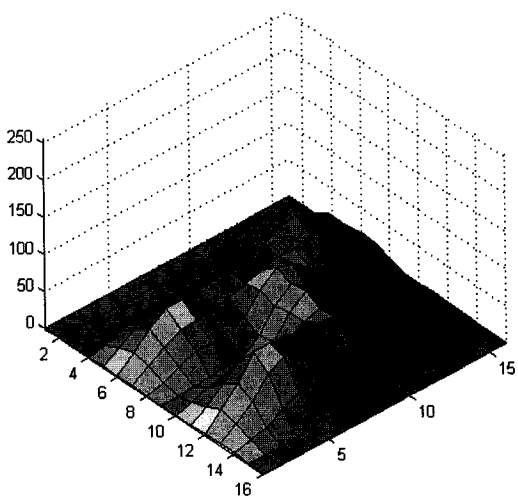
In TS GUI, for 3-D representations, the following settings are used: *plotting method* = surfl, *colormap* = grey, *shading* = faceted; whereas for 2-D representations, the settings are: *plotting method* = surf, *colormap* = grey, *shading* = interp.

Figures 5.7 to 5.14 present the above mentioned filtered images of the horizontal characters, and figures 5.15 to 5.22 contain the filtered images of the rotated characters. For each case, the raw image is also depicted at the top of the figure.

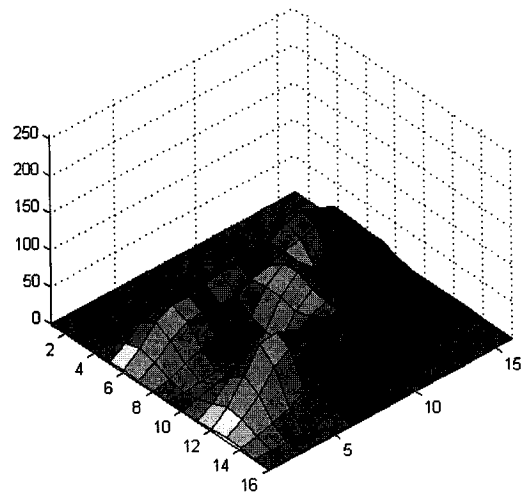
5.2.1. Orthogonal Characters



a) Raw data.

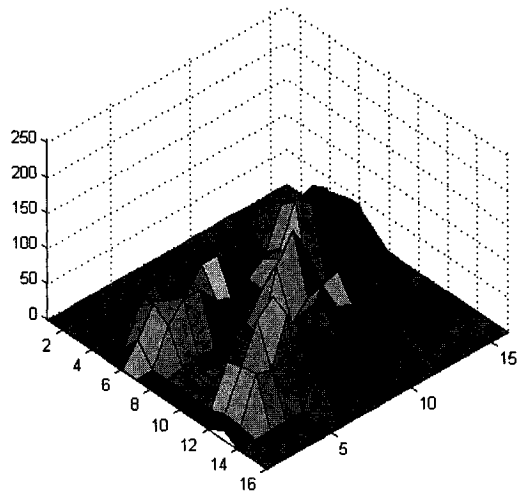


b) Average filter, n = [3 3].

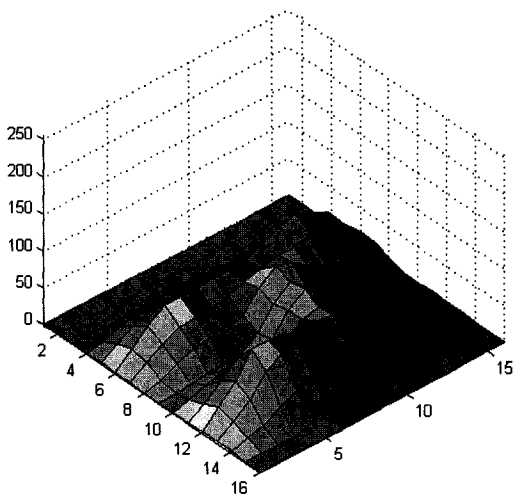


c) Gaussian filter, n = [3 3], sigma = 1.0.

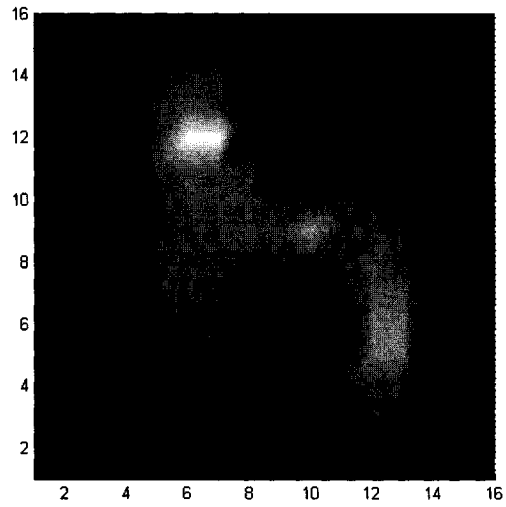
Figure 5.7 *Orthogonal character "h": linear filtering.*



a) Raw data.

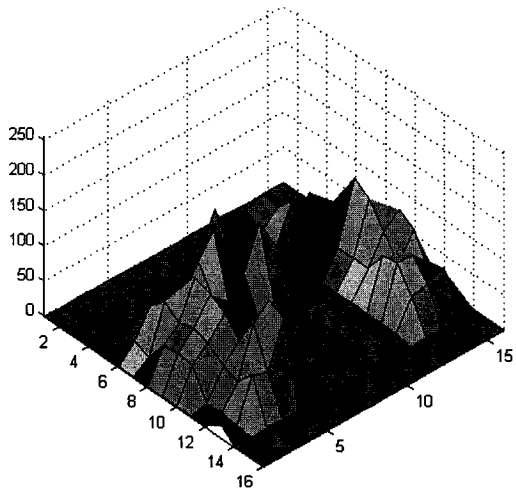


b) Median filter, n = [3 3] (3-D representation).

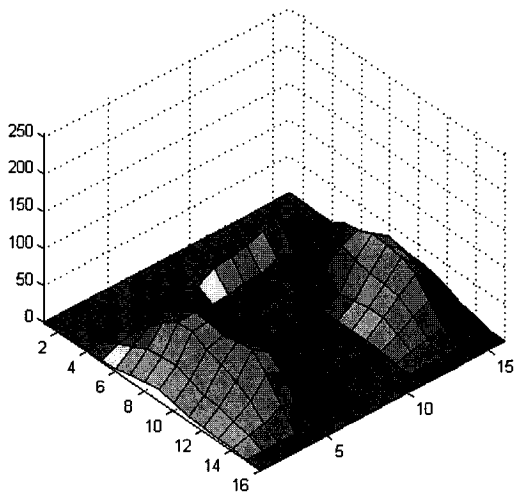


c) Median filter, n = [3 3] (2-D representation).

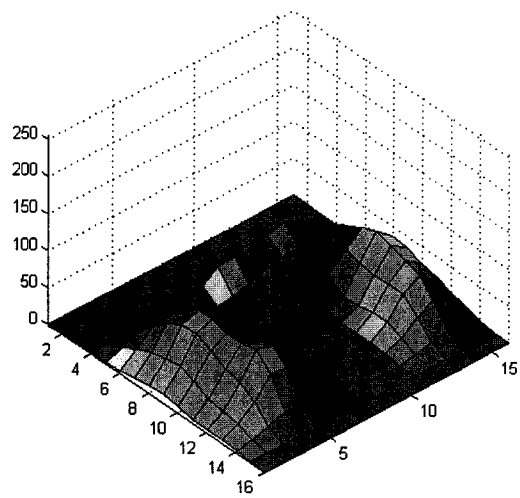
Figure 5.8. *Orthogonal character "h": non-linear filtering.*



a) Raw data.

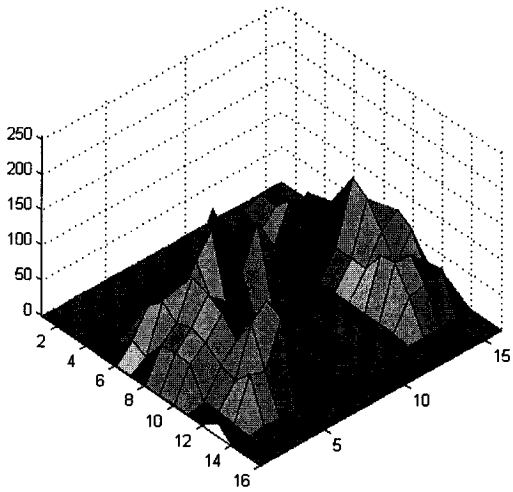


b) Average filter, n = [3 3].

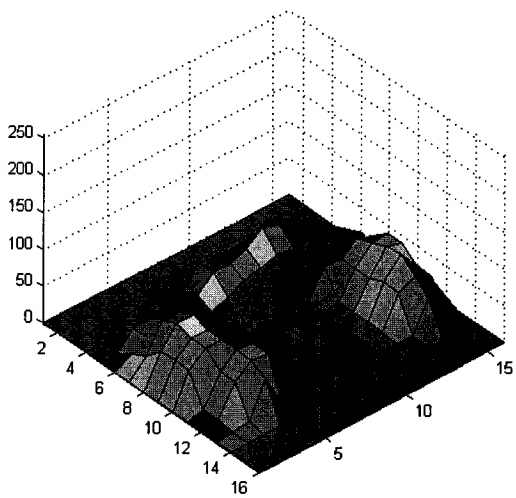


c) Gaussian filter, n = [3 3], sigma = 1.0.

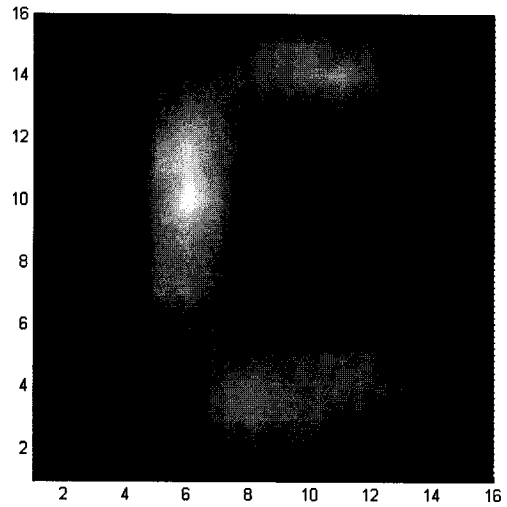
Figure 5.9 *Orthogonal character "C": linear filtering.*



a) Raw data.

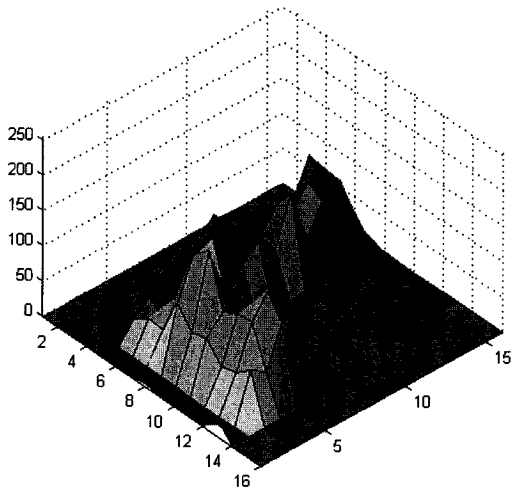


b) Median filter, n = [3 3] (3-D representation).

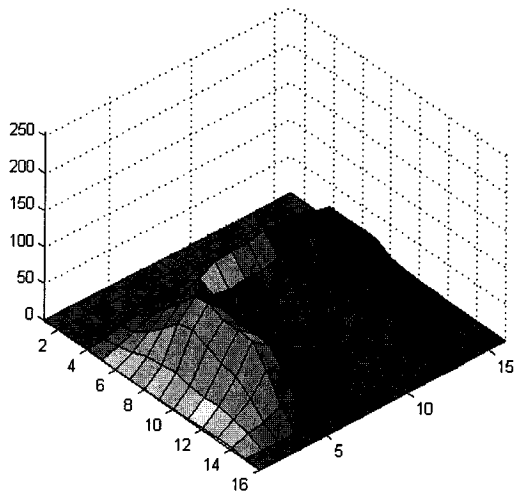


c) Median filter, n = [3 3] (2-D representation).

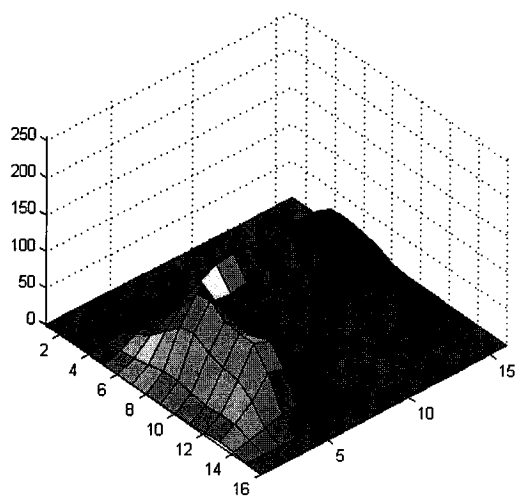
Figure 5.10. *Orthogonal character "C": non-linear filtering.*



a) Raw data.

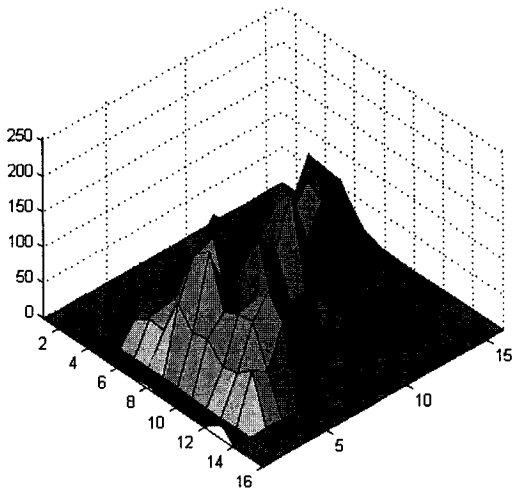


b) Average filter, n = [3 3].

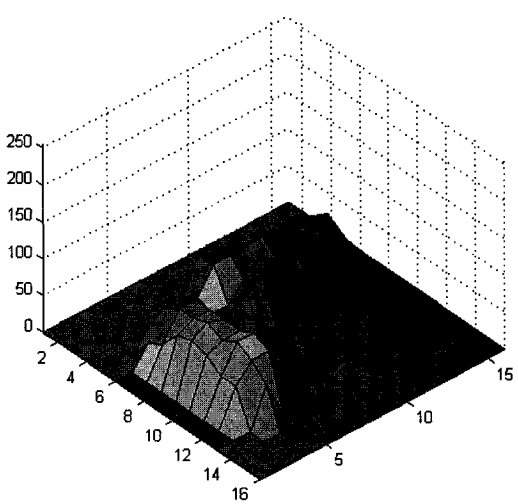


c) Gaussian filter, n = [3 3], sigma = 1.0.

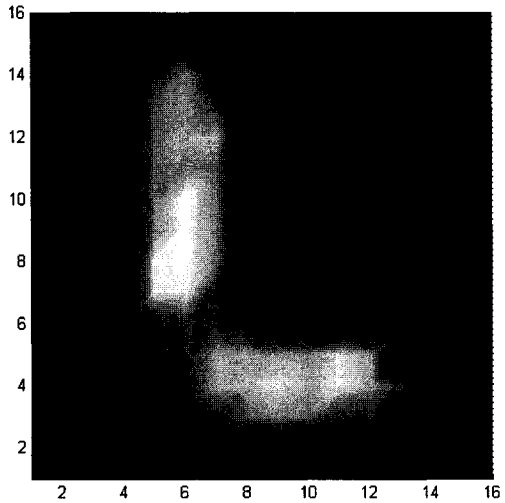
Figure 5.11 *Orthogonal character "L": linear filtering.*



a) Raw data.

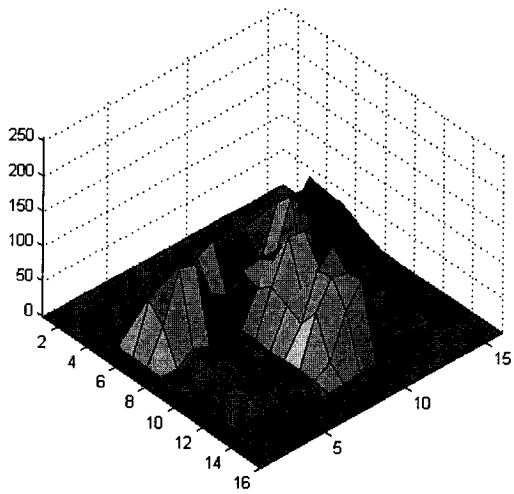


b) Median filter, n = [3 3] (3-D representation).

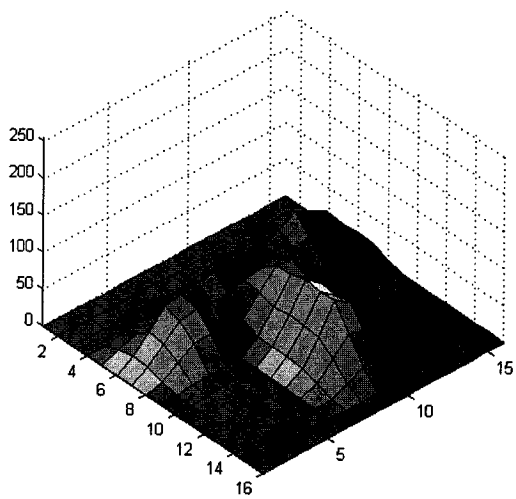


c) Median filter, n = [3 3] (2-D representation).

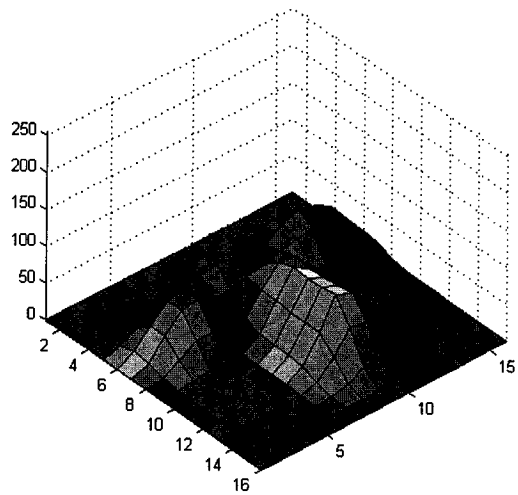
Figure 5.12. *Orthogonal character "L": non-linear filtering.*



a) Raw data.

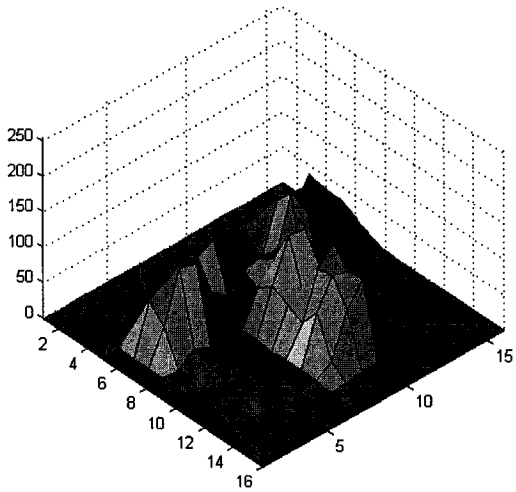


b) Average filter, n = [3 3].

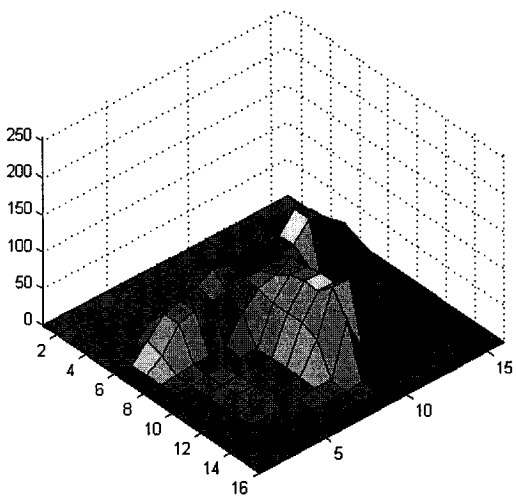


c) Gaussian filter, n = [3 3], sigma = 1.0.

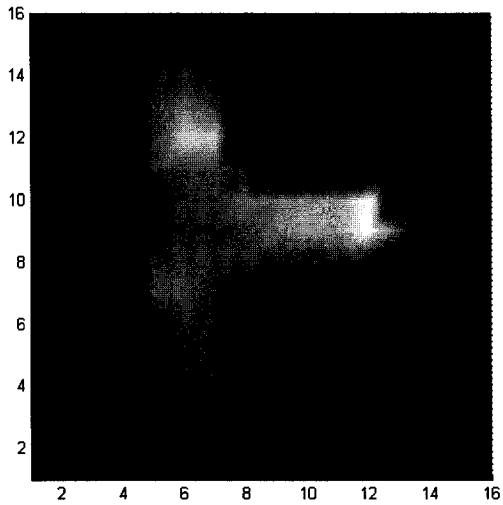
Figure 5.13 *Orthogonal character "T": linear filtering.*



a) Raw data.



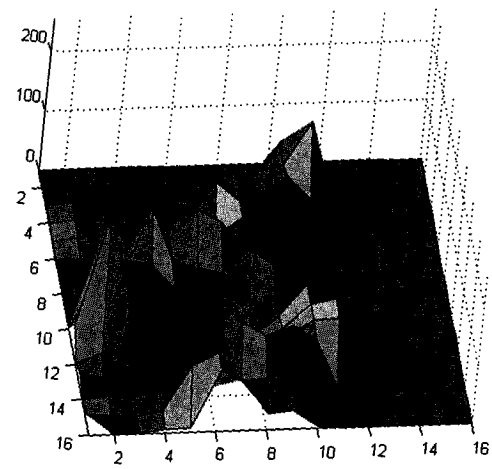
b) Median filter, n = [3 3] (3-D representation).



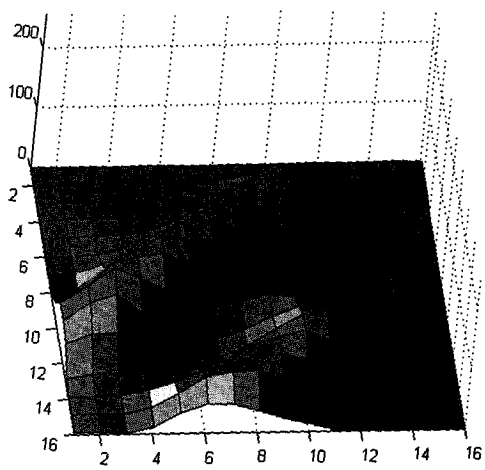
c) Median filter, n = [3 3] (2-D representation).

Figure 5.14. *Orthogonal character "T": non-linear filtering.*

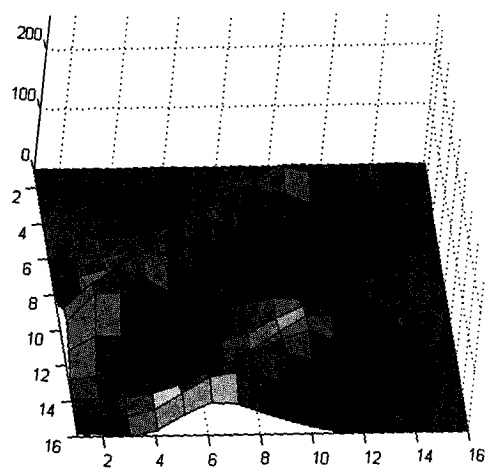
5.2.2. Rotated Characters



a) Raw data.

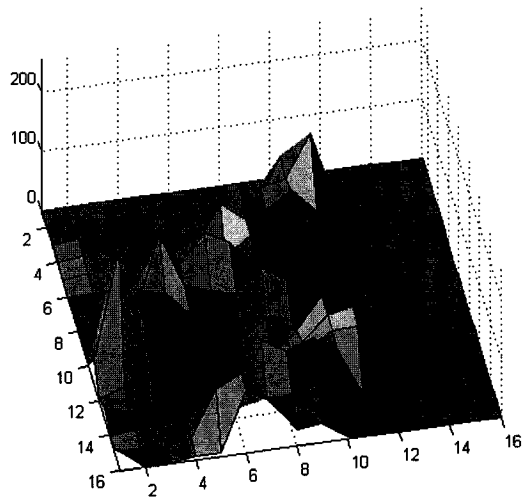


b) Average filter, n = [3 3].

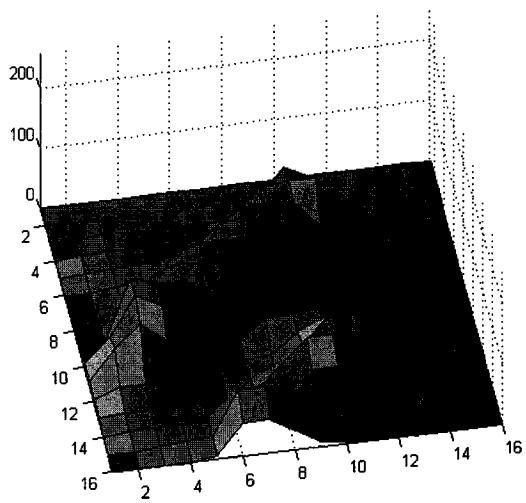


c) Gaussian filter, n = [3 3], sigma = 1.0.

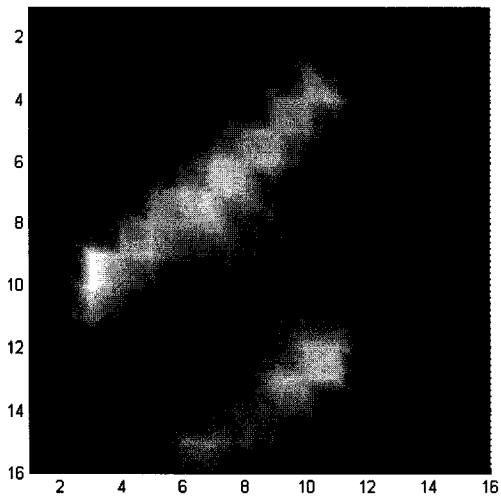
Figure 5.15 *Rotated character "h": linear filtering.*



a) Raw data.

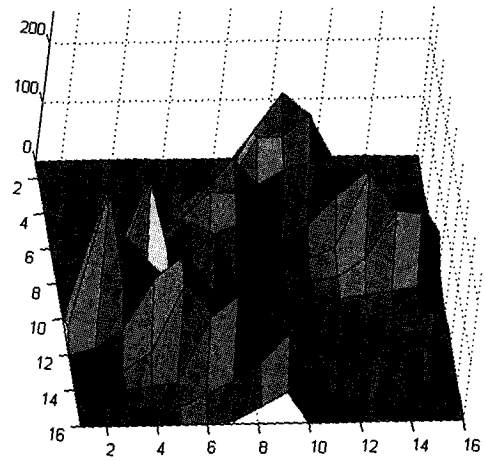


b) Median filter, n = [3 3] (3-D representation).

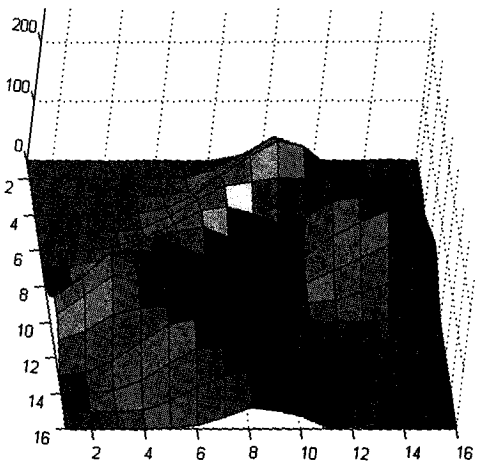


c) Median filter, n = [3 3] (2-D representation).

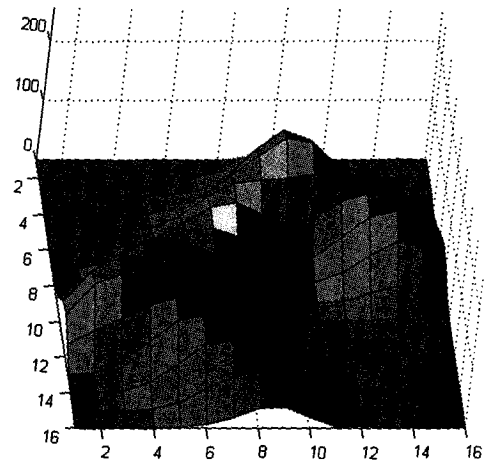
Figure 5.16. *Rotated character "h": non-linear filtering.*



a) Raw data.

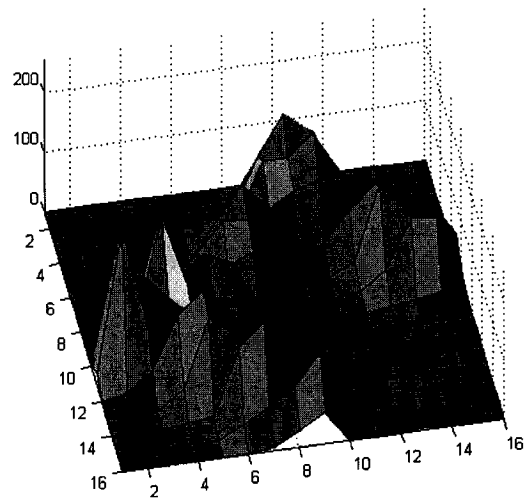


b) Average filter, n = [3 3].

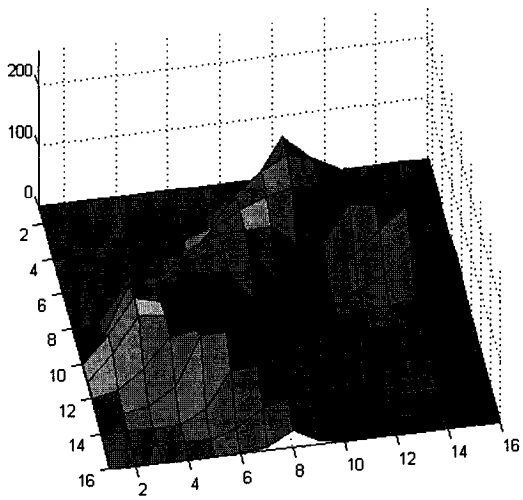


c) Gaussian filter, n = [3 3], sigma = 1.0.

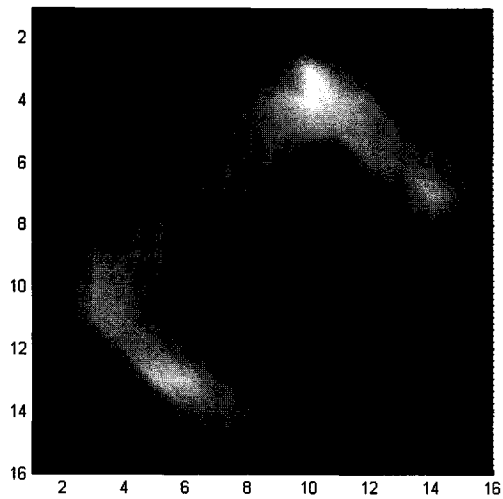
Figure 5.17 *Rotated character "C": linear filtering.*



a) Raw data.

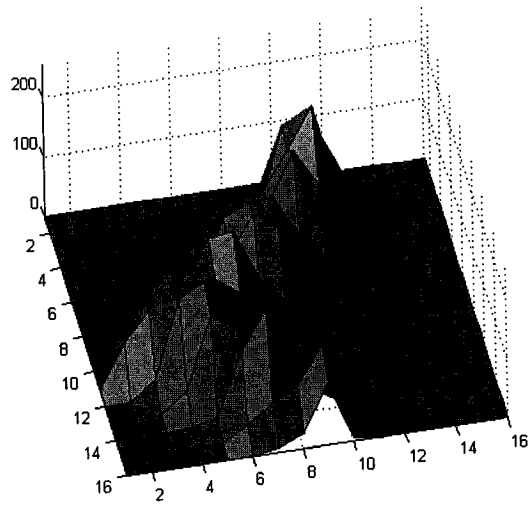


b) Median filter, n = [3 3] (3-D representation).

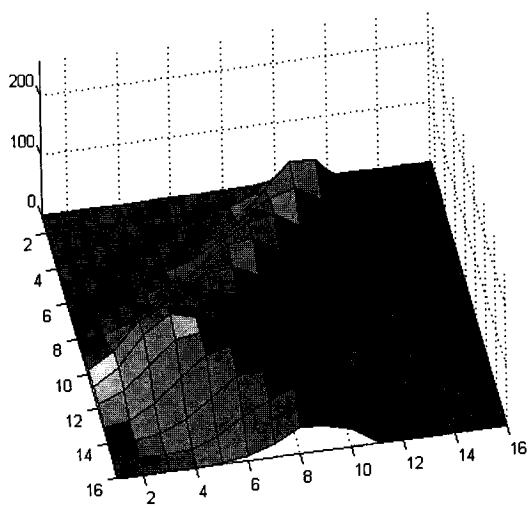


c) Median filter, n = [3 3] (2-D representation).

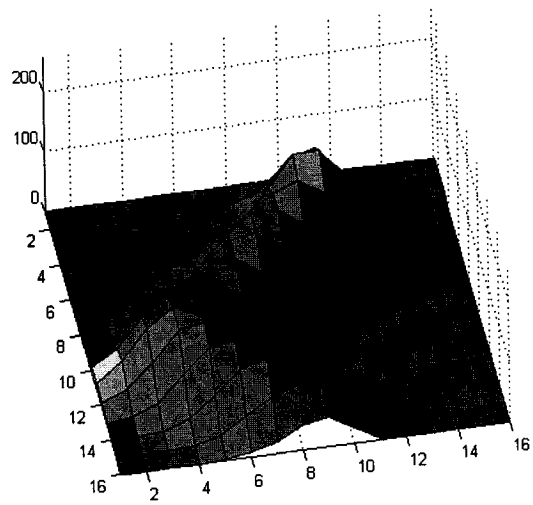
Figure 5.18. *Rotated character "C": non-linear filtering.*



a) Raw data.

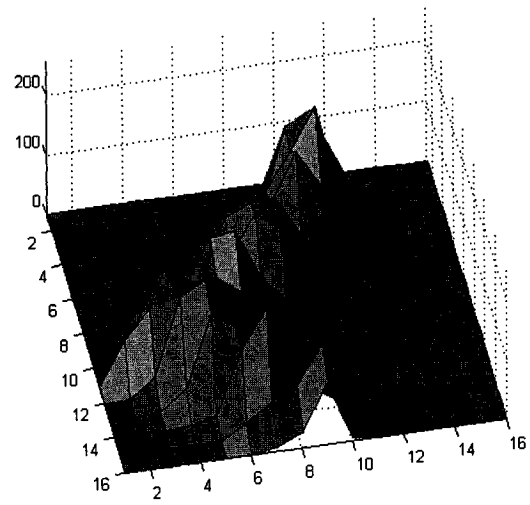


b) Average filter, n = [3 3].

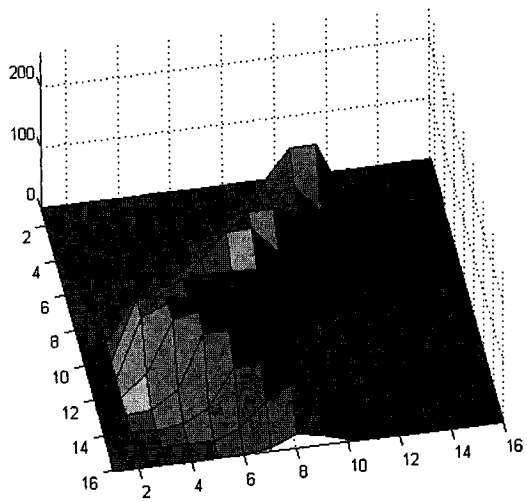


c) Gaussian filter, n = [3 3], sigma = 1.0.

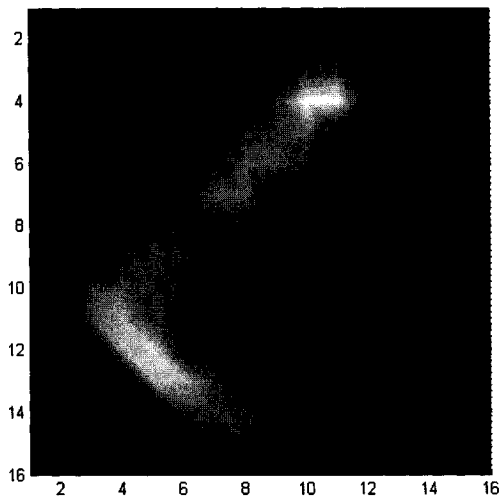
Figure 5.19 *Rotated character "L": linear filtering.*



a) Raw data.

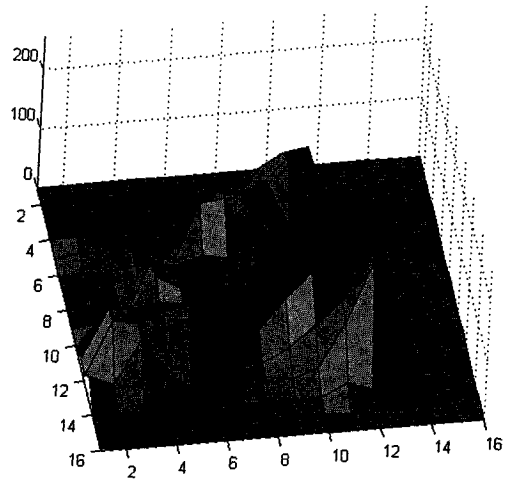


b) Median filter, n = [3 3] (3-D representation).

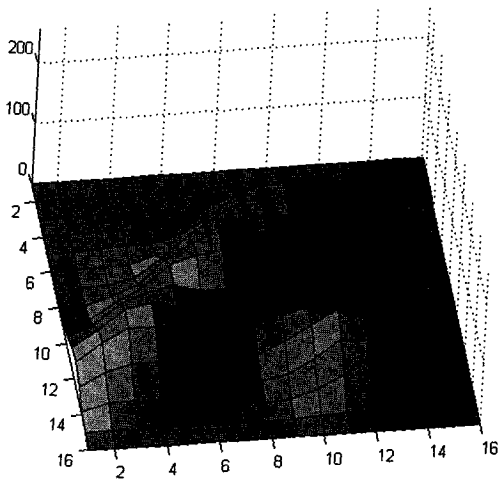


c) Median filter, n = [3 3] (2-D representation).

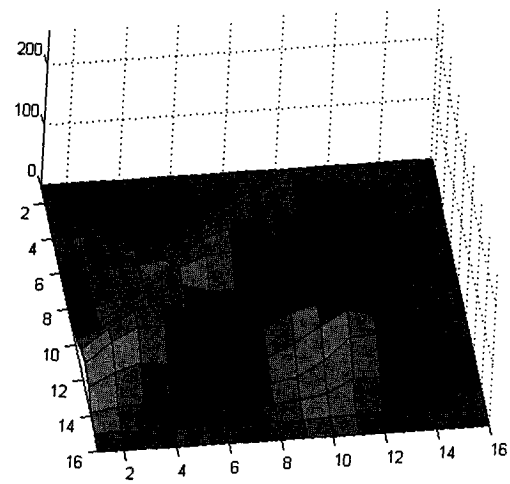
Figure 5.20. *Rotated character "L": non-linear filtering.*



a) Raw data.

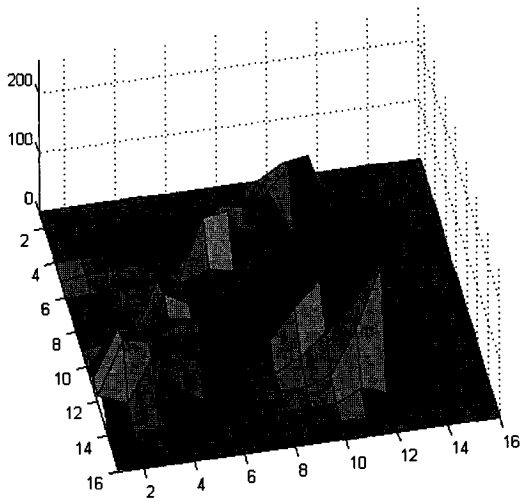


b) Average filter, n = [3 3].

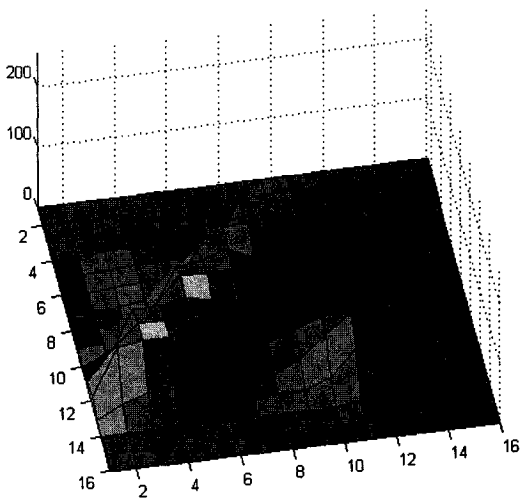


c) Gaussian filter, n = [3 3], sigma = 1.0.

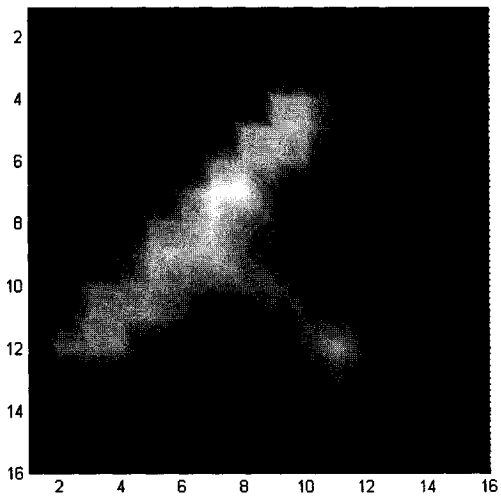
Figure 5.21 *Rotated character "T": linear filtering.*



a) Raw data.



b) Median filter, n = [3 3] (3-D representation).



c) Median filter, n = [3 3] (2-D representation).

Figure 5.22. *Rotated character "T": non-linear filtering.*

5.2.3. Discussion

The following conclusions can be drawn:

- Even though it is not depicted in the figures, during the experiments I noticed that if the size of the average and Gaussian filter masks increase, then the images have more blur, in other words, the images lose more details.
- Linear filters (averaging and Gaussian) can adjust the weights so that more smoothing is obtained in a relatively uniform area of the image and little smoothing is obtained across sharp changes in the image.
- The Gaussian mask is a better low-pass filter than the average filter. Impulsive noise has only been attenuated; in fact, each spike has also been spread in space.
- Median filters (the non linear filters) seem to suppress impulsive noise completely, create less blur in the images and preserve more details than linear filters. Therefore, a median filter preserves discontinuities better than linear filters.
- In conclusion, the best noise removal performance seems to be in the image that has been filtered with the median filter.

5.3. Image Averaging

So far, we considered the acquisition of only one instance of a raw image and tried to suppress the added noise by applying the linear or nonlinear filters described above. In practical situations, in addition to filtering, it might be a good idea to acquire, during a very short period of time, a number of instances of raw images of the same object, and then calculating the average of the acquired images. This technique is called image averaging [35] and is an attempt to compensate for instantaneous measurement errors and noise. The goal is to see how many images are needed to be acquired in order to keep the results in a certain limit of error.

Consider an image *v(x,y)*, composed of noise *n(x,y)* added to the ideal, uncorrupted image *u(x,y)* [31, 32, 35]:

$$v(x,y) = u(x,y) + n(x,y)$$

If the noise is considered uncorrelated and having zero average value, and if an image $\bar{v}(x,y)$ is formed by averaging *k* different noisy images,

$$\bar{v}(x,y) = \frac{1}{k}\sum_{i=1}^{k} v_i(x,y)$$

then the expected value of $\bar{v}$ at coordinates (x,y) is:

$$E\{\bar{v}(x,y)\} = u(x,y)$$

and the variance of $\bar{v}$ at coordinates (x,y), $\sigma_{\bar{v}}^2(x,y)$, as a function of the variance of *n* at coordinates (x,y), $\sigma_n^2(x,y)$, can be written as:

$$\sigma_{\bar{v}}^2(x,y) = \frac{1}{k}\sigma_n^2(x,y)$$

Moreover, the standard deviation at any point in the average image is given by:

$$\sigma_{\bar{v}}(x,y) = \frac{1}{\sqrt{k}}\sigma_n(x,y)$$

The last two equations indicate that if the number k of the acquired noisy images is larger, then the variability of the pixel values is smaller. But we know that $E\{\bar{v}(x,y)\} = u(x,y)$. Thus, the average image $\bar{v}(x,y)$ will tend to approach the uncorrupted image $u(x,y)$ as the number of noisy images k used in the averaging process increases [35].

The standard deviation can be seen as a measure of spread. Thus, if all the noisy images were identical, then the standard deviation would be equal to zero [37].

To validate this theory, I made the following experiment:

- With a character in the upper slot, I blocked the handle of the mechanical test set up so that the character maintained a constant force over the tactile sensor.
- I acquired $k = 1024$ raw data noisy images.
- From each image, I extracted a pixel having the same coordinates *(5, 5)* obtaining a *1 x 1024* vector.
- Then, I calculated the moving average for each possible window size (1 to 1024). For a given sequence a_i, where $i = 1,\ldots,N$, the n-moving average is defined [38] as a new sequence s_i, where $i = 1,\ldots,N-n+1$, derived from the original sequence a_i by calculating the average of n subsequent terms: $s_i = \frac{1}{n}\sum_{j=1}^{i+n-1} a_j$.

  I implemented the moving average with the MATLAB *filter* function [30].
- For each window size, I calculated the standard deviation.
- Finally, I plotted the standard deviation versus window size (figure 5.23).

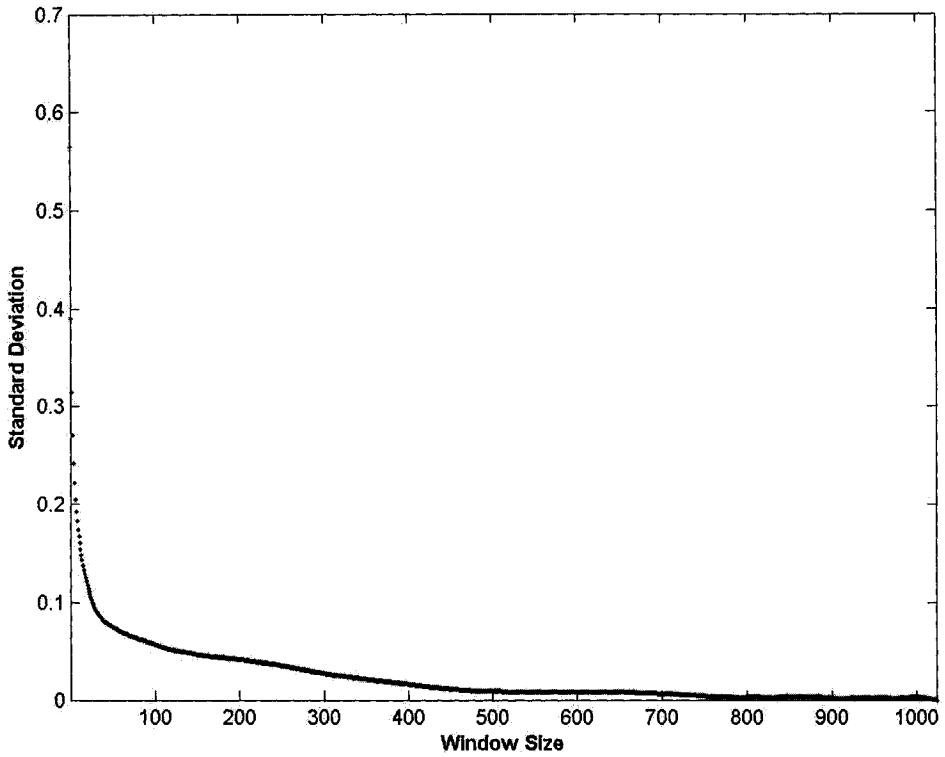


Figure 5.23. *Standard deviation versus window size.*

As we can see, for a window size greater than about 800, the standard deviation is almost equal to zero. Thus, the average image is practically identical to the ideal, uncorrupted image.

On the other hand, quite acceptable results (standard deviation less than 0.1) are obtained for a window size of about 25.

Chapter 6

Robotic Tactile Object Recognition

Based on the Smart Tactile Sensor described above, this section presents a compliant haptic sensing system that is able to provide an estimate of an object surface orientation along with a fine description of features located within the sensing area [36].

6.1. Introduction

Many researches focus on the application of haptic (tactile) sensing technology in robotics and automation with various applications in industrial assembly; assisted surgery, where palpation is important [39]; or in safe human-robot cooperation [40, 41]. More specifically, a strong interest has been put in the development of articulated hands made of a few tactile sensitive fingers for dexterous manipulation [42, 43, 44]. Germagnoli *et al.* [45] present an approach to drive a robot gripper during the exploration of unknown objects based on the recognition of a limited set of tactile primitives and the interpretation of stress maps with neural networks to locate and follow edges on the object. Pedreno-Molina *et al.* [46] propose a neural-network based approach that uses artificial skins in guiding grasping operations to ensure stable grasp of object with a two-finger robot hand. Even though

relatively good performances can now be achieved with this technology [47], the area and the complexity of the space that can be explored is limited due to the small size of the tactile sensors (mostly single point) and the fact that they are usually mounted inside the gripper fingertips [48].

On the other hand, the merge of haptic information with 3-D models obtained from optical data for telerobotic manipulation of complex objects has not yet been widely explored in spite of the numerous new possibilities that it might open. As vision-based modeling is highly sensitive to object's surface reflection characteristics, tactile sensors can be advantageously integrated to complement 3-D models. Canepa *et al.* [49] propose to extend computer vision approaches to tactile data in order to extract cutaneous primitives (measurable features of the objects, like edge and consistent properties surface). Taking advantage of the fact that tactile sensors are directly in contact with the object surface allows to precisely identifying fine shape primitives that remain invisible to vision systems.

Since a detailed representation of surface shape clearly revealed to be critical in controlling fine interactions between a robot manipulator and complex objects, these examples demonstrate the necessity for the development of haptic sensing systems targeted to robotic manipulation operations.

6.2. Kinesthetic Sensing System

The robotic haptic sensing system consists of a commercial manipulator; a custom designed instrumented passive-compliant wrist; and a tactile sensor [5], as shown in figure 6.1. Position sensors placed in the robot's joints and on the instrumented passive-compliant wrist provide the *kinesthetic* information.

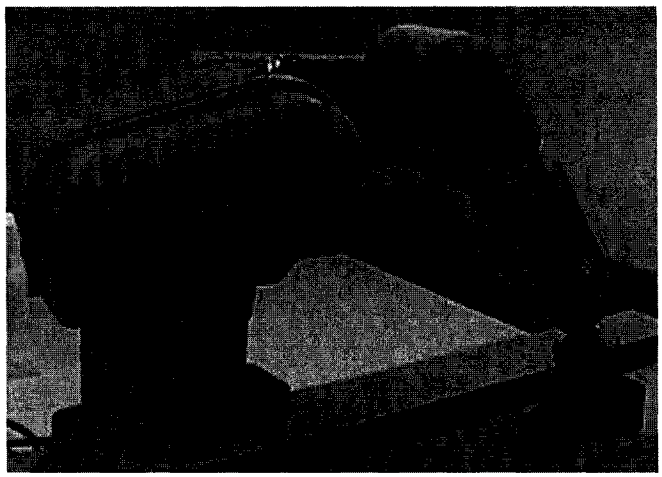


Figure 6.1. *Robotic haptic perception system.*

The amount of information acquired by haptic exploration depends on how well the probing tactile sensor accommodates the constraints of the local geometry of the object. The instrumented passive-compliant wrist shown in figure 6.2 allows the tactile sensor to better accommodate the constraints of the explored object surface and thus to increase the cutaneous information acquired by the tactile sensor. It consists of two planar plates having three relative degrees of freedom: two rotations about the x and y axes of the tactile probe's plane and one displacement along the z axis (normal to the tactile sensor's plane). A set of four linear displacement transducers are used to measure the distances between the two plates of the wrist, enabling the calculation of the relative orientation and distance between the wrist's plates.

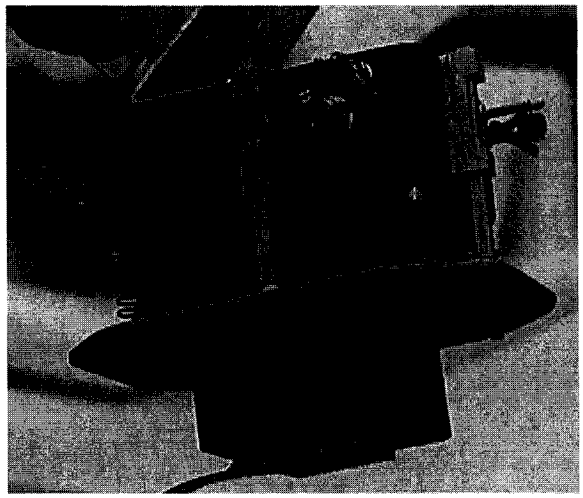


Figure 6.2. The instrumented passive-compliant wrist with the tactile sensor probe.

6.3. Object Recognition with Tactile Data

The smart haptic sensor system has been tested for model-based blind tactile recognition of 3-D objects [50]. Conveniently shaped geometric symbols are embossed on object surfaces. The symbols represent quaternary terms of a *pseudo-random array* (PRA) defined over the Galois field GF(4) ={0, 1, A, A^2}.

Using a *neural network,* the symbols recovered by tactile probing are first recognized and then clustered in a 2-by-2 PRA window that contains enough information to fully identify the absolute coordinates of the recovered window within the encoding PRA. By knowing how different object models were mapped to the PRA, it is possible to unambiguously identify the object face and the exact position of the recovered symbols on the face [50].

A two-layer feedforward *neural network* architecture is employed in order to classify tactile data representing the four encoding symbols. It has 8 neurons in the first hidden layer and 4 in the second one (the output layer), as shown in figure 6.3.

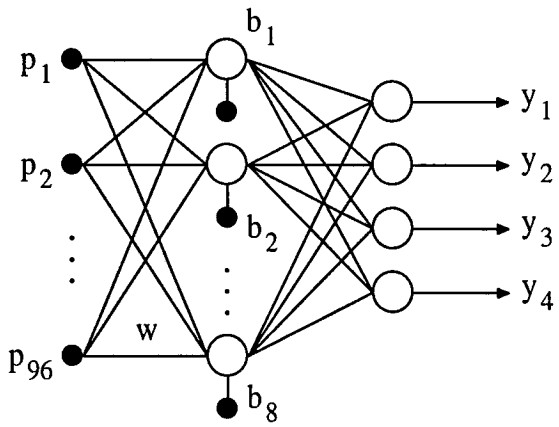


Figure 6.3. Two-layer feedforward neural network architecture for four character classification.

The network receives as inputs four 8x12-element vectors and the corresponding target will be indicated by a *one* in the position of the recognized character and *zero* elsewhere. The network is trained with 8x12 binary maps of the tactile images of the centred four characters as illustrated in figure 6.4.

The training algorithm uses a gradient descent backpropagation with momentum and adaptive learning rate. A value of 0.95 for the momentum constant and a sum-squared-error goal of 0 are applied over 5000 epochs.

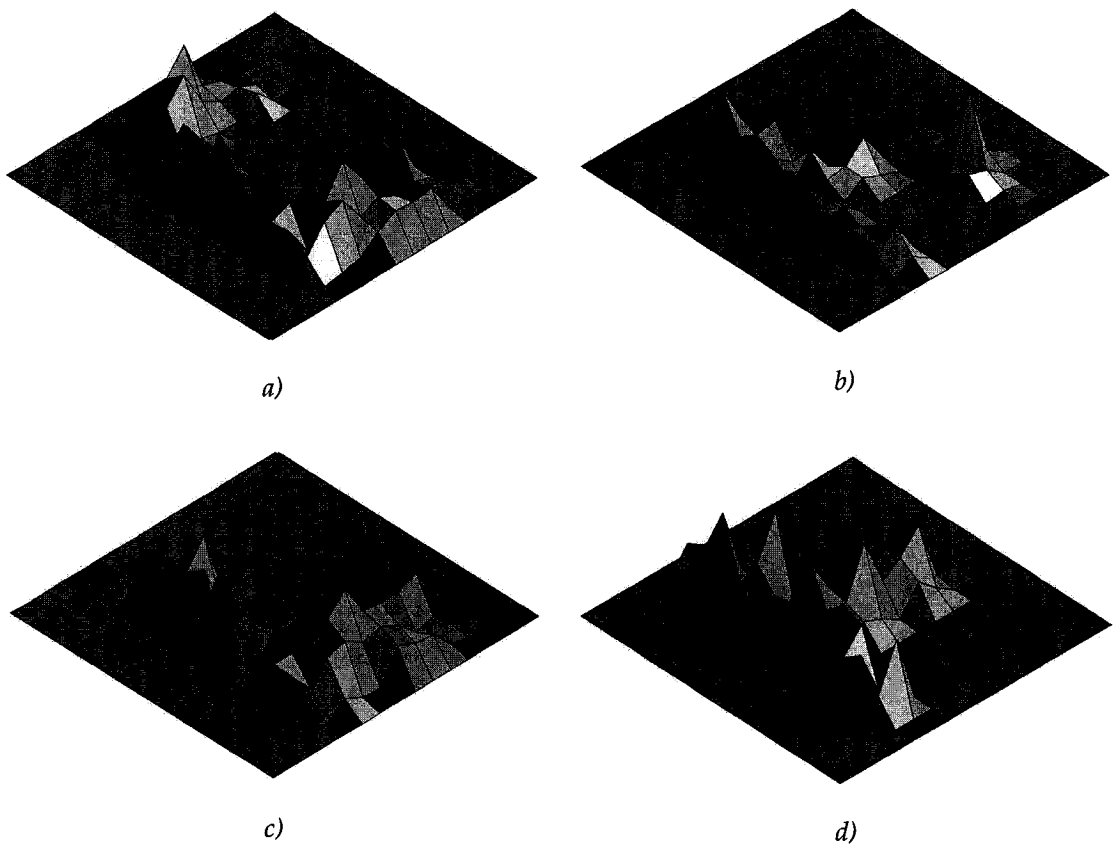


Figure 6.4. *Tactile images for the tactile recognition of the PRA encoded objects: a) character "C"; b) character "h"; c) character "L"; d) character "Y".*

The neural network was trained three times: first using two noise free images of the characters with signal levels between 0 and 1; then with a series of images corrupted with noise levels of 0.1, 0.15 and 0.2 respectively; and finally with noise free images in order to make sure that noise free images are always identified correctly.

To evaluate the generalization capability, the network was tested for different levels of noise with a mean of 0 and a standard deviation varying from 0 to 0.5. The average recognition rate and the error rate are shown in figure 6.5 and figure 6.6 respectively.

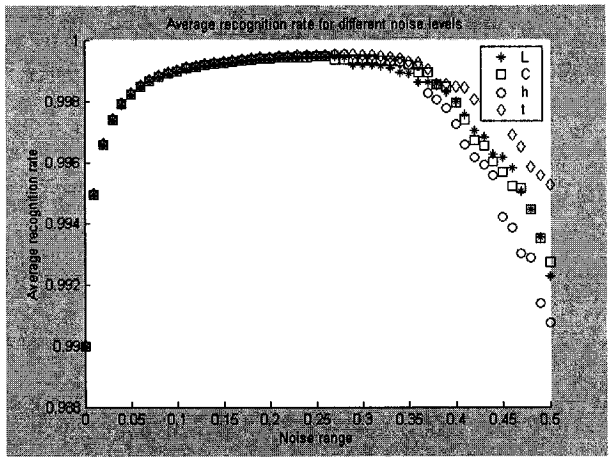


Figure 6.5. *Average recognition rate for noise ranging between 0 and 0.5.*

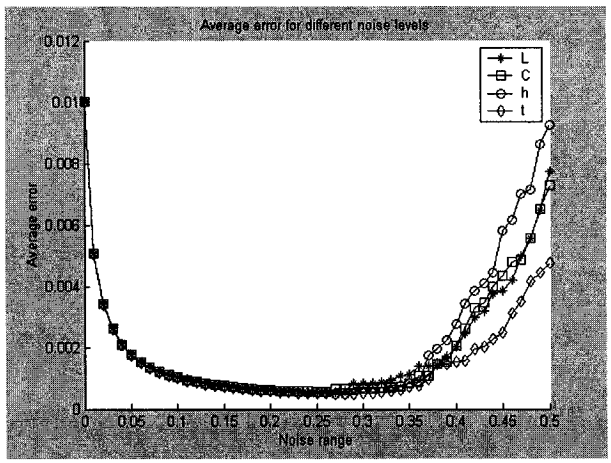


Figure 6.6. *Average error rate for noise ranging between 0 and 0.5.*

The above neural network was only able to detect characters rotated by 90 degrees. If the characters are not in vertical position, then this architecture will not work. To deal with characters rotated at different angles, the initial character recognition module is cascaded with a transformation module, which recuperates the displacement information between a raw character and the characters stored in a database of centered and aligned objects.

The transformation network has no hidden layer and receives as inputs a map of the raw character and as outputs a map of the character in the database. It has a linear activation function and is trained using gradient descent back propagation with momentum and adaptive learning rate, with a value of 0.95 for the momentum constant and a sum-squared-error goal of 0, for 530 epochs in 5 steps. The training procedure lasts for about 5 sec. The neural network learns and stores in its weights the displacement information between the vertical character and the rotated one. This information is then used to align the rotated character to the vertical position in order to be subsequently recognized.

A sample of rotated tactile data and the corresponding aligned data are shown in figure 6.7 and figure 6.8 respectively. As expected, once the training is completed, the raw object is aligned with the character in the database and this aligned object becomes the input in the tactile recognition module.

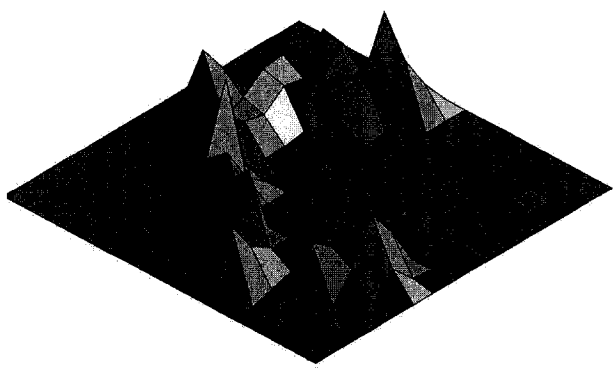


Figure 6.7. *Raw rotated data.*

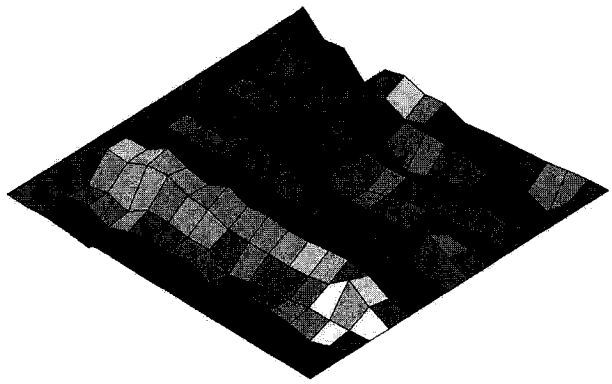


Figure 6.8. *Raw aligned data.*

In conclusion, simulation and experimental results have shown that the neural network recognition of the tactile images of the specially designed symbols embossed on object surfaces has error rates lower than 0.6% even in the case of images having up to a 50% noise ratio.

Chapter 7

Conclusions

7.1. Summary of the Research

This thesis presents a compact, fully functional prototype of a smart tactile sensor developed in the Sensing and Modeling Laboratory of the University of Ottawa. The objectives set at the beginning of the thesis were all attained.

The starting point of this work was the understanding of how different types of tactile sensors are constructed and how they work. We choose to work with a commercial Force Sensing Resistor Array Transducer from Interlink Electronics, which is composed of 256 pressure-sensing elements – named Force Sensing Resistor (FSR) transducers – arranged in a 16 by 16 matrix. Each of these elements is a variable resistor, thus its resistance decreases (the conductance increases) as a force applied to the sensor increases.

In order to find out the resistance of an FSR, this has to be isolated. The method used is to apply a positive voltage of 5 V to one node and 0 V to all other nodes of the tactile sensor. In this way, current flows only through the isolated transducer and its real resistance can be accurately measured.

Based on these observations and following recommendations from Interlink, an electronic interface was designed and implemented. Analog circuitry deal with the sensor's large resistance range and a PICmicro microcontroller is used to: convert the analog voltage into a digital value; perform the tactile sensor address selection; and transfer, through a serial communication port, the acquired data to a personal computer.

The code for the PICmicro microcontroller was written in assembly language and was transferred to the onboard microcontroller using its In Circuit Serial Programming capability that allows the user to program the microcontroller even after it is already wired in the circuitry.

A PC application, TS GUI, was designed from scratch using MATLAB's Graphical User Interface Development Environment in order to be able to acquire, process (filter), and display the tactile data received from the microcontroller. A great deal of work has been done in order to obtain a user-friendly graphical user interface. Besides the previously mentioned functions, TS GUI can also be used to save the processed images and to import previously saved tactile data. The colours of the graphical user interface can be modified and a set of messages guide the user to take proper actions.

I felt that it is a good idea to provide a demo version of TS GUI, which is a fully functional independent application that does not need an electronic interface but uses a library of previously saved tactile data files. Both versions of the software come with a comprehensive help system that provides full descriptions of the integrated functions, as well as a complete tutorial.

A set of experiments were conducted, part of which are presented in this thesis. Handcrafted wooden geometric symbols and a commercial drill press turned into a mechanical test set up were used to easily acquire tactile images. Using TS GUI, the raw tactile images were then filtered and compared. Average, Gaussian and median filtered images were obtained and presented. I found that the best images were obtained by applying median filters.

Finally, a robotic tactile object recognition application is presented. The robotic haptic sensing system consists of a commercial manipulator equipped with an instrumented passive-compliant wrist that holds the tactile sensor. Geometric symbols, representing the terms of a pseudo-random-array, are embossed on object surfaces. Using a neural network, the symbols recovered by tactile probing are first recognized and then their absolute coordinates are identified. Thus, the object face and the exact position of the recovered symbols on the face are unambiguously identified.

7.2. Future Work

This thesis presents just a prototype of the tactile sensor system, and of course, there is room for improvements. The basic idea was to develop a simple, basic, and compact tactile sensor interface that can be used chiefly in robotic applications. Further developments could include the following:

- Design of a miniaturized printed circuit board that would integrate all the electronic components in a very small area suitable to attach the electronic interface to a robot arm.
- We used a 4 MHz ceramic oscillator, but the PICmicro microcontroller can be sped up by using a 20 MHz crystal oscillator.
- The transmission speed can also be increased up to 2MHz/s by choosing a newly available PICmicro microcontroller that contains a USB interface.
- Wireless interfaces could also be considered, widening the possible applications that can use the smart tactile sensor.
- Once decided which filter best suits a specific application, the filtering function can be transferred to the microcontroller so that all the processing takes place before the transmission process. Also, calibration routines can be easily implemented in assembly code, thus the microcontroller can do this task too.

- Finally, the PC application, presently written in MATLAB code, can be rewritten in a high level language, such as C++ or Java, obtaining a platform independent executable software application.

References

[1] M. H. Lee, H. R. Nicholls, "Tactile Sensing for Mechatronics – A State of the Art Survey", in *Mechatronics – The Science of Intelligent Machines*, Pergamon Press, Vol. 9, NO. 1, pp. 1-31, February 1999.

[2] C. S. Tzafestas, "Whole-Hand Kinesthetic Feedback and Haptic Perception in Dexterous Virtual Manipulation", in *IEEE Transactions On Systems, Man and Cybernetics - Part.A: Systems and Humans*, vol. 33, no. 1, pp. 100-113, 2003.

[3] S. J. Lederman, R. L. Klatzky, D.T. Pawluk, "Lessons From the Study of Biological Touch for Robotic Haptic Sensing", in *Advanced Tactile Sensing for Robotics* (H. R. Nicholls – Editor), World Scientific, 1992.

[4] F. Castelli, "An Integrated Tactile-Thermal Robot Sensor with Capacitive Tactile Array" in *IEEE Transactions on Industry Applications*, vol. 38, no. 1, pp. 85-90, 2002.

[5] E. M. Petriu, W. S. McMath, S. K. Yeung, N. Trif, "Active Tactile Perception of Object Surface Geometric Profiles", in *IEEE Transactions on Instrumentation and Measurement*, Vol. 41, NO. 1, pp. 87-92, February 1992.

[6] ***, *The American Heritage Dictionary of the English Language*. Boston: Houghton Mifflin Company, 4th edition, 2000.

[7] S. K. Yeung, E. M. Petriu, W. S. McMath, D. C. Petriu, "High Sampling Resolution Tactile Sensor for Object Recognition", *IEEE Transactions on Instrumentation and Measurement*, Vol. 43, NO. 2, pp. 277-282, April 1994.

[8] P. P. L. Regtien, "Silicon Tactile Image Sensors", in *NATO ASI Series, Traditional and Non-Traditional Robotic Sensors*. T.C. Henderson, Springer-Verlag Berlin Heidelberg, Vol. F 63, pp 117-125, 1990.

[9] H. R. Nicholls, "Tactile Sensing Using an Optical Transduction Method", in *NATO ASI Series, Traditional and Non-Traditional Robotic Sensors*. T.C. Henderson, Springer-Verlag Berlin Heidelberg, Vol. F 63, pp 83-99, 1990.

[10] J. G. da Silva, A. A. de Carvalho, D. D. da Silva, "A Strain Gauge Tactile Sensor for Finger-Mounted Applications", in *IEEE Transactions on Instrumentation and Measurement*, Vol. 51, NO. 1, pp. 18-22, February 2002.

[11] H. Kawasaki, T. Komatsu, K Uchiyama, "Dextrous Anthropomorphic Robot Hand with Distributed Tactile Sensor: Gifu Hand II", in *IEEE/ASME Transactions on Mechatronics*, Vol. 7, NO. 3, pp. 296-303, September 2002.

[12] B. Peine, J. Son, R. Howe, "Remote Palpation Instruments for Minimally Invasive Surgery", [Online]. Available: *http://biorobotics.harvard.edu/research/bill.html.*

[13] ***, "Massachusetts Institute of Technology and London teams report first transatlantic touch", [Online], Available: *http://web.mit.edu/newsoffice/nr/2002/touchlab3.html.*

[14] S. Baglio, G. Muscato, N. Savalli, "Tactile Measuring Systems for the Recognition of Unknown Surfaces", in *IEEE Transactions on Instrumentation and Measurement*, Vol. 51, NO. 3, pp. 522-531, June 2002.

[15] ***, *Interlink Electronics Tactile Array Robotics*, Interlink Electronics Inc., Santa Barbara, CA, 1989.

[16] A. Gauthier, P. Perras, M. Gibault, *Smart Tactile Sensor Array Interface.* CEG 4392 – Computer Systems Design Project, University of Ottawa, School of Information Technology and Engineering, 2001, pp 6-17.

[17] ***, *FSR Force Sensing Resistor Integration Guide and Evaluation Parts Catalog.* Interlink Electronics Inc., Camarillo, CA, [Online]. Available: *http://www.interlinkelec.com/.*

[18] S. Yaniger, J. P. Rivers, *Force and Position Sensing Resistors – An Emerging Technology.* Interlink Electronics Inc., Santa Barbara, CA.

[19] ***, *FSR Technical Specifications.* Interlink Electronics Inc., Santa Barbara, CA, 1991.

[20] E. M. Petriu, T. E. Whalen, V. Z. Groza, "Haptic Perception System for Robotic Tele-Manipulation", in *Proceedings of INES 2002, 6th International Conference on Intelligent Engineering Systems 2002* , pp. 51-55, Opatija, Croatia, May 2002.

[21] R. Yang, P.S. Krishnapr R. Yang, P.S. Krishnaprasad, "Tactile Sensing and Inverse Problems", *Technical Report – Electrical Eng. Dept. and System Research Center*, University of Maryland, 1988.

[22] R.E. Saad, A. Bonen, K.C. Smith, B. Benhabib, "Distributed-Force Recovery for a Planar Photoelastic Tactile Sensor", in *IEEE Transactions on Instrumentation and Measurement*, vol. 45, no. 2, pp. 541-546, 1996.

[23] E. M. Petriu, S. R. Das, S. K. Yeung "Robotic Tactile Perception", in *Proceedings of the IEEE Instrumentation and Measurement Technology Conference*, pp. 1266-1271, Venice, Italy, May 1999.

[24] T. H. Speeter, "A Tactile Sensing System for Robotic Manipulation", in *The Int. Journal of Robotics Research*, vol. 9, no. 6, pp. 25-36, 1990.

[25] Y. C. Pati, P. S. Krisnaprasad, M. C. Peckerar, "An Analog Neural Network Solution to the Inverse Problem of Early Taction", in *IEEE Transactions on Robotics and Automation*, vol. 8, no. 2, pp. 196-212, 1992.

[26] R. Q. Yang, M. W. Siegel, "A Finger-Tip Optical Sensor Array Sorting System", in *Proceedings of Sensors Conf., CASA/SME Technical Paper MS85-991*, pp. 1-11, 1995.

[27] A. W. De Groot, "Effect of Sensor Size in Robotic Tactile Sensor Arrays", in *Robotica*, vol. 6, pp. 285-287, 1988.

[28] ***, PIC18FXX2 Data Sheet: High Performance, Enhanced FLASH Microcontrollers with 10-Bit A/D. Microchip Technology Inc., [Online]. Available: *http://www.microchip.com/.*

[29] M. Predko, *Programming and Customizing PICmicro Microcontrollers*. New York: McGraw-Hill, 2002.

[30] ***, MATLAB Help: The MathWorks, Inc., [Online]. Available: *http://www.mathworks.com/access/helpdesk/help/techdoc/matlab.shtml*

[31] E. Trucco, A. Verri, *Introductory Techniques for 3-D Computer Vision*. Upper Saddle River, New Jersey: Prentice Hall, 1998, pp. 51-64.

[32] L. G. Shapiro, G. C. Stockman, *Computer Vision*. Upper Saddle River, New Jersey: Prentice Hall, 1998, pp. 128-151.

[33] R. Jain, R. Kasturi, B. G. Schunck, *Machine Vision*. New York: McGraw-Hill, 1995, pp. 112-137.

[34] D. A. Forsyth, J. Ponce, *Computer Vision: A Modern Approach*. Upper saddle River, New Jersey: Prentice Hall, 2003, pp. 135-140.

[35] K. S. Fu, R. C. Gonzales, C. S. G. Lee, *Robotics: Control, Sensing, Vision, and Intelligence*. New York: McGraw-Hill, 1987, pp. 331-342.

[36] C. Pasca, P. Payeur, E. M. Petriu, A. M. Cretu, "Intelligent Haptic Sensor System for Robotic Manipulation", in *Proceedings of the IEEE Instrumentation and Measurement Technology Conference*, pp. 279-284, Como, Italy, May 2004.

[37] C. J. Wild, G. A. F. Seber, *Chance Encounters: A First Course in Data Analysis and Inference*. New York, NY: John Wiley & Sons, Inc., 2000, pp. 61-66, 71-72.

[38] E. W. Weisstein, "Moving Average", from "MathWorld – A Wolfram Web Resource", [Online]. Available: *http://mathworld.wolfram.com/MovingAverage.html/.*

[39] R. D. Howe, W. Peine, D. A. Kontarinis, J.S. Son, "Remote Palpation Technology", in *IEEE Engineering in Medicine and Biology*, pp. 318-323, 1998.

[40] H. Tobita, T. Kawamure, Y. Sugimoto, H. Nakamura, "The Development of 'Safe Partner Equipment' ", in *Proceedings of the IEEE Int. Conf. on Robotics and Automation*, pp. 2420-2426, Nagoya, Japan, 1995.

[41] T. Itoh, K. Kosuge, T. Fukuda, "Human-Machine Cooperative Telemanipulation with Motion and Force Scaling Using Task-Oriented Virtual Tool Dynamics", in *IEEE Transactions on Robotics and Automation*, vol. 16, no. 5, pp. 505-516, 2000.

[42] R. D. Howe, M. R. Cutkosky, "Dynamic Tactile Sensing–Perception of Fine Surface-Features with Stress Rate Sensing", in *IEEE Transactions on Robotics and Automation*, vol. 9, pp. 140-151, 1993.

[43] D. Johnston, P. Zhang, J. Hollerbach, S. Jacobsen, "A Full Tactile Sensing Suite for Dexterous Robot Hands and Use in Contact Force Control", in *Proceedings of the IEEE Int. Conf. on Robotics and Automation*, pp. 3222-3227, Minneapolis, MN, 1996.

[44] H. Zhan, N. N. Chen, "Control of Contact via Tactile Sensing", in *IEEE Transactions on Robotics and Automation*, vol. 16, no. 5, pp. 482-495, 2000.

[45] F. Germagnoli, G. Magenes, "A Neural Network-Based System for Tactile Exploratory Tasks", in *Proceedings of the Int. Workshop on Neural Networks for Identification, Control, Robotics and Signal/Image Processing*, pp. 458-466, 1996.

[46] J. L. Pedrero-Molina, A. Guerrero-Gonzalez, J. Lopez-Coronado, "A Neural Controller for a Robotic Hand with Artificial Tactile Skins in Grasping Tasks", in *Proceedings of the IEEE Int. Conf. on Systems, Man and Cybernetics*, vol. 1, pp. 161-165, 2000.

[47] D. Taddeucci, C. Laschi, R. Lazzarini, R. Magni, P. Dario, A. Starita, "An Approach to Integrated Tactile Perception", in *Proceedings of the IEEE Int. Conf. on Robotics and Automation*, pp. 3100-3105, Albuquerque, NM, 1997.

[48] T. Fischer, D. Rapela, H. Woern, "Joint Controller for the Object-Pose Controlling on Multifinger Grippers", in *Proceedings of the Int. Conf. on Advanced Intelligent Mechatronics*, 1999.

[49] G. Canepa, O. Sottile, D. De Rossi, "Extraction of Cutaneous Primitives from Tactile Sensor Images", in *Proceedings of the IEEE Int. Conf. on Systems, Man and Cybernetics*, vol. 3, pp. 2641-2646, 1994.

[50] E. M. Petriu, S.K.S. Yeung, S.R. Das, H.J.W. Spoelder, "Robotic Tactile Recognition of Pseudo-Random Encoded Objects", in *Proceedings of the IEEE Int. Conf. on Instrumentation and Measurement*, pp. 1397-1401, Vail, CO, 2003.

Appendix A

Schematic Diagrams

Figure A.1 contains the general schematic diagram of the tactile sensor electronic interface. The schematic diagrams of the Voltage Follower and the Current to Voltage Converter blocks are detailed in figure A.2 and A.3, respectively. The regulated power supplies are presented in figure A.4.

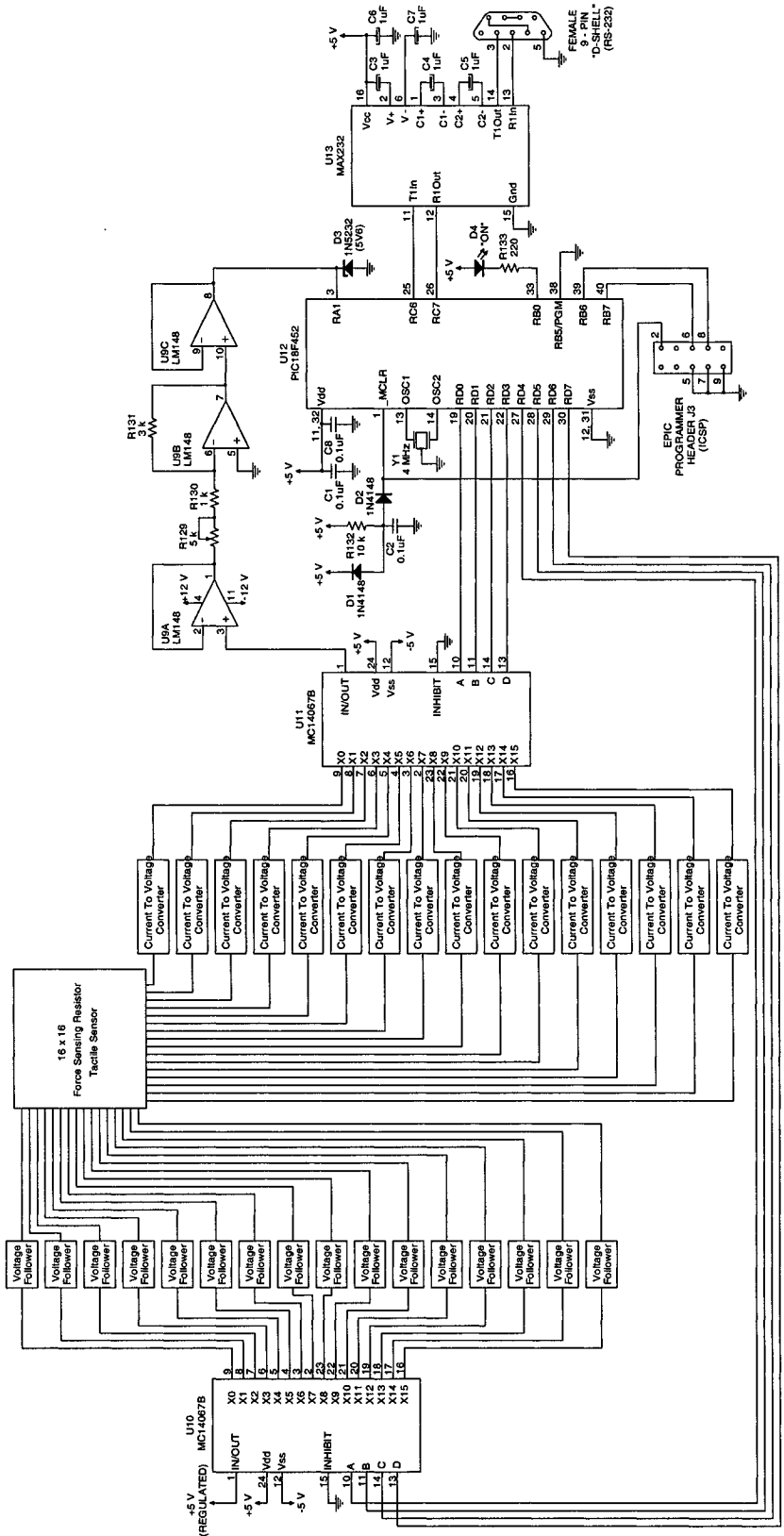
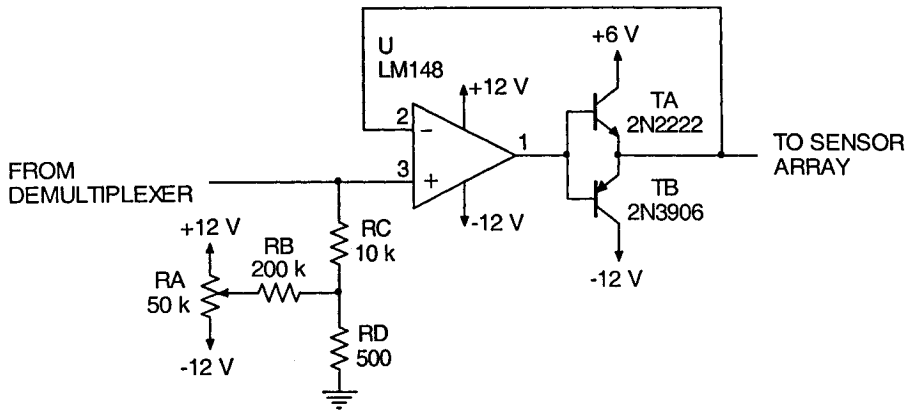


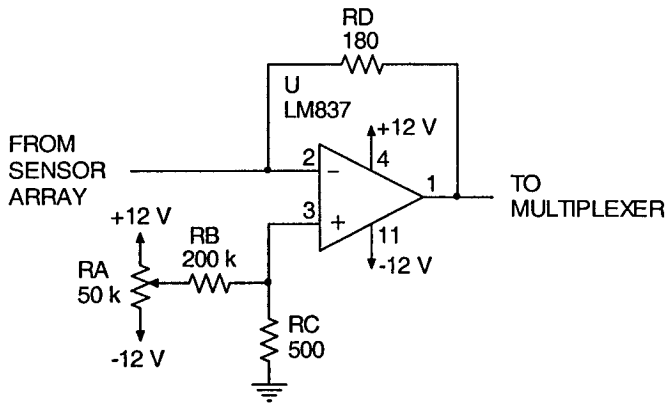
Figure A.1. *Schematic diagram.*



U	RA	RB	RC	RD	TA	TB
U1A	R1	R2	R3	R4	T1	T2
U1B	R5	R6	R7	R8	T3	T4
U1C	R9	R10	R11	R12	T5	T6
U1D	R13	R14	R15	R16	T7	T8
U2A	R17	R18	R19	R20	T9	T10
U2B	R21	R22	R23	R24	T11	T12
U2C	R25	R26	R27	R28	T13	T14
U2D	R29	R30	R31	R32	T15	T16
U3A	R33	R34	R35	R36	T17	T18
U3B	R37	R38	R39	R40	T19	T20
U3C	R41	R42	R43	R44	T21	T22
U3D	R45	R46	R47	R48	T23	T24
U4A	R49	R50	R51	R52	T25	T26
U4B	R53	R54	R55	R56	T27	T28
U4C	R57	R58	R59	R60	T29	T30
U4D	R61	R62	R63	R64	T31	T32

Figure A.2. *Voltage follower.*

The table above containes the part reference numbers for all of the 16 instances of the voltage follower that are shown in the schematic diagram (figure A.1).



U	RA	RB	RC	RD
U5A	R65	R66	R67	R68
U5B	R69	R70	R71	R72
U5C	R73	R74	R75	R76
U5D	R77	R78	R79	R80
U6A	R81	R82	R83	R84
U6B	R85	R86	R87	R88
U6C	R89	R90	R91	R92
U6D	R93	R94	R95	R96
U7A	R97	R98	R99	R100
U7B	R101	R102	R103	R104
U7C	R105	R106	R107	R108
U7D	R109	R110	R111	R112
U8A	R113	R114	R115	R116
U8B	R117	R118	R119	R120
U8C	R121	R122	R123	R124
U8D	R125	R126	R127	R128

Figure A.3. *Current-to-voltage converter.*

The table above containes the part reference numbers for all of the 16 instances of the current-to-voltage converter that are shown in the schematic diagram (figure A.1).

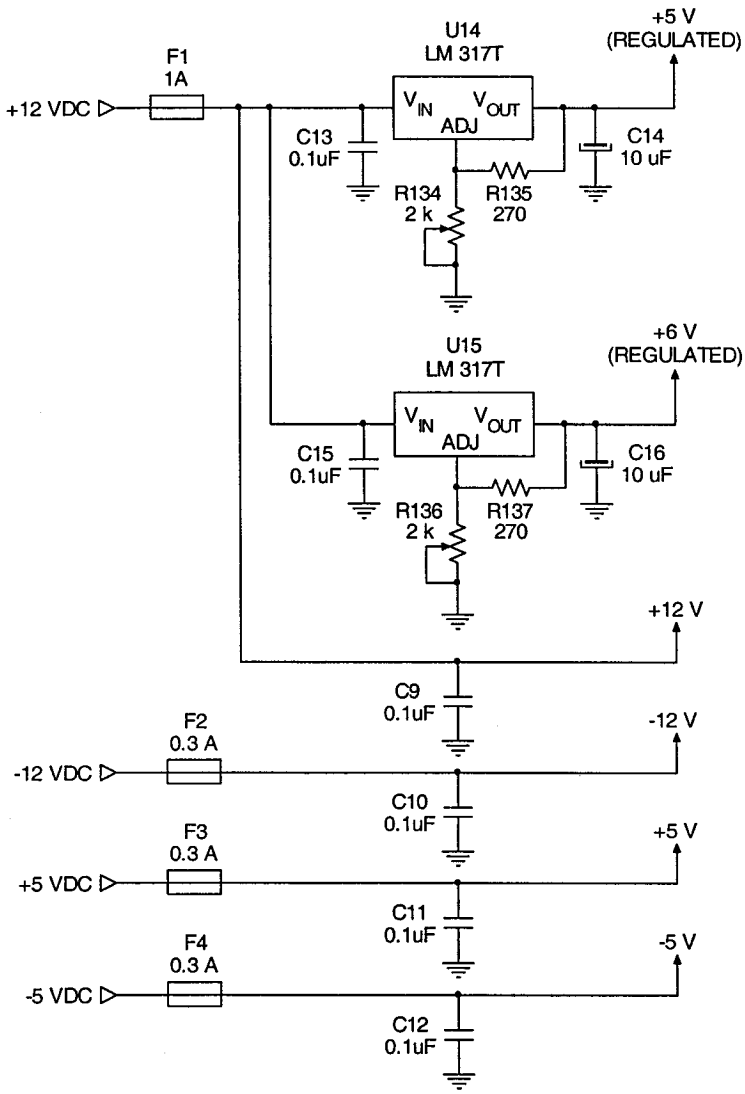


Figure A.4. *Regulated power supplies.*

Appendix B

Source Code for the PIC18F452 Microcontroller

The following assembly code is implemented in the PIC18F452 microcontroller. It was developed using the MPLAB Integrated Development Environment (IDE) from Microchip. The microcontroller was programmed using the EPIC Plus PICmicro MCU Programmer from microEngineering Labs, Inc.

```
; Author: Codrin Pasca
; Supervisor: Dr. Emil M. Petriu
; University of Ottawa
; School of Information Technology and Engineering
; Sensing and Modeling Research Laboratory
; 800 King Edward
; Ottawa, Ontario, Canada, K1N 6N5
; Date: January 2004

      LIST R=DEC, F=INHX32
      INCLUDE "P18F452.INC"

;Program Configuration Register 1H
            __CONFIG    _CONFIG1H, _OSCS_OFF_1H & _XT_OSC_1H
;Program Configuration Register 2L
            __CONFIG    _CONFIG2L, _BOR_OFF_2L & _PWRT_ON_2L
;Program Configuration Register 2H
            __CONFIG    _CONFIG2H, _WDT_OFF_2H
;Program Configuration Register 3H
            __CONFIG    _CONFIG3H, _CCP2MX_OFF_3H
;Program Configuration Register 4L
            __CONFIG    _CONFIG4L, _STVR_OFF_4L & _LVP_ON_4L & _DEBUG_OFF_4L
;Program Configuration Register 5L
            __CONFIG    _CONFIG5L, _CP0_OFF_5L & _CP1_OFF_5L & _CP2_OFF_5L &
_CP3_OFF_5L
;Program Configuration Register 5H
            __CONFIG    _CONFIG5H, _CPB_OFF_5H & _CPD_OFF_5H
;Program Configuration Register 6L
            __CONFIG    _CONFIG6L, _WRT0_OFF_6L & _WRT1_OFF_6L &
_WRT2_OFF_6L & _WRT3_OFF_6L
;Program Configuration Register 6H
            __CONFIG    _CONFIG6H, _WRTC_OFF_6H & _WRTB_OFF_6H &
_WRTD_OFF_6H
;Program Configuration Register 7L
            __CONFIG    _CONFIG7L, _EBTR0_OFF_7L & _EBTR1_OFF_7L &
_EBTR2_OFF_7L & _EBTR3_OFF_7L
;Program Configuration Register 7H
            __CONFIG    _CONFIG7H, _EBTRB_OFF_7H

      cblock 0x000              ; Start of general purpose registers
            count1              ; Used in delay routine
            counta              ; Used in delay routine
            countb              ; Used in delay routine
            countL              ; Used in delay routine
            i
            received_ch         ; Received character
            muxaddress          ; Analog mux address
      endc
```

```
      cblock 0x100              ; FSR000 - FSR255
fsr000, fsr001, fsr002, fsr003, fsr004, fsr005, fsr006, fsr007, fsr008, fsr009
fsr010, fsr011, fsr012, fsr013, fsr014, fsr015, fsr016, fsr017, fsr018, fsr019
fsr020, fsr021, fsr022, fsr023, fsr024, fsr025, fsr026, fsr027, fsr028, fsr029
fsr030, fsr031, fsr032, fsr033, fsr034, fsr035, fsr036, fsr037, fsr038, fsr039
fsr040, fsr041, fsr042, fsr043, fsr044, fsr045, fsr046, fsr047, fsr048, fsr049
fsr050, fsr051, fsr052, fsr053, fsr054, fsr055, fsr056, fsr057, fsr058, fsr059
fsr060, fsr061, fsr062, fsr063, fsr064, fsr065, fsr066, fsr067, fsr068, fsr069
fsr070, fsr071, fsr072, fsr073, fsr074, fsr075, fsr076, fsr077, fsr078, fsr079
fsr080, fsr081, fsr082, fsr083, fsr084, fsr085, fsr086, fsr087, fsr088, fsr089
fsr090, fsr091, fsr092, fsr093, fsr094, fsr095, fsr096, fsr097, fsr098, fsr099
fsr100, fsr101, fsr102, fsr103, fsr104, fsr105, fsr106, fsr107, fsr108, fsr109
fsr110, fsr111, fsr112, fsr113, fsr114, fsr115, fsr116, fsr117, fsr118, fsr119
fsr120, fsr121, fsr122, fsr123, fsr124, fsr125, fsr126, fsr127, fsr128, fsr129
fsr130, fsr131, fsr132, fsr133, fsr134, fsr135, fsr136, fsr137, fsr138, fsr139
fsr140, fsr141, fsr142, fsr143, fsr144, fsr145, fsr146, fsr147, fsr148, fsr149
fsr150, fsr151, fsr152, fsr153, fsr154, fsr155, fsr156, fsr157, fsr158, fsr159
fsr160, fsr161, fsr162, fsr163, fsr164, fsr165, fsr166, fsr167, fsr168, fsr169
fsr170, fsr171, fsr172, fsr173, fsr174, fsr175, fsr176, fsr177, fsr178, fsr179
fsr180, fsr181, fsr182, fsr183, fsr184, fsr185, fsr186, fsr187, fsr188, fsr189
fsr190, fsr191, fsr192, fsr193, fsr194, fsr195, fsr196, fsr197, fsr198, fsr199
fsr200, fsr201, fsr202, fsr203, fsr204, fsr205, fsr206, fsr207, fsr208, fsr209
fsr210, fsr211, fsr212, fsr213, fsr214, fsr215, fsr216, fsr217, fsr218, fsr219
fsr220, fsr221, fsr222, fsr223, fsr224, fsr225, fsr226, fsr227, fsr228, fsr229
fsr230, fsr231, fsr232, fsr233, fsr234, fsr235, fsr236, fsr237, fsr238, fsr239
fsr240, fsr241, fsr242, fsr243, fsr244, fsr245, fsr246, fsr247, fsr248, fsr249
fsr250, fsr251, fsr252, fsr253, fsr254, fsr255
      endc

      org    0                  ; org sets the origin, 0x0000 for the 18F452
                                ; This is where the program starts running

;******** Initialize (clear) fsr000 - fsr255 (write 0 in every fsrxxx) (start)
      movlw  0                  ; Initialize counter
      movwf  i
      lfsr   0, fsr000          ; FSR0 = address of fsr000

Loop_init
      clrf   INDF0              ; Clear the content of fsrxxx
      incf   FSR0L, f           ; Jump to the address of the next fsrxxx
      decfsz i, f               ; Decrement counter, skip if 0
            bra Loop_init       ; Counter not 0, so loop to clear next fsrxxx
;******** Initialize (clear) fsr000 - fsr255 (write 0 in every fsrxxx) (end)

;*************** Initialize PORTD (start)
      setf   PORTD              ; Initialize PORTD (muxaddress)
      clrf   TRISD              ; Set all the PORTD bits to Output
```

```
;*************** Initialize PORTD (end)

;*************** Set up the ADC (start)
       nop
       clrf   PORTA                ; Use PORTA as the Input

       movlw  B'00000000'          ; For 18F452, Make 10 Bit ADC "Left
                                   ; Justified"
       movwf  ADCON1

       movlw  0x089                ; Setup ADCON0 for ADC Conversion
       movwf  ADCON0               ;   ADCS1:ADCS0 - 10 for /32 Clock
                                   ;   CHS2:CHS0 - 001 for RA1/AN1
                                   ;   Go/_Done - 0
                                   ;   Unimplemented - 0
                                   ;   ADON - 1
;*************** Set up the ADC (end)

;*************** Set up the Transmitter (start)
       movlw  12                   ; Set Baud Rate:
                                   ;  1200 bps (SPBRG = 51,  BRGH = 0)
                                   ;  2400 bps (SPBRG = 25,  BRGH = 0)
                                   ;  1200 bps (SPBRG = 207, BRGH = 1)
                                   ;  2400 bps (SPBRG = 103, BRGH = 1)
                                   ;  9600 bps (SPBRG = 25,  BRGH = 1)
                                   ; 19200 bps (SPBRG = 12,  BRGH = 1)
       movwf  SPBRG                ; Baud Rate Generator Register

       movlw  B'00100100'          ; 8-bit transmission, transmit enable,
       movwf  TXSTA                ; asynchrounous mode, high speed (BRGH = 1).
       movlw  B'10010000'          ; Enable transmission by enabling Serial
       movwf  RCSTA                ; port (continuous receiveing)
;*************** Set up the Transmitter (end)

Loop_main

;*************** LED ON = PIC18F452 OK (start)
       bcf    PORTB, 0
       bcf    TRISB, 0

       call   Delay_250
       bsf    PORTB, 0
       call   Delay_250
       bcf    PORTB, 0
;*************** LED ON = PIC18F452 OK (stop)

;*************** Wait for a character from Matlab (start)
       movlw  b'01110010'          ; 'r' (Raw Data) is the character to be
```

```
                                ; received and compared
      movwf  received_ch        ; with the contents of received_ch

Loop_receive
      btfss   PIR1, RCIF        ; Wait for received character set
            bra   Loop_receive  ; Nothing received
      movf    RCREG, w          ; Get the received character (put it in w)
      bcf     PIR1, RCIF        ; Clear the character present flag

      cpfseq received_ch        ; Received character equal 'a', so skip
            bra    Loop_receive ; Different character received, so try again
;*************** Wait for a character from Matlab (stop)

;*************** Writing the memory (start)
      movlw  0                  ; Initialize counter
      movwf  i

      lfsr   0, fsr000          ; FSR0 = address of fsr000

      clrf   muxaddress         ; Initialize fsr starting address (for muxs)
                                ; muxaddress = 0000 0000

Loop_write
                                ; Select MUX Address: write address to PORTD
      movf   muxaddress, w      ;
      movwf  PORTD              ; PORTD = muxaddress

            ; Analog to Digital Conversion (start)
      movlw  3                  ; Wait 12 usec for ADC to Charge
      addlw  0x0FF              ; Take One Away to Setup the Charge
      btfss  STATUS, Z
            bra    $ - 4

      bsf    ADCON0, GO         ; Turn on the ADC
      btfsc  ADCON0, GO         ; Wait for it to Complete
            bra    $ - 2
            ; Analog to Digital Conversion (end)

      movf   ADRESH, w          ; Conversion result
      movwf  INDF0              ; INDF0 = value of fsr000

      incf   muxaddress, f      ; Increment MUX address
      incf   FSR0L, f           ; Increment memory location
      decfsz i, f               ; Decrement counter, skip if 0
            bra Loop_write      ; Counter not 0, so loop to write next fsrxxx
;*************** Writing the memory (end)
```

```
;*************** Reading the memory, then transmission (start)
       movlw  0                    ; Initialize counter
       movwf  i

       lfsr   0, fsr000            ; FSR0 = address of fsr000
Loop_read
       movf   INDF0, w
       movwf  TXREG                ; Transmit the fsr value
       call   Delay

       incf   FSR0L, f             ; Jump to the address of the next fsrxxx
       decfsz i, f                 ; Decrement counter, skip if 0
              bra Loop_read        ; Counter not 0, so loop to transmit
                                   ; next fsrxxx
;*************** Reading the memory, then transmission (end)

              bra Loop_main

;*************** Delay routines (start)
Delay                              ; Delay the code

       movlw  255
       movwf  countL               ; countL = 25 = adjustable delay time
Delay1
       decfsz countL,f             ; Loop countL = 25 times
                                   ; When zero, skip and return (finish delay)
              bra    Delay1
       return
;---------------------------------------------------------------------
Delay_250
       movlw  d'250'               ;Delay 250 ms (4 MHz clock)
       movwf  count1
d1     movlw  0xC7
       movwf  counta
       movlw  0x01
       movwf  countb
Delay_0
       decfsz counta, f
              bra    $+4
       decfsz countb, f
              bra    Delay_0

       decfsz count1 ,f
              bra    d1
       retlw  0x00
;*************** Delay routines (end)
       end
```

Appendix C

Source Code for the Graphical User Interface

The following is the MATLAB source code for the TS GUI graphical user interface. It was developed using GUIDE (Graphical User Interface Development Environment) from MATLAB.

```
% TS GUI - version 1.0, March 2004
% Author: Codrin Pasca
% Supervisor: Dr. Emil M. Petriu
% University of Ottawa
% School of Information Technology and Engineering
% Sensing and Modeling Research Laboratory
% 800 King Edward
% Ottawa, Ontario, Canada, K1N 6N5
% ------------------------------------------------------------------------
% ------------------------------------------------------------------------
function varargout = ts(varargin)
% TS M-file for ts.fig
%      TS, by itself, creates a new TS or raises the existing
%      singleton*.
%
%      H = TS returns the handle to a new TS or the handle to
%      the existing singleton*.
%
%      TS('Property','Value',...) creates a new TS using the
%      given property value pairs. Unrecognized properties are passed via
%      varargin to ts_OpeningFcn.  This calling syntax produces a
%      warning when there is an existing singleton*.
%
%      TS('CALLBACK') and TS('CALLBACK',hObject,...) call the
%      local function named CALLBACK in TS.M with the given input
%      arguments.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES
% Edit the above text to modify the response to help ts
% Last Modified by GUIDE v2.5 18-Mar-2004 18:00:35

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @ts_OpeningFcn, ...
                   'gui_OutputFcn',  @ts_OutputFcn, ...
                   'gui_LayoutFcn',  [], ...
                   'gui_Callback',   []);
if nargin & isstr(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
```

```
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT
% --------------------------------------------------------------------
% --------------------------------------------------------------------
% --- Executes just before ts is made visible.
function ts_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   unrecognized PropertyName/PropertyValue pairs from the
%            command line (see VARARGIN)

% Display the Menu Bar and the Tool Bar
% MenuBar is set to 'figure' in the "Property Inspector" in Guide, so we
%   don't need the following line of code:
%   set(handles.figure1, 'MenuBar', 'figure')
set(handles.figure1, 'ToolBar', 'figure')

% Initialize menu items
handles.current_data = 0;

handles.average_filter = 0;
handles.mean_equal_weights_filter = 0;
handles.mean_single_peak_filter = 0;
handles.gaussian_filter = 0;
handles.median_filter = 0;

handles.LastFile_raw = 0;
handles.LastFile_average = 0;
handles.LastFile_mean_e_w = 0;
handles.LastFile_mean_s_p = 0;
handles.LastFile_gaussian = 0;
handles.LastFile_median = 0;

handles.selected_filter = 1;
handles.selected_neighborhood = 1;
handles.selected_deviation = 1;
handles.selected_plotting = 1;
handles.selected_colormap = 1;
handles.selected_shading = 1;

handles.zeros_line = zeros(1,16);
handles.zeros_column = zeros(18,1);

% Don't display anything at start up
```

```
axes(handles.axes_raw)
nothing_in_raw = (1000)*ones(256);
surfl(nothing_in_raw)
axis([1 16 1 16 0 256])
grid on

axes(handles.axes_filtered)
axis([1 16 1 16 0 256])
grid on

% Disable the "Deviation (Sigma)" popup menu because the default filter is
% the Average Filter, which only depends on the Neighborhood value
set(handles.text_deviation, 'Visible', 'off');
set(handles.popupmenu_deviation, 'Visible', 'off');

% Display the "Welcome message" at start up
set(handles.text_message_welcome, 'Visible', 'on')
set(handles.text_message_apply, 'Visible', 'off')

% Choose default command line output for ts
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes ts wait for user response (see UIRESUME)
% uiwait(handles.figure1);

% --------------------------------------------------------------------------
% --------------------------------------------------------------------------
% --- Outputs from this function are returned to the command line.
function varargout = ts_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

% --------------------------------------------------------------------------
% --------------------------------------------------------------------------
% --- Executes on button press in pushbutton_get.
function pushbutton_get_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton_get (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% In the Demo Version of the TS GUI the function "pushbutton_get_Callback"
% is the same as the menu function "Workspace/Raw/Import..."  because
% there is no electronic interface available. Thus, the following
% line is commented and used only in TS GUI demo
% menu_workspace_raw_import_Callback(hObject, eventdata, handles)

% In the full Version of the TS GUI, the function "pushbutton_get_Callback"
% acquires a new raw image through the electronic interface:

% ********************************************************************************
% Output object
s = serial('COM1');
s.InputBufferSize = 256;
s.BaudRate = 19200;
fopen(s);
% --------------------------------------------------------------------------------
% Syncronize with the microcontroller
fprintf(s, 'r')
out = fread(s, [16 16], 'uint8');
fclose(s);
delete(s);
clear s;
% --------------------------------------------------------------------------------
raw = out';
% ********************************************************************************

handles.current_data = raw;

% Plot the raw image
axes(handles.axes_raw)
surfl(handles.current_data)
colormap gray
axis([1 16 1 16 0 256])
% Plot the filtered image
axes(handles.axes_filtered)
nothing_in_filtered = (1000)*ones(256);
surfl(nothing_in_filtered)
axis([1 16 1 16 0 256])

% Display the "Apply message" and erase the "Welcome message"
set(handles.text_message_apply, 'Visible', 'on')
set(handles.text_message_welcome, 'Visible', 'off')

Initialize filters.
handles.average_filter = 0;
handles.mean_equal_weights_filter = 0;
handles.mean_single_peak_filter = 0;
handles.gaussian_filter = 0;
```

```
handles.median_filter = 0;

% Update handles structure
guidata(hObject, handles);

% --------------------------------------------------------------------
% --------------------------------------------------------------------
% --- Executes on button press in pushbutton_help.
function pushbutton_help_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton_help (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%open help_index
HelpPath = which('help_index.html');
web(HelpPath);
% --------------------------------------------------------------------
% --------------------------------------------------------------------
% --- Executes on button press in pushbutton_close.
function pushbutton_close_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton_close (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
user_response = close_dialog('Title','Close TS GUI Demo?');
switch lower(user_response)
case 'no'
       % take no action
case 'yes'
       % Prepare to close GUI application window
       delete(handles.figure1)
end
% --------------------------------------------------------------------
% --------------------------------------------------------------------
% --- Executes on button press in pushbutton_apply.
function pushbutton_apply_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton_apply (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

raw_data = handles.current_data;
if raw_data == 0
    Warning_apply_mat;
else
    axes(handles.axes_filtered)

    % Selected Neighborhood determines the Applied Neighborhood
    switch handles.selected_neighborhood;
        case 1 % User selects [1 1]
            applied_neighborhood = [1 1];
```

```
    case 2 % User selects [2 2]
        applied_neighborhood = [2 2];
    case 3 % User selects [3 3]
        applied_neighborhood = [3 3];
    case 4 % User selects [4 4]
        applied_neighborhood = [4 4];
    case 5 % User selects [5 5]
        applied_neighborhood = [5 5];
    case 6 % User selects [6 6]
        applied_neighborhood = [6 6];
    case 7 % User selects [7 7]
        applied_neighborhood = [7 7];
    case 8 % User selects [8 8]
        applied_neighborhood = [8 8];
    case 9 % User selects [9 9]
        applied_neighborhood = [9 9];
end

% Selected Deviation determines the Applied Deviation
switch handles.selected_deviation;
    case 1 % User selects 0.1
        applied_deviation = 0.1;
    case 2 % User selects 0.2
        applied_deviation = 0.2;
    case 3 % User selects 0.3
        applied_deviation = 0.3;
    case 4 % User selects 0.4
        applied_deviation = 0.4;
    case 5 % User selects 0.5
        applied_deviation = 0.5;
    case 6 % User selects 0.6
        applied_deviation = 0.6;
    case 7 % User selects 0.7
        applied_deviation = 0.7;
    case 8 % User selects 0.8
        applied_deviation = 0.8;
    case 9 % User selects 0.9
        applied_deviation = 0.9;
    case 10 % User selects 1.0
        applied_deviation = 1.0;
end

% Selected filter
switch handles.selected_filter;

    case 1 % User selects Average
        % Average Filter
        % Apply the filter
```

```
    h = fspecial('average', applied_neighborhood);
    handles.average_filter = filter2(h, handles.current_data, 'same');
    % Plotting method
    switch handles.selected_plotting;
        case 1 % User selects Surf
            surf(handles.average_filter);
        case 2 % User selects Surfl
            surfl(handles.average_filter);
    end

case 2 % User selects Mean Equal Weights
    % Mean Filter (Averaging) - Equal Weights
    % Zero-pad the edges of handles.current_data
    padded_current_data = handles.current_data;
    padded_current_data = [handles.zeros_line; ...
            padded_current_data; handles.zeros_line];
    padded_current_data = [handles.zeros_column, ...
            padded_current_data, handles.zeros_column];
    % Apply the filter
    for i=2:17
        for j=2:17
            handles.mean_equal_weights_filter(i-1,j-1) = (1/9) * ...
                (padded_current_data(i-1,j-1) + ...
                padded_current_data(i-1,j) + ...
                padded_current_data(i-1,j+1) + ...
                padded_current_data(i,j-1) + ...
                padded_current_data(i,j) + ...
                padded_current_data(i,j+1) + ...
                padded_current_data(i+1,j-1) + ...
                padded_current_data(i+1,j) + ...
                padded_current_data(i+1,j+1));
        end
    end
    % Plotting method
    switch handles.selected_plotting;
        case 1 % User selects Surf
            surf(handles.mean_equal_weights_filter);
        case 2 % User selects Surfl
            surfl(handles.mean_equal_weights_filter);
    end

case 3 % User selects Mean Single Peak
    % Mean Filter (Averaging) - Single Peak
    % Zero-pad the edges of handles.current_data
    padded_current_data = handles.current_data;
    padded_current_data = [handles.zeros_line; ...
            padded_current_data; handles.zeros_line];
    padded_current_data = [handles.zeros_column, ...
```

```
            padded_current_data, handles.zeros_column];
    % Apply the filter
    for i=2:17
        for j=2:17
            handles.mean_single_peak_filter(i-1,j-1) = (1/16) * ...
                (padded_current_data(i-1,j-1) + ...
                (2*padded_current_data(i-1,j)) + ...
                padded_current_data(i-1,j+1) + ...
                (2*padded_current_data(i,j-1)) + ...
                (4*padded_current_data(i,j)) + ...
                (2*padded_current_data(i,j+1)) + ...
                padded_current_data(i+1,j-1) + ...
                (2*padded_current_data(i+1,j)) + ...
                padded_current_data(i+1,j+1));
        end
    end
    % Plotting method
    switch handles.selected_plotting;
        case 1 % User selects Surf
            surf(handles.mean_single_peak_filter);
        case 2 % User selects Surfl
            surfl(handles.mean_single_peak_filter);
    end

case 4 % User selects Gaussian
    % Gaussian Filter
    % Apply the filter
    h = fspecial('gaussian', applied_neighborhood, applied_deviation);
    handles.gaussian_filter = filter2(h, handles.current_data, ...
        'same');
    % Plotting method
    switch handles.selected_plotting;
        case 1 % User selects Surf
            surf(handles.gaussian_filter);
        case 2 % User selects Surfl
            surfl(handles.gaussian_filter);
    end

case 5 % User selects Median
    % Median Filter
    % Apply the filter
    handles.median_filter = medfilt2(handles.current_data, ...
        applied_neighborhood);
    % Plotting method
    switch handles.selected_plotting;
        case 1 % User selects Surf
            surf(handles.median_filter);
        case 2 % User selects Surfl
```

```
                surfl(handles.median_filter);
        end
end

% Colormap
switch handles.selected_colormap;
    case 1 % User selects Autumn
        colormap autumn;
    case 2 % User selects Bone
        colormap bone;
    case 3 % User selects Colorcube
        colormap colorcube;
    case 4 % User selects Cool
        colormap cool;
    case 5 % User selects Copper
        colormap copper;
    case 6 % User selects Flag
        colormap flag;
    case 7 % User selects Gray
        colormap gray;
    case 8 % User selects Hot
        colormap hot;
    case 9 % User selects Hsv
        colormap hsv;
    case 10 % User selects Jet
        colormap jet;
    case 11 % User selects Lines
        colormap lines;
    case 12 % User selects Pink
        colormap pink;
    case 13 % User selects Prism
        colormap prism;
    case 14 % User selects Spring
        colormap spring;
    case 15 % User selects Summer
        colormap summer;
    case 16 % User selects White
        colormap white;
    case 17 % User selects Winter
        colormap winter;
end

% Shading
switch handles.selected_shading;
    case 1 % User selects Faceted
        shading faceted; % default
    case 2 % User selects Flat
        shading flat;
```

```
        case 3 % User selects Interp
            shading interp;
    end

    % Axis
    axis([1 16 1 16 0 256])

    % Erase any message
    set(handles.text_message_apply, 'Visible', 'off')
    set(handles.text_message_welcome, 'Visible', 'off')

    % Update handles structure
    guidata(hObject,handles)
end
% --------------------------------------------------------------------
% --------------------------------------------------------------------
% --- Executes on selection change in popupmenu_select.
function popupmenu_select_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu_select (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns popupmenu_select
%                                                 contents as cell array
%        contents{get(hObject,'Value')} returns selected item from
%                                                 popupmenu_select

val = get(hObject,'Value');
str = get(hObject, 'String');
switch str{val};
case 'Average' % User selects Average
    handles.selected_filter = 1;
    % Enable the "Neighborhood" popup menu
    set(handles.text_neighborhood, 'Visible', 'on');
    set(handles.popupmenu_neighborhood, 'Visible', 'on');
    % Disable the "Deviation (Sigma)" popup menu
    set(handles.text_deviation, 'Visible', 'off');
    set(handles.popupmenu_deviation, 'Visible', 'off');
case 'Mean Equal Weights' % User selects Mean Equal Weights
    handles.selected_filter = 2;
    % Disable the "Neighborhood" popup menu
    set(handles.text_neighborhood, 'Visible', 'off');
    set(handles.popupmenu_neighborhood, 'Visible', 'off');
    % Disable the "Deviation (Sigma)" popup menu
    set(handles.text_deviation, 'Visible', 'off');
    set(handles.popupmenu_deviation, 'Visible', 'off');
case 'Mean Single Peak' % User selects Mean Single Peak
    handles.selected_filter = 3;
```

```
    % Disable the "Neighborhood" popup menu
    set(handles.text_neighborhood, 'Visible', 'off');
    set(handles.popupmenu_neighborhood, 'Visible', 'off');
    % Disable the "Deviation (Sigma)" popup menu
    set(handles.text_deviation, 'Visible', 'off');
    set(handles.popupmenu_deviation, 'Visible', 'off');
case 'Gaussian' % User selects Gaussian
    handles.selected_filter = 4;
    %Enable the "Neighborhood" popup menu
    set(handles.text_neighborhood, 'Visible', 'on');
    set(handles.popupmenu_neighborhood, 'Visible', 'on');
    % Enable the "Deviation (Sigma)" popup menu
    set(handles.text_deviation, 'Visible', 'on');
    set(handles.popupmenu_deviation, 'Visible', 'on');
case 'Median' % User selects Median
    handles.selected_filter = 5;
    % Enable the "Neighborhood" popup menu
    set(handles.text_neighborhood, 'Visible', 'on');
    set(handles.popupmenu_neighborhood, 'Visible', 'on');
    % Disable the "Deviation (Sigma)" popup menu
    set(handles.text_deviation, 'Visible', 'off');
    set(handles.popupmenu_deviation, 'Visible', 'off');
end
% Update handles structure
guidata(hObject,handles)
% --------------------------------------------------------------------------------
% --------------------------------------------------------------------------------
% --- Executes on selection change in popupmenu_neighborhood.
function popupmenu_neighborhood_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu_neighborhood (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns popupmenu_neighborhood
%                                                contents as cell array
%        contents{get(hObject,'Value')} returns selected item from
%                                                popupmenu_neighborhood

val = get(hObject,'Value');
str = get(hObject, 'String');
switch str{val};
case '[1 1]' % User selects [1 1]
    handles.selected_neighborhood = 1;
case '[2 2]' % User selects [2 2]
    handles.selected_neighborhood = 2;
case '[3 3]' % User selects [3 3]
    handles.selected_neighborhood = 3;
case '[4 4]' % User selects [4 4]
```

```
    handles.selected_neighborhood = 4;
case '[5 5]' % User selects [5 5]
    handles.selected_neighborhood = 5;
case '[6 6]' % User selects [6 6]
    handles.selected_neighborhood = 6;
case '[7 7]' % User selects [7 7]
    handles.selected_neighborhood = 7;
case '[8 8]' % User selects [8 8]
    handles.selected_neighborhood = 8;
case '[9 9]' % User selects [9 9]
    handles.selected_neighborhood = 9;
end

% Update handles structure
guidata(hObject,handles)
% --------------------------------------------------------------------
% --------------------------------------------------------------------
% --- Executes on selection change in popupmenu_deviation.
function popupmenu_deviation_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu_deviation (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns popupmenu_deviation
%                                                 contents as cell array
%        contents{get(hObject,'Value')} returns selected item from
%                                                 popupmenu_deviation

val = get(hObject,'Value');
str = get(hObject, 'String');
switch str{val};
case '0.1' % User selects 0.1
    handles.selected_deviation = 1;
case '0.2' % User selects 0.2
    handles.selected_deviation = 2;
case '0.3' % User selects 0.3
    handles.selected_deviation = 3;
case '0.4' % User selects 0.4
    handles.selected_deviation = 4;
case '0.5' % User selects 0.5
    handles.selected_deviation = 5;
case '0.6' % User selects 0.6
    handles.selected_deviation = 6;
case '0.7' % User selects 0.7
    handles.selected_deviation = 7;
case '0.8' % User selects 0.8
    handles.selected_deviation = 8;
case '0.9' % User selects 0.9
```

```
    handles.selected_deviation = 9;
case '1.0' % User selects 1.0
    handles.selected_deviation = 10;
end

% Update handles structure
guidata(hObject,handles)
% --------------------------------------------------------------------
% --------------------------------------------------------------------
% --- Executes on selection change in popupmenu_plotting.
function popupmenu_plotting_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu_plotting (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns popupmenu_plotting
%                                                 contents as cell array
%        contents{get(hObject,'Value')} returns selected item from
%                                                 popupmenu_plotting

val = get(hObject,'Value');
str = get(hObject, 'String');
switch str{val};
case 'Surf' % User selects Surf
    handles.selected_plotting = 1;
case 'Surf1' % User selects Surf1
    handles.selected_plotting = 2;
end

% Update handles structure
guidata(hObject,handles)
% --------------------------------------------------------------------
% --------------------------------------------------------------------
% --- Executes on selection change in popupmenu_colormap.
function popupmenu_colormap_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu_colormap (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns popupmenu_colormap
%                                                 contents as cell array
%        contents{get(hObject,'Value')} returns selected item from
%                                                 popupmenu_colormap

val = get(hObject,'Value');
str = get(hObject, 'String');
switch str{val};
case 'Autumn' % User selects Autumn
```

```
    handles.selected_colormap = 1;
case 'Bone' % User selects Bone
    handles.selected_colormap = 2;
case 'Colorcube' % User selects Colorcube
    handles.selected_colormap = 3;
case 'Cool' % User selects Cool
    handles.selected_colormap = 4;
case 'Copper' % User selects Copper
    handles.selected_colormap = 5;
case 'Flag' % User selects Flag
    handles.selected_colormap = 6;
case 'Gray' % User selects Gray
    handles.selected_colormap = 7;
case 'Hot' % User selects Hot
    handles.selected_colormap = 8;
case 'Hsv' % User selects Hsv
    handles.selected_colormap = 9;
case 'Jet' % User selects Jet
    handles.selected_colormap = 10;
case 'Lines' % User selects Lines
    handles.selected_colormap = 11;
case 'Pink' % User selects Pink
    handles.selected_colormap = 12;
case 'Prism' % User selects Prism
    handles.selected_colormap = 13;
case 'Spring' % User selects Spring
    handles.selected_colormap = 14;
case 'Summer' % User selects Summer
    handles.selected_colormap = 15;
case 'White' % User selects White
    handles.selected_colormap = 16;
case 'Winter' % User selects Winter
    handles.selected_colormap = 17;
end

% Update handles structure
guidata(hObject,handles)
% --------------------------------------------------------------------------
% --------------------------------------------------------------------------
% --- Executes on selection change in popupmenu_shading.
function popupmenu_shading_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu_shading (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns popupmenu_shading
%                                                 contents as cell array
%        contents{get(hObject,'Value')} returns selected item from
```

```
%                                                    popupmenu_shading

val = get(hObject,'Value');
str = get(hObject, 'String');
switch str{val};
case 'Faceted' % User selects Faceted
    handles.selected_shading = 1;
case 'Flat' % User selects Flat
    handles.selected_shading = 2;
case 'Interp' % User selects Interp
    handles.selected_shading = 3;
end

% Update handles structure
guidata(hObject,handles)
% --------------------------------------------------------------------
% --------------------------------------------------------------------
% --------------------------------------------------------------------
function menu_colours_Callback(hObject, eventdata, handles)
% hObject    handle to menu_colours (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% --------------------------------------------------------------------
function menu_colours_background_Callback(hObject, eventdata, handles)
% hObject    handle to menu_colours_background (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% --------------------------------------------------------------------
function menu_colours_frames_Callback(hObject, eventdata, handles)
% hObject    handle to menu_colours_frames (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% --------------------------------------------------------------------
function menu_colours_pushbuttons_Callback(hObject, eventdata, handles)
% hObject    handle to menu_colours_pushbuttons (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% --------------------------------------------------------------------
function menu_colours_raw_Callback(hObject, eventdata, handles)
% hObject    handle to menu_colours_raw (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% --------------------------------------------------------------------
function menu_colours_filtered_Callback(hObject, eventdata, handles)
% hObject    handle to menu_colours_filtered (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% --------------------------------------------------------------------
```

```
% --------------------------------------------------------------------
function menu_colours_background_blue_Callback(hObject, eventdata, handles)
% hObject    handle to menu_colours_background_blue (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.figure1, 'Color', 'blue')
set(handles.text_raw, 'BackgroundColor', 'blue')
set(handles.text_filtered, 'BackgroundColor', 'blue')
% --------------------------------------------------------------------
function menu_colours_background_cyan_Callback(hObject, eventdata, handles)
% hObject    handle to menu_colours_background_cyan (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.figure1, 'Color', 'cyan')
set(handles.text_raw, 'BackgroundColor', 'cyan')
set(handles.text_filtered, 'BackgroundColor', 'cyan')
% --------------------------------------------------------------------
function menu_colours_background_gray_Callback(hObject, eventdata, handles)
% hObject    handle to menu_colours_background_gray (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.figure1, 'Color', [0.5,0.5,0.5])
set(handles.text_raw, 'BackgroundColor', [0.5,0.5,0.5])
set(handles.text_filtered, 'BackgroundColor', [0.5,0.5,0.5])
% --------------------------------------------------------------------
function menu_colours_background_green_Callback(hObject, eventdata, handles)
% hObject    handle to menu_colours_background_green (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.figure1, 'Color', 'green')
set(handles.text_raw, 'BackgroundColor', 'green')
set(handles.text_filtered, 'BackgroundColor', 'green')
% --------------------------------------------------------------------
function menu_colours_background_lightgray_Callback(hObject, eventdata, ...
    handles)
% hObject    handle to menu_colours_background_lightgray (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.figure1, 'Color', [0.75,0.75,0.75])
set(handles.text_raw, 'BackgroundColor', [0.75,0.75,0.75])
set(handles.text_filtered, 'BackgroundColor', [0.75,0.75,0.75])
% --------------------------------------------------------------------
function menu_colours_background_magenta_Callback(hObject, eventdata, handles)
% hObject    handle to menu_colours_background_magenta (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.figure1, 'Color', 'magenta')
set(handles.text_raw, 'BackgroundColor', 'magenta')
```

```
set(handles.text_filtered, 'BackgroundColor', 'magenta')
% --------------------------------------------------------------------
function menu_colours_background_pink_Callback(hObject, eventdata, handles)
% hObject    handle to menu_colours_background_pink (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.figure1, 'Color', [1.0,0.4,0.6])
set(handles.text_raw, 'BackgroundColor', [1.0,0.4,0.6])
set(handles.text_filtered, 'BackgroundColor', [1.0,0.4,0.6])
% --------------------------------------------------------------------
function menu_colours_background_red_Callback(hObject, eventdata, handles)
% hObject    handle to menu_colours_background_red (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.figure1, 'Color', 'red')
set(handles.text_raw, 'BackgroundColor', 'red')
set(handles.text_filtered, 'BackgroundColor', 'red')
% --------------------------------------------------------------------
function menu_colours_background_teal_Callback(hObject, eventdata, handles)
% hObject    handle to menu_colours_background_teal (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.figure1, 'Color', [0.5,0.75,0.75])
set(handles.text_raw, 'BackgroundColor', [0.5,0.75,0.75])
set(handles.text_filtered, 'BackgroundColor', [0.5,0.75,0.75])
% --------------------------------------------------------------------
function menu_colours_background_white_Callback(hObject, eventdata, handles)
% hObject    handle to menu_colours_background_white (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.figure1, 'Color', 'white')
set(handles.text_raw, 'BackgroundColor', 'white')
set(handles.text_filtered, 'BackgroundColor', 'white')
% --------------------------------------------------------------------
function menu_colours_background_yellow_Callback(hObject, eventdata, handles)
% hObject    handle to menu_colours_background_yellow (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.figure1, 'Color', 'yellow')
set(handles.text_raw, 'BackgroundColor', 'yellow')
set(handles.text_filtered, 'BackgroundColor', 'yellow')
% --------------------------------------------------------------------
% --------------------------------------------------------------------
function menu_colours_frames_blue_Callback(hObject, eventdata, handles)
% hObject    handle to menu_colours_frames_blue (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.frame1, 'BackgroundColor', 'blue')
```

```
set(handles.frame2, 'BackgroundColor', 'blue')
set(handles.text_select, 'BackgroundColor', 'blue')
set(handles.text_neighborhood, 'BackgroundColor', 'blue')
set(handles.text_deviation, 'BackgroundColor', 'blue')
set(handles.text_plotting, 'BackgroundColor', 'blue')
set(handles.text_colormap, 'BackgroundColor', 'blue')
set(handles.text_shading, 'BackgroundColor', 'blue')
set(handles.text_message_welcome, 'BackgroundColor', 'blue')
set(handles.text_message_apply, 'BackgroundColor', 'blue')
% --------------------------------------------------------------------
function menu_colours_frames_cyan_Callback(hObject, eventdata, handles)
% hObject    handle to menu_colours_frames_cyan (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.frame1, 'BackgroundColor', 'cyan')
set(handles.frame2, 'BackgroundColor', 'cyan')
set(handles.text_select, 'BackgroundColor', 'cyan')
set(handles.text_neighborhood, 'BackgroundColor', 'cyan')
set(handles.text_deviation, 'BackgroundColor', 'cyan')
set(handles.text_plotting, 'BackgroundColor', 'cyan')
set(handles.text_colormap, 'BackgroundColor', 'cyan')
set(handles.text_shading, 'BackgroundColor', 'cyan')
set(handles.text_message_welcome, 'BackgroundColor', 'cyan')
set(handles.text_message_apply, 'BackgroundColor', 'cyan')
% --------------------------------------------------------------------
function menu_colours_frames_gray_Callback(hObject, eventdata, handles)
% hObject    handle to menu_colours_frames_gray (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.frame1, 'BackgroundColor', [0.5,0.5,0.5])
set(handles.frame2, 'BackgroundColor', [0.5,0.5,0.5])
set(handles.text_select, 'BackgroundColor', [0.5,0.5,0.5])
set(handles.text_neighborhood, 'BackgroundColor', [0.5,0.5,0.5])
set(handles.text_deviation, 'BackgroundColor', [0.5,0.5,0.5])
set(handles.text_plotting, 'BackgroundColor', [0.5,0.5,0.5])
set(handles.text_colormap, 'BackgroundColor', [0.5,0.5,0.5])
set(handles.text_shading, 'BackgroundColor', [0.5,0.5,0.5])
set(handles.text_message_welcome, 'BackgroundColor', [0.5,0.5,0.5])
set(handles.text_message_apply, 'BackgroundColor', [0.5,0.5,0.5])
% --------------------------------------------------------------------
function menu_colours_frames_green_Callback(hObject, eventdata, handles)
% hObject    handle to menu_colours_frames_green (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.frame1, 'BackgroundColor', 'green')
set(handles.frame2, 'BackgroundColor', 'green')
set(handles.text_select, 'BackgroundColor', 'green')
set(handles.text_neighborhood, 'BackgroundColor', 'green')
```

```
set(handles.text_deviation, 'BackgroundColor', 'green')
set(handles.text_plotting, 'BackgroundColor', 'green')
set(handles.text_colormap, 'BackgroundColor', 'green')
set(handles.text_shading, 'BackgroundColor', 'green')
set(handles.text_message_welcome, 'BackgroundColor', 'green')
set(handles.text_message_apply, 'BackgroundColor', 'green')
% --------------------------------------------------------------------
function menu_colours_frames_lightgray_Callback(hObject, eventdata, handles)
% hObject    handle to menu_colours_frames_lightgray (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.frame1, 'BackgroundColor', [0.75,0.75,0.75])
set(handles.frame2, 'BackgroundColor', [0.75,0.75,0.75])
set(handles.text_select, 'BackgroundColor', [0.75,0.75,0.75])
set(handles.text_neighborhood, 'BackgroundColor', [0.75,0.75,0.75])
set(handles.text_deviation, 'BackgroundColor', [0.75,0.75,0.75])
set(handles.text_plotting, 'BackgroundColor', [0.75,0.75,0.75])
set(handles.text_colormap, 'BackgroundColor', [0.75,0.75,0.75])
set(handles.text_shading, 'BackgroundColor', [0.75,0.75,0.75])
set(handles.text_message_welcome, 'BackgroundColor', [0.75,0.75,0.75])
set(handles.text_message_apply, 'BackgroundColor', [0.75,0.75,0.75])
% --------------------------------------------------------------------
function menu_colours_frames_magenta_Callback(hObject, eventdata, handles)
% hObject    handle to menu_colours_frames_magenta (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.frame1, 'BackgroundColor', 'magenta')
set(handles.frame2, 'BackgroundColor', 'magenta')
set(handles.text_select, 'BackgroundColor', 'magenta')
set(handles.text_neighborhood, 'BackgroundColor', 'magenta')
set(handles.text_deviation, 'BackgroundColor', 'magenta')
set(handles.text_plotting, 'BackgroundColor', 'magenta')
set(handles.text_colormap, 'BackgroundColor', 'magenta')
set(handles.text_shading, 'BackgroundColor', 'magenta')
set(handles.text_message_welcome, 'BackgroundColor', 'magenta')
set(handles.text_message_apply, 'BackgroundColor', 'magenta')
% --------------------------------------------------------------------
function menu_colours_frames_pink_Callback(hObject, eventdata, handles)
% hObject    handle to menu_colours_frames_pink (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.frame1, 'BackgroundColor', [1.0,0.4,0.6])
set(handles.frame2, 'BackgroundColor', [1.0,0.4,0.6])
set(handles.text_select, 'BackgroundColor', [1.0,0.4,0.6])
set(handles.text_neighborhood, 'BackgroundColor', [1.0,0.4,0.6])
set(handles.text_deviation, 'BackgroundColor', [1.0,0.4,0.6])
set(handles.text_plotting, 'BackgroundColor', [1.0,0.4,0.6])
set(handles.text_colormap, 'BackgroundColor', [1.0,0.4,0.6])
```

```
set(handles.text_shading, 'BackgroundColor', [1.0,0.4,0.6])
set(handles.text_message_welcome, 'BackgroundColor', [1.0,0.4,0.6])
set(handles.text_message_apply, 'BackgroundColor', [1.0,0.4,0.6])
% --------------------------------------------------------------------
function menu_colours_frames_red_Callback(hObject, eventdata, handles)
% hObject    handle to menu_colours_frames_red (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.frame1, 'BackgroundColor', 'red')
set(handles.frame2, 'BackgroundColor', 'red')
set(handles.text_select, 'BackgroundColor', 'red')
set(handles.text_neighborhood, 'BackgroundColor', 'red')
set(handles.text_deviation, 'BackgroundColor', 'red')
set(handles.text_plotting, 'BackgroundColor', 'red')
set(handles.text_colormap, 'BackgroundColor', 'red')
set(handles.text_shading, 'BackgroundColor', 'red')
set(handles.text_message_welcome, 'BackgroundColor', 'red')
set(handles.text_message_apply, 'BackgroundColor', 'red')
% --------------------------------------------------------------------
function menu_colours_frames_teal_Callback(hObject, eventdata, handles)
% hObject    handle to menu_colours_frames_teal (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.frame1, 'BackgroundColor', [0.5,0.75,0.75])
set(handles.frame2, 'BackgroundColor', [0.5,0.75,0.75])
set(handles.text_select, 'BackgroundColor', [0.5,0.75,0.75])
set(handles.text_neighborhood, 'BackgroundColor', [0.5,0.75,0.75])
set(handles.text_deviation, 'BackgroundColor', [0.5,0.75,0.75])
set(handles.text_plotting, 'BackgroundColor', [0.5,0.75,0.75])
set(handles.text_colormap, 'BackgroundColor', [0.5,0.75,0.75])
set(handles.text_shading, 'BackgroundColor', [0.5,0.75,0.75])
set(handles.text_message_welcome, 'BackgroundColor', [0.5,0.75,0.75])
set(handles.text_message_apply, 'BackgroundColor', [0.5,0.75,0.75])
% --------------------------------------------------------------------
function menu_colours_frames_white_Callback(hObject, eventdata, handles)
% hObject    handle to menu_colours_frames_white (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.frame1, 'BackgroundColor', 'white')
set(handles.frame2, 'BackgroundColor', 'white')
set(handles.text_select, 'BackgroundColor', 'white')
set(handles.text_neighborhood, 'BackgroundColor', 'white')
set(handles.text_deviation, 'BackgroundColor', 'white')
set(handles.text_plotting, 'BackgroundColor', 'white')
set(handles.text_colormap, 'BackgroundColor', 'white')
set(handles.text_shading, 'BackgroundColor', 'white')
set(handles.text_message_welcome, 'BackgroundColor', 'white')
set(handles.text_message_apply, 'BackgroundColor', 'white')
```

```
% --------------------------------------------------------------------
function menu_colours_frames_yellow_Callback(hObject, eventdata, handles)
% hObject    handle to menu_colours_frames_yellow (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.frame1, 'BackgroundColor', 'yellow')
set(handles.frame2, 'BackgroundColor', 'yellow')
set(handles.text_select, 'BackgroundColor', 'yellow')
set(handles.text_neighborhood, 'BackgroundColor', 'yellow')
set(handles.text_deviation, 'BackgroundColor', 'yellow')
set(handles.text_plotting, 'BackgroundColor', 'yellow')
set(handles.text_colormap, 'BackgroundColor', 'yellow')
set(handles.text_shading, 'BackgroundColor', 'yellow')
set(handles.text_message_welcome, 'BackgroundColor', 'yellow')
set(handles.text_message_apply, 'BackgroundColor', 'yellow')
% --------------------------------------------------------------------
% --------------------------------------------------------------------
function menu_colours_pushbuttons_blue_Callback(hObject, eventdata, handles)
% hObject    handle to menu_colours_pushbuttons_blue (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.pushbutton_get, 'BackgroundColor', 'blue')
set(handles.pushbutton_help, 'BackgroundColor', 'blue')
set(handles.pushbutton_close, 'BackgroundColor', 'blue')
set(handles.pushbutton_apply, 'BackgroundColor', 'blue')
% --------------------------------------------------------------------
function menu_colours_pushbuttons_cyan_Callback(hObject, eventdata, handles)
% hObject    handle to menu_colours_pushbuttons_cyan (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.pushbutton_get, 'BackgroundColor', 'cyan')
set(handles.pushbutton_help, 'BackgroundColor', 'cyan')
set(handles.pushbutton_close, 'BackgroundColor', 'cyan')
set(handles.pushbutton_apply, 'BackgroundColor', 'cyan')
% --------------------------------------------------------------------
function menu_colours_pushbuttons_gray_Callback(hObject, eventdata, handles)
% hObject    handle to menu_colours_pushbuttons_gray (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.pushbutton_get, 'BackgroundColor', [0.5,0.5,0.5])
set(handles.pushbutton_help, 'BackgroundColor', [0.5,0.5,0.5])
set(handles.pushbutton_close, 'BackgroundColor', [0.5,0.5,0.5])
set(handles.pushbutton_apply, 'BackgroundColor', [0.5,0.5,0.5])
% --------------------------------------------------------------------
function menu_colours_pushbuttons_green_Callback(hObject, eventdata, handles)
% hObject    handle to menu_colours_pushbuttons_green (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
set(handles.pushbutton_get, 'BackgroundColor', 'green')
set(handles.pushbutton_help, 'BackgroundColor', 'green')
set(handles.pushbutton_close, 'BackgroundColor', 'green')
set(handles.pushbutton_apply, 'BackgroundColor', 'green')
% --------------------------------------------------------------------
function menu_colours_pushbuttons_lightgray_Callback(hObject, eventdata, ...
    handles)
% hObject    handle to menu_colours_pushbuttons_lightgray (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.pushbutton_get, 'BackgroundColor', [0.75,0.75,0.75])
set(handles.pushbutton_help, 'BackgroundColor', [0.75,0.75,0.75])
set(handles.pushbutton_close, 'BackgroundColor', [0.75,0.75,0.75])
set(handles.pushbutton_apply, 'BackgroundColor', [0.75,0.75,0.75])
% --------------------------------------------------------------------
function menu_colours_pushbuttons_magenta_Callback(hObject, eventdata, ...
    handles)
% hObject    handle to menu_colours_pushbuttons_magenta (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.pushbutton_get, 'BackgroundColor', 'magenta')
set(handles.pushbutton_help, 'BackgroundColor', 'magenta')
set(handles.pushbutton_close, 'BackgroundColor', 'magenta')
set(handles.pushbutton_apply, 'BackgroundColor', 'magenta')
% --------------------------------------------------------------------
function menu_colours_pushbuttons_pink_Callback(hObject, eventdata, handles)
% hObject    handle to menu_colours_pushbuttons_pink (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.pushbutton_get, 'BackgroundColor', [1.0,0.4,0.6])
set(handles.pushbutton_help, 'BackgroundColor', [1.0,0.4,0.6])
set(handles.pushbutton_close, 'BackgroundColor', [1.0,0.4,0.6])
set(handles.pushbutton_apply, 'BackgroundColor', [1.0,0.4,0.6])
% --------------------------------------------------------------------
function menu_colours_pushbuttons_red_Callback(hObject, eventdata, handles)
% hObject    handle to menu_colours_pushbuttons_red (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.pushbutton_get, 'BackgroundColor', 'red')
set(handles.pushbutton_help, 'BackgroundColor', 'red')
set(handles.pushbutton_close, 'BackgroundColor', 'red')
set(handles.pushbutton_apply, 'BackgroundColor', 'red')
% --------------------------------------------------------------------
function menu_colours_pushbuttons_teal_Callback(hObject, eventdata, handles)
% hObject    handle to menu_colours_pushbuttons_teal (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.pushbutton_get, 'BackgroundColor', [0.5,0.75,0.75])
```

```
set(handles.pushbutton_help, 'BackgroundColor', [0.5,0.75,0.75])
set(handles.pushbutton_close, 'BackgroundColor', [0.5,0.75,0.75])
set(handles.pushbutton_apply, 'BackgroundColor', [0.5,0.75,0.75])
% --------------------------------------------------------------------
function menu_colours_pushbuttons_white_Callback(hObject, eventdata, handles)
% hObject    handle to menu_colours_pushbuttons_white (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.pushbutton_get, 'BackgroundColor', 'white')
set(handles.pushbutton_help, 'BackgroundColor', 'white')
set(handles.pushbutton_close, 'BackgroundColor', 'white')
set(handles.pushbutton_apply, 'BackgroundColor', 'white')
% --------------------------------------------------------------------
function menu_colours_pushbuttons_yellow_Callback(hObject, eventdata, handles)
% hObject    handle to menu_colours_pushbuttons_yellow (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.pushbutton_get, 'BackgroundColor', 'yellow')
set(handles.pushbutton_help, 'BackgroundColor', 'yellow')
set(handles.pushbutton_close, 'BackgroundColor', 'yellow')
set(handles.pushbutton_apply, 'BackgroundColor', 'yellow')
% --------------------------------------------------------------------
% --------------------------------------------------------------------
function menu_colours_rawimagetext_black_Callback(hObject, eventdata, handles)
% hObject    handle to menu_colours_rawimagetext_black (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.text_raw, 'ForegroundColor', 'black')
% --------------------------------------------------------------------
function menu_colours_rawimagetext_blue_Callback(hObject, eventdata, handles)
% hObject    handle to menu_colours_rawimagetext_blue (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.text_raw, 'ForegroundColor', 'blue')
% --------------------------------------------------------------------
function menu_colours_rawimagetext_cyan_Callback(hObject, eventdata, handles)
% hObject    handle to menu_colours_rawimagetext_cyan (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.text_raw, 'ForegroundColor', 'cyan')
% --------------------------------------------------------------------
function menu_colours_rawimagetext_gray_Callback(hObject, eventdata, handles)
% hObject    handle to menu_colours_rawimagetext_gray (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.text_raw, 'ForegroundColor', [0.5,0.5,0.5])
% --------------------------------------------------------------------
function menu_colours_rawimagetext_green_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to menu_colours_rawimagetext_green (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.text_raw, 'ForegroundColor', 'green')
% --------------------------------------------------------------------
function menu_colours_rawimagetext_lightgray_Callback(hObject, eventdata, ...
    handles)
% hObject    handle to menu_colours_rawimagetext_lightgray (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.text_raw, 'ForegroundColor', [0.75,0.75,0.75])
% --------------------------------------------------------------------
function menu_colours_rawimagetext_magenta_Callback(hObject, eventdata, ...
    handles)
% hObject    handle to menu_colours_rawimagetext_magenta (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.text_raw, 'ForegroundColor', 'magenta')
% --------------------------------------------------------------------
function menu_colours_rawimagetext_pink_Callback(hObject, eventdata, handles)
% hObject    handle to menu_colours_rawimagetext_pink (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.text_raw, 'ForegroundColor', [1.0,0.4,0.6])
% --------------------------------------------------------------------
function menu_colours_rawimagetext_red_Callback(hObject, eventdata, handles)
% hObject    handle to menu_colours_rawimagetext_red (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.text_raw, 'ForegroundColor', 'red')
% --------------------------------------------------------------------
function menu_colours_rawimagetext_teal_Callback(hObject, eventdata, handles)
% hObject    handle to menu_colours_rawimagetext_teal (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.text_raw, 'ForegroundColor', [0.5,0.75,0.75])
% --------------------------------------------------------------------
function menu_colours_rawimagetext_white_Callback(hObject, eventdata, handles)
% hObject    handle to menu_colours_rawimagetext_white (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.text_raw, 'ForegroundColor', 'white')
% --------------------------------------------------------------------
function menu_colours_rawimagetext_yellow_Callback(hObject, eventdata, ...
    handles)
% hObject    handle to menu_colours_rawimagetext_yellow (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
set(handles.text_raw, 'ForegroundColor', 'yellow')
% --------------------------------------------------------------------
% --------------------------------------------------------------------
function menu_colours_filteredimagetext_black_Callback(hObject, eventdata, ...
    handles)
% hObject    handle to menu_colours_filteredimagetext_black (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.text_filtered, 'ForegroundColor', 'black')
% --------------------------------------------------------------------
function menu_colours_filteredimagetext_blue_Callback(hObject, eventdata, ...
    handles)
% hObject    handle to menu_colours_filteredimagetext_blue (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.text_filtered, 'ForegroundColor', 'blue')
% --------------------------------------------------------------------
function menu_colours_filteredimagetext_cyan_Callback(hObject, eventdata, ...
    handles)
% hObject    handle to menu_colours_filteredimagetext_cyan (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.text_filtered, 'ForegroundColor', 'cyan')
% --------------------------------------------------------------------
function menu_colours_filteredimagetext_gray_Callback(hObject, eventdata, ...
    handles)
% hObject    handle to menu_colours_filteredimagetext_gray (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.text_filtered, 'ForegroundColor', [0.5,0.5,0.5])
% --------------------------------------------------------------------
function menu_colours_filteredimagetext_green_Callback(hObject, eventdata, ...
    handles)
% hObject    handle to menu_colours_filteredimagetext_green (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.text_filtered, 'ForegroundColor', 'green')
% --------------------------------------------------------------------
function menu_colours_filteredimagetext_lightgray_Callback(hObject, ...
    eventdata, handles)
% hObject    handle to menu_colours_rawimagetext_lightgray (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.text_filtered, 'ForegroundColor', [0.75,0.75,0.75])
% --------------------------------------------------------------------
function menu_colours_filteredimagetext_magenta_Callback(hObject, ...
    eventdata, handles)
% hObject    handle to menu_colours_filteredimagetext_magenta (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.text_filtered, 'ForegroundColor', 'magenta')
% --------------------------------------------------------------------
function menu_colours_filteredimagetext_pink_Callback(hObject, eventdata, ...
    handles)
% hObject    handle to menu_colours_filteredimagetext_pink (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.text_filtered, 'ForegroundColor', [1.0,0.4,0.6])
% --------------------------------------------------------------------
function menu_colours_filteredimagetext_red_Callback(hObject, eventdata, ...
    handles)
% hObject    handle to menu_colours_filteredimagetext_red (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.text_filtered, 'ForegroundColor', 'red')
% --------------------------------------------------------------------
function menu_colours_filteredimagetext_teal_Callback(hObject, eventdata, ...
    handles)
% hObject    handle to menu_colours_filteredimagetext_teal (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.text_filtered, 'ForegroundColor', [0.5,0.75,0.75])
% --------------------------------------------------------------------
function menu_colours_filteredimagetext_white_Callback(hObject, eventdata, ...
    handles)
% hObject    handle to menu_colours_filteredimagetext_white (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.text_filtered, 'ForegroundColor', 'white')
% --------------------------------------------------------------------
function menu_colours_filteredimagetext_yellow_Callback(hObject, eventdata,...
    handles)
% hObject    handle to menu_colours_filteredimagetext_yellow (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.text_filtered, 'ForegroundColor', 'yellow')
% --------------------------------------------------------------------
% --------------------------------------------------------------------
function menu_workspace_Callback(hObject, eventdata, handles)
% hObject    handle to menu_workspace (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% --------------------------------------------------------------------
function menu_workspace_raw_Callback(hObject, eventdata, handles)
% hObject    handle to menu_workspace_raw (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)
% --------------------------------------------------------------------
function menu_workspace_average_Callback(hObject, eventdata, handles)
% hObject    handle to menu_workspace_average (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% --------------------------------------------------------------------
function menu_workspace_meanequalweights_Callback(hObject, eventdata, handles)
% hObject    handle to menu_workspace_meanequalweights (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% --------------------------------------------------------------------
function menu_workspace_meansinglepeak_Callback(hObject, eventdata, handles)
% hObject    handle to menu_workspace_meansinglepeak (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% --------------------------------------------------------------------
function menu_workspace_gaussian_Callback(hObject, eventdata, handles)
% hObject    handle to menu_workspace_gaussian (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% --------------------------------------------------------------------
function menu_workspace_median_Callback(hObject, eventdata, handles)
% hObject    handle to menu_workspace_median (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% --------------------------------------------------------------------
% --------------------------------------------------------------------
function menu_workspace_raw_save_Callback(hObject, eventdata, handles)
% hObject    handle to menu_workspace_raw_save (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Get the raw data array
raw_data = handles.current_data;
if raw_data == 0
    Warning_raw_mat;
else
    % If data was acquired but never saved, open the Save As dialog box
    if handles.LastFile_raw == 0
        menu_workspace_raw_saveas_Callback(hObject, eventdata, handles);
    else
        % Save to the default raw_data file
        File = handles.LastFile_raw;
        save(File,'raw_data')
    end
end
% --------------------------------------------------------------------
function menu_workspace_raw_saveas_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to menu_workspace_raw_saveas (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Get the raw data array
raw_data = handles.current_data;
if raw_data == 0
    Warning_raw_mat;
else
    % Allow the user to select the file name to save to
    [filename, pathname] = uiputfile({'*.mat';'*.*'},'Save As');
    % If 'Cancel' was selected then return
    if isequal([filename,pathname],[0,0])
        return
    else
        % Construct the full path and save
        File = fullfile(pathname,filename);
        save(File,'raw_data')
        handles.LastFile_raw = File;
        guidata(hObject,handles)
    end
end
% --------------------------------------------------------------------------
function menu_workspace_raw_import_Callback(hObject, eventdata, handles)
% hObject    handle to menu_workspace_raw_import (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Use UIGETFILE to allow for the selection of a custom file.
[filename, pathname] = uigetfile({'*.mat', 'All MAT-Files (*.mat)'; ...
        '*.*','All Files (*.*)'}, 'Select File');
% If "Cancel" is selected then return
if isequal([filename,pathname],[0,0])
    return
    % Otherwise construct the full filename and load the file.
else
    File = fullfile(pathname,filename);
    data = load(File);
    % Extract the field name of the loaded file
    field_name = fieldnames(data);
    % Compare the field_name with the known strings, which are used when
    % saving files in this application
    if strcmp(field_name,'raw_data')
        handles.current_data = data.raw_data;
    elseif strcmp(field_name,'average_filter')
        handles.current_data = data.average_filter;
    elseif strcmp(field_name,'mean_equal_weights_filter')
        handles.current_data = data.mean_equal_weights_filter;
    elseif strcmp(field_name,'mean_single_peak_filter')
```

```
        handles.current_data = data.mean_single_peak_filter;
    elseif strcmp(field_name,'gaussian_filter')
        handles.current_data = data.gaussian_filter;
    elseif strcmp(field_name,'median_filter')
        handles.current_data = data.median_filter;
    else
        Warning_file_not_valid;
        return
    end
    % Plot the raw image
    axes(handles.axes_raw)
    surfl(handles.current_data)
    colormap gray
    axis([1 16 1 16 0 256])
    % Plot the filtered image
    axes(handles.axes_filtered)
    nothing_in_filtered = (1000)*ones(256);
    surfl(nothing_in_filtered)
    axis([1 16 1 16 0 256])

    % Display the "Apply message" and erase the "Welcome message"
    set(handles.text_message_apply, 'Visible', 'on')
    set(handles.text_message_welcome, 'Visible', 'off')

    % Initialize Last File variables. Otherwise after Importing and
    % Saving, the last file is overwritten. If the user selects Save, it
    % should only allow to Save As in order to avoid overwriting the last
    % saved file. After the user saves the file once, and a new image is
    % not acquired or imported, then the user can just Save the image.
    handles.LastFile_raw = 0;
    handles.LastFile_average = 0;
    handles.LastFile_mean_e_w = 0;
    handles.LastFile_mean_s_p = 0;
    handles.LastFile_gaussian = 0;
    handles.LastFile_median = 0;
    handles.average_filter = 0;
    handles.mean_equal_weights_filter = 0;
    handles.mean_single_peak_filter = 0;
    handles.gaussian_filter = 0;
    handles.median_filter = 0;

    % Update handles structure
    guidata(hObject, handles);
end
% --------------------------------------------------------------------------
% --------------------------------------------------------------------------
function menu_workspace_average_save_Callback(hObject, eventdata, handles)
% hObject    handle to menu_workspace_average_save (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Get the average_filter data array
average_filter = handles.average_filter;
if average_filter == 0
    Warning_average_mat;
else
    % If data was acquired but never saved, open the Save As dialog box
    if handles.LastFile_average == 0
        menu_workspace_average_saveas_Callback(hObject, eventdata, handles);
    else
        % Save to the default average_filter file
        File = handles.LastFile_average;
        save(File,'average_filter')
    end
end
% --------------------------------------------------------------------------
function menu_workspace_average_saveas_Callback(hObject, eventdata, handles)
% hObject    handle to menu_workspace_average_saveas (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Get the average_filter data array
average_filter = handles.average_filter;
if average_filter == 0
    Warning_average_mat;
else
    % Allow the user to select the file name to save to
    [filename, pathname] = uiputfile({'*.mat';'*.*'},'Save As');
    % If 'Cancel' was selected then return
    if isequal([filename,pathname],[0,0])
        return
    else
        % Construct the full path and save
        File = fullfile(pathname,filename);
        save(File,'average_filter')
        handles.LastFile_average = File;
        guidata(hObject,handles)
    end
end
% --------------------------------------------------------------------------
% --------------------------------------------------------------------------
function menu_workspace_meanequalweights_save_Callback(hObject, eventdata, ...
    handles)
% hObject    handle to menu_workspace_meanequalweights_save (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Get the mean_equal_weights_filter data array
mean_equal_weights_filter = handles.mean_equal_weights_filter;
```

```
if mean_equal_weights_filter == 0
    Warning_mean_equal_weights_mat;
else
    % If data was acquired but never saved, open the Save As dialog box
    if handles.LastFile_mean_e_w == 0
        menu_workspace_meanequalweights_saveas_Callback(hObject,...
            eventdata, handles);
    else
        % Save to the default mean_equal_weights_filter file
        File = handles.LastFile_mean_e_w;
        save(File,'mean_equal_weights_filter')
    end
end
% --------------------------------------------------------------------------
function menu_workspace_meanequalweights_saveas_Callback(hObject, ...
    eventdata, handles)
% hObject    handle to menu_workspace_meanequalweights_saveas (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Get the mean_equal_weights_filter data array
mean_equal_weights_filter = handles.mean_equal_weights_filter;
if mean_equal_weights_filter == 0
    Warning_mean_equal_weights_mat;
else
    % Allow the user to select the file name to save to
    [filename, pathname] = uiputfile({'*.mat';'*.*'},'Save As');
    % If 'Cancel' was selected then return
    if isequal([filename,pathname],[0,0])
        return
    else
        % Construct the full path and save
        File = fullfile(pathname,filename);
        save(File,'mean_equal_weights_filter')
        handles.LastFile_mean_e_w = File;
        guidata(hObject,handles)
    end
end
% --------------------------------------------------------------------------
% --------------------------------------------------------------------------
function menu_workspace_meansinglepeak_save_Callback(hObject, eventdata, ...
    handles)
% hObject    handle to menu_workspace_meansinglepeak_save (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Get the mean_single_peak_filter data array
mean_single_peak_filter = handles.mean_single_peak_filter;
if mean_single_peak_filter == 0
    Warning_mean_single_peak_mat;
```

```
else
    % If data was acquired but never saved, open the Save As dialog box
    if handles.LastFile_mean_s_p == 0
        menu_workspace_meansinglepeak_saveas_Callback(hObject, eventdata, ...
            handles);
    else
        % Save to the default mean_single_peak_filter file
        File = handles.LastFile_mean_s_p;
        save(File,'mean_single_peak_filter')
    end
end
% --------------------------------------------------------------------------
function menu_workspace_meansinglepeak_saveas_Callback(hObject, eventdata, ...
    handles)
% hObject    handle to menu_workspace_meansinglepeak_saveas (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Get the mean_single_peak_filter data array
mean_single_peak_filter = handles.mean_single_peak_filter;
if mean_single_peak_filter == 0
    Warning_mean_single_peak_mat;
else
    % Allow the user to select the file name to save to
    [filename, pathname] = uiputfile({'*.mat';'*.*'},'Save As');
    % If 'Cancel' was selected then return
    if isequal([filename,pathname],[0,0])
        return
    else
        % Construct the full path and save
        File = fullfile(pathname,filename);
        save(File,'mean_single_peak_filter')
        handles.LastFile_mean_s_p = File;
        guidata(hObject,handles)
    end
end
% --------------------------------------------------------------------------
% --------------------------------------------------------------------------
function menu_workspace_gaussian_save_Callback(hObject, eventdata, handles)
% hObject    handle to menu_workspace_gaussian_save (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Get the gaussian_filter data array
gaussian_filter = handles.gaussian_filter;
if gaussian_filter == 0
    Warning_gaussian_mat;
else
    % If data was acquired but never saved, open the Save As dialog box
    if handles.LastFile_gaussian == 0
```

```
        menu_workspace_gaussian_saveas_Callback(hObject, eventdata, handles);
    else
        % Save to the default gaussian_filter file
        File = handles.LastFile_gaussian;
        save(File,'gaussian_filter')
    end
end
% --------------------------------------------------------------------------
function menu_workspace_gaussian_saveas_Callback(hObject, eventdata, handles)
% hObject    handle to menu_workspace_gaussian_saveas (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Get the gaussian_filter data array
gaussian_filter = handles.gaussian_filter;
if gaussian_filter == 0
    Warning_gaussian_mat;
else
    % Allow the user to select the file name to save to
    [filename, pathname] = uiputfile({'*.mat';'*.*'},'Save as');
    % If 'Cancel' was selected then return
    if isequal([filename,pathname],[0,0])
        return
    else
        % Construct the full path and save
        File = fullfile(pathname,filename);
        save(File,'gaussian_filter')
        handles.LastFile_gaussian = File;
        guidata(hObject,handles)
    end
end
% --------------------------------------------------------------------------
% --------------------------------------------------------------------------
function menu_workspace_median_save_Callback(hObject, eventdata, handles)
% hObject    handle to menu_workspace_median_save (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Get the median_filter data array
median_filter = handles.median_filter;
if median_filter == 0
    Warning_median_mat;
else
    % If data was acquired but never saved, open the Save As dialog box
    if handles.LastFile_median == 0
        menu_workspace_median_saveas_Callback(hObject, eventdata, handles);
    else
        % Save to the default median_filter file
        File = handles.LastFile_median;
        save(File,'median_filter')
```

```
    end
end
% --------------------------------------------------------------------
function menu_workspace_median_saveas_Callback(hObject, eventdata, handles)
% hObject    handle to menu_workspace_median_saveas (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Get the median_filter data array
median_filter = handles.median_filter;
if median_filter == 0
    Warning_median_mat;
else
    % Allow the user to select the file name to save to
    [filename, pathname] = uiputfile({'*.mat';'*.*'},'Save As');
    % If 'Cancel' was selected then return
    if isequal([filename,pathname],[0,0])
        return
    else
        % Construct the full path and save
        File = fullfile(pathname,filename);
        save(File,'median_filter')
        handles.LastFile_median = File;
        guidata(hObject,handles)
    end
end
% --------------------------------------------------------------------
% --------------------------------------------------------------------
function menu_export_Callback(hObject, eventdata, handles)
% hObject    handle to menu_export (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% --------------------------------------------------------------------
function menu_export_raw_Callback(hObject, eventdata, handles)
% hObject    handle to menu_export_raw (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Export the raw image
% Allow the user to select the file name to save to
% User MUST provide the extension of the file
[filename, pathname] = uiputfile({'*.jpg','JPEG images (*.jpg)'; ...
        '*.bmp','Bitmap files(*.bmp)'; ...
        '*.*','All Files (*.*)'},'Save Raw Image As');
% If 'Cancel' was selected then return
if isequal([filename,pathname],[0,0])
    return
else
    % Construct the full path and save
    File = fullfile(pathname,filename);
```

```
    Raw_frame = getframe(handles.figure1, [12 185 444 402]);
    imwrite(Raw_frame.cdata, File);
    % These two lines would be used only if we had a Save menu too, but in
    % the case of export we only have Save As
    % handles.LastFile_raw = File;
    % guidata(hObject,handles)
end
% --------------------------------------------------------------------------
function menu_export_filtered_Callback(hObject, eventdata, handles)
% hObject    handle to menu_export_filtered (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Export the filtered image
% Allow the user to select the file name to save to
% User MUST provide the extension of the file
[filename, pathname] = uiputfile({'*.jpg','JPEG images (*.jpg)'; ...
        '*.bmp','Bitmap files(*.bmp)'; ...
        '*.*','All Files (*.*)'},'Save Filtered Image As');
% If 'Cancel' was selected then return
if isequal([filename,pathname],[0,0])
    return
else
    % Construct the full path and save
    File = fullfile(pathname,filename);
    Filtered_frame = getframe(handles.figure1, [453 185 444 402]);
    imwrite(Filtered_frame.cdata, File);
end
% --------------------------------------------------------------------------
function menu_export_rawandfiltered_Callback(hObject, eventdata, handles)
% hObject    handle to menu_export_rawandfiltered (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Export the filtered image
% Allow the user to select the file name to save to
% User MUST provide the extension of the file
[filename, pathname] = uiputfile({'*.jpg','JPEG images (*.jpg)'; ...
        '*.bmp','Bitmap files(*.bmp)'; ...
        '*.*','All Files (*.*)'},'Save Raw and Filter Image As');
% If 'Cancel' was selected then return
if isequal([filename,pathname],[0,0])
    return
else
    % Construct the full path and save
    File = fullfile(pathname,filename);
    Raw_and_Filtered_frame = getframe(handles.figure1, [12 185 888 402]);
    imwrite(Raw_and_Filtered_frame.cdata, File);
end
% --------------------------------------------------------------------------
```

```
function menu_export_figure_Callback(hObject, eventdata, handles)
% hObject    handle to menu_export_figure (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Export the filtered image
% Allow the user to select the file name to save to
% User MUST provide the extension of the file
[filename, pathname] = uiputfile({'*.jpg','JPEG images (*.jpg)'; ...
        '*.bmp','Bitmap files(*.bmp)'; ...
        '*.*','All Files (*.*)'},'Save the Entire Figure As');
% If 'Cancel' was selected then return
if isequal([filename,pathname],[0,0])
    return
else
    % Construct the full path and save
    File = fullfile(pathname,filename);
    %Figure_frame = getframe(handles.figure1, [-2 -2 927 717]);
    Figure_frame = getframe(handles.figure1, [70 43 170 30]);
    imwrite(Figure_frame.cdata, File);
end
% --------------------------------------------------------------------
% --------------------------------------------------------------------
function menu_guihelp_Callback(hObject, eventdata, handles)
% hObject    handle to menu_guihelp (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% --------------------------------------------------------------------
function menu_guihelp_helpfile_Callback(hObject, eventdata, handles)
% hObject    handle to menu_guihelp_helpfile (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%open help_index
HelpPath = which('help_index.html');
web(HelpPath);
% --------------------------------------------------------------------
function menu_guihelp_readme_Callback(hObject, eventdata, handles)
% hObject    handle to menu_guihelp_readme (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
ReadMePath = which('ReadMe.html');
web(ReadMePath);
% --------------------------------------------------------------------
function menu_guihelp_about_Callback(hObject, eventdata, handles)
% hObject    handle to menu_guihelp_about (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
About;
% --------------------------------------------------------------------
```

```
% --------------------------------------------------------------------
% --- Executes on mouse press over figure background.
function figure1_ButtonDownFcn(hObject, eventdata, handles)
% hObject    handle to figure1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% --------------------------------------------------------------------
% --------------------------------------------------------------------
```