Design of Automated Digital Eye Palpation Exam for Intraocular Pressure Measurement

ALEXANDER VALLEJO LUCE

A Thesis Submitted to The Honors College
In Partial Fulfillment of the Bachelors degree
With Honors in

Engineering Physics

THE UNIVERSITY OF ARIZONA

May 2009

Approved by:

_____
Dr. Eniko Enikov
Department of Aerospace and Mechanical Engineering

Statement by Author

I hereby grant to the University of Arizona Library the nonexclusive worldwide right to reproduce and distribute my thesis and abstract (herein, the licensed materials), in whole or in part, in any and all media of distribution and in any format in existence now or developed in the future. I represent and warrant to the University of Arizona that the licensed materials are my original work, that I am the sole owner of all rights in and to the licensed materials, and that none of the licensed materials infringe or violate the rights of others. I further represent that I have obtained all necessary rights to permit the University of Arizona Library to reproduce and distribute any nonpublic third party software necessary to access, display, run, or print my thesis. I acknowledge that University of Arizona Library may elect not to distribute my thesis in digital format if, in its reasonable judgment, it believes all such rights have not been secured.

Signed:

To my family, thank you for your support.

Preface

I would like to thank Dr. Eniko Enikov for giving me the opportunity to work on such an outstanding project and by providing the opportunity to travel to Switzerland and extend the project beyond what I thought was possible. I would like to further thank Prof. Dr. Bradley Nelson, my host at ETH Zurich, for allowing me to work on his lab and supporting me during my time in Switzerland.

This work would not have been possible without the contributions of a number of outstanding individuals. At the University of Arizona, support from Ed White and Dr. Kai Deng helped during the initial stages of the project. Vasco Polyzoev's lab expertise and experience with microelectronics has been indispensable. Peter Polyvas has helped to take over the experimental aspect of the project since returning to Arizona and is working to take the experiment into the next phase.

At ETH Zurich, great support was provided by the entire crew at the Institute of Robotics and Intelligent Systems. I would like to further thank Kamran Shamai, Chauncey Graetzel, Michael Kummer, and Rudi Borer for their support and guidance during my time working in their lab.

Abstract

Elevated intraocular pressure (IOP) is a major risk factor for the degenerative eye disease glaucoma. Accurate indirect measurements of IOP are essential for glaucoma diagnosis and screening. This work presents an experiment developed to measure IOP in-vitro by simulating the technique of digital palpitation tonometry, a technique in which a trained examiner palpates the eyeball using the fingertips of both index fingers to feel the stiffness of the eye. The qualitative nature of this method and errors introduced by the subjectivity of the examiner mean that it is rarely used in comparison with other modern-day tonometry methods. However, this technique offers several potential advantages in that it can be performed outside of a clinical setting without the need for instrument sterilization or local anesthesia and may be less subject to measurement errors occurring in patients who have undergone refractive laser eye surgery.

In order to quantify the mechanics of digital palpation tonometry, an automated experiment to measure the intraocular pressure of enucleated porcine eyeballs using mechanized digital palpation was designed and tested. This experiment has direct applications towards the development of a next-generation tonometer for glaucoma treatment.

Contents

List of Tables

List of Figures

1 Introduction

1.1 Glaucoma and Intraocular Pressure

The degenerative disease glaucoma is the world's leading cause of preventable blindness. This ocular disease is characterized by progressive loss of the visual field and stems from optic nerve damage due to a build up of intraocular pressure (IOP) in the eye [47]. It is estimated that primary glaucoma will cause bilateral blindness in over 8.4 million people worldwide in 2010, with this number increasing to 11.1 million by 2020. The number of people with glaucoma is projected to reach 79.6 million people worldwide in 2020 [38].

Elevated intraocular pressure (IOP) of the eyeball is the major risk factor for the disease [46]. IOPs between 10 and 21 mm Hg are considered normal, with a mean of 16 mm Hg. The risk of developing glaucoma has been shown to increase tenfold if IOP is greater than 23 mmHg as compared with people with IOP less than 16 mmHg [46] . Current glaucoma treatment is based on attempts to reduce the intraocular pressure and slow the progression of the disease. Therefore, accurate methods to determine IOP are crucial for early detection, screening, and treatment of glaucoma.

1.2 Methods to measure IOP

The current accepted standard method for IOP measurement is the Goldman applanation tonometer [12] . This method measures the force necessary to flatten the cornea to predetermined contact area with a diameter of approximately 3 mm. Greater forces will be required for higher values of IOP. The most important characteristic of the Goldman method is that measurements are done virtually free of the influence of schleral rigidity [34].

An alternative tonometry method in common use is MacKay-Marg tonometry, a type of noncontact tonometer, which measures the resulting deformation of the cornea from a puff of compressed air. Other methods include transpalpebral tonometry and digital palpitation tonometry. Unfortunately, both of these methods typically under-predict the IOP with significant clinical error[49].

The current diagnosis techniques of Glaucoma are cumbersome and must typically be done in a clinical setting. However, there are are several drawbacks to

these techniques, one of which is do to diurnal variations in the IOP on the order of 3-10 mm Hg [11]. Consequently, the ability to measure IOP multiple times during a 24-hr time period is desirable. Further drawbacks of the standard techniques is that they generally require topical anesthesia and there is a risk of infection associated with placing an object in contact with the cornea. Furthermore, it has been shown the Goldman tonometry method is not accurate for individuals who have undergone refractive laser eye surgery due to structural changes in the cornea [35].

1.3 Digital Palpation Tonometry

Digital palpation tonometry, the oldest method of rough IOP estimation, is a qualitative technique in which a trained examiner lightly presses, or palpates, the eye with the fingertips of both index fingers. A small force is thus applied in alternating fashion on the upper part of the eye, through the eyelid, and the examiner is able to feel the flexibility of the sclera, gauge its tension, and deduce the IOP to within a margin of error. Currently, palpation is the regarded as being the simplest, least expensive, and least accurate method of measuring determining IOP [24]. Training sessions can improve the ability of an inexperienced examiner in measuring IOP. An experienced examiner is able to estimate IOP to within 5 mm Hg in an interoperative setting 100% of the time, and estimate the correct values 46% of the time. This accuracy is enough for estimating IOP in intraoperative settings when judging the tightness of scleral sutures during surgery [5].

The human digital palpation exam is not new, having been first reported in 1862 by Bowman [24]. It served as the standard technique to measure IOP until the advent of the Schiotz tonometer in 1905 [5]. Today, measurement of IOP by digital palpation is only used infrequently due to inherent inaccuracies in palpating the globe through the eyelid, and the advent of more reliable measurement techniques that provide optometrists with quantitative data. Although palpation is an imperfect technique it still plays a limited role in screening for considerable increases in intraocular pressure, especially in cases in which other measurement devices are not readily available or sterile [4].

1.4 measurement of IOP through palpation

A sensor based on the concept of digital palpation tonometry could address many of the shortcomings present in current tonometers. Desired characteristics are portability, ease of use, no requirements for topical anesthesia, and insensitivity due to changes in corneal structure resulting from laser eye surgery.

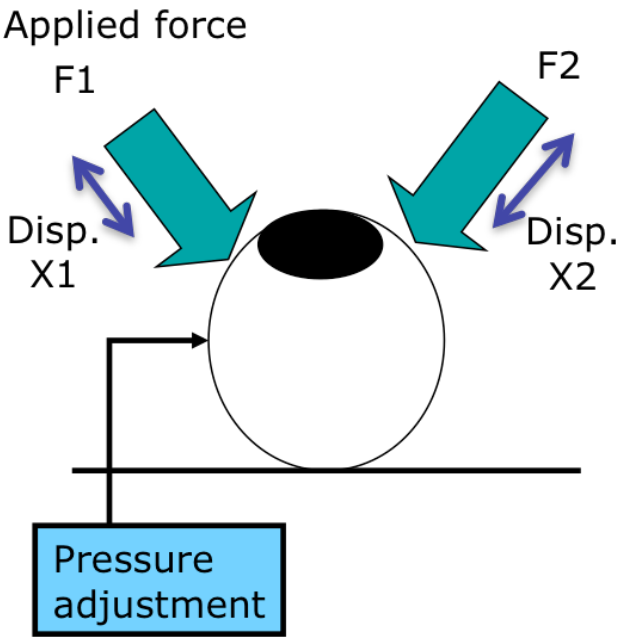


Figure 1: The concept of digital palpation tonometry. A trained examiner uses both index finger to gauge the IOP by palpating the sclera. In the simulation experiment, forces from 2 sensors are measured at a given displacement as function of IOP.

In order to develop such a sensor, an in-depth study on the mechanics of the human digital palpation exam is desired. We propose to simulate the technique of digital palpation tonometry using a mechatronic sensor setup. In the simplest sense, the digital palpation exam may be thought of as 2 degree of freedom system (each index finger representing 1 experimental degree of freedom) where a force $\mathbf{F}$ is applied the eyeball. Figure 1 presents a diagram of the proposed digital palpation measurement modality. The displacement of the sclera at the point of applanation will depend on the IOP. In this sense, each finger acts as a tactile sensor, measuring the normal surface force as a function of the relative displacement from a reference point on the eyeball. In the palpation examination,

patients are typically asked to look down with closed eyes. The redundant skin of the upper eyelid is moved with the examiner's index fingers and downward pressure is applied with each index finger alternately to the globe of the eye at the 12 O'clock meridian [4]. This measurement procedure could be significantly improved by augmenting the examiner's sensory capabilities with a system providing quantifiable data. Thus, we propose to use a tactile sensor, measuring both force and spatial information to replicate this technique.

2 Methods and Materials

2.1 Pressure Regulation

Manometry is commonly regarded as an invasive technique that can precisely measure the pressure inside the eye and is a common laboratory technique for a evaluating changes in IOP over time and for providing reference pressure by which all other tonometers can be judged [24]. For the purposes of this experiment, a pressure regulation system was constructed to regulate the IOP of an enucleated porcine during measurement. Porcine eyes have been shown to be suitable replacements for human eyes in glaucoma related studies[43]. The system consisted of a fluid column with a plastic syringe on top connected via PVC tubing to a three way valve. This attached to a commercial pressure sensor on one end (OMEGA), and a thin walled cannula on the other end. The syringe was attached to a standard test tube stand and was adjustable in height such that it was used to regulate the hydrostatic pressure at the bottom of the system. The pressure of the system was measured in units of mm Hg and determined by the height of the syringe [6].

2.2 Force Sensing

Tests of the human digital palpation exam were conducted and the magnitude of force applied to the eye was found to be in the range of approximately 0 to 2 N. To measure this force, we choose to use bending beam load cells (FUTEK, LBB200) with a 4.5N capacity. The load cells operate using a piezoresistive strain gauge sensor with a full bridge mounted on stainless steel cantilevers with a nominal deflection of 0.28 mm. The point of contact with the eye was a hemispherical plastic dome, referred to as the 'indenter,' with radius of curvature 1.0 cm. Its geometry was chosen to approximate the shape of a human fingertip. One indenter was mounted on the tip of each cantilever force sensor.

2.3 Data Acquisition

A custom data acquisition board (DAQ) was developed for the purposes of this experiment. Instrumentation amplifiers (INA101, Texas Instruments) were used to amplify the voltage difference from the force sensor strain gauges and the

signal from the pressure sensor. 10-bit analog to digital conversion was achieved via a programmable flash microcontroller (PIC16F684, Microchip). This was interfaced to a personal computer via the RS-232 connection.

2.4 Manual Actuation

In order to displace the force sensors along the axis of palpation, a linear translation stage was used. In the first experimental setup (hereafter referred to as the manual actuation experiment), the force sensors were attached with a rigid right angle bracket to an optical linear translation stage driven by a micrometer (Edmund Optics, NT38-958). Figure 2 presents a picture of the manual actuation experimental setup.

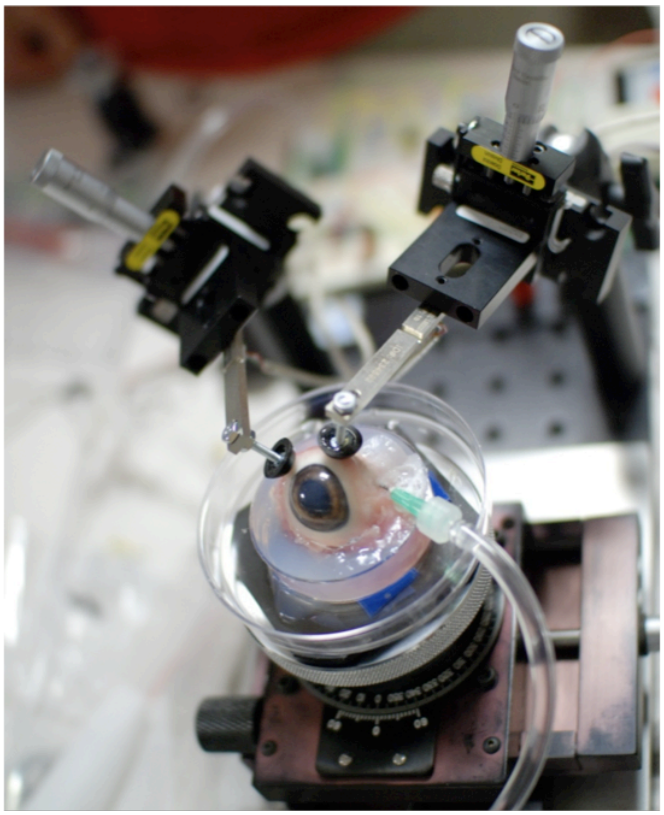


Figure 2: Manual actuation setup.

2.5 Mechatronic Sensor

The human digital palpation exam is typically performed using two index fingers, each free to move independently. In order to mimic this, it was desired to have an experiment able to apply force to the eyeball in a number of time-varying fashions and to have control over a wide range of independent force sensor displacements. Thus a chief design requirement was to motorize the linear translation stages and apply a computer based position control system so that the experiment would be automated.

The optical linear translation stage from the manual actuation experiment was replaced with a stage consisting of linear actuators (Nanotec GMBH, LP3575). The linear actuators, driven by stepper motors, were able to provide a linear resolution of 0.0254 mm per step, making them ideal for high precision displacement measurements. Figure 3 presents a picture of the mechatronic eye palpation experimental setup.

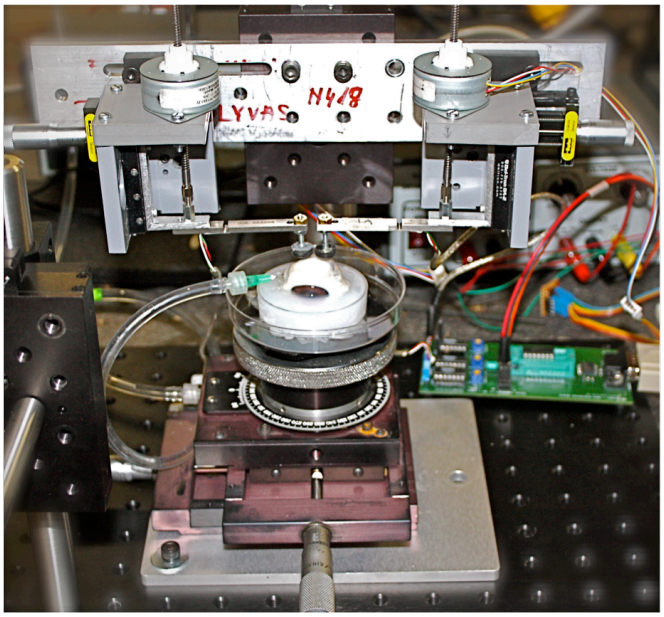


Figure 3: Mechatronic eye palpation experiment.

2.6 Translational Control

Control of the stepper motor linear actuators was realized through a personal computer using MATLAB via an open loop setup. A stepper motor controller

using a programmable flash-based microcontroller (PIC16F684, Microchip) provided the stepping signals. This controller was interfaced to the computer via the parallel port. A custom MATLAB program was used to control the motion of the motors.

2.7 Computer Control

A graphical user interface (GUI) was developed using the MATLAB programming environment. This program allowed a user having little programming knowledge to operate and control the mechatronic digital palpation device with an intuitive point and click user interface. The GUI was developed using the MATLAB GUI design environment (GUIDE). The GUI integrates several MATLAB subfunctions for data analysis and control of the experiment. The main purpose of the program was twofold: it gathered and stored data from the Data Acquisition System, and it provided an interface for open loop control of the linear actuators. Several subroutines were developed to control the linear actuators in a specific fashion and utilized feedback from the force sensors to position the sensors in a desired fashion. These subroutines are described in detail in the *experimental procedure* section. Figure 4 presents showcases the GUI interface. The MATLAB code is found in the appendix section.

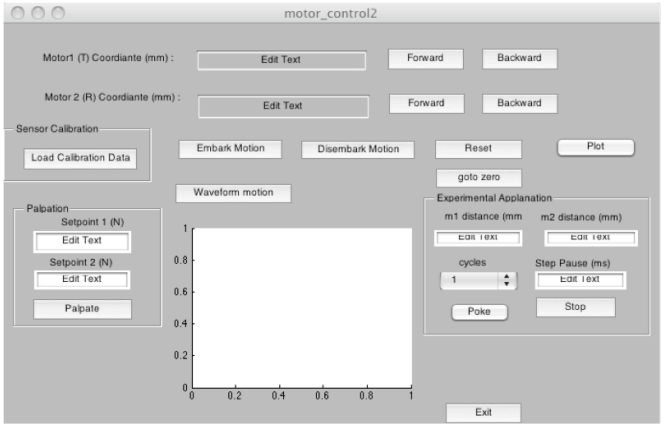


Figure 4: The MATLAB GUI interface.

2.8 Overview of Experimental Setup

Figure 5, adapted from [29], presents an overview of the manual actuation experiment. An agarose gel solution (1) acted as a socket to anchor the eyeball (2) in a Petri Dish (3). IOP was regulated by changing the height of the saline column (4) and connected to the eye with PVC tubing (5). A 3-way valve (6) was used to seal off the eye during measurements, and connected to a pressure sensor (7). Each force sensor was attached to an L bracket (8) and mounted on an articulating arm (9) that allows for positioning of the force sensors (10). The displacement of the applanation tip was varied with a micrometer-actuated optical linear translation stage (11), situated between the L-bracket and the end of the articulating arm.

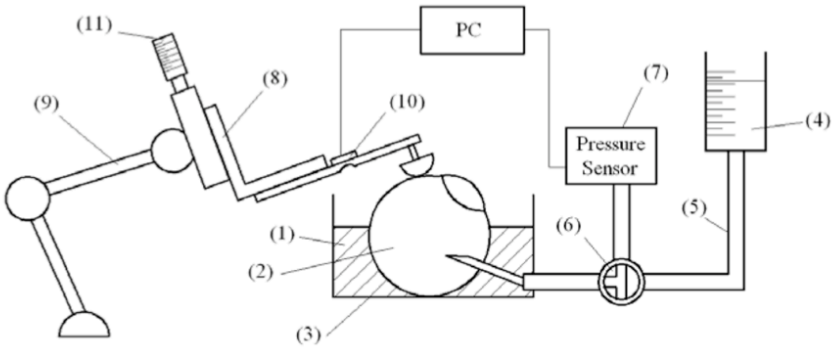


Figure 5: Schematic of manual actuation experiment.

3 Experimental Procedure

3.1 Setup

The setup procedure for testing eyeballs *in vitro* is similar to that described in Halberg, et al. [18]. Enucleated porcine eyeballs were kept in refrigerated storage until ready for measurements and excess tissue was removed with a knife. The eyeballs were mounted firmly in a petri dish containing an agar (Ag) gel solution [Ag] = 30.0 g/l, as shown in Figure 6. The solution was allowed to solidify and a thin walled cannula diameter 0.89 mm, length 25.4 mm was inserted through the side of the eyeball with the tip located approximately in the middle of the vitreous chamber. A cyanoacrylate adhesive was used to seal the interface between the sclera and the cannula. The cannula was connected to the pressure regulation system consisting of an adjustable height saline column via PVC tubing and a three-way valve.

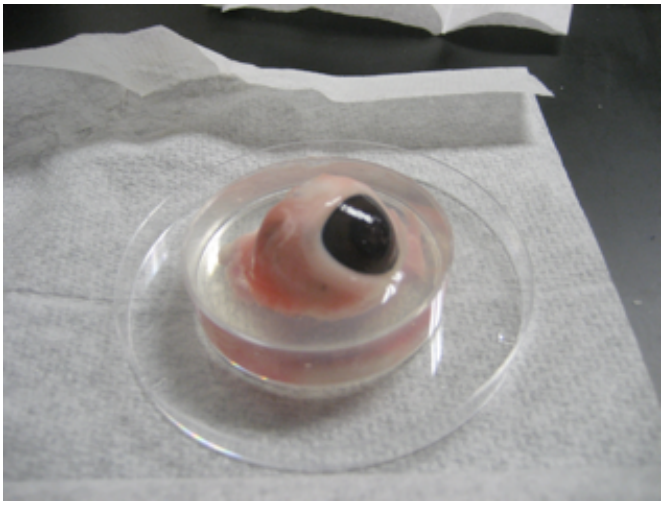


Figure 6: Detail of porcine eyeball fixed in the agar gel socket.

The pressure in the eye was adjusted by varying the height of the saline column with the valve open. The pressure level was calculated from the measured height of the saline column and allowed to equilibrate for approximately 10 seconds. The valve was then closed, allowing the eye to form a closed system at the desired pressure. During the experiment, the pressure of the eyeball was monitored with the pressure sensor. In between measurements, the eye was periodically moistened with a saline solution to keep from drying out.

The force sensors were calibrated for each eye trial. The cantilevers were oriented such that the applied force was normal to the surface and weights of known mass were placed on the cantilever tip. The linearity was checked and recorded and the sensors were later zeroed after being positioned in the desired measurement configuration relative to the eyeball.

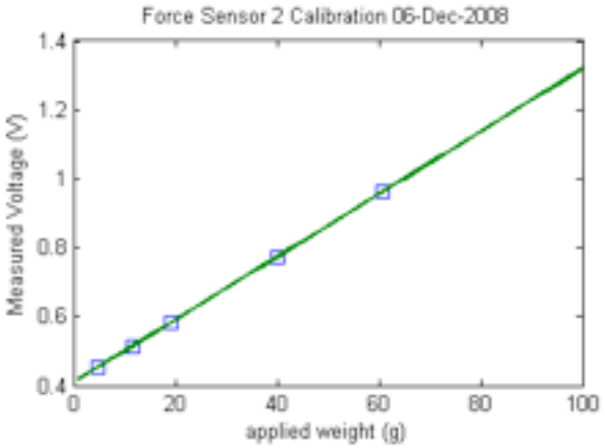


Figure 7: Calibration curve for bending beam load cell force sensor. The voltage offset was zeroed when the sensors were positioned at 0 mm displacement relative to the eye.

3.2 Manual Actuation Measurements

Eye pressure measurements were performed for IOP's in the range of 10 to 35 mm Hg in increments of 5 mm Hg. These values were chosen based on ISO standards which specify the minimum measurement requirements for tonometers intended to estimate IOP in a routine clinical setting [18]. One of the requirements (ISO 8612:2001 and 15004:1997) specifies accuracy greater than $\pm$ 5 mmHg and the ability to distinguish between IOPs of IOP $<$ 16 mm Hg, 16mm Hg $<$ IOP $<$ 23 mm Hg, and IOP $>$ 23 mm Hg. The height of the saline column was used to set the desired reference pressure during experiments. Table 1 presents the height of the water column relative to the syringe for the desired IOP values.

The palpation of the eye was performed over the sclera along the center axis of the eye at a point approximately 45 degrees from the center of the cornea. The sensors were positioned such that the hemispherical plastic indenter was just out of contact with the eyeball and the force sensors were zeroed. The micrometer

Table 1: Height of Water Column.

IOP [mm Hg]	**Height [cm]**
10	13.6
15	20.4
20	27.2
25	34
30	40.8
35	47.6

was then incremented in 0.635 mm steps; at each step the normal force of the cantilever was measured. This was continued until the desired force of 3.0 N was reached. The indenter was then removed from the eyeball and the IOP was adjusted to the next desired value.

3.3 Mechatronic Measurements

During development of the experiment at ETH, the positioning procedure for testing the mechatronic sensors was the same as described above. The eyes were also prepared in the same fashion as described in the *setup* section. Using the MATLAB GUI, several different modes of measurement were developed and described in the following sub-sections.

3.3.1 Experimental Applanation function

The Experimental Applanation function was designed for the user to control the displacement of the indenters and measure the response force. A desired displacement could be set for each indenter and the program would palpate the eye until this postion was reach and return to the starting position. This could be repeated for a preset number of cycles from 1 to 5.

3.3.2 Waveform motion function

The Waveform motion function allows the user to input the desired position waveform of the indenters into a MATLAB array. The program will load this array and position the indenters according to the values stored in the array. Thus it is possible to palpate the eye and position the indenters in any user-specified combination.

3.3.3 Palpation function

The Palpation function was developed to mimic the procedure of the clinical digital eye palpation exam. A more in-depth explanation of this measurement modality is found in the *Results and Discussion* section. It uses a feedback loop to position the first indenter to a desired force setpoint, termed *Force setpoint 1*. After the force sensor of the first indenter has reached the desired force setpoint, the second intender is driven forward against the eye until a second desired setpoint is reached, *Force setpoint 2*. After the second force sensor measures reaches *Force setpoint 2*, the second indenter was removed from the eye to return to the initial position, and finally the first intender was driven backwards to the initial position.

4 Results and Discussion

There have been few previous studies on the deformation of the sclera as a function of IOP. Due to the nonlinear biomechanical properties of the sclera, developing an analytical model for the displacement of the sclera as a function of pressure during an external indentation is difficult. However, finite element simulations may be used as a reasonable approximation [29].

4.1 Manual Actuation Setup

Measurements performed using a single sensor in isolation show a nonlinear relation between force and displacement for a given IOP. These data are represented in Figure 8. As IOP was increased from the normal range of 10 to 15 mm Hg (green line), past the glaucoma suspicion threshold of approximately 20mm Hg (red line) to an upper limit of 35 to 40 mm Hg, the slope of the force vs. displacement curves increased.

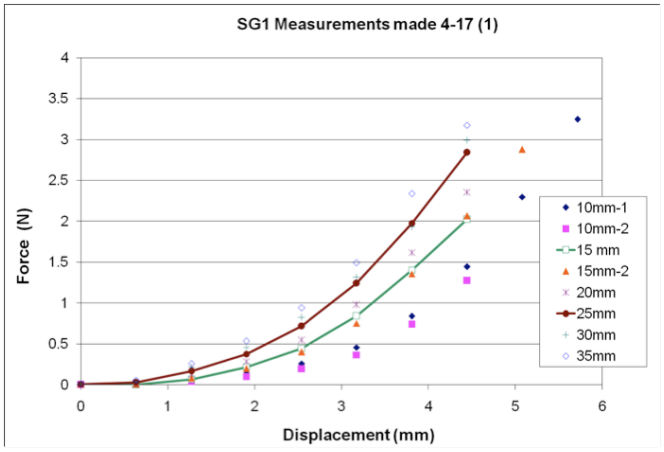


Figure 8: Force vs. displacement during the 1 degree of freedom manual palpation test.

4.1.1 Comparison with finite element model

A finite element model of the digital palpation exam was developed in the University of Arizona's Advanced Micro and Nanosystems lab [29]. This model allowed computer simulations of the 1 degree of freedom eye applanation to be conducted

and compared to the experimental results. Figure 9 shows the comparison of simulation results with experimental results for for the force vs. displacement data. Three different finite element simulations were done accounting for differences in the schleral wall thickness and changes in the stiffness of the eye support. The finite element model was found to be in agreement with the experimental data at small displacements less than approximately 2.5 mm.

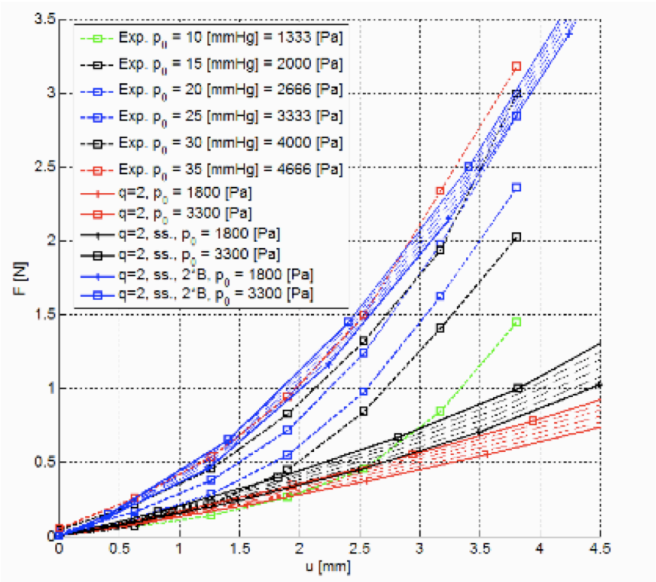


Figure 9: Comparison of experimental results with finite element model simulation results.

4.2 Mechatronic Setup

The mechatronic digital palpation exam was initially tested using conditions similar to the manual actuation experiment in order to verify the functionality of the experiment. To begin with, each force sensor was independently applanated against the eyeball. The results of the single force sensor applantion experiment using the mechatronic setup are shown in Figure 10. These curves agree nicely with those obtained while using the manual actuation experiment although it should be noted that the pig eyeballs obtained at ETH were of a different species of pig than those used at UA. This could account for differences in the exact fit of the curves and the magnitude of the measured force as a function of displacement.

Subsequently, it was desired to test the repeatability of the mechatronic exam

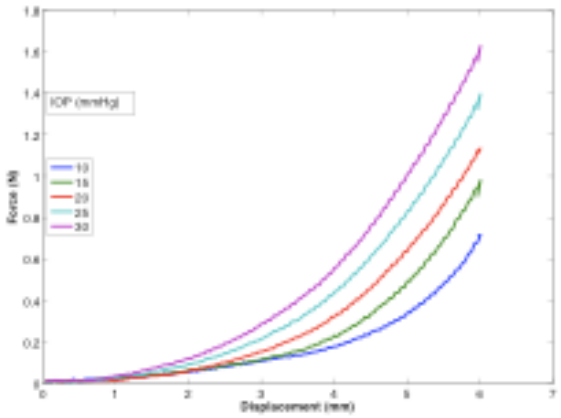


Figure 10: Force vs. displacement for the 1 degree of freedom mechatronic test.

for given eye and IOP. Thus at a given IOP, each eyeball was palpated 5 times in a cyclical fashion. It was found that the measured force at a given displacement differed during the forward and backward motion, as evidenced by the hysteresis in the force versus displacement curves shown in Figure 11. This hysteresis appears in subsequent force vs. displacement measurements of the eye conducted at both ETH and UA. The exact cause for this remains to be determined.

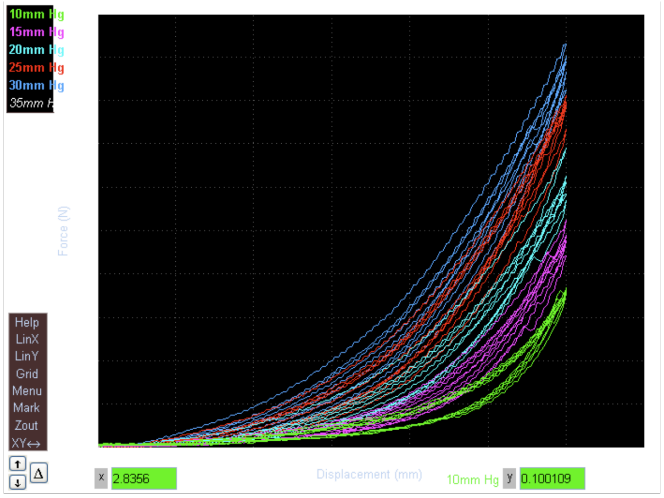
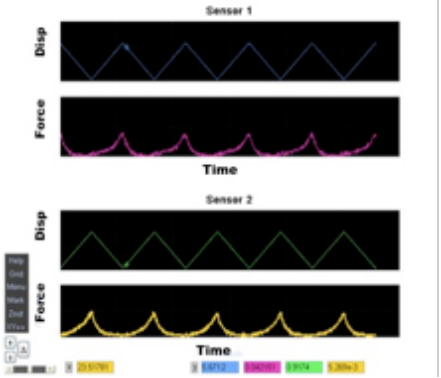


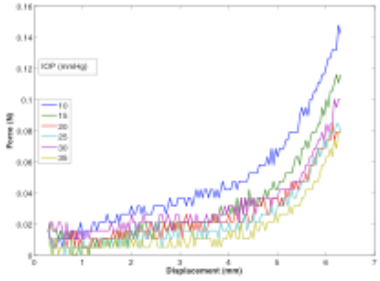
Figure 11: Force vs. displacement during a cyclical test.

As a closer approximation of the human palpation exam, the mechatronic sensor was tested using two degrees of experimental freedom. Initially, the force was applied to the eyeball in an alternating fashion: when one indenter was fully

displaced against the eye, the other indenter was not touching the eye. The sensors were then actuated such that the fully displaced indenter was retracted and the other indenter was moved into full displacement against the eye. This cycle was repeated 5 times for a given IOP. Figure 12 a) shows the motion of each indenter vs. time and the force measurements vs. time for the case of IOP = 10 mm Hg. The resulting force vs. displacement curves are shown in Figure 12 b). In contrast to the previous measurements, the slope of the force vs. displacement curves decreased with increasing IOP. As the IOP is increased, it would appear as though less force is required to deform the eyeball by a given amount. However, this analysis does not take into account the effect of the motion of the 2nd indenter, which clearly plays a significant role in the measurement outcome.



(a) Force and displacement vs. time for each sensor during the 2 degree of freedom mechatronic palpation test for the case IOP = 10 mm Hg.



(b) Resulting force vs. displacement curves for IOP = 10 mm Hg to 35 mm Hg.

Figure 12: Alternating palpation experiment results

4.3 Simulation of digital palpation

In an actual digital palpation exam the examiner makes decisions on how much force to apply to the eyeball with the applanating finger based on how hard the eyeball feels, essentially an indirect measure of the force at the fingertip. The sense of human touch is quite complicated in actuality, but for the purposes of these IOP measurements, a reasonable first approximation was desired. Unlike

previous measurement modalities, the *Palpation* function of the MATLAB program introduces a simple control loop to adjust the displacements of the indenters with respect to the eyeball using feedback measured by the force sensors.

During the palpation measurement, both sensors initially started not in contact with the eyeball. The first sensor was then incremented to press against the eye until it measured a certain force *Force setpoint 1*. Once this force value was reached, the motion of indenter 1 was stopped, indenter 2 was then incremented against the eyeball until it reached a certain force *Force setpoint 2*. During this time, indenter 1 is still slightly pressing on the eye, but left at a fixed position. During the entire experiment values of the measured forces and displacements are recorded.

In the proposed tonometer application, measurements of the IOP are desired to be independent of measurements of relative displacement. Thus, an important way to characterize the results of the experiment are to track the forces measured by each force sensor during course of the palpation experiment. Figure 13 presents the result of a palpation experiment conducted at ETH. It is interesting to note that an approximately linear relation appears to exist between Force 1 and Force 2. What is more significant, however, is that this linear relation changes as a function of IOP. In this particular experiment, increasing the IOP increases the slope of the Force 1 vs. Force 2 relation.

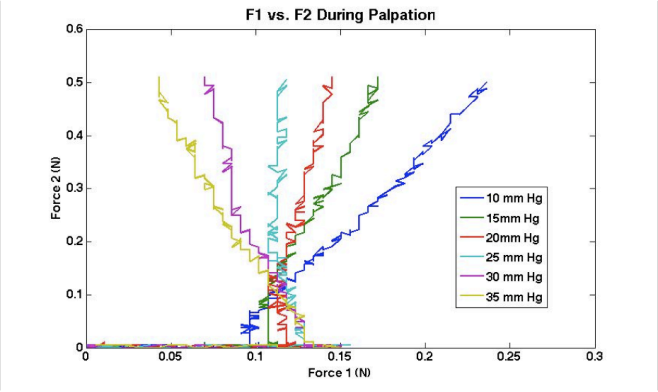


Figure 13: Forces measured by each indenter during the palpation simulation experiment performed at ETH.

In subsequent experiments, it was desired to measure force and pressure as a function of indenter displacement while the indenters while the motion is reversed and the indenters are removed from the eyeball. Thus, after *Force setpoint 2* has

been reached, indenter 2 is driven backwards and removed from the eye until it returns to its initial position. After this, indenter 1 is removed from the eye in similar fashion.

5 Conclusion

In this work an experiment designed to simulate the technique of digital palpation tonometry was developed. Tests were carried out using enucleated porcine eyeballs pressurized in the range of 10 to 35 mm Hg. Measurements of force and displacement with a single indenter could distinguish IOP to within 5 mm hg. This was done initially with the mechanical actuation setup and repeated with the mechatronic experiment. One important consideration is that the eyes were fixed by the agar gel socket in both situations and measurements of the absolute displacement of the sclera from its initial position may have inherent inaccuracies due to small displacements of the eyeball from its equilibrium position. Another consideration is that measuring the absolute displacement of the sclera during an indentation would be very difficult to achieve in practice, such as when the exam is being performed on a human patient. This is due to motion of the eye in the eye socket, the ability of the patient to move his head during the experiment, and any effects introduced by the nonlinear biomechanical properties of the patient's eyeball.

6 Summary and Contributions

A novel biosensor for measurements of intraocular pressure (IOP) on *in vitro* eyeballs has been designed and tested. Initially, a mechanically actuated setup was constructed at the Advanced Micro and Nanosystems Lab, University of Arizona to prove the experimental concept. The eye palpation exam was then automated with the addition of mechatronic control and computer feedback at the Institute for Robotics and Intelligent Systems, ETH Zurich. Upon returning to the University of Arizona, the experiment was retrofitted with a positioning system which allowed precise placement of the indenters relative to each other.

Experiments have been conducted to measure the IOP based on the technique of simulated digital palpation tonometry. Tests using enucleated porcine eyeballs demonstrate the ability to measure IOP in the range 10 to 35 mm Hg to an accuracy of 5mm Hg. Future work will continue the testing and development of the mechatronic sensor. The ability of the sensor to conduct independent 2 degree-of-freedom measurements has not been fully tested experimentally. Furthermore, the addition of a control feedback system into the mechatronic sensor may be a step closer to simulating and providing a platform to quantify the human digital palpation exam. This measurement modality has the potential to be applied to a new type of IOP sensor, providing a means to facilitate easier treatment and diagnosis of the degenerative eye disease glaucoma.

6.1 Future Work

Future work will address the dynamics and analysis of simultaneous force sensor measurements. The ultimate goal is to develop a new type of tonometer using the technique of digital palpation tonometry. An important future research direction is to demonstrate the accurate assessment of IOP on *in vitro* porcine eyes before human trials can begin. The measurement modality developed in this work proposes using force sensor feedback in positioning the indenters to avoid using measurements of relative displacements. Future efforts, however, may explore a way to accurately measure the displacement of indenter relative to the eyeball. Doing so could provide a different means to extract IOP data.

Another area to address is the effect of the eyelid on the measurement results. In this work, the indenters were placed into direct contact with the porcine eyeball

and no attempt was made to simulate the effect that the eyelid would cause during the measurement. The part of the eyelid covering the human eye has a thickness of 2 mm, is composed of many layers of cells, and is elastic. In a proposal for a through-the-eyelid tonometer, the author recommends that measurements of IOP be done with a suitable technique such that the results are independent of eyelid properties [9].

6.2 Contributions

The author would also like to acknowledge the support of NSF grant CBET-0603198.

References

[1] *Importance of intraocular pressure in glaucoma*, volume 3591, 1999.

[2] A. Baldi, W. Choi, and B. Ziaie. A self-resonant frequency-modulated micromachined passive pressure transensor. *Sensors Journal, IEEE*, 3(6):728 – 733, Dec 2003.

[3] E. BARANY. Simultaneous measurement of changing intraocular pressure and outflow facility in the vervet monkey .... *Investigative ophthalmology & visual science*, Jan 1964.

[4] N. J. Baum. Assessment of intraocular pressure by palpation. *American journal of ophthalmology*, 119(5), 1995.

[5] C. Birnbach. Digital palpation of intraocular pressure. *Ophthalmic surgery and lasers*, 29(9):754–7, 1998. UA ILL/Express Documents.

[6] F. Block, G. Schulte, T. Morris, and M. Ramsey. The "torrstick": A ruler calibrated in millimeters of mercury for measurement of hydrostatic .... *Journal of Clinical Monitoring and Computing*, Jan 1992.

[7] J. Brandt. Central corneal thickness, tonometry, and glaucoma risk-a guide for the perplexed. *Can J Ophthalmol*, Jan 2007.

[8] R. Brubaker. The effect of intraocular pressure on conventional outflow resistance in the enucleated human eye. *Investigative ophthalmology & visual science*, Jan 1975.

[9] F. Chiu. An exploration of a through-the-eyelid intraocular pressure measurement device. *dspace.mit.edu*, Jan 2007.

[10] J. Coosemans, M. Catrysse, and R. Puers. A readout circuit for an intraocular pressure sensor. *Sensors & Actuators: A. Physical*, Jan 2004.

[11] G. Davis. Diurnal variation of intraocular pressure in the normal eye. *Archives of Ophthalmology*, 69:752–757, 1963.

[12] H. Davson and H. Davson. *The Intraocular Pressure*, volume 1a. 1984.

[13] P. Dekker, Y. Robert, H. Kanngiesser, and P. Pirani. Principles of contact lens tonometry. *International Ophthalmology*, Jan 1998.

[14] D. Eisenberg, B. Sherman, C. McKeown, and J. Schuman. Tonometry in adults and children: a manometric evaluation of pneumatonometry, applanation, and .... *Ophthalmology(Rochester*, Jan 1998.

[15] A. Eklund, T. Backlund, and O. Lindahl. A resonator sensor for measurement of intraocular pressure-evaluation in an in vitro pig-eye model. *Physiol Meas*, Jan 2000.

[16] A. Eklund, P. Hallberg, C. Linden, and O. Lindahl. An applanation resonator sensor for measuring intraocular pressure using combined continuous force .... *Investigative ophthalmology & visual science*, Jan 2003.

[17] R. Feldman. Ocular palpation in pseudophakia. *American journal of ophthalmology*, 104(3):307, 1987.

[18] P. Hallberg. *Applanation Resonance Tonometry for Intraocular Pressure Measurement*. PhD thesis, 2006.

[19] P. Hallberg, A. Eklund, T. Backlund, and C. Linden. Clinical evaluation of applanation resonance tonometry - a comparison with goldmann applanation tonometry. *J Glaucoma*, 16(1):88–93, Jan 2007.

[20] P. Hallberg, A. Eklund, K. Santala, and T. Koskela. Underestimation of intraocular pressure after photorefractive keratectomy: a biomechanical analysis. *Medical and Biological Engineering and Computing*, Jan 2006.

[21] P. Hallberg, C. Lindén, T. Bäcklund, and A. Eklund. Symmetric sensor for applanation resonance tomometry of the eye. *Medical and Biological Engineering and Computing*, Jan 2006.

[22] P. Hallberg, C. Linden, O. Lindahl, T. Backlund, and A. Eklund. Applanation resonance tonometry for intraocular pressure in humans, Jan 2004.

[23] M. Johnson and R. Kamm. The role of schlemm's canal in aqueous outflow from the human eye. *Investigative ophthalmology & visual science*, Jan 1983.

[24] C. Kniestedt, O. Punjabi, S. Lin, and R. Stamper. Tonometry through the ages. *Survey of Ophthalmology*, Jan 2008.

[25] T. Kutoglu, B. Yalcin, N. Kocabiyik, and H. Ozan. Vortex veins: Anatomic investigations on human eyes. *Clinical Anatomy*, Jan 2005.

[26] N. O. LAWRENCE. Flow rate measurment of in situ perfusion through the aqueous humor outflow network. *MASTER OF SCIENCE IN BIOMEDICAL ENGINEERING*, 2006.

[27] S. Macha and A. Mitra. Ocular pharmacokinetics in rabbits using a novel dual probe microdialysis technique. *Experimental eye research*, Jan 2001.

[28] R. Mackay. Electronics in clinical research. *Proceedings of the IRE*, 50(5):1177 – 1189, May 1962.

[29] B. Magyar, D. E. T. Enikov, and D. G. Stépán. *Mechanical Analysis of Digital Palpation Tonometry of the Eye*. PhD thesis, 2008.

[30] S. Martin, M. Butler, J. Spates, W. K. Schubert, and M. Mitchell. Magnetically-excited flexural plate wave resonator. *Ultrasonics, Ferroelectrics and Frequency Control, IEEE Transactions on*, 45(5):1381 – 1387, Sep 1998.

[31] C. McMonnies. Management of chronic habits of abnormal eye rubbing. *Contact Lens and Anterior Eye*, Jan 2008.

[32] C. McMonnies and G. Boneham. Corneal curvature stability with increased intraocular pressure. *Eye & Contact Lens: Science and Clinical Practice*, Jan 2007.

[33] C. McMonnies and G. Boneham. Experimentally increased intraocular pressure using digital forces. *Eye & Contact Lens: Science and Clinical Practice*, Jan 2007.

[34] R. Moses. The goldmann applanation tonometer. *American journal of ophthalmology*, 46(6):865, 1958.

[35] R. Munger, A. Dohadwala, W. Hodge, W. Jackson, G. Mintsioulis, and K. Damji. Changes in measured intraocular pressure after hyperopic photorefractive keratectomy. *Journal of Cataract and Refractive Surgery*, 27:1254–1262, 2001. [1] doi:10.1016/S0886-3350(01)00971-3.

[36] T. Pan, M. Stay, V. Barocas, J. Brown, and B. Ziaie. Modeling and characterization of a valved glaucoma drainage device with implications for enhanced therapeutic efficacy. *IEEE Transactions on Biomedical Engineering*, Jan 2005.

[37] B. Pierscionek, M. Asejczyk-Widlicka, and R. Schachar. The effect of changing intraocular pressure on the corneal and scleral curvatures in the fresh .... *British Medical Journal*, Jan 2007.

[38] H. Quigley and A. Broman. The number of people with glaucoma worldwide in 2010 and 2020. *British Journal of Ophthalmology*, Jan 2006.

[39] R. Rizq, W. Choi, D. Eilers, M. Wright, and B. Ziaie. Intraocular pressure measurement at the choroid surface: a feasibility study with implications for .... *British Medical Journal*, Jan 2001.

[40] D. Robinson, D. O'meara, A. Scott, and C. Collins. Mechanical components of human eye movements. *Journal of Applied Physiology*, Jan 1969.

[41] R. Rubinfeld. The accuracy of finger tension for estimating intraocular pressure after penetrating keratoplasty. *Ophthalmic surgery and lasers*, 29(3):213–5, 1998.

[42] D. Rudnick, J. Noonan, D. Geroski, and M. Prausnitz. The effect of intraocular pressure on human and rabbit scleral permeability. *Investigative ophthalmology & visual science*, Jan 1999.

[43] J. Ruiz-Ederra, M. García, M. Hernández, and H. Urcola. The pig eye as a novel model of glaucoma. *Experimental eye research*, Jan 2005.

[44] D. Sandner, A. Böhm, S. Kostov, and L. Pillunat. Measurement of the intraocular pressure with the "transpalpebral tonometer" tgdc-01 in comparison with applanation tonometry. *Graefe's Arch Clin Exp Ophthalmol*, 243(6):563–569, 2005. 10.1007/s00417-004-1037-1.

[45] S. Shah. Accurate intraocular pressure measurement—the myth of modern ophthalmology? *Ophthalmology*, Jan 2000.

[46] A. Sommer. Intraocular pressure and glaucoma. *American journal of ophthalmology*, 107(2):186–8, 1989.

[47] A. Sommer. Glaucoma screening. *Journal of General Internal Medicine*, 5(0):S33–S37, 1990. 10.1007/BF02600838.

[48] W. Śródka and D. Iskander. Optically inspired biomechanical model of the human eyeball. *Journal of Biomedical Optics*, Jan 2008.

[49] A. Troost, S. Yun, K. Specht, F. Krummenauer, and O. Schwenn. Transpalpebral tonometry: reliability and comparison with goldmann applanation tonometry and palpation in healthy volunteers. *Br J Ophthalmol*, 89(3):280–283, Mar 2005.

[50] R. H. Tufflas and M. Anliker. Dynamic behavior of eye globes. *PhD Thesis*, 1967.

[51] M. Ulieru, O. Cuzzani, S. Rubin, and M. Ceruti. Application of soft computing methods to the diagnosis andprediction of glaucoma. *2000 IEEE International Conference on Systems*, Jan 2000.

[52] A. Vaajanen, H. Vapaatalo, and O. Oksala. A modified in vitro method for aqueous humor outflow studies in enucleated porcine eyes. *Journal of Ocular Pharmacology and Therapeutics*, Jan 2007.

[53] K. D. Voorhies. Static and dynamic stress/strain properties for human and porcine eyes. *Master Thesis*, 2003.

[54] J. Wagner, A. Edwards, and J. Schuman. Characterization of uveoscleral outflow in enucleated porcine eyes perfused under constant pressure. *Investigative ophthalmology & visual science*, Jan 2004.

[55] S. Woo, A. Kobayashi, C. Lawrence, and W. Schlegel. Mathematical model of the corneo-scleral shell as applied to intraocular pressure-volume relations and applanation tonometry. *Annals of Biomedical Engineering*, 1(1):87–98, 1972. 10.1007/BF02363420.

A Appendix

The following is the MATLAB source code for the GUI

motor_control2.m

```
% --------------------------------------------------------------------
% Written by Alex Luce
% Institute of Robotics and Intelligent Systems, ETH Zurich
% email: lucea@student.ethz.ch or aluce@email.arizona.edu
% Created on: 08.10.2008
% Current Version: 0.3.3
% Last updated: 4.15.2009
%
%Partial Version History
%
%Version 0.3.3 - added data collection for reverse motors on
%'test_palpation' function
%               -replaced 'get_DAQ' with 'get_DAQ2'
%               -added Data_resolution function dropdown textbox and
%               functionality to 'test_palpation' function
%Version 0.3.2 -added save coordinate functinality to 'disembark' function
%               -.0417 changed to .0254 in various functions
%               -added 'file_name' functionality  on 3.11.2009
%Version 0.3.1 -added palpation function
%               -added sensor calibration function
%               -grouped 'experimental applanation functions'
%Version 0.2.2: -'select motor button group removed'
%              -added 'alternate poke' function
%               -motor resolution constant changed to 0.0254 mm/step for LP3575
%Version 0.2.1 -added Waveform motion functions
%Version 0.1.2 -Added Experimental Applanation functions
%Version 0.1 -First version, allowed indedpendent forward and reverse motor
%control
%
% --------------------------------------------------------------------

function varargout = motor_control2(varargin)
% MOTOR_CONTROL2 M-file for motor_control2.fig
%      MOTOR_CONTROL2, by itself, creates a new MOTOR_CONTROL2 or raises the existing
%      singleton*.
%
%      H = MOTOR_CONTROL2 returns the handle to a new MOTOR_CONTROL2 or the handle to
%      the existing singleton*.
%
%      MOTOR_CONTROL2('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in MOTOR_CONTROL2.M with the given input arguments.
%
%      MOTOR_CONTROL2('Property','Value',...) creates a new MOTOR_CONTROL2 or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before motor_control2_OpeningFunction gets called.  An
%      unrecognized property name or invalid value makes property application
```

```
%      stop.  All inputs are passed to motor_control2_OpeningFcn via varargin.
%
%      *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 motor_control2

% Last Modified by GUIDE v2.5 20-Apr-2009 15:38:28

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @motor_control2_OpeningFcn, ...
                   'gui_OutputFcn',  @motor_control2_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

end

% --- Executes just before motor_control2 is made visible.
function motor_control2_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   command line arguments to motor_control2 (see VARARGIN)

% Choose default command line output for motor_control2
handles.output = hObject;

%set(handles.select_motor_buttongroup,'SelectionChangeFcn',@select_motor_buttongroup_SelectionChangeFcn);

% Update handles structure
guidata(hObject, handles);
    %default values of variables
    global poke_cycles m1_applanation_distance;
    poke_cycles=1;
    m1_applanation_distance=1;
```

```
% UIWAIT makes motor_control2 wait for user response (see UIRESUME)
% uiwait(handles.figure1);
imaqreset;
set(handles.motor1_coordinate_value,'String','Press Embark');                          % exporting the Trans coordin
set(handles.motor2_coordinate_value,'String','Press Embark');                          % exporting the Trans coordin
set(handles.pause_time,'String',0);
set(handles.m1_applanation_dist,'String','enter dist.');

end

% --- Outputs from this function are returned to the command line.
function varargout = motor_control2_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;

end
%**********************************************************************

% --- Executes during object creation, after setting all properties.
function motor1_coordinate_value_CreateFcn(hObject, eventdata, handles)
% hObject    handle to motor1_coordinate_value (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



end

% ***************** Motor 1 Coordinate ******************
function motor1_coordinate_value_Callback(hObject, eventdata, handles)
% hObject    handle to motor1_coordinate_value (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of motor1_coordinate_value as text
%        str2double(get(hObject,'String')) returns contents of motor1_coordinate_value as a double

global Z;  %value of motor1 in number of steps
global delay;
```

```
global dio;

z=str2double(get(hObject,'String'))/(0.0254); %divide by 0.0254 convert from number of steps to mm
if (z>Z)                                        % if the disgnated coordinate is more
    while (z>Z)
        Z=forward2(Z,delay,dio);
        set(handles.motor1_coordinate_value,'String',...
            num2str(Z*0.0254));                             % exporting motor1 coordinate to display
    end
elseif (z<Z)
    while (z<Z)                                    % if the disgnated coordinate is less
        Z=backward2(Z,delay,dio);
        set(handles.motor1_coordinate_value,'String',...
            num2str(Z*0.0254));                             % exporting motor1 coordinate to display
    end

end

end
% --- Executes during object creation, after setting all properties.

function motor2_coordinate_value_CreateFcn(hObject, eventdata, handles)
% hObject    handle to motor2_coordinate_value (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

end

% ***************** Motor2 Coordiante ******************
function motor2_coordinate_value_Callback(hObject, eventdata, handles)
% hObject    handle to motor2_coordinate_value (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of motor2_coordinate_value as text
%        str2double(get(hObject,'String')) returns contents of motor2_coordinate_value as a double
global TETA; %value of motor2 in number of steps
global delay;
global dio;


teta=str2double(get(hObject,'String'))/(0.0254);
if (teta>TETA)                                      % if the disgnated coordinate is more
    while (teta>TETA)
```

```
        TETA=CW(TETA,delay,dio);
        set(handles.motor2_coordinate_value,'String',...
            num2str(TETA*0.0254));                              % exporting motor2 coordinate to display
    end
elseif (teta<TETA)
    while (teta<TETA)                                       % if the disgnated coordinate is less
        TETA=CCW(TETA,delay,dio);
        set(handles.motor2_coordinate_value,'String',...
            num2str(TETA*0.0254));                              % exporting motor2 coordinate to display
    end

end

end

% *****************M1 Forward ******************
% --- Executes on button press in forward_button.
function forward_button_Callback(hObject, eventdata, handles)
% hObject    handle to forward_button (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global Z;
global delay;
global dio;
%setpoint=Z_check(dio);
%if setpoint(2)=1
Z=forward2(Z,delay,dio);

set(handles.motor1_coordinate_value,'String',...
    num2str(Z*0.0254));                             % exporting the Trans coordinate


end

% *****************M2 Backward ******************
% --- Executes on button press in backward_button.
function backward_button_Callback(hObject, eventdata, handles)
% hObject    handle to backward_button (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global Z;
global delay;
global dio;
Z=backward2(Z,delay,dio);
set(handles.motor1_coordinate_value,'String',...
    num2str(Z*0.0254));                             % exporting the Trans coordinate

end

% ***************** M2_Forward **********************
% --- Executes on button press in M2_F_button.
function M2_F_button_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to M2_F_button (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global TETA;
global delay;
global dio;
TETA=CCW(TETA,delay,dio);
set(handles.motor2_coordinate_value,'String',...
    num2str(TETA*0.0254));                          % exporting the Trans coordinate
end

% ****************** M2_Backwards ********************
% --- Executes on button press in M2_B_button.
function M2_B_button_Callback(hObject, eventdata, handles)
% hObject    handle to M2_B_button (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global TETA;
global delay;
global dio;
TETA=CW(TETA,delay,dio);
set(handles.motor2_coordinate_value,'String',...
    num2str(TETA*0.0254));                          % exporting the Trans coordinate
end

% ***************** Embark Motion******************
% --- Executes on button press in Embark_press.
%This function initializes the motors
function Embark_press_Callback(hObject, eventdata, handles)
% hObject    handle to Embark_press (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%
%clear all

global Z TETA Data_resolution_value;
fid = fopen('status.dat', 'r');
Z=str2num(fgetl(fid))/0.0254;
TETA=str2num(fgetl(fid))/0.0254;
fclose(fid);
Data_resolution_value=1;

set(handles.motor1_coordinate_value,'String',num2str(Z*0.0254));  % exporting the Trans coordinate

set(handles.motor2_coordinate_value,'String',num2str(TETA*0.0254));   % exporting the Trans coordinate

% creating the Digital I/O object
global dio;
dio = digitalio('parallel','LPT1');

addline(dio,0:5,'out');       % Adding Line to the object
                                                    % LPT1:0,1,2 output
```

```
                                                    % LPT1:3,4,5   input
global delay; %this is the delay value for the stepper motor (utilizes MATLAB pause function
delay=0.00000;


%initialize DAQ
out=instrfind;
delete(out);

%clear all

global S;
S = serial('COM1');   %Serial port configuration
set(S,'BaudRate',19200);
set(S,'InputBufferSize',6);
fopen(S)   %Open serial port


end

% ***************** Disembark Motion******************
% --- Executes on button press in disembark_press.
%This function disables the motors
function disembark_press_Callback(hObject, eventdata, handles)
% hObject    handle to disembark_press (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.motor1_coordinate_value,'String','Press Embark');                          % exporting the Trans coordin
set(handles.motor2_coordinate_value,'String','Press Embark');                          % exporting the Trans coordin
global dio Z TETA;
delete(dio);
clear dio;

fid = fopen('status.dat', 'wt');
fprintf(fid, '%2.4f\n', Z*0.0254);
fprintf(fid, '%2.4f\n', TETA*0.0254);
fclose(fid);

end

% ***************** exit_press******************
% --- Executes on button press in exit_press.
%Close the program and relevant ports
function exit_press_Callback(hObject, eventdata, handles)
% hObject    handle to exit_press (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global  dio Z TETA S poke_cycles;
   %poke_cycles=1;

fid = fopen('status.dat', 'wt');
fprintf(fid, '%2.4f\n', Z*0.0254);
```

```
fprintf(fid, '%2.4f\n', TETA*0.0254);
fclose(fid);

%fclose(S)  %Close serial port
delete(S)
clear S

delete(dio);
clear dio;

%close all;
close;

end

% --- Executes on button press in reset_press.
%resets M1 and M2 coordinate value to 0
function reset_press_Callback(hObject, eventdata, handles)
% hObject    handle to reset_press (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

global Z TETA delay dio;
z=1/0.0254;

%    while (z<Z)                                    % if the disgnated coordinate is less
%        Z=backward2(Z,delay,dio);
%        set(handles.motor1_coordinate_value,'String',...
%            num2str(Z*0.0254));                          % exporting the Trans coordinate
%    end

Z=0;
set(handles.motor1_coordinate_value,'String',...
    num2str(Z*0.0254));
TETA=0;
set(handles.motor2_coordinate_value,'String',num2str(TETA*0.0254));

end

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

global Z TETA S delay dio m1_applanation_distance m2_applanation_distance poke_cycles file_name;
%initialize the flag variable
set(handles.poke,'UserData',1);
mypause=str2double(get(handles.pause_time,'String'));

i=0;
j=0;
```

```
tic
%while the flag variable is one, the loop continues
    while ((j<poke_cycles) && (get(handles.poke,'UserData') ==1))
        j=j+1;
        m1_target=Z+m1_applanation_distance;
        m2_target=TETA+m2_applanation_distance;
                                    % if the disgnated coordinate is more

    while (Z<m1_target || TETA<m2_target)
        i=i+1;
        if(Z<m1_target)
            Z=forward2(Z,delay,dio);
            set(handles.motor1_coordinate_value,'String',...
            num2str(Z*0.0254));                          % exporting the Trans coordinate

        end
        if(TETA<m2_target)
            TETA=CW(TETA,delay,dio);
            set(handles.motor2_coordinate_value,'String',...
            num2str(TETA*0.0254));
        end

        [V1(i),V2(i),V3(i)] = get_DAQ2(S);
        t(i)=toc;
        m1(i)=Z*.0254;
        m2(i)=TETA*.0254;
        pause(mypause);
    end

    m1_target=Z-m1_applanation_distance;
    m2_target=TETA-m2_applanation_distance;
    while (Z>m1_target  || TETA>m2_target)% if the disgnated coordinate is less
        i=i+1;
        if(Z>m1_target )
            Z=backward2(Z,delay,dio);
            set(handles.motor1_coordinate_value,'String',...
              num2str(Z*0.0254));                        % exporting the Trans coordinate

        end
        if(TETA>m2_target)
            TETA=CCW(TETA,delay,dio);
             set(handles.motor2_coordinate_value,'String',...
             num2str(TETA*0.0254));
        end
        [V1(i),V2(i),V3(i)] = get_DAQ2(S);
        t(i)=toc;
        m1(i)=Z*.0254;
        m2(i)=TETA*.0254;
        pause(mypause);
    end

    end %poke_cycles loop
```

```
         %save output to file
        fid = fopen('data_temp.dat','w');
        fprintf(fid,'');
        fprintf(fid,'%g %g %g %6.4f %6.4f %6.4f\n',t,m1,m2,V1,V2,V3);

        date=datestr(now, 'dd.mm.yyyy');
        mkdir('data',date);
        cd('data');
        cd(date);
       % save ('datafile','t', 'm1', 'm2' ,'V1' ,'V2' ,'V3');
       save (file_name,'t', 'm1', 'm2' ,'V1' ,'V2' ,'V3');
        cd('..');
        cd('..');

        %additional plotting
        create_plots;

    set(handles.motor1_coordinate_value,'String',...
    num2str(Z*0.0254));                          % exporting the Trans coordinate


end

function m1_applanation_dist_Callback(hObject, eventdata, handles)
% hObject    handle to m1_applanation_dist (see GCBO)
%set the applanation distance from GUI textbox
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of m1_applanation_dist as text
%        str2double(get(hObject,'String')) returns contents of m1_applanation_dist as a double

global m1_applanation_distance;
m1_applanation_distance=str2double(get(hObject,'String'))/(0.0254);

end

% --- Executes during object creation, after setting all properties.
function m1_applanation_dist_CreateFcn(hObject, eventdata, handles)
% hObject    handle to m1_applanation_dist (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

end
```

```
function Embark_press_ButtonDownFcn(hObject, eventdata, handles)
% hObject    handle to Embark_press (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

end

function disembark_press_ButtonDownFcn(hObject, eventdata, handles)
% hObject    handle to disembark_press (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
end


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

global Z TETA;

i=0;
%figure, grid, xlabel ('Time (s)'), ylabel('Voltage (V)');


%axes(handles.axes10)
%selects axes1 as the current axes, so that
%Matlab knows where to plot the data
axes(handles.axes10)

%creates a vector from 0 to 10, [0 1 2 3 . . . 10]
x = 0:10;
%creates a vector from 0 to 10, [0 1 2 3 . . . 10]
y = 0:10;

%plots the x and y data
%plot(Z,y);
%adds a title, x-axis description, and y-axis description
title('Axes 1');
xlabel('X data');
ylabel('Y data');

%axis ([0 30 0 5])
hold on
tic
while i < 10
    i=i+1;
     t(i)=toc;

         plot(t,Z*0.0254,'r');
      %   plot(t,TETA*0.0254,'b');
```

```
      %  drawnow;
end
%hold off
guidata(hObject, handles); %updates the handles
end

function select_motor_buttongroup_SelectionChangeFcn(hObject, eventdata)
%selects which motor will be called by the 'poke' function
%retrieve GUI data, i.e. the handles structure
handles = guidata(hObject);

switch get(eventdata.NewValue,'Tag')   % Get Tag of selected object
    case 'motor1_select'

      %execute this code when fontsize08_radiobutton is selected
      %set(handles.display_staticText,'motor_1',1);
      set(handles.poke,'String', 'poke m1')
    case 'motor2_select'
      %execute this code when fontsize12_radiobutton is selected
      set(handles.poke,'String', 'poke m2')

    otherwise
       % Code for when there is no match.

end
%updates the handles structure
guidata(hObject, handles);

end

% --- Executes on selection change in poke_number.
function poke_number_Callback(hObject, eventdata, handles)
% hObject    handle to poke_number (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 poke_number contents as cell array
%        contents{get(hObject,'Value')} returns selected item from poke_number

%gets the selected option
global poke_cycles;
        poke_cycles=1;
switch get(handles.poke_number,'Value')
    case 1
        poke_cycles=1;
    case 2
        poke_cycles=2;
    case 3
        poke_cycles=3;
    case 4
        poke_cycles=4;
```

```
    case 5
        poke_cycles=5;
    case 6
        poke_cycles=0;
    otherwise
        poke_cycles=1;
end

end

% --- Executes during object creation, after setting all properties.
function poke_number_CreateFcn(hObject, eventdata, handles)
% hObject    handle to poke_number (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
end

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

%toggle the flag variable so that the other process will stop
set(handles.poke,'UserData',0);
guidata(hObject, handles);

end

function pause_time_Callback(hObject, eventdata, handles)
% hObject    handle to pause_time (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of pause_time as text
%        str2double(get(hObject,'String')) returns contents of pause_time as a double

global delay;
handles.pausetime=str2double(get(hObject,'String'));
delay=str2double(get(hObject,'String'));
end

% --- Executes during object creation, after setting all properties.
function pause_time_CreateFcn(hObject, eventdata, handles)
% hObject    handle to pause_time (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
end

function m2_applanation_dist_Callback(hObject, eventdata, handles)
% hObject    handle to m2_applanation_dist (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of m2_applanation_dist as text
%        str2double(get(hObject,'String')) returns contents of m2_applanation_dist as a double
global m2_applanation_distance;
m2_applanation_distance=str2double(get(hObject,'String'))/(0.0254);

end

% --- Executes during object creation, after setting all properties.
function m2_applanation_dist_CreateFcn(hObject, eventdata, handles)
% hObject    handle to m2_applanation_dist (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
end

% --- Executes on button press in Waveform.
function Waveform_Callback(hObject, eventdata, handles)
% hObject    handle to Waveform (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global Z delay dio TETA S file_name;

%initialization

load waveform;
disp('waveforms loaded')
if length(waveform1)==length(waveform2)
    array_length=length(waveform2);
else
    disp('error: waveforms must be same length')
    array_length=0;
end
%index for loop
```

```
i=1;
%index for arrays
index1=1;
index2=1;
array_length1=length(waveform1);
array_length2=length(waveform2);

%goto initial M1 value
z=waveform1(1)/.0254;
if (z>Z)                                        % if the disgnated coordinate is more
    while (z>Z)
        Z=forward2(Z,delay,dio);
        set(handles.motor1_coordinate_value,'String',...
            num2str(Z*0.0254));                              % exporting the Trans coordinate
    end
elseif (z<Z)
    while (z<Z)                                    % if the disgnated coordinate is less
        Z=backward2(Z,delay,dio);
        set(handles.motor1_coordinate_value,'String',...
            num2str(Z*0.0254));                              % exporting the Trans coordinate
    end
end

Z

%goto initial M2 value
teta=waveform2(1)/.0254;
if (teta>TETA)                                     % if the disgnated coordinate is more
    while (teta>TETA)
        TETA=CW(TETA,delay,dio);
        set(handles.motor2_coordinate_value,'String',...
            num2str(TETA*0.0254));                           % exporting the Trans coordinate
    end
elseif (teta<TETA)
    while (teta<TETA)                                 % if the disgnated coordinate is less
        TETA=CCW(TETA,delay,dio);
        set(handles.motor2_coordinate_value,'String',...
            num2str(TETA*0.0254));                           % exporting the Trans coordinate
    end
end

TETA

%start the waveform loop
tic;
while index1<=array_length1 || index2<=array_length2;
    %set target values (z,teta) if array values are still valid
    if index1<=array_length1, z=waveform1(index1)/.0254;, end;
    if index2<=array_length2, teta=waveform2(index2)/.0254;, end;

    if (z>Z) && index1<=array_length1, Z=forward2(Z,delay,dio);,
        if (z<=Z), index1=index1+1;, end
```

```
    elseif (z<Z) && index1<=array_length1, Z=backward2(Z,delay,dio);,
        if (z>=Z), index1=index1+1;, end
    end

   set(handles.motor1_coordinate_value,'String',...
            num2str(Z*0.0254));

    if (teta>TETA) && index2<=array_length2, TETA=CW(TETA,delay,dio);,
        if (teta<=TETA), index2=index2+1;, end
    elseif (teta<TETA) && index2<=array_length2, TETA=CCW(TETA,delay,dio);,
        if (teta>=TETA), index2=index2+1;, end
    end

     set(handles.motor2_coordinate_value,'String',...
             num2str(TETA*0.0254));


    %disp(index1)
    %disp(index2)

    [V1(i),V2(i),V3(i)] = get_DAQ2(S);
    t(i)=toc;
    m1(i)=Z*.0254;
    m2(i)=TETA*.0254;
    i=i+1;
end
disp('done');

    %save output to file

        %this output is not quite working...
        fid = fopen('data_temp.dat','w');
        fprintf(fid,'');
        fprintf(fid,'%g %g %g %6.4f %6.4f %6.4f\n',t,m1,m2,V1,V2,V3);

        date=datestr(now, 'dd.mm.yyyy');
        mkdir('data',date);
        cd('data');
        cd(date);

        save (file_name,'t', 'm1', 'm2' ,'V1' ,'V2' ,'V3');
        cd('..');
        cd('..');

        create_plots;
end

% --- Executes on button press in goto_zero_button.
%move M1 and M2 to 0 position
function goto_zero_button_Callback(hObject, eventdata, handles)
% hObject    handle to goto_zero_button (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)
global Z delay dio TETA S;

z_desired=0;
teta_desired=0;

%goto initial M1 value
z=z_desired/.0254;
if (z>Z)                                        % if the disgnated coordinate is more
    while (z>Z)
        Z=forward2(Z,delay,dio);
        set(handles.motor1_coordinate_value,'String',...
            num2str(Z*0.0254));                             % exporting the Trans coordinate
    end
elseif (z<Z)
    while (z<Z)                                     % if the disgnated coordinate is less
        Z=backward2(Z,delay,dio);
        set(handles.motor1_coordinate_value,'String',...
            num2str(Z*0.0254));                             % exporting the Trans coordinate
    end
end

Z

%goto initial M2 value
teta=teta_desired/.0254;
if (teta>TETA)                                      % if the disgnated coordinate is more
    while (teta>TETA)
        TETA=CW(TETA,delay,dio);
        set(handles.motor2_coordinate_value,'String',...
            num2str(TETA*0.0254));                          % exporting the Trans coordinate
    end
elseif (teta<TETA)
    while (teta<TETA)                                   % if the disgnated coordinate is less
        TETA=CCW(TETA,delay,dio);
        set(handles.motor2_coordinate_value,'String',...
            num2str(TETA*0.0254));                          % exporting the Trans coordinate
    end
end

end

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

tic;
global Z delay dio TETA S exp_slope1 exp_offset1 exp_slope2 exp_offset2  setpoint1_value setpoint2_value file_name;
global Data_resolution_value;
%constants
```

```
%setpoint force in N
%setpoints 1 and 2 are set by a text box in GUI
    SETPOINT1=setpoint1_value;
    SETPOINT2=setpoint2_value;
%increment motor 1 to specified amount

    sensor1_force_setpoint=SETPOINT1;
    sensor2_force_setpoint=SETPOINT2;
    i=1;
    j=1
    [V1(i),V2(i),V3(i)] = get_DAQ2(S);

    %adjust the output values according to the stored calibration data
    V1(i)=(V1(i)-exp_offset1)./exp_slope1/1000*9.81;
    V2(i)=(V2(i)-exp_offset2)./exp_slope2/1000*9.81;

    while V1(i)<sensor1_force_setpoint

        j=j+1;

        %value of Data_resolution_value set by drop down textbox.  If the
        %remainder is 0, the DAQ will make a measurement, otherwise it will
        %advance to the next step.  This will allow the motors to actuate
        %faster if less measurements are made
        if rem(j,Data_resolution_value)==0
        i=i+1;
        [V1(i),V2(i),V3(i)] = get_DAQ2(S);

        V1(i)=(V1(i)-exp_offset1)./exp_slope1/1000*9.81
        V2(i)=(V2(i)-exp_offset2)./exp_slope2/1000*9.81


        t(i)=toc;
        m1(i)=Z*.0254;
        m2(i)=TETA*.0254;
        end
        Z=forward2(Z,delay,dio);
        set(handles.motor1_coordinate_value,'String',...
            num2str(Z*0.0254));

    end

    %now we increment motor 2

    while V2(i)<sensor2_force_setpoint
         disp('incrementing motor 2')

        j=j+1;
        if rem(j,Data_resolution_value)==0
        i=i+1;
        [V1(i),V2(i),V3(i)] = get_DAQ2(S);
```

```
    V1(i)=(V1(i)-exp_offset1)./exp_slope1/1000*9.81
    V2(i)=(V2(i)-exp_offset2)./exp_slope2/1000*9.81


    t(i)=toc;
    m1(i)=Z*.0254;
    m2(i)=TETA*.0254;
    end
    TETA=CW(TETA,delay,dio);
     set(handles.motor2_coordinate_value,'String',...
        num2str(TETA*0.0254));

end

%Remove the force sensor2
while 0 < TETA
     disp('decrementing motor 2')

    j=j+1;
    if rem(j,Data_resolution_value)==0
        i=i+1;
    [V1(i),V2(i),V3(i)] = get_DAQ2(S);


    V1(i)=(V1(i)-exp_offset1)./exp_slope1/1000*9.81
    V2(i)=(V2(i)-exp_offset2)./exp_slope2/1000*9.81


    t(i)=toc;
    m1(i)=Z*.0254;
    m2(i)=TETA*.0254;
    end
    TETA=CCW(TETA,delay,dio);
     set(handles.motor2_coordinate_value,'String',...
        num2str(TETA*0.0254));

end

%Remove Force Sensor1
while 0<Z

    j=j+1;
    if rem(j,Data_resolution_value)==0
        i=i+1;
    [V1(i),V2(i),V3(i)] = get_DAQ2(S);

    V1(i)=(V1(i)-exp_offset1)./exp_slope1/1000*9.81
    V2(i)=(V2(i)-exp_offset2)./exp_slope2/1000*9.81


    t(i)=toc;
```

```
        m1(i)=Z*.0254;
        m2(i)=TETA*.0254;
        end
        Z=backward2(Z,delay,dio);
        set(handles.motor1_coordinate_value,'String',...
            num2str(Z*0.0254));

    end

    %save the result

    datamatrix=[V1' V2' V3' m1' m2' t']
    %save ('palpation_data','t', 'm1', 'm2' ,'V1' ,'V2' ,'V3');
    save ('palpation_data_matrix','datamatrix')
    disp('datamatrix saved')

     %save output to file
        fid = fopen('data_temp.dat','w');
        fprintf(fid,'');
        fprintf(fid,'%g %g %g %6.4f %6.4f %6.4f\n',t,m1,m2,V1,V2,V3);

        date=datestr(now, 'dd.mm.yyyy');
        mkdir('data',date);
        cd('data');
        cd(date);
       % save ('datafile','t', 'm1', 'm2' ,'V1' ,'V2' ,'V3');
       save (file_name,'t', 'm1', 'm2' ,'V1' ,'V2' ,'V3');
        cd('..');
        cd('..');
end

% --- Executes on button press in Calibrate_sensors.
function Calibrate_sensors_Callback(hObject, eventdata, handles)
% hObject    handle to Calibrate_sensors (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global Z delay dio TETA S exp_slope1 exp_offset1 exp_slope2 exp_offset2;

[V1(1),V2(1),V3(1)] = get_DAQ2(S);
exp_offset1=V1(1)
exp_offset2=V2(1)

%load the data from the calibration files.  The program 'calibrate force
%sensors must be used to generate this data before starting the experiment

load sensor1_calib
load sensor2_calib

exp_slope1=slope1
exp_slope2=slope2

end
```

```
function setpoint1_value_Callback(hObject, eventdata, handles)
% hObject    handle to setpoint1_value (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of setpoint1_value as text
%        str2double(get(hObject,'String')) returns contents of setpoint1_value as a double
global setpoint1_value;
setpoint1_value=str2double(get(hObject,'String'));

end

% --- Executes during object creation, after setting all properties.
function setpoint1_value_CreateFcn(hObject, eventdata, handles)
% hObject    handle to setpoint1_value (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
end


function setpoint2_value_Callback(hObject, eventdata, handles)
% hObject    handle to setpoint2_value (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of setpoint2_value as text
%        str2double(get(hObject,'String')) returns contents of setpoint2_value as a double
global setpoint2_value;
setpoint2_value=str2double(get(hObject,'String'));
end

% --- Executes during object creation, after setting all properties.
function setpoint2_value_CreateFcn(hObject, eventdata, handles)
% hObject    handle to setpoint2_value (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
end
```

```
function file_name_Callback(hObject, eventdata, handles)
%sets the file name for the relevant save file
% hObject    handle to file_name (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of file_name as text
%        str2double(get(hObject,'String')) returns contents of file_name as a double

global file_name;
file_name=get(hObject,'String');

end

% --- Executes during object creation, after setting all properties.
function file_name_CreateFcn(hObject, eventdata, handles)
% hObject    handle to file_name (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

end



% --- Executes on selection change in Data_resolution.
function Data_resolution_Callback(hObject, eventdata, handles)
% hObject    handle to Data_resolution (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 Data_resolution contents as cell array
%        contents{get(hObject,'Value')} returns selected item from Data_resolution

global data_resolution_value;
        data_resolution_value=1;
switch get(handles.Data_resolution,'Value')
    case 1
        data_resolution_value=1;
    case 2
        data_resolution_value=2;
    case 5
        data_resolution_value=5;
    case 10
        data_resolution_value=10;
    otherwise
        data_resolution_value=1;
```

```
end


end
% --- Executes during object creation, after setting all properties.
function Data_resolution_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Data_resolution (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

global data_resolution_value;
        data_resolution_value=1;

end
```