THE HAND-HELD FORCE MAGNIFIER: SURGICAL INSTRUMENTS TO AUGMENT THE SENSE OF TOUCH

by

Randy Lee

B.S., University of California, San Diego, 2010

M.S., Carnegie Mellon University, 2012

Submitted to the Graduate Faculty of

Swanson School of Engineering in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

University of Pittsburgh

2017

UNIVERSITY OF PITTSBURGH

SWANSON SCHOOL OF ENGINEERING

This dissertation was presented

by

Randy Lee

It was defended on

March 21, 2017

and approved by

Roberta L. Klatzky, Ph.D., Professor, Department of Psychology, Carnegie Mellon University

Zhi-Hong Mao, Ph.D., Associate Professor
Department of Electrical and Computer Engineering
and Department of Bioengineering

Anne Robertson, Ph.D., Professor
Department of Mechanical Engineering and Materials Science
and Department of Bioengineering

Dissertation Director: George Stetten, M.D., Ph.D., Professor, Department of Bioengineering

Copyright © by Randy Lee

2017

THE HAND-HELD FORCE MAGNIFIER: SURGICAL TOOLS TO AUGMENT THE SENSE OF TOUCH

Randy Lee, Ph.D.

University of Pittsburgh, 2017

Modern surgeons routinely perform procedures with noisy, sub-threshold, or obscured visual and haptic feedback, either due to the necessary approach, or because the systems on which they are operating are exceeding delicate. For example, in cataract extraction, ophthalmic surgeons must peel away thin membranes in order to access and replace the lens of the eye. Elsewhere, dissection is now commonly performed with energy-delivering tools – rather than sharp blades – and damage to deep structures is possible if tissue contact is not well controlled. Surgeons compensate for their lack of tactile sensibility by relying solely on visual feedback, observing tissue deformation and other visual cues through surgical microscopes or cameras. Using visual information alone can make a procedure more difficult, because cognitive mediation is required to convert visual feedback into motor action. We call this the "haptic problem" in surgery because the human sensorimotor loop is deprived of critical tactile afferent information, increasing the chance for intraoperative injury and requiring extensive training before clinicians reach independent proficiency.

Tools that enhance the surgeon's direct perception of tool-tissue forces can therefore potentially reduce the risk of iatrogenic complications and improve patient outcomes. Towards

this end, we have developed and characterized a new robotic surgical tool, the Hand-Held Force Magnifier (HHFM), which amplifies forces at the tool tip so they may be readily perceived by the user, a paradigm we call "*in-situ*" force feedback.

In this dissertation, we describe the development of successive generations of HHFM prototypes, and the evaluation of a proposed human-in-the-loop control framework using the methods of psychophysics. Using these techniques, we have verified that our tool can reduce sensory perception thresholds, augmenting the user's abilities beyond what is normally possible. Further, we have created models of human motor control in surgically relevant tasks such as membrane puncture, which have shown to be sensitive to push-pull direction and handedness effects. Force augmentation has also demonstrated improvements to force control in isometric force generation tasks. Finally, in support of future psychophysics work, we have developed an inexpensive, high-bandwidth, single axis haptic renderer using a commercial audio speaker.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ACRONYMS AND SYMBOLS USED

AAA – Abdominal Aortic Aneurysm

ABS – Acrylonitrile Butadiene Styrene

A/D, ADC – Analog to Digital (Conversion)

ANOVA – Analysis of Variance statistical test

API – Application Programming Interface

CAD – Computer-Aided Design

CCC – Continuous Curvilinear Capsulorhexis

CDF – Cumulative Distribution Function

CTE – Coefficient of Thermal Expansion

CPU – Central Processing Unit

D/A, DAC – Digital to Analog (Conversion)

DAQ – Data Acquisition Unit

DC – Direct Current

DoF – Degree of Freedom

ECCE – Extracapsular Cataract Extraction

EHI – Edinburgh Handedness Inventory

ERM – Epiretinal Membrane

FA – Fast-Adapting (receptor)

I^2C – Inter-Integrated Circuit

I/O, IO – Input/Output

IR – Infrared

JND – Just Noticeable Difference

K-S – Kolmogorov-Smirnov statistical test

LVCM – Linear Voice Coil Motor

MFC – Microsoft Foundation Class (libraries)

MIS – Minimally Invasive Surgery (aka laparoscopy)

MLHD – Magnetically Levitated Haptic Device

MMR – Memory Mapped Register

OCT – Optical Coherence Tomography

PCT – Posterior Capsule Tear

PID – Proportional Integral Differential (control)

PVDF – Polyvinylidene fluoride

RA – Rapidly-Adapting (receptor)

RCM – Remote Center of Motion

RMS – Root-Mean-Square

SA – Slowly-Adapting (receptor)

SD – Standard Deviation

SEM – Standard Error of the Mean

SHR – Steady Hand Robot

SLA – Stereolithography

SPI – Serial Peripheral Interface

VI – Virtual Instrument

USB – Universal Serial Bus

Z-N – Ziegler-Nichols

A – area, in units square-meters, [m^2]

$Bl(x_{speaker})$ – electro-dynamical force factor

c – Weber's constant

C – 4th order stiffness tensor

$C_{ms}(x_{speaker})$ – compliance, lumped parameter mechanical analogue to capacitance

d_{ij} – dieletric constant on the i-th face and j-th direction

$e(t)$ – error signal, units of which depend on quantity measured

f – force, in units Newtons, [N]

$f(t)$ – tool-tissue interaction force measured by HHFM

$F(t)$ – feedback force generated by HHFM

$F_a(t)$ – afferent (sensed) force perceived by the user

$F_e(t)$ – efferent (output) force generated by the user

F – current force applied to speaker, in units Newtons, [N]

F_{sp} – force set point, in units Newtons, [N]

$G(s)$ – transfer function

j – the complex (or imaginary) variable, where j = $\sqrt{-1}$

k – HHFM gain

k_s – Hookean spring constant, in units Newtons/meters,[$N * m^{-1}$]

k_τ – HHFM torque gain

K_c – PID controller gain

K_d – Differential PID gain

K_i – Integral PID gain

K_p – Proportional PID gain

K_u – Ziegler-Nichols gain at oscillation, the “ultimate gain”

l – length, in units meters, [m]

l_c – distance between capacitor electrodes, in units meters, [m]

L – inductance, in units Henries, [H]

M_{ms} – mass, lumped parameter mechanical analogue to inductance

P_u – Ziegler-Nichols oscillation period, in units seconds, [s]

r – reference point, set point

R – resistance, in units ohms, [Ω]

R_{ms} – air resistance and viscous damping, lumped parameter mechanical analogue to resistance

S_ε – Strain gage factor, unitless

t – time, in units seconds [s]

x – displacement, in units meters, [m] or millimeters, [mm]

x_{sp} – speaker displacement set point, in units millimeters, [mm]

V_{IR} – output voltage from IR position sensor, in units volts [V]

y – control process output

α – statistical significance level

ε – strain, unitless

ε_0 – the vacuum permittivity, a constant of nature with value 8.8542×10^{-2} [$C^2 * N^{-1} * m^{-2}$]

κ – the dielectric constant, a ratio relative to the permittivity of vacuum

ρ – material resistivity, in units ohm-meters, $[\Omega * m]$

σ – stress, in units Newtons/squared meters, $[N * m^2]$ or Pascals, [Pa]

τ – biomechanical damping constant, from membrane puncture model, in units seconds, [s]

τ_a – afferent torque, the torque perceived while holding a hand-held tool

τ_d – distal torque, the torque at the grip point due to a force at the distal tip of a tool

τ_p – proximal torque, the torque at the grip point due to the force at the proximal end of a tool

τ_I – integral reset time, in units seconds [s]

τ_D – derivative time, in units seconds [s]

Φ – stimulus strength/ intensity

Ψ – sensation magnitude

$\mathscr{F}\{\cdot\}$ – the Fourier transform

$\mathscr{L}\{\cdot\}$ – the Laplace transform

PREFACE

This document certainly would not have been possible without the support of the many world-class scientists and engineers I have been proud to call my colleagues over my time as a graduate student.

I first want to thank Professor George Stetten, my dissertation advisor from the beginning of my graduate career, for his generous and patient mentorship. Most of all I appreciate his good humor, intellectual rigor, and willingness to believe in and support me. I am a much more confident and independent scientist and engineer as a result of his expert guidance.

Next, I want to thank Professor Roberta Klatzky, our psychophysics collaborator with an incredible intuition for good science through clever experimentation. Her expertise is a wonder to behold, and I will forever attribute my interest in perception, and the manipulation thereof, to her. In addition, I want to thank our other collaborators in psychology, Professor Bing Wu and Dr. Pnina Gershon.

I want to acknowledge the other professors and researchers in our group, the Visualization and Image Analysis (VIA) Laboratory: Professors John Galeotti and Mel Siegel, who have provided additional insight and expertise whenever needed. Professors Joel S. Schuman (Ophthalmology) and Vijay Gorantla (Plastic and Transplant Surgery) have been our major clinical collaborators, grounding our work in medical reality. I want to especially thank

Professor Ralph Hollis, for generously lending the MLHD for a number of psychophysics experiments and projects.

The VIA Lab would not be without its graduate students, to whom I am indebted for their help overcoming various technical challenges and braving the rigors of graduate school: Drs. Vikas Shivaprabhu, Jihang Wang, and Samantha Horvath; in addition to Bo Wang, Tejas Mathai, Zhixuan Yu, and Bruce Che.

My many undergraduate research assistants have all done excellent work, many results of which are presented in this dissertation: Claire Hoelmer, Cindy Wong, Ma Luo, Avin Khera, Avi Marcovici, and Oliver Snyder. I am proud to say that, of the four that have since graduated, all have found positions in industry or are pursuing their own doctoral degrees.

A number of resources at the University of Pittsburgh have made this dissertation possible. In addition to Professors Stetten and Klatzky, I want to acknowledge Professor Anne Robertson and Professor Zhi-Hong Mao as critical members of my doctoral committee. Their feedback and questions have helped develop the dissertation to be as robust as possible. Andy Holmes and his team have been a great help in the Swanson School of Engineering machine shop. Professor Alan Hirschman and the many people in the Innovation Institute have been great resources as I conclude my dissertation.

I also want to thank Mr. Gerald McGinnis, for his generous sponsorship of the McGinnis Bioengineering Graduate Fellowship in Innovative Medical Technology, which has supported me throughout my Ph.D. work. In addition, grants from the Coulter Foundation, Research to Prevent Blindness, the National Institutes of Health, and the National Science Foundation have all supported this work.

Finally, I want to thank my friends and family without whose support this marathon run through academia would not have been possible. My friends Greg Johnson, Arush Kalra, Molly Blank, Rebecca Duffy, Kevin Fok, and Kristine Ojala have been of particular help. As have my sisters, Sharon, Tammy, and Valerie.

I dedicate this dissertation to my parents, Vason and Brenda, who have always been a constant source of support and inspiration. Having arrived in the United States of America in July 1979 with nothing but hope, that their children could achieve as much as we have in less than 40 years is a testament to their hard work and dedication.

This work has generated a number of scientific publications, which are listed here for convenience:

G. Stetten, B. Wu, R.L. Klatzky, J. Galeotti, M. Siegel, **R. Lee**, R. Hollis. "Hand-Held Force Magnifier for Surgical Instruments," *2nd International Conference on Information Processing in Computer-Assisted Interventions (IPCAI)*, June 22-23, 2011, Berlin, Germany, Proceedings in LNCS 6689, pp. 90–100.

R. Lee, B. Wu, R.L. Klatzky, V. Shivaprabhu, J. Galeotti, S. Horvath, M. Siegel, J.S. Schuman, R. Hollis, G. Stetten, "Hand-Held Force Magnifier for Surgical Instruments: Evolution toward a Clinical Device," *Augmented Environments & Computer-Aided Interventions (AE-CAI)*, October 25, 2012, Nice, France, Proceedings in LNCS 7815, pp. 77-89.

R.L. Klatzky, P. Gershon, V. Shivaprabhu, **R. Lee**, B. Wu, G. Stetten, R.H. Swendsen, "A model of motor performance during surface penetration: From physics to voluntary control," *Experimental Brain Research*, vol. 230, no. 2, pp. 251–260, 2013.

B. Wu, R.L. Klatzky, **R. Lee**, V. Shivaprabhu, J. Galeotti, M. Siegel, J.S. Schuman, R. Hollis, G. Stetten, "Psychophysical Evaluation of Haptic Perception under Augmentation by a Hand-Held Device," *Human Factors*, vol. 57, no. 3, pp. 523-537, 2015.

1.0 INTRODUCTION

Beginning in the 1800's, rapid advances in the art and science of surgical intervention have hinged on bridging four major gaps in understanding [1]:

1. Knowledge of human anatomy;
2. Method of controlling hemorrhage and maintaining intraoperative hemostasis;
3. Anesthesia to permit the performance of pain-free procedures;
4. Explanation of the nature of infection, along with the elaboration of methods necessary to achieve an antiseptic and aseptic operating room environment.

Overcoming these obstacles took the greater part of a century. Surgeons have since distinguished themselves as respected professionals, pushing the boundaries of medicine through active intervention. At first, large incisions were made into the body to expose the patient's tissues and organs, giving the surgeon a relatively large workspace and direct line of sight. As their collective skill has improved, the surgeon's tools have also evolved, specializing as procedures warrant. Evolution in tool design is sometimes attributable to novel approaches like laparoscopy, also known as minimally invasive surgery (MIS). In parallel with the development of MIS, surgery is pushing on to increasingly demanding frontiers, like surgery of the eye, the vasculature, or the brain and nervous tissue. As we will see, these new surgical disciplines begin to approach the fundamental boundaries of human sensory and motor control abilities. These challenges require a new class of surgical instrument.

1.1 MOTIVATION: EMERGENCE OF THE "HAPTIC" PROBLEM

As surgery has progressed as a profession and skill, we have seen the emergence of what we call the "haptic problem." That is, as novel surgical approaches are developed, or as surgery expands to new systems in the body, the sense of touch is sacrificed or lost completely. Upon losing their tactile sense, surgeons rely on visual feedback to guide their procedures. Relying solely on visual feedback can increase the difficulty of the surgery, thereby extending the time and practice required to reach independent proficiency. Specifically, these types of surgical procedures are difficult because *cognitive mediation* is required to actuate motor commands in response to visual feedback.

Cognitive mediation describes the central planning or processing between a stimulus and an associated response, which results in cognitive delays [2]. In some cases, cognitive delays are not cortical in nature, such as those related to motor reflexes, which are mediated by only a few neurons in the dorsal root ganglia of the spinal cord. The well-known patellar reflex falls within this type of delay. Other, longer delays are cortical, such as those due to the complex environment in which a person is interacting, requiring that one split one's attention between multiple stimuli. This includes everything from maintaining a conversation with a friend while walking along a busy street, to performing a complicated surgery [3]. In designing assistive technologies for the blind, for example, these longer cognitive delays suggest that only some essential subset of the total visual information should be presented [4]. Another term that describes the extent to which cognitive mediation is required to interact with a technology is its *cognitive load*.

Considering these cognitive concerns with respect to surgery, while the trainee surgeon gains experience, their motor actions are in some part *reactive*. This is because novice surgeons

refine their motor skills and build a mental model of the surgical environment only by actually performing procedures alongside an experienced mentor. In contrast to the novice, the expert surgeon judiciously manipulates tissue, moving economically and confidently to accomplish a specific surgical goal. Objective measures of surgical skill, which highlight these differences between experts and novices, are becoming more prevalent in surgical education, and are sufficiently precise to discriminate between surgeons at differing points in their training and careers [5]–[9].

Two surgical specialties in which the haptic problem potentially contributes to surgical difficulty are MIS and microsurgery. These surgical specialties share some similarities, even though the scales on which they operate are vastly different. Visual feedback during both MIS and microsurgery, for example, is magnified compared to the normal visual experience. Furthermore, in both disciplines, surgical procedures are performed where clinicians also do not have direct tactile feedback.

1.1.1 Minimally Invasive Surgery

MIS describes a surgical approach that all but eliminates the need for surgeons to make large incisions into the body. Access to an internal cavity is achieved through several ports in the body wall, through which cameras, lights, and surgical tools are inserted, as shown in **Figure 1** [10].

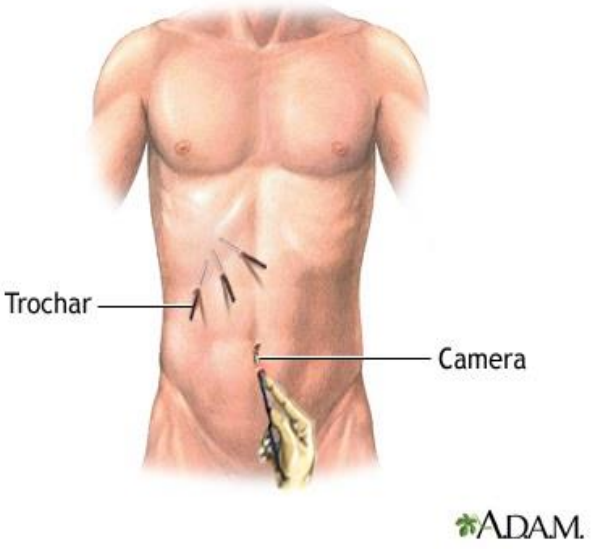


Figure 1. Laparoscopic, or minimally invasive, surgery targeting the liver. [10]

An inert gas (e.g. CO_2) can be used to inflate the cavity and increase the surgeon's workspace [11]. Minimally invasive approaches are associated with reduced postoperative pain and infection, and shorter periods of disability [12]–[14].

However, the minimally invasive approach still presents significant challenges. Surgical tools must be specially designed to accommodate these new constraints. MIS tools are commonly actuated using a scissor grip, and feature long bodies to allow for tool insertion and tissue manipulation. A significant problem associated with these designs is that mechanical torques and friction forces are generated between the tool and the trocar, the port through which MIS instruments are inserted into the body. These extraneous forces mask the forces applied at the distal tip, confounding the tactile feedback usually obtained while manipulating tissue free-hand [15]. The extended length of the tool itself also masks the haptic feedback obtained from the interaction at the tool tip. Research has shown that inaccurate control of grasping due to these difficulties can result in tissue damage or tissue slip [16], [17].

Operating through instrumented ports drastically reduces the clinician's workspace and field of view, compared to the traditional open approach, requiring frequent adjustment of the camera. In addition, minimally invasive approaches almost always eliminate the sense of depth obtained through stereopsis, since a single camera is generally inserted into the body cavity, providing only monocular visual feedback displayed on a screen or through a tube containing lenses. Finally, because MIS tools are pivoted about the port through which they are inserted into the body, the motor mapping required to manipulate them is reversed (e.g., to move the distal end of the instrument rightward requires movement of the proximal end to the left). Physiological tremor and other unintended movements may also be magnified through this lever arm effect.

Through all of these difficulties, specialists have been able to develop a number of minimally invasive procedures that significantly improve patient outcomes. Common minimally invasive procedures include orthopaedic surgeries like anterior cruciate ligament repair; *radical prostectomies* and *vasectomies* in urology; and *cholecystectomies* (gallbladder removal) and *gastrectomies* (removal of all or part of the stomach) in abdominal surgery, among many others.

1.1.2 Microsurgery

In parallel with advances in MIS, microsurgical approaches have developed in recent years as a subspecialty of general surgery. Microsurgery refers in particular to the set of surgical techniques that are performed beyond the limits of normal human eyesight [18]. In microsurgery, clinicians may still make incisions to directly access tissues of interest, but in general they operate on relatively small structures including nerves, vessels, or thin basement membranes, using correspondingly smaller forces [19]. For example, Jagtap and Riviere measured forces during *in-vivo* microvascular puncture tasks, where average axial forces were found to be as low as 75 mN [20]. Further, movements are performed over such a small scale that kinesthetic afferent signals are also not differentiable (see Section 2.2.3) [21].

A core assistive technology in these delicate procedures is the surgical microscope. As in MIS, using the surgical microscope constitutes an altered visual experience compared to everyday life, due to the optical characteristics of the magnifying lenses. Using the surgical microscope, for example, limits the surgeon's depth of focus, causing structures above or below the focal plane to appear blurred. Visual magnification itself can affect one's motor control capabilities when manipulating delicate tissue, due to the discrepancy between one's movement and the *observation* of that movement [22]. Compared to MIS, however, the underlying cause of

the haptic problem in microsurgery is vastly different. Whereas the haptic problem in MIS is characterized by friction or other factors masking the tool-tissue interaction, the haptic problem in microsurgery derives from fundamental properties of the tissues of interest. Surgical specialties that are commonly viewed as being microsurgical include vascular surgery and ophthalmic surgery.

1.1.2.1 Vascular Surgery

Vascular surgery is concerned with the surgical treatment and repair of the body's arteries and veins. A common task in vascular surgery, for example, involves achieving a leak-proof *anastomosis*, a connection between two vessels. There are many different suturing patterns and approaches at the surgeon's disposal, which are indicated depending on the particular system or location of repair [23], [24]. With each pass of the needle and suture, the surgeon must be careful to only puncture one side of the vessel at a time; crossing the diameter of the vessel and puncturing the far wall can result in leaks or additional trauma. This type of error is known as "back-walling," and is particularly difficult to avoid because vessels tend to collapse without their usual internal lumen or blood flow. All of these difficulties are further compounded as the diameter of the smallest vessels is between 1.5 – 2 mm.

Due to such issues, vascular surgeons are often called upon by their surgical colleagues to assist with difficult dissections, or in the repair or reconstruction of blood vessels [25]. For example, access to the carotid artery, the vital artery which supplies oxygenated blood to the brain, face, and neck, requires careful dissection of the surrounding musculature, and experience working around the critical nervous and vascular anatomy in the neck [26].

Vascular pathologies can have devastating consequences. For example, abdominal aortic aneurysms (AAA) are deformations of the wall in the largest artery in the body, the aorta, which

supplies blood, oxygen, and nutrients to the abdominal organs and lower extremities [27], [28]. This condition affects up to 1.1 million Americans, and is surgically treated using either open or endovascular (catheter-based) approaches [29]. Dilation of the aortic vessel weakens the strength of the wall, exposing the patient to a potentially deadly rupture. Unfortunately, AAA rupture results in death in 78% of patients, with three quarters of these deaths occurring outside the hospital [30]. Recent research has shown that both open or endovascular surgery for AAA rupture offer similar recovery rates, with patients undergoing open or endovascular intervention seeing 69.9% and 68.9% cumulative survival, respectively [31]. Other recent work has indicated that patient survival is improving, and that there are fewer major complications following emergency repair of ruptured AAA [32]. The reality is, however, that even amongst the percentage of patients that survive to undergo surgery, 20 to 50% of patients suffer intraoperative mortality.

Key in successfully treating AAA is accurate localization of the aneurysm prior to surgery. In cases where the aneurysm has not yet ruptured, the aorta is clamped, the aneurysm excised, and a synthetic graft placed to replace the excised vessel section. When the repair is done using an endovascular approach, fluoroscopy, a type of x-ray imaging, is used to visualize the progress of tools placed within the body. In ruptured AAA repair, however, the patient may very quickly bleed out internally, placing extreme time pressure on the clinical team. One aspect that contributes to the difficulty of AAA repair is navigating the surgical site in the presence of the “bloody field,” so called because blood obscures the structures of interest. Navigating to the precise location of the ruptured aneurysm requires feeling into the abdominal space to locate the rupture site, since the space is typically obscured by internal bleeding. In emergent cases preoperative imaging may not be available, further adding to the difficulty of successful repair.

1.1.2.2 Ophthalmic Surgery

Ophthalmic surgery is a discipline of microsurgery concerned with surgical intervention in the eye. This microsurgical specialty is further divided between the anterior (front) or posterior (back) segments of the eye. The cornea, capsule, and lens of the eye comprise the anterior segment, as depicted in **Figure 2,** while the vitreous and retina make up the posterior segment [33].

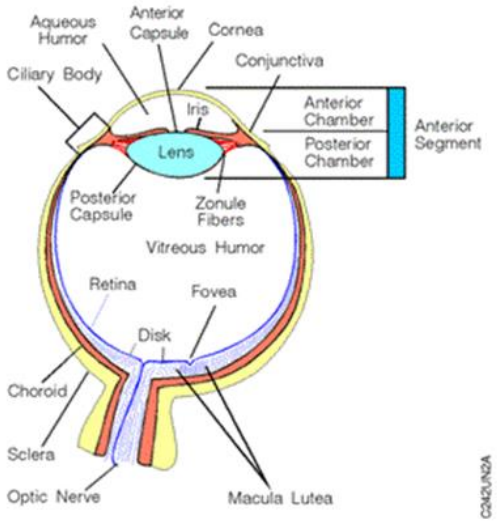


Figure 2. Ocular anatomy. [33]

Common posterior segment surgeries include epiretinal membrane (ERM) peel, which is indicated when a thin membrane covers the retinal tissue, a condition called macular pucker [34]. In severe cases, macular pucker results in significant visual acuity loss. Commonly, patients with ERM also present with posterior vitreous detachment, a condition in which the vitreous separates from the retina [35], [36]. Surgical treatment of macular pucker begins with a standard vitrectomy, in which the vitreous is removed. Care must be taken to avoid large pulling forces while removing the jelly-like vitreous, as even this can damage the delicate retina. The procedure then proceeds with ERM peel, where micro-forceps or surgical picks are used to lift and separate the pathologic membrane from the underlying retina.

Surgical treatment of macular pucker is difficult because the tissue of interest is on average only 10 μm thick, and contact with the sensitive retinal tissue must be minimized. Gupta and colleagues have found that during this type of retinal surgery, tissue forces are perceived by the surgeon only 19% of the time, and 75% of all tissue interactions generate forces smaller than 7.5 mN [37]. Further adding to the difficulty of the procedure, access to the retina is achieved

through the sclera, creating a pivot point through which the tool is controlled, as is the case in MIS. One common complication of ERM peel includes retinal detachment, in which intraoperative forces or vitreous leakage separate the retina from the back of the eye, occurring in 2-14% of cases [35].

The most common anterior segment surgery is cataract extraction, in which a pathologic, opaque lens is extracted and replaced with an artificial intraocular lens. Cataracts develop over time as a natural aspect of aging. A combination of factors, including protein aggregation, lifetime UV exposure, and systemic illness, leads to regions of opacity in the lens of the eye, which blur or tint one's vision [38]. In the United States, 1.8 million Medicare beneficiaries underwent cataract surgery in 2004 alone [39]. The National Eye Institute estimates that by the age of 80, half of all Americans will either have a cataract, or already have had cataract surgery [40]. Further, the number of persons with cataract is projected to rise to 30 million by 2020 [41].

Due to the high rate of cataract surgery utilization, projected to rise in the near future along with the elderly population, improvements to tools or procedures that can reduce overall complication rates will be extremely valuable [42]. Indeed, a patient's overall risk undergoing cataract surgery has generally diminished since the 1990's, as improved surgical techniques for lens removal – such as *extracapsular cataract extraction* (ECCE) and *phacoemulsification* – began to enter widespread practice [39]. However, as the population that requires cataract surgery continues to grow, it must be recognized that even low complication rates will translate into a significant number of adverse events.

A common technique currently employed to access the lens prior to extraction is *continuous curvilinear capsulorhexis* (CCC), in which the thin anterior capsule that constrains the lens is punctured and peeled or sheared away in a circular fashion. Usually the capsulorhexis

is initiated using a cystotome, a pick-like tool that is fashioned by bending the tip of a needle, and continued using either the cystotome or a pair of forceps [43], [44]. One hazard inherent to CCC occurs soon after the initiation of the tear. Although begun radially, the tear is turned to run circumferentially. In 0.5% to 0.8% of eyes, the tear can "run" out of control and veer beyond the pupillary margins, towards the capsular equator, and the underlying posterior capsule [39], [45]. Some surgical techniques to manage a radial tear have been proposed, but preventing further growth of the tear can sometimes require help from a more experienced specialist [46], [47]. Smooth vectoring of force is therefore essential to successful CCC. Posterior capsule tear (PCT), a breach in the integrity of the posterior capsule, is a more common intraoperative complication during cataract surgery, occurring in as many as 3.5% to 11% of cases [48], [49]. PCT may occur during either the CCC exposure stage, or even during the subsequent phacoemulsification phase. Phacoemulsification is the process by which the lens is broken up using focused ultrasound energy, and extracted by vacuum suction. The rate of PCT during phacoemulsification ranges between 0.7% and 16% [49].

The posterior pole of the capsule is only 2.8 μm in thickness, and is the thinnest part of the capsule. The incidence of PCT can vary depending on a number of factors, including: type and stage of cataract, lens extraction technique, and experience of the surgeon. For example, inaccurate control of forces on the opaque lens can lead to mechanical trauma and rupture of the posterior capsule. Successful management of PCT depends first on accurate recognition of the tear before it grows too large, and can ultimately involve the use of viscoelastic jellies to prevent prolapse of the membrane. Smaller PCTs are managed by performing a *posterior capsulorhexis*, while large tears or dislocation of nuclear debris are handled by performing a *vitrectomy*, where the vitreous of the eye is removed.

1.2 THE ROLE OF FORCE FEEDBACK IN SURGERY

We believe that many difficulties in MIS and microsurgery derive from deficiencies in the afferent information driving the human sensorimotor loop. Interaction forces are noisy or sub-threshold, and movements are performed over such a small scale that kinesthetic afferent signals are also not differentiable. Restoring or improving force sensibility in surgery can aid the surgeon by reducing the cognitive load required to balance the various emergent factors that may arise over the course of a procedure. Indeed, when designing tools to address these perceptual challenges, we aim to add as little as possible to the already considerable cognitive load associated with performing a surgical procedure. In effect, restoring the sense of touch would bring the experience of surgery closer to one's every day experience of interacting with the normal world. By providing to the surgeon a wider range of sensory stimuli, their mental models will be better informed, potentially leading to faster learning times and fewer intraoperative errors.

Auditory feedback, in contrast to haptic feedback, may not be well suited for the operating room due to conflict with ambient audio signals from medical equipment. Closed-loop visual feedback, such as graphical displays or other force-responsive visualizations, is potentially distracting and may lead to increased servo-ing when matching target forces over long periods of time. By providing additional feedback through the haptic channel, rather than through visual or auditory means, tools may be used more intuitively and be more easily integrated into existing practice. Specifically, we design tools to directly amplify action-dependent forces as they would be sensed by the user, a paradigm we call "*in-situ*" force feedback. This is opposed to using vibrotactile feedback as a force surrogate, either continuously mapped to force, or serving to signal when some critical force is reached [50].

In vascular surgery, enhancing the sense of touch can help clinicians differentiate between veins and arteries during deep dissection, helping protect the patient from intraoperative errors stemming from inaccurate recognition of the anatomy. With appropriate force magnification, for example, tactile cues related to the patient's pulse may be better detected by the clinician, ensuring intravascular lines are placed correctly. Lateral forces at the distal tip of an endovascular device may help clinicians navigate branch points in the vasculature. This is particularly important since the three dimensional anatomy is often imaged and presented in two dimensional slices (e.g., ultrasound) or projections (e.g., fluoroscopy), requiring clinicians to accurately register their knowledge of the 3D anatomy to the medical image. Endovascular access to the liver and gallbladder, where each patient's anatomy is unique, for example, may be aided by force feedback. Force feedback may also reduce the incidence of "back-walling" in vascular surgery, especially in pediatric patients, where vessels are much smaller than those in adults.

In ophthalmology, we believe that improving a surgeon's discrimination of force during anterior segment surgery could help prevent complications like radial tear and PCT by reducing unintended loads to the tissue. By amplifying distal forces to supra-threshold levels, we hypothesize that sensory-contingent responses will be facilitated, thereby leading to improvement of motor control. Specifically, surgical performance may be improved by augmenting the perception of minute variations in force that would have otherwise never been detected. Requiring the user to generate higher efferent forces to overcome the additional feedback forces employed to effect magnification could also engender improvements to distal force control.

1.3 THESIS OUTLINE

This dissertation is organized as follows.

Chapter 2 begins with a review of the principles underlying force sensing and actuation. We then present a brief overview of psychophysics, the study of the relationship between physical stimuli and mental phenomena, and haptics, the study of the sense of touch. Chapter 2 is concluded with a review of current technology that implements haptic feedback into tools for surgery.

Beginning with **Chapter 3**, we introduce the Hand-Held Force Magnifier (HHFM) concept, and a proposed human-in-the-loop control framework. In this chapter, we describe the HHFM Model-1, a one dimensional proof-of-concept prototype, capable of measuring and magnifying "push" forces, such as those encountered when inserting a needle into tissue. We also introduce the HHFM Model-2, a second proof-of-concept prototype able to magnify force in both "push" and "pull" directions. Concluding Chapter 3, we describe initial psychophysical experiments that measure the absolute force threshold, the just noticeable difference, and users' subjective experience of force and mechanical stiffness.

In **Chapter 4,** the HHFM Model-3 is described in detail. The HHFM Model-3 incorporates design lessons learned from the Model-2, resulting in a prototype that is smaller and more reliable. We also describe a number psychophysics experiments performed to examine human motor control. Two experiments develop a model of motor behavior following membrane puncture, a common task in surgery. The final two experiments characterize how one generates and sustains small forces under visual and haptic feedback.

In **Chapter 5**, we describe the HHFM Model-4, our first prototype capable of measuring and actuating force in two perpendicular axes. We describe initial prototypes and construction efforts supporting the HHFM Model-4, which is meant to implement a surgical scalpel.

Chapter 6 describes design of an inexpensive, one dimensional haptic renderer, constructed from an audio loudspeaker. We describe the implementation of virtual membrane models using this haptic renderer. Our new renderer will be used in the future to create other artificial tactile environments, on which we can perform psychophysical experiments.

In **Chapter 7**, we conclude the dissertation with a discussion of future work and directions.

2.0 LITERATURE REVIEW

In this chapter, we provide a brief background on the technological and psychological principles that are associated with the development of hand-held robotics to augment one's sensory abilities. The most common mechanisms of force sensing and actuation are reviewed in Section 2.1. Haptics, the study of the sense of touch, is reviewed in Section 2.2. The current state of surgical haptics is reviewed in Section 2.3.

2.1 MECHANISMS OF FORCE SENSING AND ACTUATION

A major consideration in the design of instrumented surgical tools is the limited number of sensing and actuation modalities available to the engineer. This is compounded by size and durability constraints required of sterilizable, hand-held surgical instruments. While there are many theoretically possible force measurement and actuation mechanisms, we focus here on the most common and simplest modalities.

2.1.1 Force Sensing

Measurement of an unknown force may be accomplished in a number of ways [51]:

1. Balancing the unknown force against the gravitational force of a known standard mass;
2. Measuring the acceleration of a body under influence of the unknown force;
3. Balancing the unknown force against an electromagnetic force;
4. Transducing the force into a fluid pressure, and measuring the pressure;
5. Applying the force to an elastic element and measuring the resulting strain;
6. Measure the change in precession of a gyroscope caused by an applied torque related to the unknown force;
7. Measuring the change in natural frequency of a wire tensioned by the unknown force.

Method 5 is the most common, and will be described in more detail in the following subsections. Methods 3 and 4 are also used occasionally.

2.1.1.1 Hookean Force Measurement

Hooke's law for ideal springs describes the force generated by, or applied to a spring undergoing displacement. Eqn. 1 describes this well-known relationship,

$$f = k_s * x \qquad \text{(Eq. 1)}$$

where k_s is the spring constant, a measure of the stiffness with units generally in Newtons per meter; and x is the displacement of the spring from its rest position, in meters.

The force f in Newtons may be signed, differentiating tensile from compressive forces. Assuming we know *a priori* the k_s constant for a given spring, force measurement using a spring

reduces to a length or distance measurement. Capacitive, optical, and electromagnetic positions sensors are all highly accurate, high-resolution methodologies for position measurement.

Expansion of Eqn. 1 to three dimensions can be achieved by defining the relationship using tensor notation,

$$\sigma = C\varepsilon \qquad \text{(Eq. 2)}$$

where σ is the stress, defined as the applied force divided by the body's cross sectional area, usually in Pascals, where 1 Pa is 1 Newton per meter-square; C is the 4th order stiffness tensor; and ε is the strain, defined as the ratio of the body's deformed length to its initial length, a unit-less measure. While bulk material force sensors may be possible in principle, in practice they can be difficult to use because of the difficulty in accurately determining the stiffness tensor, C over all degrees of freedom.

2.1.1.2 Resistive Force Measurement

Resistive approaches to force measurement rely on changes in the resistance of a wire in response to strain. These devices are therefore commonly called *strain gages*. Strain gages are constructed using thin wire resistors bonded to an elastic backing. The gage is in turn adhered to the surface from which strain measurements are desired.

The resistance of a wire is related to its physical dimensions by

$$R = \frac{\rho l}{A} \qquad \text{(Eq. 3)}$$

where l is the wire length, in meters; A is the cross-sectional area of the wire, in square meters; and ρ is the material resistivity, in ohm-meters [52]. Differential analysis of Eqn. 3 shows that changes in resistance are highly dependent on the physical properties of the wire,

$$dR = \left(\frac{L}{A}\right) d\rho + \left(\frac{\rho}{A}\right) dL - \left(\frac{\rho L}{A^2}\right) dA \qquad \text{(Eq. 4)}$$

As such, we can anticipate that strain gages will be very sensitive to effects like thermal expansion and contraction of the wire. To counteract this possibility, strain gages are constructed in a serpentine arrangement, so that changes in length in the longitudinal direction (indicative of strain in that direction) dominate the change in resistance, rather than changes in the cross-sectional area A due to thermal effects. In general, resistance decreases with compression and increases with tension. Behavior of a particular gage is determined by its *gage factor*, with the relative change in resistance described by

$$\frac{dR}{R} = S_\varepsilon \varepsilon \qquad \text{(Eq. 5)}$$

where S_ε is the unitless gage factor and ε is the strain. For most materials, $S_\varepsilon \approx 2$. In the context of strain gages, the resistance R is called the *nominal resistance* of the sensor, its resistance in the unstressed state.

Figure 3 shows the most common arrangement of strain gages, called the Wheatstone bridge [53]. The bridge is said to be "balanced" when

$$\frac{R_1}{R_2} = \frac{R_4}{R_3} \qquad \text{(Eq. 6)}$$

and therefore V_O, the voltage across the bridge, is zero. Replacing R_4 with the strain gage, which is essentially a variable resistor, allows V_O to change as a function of strain. However, this arrangement is still sensitive to temperature effects, as the coefficients of thermal expansion between the static resistors and the strain gage will be different. Therefore, changes in resistance

due to temperature changes will still affect *Vo*. By installing the gages in characteristic arrangements, measurements of resistance independent of temperature may be made.

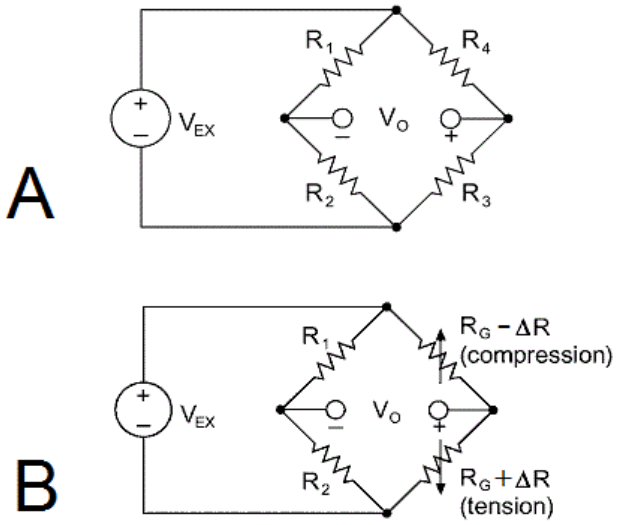


Figure 3. The Wheatstone bridge circuit in (A) Quarter-bridge and (B) Half-bridge arrangements. [53]

Figure 3B shows the Wheatstone half-bridge circuit, which enables measurement of strain independent of temperature. Because each gage in the bridge is affected by temperature similarly, the ratio of resistances in that "leg" of the bridge is maintained, and *Vo* stays at zero. Using the half-bridge arrangement also doubles the strain signal, as the two gages are generally arranged in such a way that, for a given strain, one is in compression, while the other is in tension.

2.1.1.3 Piezoelectric Force Measurement

The piezoelectric effect is a phenomenon characterized by the generation of an electric field in response to strain in a crystalline material. The most common piezoelectric materials are quartz (SiO_2) and the polymer polyvinylidene fluoride (PVDF). The piezoelectric effect originates from an uneven distribution of charge with strain in the material's crystalline lattice. Piezoelectric sensors are often assembled by placing electrodes across the material, forming a capacitor-like system with the crystal between the electrodes. The relationship between charge and force for a piezoelectric material can be given as

$$Q_x = d_{11} F_x \qquad \text{(Eq. 7)}$$

where Q_x is the charge generated due to force in the x direction; d_{11} is the piezoelectric coefficient in the direction of the applied force and orthogonal to its face; and F_x is the applied force in the x direction [52].

Therefore, the voltage generated by the piezoelectric system modeled as a capacitor is given by

$$V = \frac{Q_x}{C} = \frac{d_{11}}{C} F_x$$

$$= \frac{d_{11} l_c}{\kappa \varepsilon_0 A} F_x \qquad \text{(Eq. 8)}$$

where κ is the dielectric constant of the crystal; ε_0 is the permittivity constant, with value 8.8542×10^{-2} C^2/Nm^2; l_c is the distance between the capacitor plates; and A is the area of the electrodes.

Piezoelectric force sensors are typified by small displacements and an inability to maintain a static signal with sustained forces. This latter problem is particularly troublesome in our application, and led us to choose the strain gages to measure force instead. It should be noted that piezoelectric transduction works in both directions. Voltage across a piezoelectric material results in a physical deformation. Some devices, like the Squiggle motor (Newscale Technologies; Victor, NY), utilize this reverse relationship to actuate motion. One drawback to these actuators, however, is that very large voltages (100 – 300 V) are required to produce appreciable deformation.

2.1.1.4 Measurement Errors

Two goals paramount to measurement science are repeatability and predictability, qualities that are often marred by the properties of real materials in an imperfect world. The terminology surrounding common types of measurement errors has been largely standardized in engineering practice. This short glossary describes the most common definitions of measurement errors. Fraden has published an excellent text on sensors and sensing [52]:

Hysteresis – a hysteresis error is the deviation in the sensor output, for a specified input, when approached from opposite directions. For example, increasing a load on the sensor to a specific point will yield an output voltage different from the voltage obtained by decreasing the load to the same point. Hysteresis is commonly described as "path dependency."

Drift – drift describes the short- and long-term stability of a signal. Short-term drift may be bi-directional, and changes over timescales of minutes, hours, or days. Long-term drift arises from the aging of sensor materials, and is usually unidirectional.

Noise – noise is the stochastic disturbance of a signal that is largely irregular and unpredictable. There are many different types of noise, including Johnson noise, which arises due to the quantum nature of electric current and thermal variations, and is generally "white noise," meaning it is spread evenly across frequencies.

An important aspect of modern device design is signal conversion from the analog domain to the digital domain (A/D or ADC), and vice versa (D/A or DAC). While digital signals may be easily analyzed and stored using computers, they are quantized and only represent a particular level of resolution. Digital signals are commonly described using the number of bits used to specify the full-scale value of the signal. A 12-bit ADC, for example, can differentiate

between 2^{12}, or 4096, distinct voltage levels. Given a sensing range of 10 V, a 12-bit system has a voltage resolution of $10/4096\ V = 0.00244\ V = 2.44\ mV$. Digital signals are also discrete in the time domain, limiting the frequencies they can represent to half the sampling frequency, known as the Nyquist frequency.

2.1.2 Force Actuation

Similar to force sensing, force actuation may be accomplished using a number of strategies. Many actuation methodologies involve harnessing the electromagnetic force to generate a mechanical force. The electromagnetic force is usually generated through the interaction between coils of current carrying wire and permanent magnets. Thus the mechanism of action in all of these devices is the Lorentz force, which refers to the macroscopic force arising from moving point charges in a magnetic field. Motors, solenoids, and voice coils are the most common devices which utilize force generated in such a way.

2.1.2.1 Motors

An electric motor is a device which converts electrical energy into mechanical energy, usually as a rotation or torque. A motor has two constituent parts: the *rotor*, the moving part of the motor which delivers mechanical power, and the *stator*, the stationary portion of the motor. Depending on the type of motor, either rotor or stator may contain windings of current carrying wire, or be composed of permanent magnets [54]. The most common varieties are DC (direct current) motors, of brushed or brushless type; and stepper motors, a specialized type of brushless DC motor.

Depending on their type, motors require mechanical or electrical methods of switching the direction of current at precise timings to enable continuous rotation, a process called *commutation*. In contrast, stepper motors divide one complete rotation into discrete steps, each of which is actuated by individual electromagnetic pulses. Since stepper motors rotate in fixed angular increments, they may be controlled in an open-loop fashion, simplifying their implementation.

On a whole, however, motors are heavy and noisy, making them generally unsuitable for our application. Motors do not conveniently generate a known force, and require a screw to transduce linear translation from their rotation, introducing delays and hysteresis into the assembly. Therefore, we chose solenoids and voice coils as our principal force actuators.

2.1.2.2 Solenoids and Voice Coils

Solenoids and voice coils are electromagnetic actuators that move linearly, rather than through rotation like most motors [55]. Solenoids are devices that consist of a stationary coil of current carrying wire, through which an *armature* moves or generates force. The armature, comparable to the rotor in motors, is generally made of steel or other ferromagnetic metal. Solenoids commonly generate forces in only one direction, and require a mechanical means to return the armature to its initial position. This is usually accomplished using a small spring. Solenoids can be made to actively move in both directions by using a magnetized armature.

Voice coils are so called because they are often used in audio loudspeakers to generate sound. Compared to solenoids, where the magnetized armature moves relatives to a stationary coil, in voice coils, it is the current-carrying coil that moves relative to a permanent magnet. One key benefit of this arrangement is that the direction of current determines the direction of the force on the coil. Voice coils may therefore generate forces in two directions. Further, because

coils of wire are generally less massive than metal armatures, voice coils are able to move at much higher frequencies, making them ideal for audio applications. Like solenoids, a mechanical means is required to return the voice coil to its initial configuration. In addition, voice coils require an external means of maintaining concentricity with its magnet, such as a concentric bushing or bearing, or the paper cone of a loudspeaker.

Because the electromagnetic force varies with distance between the coil and the magnet, the force generated by both solenoids and voice coil actuators will depend strongly on the position of the moving armature or coil. Compared to solenoids, however, voice coils do not generate as much force, owing to limitations on the number of turns generally used in the wire coil.

2.2 PSYCHOPHYSICS AND HAPTICS

Haptics is the scientific study of the sense of touch, falling under the broad science of psychophysics, the study of the relationship between physical stimuli and mental phenomena [56]. The goal of psychophysics is to provide quantifiable and predictive models of psychological phenomena. Classic measures and methods in psychophysics include those to quantify the *absolute threshold* and the *just-noticeable-difference* (JND) [57]. Experimental methods in psychophysics also examine how one internally experiences a stimulus, for example, asking someone to subjectively rate the sweetness of various sugar solutions. Techniques that examine how one privately experiences a stimulus are collectively called methods of *magnitude estimation* [58].

2.2.1 Psychophysics

Our senses operate over extraordinarily wide dynamic ranges. The absolute threshold is a useful measure of individual sensory function, because it describes the smallest stimulus to elicit perception in 50% of presentations. For example, the absolute threshold of the retinal rod cell, the type of cell responsible for low-light vision, is sufficient to allow humans to see starlight through the intervening distance of many light years. To quantify this remarkable fact, Hecht and colleagues investigated the relationship between quanta and light perception. They found that human rod cells can detect the presence of just one photon when adequately dark adapted [59].

The absolute threshold is also an important diagnostic measure of sensory function. For example, in audiology, higher than average hearing thresholds may reveal abnormalities or deficits in sensory function. Indeed, the absolute threshold for hearing gives us the reference audio strength against which the calculation of sound level (in the logarithmic unit, decibels) is calculated. With respect to touch, thresholds may be measured in a number of contexts: the absolute threshold; the differential threshold, or the just-noticeable-difference (JND); and the two-point threshold. The absolute threshold and the JND together describe one's ability to perceive or differentiate the intensity of force(s) on the skin. The two-point threshold is a measurement of the spatial acuity of our tactile sense.

The absolute force threshold is the smallest force that one may reliably perceive, often defined as the smallest stimulus that elicits perception in 50% of presentations. While the 50% cutoff point is somewhat arbitrary, this definition takes into account the effects of attention and biological variability on perceiving subtle stimuli. Other commonly used cutoff points include 75% or 84%, the width of one standard deviation (SD) from the mean of a normal distribution.

The *method of constant stimuli* is one classic technique to measure the absolute threshold of one's senses. This method involves presenting a set of five to nine stimuli to participants, which range in intensity from imperceptible to almost always perceptible. Stimuli at each level are presented many times, and participants respond either "yes" if the stimulus is perceived, or "no" if the stimulus is not perceived. The proportion of yes responses with respect to stimulus intensity is an example of a *psychometric function*, which relates the physical stimulus intensity to the elicited psychological sensation.

The JND gives a measure of the sensitivity of our senses, which may change depending on the strength of the stimulus. Simply put, the JND is the smallest difference in a stimulus that elicits a differential perception. For example, it is relatively easy to detect the difference between a 2 kg weight compared to a 1 kg reference. Differentiating between a 101 kg weight and a 100 kg reference, however, is not as easy. German psychophysicist Ernst Weber found that the perceptible percent change from some reference – the JND – tends to be a constant [58]. This perceptual effect is expressed mathematically by Weber's law,

$$\Delta\Phi/\Phi = c \qquad \text{(Eq. 9)}$$

where $\Delta\Phi$ is the minimum perceptible difference between a stimulus and a reference; Φ is the intensity of the reference stimulus; and c is a constant.

Expanding on Weber's work, Gustav Fechner assumed the JND to be the basic unit of perception, reasoning that the JND is the "smallest unit of change" that elicits differential perception. Fechner therefore extended Weber's law to relate one's internal sensation, Ψ, to the physical stimulus through

$$\Psi = a \log \Phi \qquad \text{(Eq. 10)}$$

where a is some constant multiplier; and Φ is the intensity of the stimulus. Experimental evidence has since shown Fechner's assumption to be false, but nonetheless a logarithmic relationship between one's internal sensation and physical intensity has proven very useful in modeling psychophysical phenomena.

Fechner developed a number of psychophysical methods that are still used to measure sensory thresholds and JNDs today – such as the method of constant stimuli previously described – as well as the *method of limits* and *method of adjustment*. To measure a threshold using the method of limits, a series of stimuli that increase or decrease in intensity is presented to a study participant. When stimuli are presented in ascending order, participants report when they first perceive the stimulus; when stimuli are presented in descending order, participants report when they no longer perceive the stimulus. The average strength of these crossover points in perception (e.g., from no percept to clearly perceptible) is then taken as the threshold for perceiving the stimulus. The method of adjustment is similar to the method of limits, except it is the study participant who actively changes the stimulus intensity using predefined step sizes.

The JND may be measured using these methods by presenting two stimuli to the participant in each trial. One of this pair is a constant reference stimulus, against which the second stimulus is compared. The task, then, is to determine which stimulus is stronger. By varying the size of the difference between the reference and test stimulus using the methods of limits or adjustment, the JND may be determined.

S. S. Stevens went on to relate stimulus strength to the *perceived* intensity through a power function,

$$\Psi = a\ (\Phi - \Phi_0)^n \qquad \text{(Eq. 11)}$$

where Φ_0 is the absolute threshold strength of the stimulus; a is some constant multiplier; and n is an experimentally fit power [60]. Stevens' law (Eqn. 11) is used to fit rating data, as obtained from the *method of magnitude estimation*. In magnitude estimation, participants are presented with a number of stimuli of varying intensity (usually all supra-threshold), and participants are asked to assign a numerical score to a stimulus. The only restriction is that subjectively stronger stimuli should be assigned larger numerical values. The power relationship among stimuli (e.g., electric shock, sweetness) can vary widely: for electric shock $n = 3.5$, whereas for sweetness $n = 0.8$ [58].

2.2.2 Anatomy and Neurophysiology of Touch

The sense of touch, as we deal with it in everyday life, is actually composed of two major classes of sensory information: the *cutaneous* and the *kinesthetic* senses. The kinesthetic sense, commonly known as proprioception, derives from populations of mechanoreceptors embedded amongst the body's musculature and tendons – the Golgi tendon organs and the muscle spindles [21], [61]. The kinesthetic sense provides information about how one's body is arranged in space, and the forces generated by, or acting on, the various parts of the body.

Golgi tendon organs provide information about the force generated by a muscle during contraction. These mechanoreceptors are positioned "in series" between tendons and extrafugal muscle fibers, the force-producing components of the muscle. Correspondingly, with increasing muscle force, Golgi tendon organs fire at higher frequencies. Muscle spindles are small capsules that contain thin intrafusal muscle fibers, and are innervated by two types of mechanoreceptors, the primary and secondary spindle receptors [62]. Muscle spindles are found "in parallel" with extrafusal muscle fibers, and are responsive to changes in muscle length. The primary and

secondary muscle spindles respond preferentially to different aspects of the muscle length, however. The response of primary spindle receptors varies with the velocity and acceleration components of muscle contraction, with their discharge rate increasing with contraction velocity. Primary spindle receptors therefore encode information about the velocity of muscle contraction, integrating to provide a sense of limb movement. Secondary spindle receptors, on the other hand, discharge at rates that vary with static muscle length, directly encoding information about limb position.

The cutaneous sense, on which we will focus most of our attention, derives from mechanoreceptors embedded within the skin. The skin is, after all, the largest organ in our body, and correspondingly, the cutaneous sensors are the most numerous. The highest density of tactile sensors is found in the glabrous, or non-hairy, skin, found principally on our fingers and the palms of our hands. The rest of body is hairy skin, which has much lower densities of tactile sensors and therefore reduced sensitivity compared to glabrous skin. **Figure 4** shows how the absolute threshold and two-point threshold vary over different locations on the body [63].

Table 1. Cutaneous receptors in the skin

	RAPIDLY-ADAPTING	**SLOWLY-ADAPTING**
TYPE 1	RA 1 – Meissner corpuscle	SA 1 – Merkel cell
TYPE 2	RA 2 – Pacinian corpuscle	SA 2 – Ruffini ending

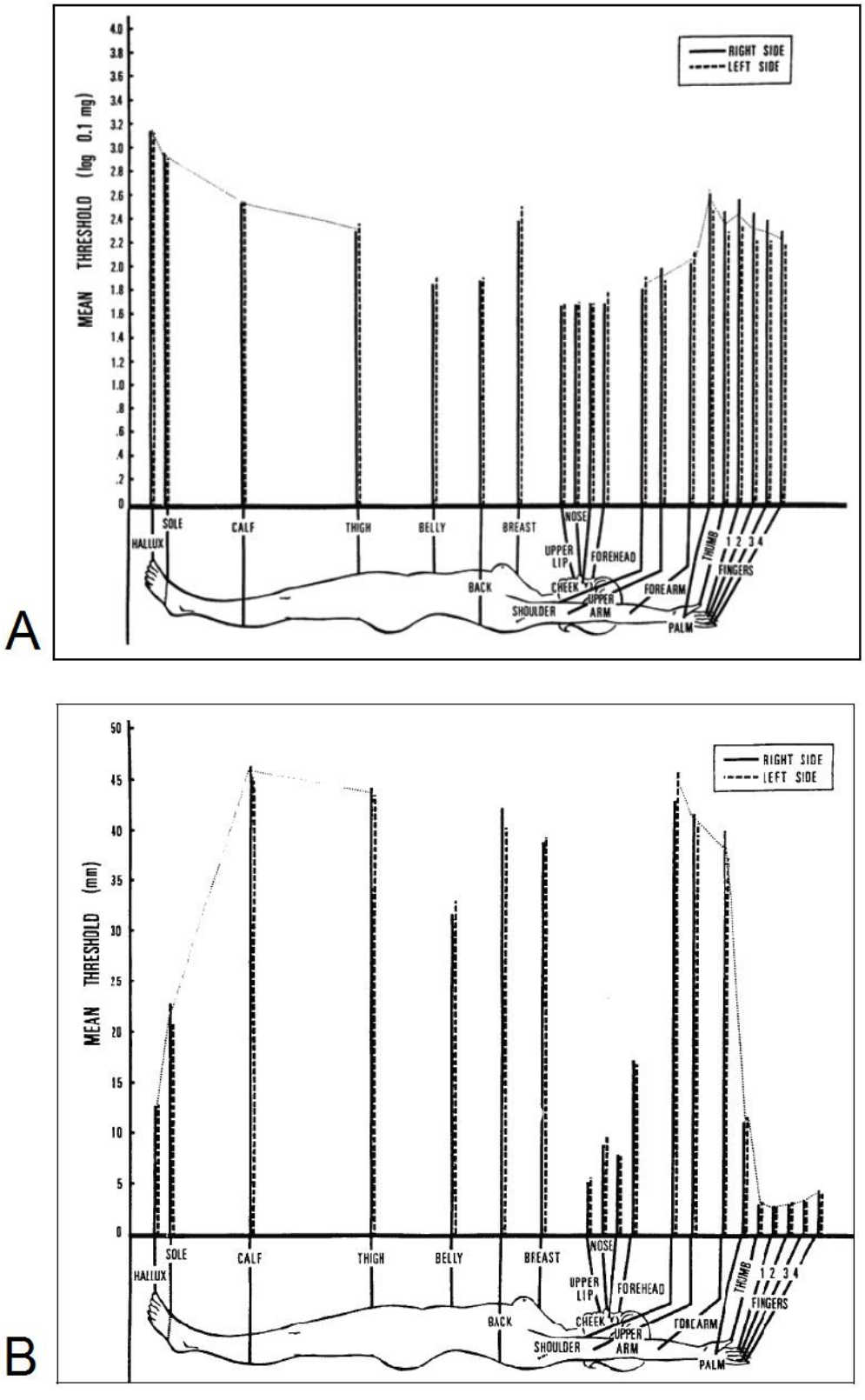


Figure 4. Variation of the (A) absolute force threshold and (B) two-point threshold over the body. [63]

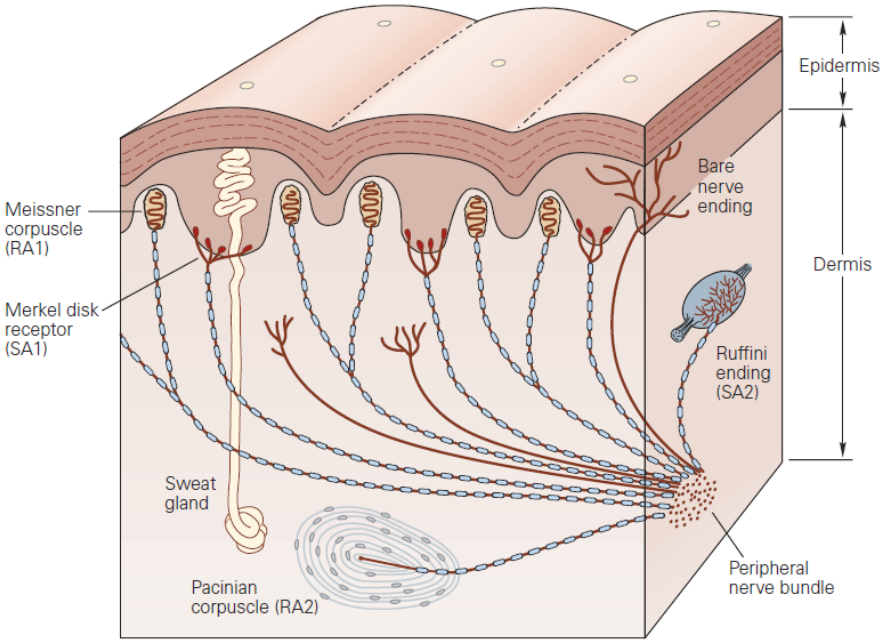


Figure 5. Cutaneous mechanoreceptors in skin. [64]

Cutaneous receptors, listed in **Table 1**, are classified as either Type 1 or Type 2, with each type having populations of *rapidly-adapting* (RA) or *slow-adapting* (SA) receptors [64]–[66]. RA receptors are also called fast-adapting (FA). Generally, RA receptors respond best to the onset or cessation of a stimulus, while SA receptors fire in response to sustained stimulation. Type 1 receptors are found near the superficial layers of the skin, and are the most numerous receptors. Type 2 receptors, on the other hand, are found in deeper layers of skin, and thus respond to stimulation over larger areas of the skin. **Figure 5** shows a schematic of the typical distribution of cutaneous sensors within our skin [64].

RA 1 receptors, also called *Meissner corpuscles*, are best stimulated by low frequency vibrations, between 5 – 50 Hz, such as those arising from exploration of textured surfaces or lateral frictional forces encountered while gripping a heavy object. RA 2 receptors, called the *Pacinian corpuscle*, are found deeper in the dermis, and fire preferentially to vibrations in the

range 50 – 700 Hz, with peak stimulation at 200 Hz [67]. These high frequency vibrations typically originate from contact between two objects or surfaces. SA 1 receptors, known as *Merkel cells*, respond to sustained stretch of the overlaying skin, aiding in texture and pattern recognition. SA 2 receptors, the *Ruffini endings*, also fire in response to lateral skin stretch as well as sustained downward pressure. There is also some evidence that SA 2 receptors may contribute kinesthetic information, as sustained skin stretch can be informative about limb position [68].

All of this cutaneous sensory information is relayed to the brain through the dorsal horn of the spinal cord in large tracts of *afferent* (incoming) nerve fibers [69]. (In contrast, bundles of motor neurons are said to be *efferent* – outgoing – fibers.) The first synapse in the sensory pathway occurs in the medulla, at about mid-neck level, in the *gracile* and *cuneate nuclei*, so

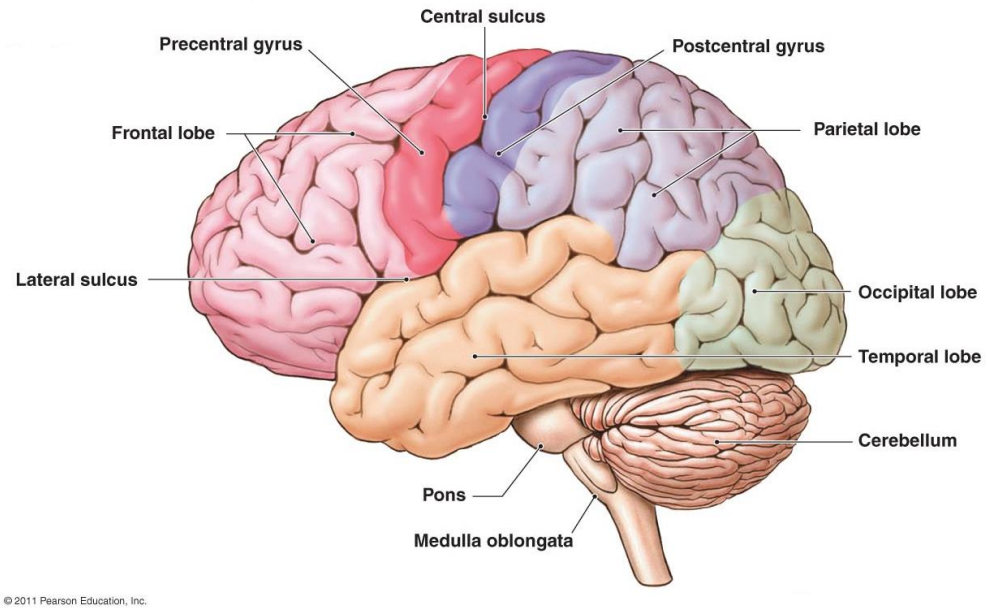


Figure 6. Lateral view of the cerebral hemispheres, highlighting the precentral (red) and postcentral (purple) gyri. Adapted from [70]**.**

called because the afferents originate from the legs or arms, respectively. This means, remarkably, there are single, continuous neurons that span almost the entire length of our bodies: the sensory afferents from the soles of our feet. Second-order neurons then cross to the opposite cortical hemisphere at the *medial lemniscus* and synapse in the *ventral posterolateral* and *ventral posteromedial* nuclei of the thalamus. Third-order neurons then terminate in somatosensory cortex in the parietal lobe, most notably in the *postcentral gyrus* of the brain. Sensory information is therefore processed in the contralateral cortical hemisphere. The pathway just described is called the dorsal column, or the *medial-lemniscal pathway*. This pathway differs from the *anterolateral pathway*, which contains afferents for temperature and noxious (painful) stimuli. While both tracts travel up the spinal cord to the thalamus, the two pathways remain separate and segregated [65].

The postcentral gyrus is located just posterior to the central sulcus, as indicated in **Figure 6** [70]. In this strip of cortex, there is a complete sensory "map" of the entire body. This can be visualized in the well-known "homunculus" of Penfield which illustrates the respective size of cortical regions devoted to receiving and processing sensory information from specific parts of the body. As depicted in **Figure 7**, we see a somatotropic organization of the cortex, and high prioritization of sensory information in the hands and face, the principal locations of glabrous skin, as previously discussed [71].

Kinesthetic information is also processed in the postcentral gyrus. Kinesthetic afferents are carried up the dorsal spinal cord, and branch in the brainstem. One branch travels to the cerebellum, while the other travels up the thalamus and to the postcentral gyrus with cutaneous afferents. By measuring the various joint angles of one's body, the brain is able to understand the

pose, or position, of one's body in space. The brain uses this information about one's pose as input into our finely tuned motor control centers.

The somatosensory cortex is adjacent to the primary motor areas of the brain, which are focused mainly in the *precentral gyrus*, anterior to the central sulcus. Motor intent is commonly thought to be initiated principally in the precentral gyrus, also known as the primary motor area, or M1 [72]. Organization and planning of voluntary movement is complex. As part of dynamic, adaptive loops, motor commands are refined in midbrain structures called the *basal ganglia*, before they descend through the thalamus and the spinal cord. Additionally, efferent copies of the motor command are carried to the cerebellum, where ongoing motor actions are compared to their sensory expectations. It is believed that the cerebellum enables high speed, closed-loop control over an individual's movement through dedicated cortico-cerebellar-cortico loops. It is well known that cutaneous input helps drive feed-forward control of one's grip, for example [73].

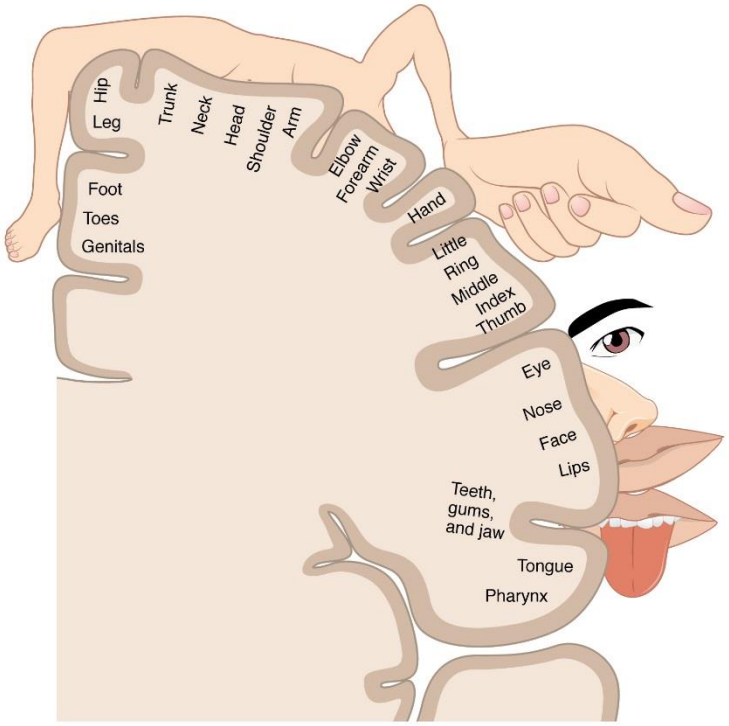


Figure 7. Penfield's homonculus depicting the relative area of sensory cortex devoted to processing afferent information from differing parts of the body. [71]

Motor neurons descend the brainstem in the ventral spinal cord, and initiate muscle contraction at a special synapse called the neuromuscular junction. Depending on the neuron at hand, each fiber can activate a wide number of muscle units at once.

2.2.3 Psychophysics of Touch

Measurements of the absolute threshold and JND can be affected by experimental design. In a classic experiment, Weinstein and colleagues measured the absolute threshold using devices called *von Frey hairs* to apply minute forces to the fingertips [63]. Von Frey hairs are thin fibers that are applied to the skin to produce a calibrated force at a single point on the skin. Using this methodology, Weinstein found that males could detect forces as low as 0.54 mN. Females were even better, able to detect 0.19 mN at the fingertip.

Other studies have measured the absolute force threshold, albeit using different methods [74]. Dosher and Hannaford found that, with tangential force pulses applied to the fingertip (50 – 150 ms), the absolute force threshold was measured as 22.5 – 28.9 mN [75]. In another study, Mesa-Munera and colleagues found that the absolute threshold was 247.8 mN when using a Phantom haptic renderer to detect force while drawing circles [76]. Evidently, the testing paradigm, in addition to factors like attention, can significantly affect one's absolute force threshold.

As we explore the environment using our sense of touch, we use the combination of our kinesthetic and cutaneous senses to drive our understanding of its physical parameters [77]. An object's stiffness, for example, makes use of information related to the force required to deform it, as well as the distance through which it is deformed. Research has shown that the JND for stiffness is between 8–22%, depending on experimental design [78]. The JNDs for force

discrimination and limb position estimation in isolation, however, are much lower. For force discrimination, the JND was found to be between 7–10%, while for limb position, the JND is just 5–8%.

In addition to absolute thresholds and JNDs, our sense of touch may also be characterized by its ability to discriminate spatially. One such measure is the two-point threshold, the minimum distance between two points on the skin such that the points are individually discriminable. A small two-point threshold corresponds to high spatial acuity, as is the case in the skin of our fingers and forehead. The two-point threshold is also a clinically relevant measure when evaluating recovery following nerve injury and repair [79]. On the fingerpad, the mean two-point threshold is generally less than 5 mm, whereas the threshold can be as high as 40 – 50 mm on the upper arms or thigh (see **Figure 4B**) [58], [63].

2.2.4 Haptic Renderers

A natural extension to studying the sense of touch is the desire to create virtual haptic environments with which participants in psychophysical experiments can interact. Virtual environments are desirable because, unlike physical objects, they do not degrade, and therefore allow subsequent participants to interact with the same object under similar experimental conditions. In the context of haptics, *rendering* refers to the process by which sensory stimuli are presented to a user to convey information about a virtual object [80]. (The term is borrowed from computer graphics, where it relates to the process of generating an image from a two or three dimensional model.)

A number of haptic rendering technologies are currently available to consumers and researchers. Some of the most popular devices are those that generate feedback forces through

mechanical linkages, such as the Force Dimension Omega (Nyon, Switzerland) or Novint Falcon (Albuquerque, NM). Broadly, these devices consist of a stationary base which supports a moveable component, upon which forces are generated in response to interaction by an external user. These devices generate forces – typically in the 3 translational degrees of freedom (DoF) – on a control ball or stylus, using motors acting on parallel linkage mechanisms. The Novint Falcon is a consumer device that can generate up to 8.9 N force over a cubic workspace with dimensions 10.2 cm × 10.2 cm × 10.2 cm. The Force Dimension Omega.6 can generate up to 12 N over a workspace 16 cm in diameter and 13 cm in length. Both the Falcon and the Omega may be programmed in C/C++ using proprietary Haptic and Robotic software development kits (SDK's), or using the open source haptics application program interface (API), Chai3D. Another popular device in the research community is the Geomagic Touch (formerly the Sensable Phantom Omni). Mechanical arms on the Geomagic Touch enable movement of a front facing stylus in all 6 DoF (including 3 degrees of rotation), although force feedback is restricted to the 3 translational DoF. The Geomagic Touch is capable of generating up to 3.3 N force over its 16 cm × 12 cm × 7 cm workspace. The Geomagic Touch family of devices may be programmed in C/C++ using the Geomagic OpenHaptics toolkit.

Given the unique nature of haptic renderers, which combine mechanical and electrical characteristics, simple haptic interfaces have been constructed as educational tools for engineering curricula. For example, Okamura and colleagues have developed the "haptic paddle," a single axis force feedback joystick, for use in dynamics systems courses at Stanford University [81]. Users can move a paddle, shaped like an inverted pendulum, side to side over a ±35° range of motion. Feedback forces are generated by a DC motor in either a capstan or friction drive arrangement, in response to changes in paddle position and velocity. The haptic

paddle can generate up to 7.5 N force on the handle. Gassert and colleagues have adapted the Stanford haptic paddle design for use at ETH Zurich, adding a number of sensors and USB data collection for ease of use in an educational laboratory setting [82].

At the University of Michigan, Gillespie and colleagues have designed two haptic interfaces for use in undergraduate mechanical engineering and electrical engineering courses. These two haptic renderers, the iTouch motor and "The Box," are single axis haptic interfaces that make use of voice coil actuators and brushed motors, respectively, for force generation [83]. Similarly to the haptic paddles at Stanford and ETH Zurich, the iTouch can move side to side over a 30° range of motion, generating up to 0.202 Nm torque on the handle. "The Box" is a larger haptic device, driven by a motor, sprocket, and chain mechanism, and can generate up to 5.4 Nm torque on a wheel that is held by the user.

The haptic rendering devices so far described are called *grounded* devices, because the feedback force is generated between the movable part (the "manipulandum") and the floor or table on which the renderer rests. *Ungrounded* haptic renderers also exist, where feedback forces are generated instead between different parts of the user's anatomy. These devices are often described as *exoskeletons*, or otherwise known as *wearable* haptic devices. For example, a number of finger mounted devices have been developed which display contact forces directly to the fingerpad [84], [85]. Depending on the device, wearable haptic renderers can even generate feedback forces in multiple DoF [86]. However, due to ergonomic and space constraints related to mounting devices on the body, such ungrounded exoskeleton type devices may not generate as much force as their grounded counterparts. Indeed, because grounded devices are mounted rigidly, they can utilize that advantage to generate much larger sustained forces than their ungrounded counterparts.

2.3 HAPTICS IN SURGICAL ROBOTICS

Robotic surgeries have been a dream of science fiction for many years. Automated surgeons are desirable because a robot does not, for example, exhibit physiological tremor. Unlike humans, robots do not fatigue over long periods of time, and can draw upon years of medical research without exhausting their memory. Increasingly dexterous robots, the connectivity offered by the Internet, and advances in research areas such as machine learning and computer vision are currently pushing to make these dreams a reality. Still, robots in surgery have only recently come to the operating room and fully automated robotic surgery is still far from a clinical reality.

Surgical tools that navigate relative to anatomy, for use in neurosurgery primarily via pre-acquired 3D images, were some of the earliest surgical robot archetypes [87], [88]. Accurate and steady tool positioning is crucial in these situations, where a centimeter error in position on the skull surface may correspond to drastically different regions of the brain. These systems also find use in orthopaedic applications, for example, in knee replacement, because, like the skull, the bones of the lower limb may be immobilized, enabling highly accurate procedures. In these cases, however, the robot only provides positioning assistance – the surgeon still ultimately performs the procedure.

Telerobotic systems, in which the clinician operates a slave robot from a master terminal, are also popular research topics, but only Intuitive Surgical's (Sunnyvale, CA) *da Vinci* robot has thus far passed through regulatory approval and into clinical use for MIS, first in 2000 [89]. Intuitive's *da Vinci* features a view of the surgical field, displayed to the clinician in stereoscopic 3D at a master terminal. Grasping controller arms in the master terminal, surgeons move MIS tools in the slave robot at the patient's side with high-fidelity and tremor filtration. Mechanical design of the MIS tools enables the surgeon to control movement of end-effector jaws naturally.

Wire pulley mechanisms in the *Endowrist* instruments can move each of the 2 tools in a full 7 DoF (the 6 degrees of translation and rotation, plus the opening/closing degree for grasping tools like tweezers, forceps, or needle holders). The combination of stereoscopic visual feedback and intuitive control mappings enable the surgeon to interact with target tissues in 3D space with considerable skill and confidence after appropriate training.

The RAVEN surgical robotic system is an open source telesurgical robot architecture originally developed at the University of Washington, and later by a consortium led by the University of California, Santa Cruz and the University of California, Berkeley [90]–[92]. This robot also utilizes cable-driven mechanisms to drive two arms in 7 DoF (6 DoF positioning and 1 DoF grasping). However, compared to the *da Vinci*, in which the slave-side robot is directly controlled by the surgeon from a master terminal, RAVEN has been designed to perform surgical tasks autonomously while under supervision of a clinician [93], [94]. This type of control opens up the possibility of long-distance surgery, in which time delays cause instability when using the traditional master-slave control.

It should be noted that telerobotic systems also suffer from the haptic problem. Because motion in the slave robot is driven by movement of controllers at the master terminal, haptic feedback is difficult to implement. Such feedback forces may in turn lead to movement of the master controls and subsequently, the slave robot. Such instabilities in telerobotic systems have shown to be due to time delays and quantization effects [95]. Thus, there are fundamental and formidable control problems that must be resolved before these haptic feedback systems may be used in telesurgery. A number of architectures have been proposed to address particular aspects of these problems. Studies have shown, for example, that force feedback during teleoperation can reduce the force applied by the robot during representative tasks like blunt dissection [96].

A number of devices are under development that address the haptic problem in surgery with hand-held tools, rather than through large telerobotic systems. Some provide vibrotactile feedback, while others directly apply forces to the surgical tool. Others convert force information so that it may be perceived by the other senses, such as audition or vision, a technique called *sensory substitution*. Braille is a long established form of sensory substitution, in which normally visual information (text) is presented through tactile stimuli. Sensory substitution (e.g., force-responsive visual feedback) has been shown to reduce the incidence of broken sutures amongst non-trained *Da Vinci* users, as well as reduce the peak force and standard deviation of forces [97]. Skin-deformation-based substitution techniques have also shown reductions in task performance time and peak force in teleoperation [98]. Still others have pioneered a technique called *sensory subtraction*, in which cutaneous, but not kinesthetic information, is used for feedback in telesurgical applications [99]–[101]. Because we are principally interested in hand-held tools, we review these technologies further in the following sections.

2.3.1 Micron

Micron is a prototype surgical tool developed by the Surgical Mechatronics Laboratory at Carnegie Mellon University. While Micron does not generate force feedback *per se*, it is one of the first technologies to address the sensory and motor limitations posed by microsurgery. The goal of the Micron project is to reduce physiological tremor through active actuation of the tool tip [102], [103]. Specifically, Micron has been developed with focus on applications in retinal surgery.

Tremor is defined as any involuntary hand movement that creates position error. Previous work has found root-mean-square (RMS) tremor during epiretinal membrane peel to be 0.182

mm [104]. Tremor of even a few hundred microns is significant because vitreoretinal surgeons routinely manipulate tissues which are only 10 µm thick, such as the internal limiting membrane of the retina. Tremor while performing puncture or peeling tasks in the retina risks tearing the membrane, or even detaching the retina itself. In frequency space, physiological tremor has been isolated to movements within a band between 7 and 13 Hz.

Micron is a completely hand-held tool that utilizes piezoelectric motors to drive movement of the tool tip, in multiple DoF, relative to the tool handle. The pose of the instrument is measured using optical tracking, obtaining positioning information with accuracy up to 4 µm. The piezoelectric motors are used to position the tool tip in 3 DoF, and are capable of generating up to 1 N of force, more than enough to resist and compensate for tool deflection due to tremor. Closed-loop control and specialized tremor filters minimize position error in the tool tip in real time. The range of motion achieved by the piezoelectric actuators in the first Micron prototypes was approximately 1 × 1 × 0.5 mm, centered on the current tool position [105]. Later iterations of Micron expanded the workspace up to 3 × 3 × 0.8 mm [106].

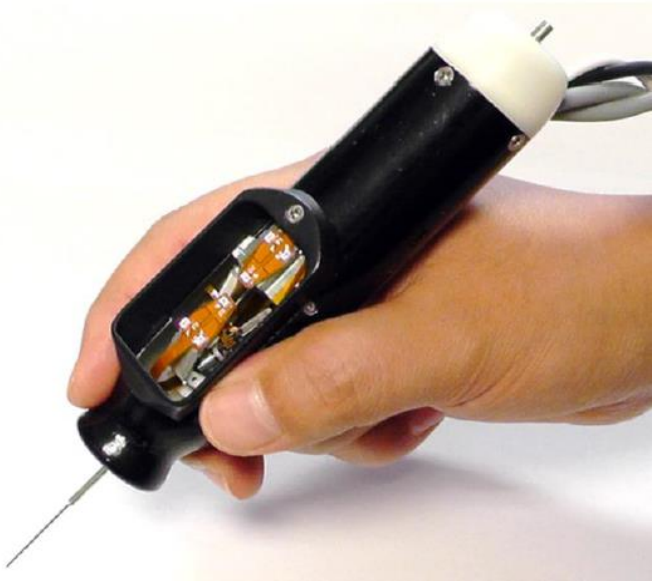


Figure 8. Micron, 6 DoF handheld micromanipulator. [108]

Characterization of the device has shown that Micron is effective in canceling unwanted movements while permitting those that are intentional [107]. The preliminary tasks include tracing lines and circles, which is representative of microcannulation in retinal vasculature. Micron has been used in *ex-vivo* cannulation tasks, and some preliminary work has been done using Micron in membrane peeling [105], [106]. The most recent iteration of Micron can move the tool tip in all 6 DoF, in a cylindrical workspace 4 mm in diameter and 4 mm in length [108]. This iteration is shown in **Figure 8**. The 6-DoF Micron has been used in tool stabilization to aid in acquiring optical coherence tomography (OCT) images, as well as for laser guided retinal ablation surgery.

2.3.2 Microtactus

Microtactus is a hand-held tool that provides vibrotactile feedback to the hand during minimally invasive exploration of orthopedic surfaces, developed by the Haptics Laboratory at McGill University [109], [110]. Exploration of cartilaginous surfaces is done using blunt instruments to minimally invasively identify anomalies in the tissue. As described earlier, in MIS, friction between the tool and the point of entry into the patient's body confounds tactile information from the tool tip, making the surface defects in hard tissues more difficult to identify. In Microtactus, a 90-degree blunt dissection tool is instrumented with an accelerometer to measure vibrations at the tool tip, perpendicular to the long dimension of the tool. In the handle, a rare-earth magnet suspended in a coil moves parallel to the long dimension of the tool, generating vibrotactile feedback. Because the sensor and actuator are orthogonal to each other, sensor input is decoupled from device output, avoiding instability.

Psychophysical experiments showed the efficacy of Microtactus in helping identify cuts on otherwise smooth surfaces. While deep cuts were identifiable with or without feedback, identification of shallow cuts was improved with feedback. Both vibrotactile and auditory feedback were tested, with no significant difference found between the two feedback mechanisms. In the end, however, because an accelerometer drives vibrotactile input, Microtactus only measures transient stimuli, and is therefore not well suited for sustained force feedback during surgery.

Because auditory feedback transduces force information into sound, this is a type of *sensory substitution*. Because the surgeon no longer has the tool actuating forces to feel, converting the force information into another sense can potentially increase the cognitive load associated with using the tool. Further, auditory cues may not be well suited for the operating theatre, where many other devices already use one's auditory channel to convey information.

2.3.3 Steady Hand Robot

The Steady Hand Robot (SHR), developed by the Computer Integrated Interventional Systems Laboratory at Johns Hopkins University, is one of the first technologies to provide sustained force feedback during surgery, directly addressing the haptic problem in microsurgery. The SHR has been designed to operate under *cooperative control*, which describes the paradigm in which both surgeon and robot *share* control over an instrumented surgical tool. In the cooperative control paradigm, as the surgeon uses the tool to perform a surgical procedure, the SHR actively generates force on the tool to enhance the user's sense of touch. In essence, this may be considered a type of *admittance control*, which generally describes robot actuation in response to an input force from the human operator [111]. Further, although the tool is manipulated by the

surgeon's hand, the SHR is a grounded robot, meaning feedback forces are generated between the sensorized tool and, ultimately, the table or ground on which the robot is mounted.

The SHR Eye Robot 1 (see **Figure 9**) is a surgical robot capable of generating force feedback in 5 DoF, three translations and two rotations, actuated with a mechanical remote center of motion (RCM), normally the point of contact or entry between the tool and the patient [112]–[114]. The 3 translational DoF are actuated through XYZ translational stages, where positioning is accurate to 2.5 μm. The remaining roll and tilt degrees are actuated using a rotary stage and slider-crank mechanism, respectively. The Eye Robot 1 has been used for microcannulation of the chorioallantonic membrane, an embryonic chicken tissue, which has been recognized as an reasonable surgical replicate of the human retina [115].

The Eye Robot 2 (see **Figure 10**) is the second generation of the SHR, capable of actuating force feedback in the same 5 DoF – three translations and two rotations [116], [117]. In this new iteration of the SHR, tilt is achieved through a six-bar linkage, while the remaining three translations and rotation are achieved through linear or rotary stages, respectively. The Eye Robot 2 has been used in membrane peeling simulations, implementing position and velocity

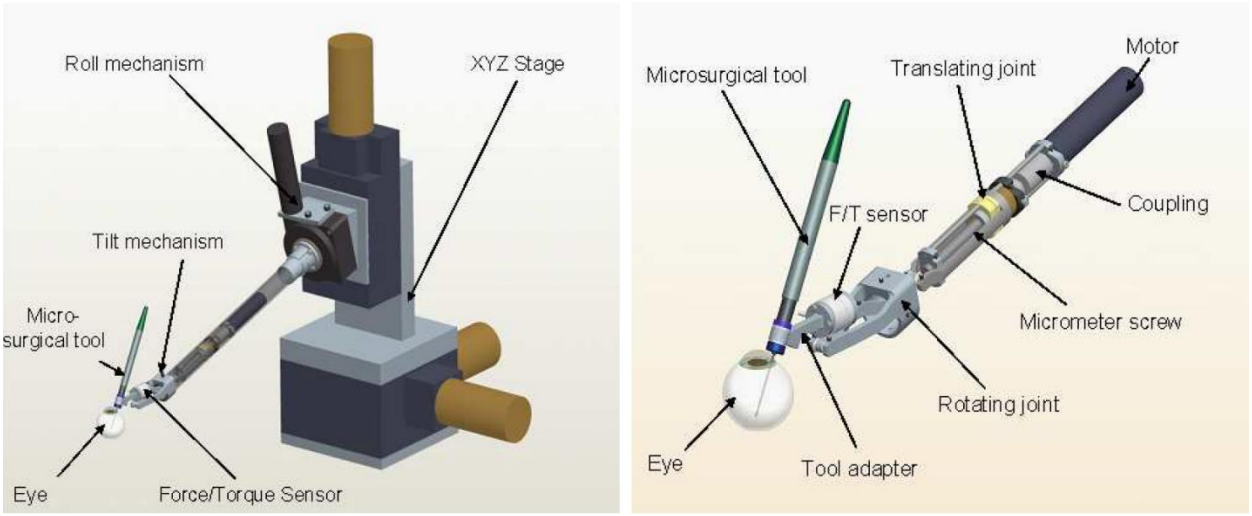


Figure 9. The Steady Hand Robot, Eye Robot 1. [113]

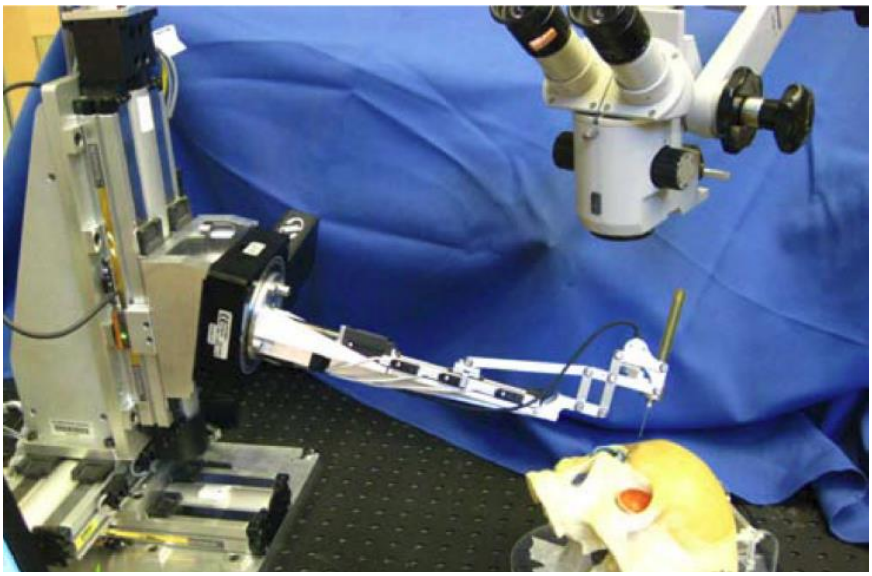


Figure 10. The Steady Hand Robot, Eye Robot 2. [117]

controls on the surgical tool. In this control paradigm, the surgeon is prevented from exceeding global limits by action by the robot. Force to audio sensory substitution has also investigated using this platform.

Major contributions by this group at Johns Hopkins include the development of miniature force sensors suitable for use in microsurgery. For example, an assembly of flexure beams and strain gages have been deployed in the Eye Robot 1, which is capable of sub-millinewton force resolution [118]. This sensor is only 12.5 mm in diameter and 15 mm in height, making it sufficiently small to be implemented into microsurgical tools. Fiber optic Bragg sensors have also been developed that exhibit sub-millinewton force resolution [119]. Bragg sensors utilize a grating within an optical fiber, the spacing of which changes with strain in the cable, changing the wavelength for light reflected back up the fiber optic cable. By integrating multiple fibers, sensors have been assembled to measure tool tip force in 3 DoF [120].

2.3.4 Pneumatic Haptic Feedback in Telesurgery

Another technology that directly addresses the haptic problem in surgery is a pneumatic system developed by the Center for Advanced Surgical and Interventional Technology at the University of California, Los Angeles. This system has been integrated into Intuitive Surgical's *da Vinci* robot, making it one of the few technologies to address the haptic problem in a telesurgical application. To this point, efforts to improve haptic feedback in telesurgery have largely been focused on providing either partial force feedback or vibrotactile feedback, or by using sensory substitution techniques such as colored visual overlays [121]. Indeed, meta-analyses on the effect of haptic feedback in teleoperation have found that vibrotactile feedback can improve task completion time significantly, but does not reduce applied forces or handling errors [122]. In the same exploratory analysis (where small sample sizes may preclude reproducibility), force feedback was found to reduce handling errors, task completion time, and applied force.

As mentioned previously, directly actuating forces on the master controllers of a telesurgical robot can contribute to system instability. The key to the pneumatic array approach is that no actuators directly act on the control graspers at the master terminal. Instead, an array of pneumatic balloons inflates in response to force at the slave-side robot. The pneumatic array deforms the fingerpad, using the master terminal controllers as a mobile ground upon which feedback forces are generated [123]–[125]. The actuator balloons are 3 mm in diameter, and arranged in a 3×2 pattern with 1.5 mm spacing. The optimal actuator configuration was chosen based on psychophysical experimentation [126]. The array is manufactured from polydimethylsiloxane (PDMS) polymer and cast using precision machined aluminum molds. The response time of the pneumatic display is 50 ms, and can therefore deliver haptic feedback at 20 Hz. The pneumatic actuators can generate up to 15 PSI (103.4 kPa).

On the slave-side robot, Cadiere graspers were fitted with small strain gages to measure the force applied to the tissue. Preliminary tests with this technology showed that grasping forces on force sensitive neoprene strips were lower when haptic feedback was enabled, compared to the no feedback conditions. Grip forces in peg transfer tasks were also lower with grip force feedback enabled.

2.3.5 Imperial College London Hand-Held Manipulators

At the Hamlyn Centre for Robotic Surgery at Imperial College London, a number of hand-held manipulators have been developed to provide haptic feedback during surgery. The first was a tool developed for use in micromanipulation, as shown in **Figure 11A** [127]. To generate feedback, a slider moves relative to the rest of the tool, and is contacted with the index finger while the tool is held in one's grip. A distal force sensor at the tool tip informs a PID controller, which controls an internal electromagnetic linear motor that moves the slider. The motor can generate up to 2.72 N, and is exceptionally small, weighing only 15 g. Using integrated Hall position sensors in the motor, the controller generates a change in position against the user's finger, which in turn produces the feedback force. In benchtop testing, force scaling factors as high as 15× were reported. Psychophysics experiments show that a user's absolute force threshold is reduced when using the tool.

Along the same lines as this first tool, another device was developed for motion compensation and force feedback [128]. This device is shown in **Figure 11B**. By motion compensation, it is meant that global constraints are placed on the position of the tool, so it does not stray away from a preoperative plan, or damage nearby structures. This concept is also known as "active constraints" or "virtual fixtures." Optical trackers are placed on the tool body

to measure pose and position. A linear motor is used to retract the surgical tool in response to contact with the virtual fixture. A force feedback display is constructed from a voice coil actuator and is contacted by the index finger while the user grips the tool. Haptic feedback is also delivered upon contact with the virtual fixture. A force sensor between the voice coil and the finger allows for closed-loop force feedback. Virtual fixture stiffnesses up to 0.475 N/mm were generated in benchtop testing.

A third device has been developed by the Hamlyn group, with applications to neurological dissection [129]. This device, shown in **Figure 11C**, generates vibrotactile feedback at predetermined force limits. A blunt dissector is rigidly connected to a machined tool shaft, and

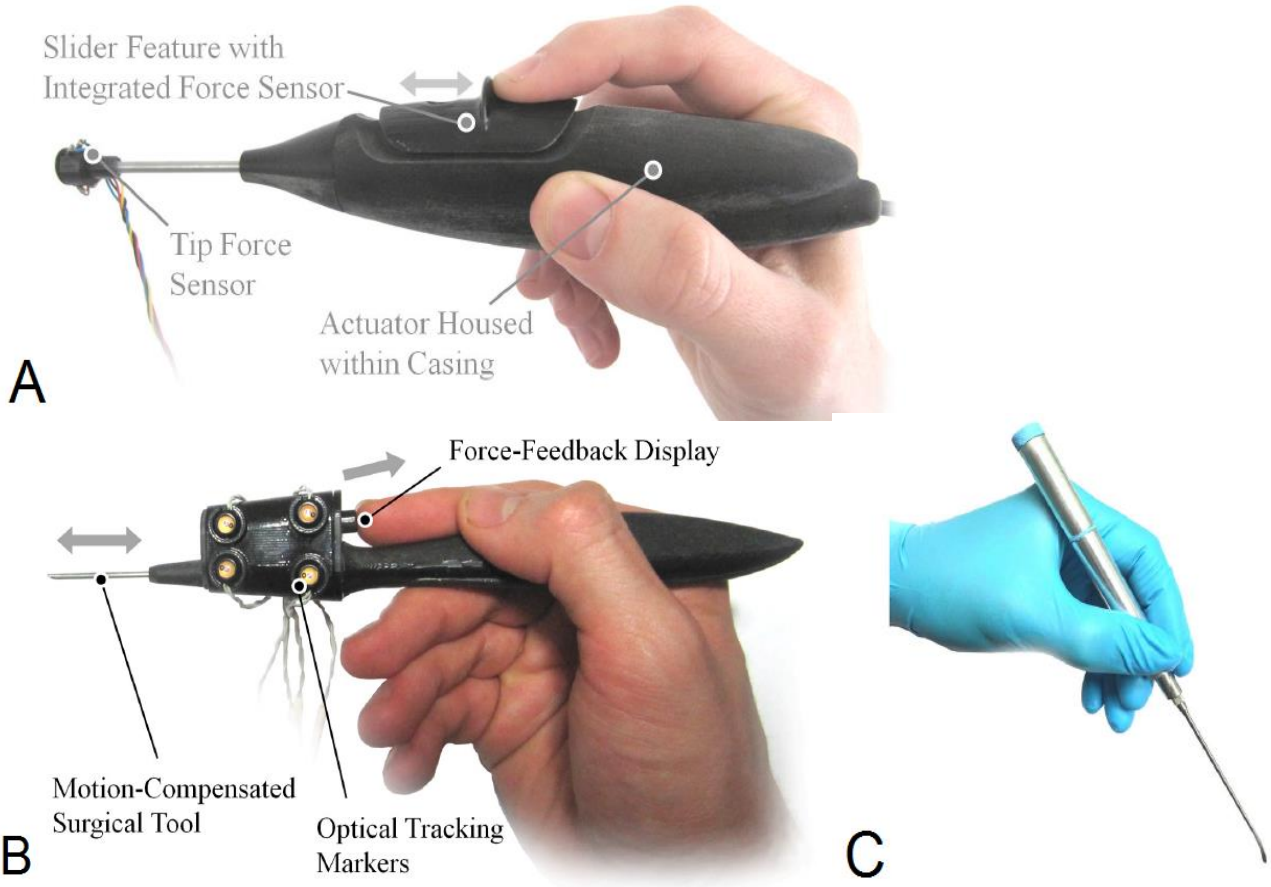


Figure 11. ICL (A) Slider micromanipulator (B) Motion compensated manipulator (C) Neurological blunt dissector. [127]–[129]

strain gages secured to the shaft faces. The tool measures forces tangent to the tissue plane, such as those encountered when retracting tissue sideways to expose deep structures. Vibrotactile feedback at 250 Hz is generated by an internal voice coil actuator connected to a spring-damper assembly. The device was used in a number of studies that examined the effect of vibrotactile feedback on the distal tool tip force in a bimanual peg transfer task, in isometric force generation, and in short cadaveric dissections. These experiments showed that vibrotactile feedback can significantly reduce the maximum applied force, and reduce the amount of time spent above a chosen critical force value of 0.3 N.

2.4 DISCUSSION

As we can see, the study of haptics brings together many disparate bodies of knowledge: from mechanical engineering and electrical engineering to human factors and psychophysics. Integrating haptic feedback into tools for use in surgery therefore also requires an understanding of how sensory information can influence motor control behavior. In the proceeding chapters, we introduce the Hand-Held Force Magnifier, a prototype surgical tool that provides sustained force feedback during delicate surgery. In support of our engineering development, we also conduct psychophysics experiments to investigate the internal relationships between sensory and motor behavior.

3.0 THE HAND-HELD FORCE MAGNIFIER

The Hand-Held Force Magnifier (HHFM) is a prototype surgical instrument that provides sustained haptic feedback to the user's hand and fingers. **Figure 12** illustrates this concept [130]. A brace anchors the back end of the tool to the palm, while the handle of the tool is gripped by one's fingers. By generating feedback forces between the handle and brace, *in-situ* force magnification is possible. One experiences this haptic illusion because the fingers match both the distal force and the feedback force. While the device is not grounded to the earth, it is grounded to the body, with the hand acting as a mobile platform from which forces are generated. This configuration classifies the HHFM as a "wearable" or ungrounded haptic renderer.

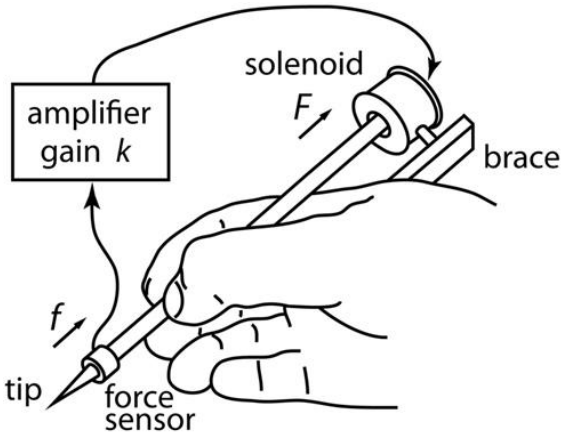


Figure 12. The Hand-Held Force Magnifier. [130]

We define the following tool-centric orientation terminology for the HHFM. "Axial" refers to forces and directions parallel to the long dimension of the tool. For example, "push" and "pull" forces act in the axial directions. Thus, push forces are those encountered when inserting a needle into tissue. Rotations about the long dimension of the tool are termed "axial torques." Perpendicular to the axial dimension, we define "left" and "right" forces as acting in the

"horizontal" axis, as would be encountered during surgical peeling tasks like CCC. The remaining axis is orthogonal to the first two, acting in the "vertical" axis, representing, for example, the "upward" tissue forces encountered when using a scalpel to cut into the tissue plane. We define "distal" aspects of the HHFM as those near the tool tip, whereas "proximal" indicates the end of the device closer to the hand and brace. For convenience, we generally center reference axes at the point the user grips the tool, as this is the confluence of haptic perception and action, the afferent and efferent pathways that feature prominently in our control framework.

We hypothesize that the HHFM will improve performance in surgery by improving the perception of forces encountered at the distal end of the tool, such that, with magnification, the net tissue-tool forces will be better controlled. In this framework (see **Figure 13**), efferent forces generated by the user during delicate surgery are normally impaired because one's internal sensorimotor loop is deprived of afferent information. Therefore it is the *perception* of force that we control, not the force generated by the user.

We believe that this method of operation will be particularly effective because of a psychological phenomenon called *distal attribution* [131]. The precise origin of distal attribution is still subject to debate, but it is generally agreed to be either "the direct perception of objects in our external environment," or an "inference based on afferent stimulation" [132]. Regardless of its origin, this phenomenon is ubiquitous in our everyday life. Using a hand-held tool (i.e., a pen) to interact with external objects, we "feel" the distal object or surface, as opposed to consciously "experiencing" the individual shear and normal forces in the fingerpad that arise from tool interaction. With respect to the HHFM, this means that the user perceives (or "attributes") the

feedback force as originating from the distal end of the tool, where the tip contacts the tissue, rather than from the proximal end, where the actuator is really producing force.

Further underlying our design effort is recognition that the clinician plays a critical part in the human-machine loop. We respect this executive role by enforcing no external restrictions on the surgeon's behavior. Compared to other robotic tools where constraints may enforce global rules on applied force or tool velocity, we believe that an open-ended control scheme will better present a perceptually seamless and a more intuitive experience when using tools like the HHFM. We do this by introducing all forces in a way that the operator can attribute them to having originated at the tool tip.

To implement this design philosophy will require extensive knowledge of how the user perceives force, and the interplay between sensory and motor control, such that a user's natural reactions are anticipated by our system and corrected as necessary. This effort is well worth it, however, because in the end, the surgeon is the ultimate judge of whether or not to incorporate a technology into their practice. The progress of medical technology is, after all, an exercise in controlled chaos, and clinicians are reluctant to gamble away their hard-earned collective knowledge with the advent of each new technology. Perhaps a new class of surgical technique is necessary before augmented reality surgical tools like the HHFM are adopted in the surgical theatre. These kinds of advances only come when a dedicated band of clinicians devote themselves to performing surgery with such tools. Therefore, it is prudent to design our tools such that they are extensible and adaptable as needed far into the future, rather than introduce hard limits based on our current (and imperfect) knowledge.

3.1 HAND-HELD FORCE MAGNIFIER CONTROL FRAMEWORK

Figure 13 shows a proposed human-in-the-loop control framework comparing traditional surgery to surgery with the *in-situ* force feedback, where we define the following variables:

- $F_e(t)$ is the efferent (output, or motor) force applied by the user onto the tool;
- $F_a(t)$ is the afferent (input, or sensed) force perceived by the user;
- $F(t)$ is the feedback force applied by the HHFM actuator onto the handle; and
- $f(t)$ is the force of the tissue on the tool, measured by the HHFM's internal sensor.

As depicted in **Figure 13A**, in traditional surgery, the operator's efferent force F_e is

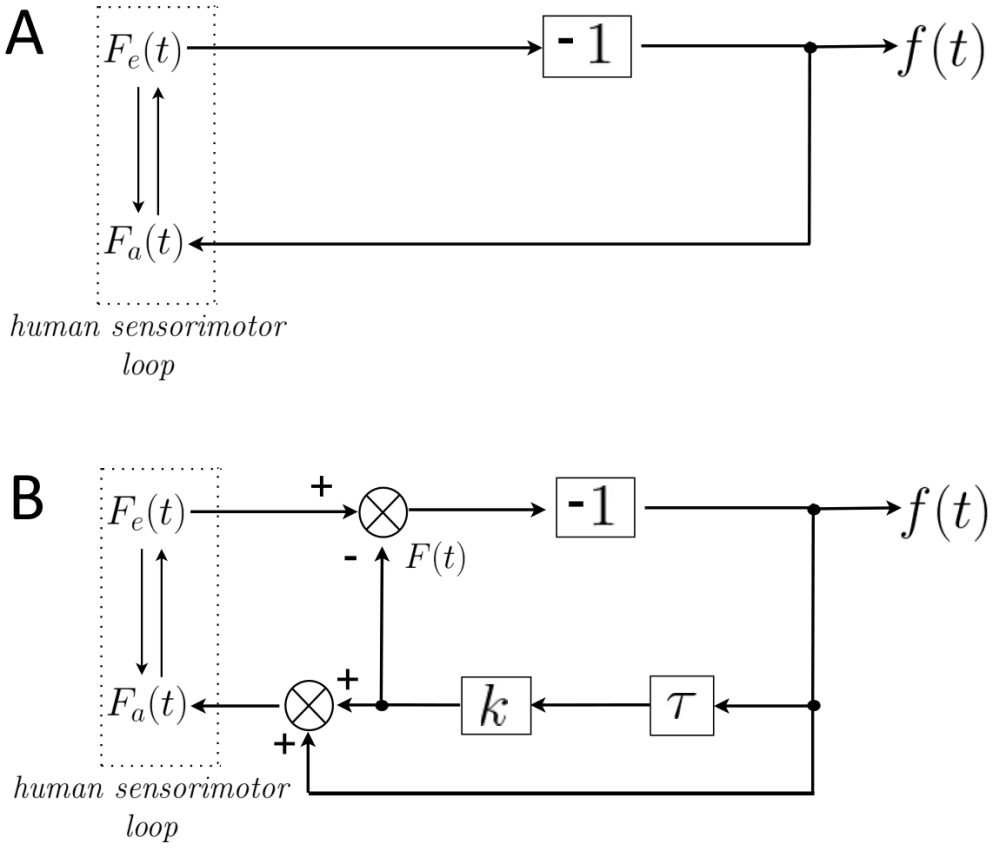


Figure 13. Human-in-the-loop control system describing (A) traditional surgery and (B) with surgery with *in-situ* force magnification.

transmitted directly to the tissue, resulting in an equal but opposite force *f*, the force of the tissue on the tool, that is felt directly as the afferent force F_a. As discussed in Chapter 1, this interaction force is commonly noisy or sub-threshold. This is especially true in delicate environments, where movements are also very small (150-200 μm) and performed slowly [21]. In this regime, kinesthetic inputs do not much change the sensory experience, and physiological tremor becomes significant. For these reasons, the human sensorimotor loop is deprived of useful afferent information, leading to inaccurate efferent forces. Because the efferent force is directly transmitted to the target, these forces may cause damage to the tissue if not well controlled.

In *in-situ* force magnification, represented in **Figure 13B**, we use a simple proportional magnification scheme,

$$F = k * f \qquad \text{(Eq. 12)}$$

where k is the magnifier gain; F is the feedback force; and the f is the distal force. The device applies the feedback force F which not only can magnify the sub-threshold distal force f to supra-threshold levels, but also works in the opposite direction of F_e, lowering the net force applied to the tissue. Since the fingers must match the sum of the distal force f and the feedback force F both cutaneous and kinesthetic senses are stimulated by this type of force feedback. According to **Figure 13B**, the perceived force under magnification is described by

$$F_a(t) = f(t) + F(t)$$

$$= f(t) + k * f(t)$$

$$F_a(t) = (k+1) * f(t) \qquad \text{(Eq. 13)}$$

In considering **Figure 13B**, it bears repeating that the HHFM does not explicitly affect the motor control of the user by means of actuation. Rather, it intervenes with the user's *perception* of the distal interaction, which in turn is spontaneously leveraged by the sensorimotor system to improve motor control. These claims of afferent and efferent benefits with force magnification are examined in our psychophysics work, described elsewhere in this dissertation, so that we may understand how magnified forces are perceived, and how these afferent changes can modulate motor behavior while using the HHFM.

3.2 THE MODEL-1 HAND-HELD FORCE MAGNIFIER

The HHFM Model-1 was constructed as part of my Master's thesis work [133]. We review the design of this proof-of-concept prototype, and our initial psychophysical evaluation in the following section.

3.2.1 HHFM Model-1 Design and Development

The HHFM Model-1 is shown in **Figure 14** [130]. It is a uni-directional proof of concept intended to measure and magnify "push" forces. The distal tip of the device consists of a force sensor (Honeywell FS01; Morris Plains, NJ) rigidly connected to the body of a 1 mL syringe, measuring 6.35 mm (0.25 inches) in diameter. The FS01 is a single-axis resistive force sensor, and contains integrated temperature compensation circuitry. The FS01 is capable of measuring forces up to 6.8 N, with 0.068 N accuracy (1% of full-scale) and 0.034 N hysteresis (0.5% of full-scale).

Coaxial to and extending the syringe is a 6.35 mm diameter brass tube, within which eight rare earth permanent magnets (3/16” Radioshack 64-1895; Fort Worth, TX) are stacked. The handle assembly is fit inside a coil of current-carrying wire (250 feet of 30-gauge wire, 25 Ω, approx. 2360 turns), creating a solenoid actuator capable of pulling and pushing. The entire assembly is then mounted on a gimbal which allows movement of the tool in the horizontal and vertical planes. The entire assembly is grounded to the body using an orthopaedic wrist support, which enabled users to easily don and remove the device. The HHFM Model-1 has a total mass of 164 grams.

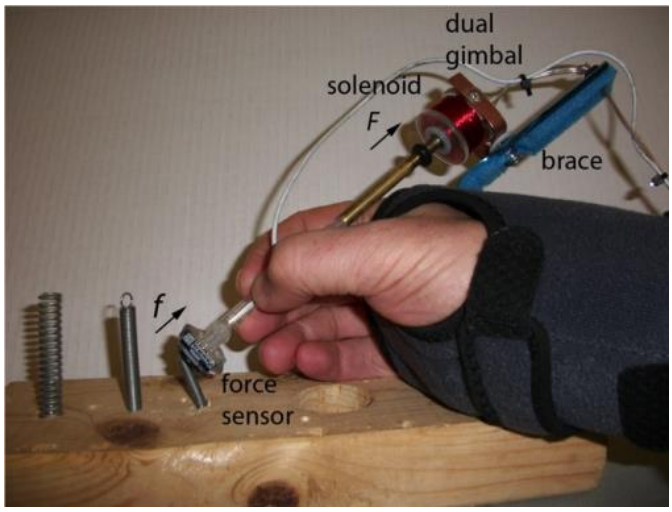


Figure 14. The HHFM Model-1. [130]

Control of the device is accomplished using analog circuitry. Electronic schematics for the Model-1 are included in **Appendix A**. In essence, changes in voltage in the distal sensor completely determine the current in the solenoid. A potentiometer knob is provided to modulate the gain, *k*, of the system. The system can scale force by factors up to $k = 5.8$ before stability effects are seen. While the device was somewhat bulky, the perceptual effects are very strong. As shown in **Figure 14**, springs that are “soft” in reality are perceived to be much stiffer when the device is turned on. To investigate how humans interact with such a device, we conducted three psychophysical experiments.

3.2.2 HHFM Model-1 Psychophysical Experiments

In this first set of experiments with the HHFM Model-1, we wanted to characterize changes in the afferent pathway under *in-situ* force magnification [74], [130]. We therefore measured users' absolute force threshold; their JND while using the device; and their subjective estimation of force and mechanical stiffness in simulated springs. In each experiment, participants were tested under three conditions:

1. Control, in which participants used the body of a 1 mL syringe to interact with the target;
2. HHFM-off, in which participants wore the HHFM, but the device was turned off;
3. HHFM-on, in which participants wore the HHFM and the device was turned on.

These experiments were performed using a device called the Magnetically Levitated Haptic Device (MLHD), developed by the Microdynamic Systems Laboratory at Carnegie Mellon University [134]. The MLHD is a haptic renderer that is capable of actuating feedback

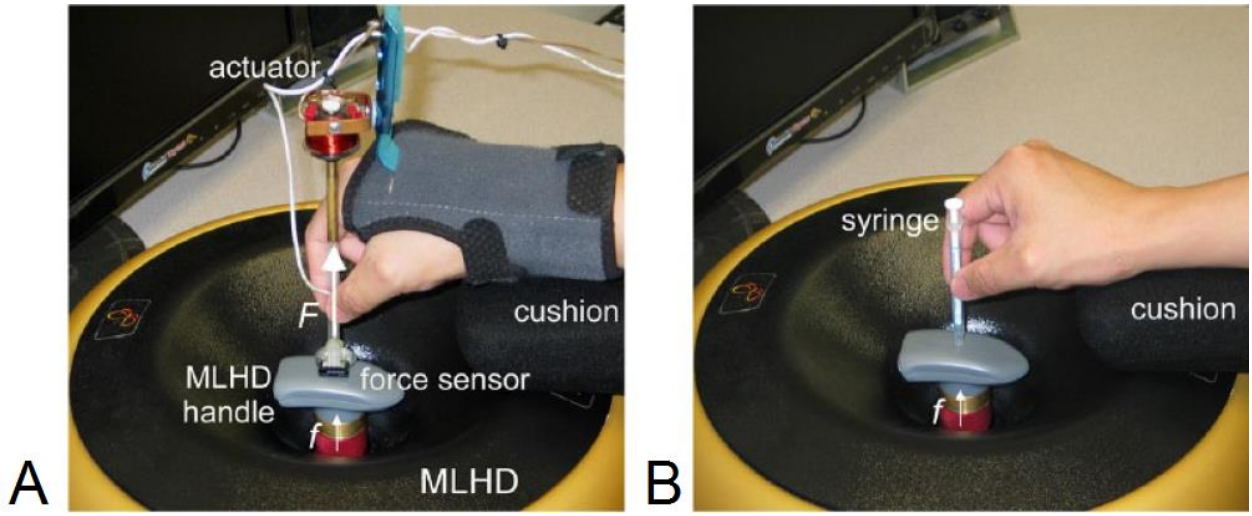


Figure 15. HHFM Model-1 psychophysics experimental setup in (A) HHFM-on/off and (B) control conditions. [74]

forces in all 6 DoF using three pairs of electromagnets acting on a "flotor," the floating manipulandum with which users interact (grey handle in **Figure 15**). The flotor can move within a spherical workspace 24 mm in diameter. The MLHD can generate up to 40 N on the flotor in the vertical direction, and up to 3.6 Nm torque. Because the MLHD uses Lorentz forces for actuation, feedback forces are generated without frictional losses or mechanical backlash, which are typical drawbacks of mechanical linkage type haptic renderers described in Chapter 2. The MLHD can update force at a maximum rate of 4000 Hz and can update position at 140 Hz. The MLHD uses three optical sensors for position measurement, which report location with approximately 2-micron accuracy. Virtual environments are programmed in C++ using a structured API on either Windows and Linux operating systems. The MLHD is connected to a controller computer by a 100 Mbps Ethernet cable, through which commands are sent to produce the desired stimuli and record participants' keypad responses.

During each experiment, participants wore noise-cancelling headphones and closed their eyes, as applicable, to eliminate potential distractions. Feedback forces in the MLHD were updated at 2000 Hz and flotor motion was restricted to the vertical axis. At the start of each trial, participants rested the tool on the MLHD flotor, as shown in **Figure 15** [74]. To ensure comfort throughout the experiment, a cushion was provided to allow participants to contact the flotor in a perpendicular orientation. The gain of the HHFM was set to $k = 2.4$, so the overall perceptual gain was assumed to be 3.4, as described by Eqn. 13.

3.2.2.1 Experiment 1: Absolute Force Threshold

To measure users' absolute force detection threshold, we instructed study participants to rest the tool on the MLHD flotor as described previously. Twelve volunteers (eight male and four female), aged 22 to 35 years old, participated in this experiment with informed consent. To

eliminate the possible confounding variable of handedness, all participants were right handed by self-report. Participants were tested individually, and the experiment lasted approximately one half hour.

The method of adjustment was utilized to measure the absolute threshold. Each trial began with a clearly sub-threshold force (0 N) or supra-threshold force (0.3 N) applied in the upward direction (counter gravitational) onto the tool tip. Participants were then instructed to use a keypad to increase or decrease the applied force until they could just perceive, or could no longer perceive, the applied force. Each keypress changed the applied force in the desired direction by 0.02 N, with 70% probability, to prevent participants performing the task by counting keypresses. This procedure was repeated twice in both loading and unloading directions, and the average force at which the force at the tool tip "appeared" or "disappeared" was recorded as the absolute force threshold.

Results from this experiment are shown in **Figure 16**, where the error bars represent ±1 standard error of the mean (SEM). One-tailed, paired t-tests with Bonferroni correction were performed to evaluate the data. The Bonferroni correction is necessary when making multiple pairwise comparisons, such that for m comparisons, the significance level α is corrected to α/m. The critical significance level was set at $\alpha = 0.05$ and 3 paired comparisons were made.

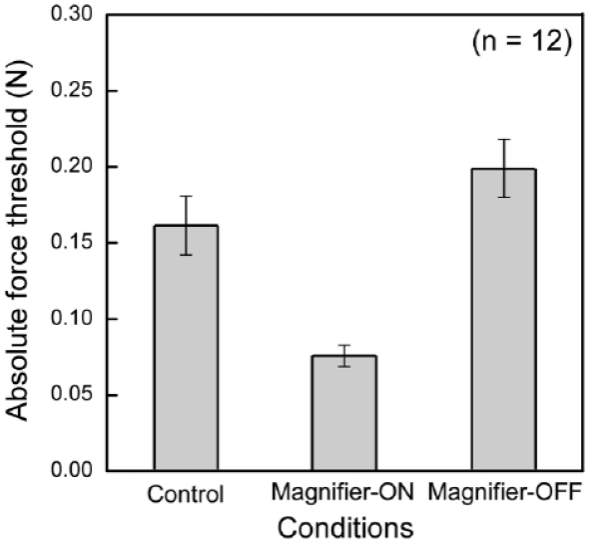


Figure 16. Effect of force magnification on the absolute force threshold. Error bars represent ±1 SEM. [74]

We found that the mean absolute threshold in the HHFM-on condition was 0.07 N, significantly lower than the 0.20 N threshold found in the HHFM-off condition ($t(11) = 5.66$, Bonferroni-adjusted $p < 0.001$). In the control condition, the absolute threshold was found to be 0.16 N, which was also found to be significantly different from the HHFM-on condition ($t(11) = 4.42$, Bonferroni-adjusted $p = 0.003$). Importantly, the difference between the HHFM-off and the control conditions was not found to be statistically significant ($t(11) = 2.49$, Bonferroni-adjusted $p = 0.09$). Therefore, we can conclude that wearing the device does not alone affect the absolute threshold. A one-way, repeated-measures ANOVA comparing all three conditions also showed that the differences in thresholds were significant ($F(2,22) = 22.92$, $p < 0.001$, partial $\eta^2 = 0.68$).

We may consider the absolute threshold an equivalent perceptual state between the various experimental conditions. We can therefore obtain a measure of the perceptual gain (afferent or behavioral gain) by taking the ratio of the threshold in the HHFM-off condition to the threshold in the HHFM-on condition. The average afferent gain was found to be 2.9, which was not significantly different from 3.4, the hypothesized perceptual magnification of the device as determined from Eqn. 13 ($t(11) = 1.19$, $p = 0.26$). From this result, we have verified that the afferent pathway in our control framework (see **Figure 13**) behaves as proposed.

3.2.2.2 Experiment 2: Just Noticeable Difference

In a second experiment, we measured how one's JND is affected by force magnification. Sixteen participants (ten male and six female), aged between 18 and 38 years old, were tested individually and with informed consent. Again, all were right-handed by self-report. Typically, a

JND session was completed in less than 15 minutes, and participants were afforded a 5-minute break between experimental sessions. The entire experiment lasted approximately one hour.

Each trial began with users contacting the MLHD flotor with the HHFM as previously described. JNDs were measured against a reference force of 0.3 N in the three experimental conditions: HHFM-on, HHFM-off, and control. A fourth experimental condition, HHFM-off with a reference force of 1.02 N, was also tested. This last testing condition is equivalent to the output of the HHFM-on condition for the 0.30 N reference force and an assumed perceptual magnification factor of 3.4.

JND was measured using an unforced adaptive procedure that targeted 75% correct detections [135]. At the beginning of each trial, a pair of force stimuli (reference and test) was presented to the participant, one at a time. Each force was associated with a color label (yellow or blue) shown on an LCD screen. The forces (reference or test) and color (yellow or blue) were assigned randomly to the first and second stimulus. Participants were asked to judge which force felt stronger and to press a colored button corresponding to that stronger stimulus. A third button, labeled "Unsure," was also available if he or she could not tell the difference between the stimuli. Participants could switch back and forth between the two presented stimuli as often as they wanted by pressing a button labeled "Switch," which would gradually change the stimulus from one to the other.

In this adaptive procedure, the difference between the reference force and the test force was updated depending on whether the participant responded correctly in the previous trial. If the participant answered correctly, the difference between the reference and test forces would be decreased by one step. Conversely, the difference was increased by three steps if the participant answered incorrectly, or increased by one step if the participant answered "Unsure." A "reversal"

is defined as a change in the direction of the adjustment, for example, when an erroneous or unsure response precedes or follows a correct response. The experiment was terminated after eight reversals, and the difference threshold was defined as the average of the test values at the fourth and eighth reversal. In each experimental condition, the threshold was estimated twice in both ascending and descending directions, and the mean of the two measurements was reported as the JND. The initial difference between the reference and test forces was 24%. The initial adjustment step was 8% and halved at the second and fourth reversals.

Results from this experiment are shown in **Figure 17**, where the error bars represent ±1 SEM. JNDs were compared using a one-way repeated-measures ANOVA and t-tests with Bonferroni correction. The mean JND for a reference force of 0.30 N was much lower in the HHFM-on condition (11.04%) when compared to HHFM-off (15.39%) and control (15.15%) conditions. The ANOVA showed a significant difference in mean JNDs across the experimental conditions ($F(3,33) = 11.59$, $p < 0.001$, partial $\eta^2 = 0.44$). Pairwise comparisons also confirmed that such differences were statistically significant (HHFM-on versus HHFM-off: $t(15) = 3.10$, Bonferroni-adjusted $p = 0.04$, $d = 0.77$; and HHFM-on versus Control: $t(15) = 3.76$, Bonferroni-adjusted $p = 0.01$, $d = 0.94$). No significant difference was detected between the Control and HHFM-off conditions ($t(15) = 0.16$, Bonferroni-adjusted $p > 0.99$, $d = 0.04$).

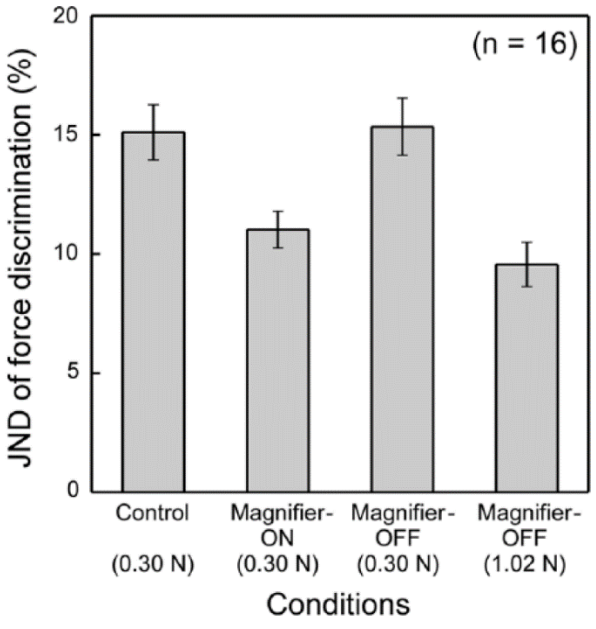


Figure 17. Effect of force magnification on JND. Error bars represent ±1 SEM. [74]

Since all forces in the HHFM-on condition were augmented by the 3.4 magnification factor, the reference force of 0.30 N was magnified to 1.02 N. We tested the HHFM-off condition with this 1.02 N reference force to compare the effect of magnification with baseline sensory abilities. We found that the JND in this last HHFM-off condition was 9.58%, which was significantly different from the JND found in the HHFM-off condition at a 0.30 N reference force ($t(15) = 4.91$, Bonferroni-adjusted $p = 0.001$, $d = 1.23$). Importantly, we did not find a difference between the JND in the HHFM-on condition with 0.30 N reference, and the JND in the HHFM-off condition with 1.02 N reference ($t(15) = 1.83$, Bonferroni-adjusted $p = 0.53$, $d = 0.46$). These data show that magnified forces are well perceived by individuals, and that the *in-situ* magnification enhances a user's ability to detect small differences in force.

3.2.2.3 Experiment 3: Subjective Estimation of Force and Stiffness

The third psychophysical experiment was split into two separate, but closely related studies. The first was a pilot study that examined how one subjectively experiences tool tip force under magnification [130]. Secondly, we investigated how force magnification affects one's estimation of mechanical stiffness [74]. Estimating force or stiffness in a medical context is valuable in identifying pathologic tissue, like malignant tumors or calcified tissue. We hypothesized that both abilities would be aided by force magnification, but the effect on stiffness estimation may be reduced due to the additional factor of deformation estimation [78]. To experimentally characterize and quantify a participant's internal experience of a stimulus, we used the method of magnitude estimation. We also sought to calculate a measure of the perceptual gain, as was done in our absolute force threshold experiment (see Experiment 1).

The pilot study examined how participants internally experienced a distal force under magnification. Six participants (four males and two females), aged between 22 and 35,

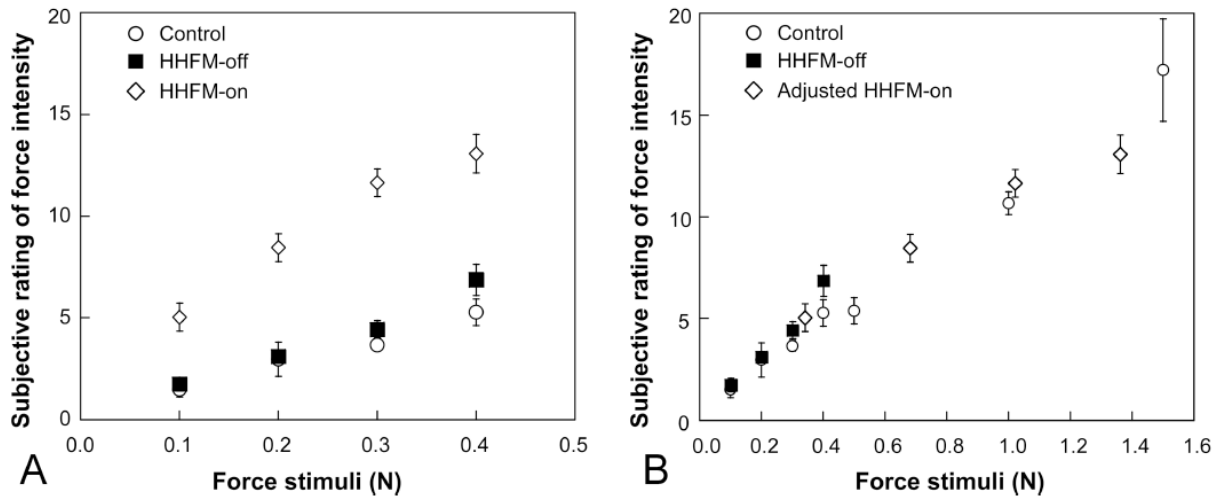


Figure 18. (A) Estimation of force magnitude under magnification. (B) Estimates replotted by multiplying HFFM-on stimuli by the perceptual gain of 3.4. Error bars represent ±1 SEM. [130]

participated in this experiment with informed consent. At the beginning of each trial, each participant rested the hand-held tool vertically on the MLHD as previously described. The participant was then presented with random force stimuli with magnitude 0.1, 0.2, 0.3, or 0.4 N. In the control condition, forces of 0.5, 1.0, and 1.5 N were also presented to produce similar stimuli to those in the HHFM-on condition, where distal forces were magnified by a factor of 3.4. Each force was presented three times in each condition. The experiment lasted approximately an hour.

Participants were instructed to assign a numerical score to each stimulus based on its perceived intensity, with the only restriction being that subjectively stronger stimuli should be assigned higher numbers. The data were normalized by dividing each response by the participant's mean for a given condition, then multiplying by the overall mean for all participants [60]. The three normalized estimates from each condition were then averaged, and this mean reported as the estimated magnitude.

Average magnitudes across subjects from this pilot study are shown in **Figure 18**, where the error bars represent ±1 SEM. As expected, for the same stimuli, distal forces were judged to be subjectively more intense with the HHFM turned on (see **Figure 18A**). Similarly to the absolute force threshold experiment, we can obtain a measurement of the perceptual gain by taking the ratio of the HHFM-on ratings to those taken in the HHFM-off condition. This ratio comes out to be 2.14 ± 0.18, somewhat lower than the expected gain of 3.4. The range of observed perceptual gains was 1.85 to 2.40.

Multiplying the stimuli by the hypothesized perceptual gain of 3.4 in the HHFM-on condition enables us to replot the magnitude data over the range of stimuli as they were actually perceived. This reorganization of data is shown in **Figure 18B**. Given the close overlap between ratings obtained in the control condition and those from the HHFM-on condition, we can conclude that feedback forces generated by our device are perceived similarly to real forces of similar magnitude.

Three two-way (force × device) repeated-measure ANOVAs were performed to assess the effect of each experimental condition on perceived force magnitude. Comparing the HHFM-off and control ratings data, significant main effects were found for force ($F(3,15) = 16.61$, $p < 0.001$), but not for the device ($F(1,5) = 0.18$, $p = 0.69$), nor any significant interaction ($F(3,15) = 0.67$, $p = 0.59$). This may be interpreted to mean that the additional weight of the HHFM itself did not affect subjective magnitude estimation in this supra-threshold task.

An ANOVA comparing HHFM-on to the control showed a significant effect for higher magnitude in the HHFM-on condition ($F(1,5) = 20.76$, $p = 0.006$). In addition, there was a significant force × device interaction ($F(3,15) = 4.45$, $p = 0.02$) and a significant main effect for force ($F(3,15) = 18.40$, $p < 0.001$). For comparisons between the HHFM-on to the HHFM-

off data, there was a similar pattern: a significant main effect for the HHFM-on condition ($F(1,5) = 37.31$, $p = 0.002$), a significant force × device interaction ($F(3,15) = 4.54$, $p = 0.02$), and a significant main effect for force ($F(3,15) = 15.11$, $p < 0.001$).

In the second half of the study, we investigated how estimates of stiffness were affected by force magnification. Stiffness is, by definition, the ratio of an applied force to the deformation caused by such a force. Perception of stiffness therefore requires estimation of both quantities. The same twelve participants from Experiment 1 participated in this study.

As in the pilot study, the method of magnitude estimation was used to quantify perceived stiffness. Participants started with the HHFM resting on the MLHD flotor as in the previous experiments. Virtual springs with stiffness 20, 40, 60, or 80 N/m were rendered in the MLHD, and participants were allowed to freely interact with them. Again, the only restriction placed on the numerical estimates was that subjectively stiffer springs should be assigned higher numbers. In addition, springs of stiffness 100 and 200 N/m were rendered in the control condition to account for range effects of HHFM magnification. All springs could be compressed up to 6 mm, resulting in end forces from 0.12 to 0.48 N. Each condition was tested three times each, and the experiment lasted approximately 1 hour.

Ratings data were normalized as described previously. Each judgement was divided by that participant's mean for that condition, the result of which was then multiplied by the grand mean rating of all participants for that condition. **Figure 19A** shows the rating data. It should be noted that the control and HHFM-off conditions largely overlapped. The HHFM-on data were significantly higher than both control and HHFM-off conditions.

As was done in the pilot study, perceptual gains were estimated for each virtual spring. The perceptual gain was highest for the weakest spring, and lowest for the strongest spring. The

average gain for the 20 N/m spring was 2.7, which was not statistically different from the hypothesized afferent gain of 3.4, as determined from a one-sample t-test ($t(11) = 1.08$, $p = 0.30$, $d = 0.31$). The perceptual gain for the springs of stiffness 40, 60, and 80 N/m were 2.3, 2.0, and 1.7, respectively. Each of the gains for stiffer springs was significantly different from the hypothesized afferent gain of 3.4 ($t(11) = 3.79$, $p = 0.003$, $d = 1.09$ for the spring of 40 N/m; $t(11) = 4.53$, $p = 0.001$, $d = 1.31$, for the spring of 60 N/m; and $t(11) = 15.19$, $p < 0.001$, $d = 4.38$, for the spring of 80 N/m). We see this compressive effect in **Figure 19B**, where ratings data were rescaled by the hypothesized perceptual gain. With increasing stiffness, the HHFM-on data diverged from the control data, which were collected with large stiffness springs at 100 and 200 N/m.

Two-way (device × stiffness) repeated-measure ANOVAs were performed on raw data (not normalized as previously described) to assess the effects of magnification on the rating data. Between the HHFM-off and HHFM-on conditions, significant main effects were found for the device ($F(1,11) = 12.15$, $p = 0.005$, partial $\eta^2 = 0.53$) and for stiffness ($F(3,33) = 14.56$, $p <$

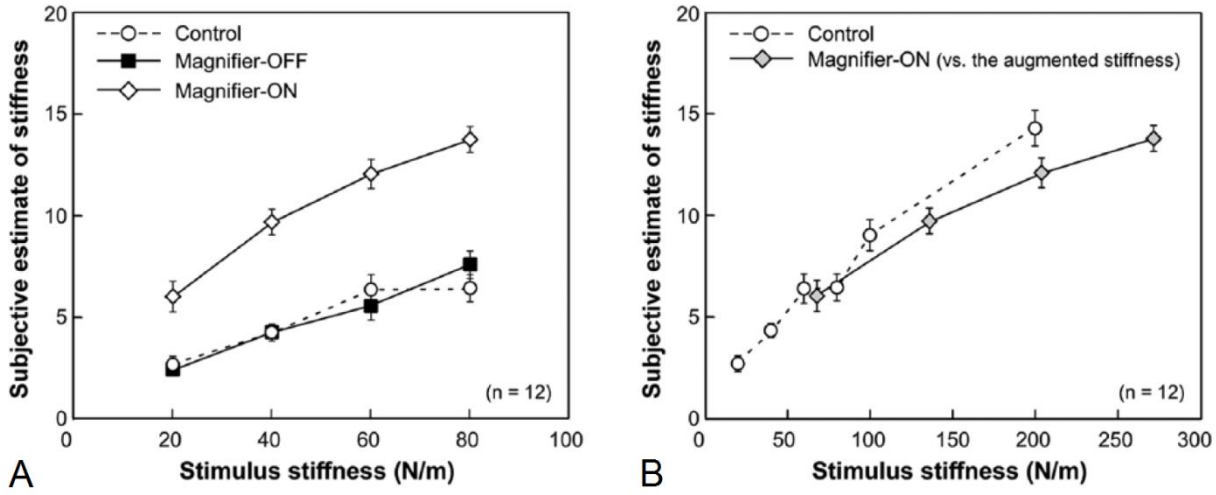


Figure 19. (A) Subjective estimation of stiffness and (B) Estimates rescaled by hypothesized afferent gain. Error bars represent ±1 SEM. [74]

0.001, partial $\eta^2 = 0.57$). No significant interaction effect between device and stiffness was found ($F(3,33) = 2.47$, $p = 0.08$, partial $\eta^2 = 0.18$). Bonferroni pairwise comparisons also found that the effects of force magnification were significant at all stiffness levels ($t(11) > 3.11$, Bonferroni-adjusted $p < 0.04$, $d > 0.89$).

We also examined the data with a 3 (device) × 4 (stiffness) repeated-measures ANOVA. Comparison of the HHFM-off and HHFM-on data found a similar pattern of significant effects to the two-way ANOVA. When comparing HHFM-off data to control data, the main effect of stiffness was found to be significant ($F(3,33) = 18.39$, $p < 0.001$, partial $\eta^2 = 0.63$), but not for the device ($F(1,11) = 0.10$, $p = 0.76$, partial $\eta^2 = 0.01$), nor for the interaction of device and stiffness ($F(3,33) = 2.64$, $p = 0.07$, partial $\eta^2 = 0.19$). This suggests that wearing the HHFM alone had little effect on the perception of stiffness.

Comparing the results from our pilot study on force estimation to these data on stiffness estimation, we find similar patterns in the behavioral gains. When estimating force, the perceptual gain ranged from 2.6 to 1.9. Similarly, for stiffness estimation, the perceptual gain ranged from 2.7 to 1.7. In both tasks, the gain was strongest for the weakest stimulus and dropped with strength in the stimulus. Paired t-tests found that there was no significant difference between the two sets of gains ($t(11) < 1.13$, $p > 0.28$, $d < 0.33$). It should be noted, however, that the calculated perceptual gain for stiffness estimation was generally lower than that for force estimation.

3.3 THE MODEL-2 HAND-HELD FORCE MAGNIFIER

While the Model-1 worked well as an initial proof-of-concept, its design exhibited a number of weaknesses. The handle of the device was prone to binding within the solenoid, which interfered with force magnification. Users with smaller hands in particular had a harder time using the Model-1 due to this binding, difficulties that were also compounded by the restrictive nature of the wrist brace. Further, while using the HHFM for extended periods of time, as in our psychophysical experiments, excess heat was generated by the solenoid, to the point that the device needed to be turned off to cool.

A new prototype was designed, the HHFM Model-2, as a proof-of-concept for bidirectional force magnification. This work was also part of my Master's thesis [133]. We review the design of this HHFM prototype in the following section.

3.3.1 HHFM Model-2 Design and Development

The HHFM Model-2 is shown in **Figure 20** [136]. This is the first bi-directional HHFM prototype, capable of magnifying push and pull forces in the axial direction. As with the Model-1, this design also uses a Honeywell FS01 force sensor (6.8 N force range), which in this case is preloaded using a steel leaf spring. The spring is 0.012 inches (0.30 mm) thick stainless steel shim stock, cut into a square (32 mm × 32 mm) and mounted to aluminum spacers surrounding the sensor. A stainless steel threaded rod (#6-32) is connected to the leaf spring and contacts the force sensor to apply a steady preload. The preload is adjustable by threading the rod through a pair of machine nuts that hold the tool tip to the spring. Relieving the preload on the sensor

enables measurement of the pull force, whereas adding to the preload occurs with a push force. Thus measurement of forces in both directions is achieved using a sensor designed only for push.

The force sensor is mounted to a 4 mm thick aluminum plate, to which an acrylic handle is press-fit. The handle of the tool is 10.35 mm in diameter, and is kept concentric to a proximal actuator by way of a linear bearing and a machined aluminum post. Sensor wiring runs through the acrylic handle, exiting out the connection between the handle and the aluminum post. Aluminum was chosen for the post between the actuator and the acrylic handle due to its low friction characteristics, as well as its relatively low density and high durability. The actuator is a linear voice coil motor (Moticont LVCM-019-022-02; Van Nuys, CA), which can generate up to 2.5 N over its 12.7 mm stroke length. The actuator and linear bearing are kept concentric by another acrylic tube, 20.2 mm in diameter.

Because the orthopaedic wrist support in the Model-1 was restrictive and bulky, work was done to design a brace with a smaller physical footprint and mass. The resulting brace was

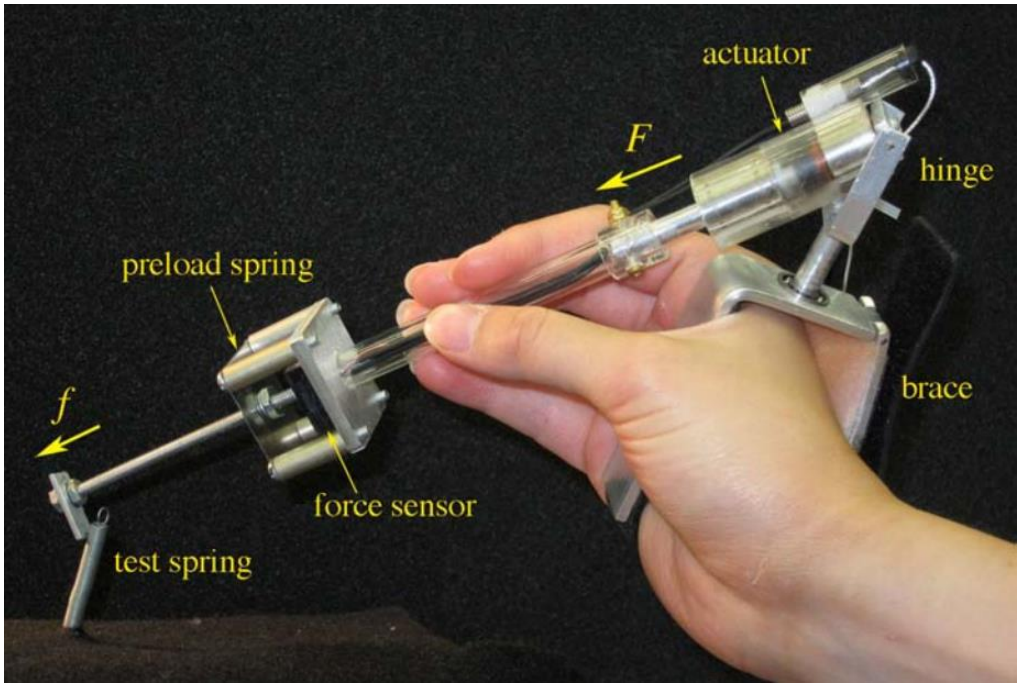


Figure 20. The HHFM Model-2. [136]

constructed from 3.175 mm (0.125 inch) thick aluminum stock. The brace has a simple hinge, and is secured to the hand on the palmar side with Velcro straps that reach to the back of the hand. Foam padding between the back of the hand and the brace improves fit and comfort. A rotary bearing was press-fit into the top of the brace, to allow for movement in the horizontal plane. A small aluminum post was press-fit into the rotary bearing and extends vertically into a gimbal that enables movement in the vertical plane. Compared to the Model-1, this new assembly is much easier to don and remove, and makes the HHFM easier to hold for those with smaller hands. With this brace, the HHFM is more rigidly connected to the hand, forming a more reliable base from which feedback forces are actuated.

3.3.2 HHFM Model-2 Control

Control over the HHFM Model-2 was initially accomplished using a National Instruments (Austin, TX) NI-6009 data acquisition (DAQ) unit running a LabVIEW virtual instrument (VI). LabVIEW is a graphical programming language which uses a variety of pre-built, multi-use components to control specific hardware. The HHFM Model-2 VI software is included in **Appendix B.2**. The NI-6009 can measure 8 analog inputs over a ±10V range with 14-bit resolution. The maximum sampling rate is 48 kS/s. The NI-6009 also has two 12-bit analog output channels, capable of generating up to 5 V. The NI-6009 unit is powered through a USB connection to a computer. One major advantage of a LabVIEW implementation is that a graphical user interface (GUI) is also implemented using pre-built components, enabling on-screen voltage visualization, custom dialogs and user inputs, and on-screen radio buttons, among other functionality.

The update rate of the LabVIEW implementation was set at 1 kHz. In using this HHFM prototype, however, we noticed a considerable difference in performance when it was connected to computers running different operating systems. **Figure 21** shows this difference in jitter on the actuator output voltage between Windows 7 and Macintosh OSX operating systems. Because LabVIEW essentially operates as a program within an operating system (OS), it is subject to OS-level interrupts, which introduce variabilities in the actual operating frequency. Delays due to these interrupts manifest as jitter in the device. To address this difficulty, we chose to implement device control using a dedicated Analog Devices (Norwood, MA) ADuC7026 microprocessor.

The ADuC7026 is an ARM7TDMI CPU, running at 41.78 MHz. The CPU has twelve 12-bit ADC channels, capable of 1 MS/s over a 0 – 2.5 V range. It also has four 12-bit DAC channels, able to generate voltages from 0 – 2.5 V. Given these specifications, the ADuC7026 is able to resolve changes at 610 μV. In addition, the ADuC7026 has a bank of 40 general purpose input/output (I/O) pins, as well as built-in Inter-Integrated Circuit (I^2C) and Serial Peripheral Interface (SPI) communication support. The I/O pins may be individually programmed as binary

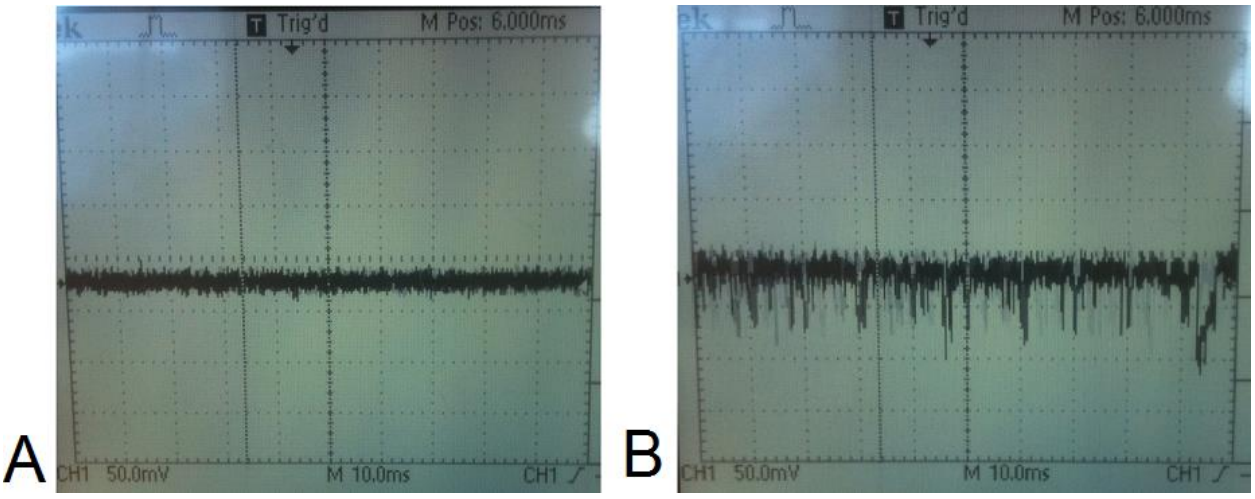


Figure 21. Observed jitter in HHFM output with LabVIEW on (A) OSX and (B) Windows 7 operating systems.

inputs or outputs. Control over all of these functions is achieved through an API of memory mapped registers (MMRs). Control software is implemented in C, and our source code is included in **Appendix B.3**. The source code is compiled using an ARM compiler through the Kiel µVision IDE on Windows 7. While the microprocessor is flash programmed over a USB connection, it does not require a continuous connection to an external computer to operate normally.

Operation of the HHFM Model-2 follows a simple sensing-actuation loop, using a voltage-centered magnification scheme. At the beginning of each loop, the current force sensor voltage, V_{sensor}, is sampled and compared to the preload voltage, $V_{preload}$. The output voltage, V_{out}, which drives the HHFM actuator, is then given by

$$V_{out} = k * (V_{sensor} - V_{preload}) + 1.25 \qquad \text{(Eq. 14)}$$

where k is the HHFM magnification factor.

Because the microprocessor DAC can only generate up to 2.5 V, we use 1.25 V as the borderline between push and pull forces. As mentioned previously, feedback forces generated by the HHFM are directed in the same direction as the sensed force. Due to the configuration of the actuator, in response to a push force at the tool tip, for example, the HHFM must pull on the handle to augment the perception of the distal force. Therefore, according to Eqn. 14, the region between 0 – 1.25 V represents a relief of the force sensor preload, and therefore should generate a "push" in the voice coil. The voltage region between 1.25 – 2.5 V represents a "pull" in the voice coil. The electronics schematics for this connection to the Moticont LVCM are shown in **Appendix B.1**.

Using the microprocessor's built-in interrupt interface, the HHFM Model-2 operates at 10 kHz without significant jitter. The sampling and actuation loop is therefore guaranteed to run in

less than 100 microseconds using this dedicated architecture. This is significantly faster than necessary, as frequency discrimination diminishes greatly past 1,000 Hz stimulation [137]. However, this high operation frequency means that as our firmware grew in complexity, as will be seen in subsequent chapters, we had sufficient computational headroom to accommodate their implementation.

The force sensor input is filtered at 1 kHz to smooth the signal. Obtaining an adequate estimation of the preload force is accomplished by randomly sampling the sensor voltage over 1 second at 200 time points. The interval between samples was drawn from a uniform distribution between 1 and 200. This procedure is used to avoid aliasing effects. The average of this random sample of voltages is retained in memory as the preload voltage whenever the microprocessor is restarted. This "tare-ing" or "zero-ing" procedure may also be initiated by a button on the hardware enclosure (see **Figure 22**). A button is also included to "mute" the output of the HHFM by forcing the controller to generate 1.25 V. Finally, a button to initiate a hardware reset of the CPU was added for ease of access. **Figure 22** shows the front panel of the electronics enclosure for the HHFM Model-2, containing the Analog Devices microprocessor and associated circuitry.

Figure 22. HHFM Model-2 hardware enclosure, front panel.

3.4 DISCUSSION

From building these two proof-of-concept prototypes, the Model-1 and Model-2, we learned much about how the afferent pathway works while using a hand-held tool. Psychophysical experiments showed, for example, that *in-situ* force feedback is able to reduce the absolute force detection threshold. Further, force feedback is well perceived by users over a wide range of stimuli, as evidenced by subjective estimations of tool tip force. The best performance was found for small forces – as one increases further up into supra-threshold levels, the benefits of force magnification somewhat diminish. The observed perceptual gain was 2.9 in Experiment 1 for detecting forces at threshold levels, and 2.7 for the smallest spring stiffness in Experiment 3, not significantly different from our predicted gain of 3.4. Force magnification has also been shown to be an effective aid in distinguishing between similar forces, showing improvements to JND. Continuing to refine our design, we will examine how these improvements on the afferent pathway influence users' motor behavior.

Another advantage that we began to see as we developed these prototypes was an inherent modularity in the sensing, actuation, and control subsystems that, together, comprise the HHFM. Therefore, given careful design of the interfaces for each of these subsystems, further prototypes could be developed that recycle hardware from previous design iterations. By choosing the Analog Devices microprocessor as the master controller, our subsequent interfaces were standardized such that they would be interchangeable, speeding up our development process.

4.0 THE MODEL-3 HAND-HELD FORCE MAGNIFIER

With our initial HHFM prototypes and psychophysical experiments, we have demonstrated that it is possible to provide *in-situ* force feedback with a hand-held tool, and that such feedback is well perceived by users. However, these prototypes are clearly lacking when we envision a device used in the surgical theatre. Obviously a large sensor like that used in the HHFM Model-2 is unacceptable. In this chapter, we describe our efforts to refine the mechanical design and control software in the HHFM Model-3 prototype. We also describe the design and results of several additional psychophysical experiments to examine how one's motor control behavior may be influenced by afferent stimulation.

4.1 HHFM MODEL-3 DESIGN AND DEVELOPMENT

A critical consideration when designing a robotic clinical tool is the arrangement of the sensing and actuation components so as to preserve device ergonomics while still achieving the desired performance. Obviously, a large distal force sensor like the Honeywell FS01 is not appropriate for a hand-held tool in surgery, where tools are normally slim and precise, especially at the tip end. Moving the sensor proximally into the handle is one solution. In order to do so, sensors with a smaller physical footprint must be identified. Smaller force sensors come with their own

difficulties, however, since they usually lack the accompanying circuitry that provide high linearity and low noise in integrated units like the FS01.

With these tradeoffs in mind, we chose a pressure sensor produced by Motorola (MPX2011DT1; Schaumburg, IL) for commercial medical equipment, because of its small size (6.60 mm × 6.07 mm × 3.81 mm), high sensitivity (maximum pressure limit 75 kPa), and low cost (less than $1 per sensor). Being a pressure sensor, rather than a traditional force sensor, this sensor effectively has a preload of 1 atmosphere, from which it reports changes in either direction, and thus can inherently measure both push and pull forces. Assuming the sensor's 75 kPa pressure limit is distributed evenly over its 5.1 mm^2 circular sensing surface, the sensor can safely measure up to 383.3 mN. The sensing element is a piezoresistive strain gage embedded within a fluid filled chamber and covered by a flexible membrane. The sensor is ethylene oxide sterilizable, another advantage when designing a surgical tool that will potentially be reusable.

A smaller voice coil actuator (Moticont LVCM-013-013-02) was also chosen to reduce the physical bulkiness of the device. This voice coil can generate up to 0.86 N over its 6.35 mm stroke length, and has a mass of only 13 g. While this is a significant reduction in the potential force output compared to the HHFM Model-2, which could generate up to 2.5 N with its actuator, our psychophysical experiments showed that magnification factors around $k = 3$ are sufficient to induce benefits to the afferent pathway.

Our hardware redesign proceeded in a sequential manner, taking advantage of our modular strategy. We began by experimenting with different arrangements for the sensing subsystem. During development of the distal segment of the tool, we used the HHFM Model-2 actuation subsystem and electronics to drive force feedback. Upon finding a suitable sensing

arrangement, we then incorporated the new sensor together with the smaller replacement actuator.

The Model-3 was manufactured using stereolithography (SLA), a high resolution 3D printing technology. This technology builds three dimensional structures, layer by layer, by focusing a laser beam into a vat of liquid polymer, which solidifies on contact with laser light. In our instance, SLA was done on a 3D Systems (Rock Hill, SC) Viper Si^2 system. The printing material (Somos WaterShed XC 11122; Herleen, Netherlands) is similar to ABS (acrylonitrile butadiene styrene), a common plastic used in injection molding. Computer-aided-designs (CADs) were developed using Solidworks (Dassault Systèmes; Waltham, MA).

4.1.1 HHFM Model-3 Sensing Subsystem

A prototype sensing subsystem developed with the small Motorola sensor is shown in **Figure 23** [136]. At one end of a length of stiff catheter guidewire, a small epoxy bead (Loctite Quickset Epoxy; Westlake, OH) was rolled and allowed to dry. Once the epoxy was set, the guidewire was secured to the pressure sensor inside a silicone bead (GE Clear Silicone II; Huntersville, NC). Once dry, this entire

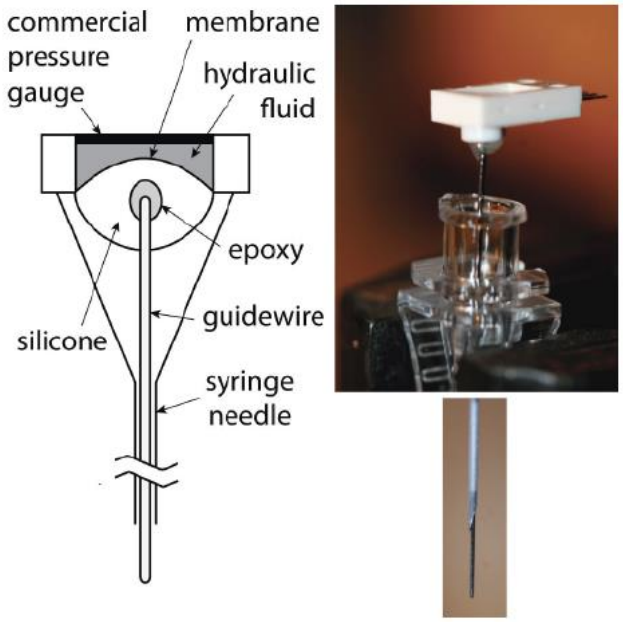


Figure 23. HHFM Model-3 guidewire sensing subsystem. [136].

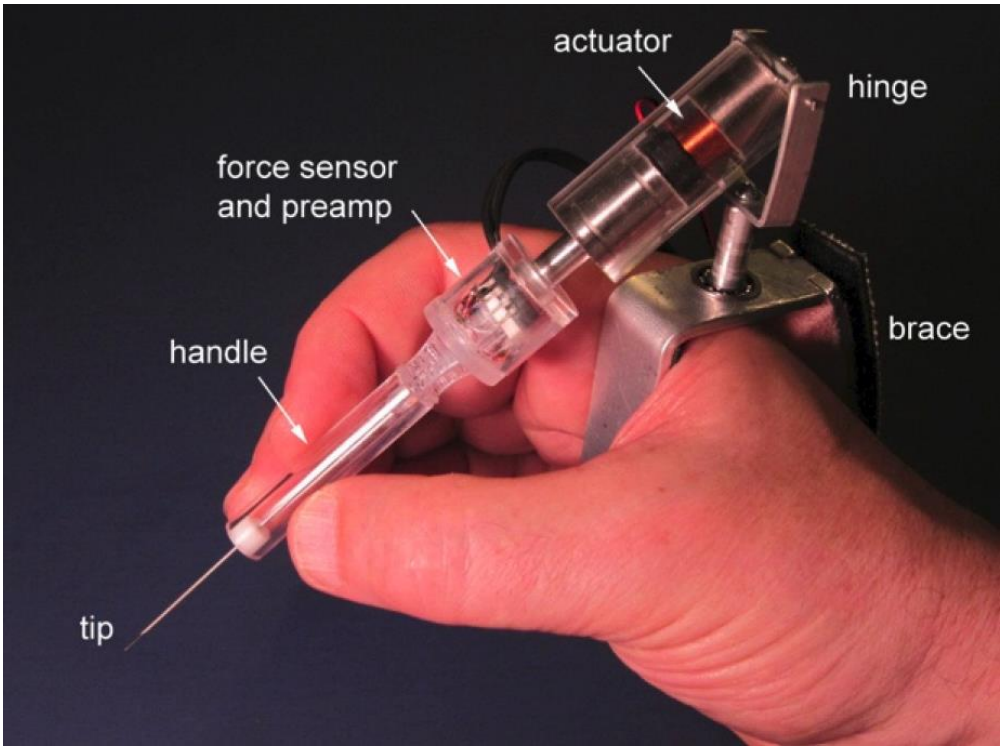


Figure 24. HHFM Model-3 guidewire prototype. [136]

assembly was then fit into a needle, with the guidewire passing through the needle bore. Voltage from the pressure sensor was amplified using an operational amplifier mounted within the tool to reduce noise associated with long wires between electronic components.

An intermediate Model-3 prototype using this sensing subsystem is shown in **Figure 24**. A handle was fashioned by fitting an acrylic tube concentric to the needle bore. The distal end of the needle bore was supported using a Delrin bushing (white, in **Figure 24**). Delrin is a low-friction polymer with low mass and high stiffness. The actuation subsystem was the same voice coil and linear bearing assembly used in the HHFM Model-2. While this design was quite sensitive to both pushing and pulling forces, it suffered from significant hysteresis, likely due to frictional losses between the guidewire and the needle shaft. Most importantly, we decided that

moving the sensor more distally into the handle was possible given the sensor's small size, shortening and simplifying the linkage, as will be described next.

The next step in the Model-3 redesign was to replace the guidewire assembly with a central shaft that contacts the pressure sensor directly. This mechanical linkage design is shown in **Figure 25**. A 0.50 mm thick mechanical spider supports the linkage at its distal end. The proximal end of the mechanical linkage is secured to the Motorola sensor using silicone. The mechanical spider at the distal end reduces the effect of off-axial forces, eliminates the bushing and related problems with friction, while adding stiffness to the sensing subsystem. A stiffer sensor assembly also protects the internal Motorola sensor from overloading. A tool tip connects to the spider and mechanical linkage through a #2-56 threaded shaft, thus permitting easy replacement of the tool tip. A chamber proximal to the linkage positions the Motorolla sensor concentric to the central shaft, such that a small space exists where silicone can adhere the linkage to the sensor membrane. Proximal to this sensor chamber is another chamber containing amplification and filtering electronics (the "preamp chamber"). Each part in the sensor assembly is 15.24 mm (0.60 inches) in diameter and comprises the HHFM handle.

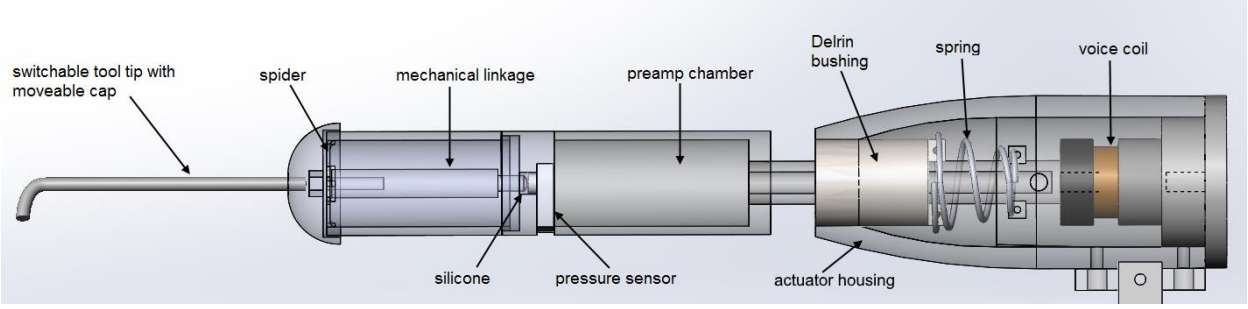


Figure 25. HHFM Model-3 mechanical linkage CAD.

4.1.2 HHFM Model-3 Actuation Subsystem

The actuation subsystem is driven by the small voice coil (Moticont LVCM-013-013-02). A hollow post connects to the coil through a threaded connection, and is press fit into the proximal end of the preamp chamber. Wires (white, in **Figure 26**) power the sensor and the operational amplifiers within the preamp chamber, and carry the amplified signal back to the microprocessor controller. The wire runs through this hollow post and exits the actuator housing through its underside, along with the fine flexible leads powering the voice coil (red and black, in **Figure 26**). The post is supported by a Delrin bushing, which itself is press fit to the distal end of an actuator housing. Replacing the metal linear ball bearing that had been used in the Model-2 reduced the mass of this subsystem dramatically.

A small spring connects the wiring post to the Delrin bushing, as shown in **Figure 25**, to ensure the voice coil remains in approximately the same position while in use. The spring was custom made by winding lengths of 0.34 mm diameter phosphor bronze guitar string around a 4.5 mm diameter brass tube. Coil position is important consideration since the force generated by a voice coil depends not only on the current powering it, but also on its position relative to the permanent magnet. The spring therefore serves a twofold purpose: to limit extension of the voice coil in either direction much past its midpoint, where the generated force is strongest, and to generate a sizeable (at least supra-threshold) restoring force when it does extend toward its extremes, encouraging the user to keep it in its neutral position. Since our device is rigidly gripped by the user, feedback force is transmitted to their fingers without any loss of transmission as a result of the spring. Indeed, the “stiffness” that can be exhibited by one’s hand is much greater than the stiffness of this spring, and the force generated by one’s hands is more than enough to extend and compress the voice coil. Nonetheless, the relatively gentle force of the

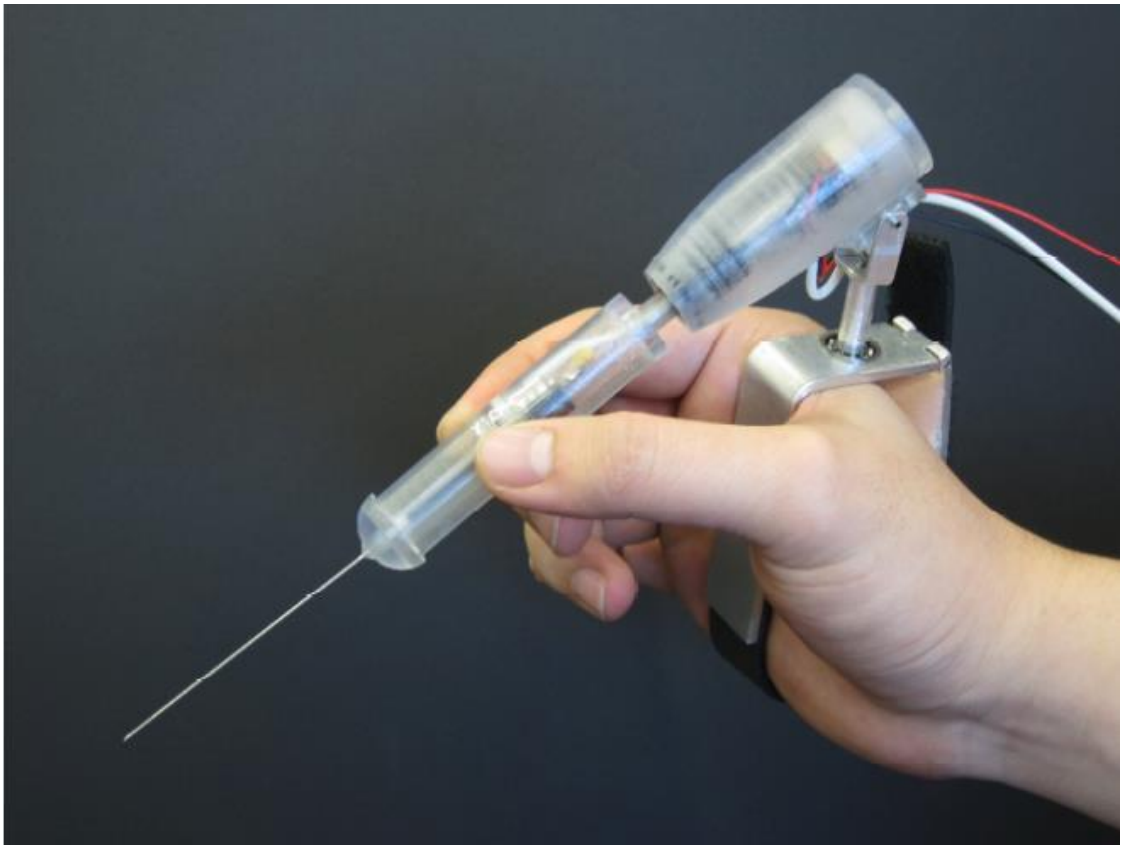


Figure 26. The HHFM Model-3. [136]

spring served to eliminate a problem we encountered with the Model-1, in which the user would tend to end up at one or the other extreme of the voice coil's range of motion.

The wearable portion of the HHFM Model-3 has a mass of 80 grams, about half the mass of the Model-1. The brace is an adapted version of the HHFM Model-2 brace, connecting to the underside of the actuator housing. The completed HHFM Model-3 is shown in **Figure 26**.

4.1.3 HHFM Model-3 Control

After the considerable mechanical redesign in the Model-3, we were satisfied with the general performance of the Analog Devices microprocessor and the electronics control box powering the Model-2, and continued to use it to control the Model-3. Some major changes were made to the

firmware driving the HHFM Model-3 to maintain future extensibility and to interface with experimental systems. The updated electronics schematics and source code for the HHFM Model-3 are included in **Appendix C.1** and **C.2**. To accommodate the greater complexity of the real-time sensing and control algorithms (described below), the HHFM Model-3 operating frequency was slowed from 10 kHz to 5 kHz, still well above the maximum 1 kHz bandwidth of touch perception.

In exchange for the voltage-centered magnification scheme utilized in the Model-2, the Model-3 sensing and actuation subsystems were calibrated and a force-centered magnification scheme was implemented. That is, instead of working primarily with sensor and actuator voltages, our firmware now manages force directly. We follow this simple algorithm in each sensing-actuation loop,

$$f = A_{sensor} * (V_{sensor} - V_{preload}) \quad \text{(Eq. 15)}$$

$$F = k * f$$

$$V_{out} = A_{VCM} * F + B_{VCM} \quad \text{(Eq. 16)}$$

where the constants A_{sensor}, A_{VCM}, and B_{VCM} correspond to calibration constants obtained by linearly fitting voltage data to the sensed or generated force. Eqn. 15 computes the force from the difference between the current sensor voltage V_{sensor} and the resting sensor voltage $V_{preload}$, determined as before by a random sampling before contact is made. The resting sensor voltage is also subject to error due to changes in pose relative to gravity and thermal drift. The calibration constant A_{sensor} assumes linearity over the dynamic range of the sensor, which we will determine as described below. Eqn. 16 determines the voltage across the coil to produce some desired feedback force, F.

Figure 27. HHFM Model-3 calibration setup.

Because the HHFM contains both a sensor and actuator, we may use the device itself to perform calibrations of its individual subsystems. Firmware routines were written to cycle the actuator voltage through its full 2.5 V range in 20 mV increments. This behavior was assigned to a button on the front of the electronics enclosure. The HHFM Model-3 was then rigidly connected to a reference force sensor (Transducer Techniques, GS0-100; Temecula, CA) using two benchtop vices. The GS0-100 is a single axis bidirectional sensor, with a calibrated 100 gram (980.7 mN) range in both compression and tension. The GS0-100 is highly linear (0.05% full scale hysteresis and nonlinearity) and provides analog output. The output of the force sensor is instrumented to ±8 V, positive voltages representing compression. A NI-6009 DAQ (described above) was used to acquire voltage data from the internal HHFM sensor, the HHFM actuator, and the GS0-100 reference force sensor at a sampling rate of 1 kHz.

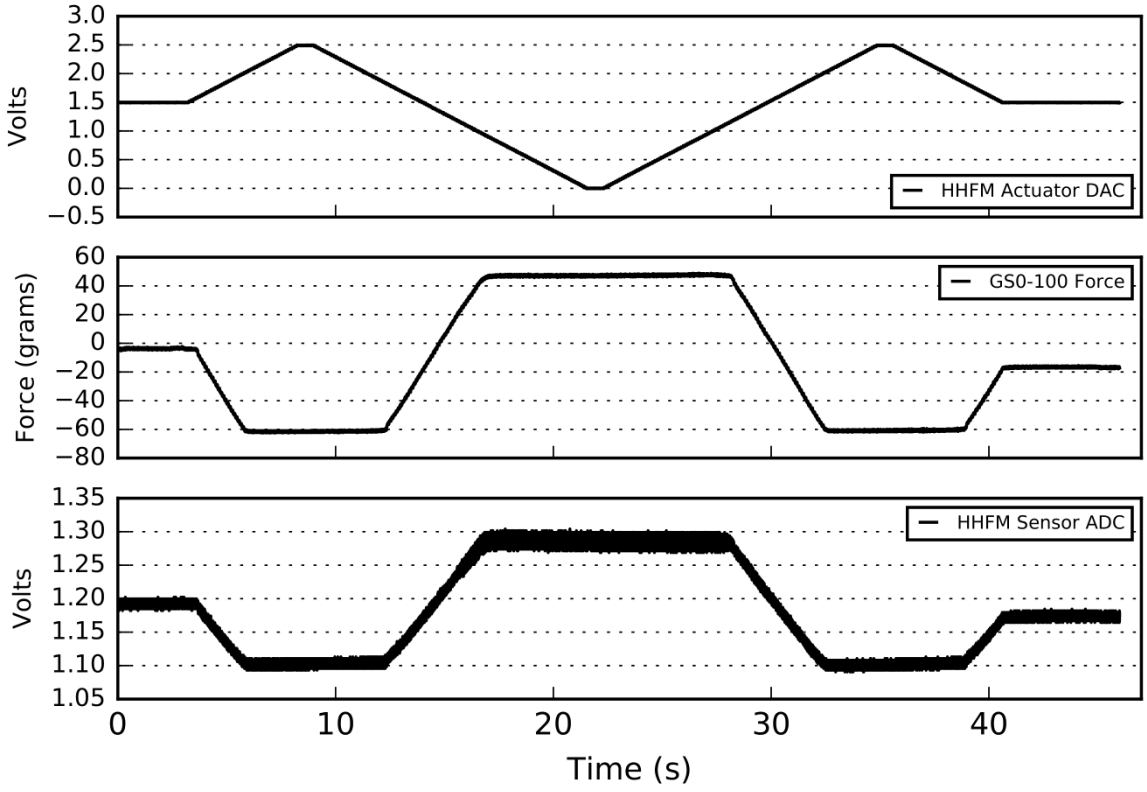


Figure 28. HHFM Model-3 calibration routine.

The calibration setup, depicted in **Figure 27**, uses magnets and a short metal tool tip to connect the HHFM to the reference force sensor**. Figure 28** shows the data gathered during calibration from the three sources (HHFM actuator voltage, GS0-100 reference force sensor, and HHFM force sensor), where positive force represents compression in the GS0-100. Care was taken to ensure that no force is read on the GS0-100 sensor in the starting position. Because there is a known conversion factor for the GS0-100 sensor (10 mV/gram-force), voltage data from our calibrated sensor were first converted to grams-force. The voltages in the sensor and actuator were then collected in 2 gram bins. Within each bin, the average sensor and actuator voltages were calculated, and a calibration curve was drawn by finding the linear least squares fit to these averages. Python scripts were written to perform this aforementioned analysis using the numpy

and matplotlib libraries. A number of calibration curves were collected, and their coefficients were averaged to obtain a mean calibration curve. Example calibration curves for each subsystem are shown in **Figure 29**.

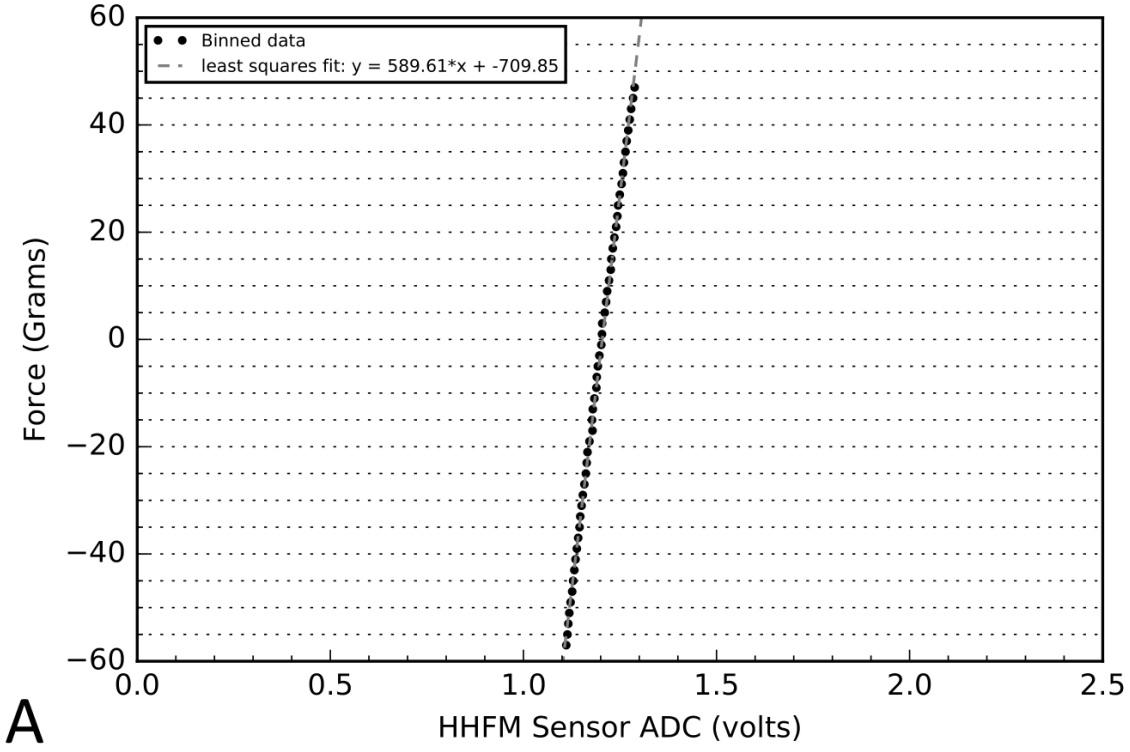
From this binning and averaging process, the force-to-voltage calibration equation for the sensing assembly is

$$y = 591.25x - 708.1 \quad \text{(Eq. 17)}$$

where we use only the slope as A_{sensor} in our firmware, given "taring" with the preload voltage (see Eqn. 15). Similarly, the average voltage-to-force calibration curve for the actuation assembly was found to be

$$y = 0.00861x + 1.45 \quad \text{(Eq. 18)}$$

Compared to the Model-2, the Model-3 normally operates in the "mute" condition, only actively rendering a feedback force when a footswitch is engaged. This architecture offers a number of benefits. First, a new estimate of the sensor preload voltage may be measured right before feedback forces are actuated. This helps reduce the effect of long-term hysteresis because changes in the preload voltage due to thermal effects or tool pose are naturally accounted for. Secondly, the foot switch is an interface commonly utilized by surgeons to control their surgical microscope or other actuated tools, such as the vacuum probes used for phacoemulsification. Using such an interface offers the clinician a familiar access to the magnification features of the device.



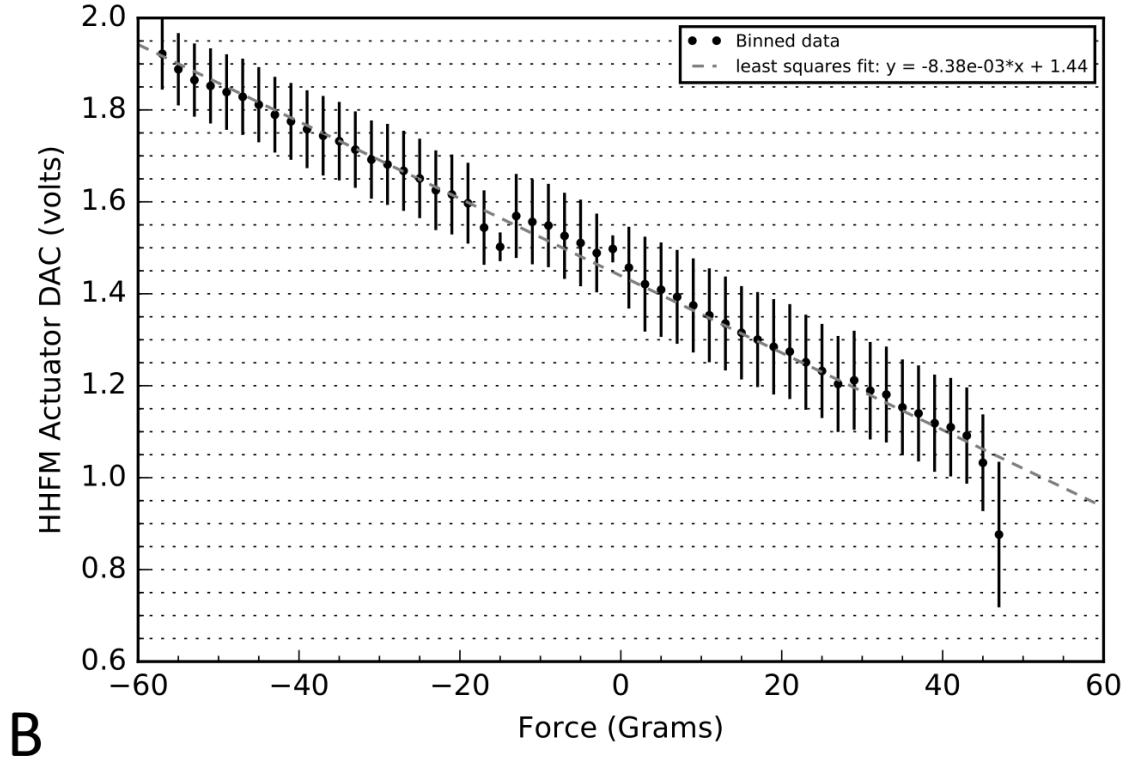


Figure 29. HHFM Model-3 (A) sensor and (B) actuator calibration curves.

4.2 HHFM MODEL-3 PSYCHOPHYSICAL EXPERIMENTS

Psychophysical experimentation using the HHFM Model-1 included three experiments focused on assessing the effects of force magnification on the afferent pathway. Another three experiments, described in the following sections as Experiments 4, 5, and 6, were conducted to characterize one's efferent pathway in surgically relevant tasks, with and without force magnification. Experiment 4 simulated a common surgical event: membrane puncture. From this work, we were able to build a model of the human motor control response using kinematic data collected from the MLHD. In Experiment 5, we showed that our model of efferent behavior was sufficiently sensitive to differentiate between a user's dominant and non-dominant hands. Because we had already shown that feedback forces are perceived as if they were real forces (see Experiment 3), both of these puncture experiments were done without force magnification – users interacted with a normal tool so as to understand baseline human motor control behavior. In Experiment 6, we directly investigated the relationship between magnification and motor control in an isometric force generation task. We discuss each of these experiments in the following sections.

4.2.1 Experiment 4: Membrane Puncture

In this experiment, we sought to simulate a common surgical task: membrane puncture [138]. Punctures are required ubiquitously in the medical context – from lumbar puncture, to intravenous line placement, to the initial puncture of the cornea to access a cataract. Suturing requires repeated punctures, and for delicate tissues or vessels, care is required to ensure the tissue does not tear. Clearly, this type of maneuver is performed over a wide range of systems

and membrane stiffness. A critical measure is the distance traveled post-puncture. Travelling too far post-puncture can mean injury to distal structures, requiring precise motor control to perform these tasks. Efforts to prevent "back-walling," the puncture of the distal wall of a vessel, for example, fall within this type of precise control.

Eight participants (4 males and 4 females) from the local university community volunteered for this study with informed consent. A long handle, made of acrylic (14.34 cm long × 0.94 cm in diameter) and weighing 11.8 g, was attached to the MLHD flotor and grasped as shown in **Figure 30**. The attachment enables the flotor to be held and manipulated with a pen grip. The mass of the flotor alone is approximately 500 g.

During this experiment, movement was restricted to a front-facing plane, boundaries of which were enforced by the MLHD. The accessible region was 16 mm square, so the flotor could be moved vertically (parallel to gravity) and side to side (left and right). Centered at the MLHD origin, a 0.5 mm thick, circular viscoelastic membrane was rendered. In this way, a quarter-circle arc of membrane was available for interaction in all quadrants of the accessible region. Participants were instructed to "push" into a membrane in Quadrant III (leftward and downward with respect **Figure 30**), or "pull" into a membrane in Quadrant I (rightward and upward in **Figure 30**). At all times, the weight of the flotor and handle was cancelled by an upward force, and experienced an external viscosity (15.0 N*s/m) to enhance stability. The radial distance to the inner wall of the membrane was randomly drawn from a uniform distribution over 0.5 mm – 1.5 mm at the beginning of each trial. This was implemented to prevent participants from anticipating the position of the membrane over the course of the experiment.

The Kelvin-Voigt model of viscoelasticity was chosen for this experiment. Viscoelastic membranes are so called because they exhibit characteristic behaviors in response to imposed stresses or strains. For example, these models exhibit creep, the slow buildup of strain under a constant stress; stress relaxation, the reduction in stress in response to a sustained strain; and hysteresis [139], [140]. These models are well suited to describe biological tissue, and have relatively simple parameters. The Kelvin-Voigt model consists of a spring in parallel with a damper, such that their strains are equal, and the force generated by each component depends respectively on the incursion into the membrane and the velocity within the membrane.

The membrane was rendered at 4 stiffness levels: 600, 1130, 1660, and 2200 N/m, and a constant viscosity of 20 N*s/m [139], [140]. Right before puncture, the component of the total force due to traditional spring action were 0.30, 0.56, 0.80, and 1.10 N, respectively. (There was also a small viscous force, which depended on the velocity within the membrane.) Puncture was

Figure 30. MLHD membrane puncture set up.

simulated by releasing the membrane force upon reaching the far membrane wall. Upon exiting the membrane, the flotor was returned to its weightless status and experienced a viscosity of only 15.0 N*s/m. The simulation was written in C++ and rendered at 1 kHz. The source code for the simulation is included in **Appendix C.3**.

The experimental variables were the 4 levels of stiffness and 2 directions of movement, push and pull. Each combination was tested 10 times per participant, in random order. Participants also completed a training block of punctures to accustom themselves with the device and task prior to the experimental blocks. The membrane stiffnesses used in the training block were different from those tested in this experiment. The entire experiment lasted approximately 1 hour.

Each trial began with a prompt on an LCD screen instructing the participant to "PUSH" or "PULL." The participant could then freely move the flotor until encountering the membrane and subsequently fully puncturing it. The only instruction thereafter was that the distance travelled post-puncture should be minimized. The trial ended 2 seconds after the participant exited the membrane, at which point a signal was given to release the handle, and the flotor was automatically returned to the origin. Flotor position data was acquired at 200 Hz.

Data were fit in three phases to differentiate between distinct stages of post-puncture motor control behavior.

Phase 1 of modeling was based on the force exerted by the user at the instant of puncture, which is given by

$$F_0 = k_s * D + \delta_\gamma * v_0 \qquad \text{(Eq. 19)}$$

where F_0 is the force exerted by the user on the flotor; k_s is the spring stiffness; D is the distance traveled in contact with the membrane, which by definition the thickness of the membrane; δ_γ is

the environment viscosity, 20 N*s/m; and v_0 is the velocity at breakthrough. The velocity at breakthrough was calculated by taking the slope of the linear fit of the final pre-breakthrough positions within the membrane.

A cubic function was then fit to the first 9 positions (45 ms) post-puncture, the first derivative of which was used to solve for the precise breakthrough time, t_0, which generally fell between data points. The second derivative of the cubic position function was used to estimate the initial acceleration, a_0, and from the known F_0, to estimate the inertial mass, m.

In Phase 2, the force on the tool is assumed to decline exponentially from the initial force F_0 calculated from Eqn. 19, with an additional viscous damping factor from the virtual environment. Therefore the force exerted on the tool is described by

$$F_e(t) = F_0 * \exp(-t/\tau) - \gamma * v \qquad \text{(Eq. 20)}$$

where $F_e(t)$ is the applied force; F_0 is the force at breakthrough; t is the time relative to breakthrough; τ is the time constant of exponential decay; and $\gamma * v$ is the viscous force generated by the virtual environment due to viscosity (γ), 15.0 N*s/m, and the current velocity (v). Exponential decay was chosen because it is a common descriptor for stress decline in biological tissues. In this second post-puncture phase, the participant has not yet initiated active control over the tool and is modeled as passively relaxing.

Eqn. 20 was fit to the collected position data to estimate the exponential time constant, τ. Using the previously obtained estimates of inertial mass, force, and acceleration at breakthrough, the model calculates the tool tip position at some time, t, in 0.5 ms steps. At each time step, the instantaneous acceleration is computed as $F_e(t)/m$, which is then used to compute velocity and position. A critical time point was chosen, generally twice the time to peak velocity, and the position at that time point was used in the cost function to fit the model. The value of τ was

chosen by iteratively running the model so as to minimize the difference between the predicted position and the measured position at the critical time point. **Figure 31** shows the effect of membrane stiffness (and therefore the initial force, F_0) and the exponential decay constant in predicting position and velocity post puncture using Eqn. 20.

Phase 3 of our modeling describes behavior long after the instant of puncture, when the participant has initiated braking, and is actively minimizing the distance travelled. The onset of this phase was determined by identifying the time at zero velocity, when the participant has finally stopped, and working backwards to identify the time at which the measured position deviates from the predicted positions from Phase 2. In this last phase of analysis, local polynomial regression was used to smooth the velocity data. For 500 ms past this deviation point oscillations (reversals in position) were identified and characterized. **Figure 32** shows real data collected from this experiment, and their associated model fits. The black dot in **Figure 32** indicates the calculated deviation point.

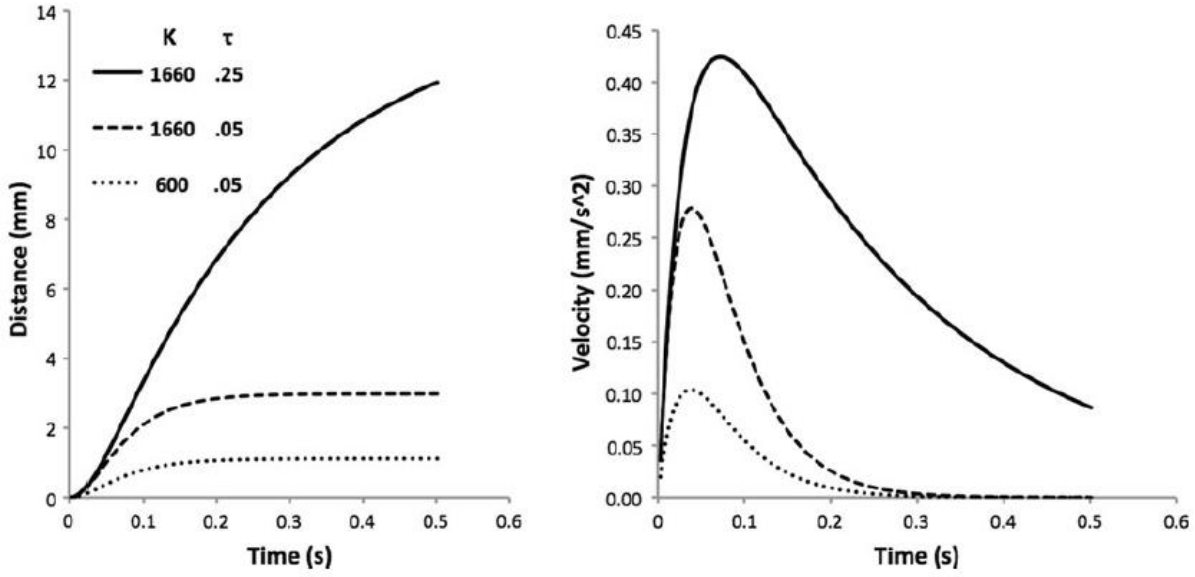


Figure 31. Effect of membrane stiffness (K) and exponential time constant (τ) on predictions of flotor position (left) and velocity (right) post-puncture. [138]

By this model fitting process, several parameters (e.g. m, τ, deviation point, etc.) were calculated for each participant over the 4 stiffness conditions and 2 directions. Repeated measures ANOVAs were performed to analyze the collected data. The critical significance value was set at $\alpha = 0.05$. Criteria for excluded trials (overall less than 7%) that were discarded from the 640 total trials are described as follows.

In the initial linear fit of pre-puncture position to obtain the velocity at burst, fits were accepted only if $r^2 \geq 0.85$. At least 3 positions, but no more than 10, were used in the first linear fit to measure velocity at breakthrough. 22 trials were thus discarded, almost all from trials with low stiffness, where the membrane was evidently not perceptible. Of the remaining 618 trials, the mean velocity at breakthrough decreased monotonically with membrane stiffness: 0.20, 0.13,

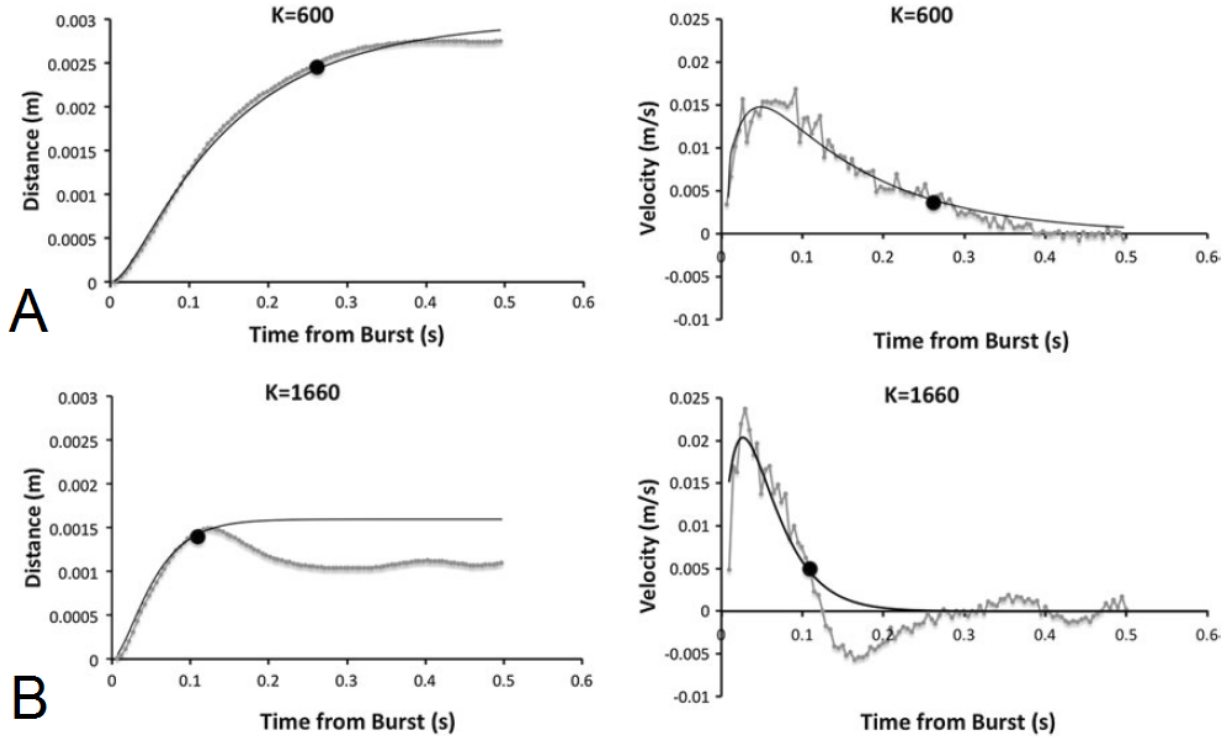


Figure 32. Motor control model fits of position (left) and velocity (right), and deviation point after puncturing membranes of stiffness (A) 600 N/m and (B) 1660 N/m. [138]

0.12, and 0.10 cm/s for stiffnesses k_s = 600, 1130, 1660, and 2200 N/m, respectively. These velocities correspond to the average total force – which includes both spring and viscous forces – within the membrane, of 0.31, 0.57, 0.84, and 1.10 N, as determined by Eqn. 19.

Trials were then fit assuming exponential decay (Eqn. 20) as previously described in Phase 2 of modeling. 25 trials were at this point eliminated because the first zero crossing of velocity came after 550 ms post-puncture (19 trials), or other erratic behavior that precluded model fitting (6 trials). Among the remaining 593 trials, the fits between the predicted and measured position at the critical time point averaged $r^2 \geq 0.99$, and only 2 trials had $r^2 \geq 0.90$.

The time to zero velocity and distance at zero velocity were extracted, as shown in **Figure 33**, where the error bars represent ±1 SEM. These measures gave some indication of stopping efficacy. We found main effects of stiffness in the ANOVA on time to zero velocity ($F(3,21) = 45.26$, $p < 0.001$) as well as on distance at zero velocity ($F(3,21) = 14.44$, $p <$

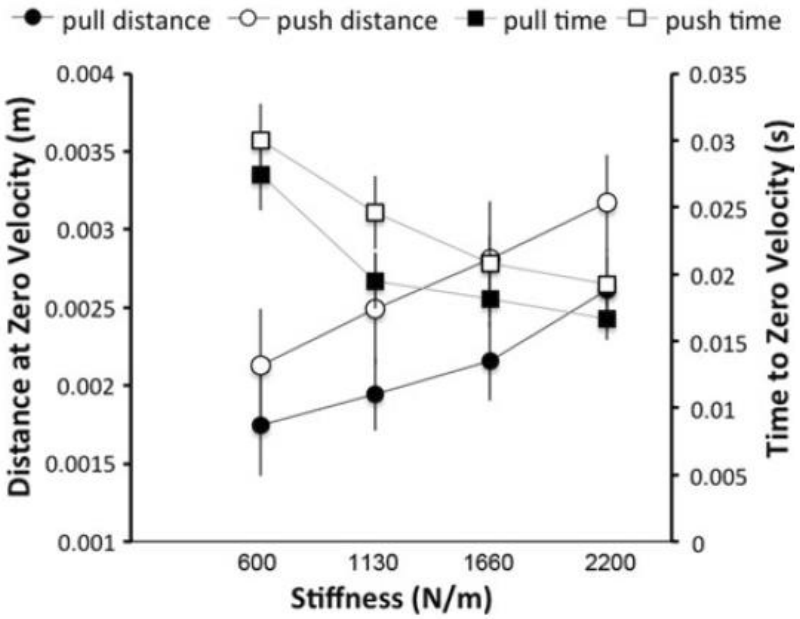


Figure 33. Stopping efficacy, as measured by distance traveled (circular markers) and time (square markers) to reach zero velocity. Error bars represent ±1 SEM. [138]

0.001). In general, participant's velocity declined faster with increasing stiffness, but not fast enough to prevent going further before it reached zero. Effects for movement direction were also found. Pull movements showed a reduced time to zero velocity ($F(1,7) = 11.83$, $p = 0.011$) and shorter distance to zero velocity ($F(1,7) = 11.33$, $p = 0.012$). This indicates that biomechanical control of stopping in the pull direction may be better than in the push direction.

Our modeling assumes a constant inertial mass during interaction with the virtual membrane. A consequence of this assumption is that we should expect a linear relationship between the acceleration at breakthrough and the force applied at burst. Such a linear relationship was found, depicted in **Figure 34**, where error bars represent ±1 SEM. The average mass for the subject-flotor coupling was found to be 522 g in the pull direction, and 507 g for push. These estimates are very close to the actual mass of the flotor-handle system, approximately 500 g in total. The range of masses for individual subjects was between 497 to

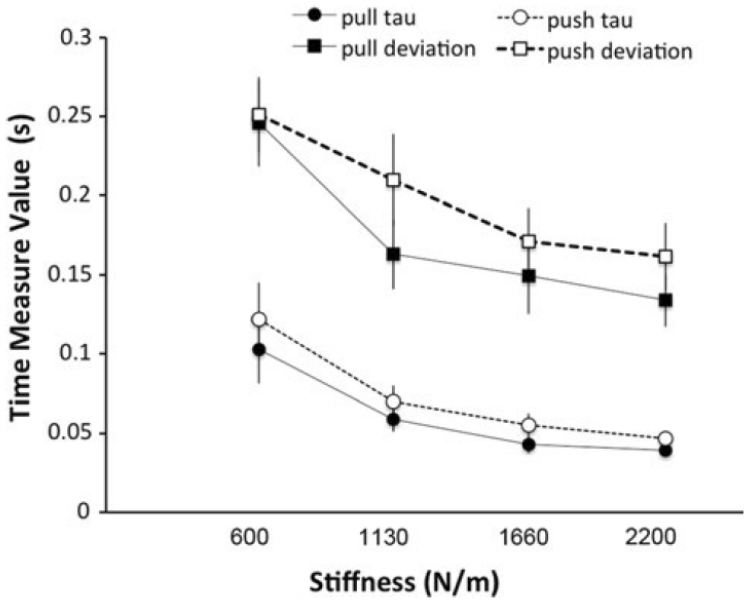


Figure 34. Time measures tau and deviation point variation with respect to membrane stiffness. Error bars represent ±1 SEM. [138]

539 g, with a standard deviation (SD) of 17 g. The within-subject SD for mass estimates was on average 81 g for pull and 83 g for push, which represents approximately 16% of the intrinsic mass estimate. From these results we can conclude that subjects added little mass to the system. ANOVAs on the mean mass estimates found no significant effects for stiffness or direction.

We found that the time constant for exponential force decay, τ, tends to decrease with membrane stiffness. This bolsters our finding that the time to zero velocity decreases with increasing membrane stiffness. The within-subject SD for τ averages 0.06 seconds in pull and 0.07 seconds for push. Further, the within-subject SD was less than the between-subject SD for all combinations of our experimental variables. Similarly, the deviation point – the time at which the recorded data deviates from the model's prediction – also decreased with membrane stiffness. This effect may be attributed to better detection of puncture due to the larger drop in force with stiffer membranes.

The relationships between τ and deviation point with membrane stiffness, respectively, are shown in **Figure 35**, where the error bars represent ±1 SEM. A $2 \times 2 \times 4$ ANOVA was conducted to assess how these time measures, τ and deviation point, changed with the experimental variables of stiffness and direction. We found significant effects of time measure ($F(1,7) = 46.60$), which reflects the temporal lag between τ and the deviation point. This lag averages 199 ms, and is relatively consistent across experimental conditions. Accordingly, we found no significant interactions with time measure. The effect of stiffness was significant ($F(3,21) = 34.08$, $p < 0.001$), as seen in the faster force decline for stiffer membranes. The main effect of movement direction was also significant ($F(1,7) = 19.50$, $p = 0.003$), due to the faster force decline while pulling than pushing.

Generally, participants did not come to a smooth stop following puncture, as would have been indicated by the exponential decay phase in applied force. Indeed, participants typically stopped closer to the membrane boundary than was predicted by Phase 2 of our modeling, after some oscillation. 500 ms after the deviation point, the actual stopping distance was 12% below the predicted distance for the lowest stiffness spring, and 23% below the predicted distance for the three higher membrane stiffnesses. These differences were not found to be significant ($F(2,14) = 1.22$, $p > 0.05$). Further, these differences were maintained between movement directions.

Oscillations were detected from the position data, after having been smoothed with an 11-sample average. An oscillation was defined as three successive reversals in movement direction.

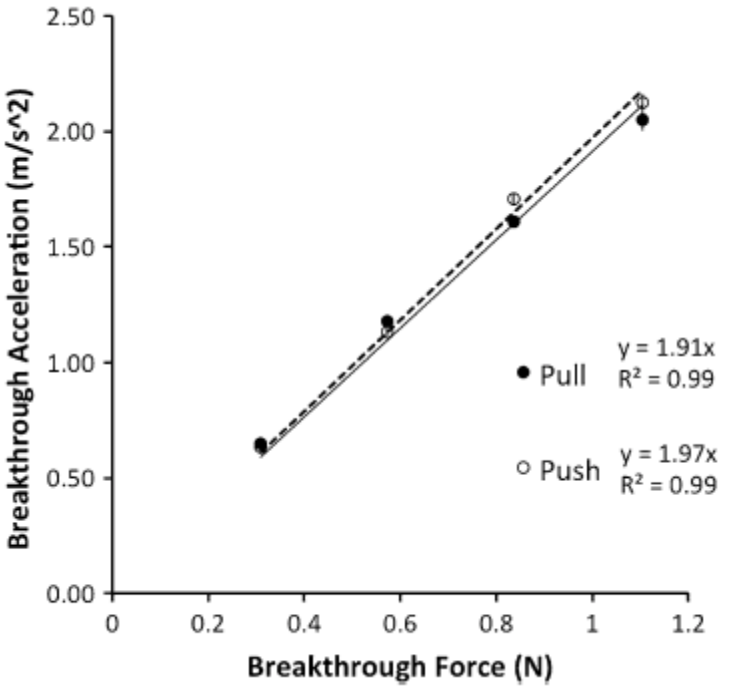


Figure 35. Relationship between flotor acceleration and breakthrough force in push and pull directions, with least squares constants forced to zero. [138]

The amplitude of oscillation was defined as the absolute difference between the position at the middle reversal and the mean position at the first and third reversals. The oscillation duration was defined as the time between the first and third reversals. Of the 593 trials analyzed, 71% exhibited at least one detectable oscillation, 32% had at least two, and 13% had three or more. Within a single trial, the range of detected oscillations was 0 to 7, with an average of 1.27.

To assess the differences in oscillatory behavior with respect to membrane stiffness, an ANOVA was conducted on data from the 192 trials that exhibited exactly two oscillations. The factors examined in this analysis included the oscillation number (first or second), membrane stiffness, and movement direction. The peak amplitude of the first and second oscillations averaged 0.12 and 0.03 mm, and lasted 182 and 122 ms, respectively. The change in amplitude from the first to second oscillation was found to be significant ($F(1,184) = 110.30$, $p < 0.001$), as was the change in duration ($F(1,184) = 40.44$, $p < 0.001$). The oscillations themselves were not found to be independent. A positive correlation was found between the first and second oscillation's amplitude ($r = 0.22$, $p = 0.003$), while a negative correlation was found between the durations ($r = -0.34$, $p < 0.001$).

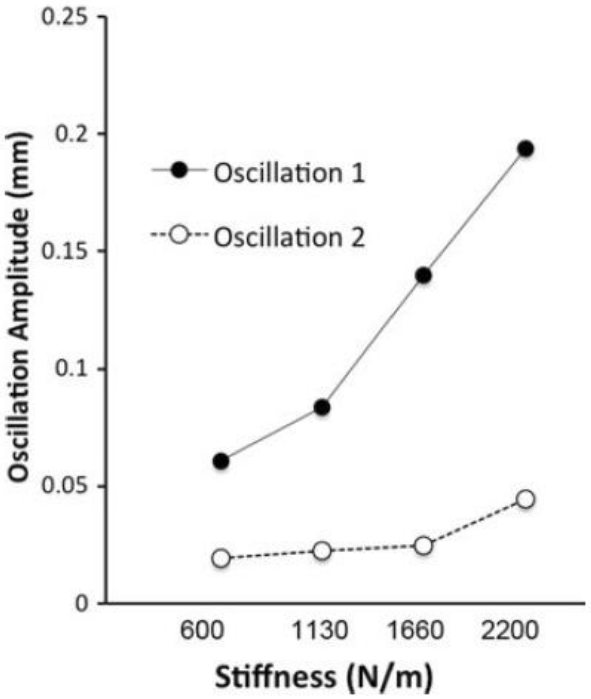


Figure 36. Effect of membrane stiffness on first and second oscillation amplitude. [138]

Stiffness affected both oscillations. Peak amplitude showed a main effect of stiffness ($F(3,184) = 14.74$), and an interaction between oscillation number and stiffness ($F(3,184) = 8.15$, p's < 0.001). These effects are shown in **Figure 36**. While the effect of stiffness was greater in the first oscillation, it was also significant for the second ($F(3,188) = 6.01$, $p < 0.001$). Oscillation duration also increased with stiffness, ranging on average from 139 ms for the softest membrane, increasing to 168 ms for the stiffest membrane. The effect of stiffness on duration was significant ($F(3,184) = 4.76$, $p = 0.003$). We did not find any significant interaction between stiffness and oscillation number, meaning the duration of either oscillation differed little.

In this work, we developed a model of the motor control response following membrane puncture. As expected, we found that the distance at zero velocity increased with membrane stiffness. We were also able to recover an estimate of the reaction time, approximately 130 ms, which is consistent with reaction times that are neurological in origin, following a "hyperdirect" pathway [141]. Reaction times at this level support a hypothesis in which terminating an initiated action is mediated by a cortical-to-subcortical pathway, originating from the right inferior frontal cortex (rIFC) to the subthalamic nucleus in the basal ganglia [142], [143]. Additional work indicates that rIFC-initiated inhibition is mediated by the supplementary motor cortex, which includes the pre-supplementary area [144], [145].

These measures – time and distance to zero velocity, reaction time, and tau – may be used in future to inform anti-skid mechanisms in the HHFM, and to evaluate their efficacy.

4.2.2 Experiment 5: Handedness in Membrane Puncture

Building on our membrane puncture work, we chose to evaluate the sensitivity of our model by examining how our motor control capabilities differ between dominant and non-dominant hands [146]. In the surgical context, this is an important consideration since many surgeries are bimanual. Highlighting the differences between hands will lead to force feedback strategies specialized for use with either the dominant or non-dominant hand.

The common understanding of handedness is that one's dominant hand is overall more dexterous and capable, but there is substantial evidence that the asymmetry in performance is in fact, task dependent. The neural mechanisms underlying handedness an active field of research with several competing hypotheses. The generally accepted model assumes that the brain differentiates control of the left and right hands, that there is some asymmetry in this control, and these differences are perpetuated through the motor control system. That each hand is controlled by the contralateral hemisphere is not sufficient to explain this difference in performance. What is needed is functional asymmetry in the hemispheres themselves, as has been demonstrated in M1, the premotor cortex, and in supplementary motor areas [147]–[153].

Further, this cortical asymmetry seems to depend on the extent to which a task requires closed-loop feedback, and the measures of performance. A number of experimenters have provided evidence for and against the feedback hypothesis, which states that the left hemisphere controls the open-loop aspects of movement, which rely mostly on pre-planning, whereas the right hemisphere controls the aspects of a task that depend on sensory feedback [150]. Task-dependent hypotheses assign responsibilities to one cortical hemisphere over another, for example, where the left hemisphere controls movement trajectory, while the right hemisphere controls limb position and posture [149], [150]. Still another view is that the contribution of each

hemisphere in controlling task movement is dynamically influenced by movement complexity, with more complex movements employing more distributed neuronal networks, often through communication between hemispheres by way of the corpus callosum [152].

Experiments in the literature cited above have concentrated on tasks like reaching, tapping, and aiming, which rely on visual feedback. Less is known, however, about whether handedness emerges when considering tasks that do not use visual information. In this experiment, therefore, we applied our membrane puncture model to test whether it can differentiate between the dominant and non-dominant hand. The membrane puncture task depends solely on haptic feedback, enabling us to begin to tease apart visual feedback effects.

Nine right-handed adults (4 male and 5 female) volunteered for this study with informed consent, and were tested according to an IRB approved protocol. Handedness was determined by the 10-item version of the Edinburgh Handedness Inventory (EHI). The EHI assesses handedness by asking the respondent whether they prefer to use their left or right hand for common tasks like writing, drawing, using scissors, or striking a match [154]. The EHI score is obtained by subtracting the number of left-handed responses from right-hand responses, dividing by the sum of both, and multiplying by 100. All participants scored above 40, indicating right-handedness.

As in Experiment 4, the MLHD was used to render a virtual viscoelastic membrane. Again, only the frontal plane was accessible to the participant. All boundaries were enforced by the MLHD. The spherical viscoelastic membrane was again rendered at a random distance 0.5 to 1.5 mm from the origin, so as to prevent participants from anticipating the location of the membrane. With either their right or left hand, subjects were instructed to move in either a forward and downward direction (push) or a backward and upward direction (pull). For left-handed movements, the push direction involves movement in Quadrant IV, whereas pull

movements happen in Quadrant II of the available fronto-parallel region (see **Figure 30**). As was the case in Experiment 4, the instruction to participants was to puncture the membrane, and to minimize the distance traveled post-puncture.

In regions of space prior to and after the virtual membrane, the flotor was rendered weightless by an upward force, and stabilized by a viscosity of 15.0 N*s/m. The Kelvin-Voigt membrane was rendered at 3 stiffness values: 700, 1400, and 2100 N/m, and a constant viscosity of 20 N*s/m. Membrane thickness was maintained at 0.50 mm, corresponding to maximum membrane forces of 0.35, 0.70, and 1.05 N. Therefore the experimental variables that were manipulated were stiffness (3 values), 2 directions (push/pull), and hand used to manipulate the flotor (right/left). The haptic simulation was written in C++ and rendered at 1 kHz.

Push and pull trials were completed in separate blocks of trials, with pull always performed first. This was done because the pull direction was known *a priori* from Experiment 4 to be a more dexterous movement. We therefore wanted to measure performance in this direction without any experience with the task. Within each block, each combination of hand and stiffness was presented 10 times in random order. Prior to attempting the experimental blocks, participants could practice the task on 10 trials. The experiment lasted approximately 1 hour.

Each trial began with an instruction of which hand to use appearing on an LCD screen in front of the participant. Subjects then feely moved the handle until the membrane was encountered and punctured. As in Experiment 4, the experiment ended 2 seconds following puncture, whether or not the participant had fully stopped. The flotor was then released and automatically returned to the origin. Position data over the course of each trial were recorded at 200 Hz.

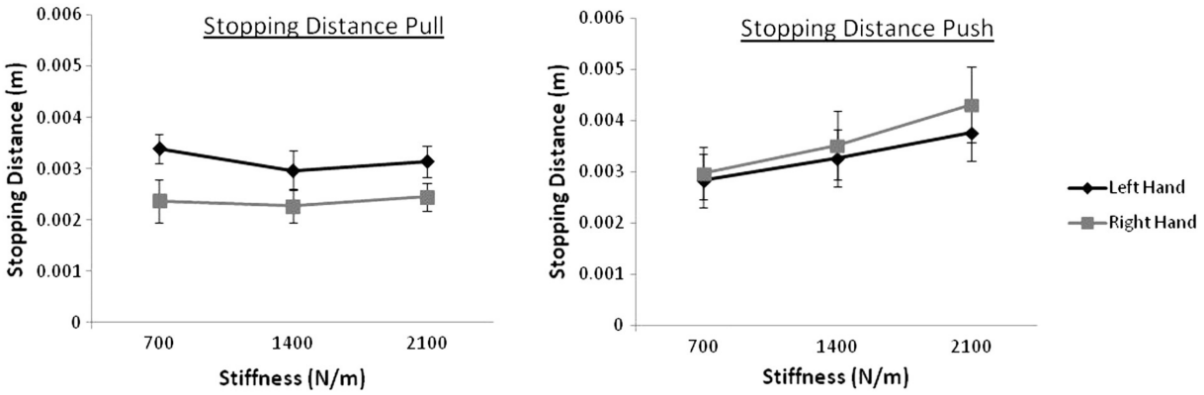


Figure 37. Comparing distance at zero velocity for pull (left) and push (right) for left and right hands. Error bars represent ±1 SEM. [146]

One participant was removed from the analysis due to their difficulty in performing the task, even after extensive practice. A total of 480 push trials and 480 pull trials were therefore obtained from this experiment. In the process of fitting the data to the previously described three-phase process (see Experiment 4), we removed trials that exhibited relatively low correlation in the fit of velocity of burst ($r^2 < 0.8$, or less than 3 successive pre-penetration position measurements). We also removed trials with an extreme in slow stopping (time to zero velocity greater than the average zero crossing time + 3 standard deviations). Finally, we removed trials with irregular behavior that prohibited model fitting ($\tau > 1$, or deviation point > 1). Following this screening process, the remaining 444 pull trials (92.5%) and 369 push trials (76.9%) were included in the analysis.

Stopping behavior was characterized by the distance and time to reach the first instance of zero velocity. **Figure 37** and **Figure 38** compare these measures, respectively, as a function of membrane stiffness and hand. For each measure, two-way ANOVAs were performed separately for push and pull movements.

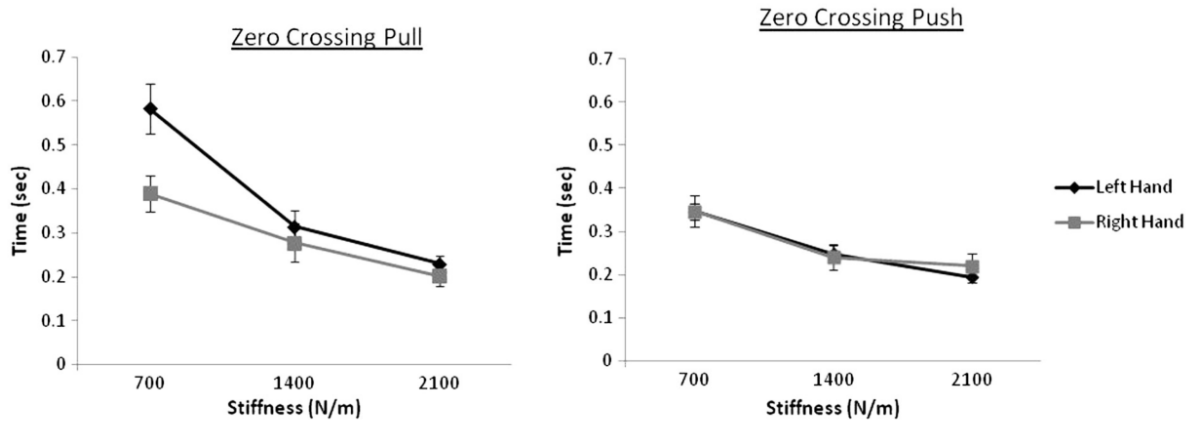


Figure 38. Comparing time to zero velocity for pull (left) and push (right), as a function of membrane stiffness and hand. Error bars represent ±1 SEM. [146]

With respect to the stopping distance (**Figure 37**), we found that only push movements yielded greater incursion post-puncture with increasing stiffness ($F(2,14) = 10.91$, $p = 0.001$), although there was a similar trend for with the right hand in the pulling direction. The pull ANOVA showed a significant main effect for the hand ($F(1,7) = 15.49$, $p = 0.006$). In this case, using the left hand led to a greater skid distance before stopping. Overall, these stopping measures indicate a disadvantage for the non-dominant hand only for the pulling movement, which places more demands on dexterity than pushing.

We found that the time to zero velocity (**Figure 38**) decreased with increasing stiffness, as was the case Experiment 4. The effect was significant for both push ($F(2,14) = 71.94$, $p < 0.001$) and pull ($F(2,14) = 30.77$, $p < 0.001$) conditions. An effect for the hand was only found in the pull direction, identified as a significant interaction effect between hand and stiffness ($F(2,14) = 6.43$, $p = 0.01$). Further analysis using the Fisher Least Significant Difference (LSD) test found a disadvantage for the left hand with the softest membrane in particular ($p = 0.039$).

Table 2. Best fit model parameters and analysis. [146]

	Mean by hand		ANOVA F statistics		
	Left	Right	Stiffness df 2, 14	Hand df 1, 7	Interaction df 2, 14
Phase I: physics of burst					
Velocity at burst (m/s)	0.0016	0.0013	$F = 15.42$ $p < .01$	$F = 1.69$ $p = .23$	$F = .70$ $p = .52$
Acceleration at burst (m/s^2)	1.408	1.388	$F = 1010.22$ $p < .01$	$F = 0.38$ $p = .56$	$F = 0.46$ $p = .64$
Mass coupling (kg)	0.52	0.51	$F = 17.85$ $p < .01$	$F = 0.07$ $p = .80$	$F = .80$ $p = .83$
Phase II: biomechanical factors					
τ (force decay: s)	0.089	0.066	$F = 35.05$ $p < .01$	$F = 16.14$ $p < .01$	$F = 6.99$ $p < .01$
Phase III: controlled braking					
Deviation point (s)	0.22	0.22	$F = 7.55$ $p < .01$	$F = 1.88$ $p = .68$	$F = 1.32$ $p = .30$

Parameters for our three phase model of motor control behavior were fit according to the process previously described. Results from ANOVAs computed on these measures are summarized in **Table 2**. A significant difference between hands was found only in the τ measure, representing the time constant of exponential decay. **Figure 39** shows how τ changes as a function of stiffness and hand. Our analysis indicated both a main effect for the hand, as well as an interaction effect between hand and stiffness. This result implies that the differences between braking performance between the two hands arose from Phase 2 of our model.

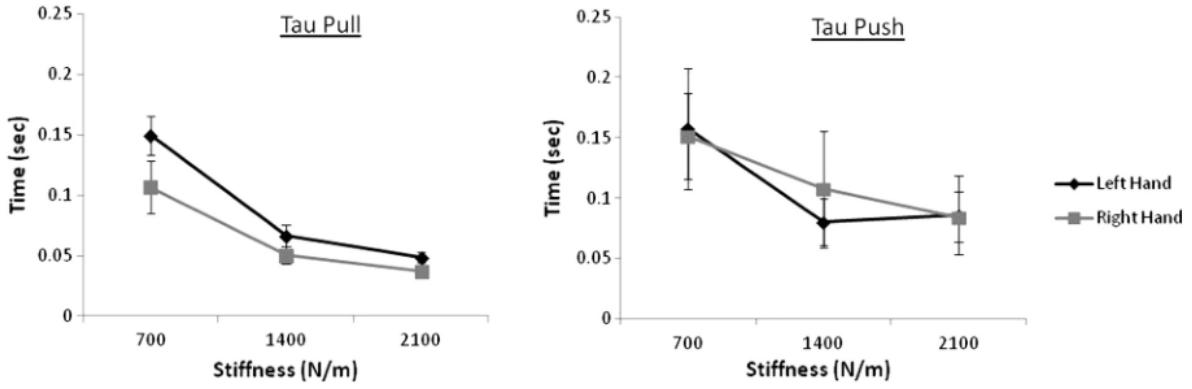


Figure 39. Comparing model parameter τ for pull (left) and push (directions), as a function of membrane stiffness and hand. Error bars represent ±1 SEM. [146]

In addition to our analysis of subject means under each condition, we calculated per-subject standardized (z) scores for each measure, within each stiffness level. Z-scores enable us to group together subjects by the various experimental conditions or parameters, adjusting for their individual performance in the task. Specifically, the z-score is the number of standard deviations by which an observation differs from the subject mean in each condition, and is calculated using

$$z = \frac{X_i - \overline{X}}{s_x} \qquad \text{(Eq. 21)}$$

where X_i is an observed data point of some experimental condition X; and s_x is the subject standard deviation of X; and $\overline{X}$ is the subject mean for X.

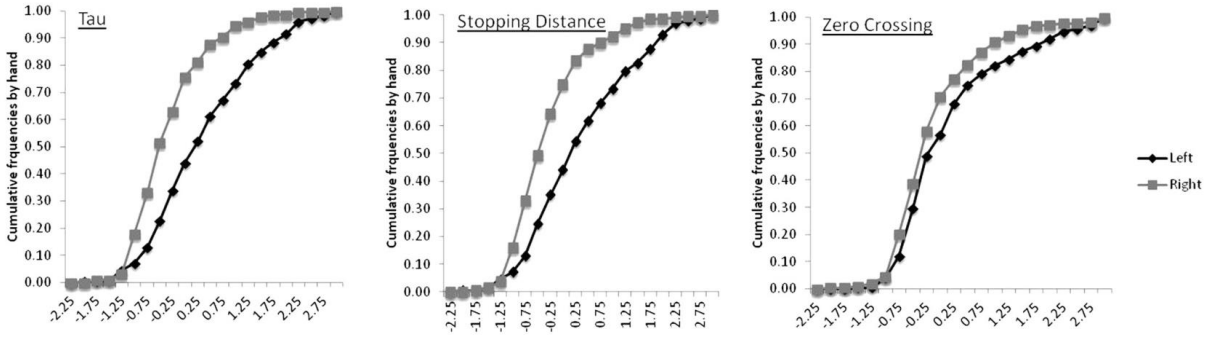


Figure 40. Cumulative distribution of model parameter τ, and two behavioral measures, stopping distance and time to zero velocity, for the left and right hands, by pooled z scores. [146]

Statistical analysis of these z-scores evaluate whether the probability distributions for each measure differ between hands, independent of any effect of stiffness or individual differences. Using the calculated z-scores, we performed two-sample Kolmogorov-Smirnov (K-S) tests on the cumulative distribution functions (CDFs), as shown in **Figure 40**. The two-sample K-S test is a nonparametric statistical test that determines whether two samples are drawn from the same distribution by calculating the test statistic

$$D_{n,n'} = \sup_x |F_{1,n}(x) - F_{2,n'}(x)| \qquad \text{(Eq. 22)}$$

where $F_{1,n}$ and $F_{2,n'}$ are the two empirical distribution functions from the two samples; and sup is the supremum function [155], [156]. The supremum function is similar to taking the maximum, but applies to special subsets of data. Precisely, the supremum of some subset S of a partially ordered set T is the least element in T that is greater than or equal to all elements in S. The K-S test may also be used to test whether a sample is drawn from a known distribution (e.g. the normal or Poisson distributions) of a random variable, or as a goodness of fit test.

In the two-sample K-S test, the null hypothesis that the two CDFs are drawn from the same distribution is rejected at confidence level α if

$$D_{n,n'} > c(\alpha)\sqrt{\frac{n+n'}{nn'}} \qquad \text{(Eq. 23)}$$

where n and n' are the sizes of the first and second samples, respectively. The value of $c(\alpha)$ is published in statistical tables: for $\alpha = 0.05$, $c(\alpha) = 1.36$; for $\alpha = 0.001$, $c(\alpha) = 1.95$.

By inspection, it is evident that the CDFs for the model parameters from the right hand (grey) differ somewhat those from the left hand (black). These differences are significant for all measures: τ (one-tailed $p < 0.001$), distance at zero velocity (one-tailed $p < 0.001$), and time to

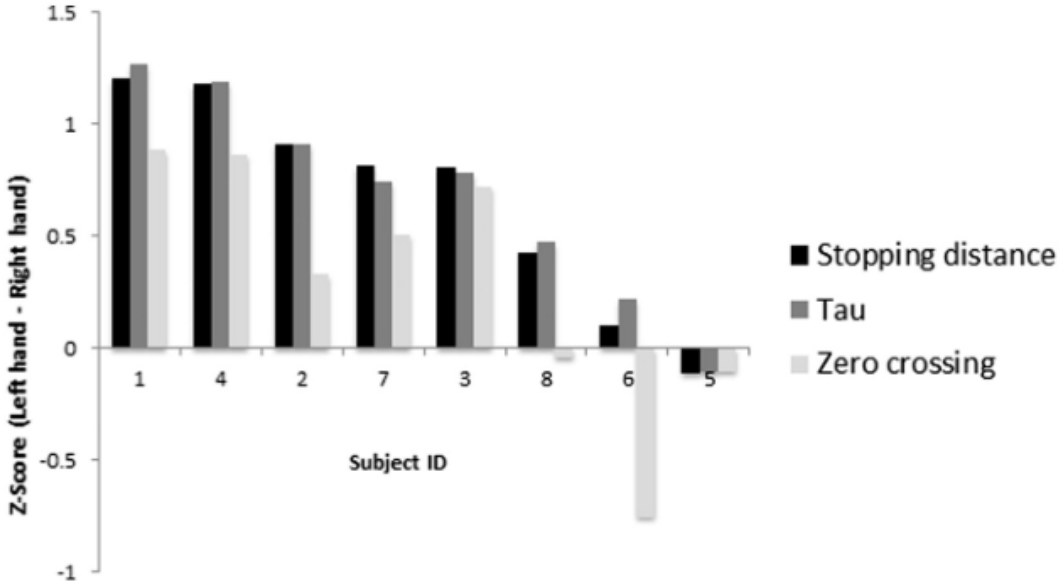


Figure 41. Z-score difference between left and right hands for model parameter τ, and two behavioral measures, stopping distance and time to zero velocity. [146]

zero velocity (one-tailed $p = 0.027$). From these results we can conclude that the samples drawn from left and right hands are drawn from two different distributions.

We also used z-scores to evaluate handedness effects at the individual participant level. **Figure 41** shows the magnitude of the difference between the left and right hand z-scores for the model parameter τ and the two behavioral measures, stopping distance and time to zero velocity. A positive difference indicates the expected pattern of hand dominance, with better performance by the right hand. As expected, five of the eight subjects showed consistent right-hand dominance across the three measures, whereas two showed little difference between the hands (less than 0.5 z-score difference for the right hand), and one showed relatively small or null effects (negative z-score difference for the right hand).

This work contributes to the growing body of knowledge regarding the asymmetry in manual dexterity due to handedness. From our kinematic observations of ballistic motion following puncture, we can conclude that major differences in motor control originate from the biomechanical damping phase of our model. Our work also bolsters the concept of a "spectrum" of handedness, given our results comparing the difference between z-scores between the left and right hands.

4.2.3 Experiment 6: Magnification Effects on Motor Control

Until recently, little attention has been paid to the interaction between augmented sensory perception and motor control. Devices like Micron (see Section 2.3.1) compensate for tremor by actively changing the pose of the tool tip to stabilize movement. The HHFM instead augments the clinician's sensory experience, such that their own tremor may be controlled because they are matching and interacting with magnified forces. However, as we saw in our membrane puncture work (Experiments 4 and 5), magnifying membrane stiffness can potentially contribute to deleterious effects, for example, increasing the distance travelled post-puncture.

The goal of this last experiment, therefore, is to characterize one's motor control behavior while performing another surgically relevant task – sustained force generation – with and without force feedback. Precisely controlling the force applied to tissue over a relatively long period of time is a common task in microsurgery, for example, when performing a capsulorhexis as part of cataract extraction. In this portion of the procedure, smooth vectoring of force can prevent intraoperative complications like PCT or radial tear, as already discussed. Elsewhere, electrocautery uses high frequency alternating current, radiofrequency, or microwave energy to dissect through tissue, rather than sharp blades. A key benefit to electrocautery is that small

blood vessels (diameter less than 1 mm) coagulate due to the heat generated by the probe, helping maintain hemostasis and all but eliminating visual occlusion of the surgical site by blood [157]. Controlling contact with a surface using hand-held tools therefore is a critical surgical skill, as incision depth depends on both contact force and contact time. Yet, precise control over these factors is more difficult in procedures performed minimally invasively, where visual feedback is obscured, and haptic feedback is hampered due to friction, or dampened by the length of the tool interposed between the hand and the tissue.

Ablation of the uterine endometrium falls under this category of minimally invasive electrosurgery. The endometrium is a thin, mucosal membrane between the uterine cavity and the muscular myometrium basal layer. During the menstrual cycle, the endometrium builds up and sloughs off, and is thought to play a part in cancers and gynecological diseases. Ablation of the endometrium has been shown to improve excessive menstrual bleeding, known as menorrhagia, when compared to hysterectomy, the surgical excision of all or part of the uterus [158]. Clinicians are careful to avoid damage to the myometrial muscle layer during endometrial ablation. This procedure is a type of MIS, where access to the uterus is achieved transvaginally, using slender tools and cameras. Due to this approach, visualization may be difficult, and haptic feedback is reduced as is typical of MIS techniques.

4.2.3.1 Force Matching Experiment

In general, humans are able to accurately reproduce target forces, generating relatively small absolute errors when asked to generate forces on the order of 2 – 18 N, particularly when visual feedback about the force is provided [21], [111]. Srinivasan and Chen performed an experiment in which participants used their index finger to apply forces to a glass plate attached to a force sensor [159]. Participants' abilities to generate target forces at 0.5, 1.0, and 1.5 N with

and without visual feedback were characterized by measuring the absolute and relative errors. The average error without visual feedback was between 11 – 15%, and generally increased with increasing target force. With visual feedback, however, the error remained constant over these force levels, at approximately 40 mN (3 – 16%). Wang and colleagues performed a similar experiment wherein participants used a hand-held tool to match target forces between 0.5 and 5.0 N [160]. Without visual feedback, the absolute error when matching 0.5 N was 50 mN, and when matching 5 N was 250 mN, 11% and 6% relative to the target force, respectively. With visual feedback, the absolute error did not exceed 50 mN across all tested forces, and relative errors did not exceed 3%. Noise and errors in force generation are generally attributed to motor unit recruitment during contraction and the stochastic nature of motor neuron firing, especially for larger forces [161], [162].

In addition to the amplitude and variability of the applied force, we can examine its frequency spectrum. Voluntary movements are thought to contribute most to the low frequency bands. Tan and colleagues estimate the bandwidth for controlling force with the fingers is approximately 6 Hz [137]. Rapid movements, for example, those associated with playing the piano, have peak frequencies in the 4 to 8 Hz range [21]. We therefore decided to calculate the power in the applied force over three bands: 1 to 4 Hz; 4 to 7 Hz; and 7 to 10 Hz. The first two bands are taken to assess voluntary control of action and involuntary motion in the form of tremor, respectively [102], [163], [164]. The last band was taken to examine what effect, if any, these higher frequency components may play.

Most of the literature has focused on either haptic or visual feedback alone, rather than the contribution of both sources of information at the same time. As surgical robotics and other advanced technologies are developed for use in healthcare, it is imperative that we are able to

differentiate between haptic and visual feedback, and understand their potential benefits in isolation and in conjunction. Towards this end, we developed an experiment to test how well subjects can generate and sustain small target forces over long periods of time, under all combinations of haptic and visual feedback.

In this experiment, participants took part in a series of trials in which they matched several target forces using the HHFM Model-3: 1, 2, 3, and 4 grams (approximately 10, 20, 30, and 40 mN). A 1-gram window was centered on the target force zone, and participants were instructed to maximize the time spent in this zone. (It should be noted that we use "grams" as the primary unit of force in this section – sometimes denoted "grams-force" – to provide a more familiar unit than Newtons.) These tight constraints were inspired by work done by Jagtap and Riviere, who found that retinal surgeons routinely perform vessel punctures while exerting only 7.5 grams (approximately 75 mN) [20].

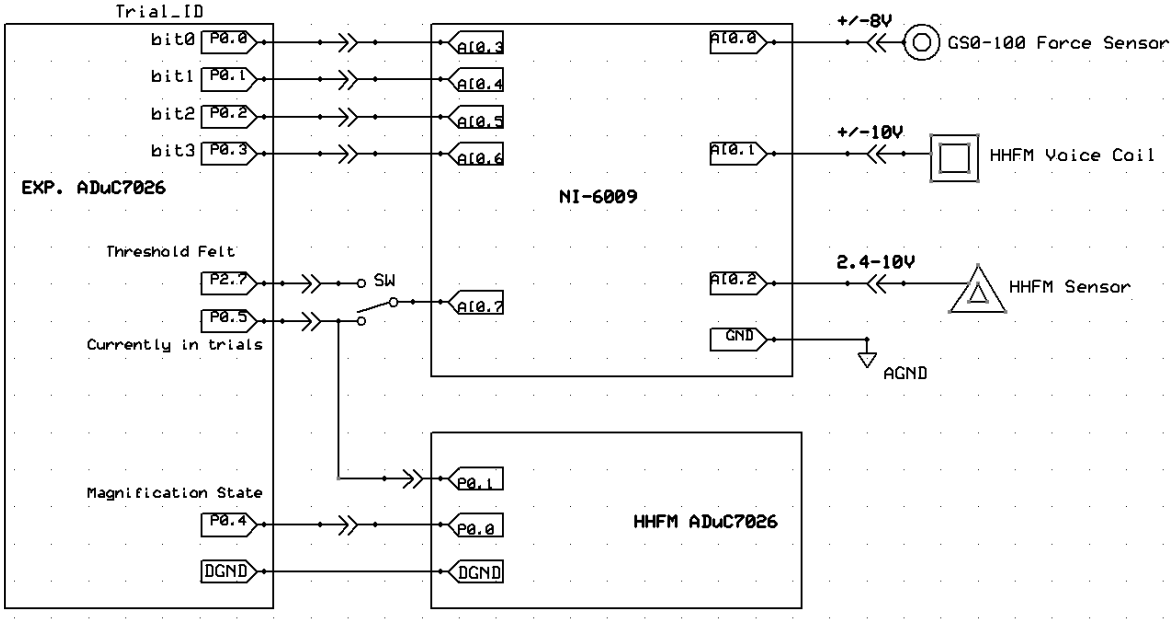


Figure 42. Force Matching Experiment system connection diagram, including HHFM and Experimental ADuC7026 controllers and NI-6009 DAQ.

In each trial, participants were given simple visual feedback about how their applied force compared to the target during an initial 6-second period, followed by another 6-second period in which visual feedback was removed and participants were instructed to maintain the target force as best they could. Throughout each trial, voltage data were collected at 1 kHz using a National Instruments NI-6009 DAQ connected to a Windows 7 machine via USB.

A custom apparatus was built to actuate the experimental conditions, as well as collect the relevant data. **Figure 42** shows a system diagram of the connections between the two controller microprocessors and the DAQ. **Figure 43** shows the two hardware boxes connected together. The HHFM Model-3 is controlled by its own ADuC7026 microprocessor (the "HHFM Microprocessor," or "HHFM ADuC7026" in **Figure 42**) as previously described, where new software functions and hardware associated with this experiment ("Experiment Mode", indicated

Figure 43. HHFM (left) and Experimental (right) microprocessor control boxes. Experiment mode indicator light and block selection buttons labeled.

by the red LED on the left-hand box labeled “Exp” in **Figure 43**) may be accessed by pressing a button on the face of the control box. “Experiment Mode” in the HHFM Microprocessor is also activated upon reset of the second microprocessor (the “Experimental Microprocessor,” or “Exp. ADuC7026” in **Figure 42**), or if an experimental block is initiated.

Most device conditions and experimental variables are manipulated by the Experimental Microprocessor, which is housed within a second control box (right-hand box in **Figure 43**). The Experimental Microprocessor relays the force feedback state, controls all visual and audio feedback presented to the participant, keeps time during each 6-second half of the trial, and actuates all blocks of trials. To control force feedback, a GPIO pin in the Experimental Microprocessor is directly connected to a GPIO pin in the HHFM Microprocessor, acting as a switch to enable or disable the HHFM actuator. The HHFM gain was set to $k = 2$. This gain was chosen as we had seen benefits to the afferent pathway under a gain of $k = 2.4$ in our afferent pathways experiments (Experiments 1 – 3, see Section 3.2.2). Visual feedback was actuated by comparing the GS0-100 input to the current target force, and switching three GPIO pins that are connected to LEDs mounted behind the target, as shown in **Figure 44**. A small tubular target (white arrow) was attached to a nylon screw threaded into the GS0-100, presenting a point at which participants could attach the

Figure 44. Force matching target (white arrow) with visual feedback LEDs mounted behind the GS0-100 force sensor.

distal tip of the HHFM, by means of a pronounced 90° hook on the tool tip that could be inserted into the target, enabling participants to push or pull on the sensor as requested.

Visual feedback was presented to the participants as follows. The red or green LED (**Figure 44**) was lit at the beginning of a trial to indicate the direction of the requested force; the instruction given to participants was to "push or pull the light to the middle." For example, in a push trial, the red LED was initially lit. When the participant reached the target zone (a 1 gram window centered around the target force), the yellow LED was illuminated. If the participant exceeded the target zone, the green LED was activated, indicating that the participant should reduce their force until they re-entered the target window. For pull trials, the order of the red and green LEDs was reversed.

The first 6-second period was initiated when the participant had passed through the target window three times. During this period, force-responsive visual feedback using the LEDs was provided to the participant, using the same signals as earlier in the trial (i.e., the participant received no alert signal that timing had begun). After this 6-second period, the LEDs were turned off, and participants were instructed to maintain the target force as best they could. An audio cue was given after 6 seconds without visual feedback, at which time participants were instructed to release contact until the next trial. **Figure 45** is an example of the sensor and actuator data collected from one trial, with the visual feedback data collection starting at the green vertical dashed line. The 6 seconds without visual feedback lies between the two red vertical dashed lines.

The matching task was repeated for the four target forces in two directions: push and pull, and with HHFM magnification off or on, $k = 0$ or 2, respectively. Each unique combination of parameters (target force, magnification state, and direction) were hardcoded into the

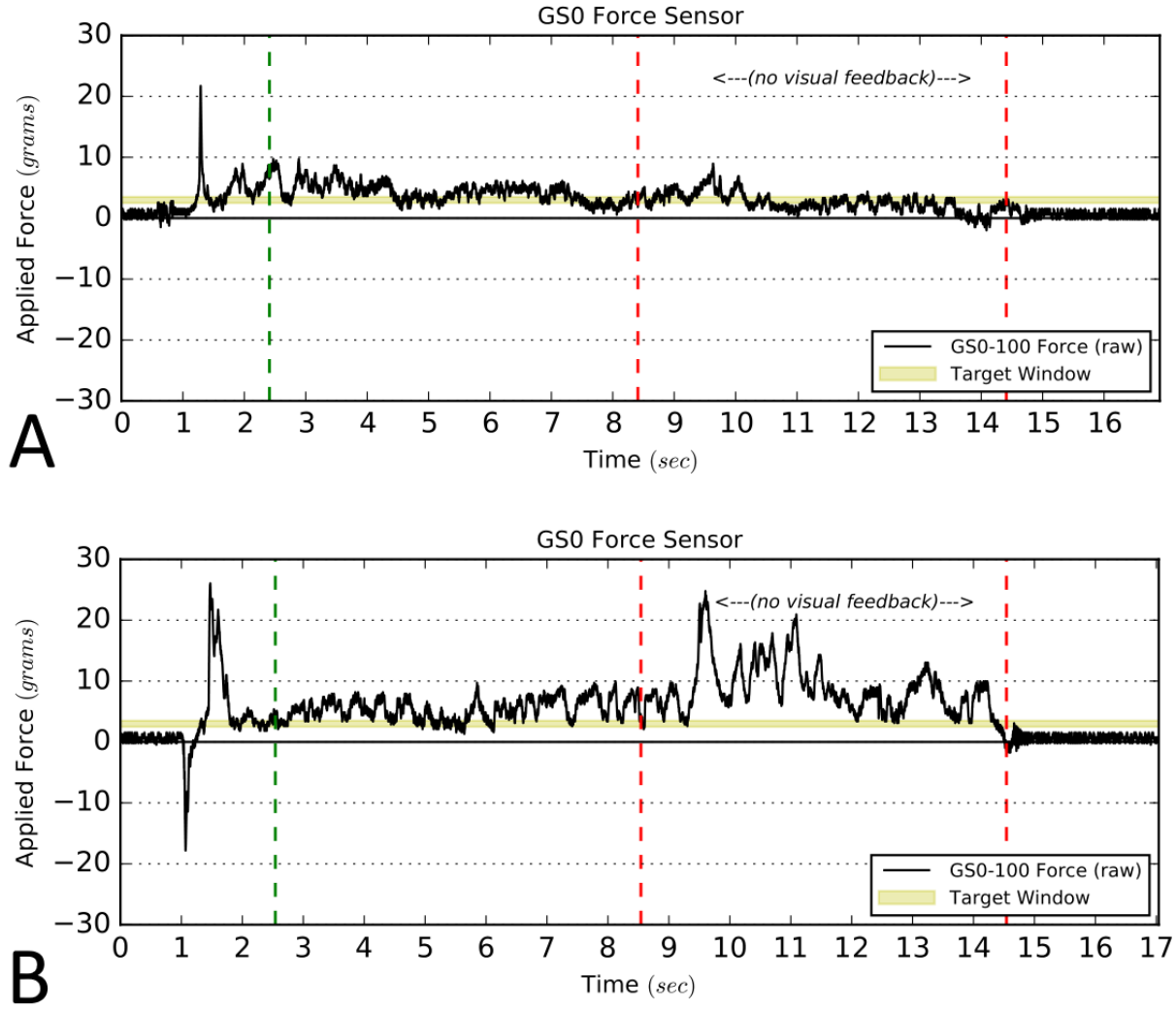


Figure 45. Example recordings from the GS0-100 Force Sensor with a participant matching a 3 gram target force (yellow band) in the pull direction (A) with magnification and (B) without magnification. The green vertical lines indicate the beginning of visual feedback data collection. The red vertical lines indicate the period without visual feedback.

Experimental Microprocessor, and assigned a Trial ID number. The order of the trials was then randomized within a block, such that each permutation was presented only once. Completing the task under each of the 16 experimental conditions, randomly ordered, constituted 1 block. Each participant completed 5 blocks of trials. Prior to these experimental blocks, participants also

completed a training block of trials to accustom themselves with the device and task. The training block presented different target forces than the experimental block. The entire experiment lasted approximately 1 hour.

The NI-6009 DAQ collected voltage data from the GS0-100 force sensor, the HHFM force sensor, and the HHFM voice coil actuator. In addition, the state of 5 GPIO pins from the Experimental Microprocessor were recorded. One GPIO pin was dedicated to showing whether the system was currently actuating a trial (Exp. ADuC7026 P0.5 in **Figure 42**). The remaining four GPIO pins (Exp. ADuC7026 P0.0 – 0.4 in **Figure 42**) were dedicated to a binary representation of the Trial ID number. As there were a total of 16 experimental conditions, 4 bits were required to uniquely identify each parameter combination – each GPIO pin acted as a single bit. From the unique Trial ID number, we could recover the current trial parameters, including target force, magnification state, and direction, for later analysis.

Four participants (three male, one female) were recruited from the local university community for the Force Matching experiment. All participants were right handed by self-report, and were tested individually under a university IRB approved protocol with informed consent. In all, 320 trials were collected from the 4 volunteers.

Analysis and visualizations were implemented in Python 2.7 using numpy, scipy, and matplotlib. Additional analyses were performed in Excel (Microsoft; Redmond, WA).

We examined the mean and standard deviation of the force applied to the GS0-100 sensor with and without both visual and haptic feedback. The force measured at the GS0-100 sensor was called the “distal force,” and was the main source of data we analyzed, as it is the force that would have been applied to tissue. We hypothesized that force magnification would increase the

accuracy of the distal force, as well as reduce its variability. Both measures are negatively affected by tremor, which we hypothesized would also be reduced under force magnification.

To determine the extent to which tremor may be affected by force magnification, we conducted spectral analyses to examine the frequency-dependent components in participants' motor behavior. Spectral analyses were done using the Fourier transform, coefficients of which we could use to examine the power of the force signal in each condition. The power was calculated using

$$P(\nu_1, \nu_2) = \int_{\nu_1}^{\nu_2} |\mathcal{F}\{x(t)\}|^2 \, d\nu \qquad \text{(Eq. 24)}$$

where $x(t)$ is a particular signal in time; $\mathcal{F}\{\cdot\}$ indicates the Fourier transform; and ν_1 and ν_2 are the frequency limits for the band in question. Note that "power" is used here not in its physical sense, but rather in its signal processing sense, to mean the square of the amplitude of the given measurement, in this case, force.

Evaluating the integral in Eqn. 24 produces a measure of the power in a frequency band. The cumulative power could in this way be compared between the various experimental conditions. We chose to examine two bands in particular: 1 – 4 Hz and 4 – 7 Hz, where the first band is taken to represent voluntary movements, and the second band to signify physiological tremor [102], [163], [164].

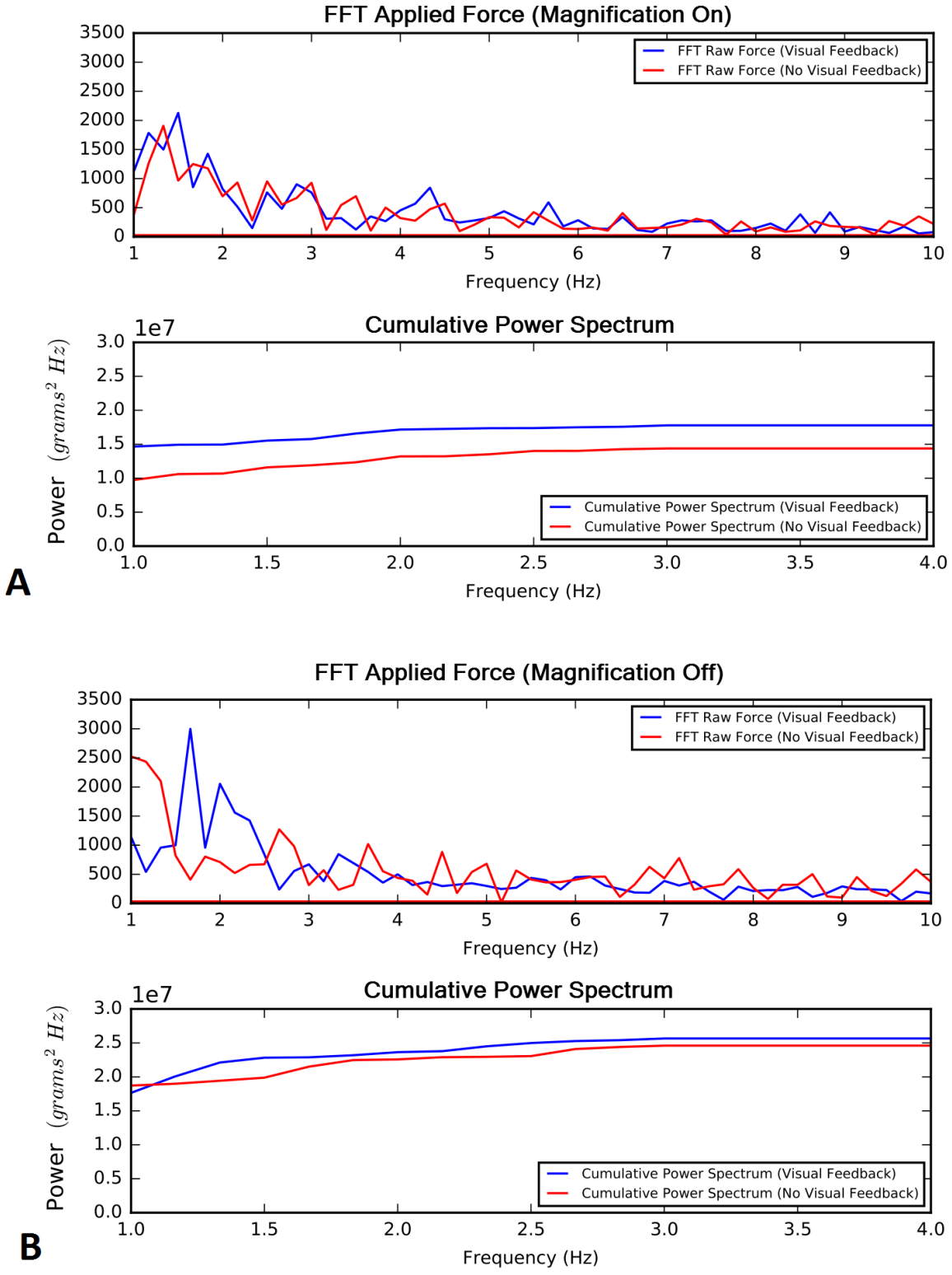


Figure 46. Fourier transform magnitude (top) and power bottom in the 1 – 4 Hz frequency band (bottom) for 3 gram target in the pull direction with (A) magnification on and (B) magnification off.

Figure 46 shows the Fourier transform magnitude and power spectrum for two sample trials. For these two trials, the cumulative power is smaller with force magnification (**Figure 46A**). We generally expect the power in the visual feedback condition (blue) to be greater than without visual feedback, indicating the subject is servo-ing around the target force, responding to the visual feedback.

Figure 47 shows the mean distal force for all subjects by visual feedback condition, and as a function of the target force. Error bars are ±1 SEM. In these graphs, the data are shifted slightly to prevent overlap. Push forces are represented as negative forces, while the pull direction is assigned positive numbers. Error-free performance is represented by the dashed $y = x$ line.

As expected, the distal force matched the target force better under the influence of visual feedback, whereas participants began to drift from the target when they did not have the benefit of visual feedback. The mean absolute error (±1 SEM) was little affected by magnification, however. The average absolute error over all target forces and directions in the visual feedback

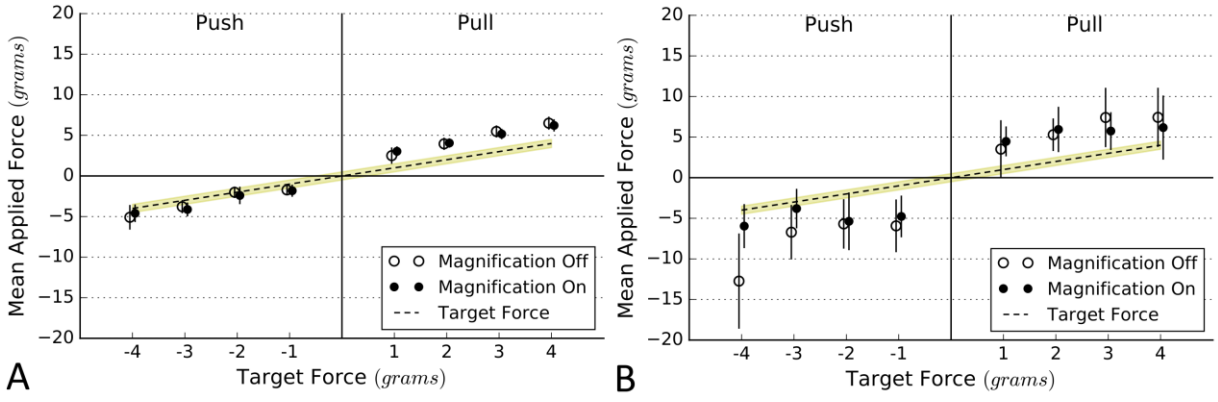


Figure 47. Mean applied force in the Force Matching experiment (A) with visual feedback and (B) without visual feedback, as a function of magnification and target force. The 1 gram window is highlighted in yellow. Error bars represent ±1 SEM.

condition was 1.38 ± 0.29 grams (13.53 ± 2.84 mN) without HHFM magnification, compared to 1.42 ± 0.25 grams (13.92 ± 2.45 mN) with HHFM magnification. The clear trends are that without visual feedback, the mean applied force is more variable, as evidenced by the larger SEM bars, and that the error is higher: the average absolute error over all target forces and directions with the HHFM-off increased to 4.34 ± 0.63 grams (42.56 ± 6.17 mN); with the HHFM-on, the average absolute error was somewhat lower, only 2.78 ± 0.36 grams (27.26 ± 3.53 mN).

Figure 48 compares the power under magnification in the 1 – 4 Hz and 4 – 7 Hz bands, respectively. Especially for the lower frequency band, where we would expect the greatest contribution from voluntary movement, we see at least some improvement in the pull direction compared to push – lower power indicates smoother action (**Figure 48A**). This reduction in power is most striking with visual feedback as the target force progresses from the push direction to pull. However, there is minimal difference between magnification conditions. Without visual feedback, the pull direction seems to exhibit generally lower power, but again the magnification state does not seem to improve performance consistently.

Physiological tremor and other high frequency movements would be represented in the higher frequency bands (**Figure 48B**). Unfortunately, there is no systematic improvement in performance that would be indicative of a reduction in physiological tremor under the influence of force magnification. Note that the power in the 4 – 7 Hz band is an order of magnitude lower than that in the 1 – 4 Hz band, indicating that these higher frequency bands contribute much less to the overall power delivered through the tool.

Statistical and spectral analyses have thus far shown little improvement in task performance under force magnification. To examine this issue further, we calculated subject z-scores for the percentage of time spent within the target zone within each 6-second half of the trial. **Figure 49** shows the cumulative distribution functions (CDF) of these z-scores under the 4 visual and haptic feedback conditions.

Two-sample Kolmogorov-Smirnov (K-S) tests were conducted on CDF pairs to assess the effects of force magnification and visual feedback. The significance level was set at $\alpha = 0.05$. Comparing the visual feedback conditions under the same magnification state showed a significant advantage for feedback, as expected (magnification off: $D = 0.325$; $p = 5.52 \times 10^{-8}$;

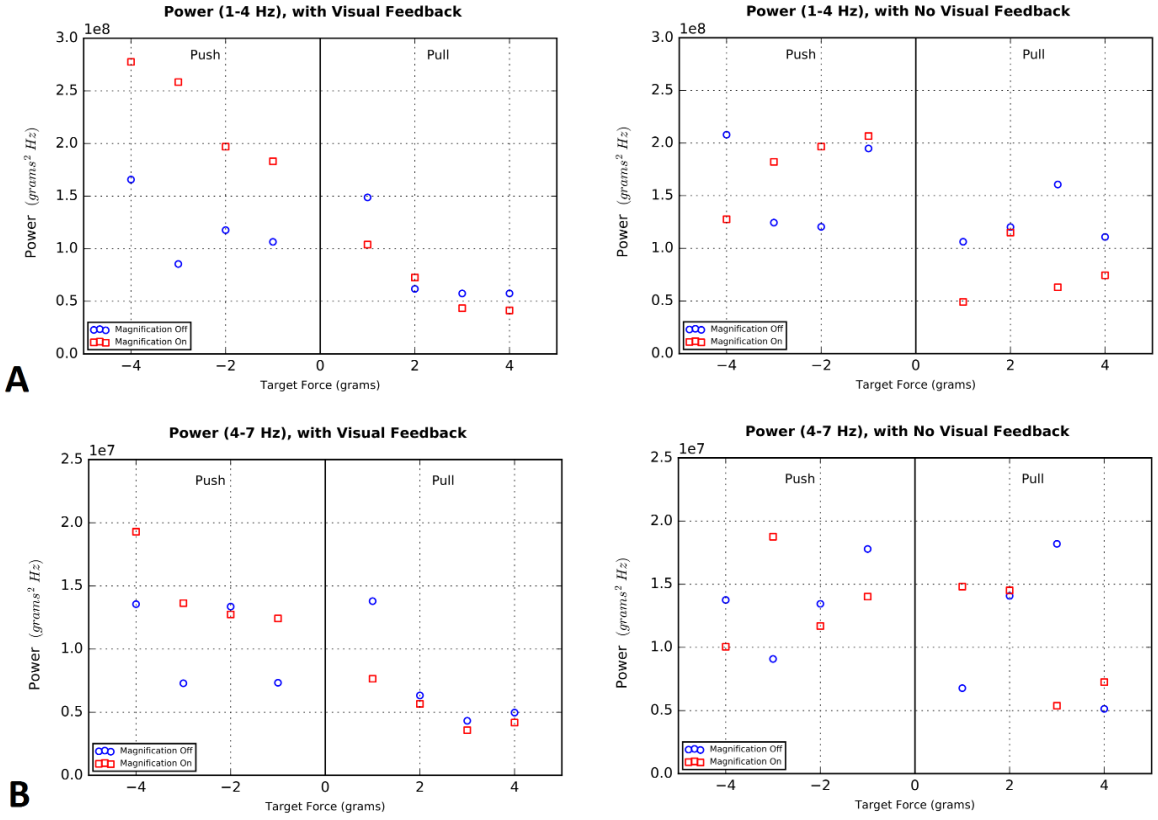


Figure 48. Cumulative power on the (A) 1 – 4 Hz and the (B) 4 – 7 Hz frequency bands, with (left) and without (right) visual feedback, as a function of magnification and target force.

magnification on: $D = 0.475$; $p = 1.43 \times 10^{-16}$). Under visual feedback (solid lines in **Figure 49**), K-S analysis did not find any significant difference between the magnification conditions (D = 0.081; $p = 0.648$). Without visual feedback (dashed lines in **Figure 49**), however, magnification significantly improved the time spent within the target window ($D = 0.156$; $p = 0.036$), as indicated by the rightward shift of the magnification-on CDF.

It should be noted that on an absolute scale, even with visual feedback and the benefit of force magnification, half the trials showed participants spending less than 15% of the time within the window. In the period without visual feedback, participants spent even less time in the target zone, approximately only 8%. These data point to the difficulty of the task, which led us to design the Minimum Contact experiment, discussed in the next section.

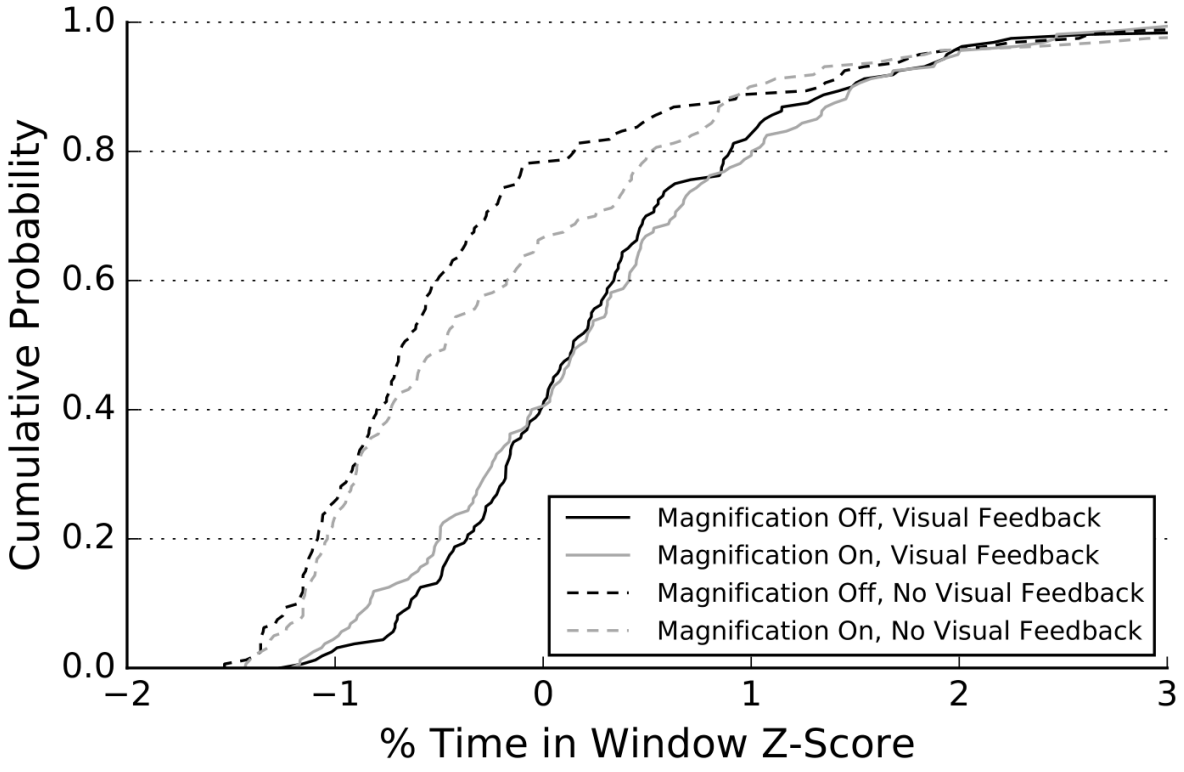


Figure 49. Cumulative distribution functions for the percentage time spent in the target window z-scores, by visual and haptic feedback conditions.

4.2.3.2 Minimum Contact Experiment

Participants in this experiment took part in a series of trials that required them to maintain contact while minimizing force. At the beginning of each trial, participants were guided to a maximum force, 15 grams (147 mN), in one of two directions: push and pull. This force level was chosen as it is supra-threshold in the unmagnified condition, as determined from our absolute threshold experiments (see Experiment 1, Section 3.2.2.1) [74], [130]. Visual feedback leading to the maximum force was given using the LEDs mounted behind the GS0-100 force sensor (see **Figure 44**). The red or green LED was lit initially to indicate the direction of the requested force, push or pull respectively; the instruction given to participants was to "push or pull the light to the opposite side." When participants reached half the target force, the yellow LED was illuminated. Upon reaching the target force, all LEDs were lit, an audio cue was given, and the LEDs were turned off.

The subsequent period without visual feedback, which lasted 6 seconds, was the critical part of the trial captured for analysis. Participants were instructed that, during this period, they should maintain contact with the target while applying as little force as possible.

After the 6-second interval, an audio cue was given to indicate the end of the trial, and participants were instructed to release contact until the next trial began. **Figure 50** is an example of the sensor and actuator data collected from one entire trial; the critical 6-second period lies between the two vertical red dashed lines.

This task was tested in the two directions: push and pull, and under three magnification levels induced by the HHFM device: $k = 0$, 2, and 4. Completing the task under each of these 6 experimental conditions, randomly ordered, constituted 1 block. Each participant completed 5

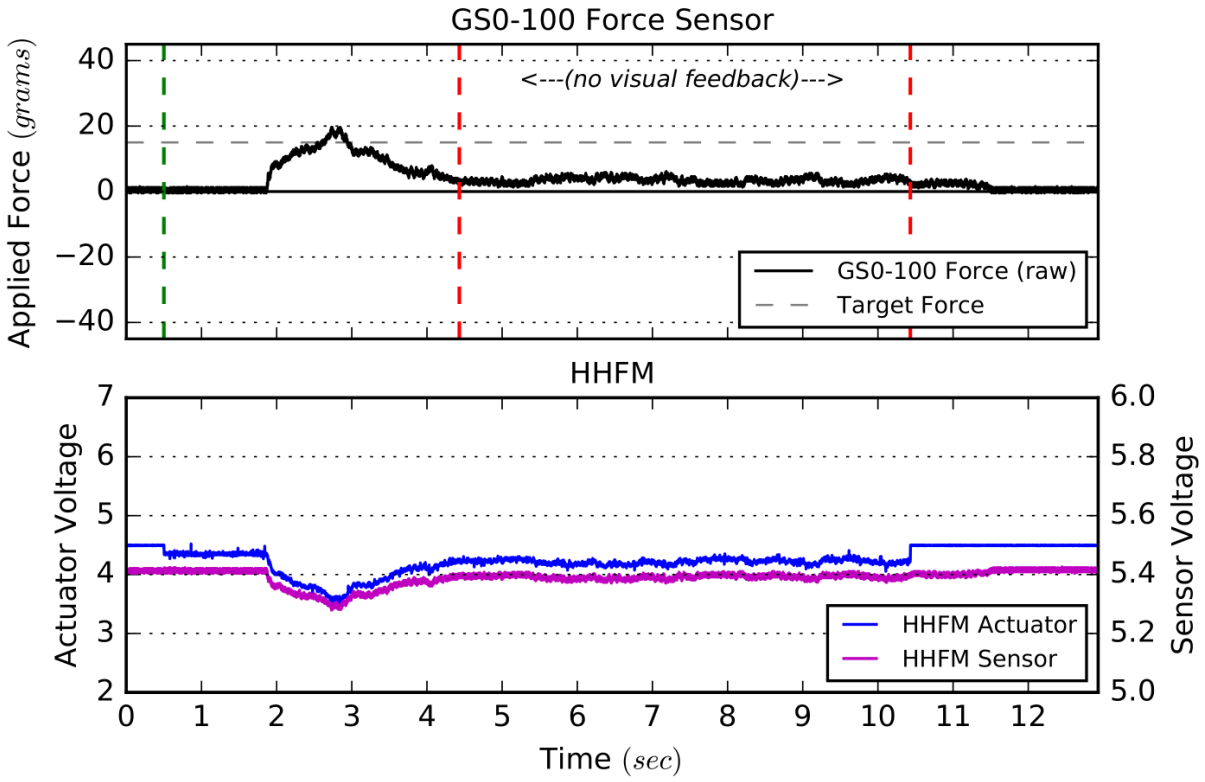


Figure 50. Sample data collected from a Minimum Contact trial in which the participant applied force in the pull direction with force magnification. The applied force (top) and the HHFM sensor and actuator voltages (bottom) are displayed. The dashed vertical line indicates the start of the trial (green), and the beginning and end of the period without visual feedback (red).

blocks of trials. Prior to these experimental blocks, participants also completed a training block of trials to accustom themselves with the device and task.

Modifications to the HHFM and Experimental Microprocessor firmware and circuitry were made to reflect these changes. Schematics for the experimental platform are included in **Appendix C.4**. Since each block only contains 6 trials, the number of bits required to represent the Trial ID was reduced to 3. However, because we tested three magnification levels, an additional bit was added to the magnification state. The updated system diagram is shown in

Figure 51. This version of the HHFM firmware is included in **Appendix C.2**. The source code for the Experimental Microprocessor is also included in **Appendix C.5**.

As in the Force Matching experiment, we analyzed the measurements from the GS0-100 sensor to recover the mean and standard deviation of force while in contact, as well as the power in three frequency bands: 1 – 4 Hz, 4 – 7 Hz, and 7 – 10 Hz. For analysis from each trial, we isolated the last 5 seconds of the period without visual feedback. This ignored the first second immediately following the loss of visual feedback, which tended to demonstrate undesired variability in performance while participants achieved minimum force.

To determine when the participant was in contact with the force sensor, we statistically compared measurements from a period of known non-contact before each trial with measurements during the 5-second test window, during which the participant might or might not be in contact. To differentiate, we applied Welch's t-test, a modification of the Student's t-test

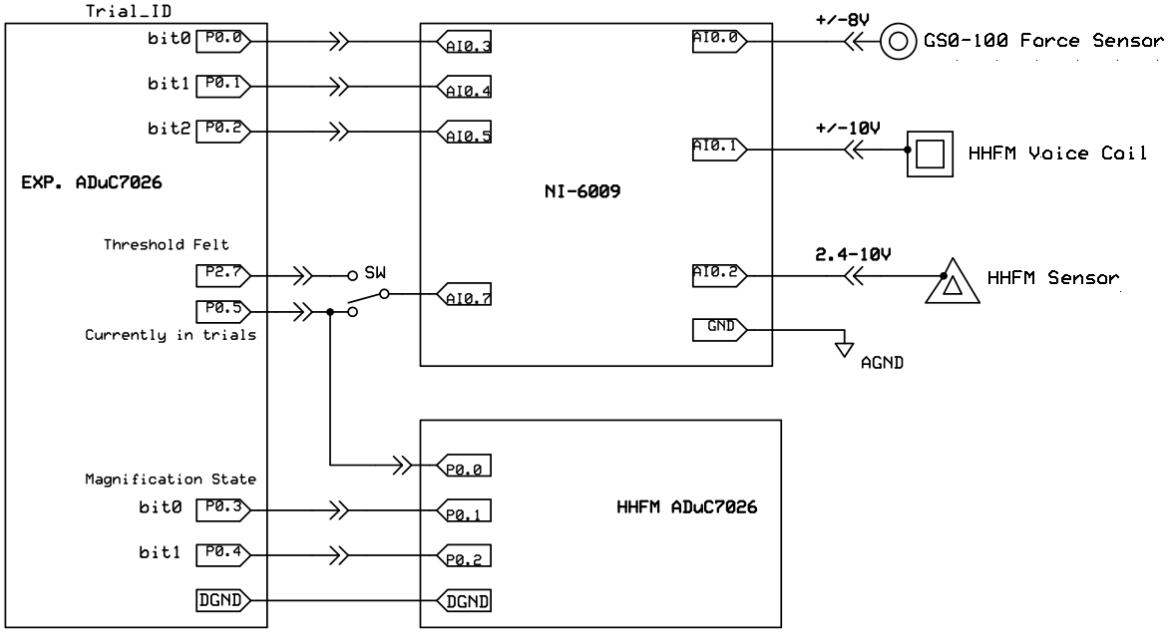


Figure 51. Minimum contact experiment system diagram.

for determining the significance of a difference between two means that accounts for both uneven variances and uneven sample sizes [155]. A key benefit to Welch's t-test is that it reduces the incidence of Type I errors (false positives, or incorrectly rejecting the null hypothesis). The statistic in this test is given by

$$t = \frac{\overline{X_1} - \overline{X_2}}{\sqrt{\frac{s_1^2}{N_1} + \frac{s_2^2}{N_2}}} \qquad \text{(Eq. 25)}$$

where s_1 and s_2 are their respective standard deviations; N_1 and N_2 are their respective sample sizes; and $\overline{X_1}$ and $\overline{X_2}$ are the means of the two samples [155].

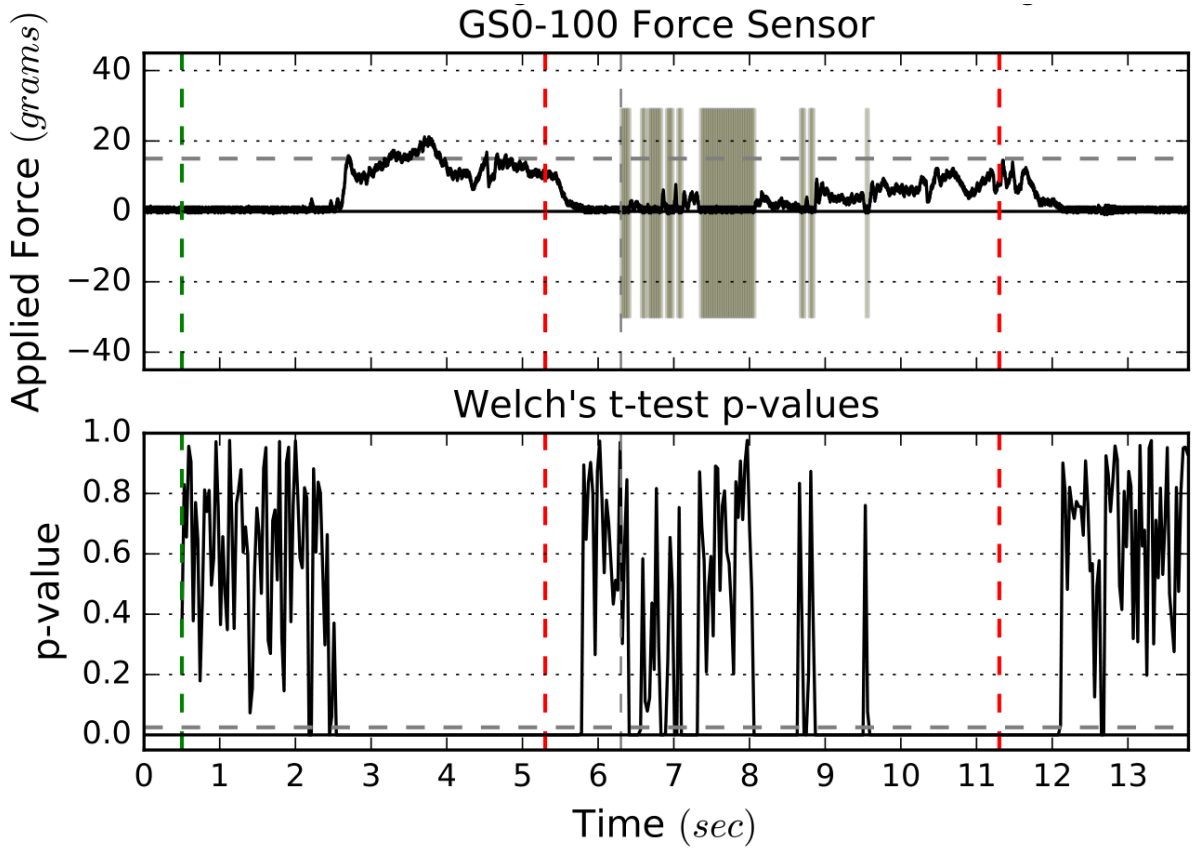


Figure 52. Contact detection using Welch's t-test for a pull trial. Time out of contact is highlighted only for the last 5 seconds of the trial. The red vertical dashed lines bound the no visual feedback section of the trial. The HHFM is considered "in contact" with the sensor when $p > 0.025$ (dashed horizontal line on bottom plot).

As participants were instructed to ensure they were not in contact with the sensor between successive trials, we used the mean and standard deviation from the 250 ms of force measurements just prior to the start of the trial to serve as a baseline reading for the absence of contact. This reference was then compared to the mean and standard deviation of force of 30 ms non-overlapping windows along the rest of the trial. (This window size was chosen because thirty 1 ms samples produced an approximately normal distribution, a key assumption of

Welch's t-test.) The significance value for the contact detector was set at $\alpha = 0.025$. When $p > 0.025$, we accepted the null hypothesis that the two samples were drawn from the same population, and concluded that the participant was not in contact with the sensor during that window. **Figure 52** shows an example of segmentation using this process, with the periods of non-contact shaded only during the 5 seconds of analyzed trial data.

The mean and standard deviation of force were analyzed only for those 30 ms periods (within the critical 5-second window) when the participant was in contact with the target. Power was calculated on the raw force data, so as to capture both the loss and restoration of contact with the target, as applicable. In this way, we could examine power delivered to the target as if it were tissue. These data and contact time were subjected to repeated-measures ANOVA with factors of direction and magnification. The significance value was set at $\alpha = 0.05$. Data

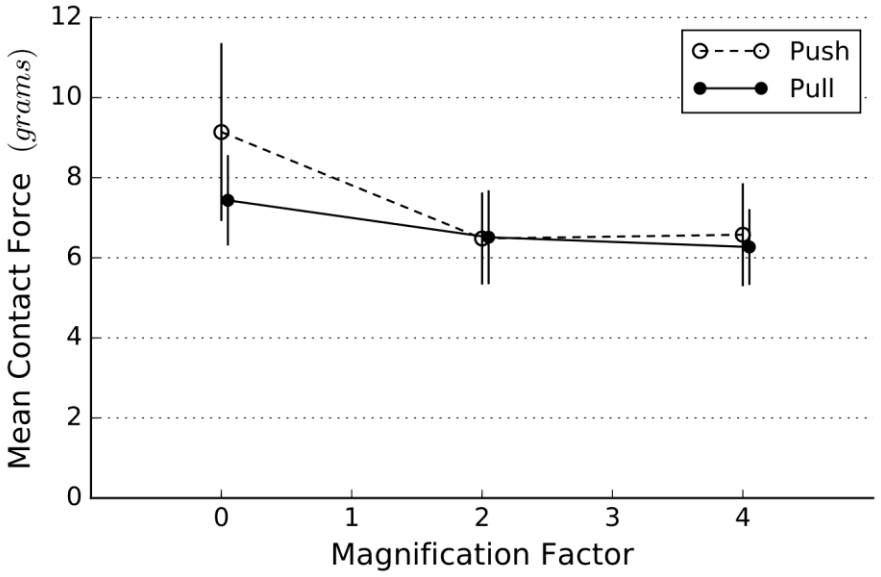


Figure 53. Mean contact force as a function of direction and magnifier gain. Error bars represent ±1 SEM.

conditioning, analysis, and visualizations were done in Python 2.7 using the numpy, scipy, and matplotlib libraries. ANOVA analyses were performed in Excel and SPSS (IBM; Armonk, NY).

Thirteen participants (5 male, 8 female) were recruited from the local university community. All were right-handed by self-report, and were tested under a university IRB approved protocol with informed consent. In all cases the HHFM was worn on the right hand. Participants were tested individually, and the experiment lasted approximately 1 hour.

Figure 53 and **Figure 54**, respectively, show the means and standard deviation of the contact force during the critical 5-second interval without visual feedback, as a function of magnification and direction. In these graphs, the pull data are shifted slightly to prevent overlap, and error bars represent ±1 SEM.

The data indicate that force magnification tends to reduce both the mean and standard deviation of applied force in both push and pull directions. Repeated measures ANOVAs with

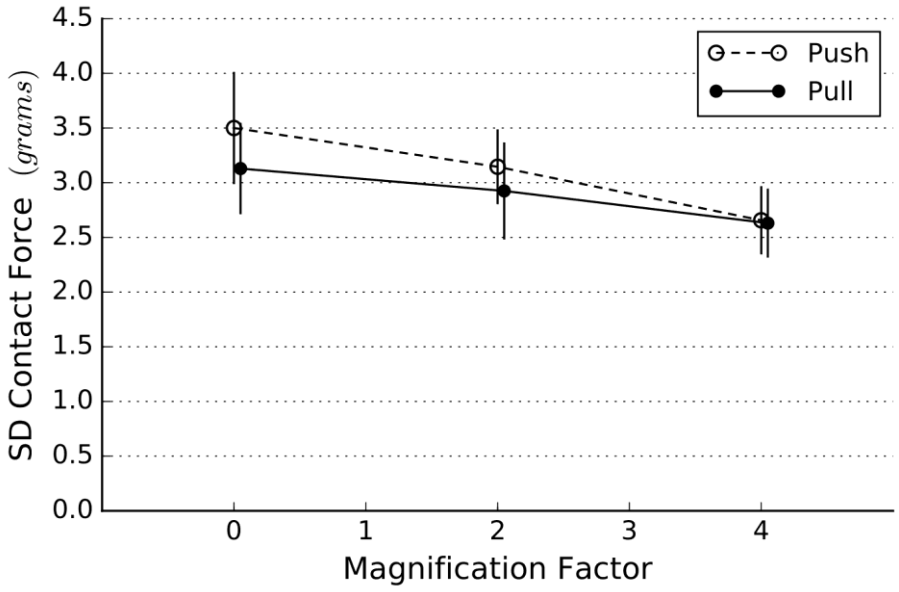


Figure 54. Standard deviation of contact force as a function of direction and magnifier gain. Error bars represent ±1 SEM.

direction and magnification level as factors found a significant main effect of magnification on mean force ($F(2,24) = 6.23$, $p = 0.007$, partial $\eta^2 = 0.342$) and the standard deviation of force ($F(2,24) = 5.56$, $p = 0.01$, partial $\eta^2 = 0.316$). Neither measure showed an effect of direction or an interaction between direction and magnification.

Figures 55A to **55C** show the power as a function of magnification and direction in the frequency bands of interest: 1 to 4 Hz; 4 to 7 Hz; and 7 to 10 Hz, respectively. **Figure 56** compares the average power over all conditions by frequency band. The higher two bands respectively contain only 20% and 15% of the power in the first band.

Although the mean power trends suggested that magnification tended to reduce power, particularly in the push direction, statistical analysis of the individual power bands showed no significant effects in the first band (1 to 4 Hz), indicative of voluntary force production, and the third band (7 to 10 Hz), where overall power was low. There was a significant interaction between magnification and direction in the second power band, 4 to 7 Hz ($F(2,24) = 3.60$, $p = 0.043$, partial $\eta^2 = 0.231$), taken to primarily represent tremor. This reflects a reduction in power with magnification for the push direction only. Between-subject variability also tended to be lower for this band.

Contact time as determined by the Welch's t-test detector is shown as a function of magnification in each direction in **Figure 57**. On the whole, participants were very good at maintaining contact with the target, averaging over 80% of the period in contact. ANOVA showed a significant main effect of magnification ($F(2,24) = 4.19$, $p = 0.027$, partial $\eta^2 = 0.259$). Notably, changes in contact time were nonmonotonic, showing reductions at $k = 2$, whereas the unmagnified and $k = 4$ magnification showed approximately equal time in contact.

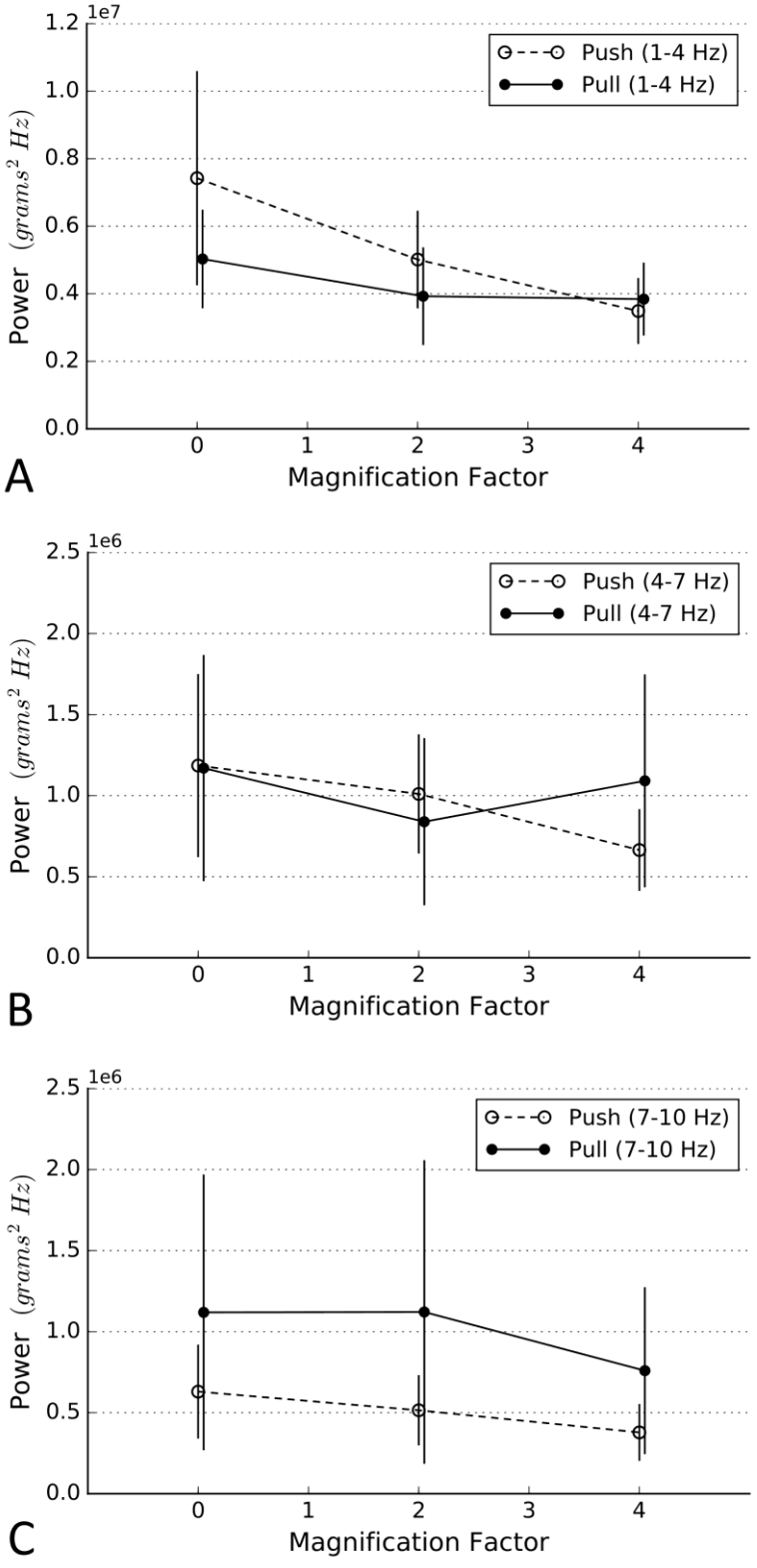


Figure 55. Power in the (A) 1 to 4 Hz; (B) 4 to 7 Hz; and (C) 7 to 10 Hz bands. Error bars represent ±1 SEM.

Pairwise Student's t-tests found significant differences between the $k = 2$ and $k = 4$ factors ($t(12) = 2.60$, $p = 0.023$), favoring the larger magnification. However, the difference was so small as to have little practical relevance (mean time in contact with the sensor was 79.7% at $k = 2$ and 84.6% at $k = 4$ magnification). Given that the unmagnified condition (84.2% time in contact) was intermediate between the magnified values, we are reluctant to reach a conclusion about the effects of magnification on near-threshold time in contact.

As systemic effects of magnification were consistently observed across the variables other than contact time, a principal component analysis (PCA) was conducted on the *slope* of the magnification function for the mean, standard deviation, and power in each band, within each direction. Nine of the 10 variables showed loadings greater than 0.5 on a first component, which accounted for 64.2% of the variance.

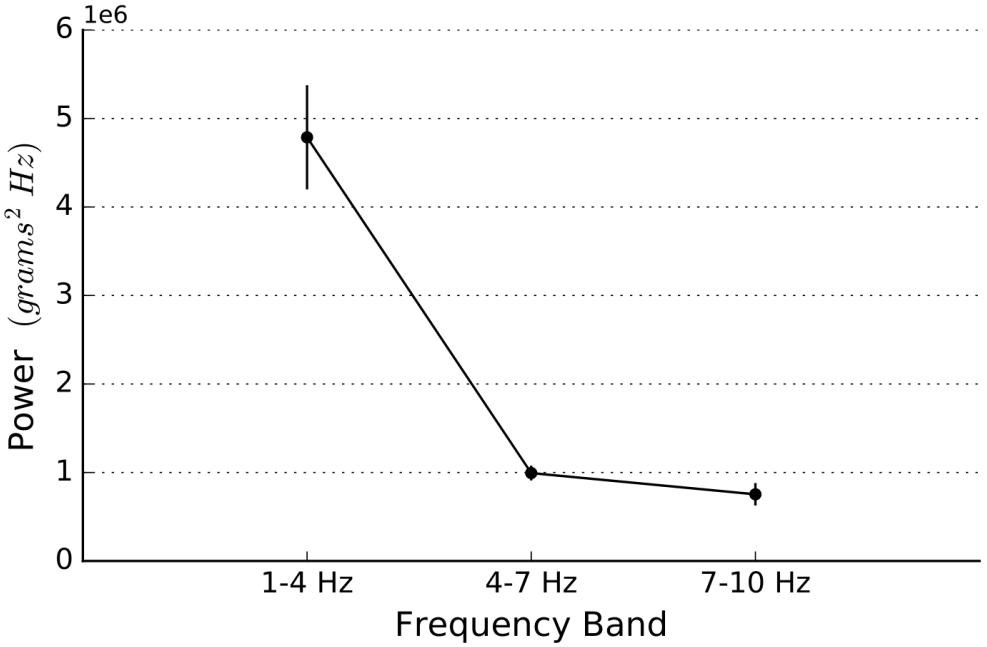


Figure 56. Power over all conditions, by frequency band. Error bars represent ±1 SEM.

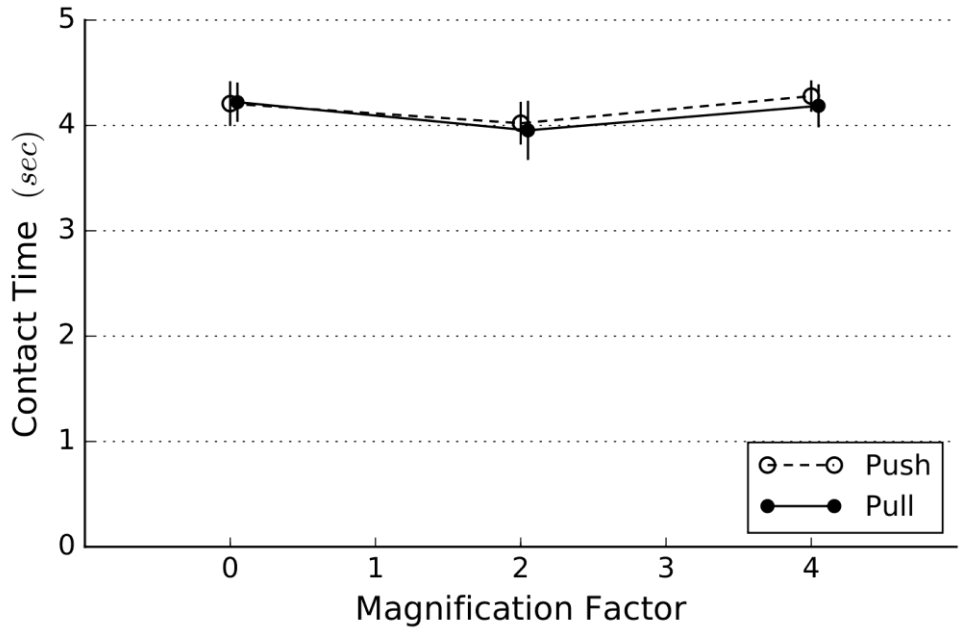


Figure 57. Time in contact, by magnification and direction.

The exception was the standard deviation of force in the pull direction, which loaded greater than 0.5 on a second factor. This second factor accounted for an additional 18.3% of variability, and also showed greater than 0.5 loadings for the power in the first two bands, exclusively for the pull direction. Note that there is inherently a direct relationship between total power and standard deviation in the signal. The overall pattern for this second factor, then, suggests it is related to an effect of magnification on the variability of the pulling motion. Movements of the wrist in the pull direction are known to involve more dexterity than pushing motions [138], [146], [165], [166].

The PCA suggests that magnification has a broad effect on controlling and stabilizing force, and an additional, more focused effect on stabilizing control of voluntary pulling motions.

4.3 DISCUSSION

Overall, the HHFM Model-3 performed well as a bidirectional force magnifier. However, two major restrictions limited its usability: first, drift and variability in the sensing components over long periods of time (weeks and months) following construction; and second, limited force output from the small voice coil actuator.

4.3.1 Failure Modes Analysis and Mitigation

We experienced two major failures over the lifetime of the HHFM Model-3. The first was a fracture of the distal spider. The second was a break in the force sensor itself due to external loading. These events happened when the Model-3 prototype was at least 1 year old, likely from age-related weakening in the 3D printed plastic. The fracture in the spider was situated at the edge where the arm meets the HHFM handle, as shown in **Figure 58**. Before this failure was identified, the senor was inadequately protected against overload, resulting in the puncture of the membrane covering the sensitive area of the sensor.

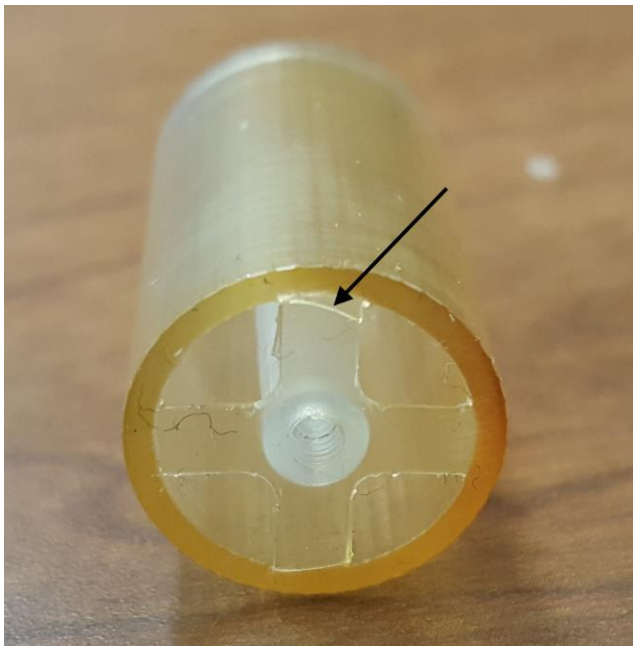


Figure 58. Fracture in HHFM Model-3 distal spider (arrow).

A new HHFM handle was printed, with minor changes intended to mitigate

this potential failure mode. Our initial Model-3 prototypes featured a distal spider only 0.25 mm thick; a thin spider was thought to best transmit force to the Motorola sensor with minimal loss. In the final Model-3 prototype used in Experiment 6, the distal spider thickness had been increased to 0.50 mm as cited in Section 4.1.1. We also added filets (radius 0.76 mm) to the intersection between the spider and the circular handle. The filets distribute stress concentrations over a wider volume of material, reducing the chance of reaching the fracture strength under load. In a future, mass-produced device, this filet feature may be preserved using injection molding methods.

With the sensor replaced and the sensing subsystem reassembled, we saw significant changes in the silicone connector between the mechanical linkage and the force sensor, requiring re-calibration. **Figure 59** shows the change in the calibration factor as calculated between just 6 months following replacement of the internal sensor. This is evidently due to hardening in the silicone material between the linkage and the sensor. In future prototypes, a novel connection material will be required to address this issue.

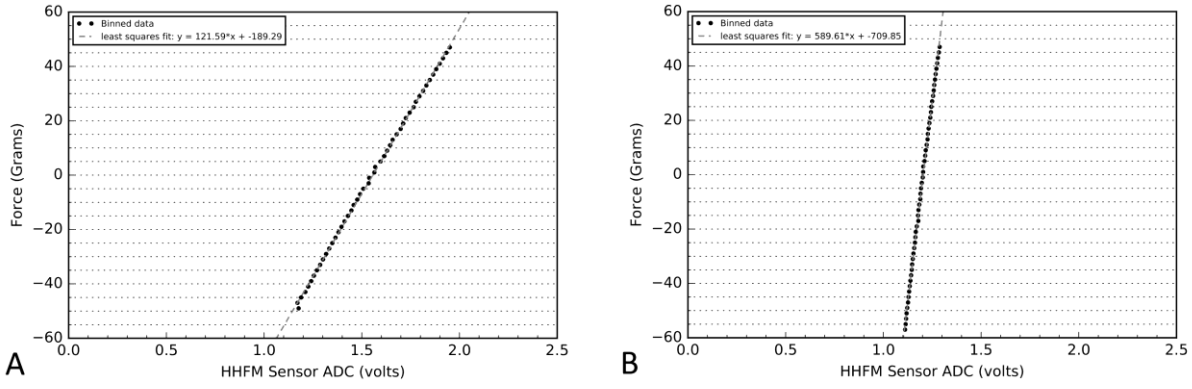


Figure 59. HHFM Model-3 sensing subsystem calibration (A) immediately following repair and (B) 6 months later.

That we have seen only two major failures over the course of demonstrating the prototype multiple times to the general public is a testament to its robustness over the course of a year's use. In an actual surgical situation, the best way to mitigate these failures is to simply limit the amount of time a tool may be used before it is serviced or replaced, or to make the tool a one-time use disposable item. Repeated sterilization can potentially reduce the lifetime of the plastic parts, whether they are 3D printed or injection molded. Ethylene oxide is the most likely sterilization technique to pursue for the HHFM, as opposed to methods like autoclaving, which relies on heat and steam to sterilize objects.

4.3.2 Insights from Psychophysics

As previously mentioned, the voice coil was chosen for its small size and mass. In calibrating the Model-3, the actuator could generate only up to approximately 60 grams force (approximately 600 mN) in either direction. This relatively small range of force obviously restricts the maximum gain that may be implemented into the HHFM. However, even small amounts of magnification, for example, $k = 2$ or so, are sufficient to produce improvements to motor control performance. In the case of isometric force generation, because the hand is gripping the tool in a stiff fashion, feedback forces are very well perceived by the user. Perhaps this is because both cutaneous and kinesthetic components of our afferent system are stimulated.

At the same time, due to this stiff grip on the tool, vibrations associated with electronic noise are readily perceptible. This suggests additional work in filtering the sensor input or modifying the actuator output to selectively eliminate this high frequency noise. Both options have drawbacks, as many haptic cues are contained in these high frequency bands, for example, when making contact with a hard surface, or evaluating surface textures.

Our psychophysical work with the Model-3 focused on assessing the motor control consequences of *in-situ* force magnification. Kinematic data from virtual membrane punctures have yielded descriptive parameterizations of post-puncture motor behavior. We found that pulling motions are more dexterous than push, which agrees with previous analyses. Our model of motor control behavior was sufficiently sensitive to detect handedness.

Matching very small target forces did not show any benefits to force magnification when comparing the means and standard deviations of the applied force. Examining the time spent within the 1-gram target window, we found the force magnification significantly improves homing the applied force to the target. The Minimum Contact task showed broad benefits to force magnification, reducing and stabilizing the force applied to the target. We also found significant magnification effects in the 4 to 7 Hz band, potentially indicating reduced physiological tremor. This effect was notably strong in the push direction. We did not find any significant effects in the 1 to 4 Hz band, which would have been indicative of smoother voluntary control. Further, participants do not trade off improvements in force control (i.e. in the mean or standard deviation of the applied force) with a reduction in contact time. If that were the case, we would have seen an inverse relationship between contact time and magnification, which is certainly not the case from **Figure 57**.

PCA on the Minimum Contact data suggested benefits of magnification on the stability in the pull direction that were common across the overall standard deviation of force and the first two power bands. In contrast, analysis of the individual power bands showed significant benefits of magnification only in the push direction for the second frequency band, associated with tremor. While preliminary, these data suggest that magnification may broadly stabilize the more dexterous pulling forces while reducing tremor in the push direction. These results are

particularly relevant to the use of powered dissection or ablation tools, in which push forces are more common.

It is worth noting that the isometric force generation experiments showed significant benefits of magnification for naïve participants who were handling the HHFM for the first time. This supports our proposal that the forces induced by the device are transparent to the user, being attributed to the distal interaction. Moreover, the cognitive load associated with using the tool appears to be relatively low. Comparison data for both tasks as performed by medical students, surgical trainees, and attending surgeons would be of interest.

Future work comparing magnification effects on motor control in more than one degree of freedom is planned. After all, few surgical tasks are exclusively performed along a single dimension. Dissection, for example, in actuality includes both the axial and vertical directions. Whether magnification improves motor control differentially along these multiple axes is an open and important question.

5.0 THE MODEL-4 HAND-HELD FORCE MAGNIFIER

Encouraged by our work with single DoF HHFM prototypes, we wanted to tackle the design of a multiple DoF tool. As with the HHFM Model-1, our goal was to construct a proof-of-concept prototype, to evaluate the feasibility of a tool with multiple DoF *in-situ* force feedback. Amongst our gravest concerns was whether the tool would be too massive to be useful, and whether each DoF would be individually sensible and controllable. Noise that we had seen in the previous HHFM prototypes would lead us to worry that the combined vibration from two actuators, held in a stiff arrangement, might interfere unduly with tactile perception.

It was decided to build a 2 DoF device, sensing and amplifying axial and vertical forces. These are the forces encountered during traditional dissection with the surgical blade, as well as those associated with electrocautery and other ablative procedures. One reason we chose these axes is that very delicate dissections are predominately done holding the blade in a pencil grip, as depicted in **Figure 60** [167]. These axes were also chosen for their implementation simplicity. Amplifying along the

Figure 60. Scalpel pencil grip. [167]

horizontal axis to enhance side-to-side forces would have required motors to generate torque. Motors are heavy, and the external circuitry required to implement these components is more complex than that required to integrate solenoids and voice coils. The side to side motion required of horizontal force magnification is also a potential safety hazard. If the tool were not well restrained, a motorized, sharp tool spinning around in one's hand seemed to offer particularly high risk, requiring additional sensing and safety mechanisms, and increasing the overall complexity of the design effort.

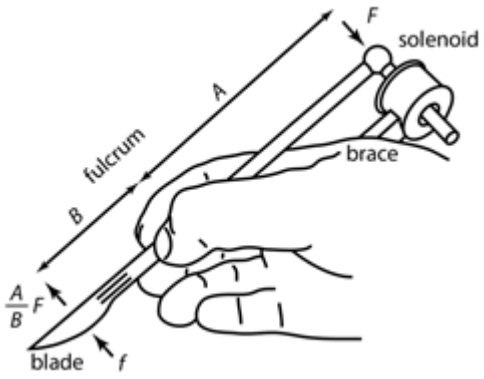


Figure 61. HHFM Model-4 lever concept.

The vertical axis, in contrast, is safer and less complicated to amplify. Because we intend for the tool to only be used in dissection-like situations, we only need to account for distal upward forces from the tissue, which may be amplified by downward forces on the proximal end of the tool, as in the lever design shown in **Figure 61**. Such feedback inherently protects the tissue, as in the HHFM Models-2 and 3, pulling the tool away from the tissue to amplify the force perceived at the tip.

The arrangement as shown in **Figure 61** takes advantage of the relatively long moment arm between the grip point and the actuator, so that even small forces generated by the proximal actuator create strong torques at the point of grip. In this arrangement, the HHFM Model-4 is considered a Class I lever, with the grip point serving as the fulcrum between the distal load *f* and proximal effort *F*. The lever is governed by the moment equilibrium equation,

$$B * f = A * F \quad \text{(Eq. 26)}$$

where A and B, respectively, are the moment arms from the fulcrum to the proximal and distal forces. In traditional tools, the afferent torque is simply $B * f$, where both the moment arm B and the distal vertical force f are relatively small. Indeed, with surgical scalpels, the blade is sharp precisely to reduce the distal vertical force f required to slice through tissue planes.

With *in-situ* force magnification, users will feel the sum of the torque due to the distal tissue force, as well as the torque generated by the proximal actuator force, so

$$\tau_a = \tau_d + \tau_p = (B * f) + (A * F) \quad \text{(Eq. 27)}$$

where τ_a is the afferent torque; τ_d is the distal torque, generated by the force of the tissue on the tool; and τ_p is the proximal torque, generated by the proximal actuator. Through this action, the distal tip force should be reduced by a factor of $(A/B)\,F$.

Here, we should note that we are assuming that torque is perceived similarly to axial force under *in-situ* magnification, as depicted in our control framework (see **Figure 13**, Section 3.1). If we define the magnification scheme simply as

$$\tau_p = k_\tau * \tau_d \quad \text{(Eq. 28)}$$

where k_τ is the torque magnification factor, we can again recover the $k_\tau + 1$ perceptual magnification factor as predicted by Eqn. 13 for axial force magnification.

5.1 HHFM MODEL-4 DESIGN AND DEVELOPMENT

We have thus far taken advantage of commercial force sensors to measure the distal force in the HHFM Models 1-3. Single-axis force sensors with a small footprint, such as those used in the HHFM Model-3 are easily available, but are more difficult to find when measuring multiple DoF

in a single sensor. We therefore set out to design a structure whose deformation could be used to measure the required forces – axial and vertical – in a small footprint and with high sensitivity. In our previous prototypes, we have positioned actuators on the proximal end of the tool. This arrangement works well for axial force feedback, but is more complicated in the other directions. The following sections will detail the design of the individual sensing and actuation subsystems for our first multi-DoF HHFM prototype.

5.1.1 HHFM Model-4 Sensing Subsystem

Our goal was to design the sensing structure in such a way that a single pair of sensors would respond independently to force along the two desired axes. We chose to base our sensor on strain gages, which may be arranged to be precise, robust, and stable. As few as two gages can produce a thermally stable sensor network, for example, using the Wheatstone half-bridge as detailed in Section 2.1.1.2. Strain gages were obtained from Omega (KFH-3-120-C1-11L3M3R; Norwalk, CT), which are small (4.5 mm × 3.9 mm), with a nominal resistance of 120 Ω, and gage factor of 2.

We took advantage of the complex designs that are achievable through SLA manufacturing techniques, for example, modeling the standard connector for a No. 10 scalpel blade. Design progressed using Solidworks, and digital prototypes were evaluated using ANSYS (Canonsburg, PA) finite element modeling (FEM) software. Finite element models were used to examine the strains that would develop in a given structure, or portion of the structure, under external load. Unless otherwise indicated, FEM simulations were computed assuming 2 N loads in both the axial push and upward vertical directions. While these loads are likely higher than those encountered during delicate surgery (2 N is approximately 200 grams-force), they were

useful in demonstrating the stress and strain distribution under load so as to best position the strain gages. Since we plan to use strain gages, we were most interested in the strain calculated from the FEM; from the strain, we can estimate the change in resistance in our gages.

The material assumed in the FEM simulations was the Somos Watershed XC 11122 SLA material mentioned in Section 4.1. This material has a density of 1.12 g/cm^3, Young's modulus of 2765 MPa, and Poisson's ratio of 0.38. FEMs were computed assuming isotropic elasticity, though this assumption may not strictly apply given the layer-by-layer SLA manufacturing process.

5.1.1.1 Prototype 1: Solid Column Sensor

The first sensor prototype that was designed was a solid rectangular column, depicted in **Figure 62**. In this design, the proximal end of the structure (labeled 'A' and highlighted in blue in **Figure 62A**) is rigidly secured into the handle, and the distal portions are free to move. Strain gages would be secured to the flat surfaces just distal to the anchor point, on the top and bottom faces of the column feature. (The top face is indicated by the arrow in **Figure 62A**; the left face is also visible.) In this arrangement, the gages would both exhibit compression or tension under an axial load, and would differentially measure compression and tension under influence of an upward vertical force. The entire sensor was 39 mm in length, and was estimated to have a mass of 1.3 grams.

As can be seen in **Figure 62B**, very little strain develops in the regions of interest. A high stress concentration was identified at the point where the scalpel connector expands into the larger sensor structure, risking potential fracture. These issues are addressed in the next iteration.

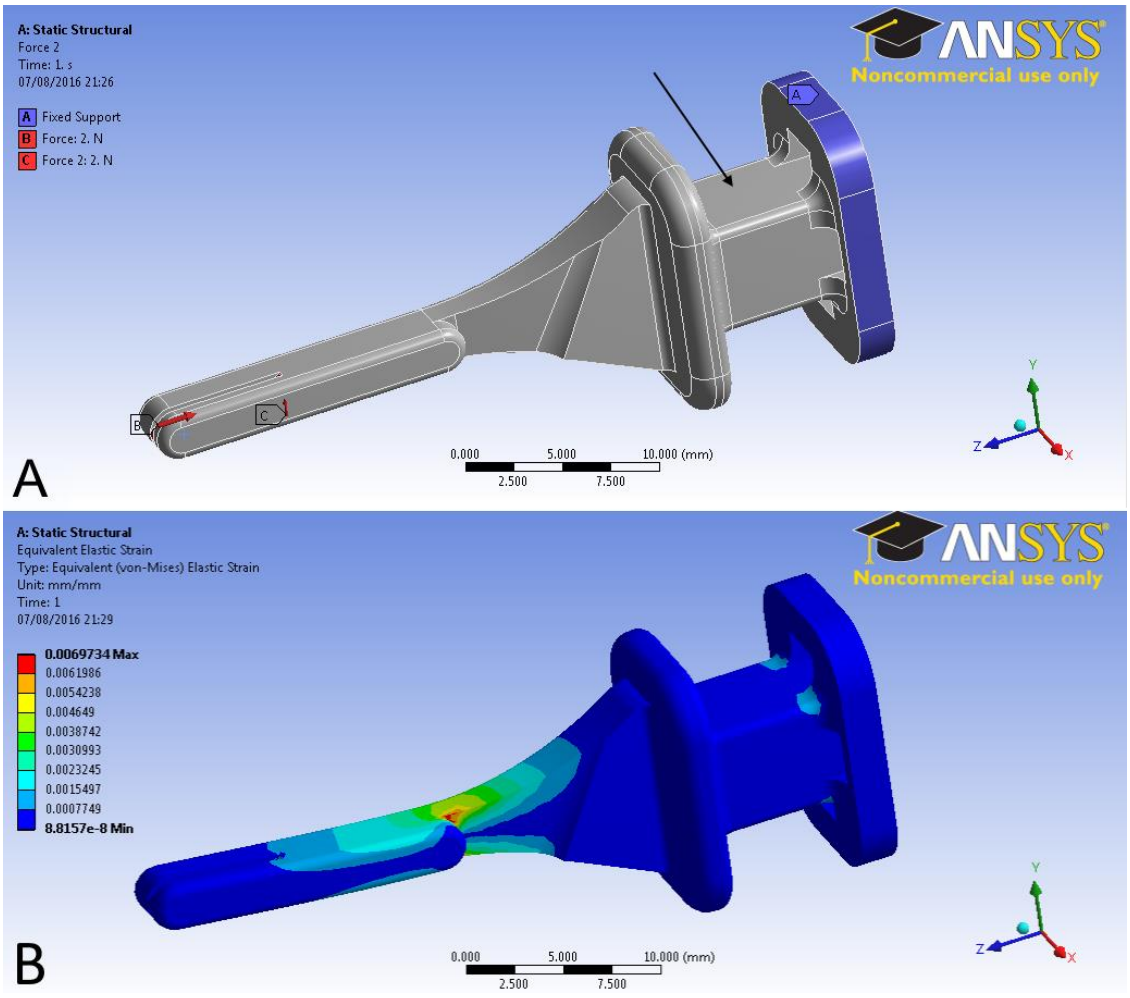


Figure 62. HHFM Model-4 Solid Column sensor (A) CAD and (B) FEM showing von Mises strain. The arrow indicates the top face of the column feature, where one of two strain gages will be adhered.

5.1.1.2 Prototype 2: Hollow Column Sensor

To address the low sensitivity with the Solid Column sensor, we added material at the point where the scalpel blade will be mounted, and hollowed out the column with a cylindrical cut. The CAD and associated FEM are shown in **Figure 63**. This design has similar dimensions to the Solid Column sensor (39 mm length), and was estimated to also have a mass of 1.3 grams.

The addition of material stiffens the distal portion of the structure, as depicted in **Figure 63B**, reducing the likelihood of fracture. We can see that the top and bottom faces of the column

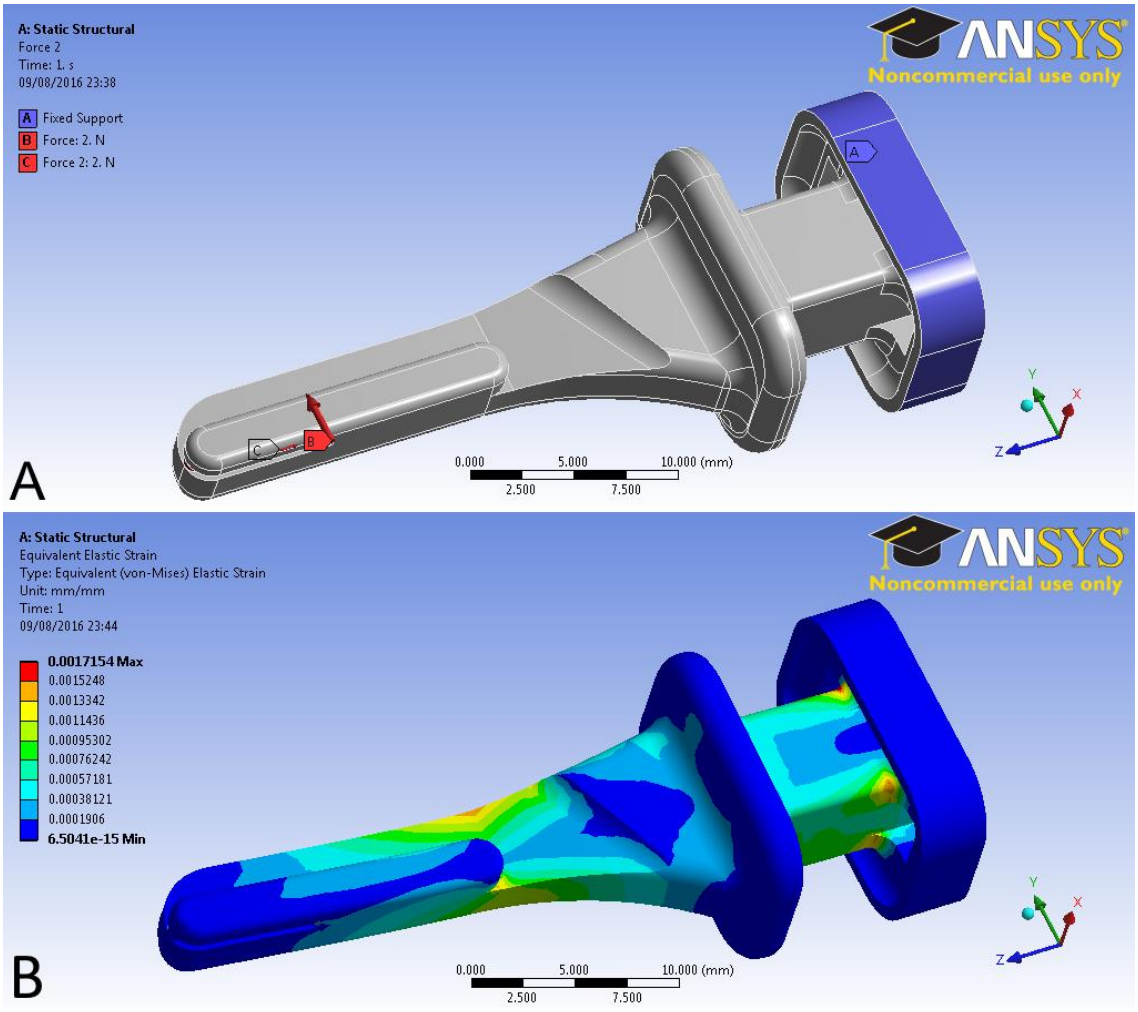


Figure 63. HHFM Model-4 Hollow Column sensor (A) CAD and (B) FEM showing von Mises strain.

now experience a moderate level of strain, due to the fact that the column is now hollowed. (Bottom and left faces are visible in **Figure 63B**.)

Using Eqn. 5 to estimate the change in resistance in the strain gage, we would see

$$\frac{dR}{R} = S_\varepsilon \varepsilon = 2 * 0.00076242$$

$$dR = 240 * 0.00076242 = 0.183\ \Omega$$

given the known gage factor $S_\varepsilon = 2$ and nominal resistance $R = 120\ \Omega$ for the Omega strain gages.

Such a small change in resistance is difficult to reliably measure. Comparing these first two designs, it seems as though the column feature – even with a hollowed out center – is still too stiff to adequately measure force. We therefore sought a design in which the proximal structure supporting the strain gages would exhibit greater strain for given axial or vertical forces at the distal tip, and which would differentiate between those two directions of force.

5.1.1.3 Prototype 3: "Z" Sensor

To better understand the use characteristics of the Omega strain gages, we conducted experiments on test pieces of SLA material. The test piece, depicted in **Figure 64** with a 2-wire Omega gage, was shaped like an I-beam. Experiments with the test piece showed that compressive forces were

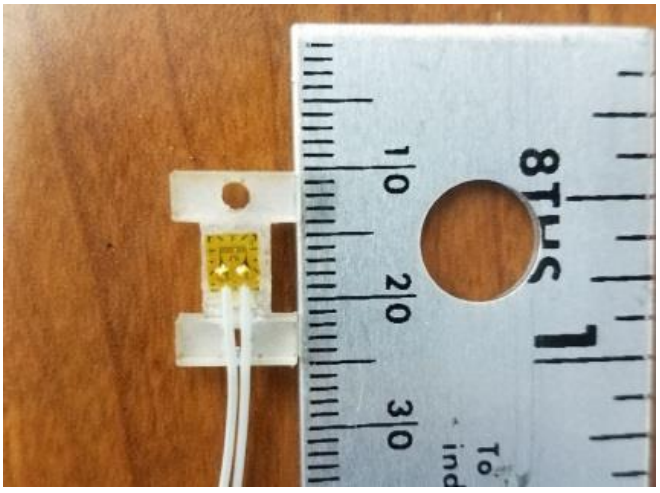


Figure 64. HHFM Model-4 strain gage test piece.

extremely difficult to measure. The gages were instead much more sensitive to bending forces. (Admittedly, our experiment validated a phenomenon well known in strain gage design.) To exploit these differences, we designed the "Z" sensor, as represented in CAD and FEM in **Figure 65**. The sensor is 40 mm long, and is estimated to have a mass of 1.9 grams.

In this design, strain gages would be adhered to the horizontal members of each "Z" formation (arrow in **Figure 65A**) – the features with two right angles just distal to the anchor point. The axial support columns are asymmetric to encourage bending in the horizontal beam.

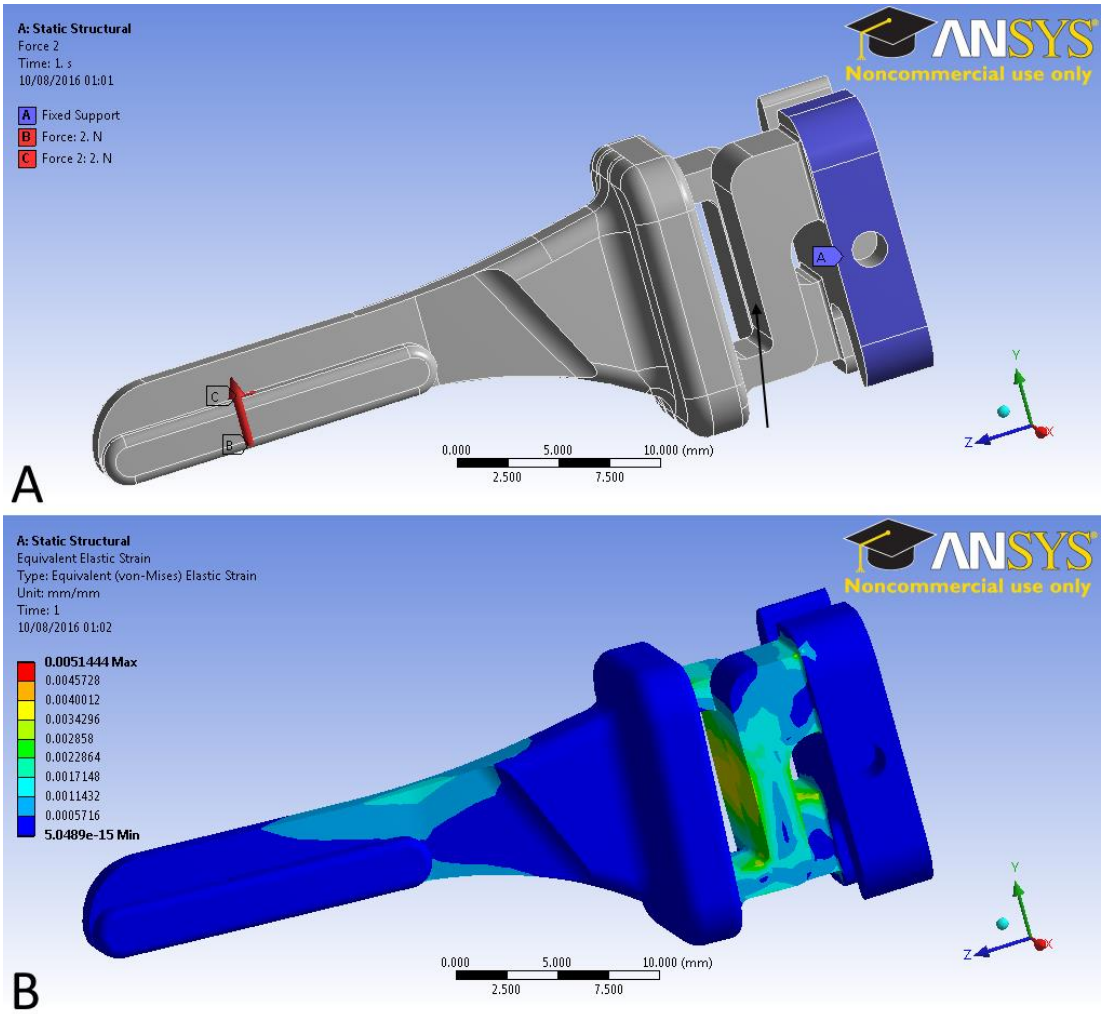


Figure 65. HHFM Model-4 Z sensor (A) CAD and (B) FEM showing von Mises strain. The arrow shows the horizontal members where gages will be adhered.

Because the "Z" features are mirrored, the two gages will change in parallel to indicate axial force, while differential action indicates vertical force. Examining the calculated strain, we found the maximum magnitude was similar to the Solid Column sensor, but the strain distribution is much closer to what we desired. Applying Eqn. 5 to estimate the expected change in resistance in our strain gage, we would see

$$\frac{dR}{R} = S_\varepsilon \varepsilon = 2 * 0.002858$$

$$dR = 240 * 0.002858 = 0.686\ \Omega$$

This anticipated change in resistance is much more reasonable to measure, especially given that we are using two sensors in a half-bridge arrangement, doubling the resultant signal.

The "Z" sensing structure was manufactured using SLA. Strain gages were then adhered to the horizontal member of each Z-structure after the underlying surfaces were roughed using a file to promote better bonding. Consumer adhesives proved to be too weak to secure the gages to the SLA material, so we used cyanoacrylate instant adhesive (3M CA30; St. Paul, MN) with good success.

Preliminary tests with this sensor showed that vertical force sensing was at least an order of magnitude more sensitive than the axial direction. FEM agreed with this difference in sensitivity, as depicted in **Figure 66**. Due to the geometry of the Z-structures, vertical forces induce bending in the proximal region (**Figure 66B**), as intended. The bending effect was much reduced, however, under axial load (**Figure 66A**). This difference in sensitivity in part led to focusing initial development of the HHFM Model-4 prototype on magnifying solely along the vertical direction.

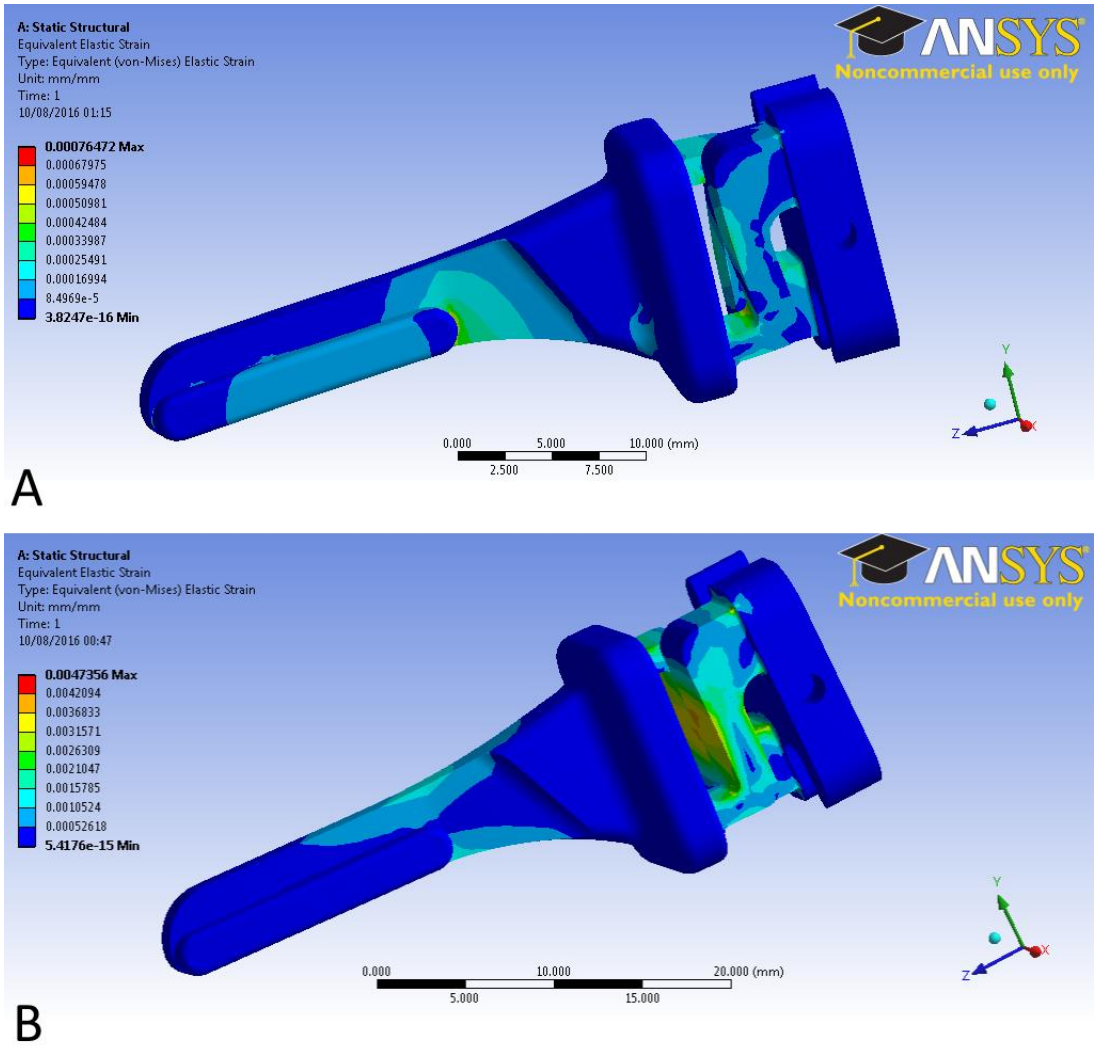


Figure 66. HHFM Model-4 Z sensor under isolated (A) axial and (B) vertical loads.

5.1.2 HHFM Model-4 Actuation Subsystem

A CAD for the HHFM Model-4, with suitable connectors between the sensing and actuation subsystems, is shown in **Figure 67**. The proximal end of the "Z" sensor is inserted into a handle that serves as a standardized grip point. We can ensure that a roughly constant perceptual gain across users is produced in the vertical direction by restricting the user to gripping this area, as will be described in the next section. A guard is also included on the distal end of the device to prevent fluids or other material from entering the inside of the tool. The grip point and preamp enclosure are hollow, allowing wiring from the strain gages to pass through into the hollow post, where they may exit the axial housing as was done in the HHFM Model-3.

We were happy with the axial force feedback mechanism in use in the HHFM Model-3, and decided to keep it with only minor changes. Axial force feedback is actuated by the same

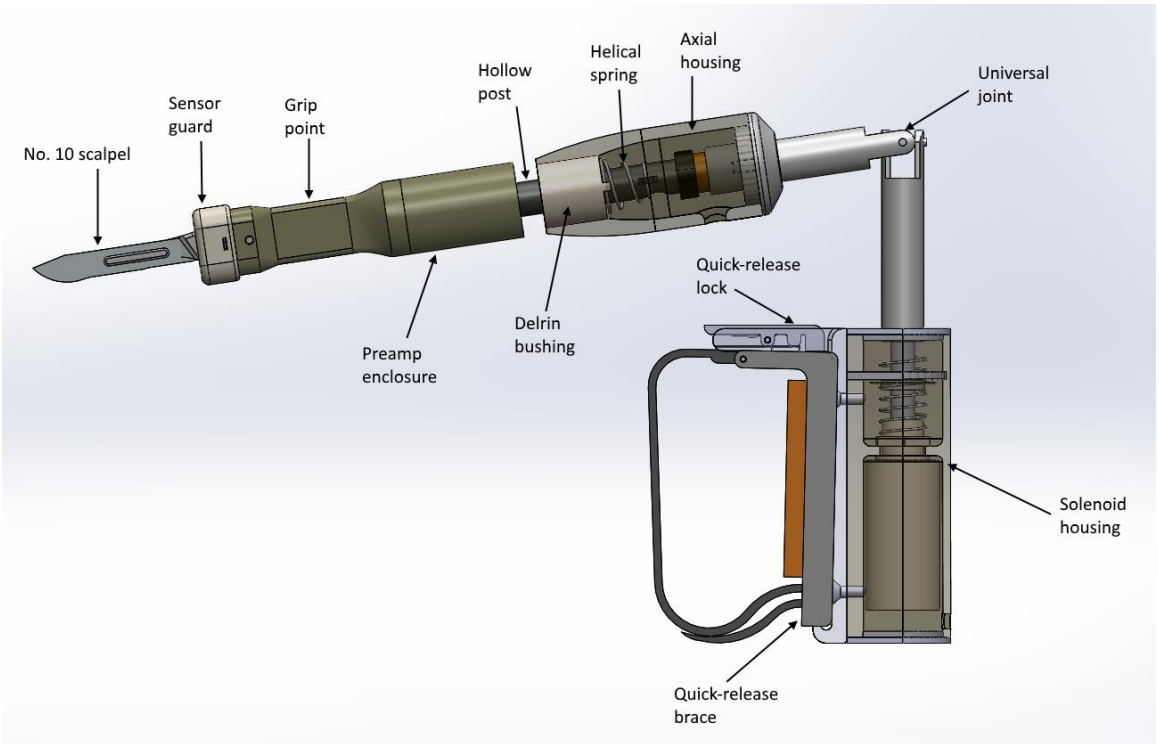


Figure 67. HHFM Model-4 CAD.

Moticont LVCM-013-013-02 voice coil that was used in the HHFM Model-3. A concentric Delrin bushing and conical spring were specified to provide the restoring force for the voice coil, relative to the actuator enclosure. The force generated by a conical spring is non-linear, with a smaller change in force per unit distance at the beginning of the displacement due to the initial compression of the larger diameter coils. Also, its shorter unstressed length (compared to traditional coil springs) makes the entire axial assembly smaller.

Force along the vertical axis is generated by a solenoid mounted to the brace on the back of the palm. Because we are only interested in the upward force, we can take advantage of commercial solenoids, which typically only provide force in one direction, and offer greater force output than voice coils. A stronger actuator is required, since our scalpel design involves moving a larger mass, compared to the small voice coil used in the HHFM Model-3. A pull

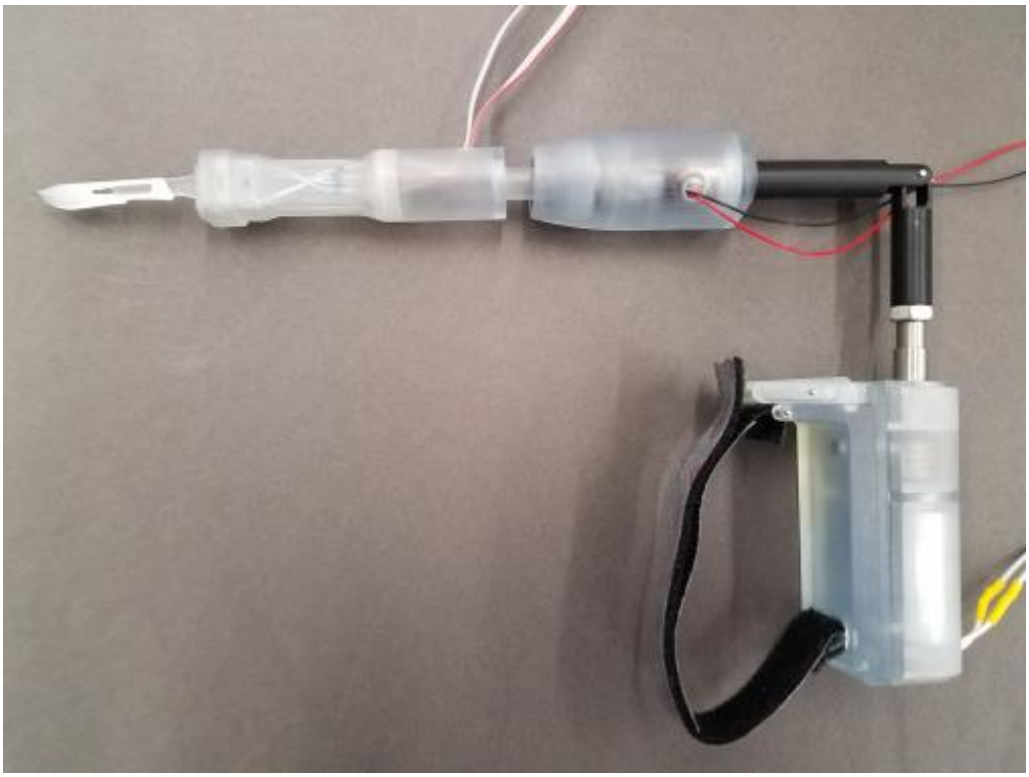


Figure 68. HHFM Model-4, original 2 DoF feedback design.

solenoid (Tai-Shing SMT-1632L; Taipei, Taiwan) was chosen for its small size (Ø16 mm) and mass (47 grams), and relatively high force output, 500 grams (4.9 N). A weak conical spring was included with this solenoid, to restore the plunger to the resting position when no current is present across the solenoid. The solenoid is encased in a housing, which also includes a novel quick-release mechanism. The user wears the brace (dark grey in **Figure 67**) and inserts this piece into the hook at the bottom of the solenoid housing. At the top, a latch holds the mated pieces together, where a small torsion spring is used to provide the restoring force to hold the latch in place. The latch may be reached by the thumb, enabling the user to take the HHFM off and on by detaching themselves from the solenoid housing.

A universal joint connects the solenoid plunger to the proximal end of the axial housing, enabling movement in the vertical direction. Horizontal rotation is free because the solenoid plunger is free to rotate within the coil. The legs of the universal joint were machined from Delrin, and rotate around a metal ball bearing. The resulting joint is smooth and stiff, with relatively little mass.

The remaining parts were manufactured using SLA, and the system was assembled as shown in **Figure 68**. A stainless steel threaded adaptor was added to connect the solenoid plunger to the vertical rod of the ball bearing part. In total, the HHFM Model-4 as depicted in **Figure 68** has a mass of 214 grams.

Preliminary use of the HHFM Model-4 as originally designed showed that the combined mass of the axial actuators and stainless steel adaptor tended to overcome the weak spring that was included with the pull solenoid. This loading on the solenoid reduced the useful dynamic range of the actuator. This discovery, combined with the inferior axial sensitivity of our 2-DoF sensor described above, convinced us that we should focus on controlling vertical force feedback using the "Z" sensor and solenoid as a first proof of concept. The axial voice coil motor and associated housing was therefore removed and a simple Delrin connector (white, **Figure 69**) was machined. Because the axial force actuator was removed, a hole was added to the preamp enclosure so sensor wires could exit to external circuitry. The Model-4 prototype with these modifications is shown in **Figure 69**. This prototype has a mass of 182 grams, similar to that of the HHFM Model-1.

Figure 69. HHFM Model-4, modified to actuate only vertical force.

5.1.3 HHFM Model-4 Control

The HHFM Model-4 was controlled by simple analog circuitry, similar to the HHFM Model-1. **Appendix D** contains the associated schematics. Briefly, the signal from the strain gages is amplified using a differential amplifier, and directly powers the solenoid. Gain is adjustable through a potentiometer in the circuit.

To create a known and perceptually constant magnification factor requires knowledge of the position of the fulcrum in relationship to the distal and proximal ends, in addition to calibrating the actuator force function. Given that A (see **Figure 61**) is some multiple of B, (i.e. $A = \lambda B$), the actuator force required to magnify the distal torque at a given magnification factor is

$$\tau_p = k_\tau * \tau_d$$

$$A * F = k_\tau * (B * f)$$

$$F = \frac{k_\tau * (B * f)}{A} = \frac{k_\tau * (B * f)}{\lambda B} = \left(\frac{k_\tau}{\lambda}\right) * f \qquad \text{(Eq. 29)}$$

where k_τ is the desired torque gain; λ is the ratio between these two distances, A/B; F is the force produced by the proximal actuator; and f is the distal force of the tissue on the tool. We can use Eqn. 29 in a future controller to produce a known torque magnification gain.

5.2 DISCUSSION

Difficulties in producing a scalpel with 2 DoF *in-situ* force magnification arose because of the arrangement of the feedback actuators and the corresponding weight of the device. Commercial components are generally too large and heavy for our application, potentially pointing development of future prototypes to depend on novel and custom sensors and actuators. Our attempts thus far with the Z force sensor, have resulted in greater sensitivity to vertical load that axial forces, but this problem should be addressable.

Interestingly, the percept when using the Model-4 is much different than the Model-3. With the axial force feedback prototypes, structures that were soft in reality were perceived to be stiffer, or harder to deform. In initial tests with the Model-4, however, force feedback made surfaces feel *easier* to deform. Magnification still reduced the distal force, but we believe the perception is different because the grip point does not act as a perfect, motionless fulcrum. We had designed the tool to rotate at the grip point, but did not anticipate the way the user typically applies vertical force, namely, by *translating* the handle vertically, rather than applying a torque. This translation in combination with the pulling of the solenoid corresponds to the perception that the surface is easier to deform, rather than harder.

6.0 THE ONE DIMENSIONAL HAPTIC RENDERER

6.1 MOTIVATION

Over the course of our psychophysical experimentation, we have made extensive use of the MLHD as a haptics rendering platform. With respect to our membrane puncture experiments specifically, described in Chapter 4, we recognize that the mass of the MLHD flotor (500 grams) may be too large to accurately simulate the microsurgical environment, since the structures and tissues of interest have masses much less than 500 grams. However, the force bandwidth offered by the MLHD are highly desirable for high fidelity simulation and modeling. We therefore sought to develop a novel haptic renderer that uses a moving part with much smaller mass, but which can generate large forces at high operating frequencies like the MLHD.

6.2 DESIGN AND DEVELOPMENT

As described in Chapter 2, existing commercial haptic renderers are generally driven by motors in grounded arrangements. This approach to rendering suffers from parasitic losses due to friction or mechanical backlash, usually at the various linkages that are a part of the device. Considerable attention has been paid in commercially available equipment to reduce these

effects, but devices like the Geomagic Touch, one of the most popular renderers, are still limited in their force output.

The MLHD is unique in this respect, as it can generate much higher forces with its set of 6 internal electromagnets. The MLHD can generate up to 40 N, compared to the Geomagic Touch's maximum force of 3.3 N. Obviously, an additional advantage with the MLHD is that it enables movement and generates feedback forces in all 6 DoF, albeit over a smaller reachable volume. One further drawback is that the MLHD is much larger, requiring mounting into a desk. To reduce the complexity involved with designing such a system, we decided to focus on building a haptic renderer which acts only in one axis. Inspired by the MLHD, which uses electromagnetic Lorenz forces as the basis for force feedback, we chose the commercial audio loudspeaker as the basis for our haptic renderer [168].

6.2.1 Haptic Renderer Hardware

Our haptic renderer – the "1DoF" – uses a 5 inch (127 mm) diameter, 80 W woofer speaker (Faital PRO 5FE120; Milan, Italy) as its actuator. Because speakers are generally used in audio applications in which its cone moves at frequencies between 20 Hz and 20 kHz, its mass is minimized. The cone of the 5FE120 speaker has a mass of only 11 grams. In initial tests, the speaker was able to lift a 10 N mass when a DC current through its coil was controlled by a hardware knob.

One advantage to this choice of actuator is that audio loudspeakers already have a mechanical spider, an elastic element between the drum and the speaker housing, to keep the coil concentric to the magnet and provide a restoring force when the drum moves past its rest position. While the speaker does not extend far past its rest position in normal operation, our

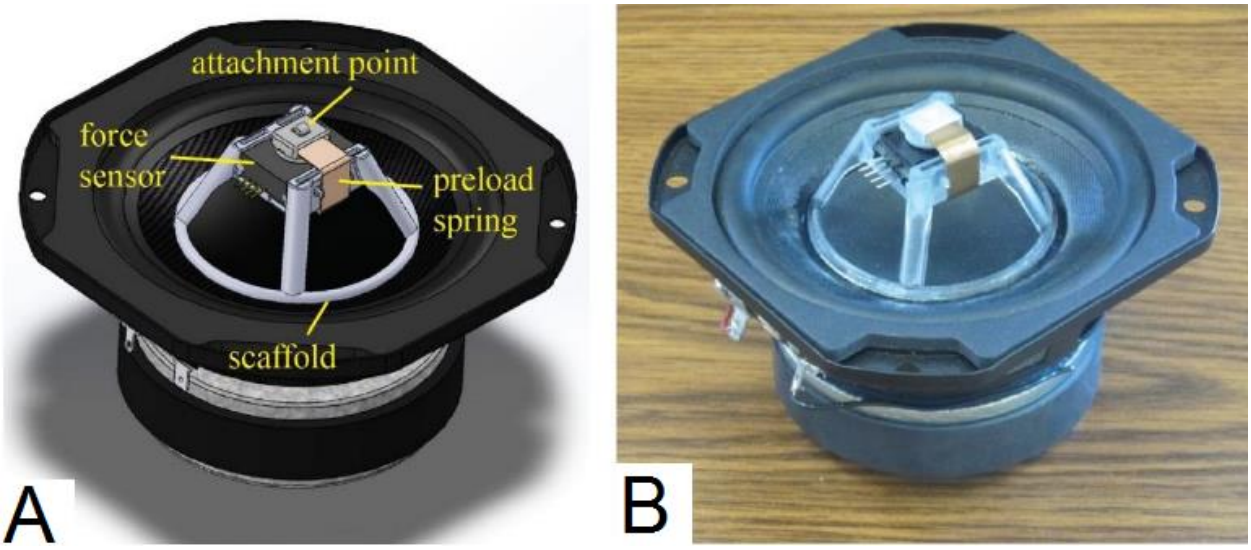


Figure 70. 1DoF Haptic Renderer sensor scaffold (A) CAD design and (B) as originally constructed. [168]

application will use the full 9.5 mm displacement range of the speaker over which we can render virtual environments. The compliance of the spider is known, but is assumed to deviate from its specified value much past the speaker's rest point. This will be an important consideration when we calibrate our device in the future.

To measure the force applied by the user, we designed a rigid scaffold which rests on the speaker cone. Within this scaffold sits a Honeywell FS03 force sensor, which is preloaded by a leaf spring that extends from a point underneath the scaffold to contact the force-sensitive plunger. The FS03 sensor is a similar form factor to the FS01 used in the HHFM Model-2, but is able to measure force over a wider range. The FS03 can measure forces up to 13.34 N, with 0.13 N accuracy (1% of full-scale) and 0.067 N hysteresis (0.5% of full-scale). This design is presented in CAD and as built, in **Figure 70**. The four posts supporting the corners of the platform, as well as the circular ring that interfaces with the cone, are all hollow to reduce the mass of the part. SLA was used to manufacture the scaffold, and its total mass came to only 6.2 grams. The scaffold was securely attached to the speaker cone using instant adhesive. The

attachment point is a small tubular structure to enable insertion of a needle or hook, and has a mass of 0.525 grams as shown in **Figure 70B**. Including the sensor, the entire scaffold structure has a mass of 12.6 grams, and therefore the mass of the entire moving structure (including the speaker cone itself) is only 23.6 grams. **Figure 71** depicts the 1DoF mounted within an aluminum box, with a modified attachment point, shaped like a post to enable users to easily push and pull on the speaker cone. The post attachment has a mass of 1.9 grams, bringing the total moving inertia to 25 grams. An aluminum cover is also included to protect the post and speaker from off-axis (non-vertical) forces.

The position of the speaker cone is measured by two mechanisms: a self-inductance based measurement, as well as an optical position measurement. Optical position measurement is accomplished by means of an infrared (IR) emitter and receiver (Vishay TCRT5000L; Malvern, PA) sensor mounted to the speaker frame. The IR sensor emits light at 940 nm and can measure distances up to 15 mm away, in a small footprint (10.2 mm x 5.8 mm x 7 mm). The IR sensor reflects light against a white surface that is glued to the speaker cone, such that, as the speaker

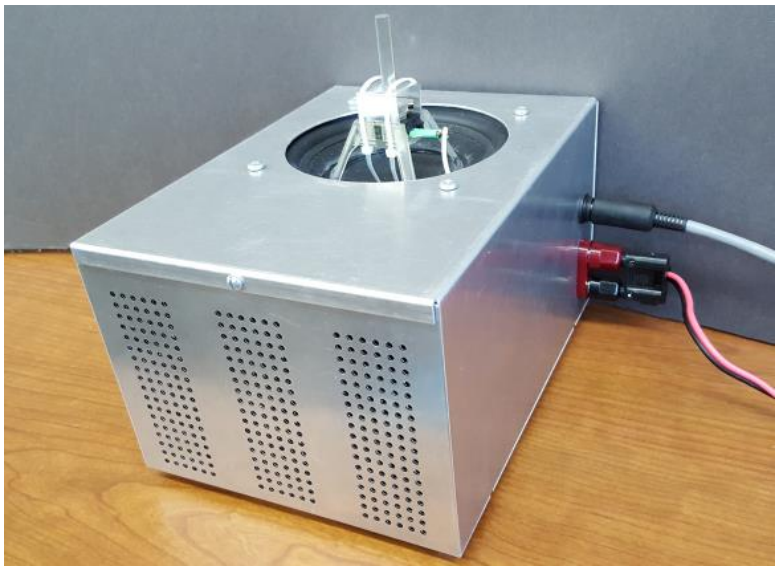


Figure 71. 1DoF inside enclosure and with post attachment.

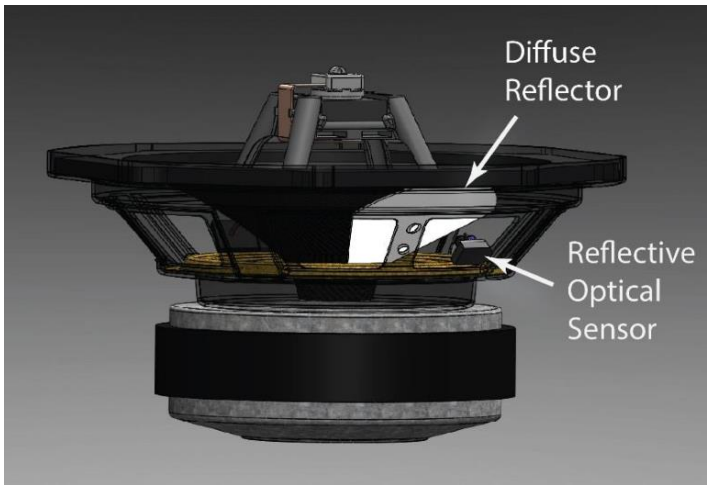


Figure 72. 1DoF haptic renderer position measurement with reflective infrared optical sensor. [168]

moves vertically, the reflective area and the distance to the sensor changes. This arrangement is shown in **Figure 72**. The IR distance measurement is robust, and is not significantly affected when exposed to sunlight or indoor lighting.

In addition to the optical measurement, we have explored methods to measure the coil position using inductance [168]. Inductance is a component of electrical impedance, whose magnitude is ascertained by injecting a sinusoidal voltage and measuring the sinusoidal current of the same frequency. Inductance is a complex quantity, given by

$$Z_L = j\omega L \qquad \text{(Eq. 30)}$$

where $j = \sqrt{-1}$; ω is the excitation frequency in radians per second; and L is the inductance, measured in Henries. **Figure 73** shows the magnitude of impedance in the Faital PRO speaker as a function of frequency, as adapted from the specification.

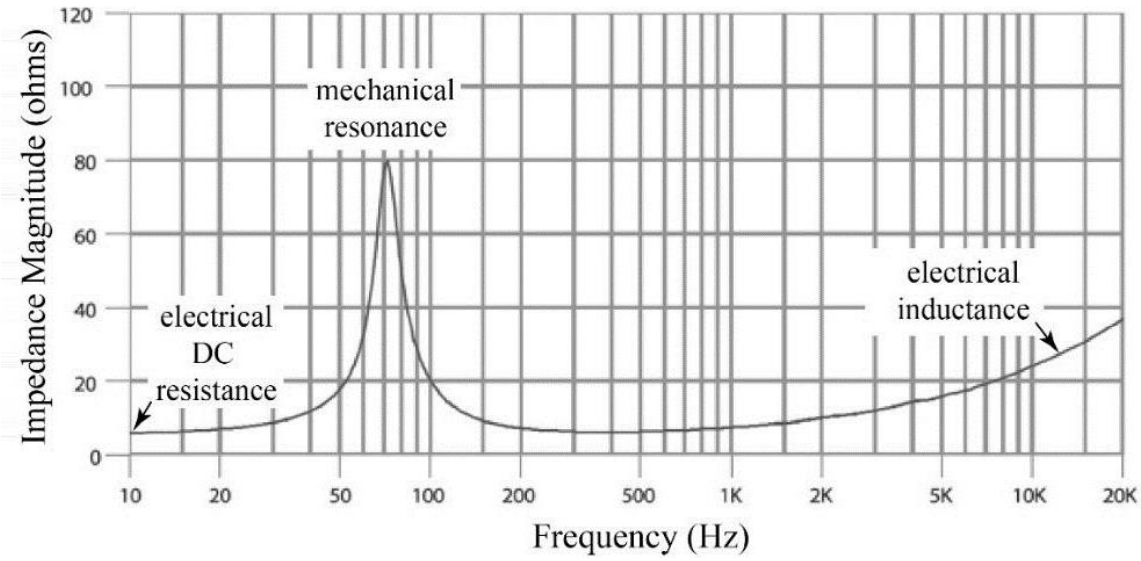


Figure 73. Speaker impedance as a function of frequency, adapted from the Faital PRO 5FE120 specification. [168]

To measure speaker position using inductance, we placed the primary winding of an air core transformer in series with the speaker, as shown in **Figure 74**. In this arrangement, the speaker is represented as a variable inductor, $Z_L(x)$, this inductance determining the amplitude of V_{out} by

$$V_{out} = V_S \frac{|Z_T|}{|Z_T| + |Z_L(x)|} \qquad \text{(Eq. 31)}$$

where x in this context is the position of the speaker cone; V_S is the source voltage; and Z_T is the impedance of the air core transformer. The "voltage divider" approximation given by Eqn. 31 assumes that the DC resistance of the air core transformer is negligible compared to the speaker, and that the transformer and peak detector circuit are both 100% efficient.

At high frequencies, Eqn. 31 is better represented by

$$V_{out} = V_S \frac{|Z_T|}{|Z_T + Z_L(x) + R_{eddy}|} \qquad \text{(Eq. 32)}$$

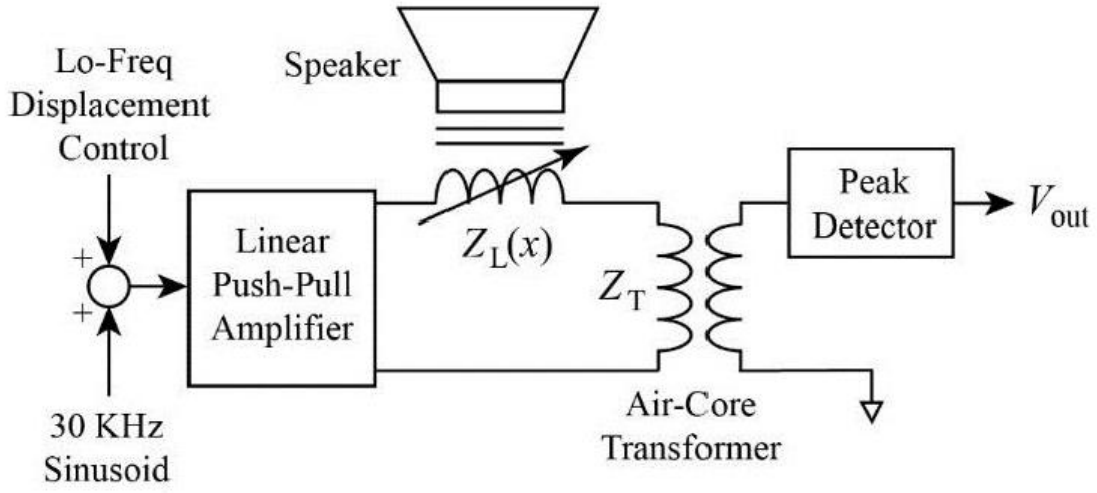


Figure 74. 1DoF inductance-based position measurement. [168]

where R_{eddy} represents the contribution of eddy currents in the permanent magnet of the speaker. The magnitude of R_{eddy} was investigated by recording the percentage change in V_{out} of our inductance measurement after placing a 400 gram mass on the cone scaffold to induce a step change in displacement. **Figure 75** shows that this percentage change decreases monotonically to zero because of the increasing influence of R_{eddy} at frequencies above our hearing capability (above 20 kHz). Due to these effects, we chose to use a 30 kHz sinusoid to drive our inductance-based measurement of position.

Impedance in the speaker may itself arise from both electrical and mechanical sources. Such electromechanical systems may be understood using *lumped parameter models*, such as the one presented in **Figure 76** [169]. The left and right halves of the lumped parameter model represent the electrical and mechanical properties of the speaker, respectively. Transduction between either side of the model is mediated by the electro-dynamical force factor, $Bl(x)$, which accounts for energy transfer in both directions, and depends on motion of the cone within the speaker. Electrical current causes a mechanical force that moves the speaker, while forced mechanical movements generate an opposing electrical voltage in the voice coil.

In this model, electrical quantities offer convenient analogies for mechanical phenomena. M_{ms} is the mechanical analogue of "inductance," the mass of the moving portion of the speaker. Air resistance working against the cone and damping due to the spider is represented as a "resistance," R_{ms}. The mechanical compliance is represented by the "capacitance," $C_{ms}(x)$, which is a function of x because the compliance is nonlinear at the extremes of displacement. Descriptions and values for the terms in this lumped parameter model for our particular speaker are summarized in **Table 3**.

Although the measurement of displacement by inductance was appealing in terms of not requiring additional instrumentation to the speaker, the superior accuracy of the optical method led us to use it instead of inductance for the purposes described hereunder in this dissertation.

Calibration of the voice coil was necessary to know precisely how much force is being generated by the haptic renderer. That force is a function of both current and position of the coil.

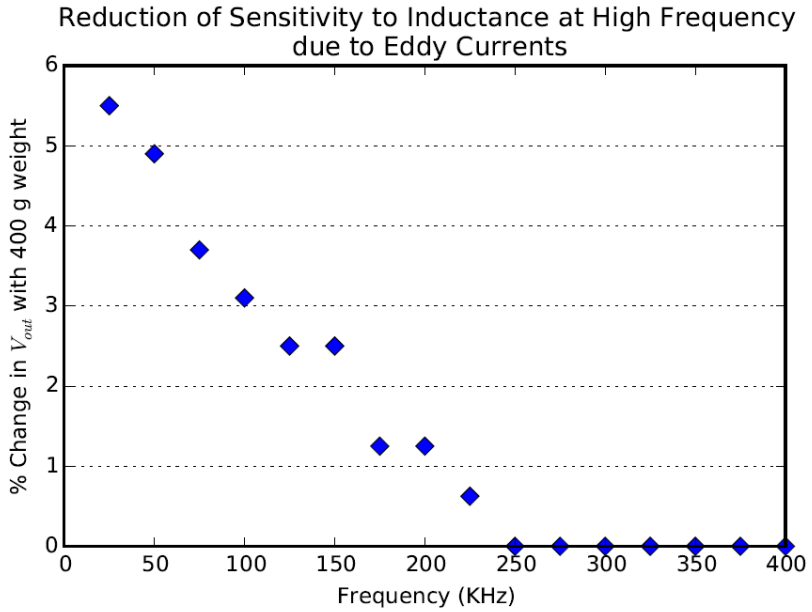


Figure 75. Percentage change in inductance-based position measurement as a function of sinusoid frequency. [168]

To perform this 2D calibration, we implemented a PID controller to simulate a "virtual wall." The virtual wall is a common simulation in haptics, in which the renderer generates an opposing force to match an external force, thereby keeping position constant. The virtual wall is a useful simulation because it allows the determination of the maximum force and stiffness that may be generated by the renderer. The speed of the response also gives a measure of its bandwidth. By moving the virtual wall to particular displacements along the voice coil's range, we can use the calibrated FS03 and the current through the coil to obtain the necessary 2D calibration for voice coil force. Our calibration results are described below.

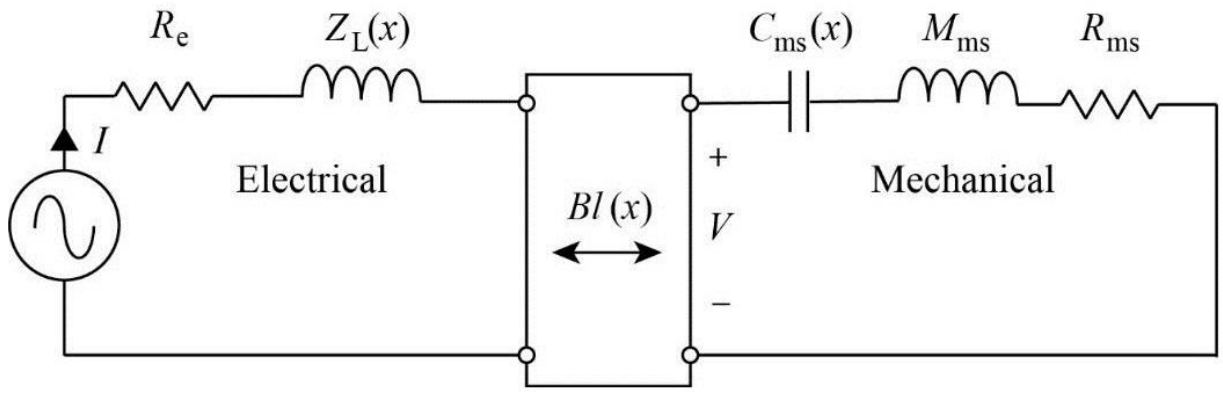


Figure 76. Lumped parameter model of audio loudspeaker, displaying electromechanical transduction and characteristics. [168]

Table 3. Electromechanical parameters -- definitions and values for the Faital PRO 5FE120 speaker.

Parameter	Description	Value in FaitalPRO 5FE120
R_e	DC electrical resistance	5.4 Ω
$Z_L(x)$	Electrical impedance due to inductance, variable with coil position	See **Figure 73**
$Bl(x)$	Electro-dynamical force factor, variable with coil position	6.9 N/amp (at rest)
$C_{ms}(x)$	Compliance of mechanical spider, variable with coil displacement	0.55 mm/N at rest ($x \approx 3.31$ mm)
M_{ms}	Mechanical mass of cone, including air load and voice coil (and added masses)	11 g (23 g with scaffold and force sensor)
R_{ms}	Mechanical friction and drag of loudspeaker cone	Negligible at low velocity

6.2.2 Haptic Renderer Control

The haptic renderer is controlled using an Analog Devices ADuC7026, the same microprocessor powering the HHFM. The ADuC7026 is designated as the "master" controller, and is connected to a "slave" laptop through a Wixel USB module (Pololu; Las Vegas, NV). **Figure 77** shows a system diagram of the various connections between the hardware.

The Wixel is a small programmable module that may be configured as a serial to USB interface using an onboard Texas Instruments (Dallas, TX) CC2511F32 microcontroller. Inter-Integrated Circuit (I^2C) and Serial Peripheral Interface (SPI) communication protocols are both supported, and the Wixel also has the capability to communicate wirelessly through a 2.4 GHz radio. The Wixel is programmed in C++ and flash loaded with a freely available configuration utility from Pololu. A custom GUI for the laptop was written in C++ using the Microsoft Foundation Class (MFC) libraries. Running on the connected laptop, the GUI adds data logging and visualization capability. **Figure 78** shows a screenshot of the GUI. While the ADuC7026 microcontroller is considered the "master" for purposes of system communication, the GUI

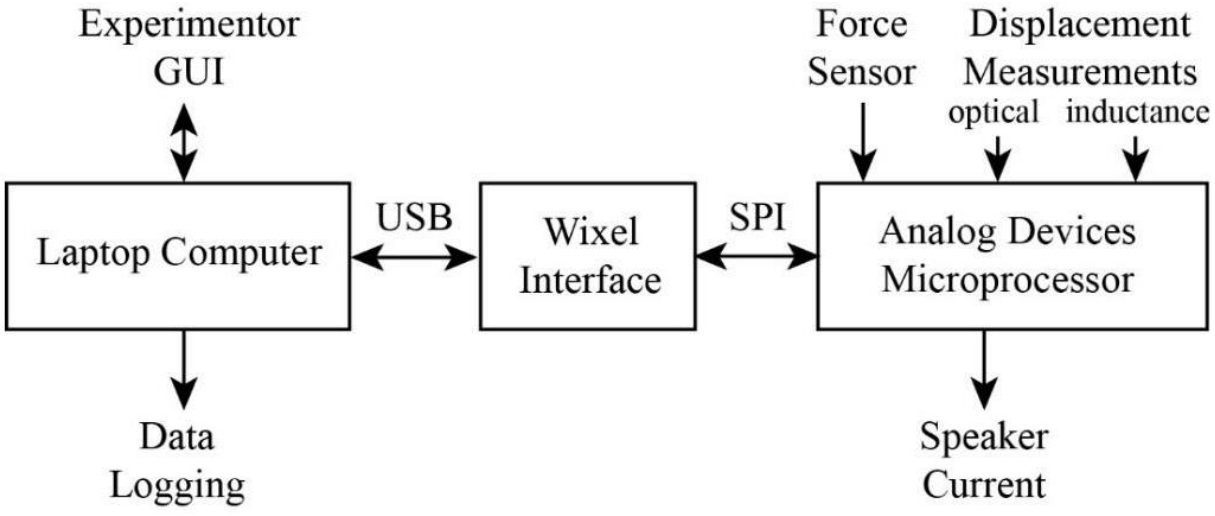


Figure 77. 1DoF haptic renderer system connections.

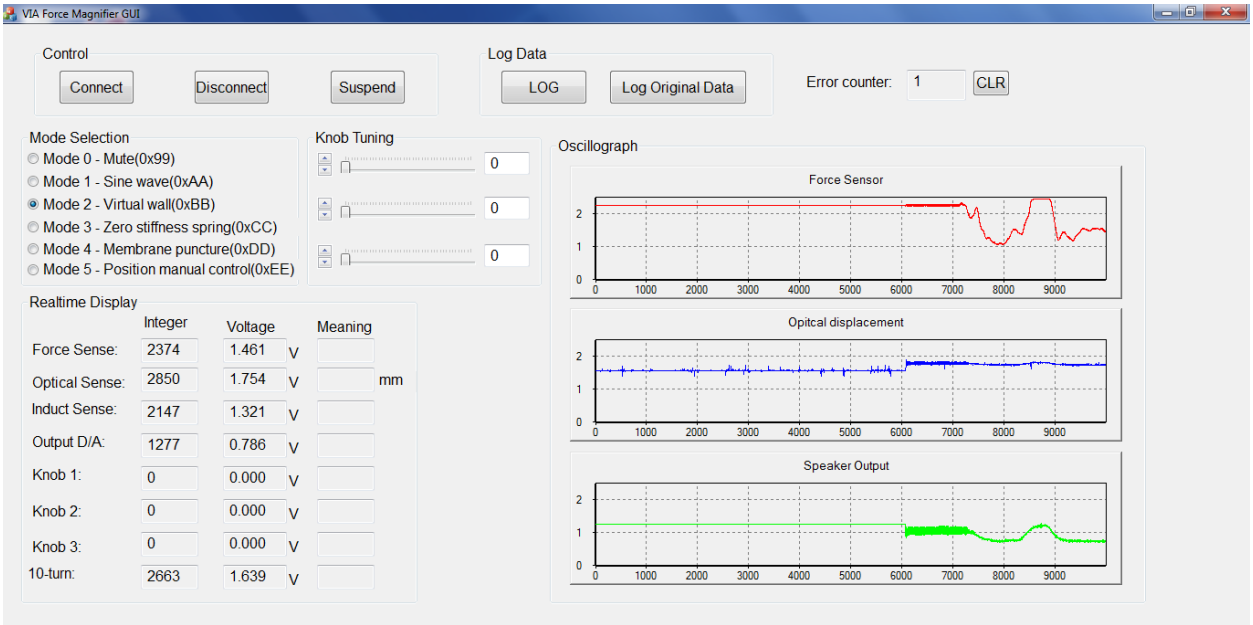


Figure 78. 1DoF haptic renderer GUI.

offers a convenient interface where one may switch between different "modes" or routines in the haptic renderer. Additionally, three virtual sliders were implemented in the GUI and used as digital knobs. Four physical potentiometers were also connected to analog inputs on the Analog Devices microprocessor to provide real-time control. The source code and interfacing electronic schematics for all three components – the ADuC7026, Wixel USB interface, and Windows GUI – are included in **Appendix E**.

The haptic renderer operates at 1 kHz using the ADuC7026's onboard timers and interrupts, ensuring a total within loop operation time of 1 ms. Communication from the ADuC7026 to the Wixel and laptop is achieved through SPI, a bidirectional communication protocol that uses 4 digital voltage lines. SPI is called bidirectional because messages are exchanged in both directions synchronously. In our system, a total of 14 bytes of data are exchanged between the master and slave at the end of each sensing-actuation loop. The master sends data from the various sensors and actuators, to be visualized in the GUI and saved to disk

Figure 79. 1DoF data (A) transmission and (B) reception, from the point of view of the ADuC7026 master.

for post processing. At the same time, the master receives control commands and other data (e.g. digital slider values) from the GUI. The ADuC7026 can transmit and receive one byte of data at a time, which requires 2.5 μs, and waits 10 μs between each byte, to ensure the Wixel's internal buffers do not overflow. In the end, bidirectional communication is accomplished within 300 μs, leaving 700 μs for computation and actuation.

The organization of the transmitted and received data is reflected in **Figure 79**. The data are represented using hexadecimal notation, where each alphanumeric character represents a 4-bit number. In hexadecimal notation, the numbers 0 – 9 represent values from zero to nine, and the letters A – F represent values from ten to fifteen. An initial handshake message of "0x11" is used by the ADuC7026 microprocessor to indicate to the Wixel that a new collection of bytes is about to be transmitted. Eight pieces of voltage data, at 12 bits each, are then transmitted sequentially. In **Figure 79**, voltage data are represented by a dummy sequence: "0xA01, 0xB02, 0xC03," etc. Data transmission is concluded with a second handshake message, "0x22," which is recognized by the Wixel as the stop command.

The ADuC7026 receives 14 bytes of data in a similar way, as shown in **Figure 79B**. An initial handshake message ("0x11") is sent by the Wixel, though this is not strictly necessary since the ADuC7026 is the master and initiates the transfer of data. Upon receipt of the master's start byte, a command byte is then sent to represent the desired mode as indicated by the GUI. We chose five distinct values to represent the various modes currently programmed into the 1DoF: "0x99" represents the "muted" condition, where no current is passed through the speaker; "0xAA" generates a sinusoidal current in the speaker with variable frequency and amplitude, controlled separately by the front panel potentiometers; "0xBB" puts the 1DoF in the "virtual wall" mode, which resists movement in response to an external force; "0xCC" calls the zero stiffness membrane mode, where the speaker cone actively minimizes the change in force on the force sensor; "0xDD" is a membrane puncture simulation using a combination of the virtual wall and zero stiffness spring settings to generate membranes of variable stiffness; and "0xEE" enforces manual control over speaker cone position using a front panel potentiometer. Following the command byte, relevant data from the GUI are transmitted to the ADuC7026. Up to seven pieces of 12-bit data may be transmitted to the Analog Devices microprocessor, for example, the values stored in digital sliders. There are a leftover 4 bits – a nibble – before the second handshake, "0x22."

Figure 80 shows data collected from the 1DoF when in its muted condition, with a user pushing and pulling on the force sensor. In this mode, only the natural stiffness of the spider opposes movement of the cone. **Figure 81** shows the 1DoF in the sine wave mode, where the amplitude and frequency of the sine wave were varied using the front panel knobs. At high frequencies and amplitudes, we can see inertial effects from the plunger on the force sensor.

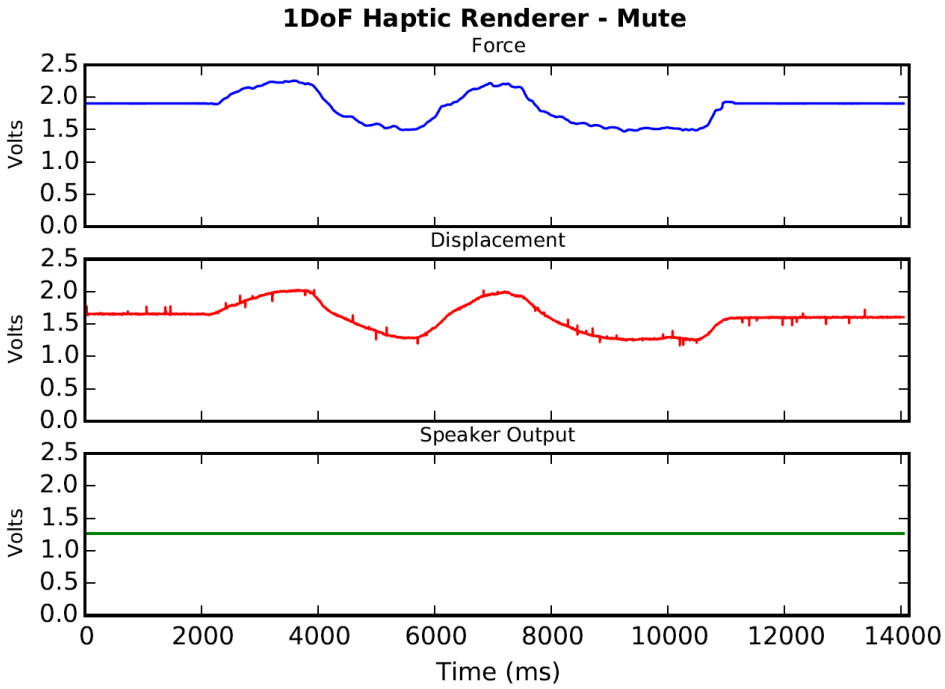


Figure 80. 1DoF haptic renderer in "mute" condition under the influence of external forces. Force (top), Displacement (middle), and Speaker (bottom) voltage data collected at 1 kHz through Wixel interface.

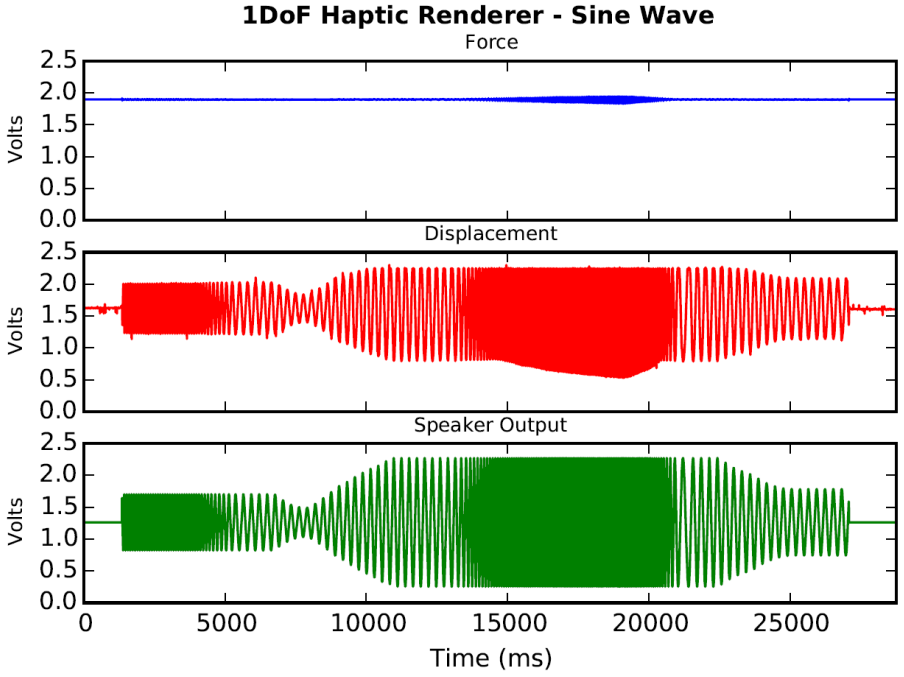


Figure 81. 1DoF haptic renderer generating sine waves of varying amplitude and frequency. Force (top), Displacement (middle), Speaker (bottom) voltage data collected at 1 kHz through Wixel interface.

6.2.2.1 Virtual Wall

As previously mentioned, a common simulation in haptics is the "virtual wall," a fixed position that resists movement in response to an external force. To simulate the virtual wall, a PID controller was implemented to control the position of the speaker cone. As already described, two sensors are available to measure position of the speaker cone: the inductance based measurement, and the infrared (IR) optical sensor mounted to the speaker frame [168]. Our simulation made use of the optical measurement, but we calibrated both methods of position measurement, as described next.

The position sensors were calibrated by manually controlling the speaker current, forcing the cone to move over its full 9.5 mm displacement range. The voltage from a front panel potentiometer was read by the Analog Devices microprocessor and used to directly control the speaker output. Ground truth displacement was measured using a standing gage (iGaging 35-128 DigiIndi; Los Angeles, CA) resting on the sensor scaffold. This gage has 0.0005 inch (0.01 mm)

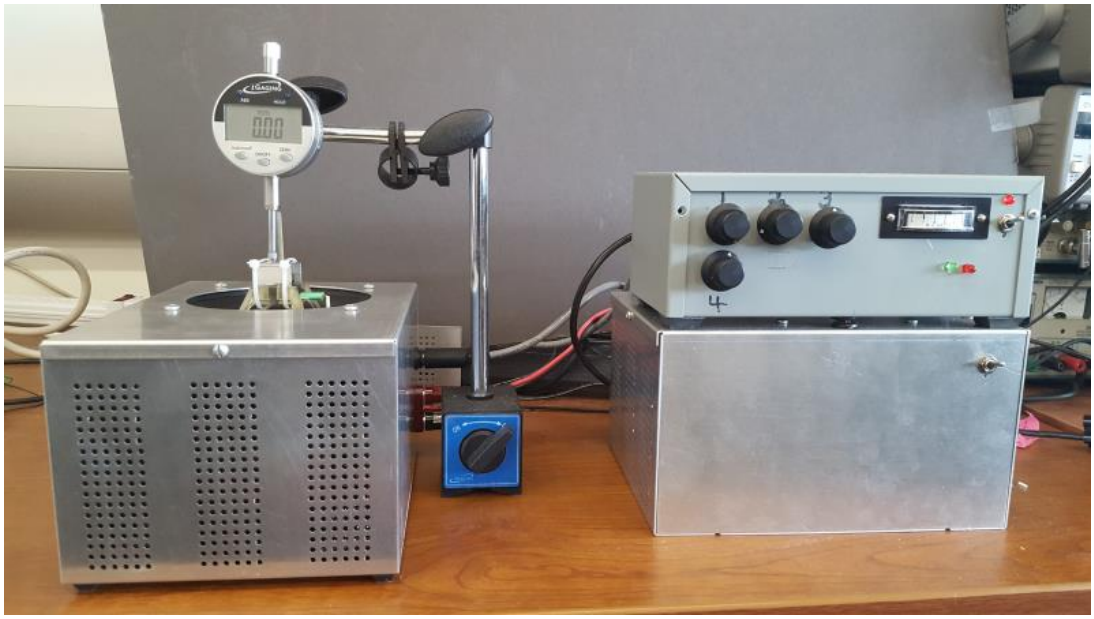


Figure 82. 1DoF displacement calibration setup.

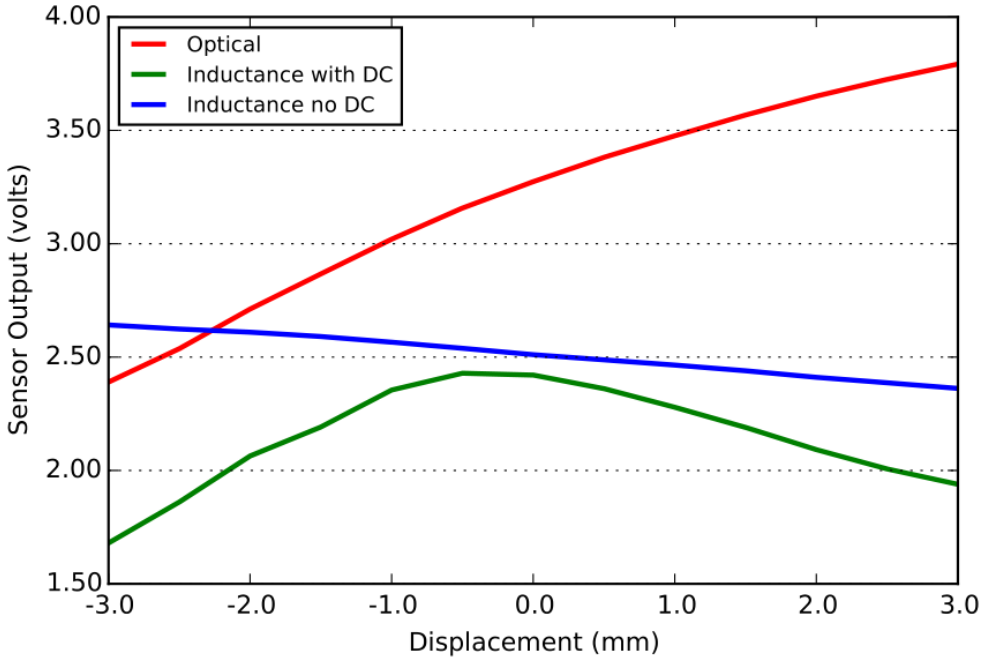


Figure 83. 1DoF position sensor output with respect to displacement from speaker rest position. [168]

resolution and is accurate to ±0.001 inches (0.024 mm) over its 1 inch (25.4 mm) total range. **Figure 82** shows this calibration setup. While the inductance measurement was available, preliminary explorations showed that the signal from this system was non-monotonic when the DC control voltage was superimposed over the 30 kHz sinusoid injected to measure inductance. **Figure 83** shows this behavior in the inductance circuit output as a function of displacement from its rest position. The "optical" and "inductance with DC" measurements were obtained using the front panel knob to control speaker position. The "inductance no DC" measurements were obtained by manually depressing or pulling on the speaker. Given this non-monotonicity in the inductance measurement, we decided to directly measure position using only the IR optical sensor.

Precise calibration of the IR sensor was accomplished using the following procedure. Voltages to the speaker and the IR sensor were recorded manually, as was the reading on the standing gage. The displacement baseline was defined as the speaker's lowest attainable position, when 2.5 V is output by the microprocessor, as controlled by the hardware potentiometer. Voltage and displacement data were then recorded in two series: first, from 2.5 V to 0 V and back to 2.5 V in even steps of the output voltage ($\Delta V_{out} = \pm 0.25$ V); and second, from 2.5 V to 0 V and back in even displacement changes as read from the standing gage ($\Delta x = \pm 0.25$ mm). **Figure 84** shows the collected data and the best fit line,

$$x = 2.1667 * V_{IR}^2 - 1.2489 * V_{IR} + 0.32 \qquad \text{(Eq. 33)}$$

where x is the current displacement of the speaker in mm; and V_{IR} is the voltage in the IR sensor.

It should be noted that the weight of the gage probe precludes simply using the gage to measure the speaker's voltage-to-displacement relationship. This is because additional voltage would be required to compensate for the weight of the gage assembly. Our strategy, then, was to first calibrate the internal IR sensor, and use that calibration to characterize the speaker voltage-to-displacement relationship without the mechanical gage. Polynomial least square fits were made to the IR measurements at the various speaker voltages. It was found that quadratic functions (polynomials of degree 2) fit the IR sensor data best. Given this known calibration factor for the IR sensor, we could then assemble a displacement versus DC voltage curve, with and without the gage. **Figure 85** shows the offset between measurements of displacement with and without the gage probe. The best fit line to these data are cubic functions, reflecting the nonlinear compliance of the spider at the extremes of displacement.

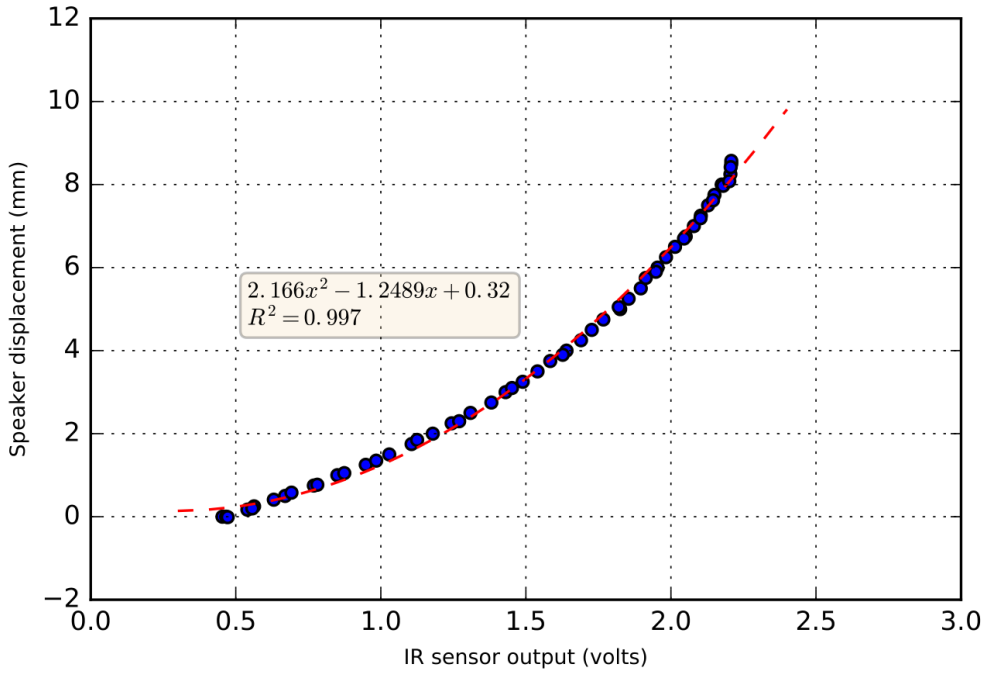


Figure 84. 1DoF infrared optical displacement sensor calibration data and best fit line.

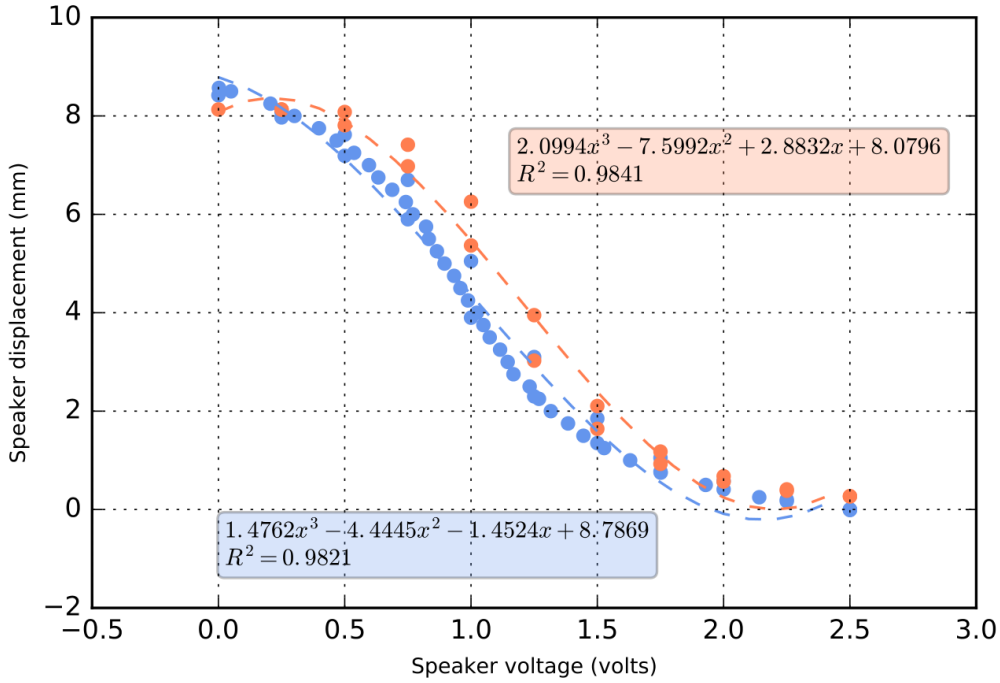


Figure 85. Speaker displacement as a function of output voltage with (blue) and without (orange) external gage, and their best fit lines.

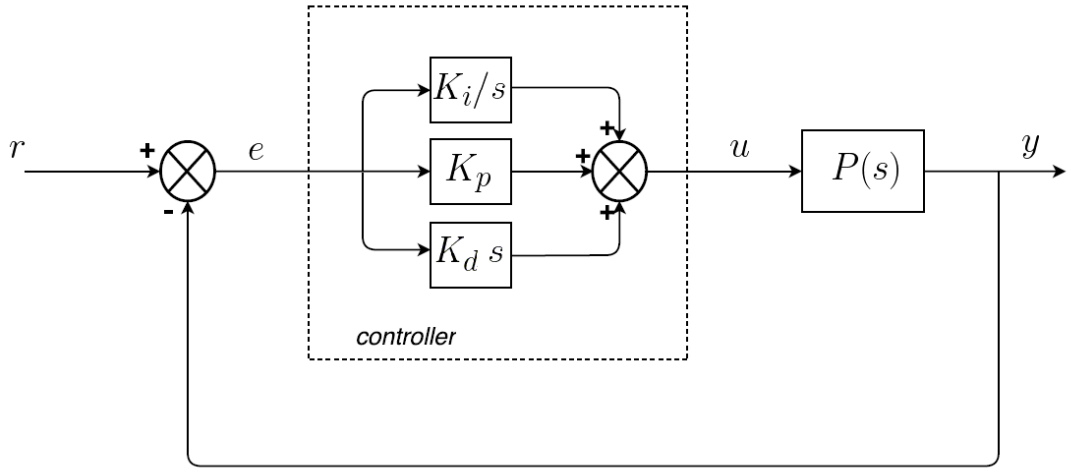


Figure 86. Canonical PID controller. [170]

To render a virtual wall, a PID controller was implemented, as presented in **Figure 86** [170]. The following generic variables are used to represent quantities in the PID controller: y is the output measurement to be controlled; r is the reference point or set-point; e is the error, the difference between the current output and the reference; u is the controller output; P(s) is the "plant" or system being controlled; and K_p, K_i, and K_d are the proportional, integral, and derivative gains, respectively.

In **Figure 86**, integrals and derivatives of the error are represented by dividing and multiplying by the complex frequency variable, s, by way of the Laplace transform. In our system the plant being controlled is the 1DoF actuator. The output measurement for the virtual wall mode is the current speaker cone position, x. The output of the canonical PID controller, in the time domain, is then calculated as

$$u(t) = K_p * e(t) + K_i \int_0^t e(\tau)\, d\tau + K_d \frac{de(t)}{dt} \qquad \text{(Eq. 34)}$$

where K_p, K_i, and K_d are the proportional, integral, and differential gains of the controller, respectively; and $e(t) = r - y$. Eqn. 34 is called the *expanded form* of the PID equation [171].

Of course, the controller is implemented on a digital system, so the output is actually represented in a discrete form in the time domain as

$$u_k = K_p * e_k + K_i \sum_{j=1}^{k} e_j \, \Delta t + K_d \frac{(e_k - e_{k-1})}{\Delta t} \qquad \text{(Eq. 35)}$$

where e_k is the current error, and Δt is the sampling period.

The transfer function for the canonical PID controller, expressed using the Laplace transform, is

$$G(s) = \frac{U(s)}{E(s)} = K_p + \frac{K_i}{s} + K_d s \qquad \text{(Eq. 36)}$$

where $G(s)$ is the generic variable for transfer functions; and $U(s)$ and $E(s)$ are the Laplace transforms of the controller output and error, respectively.

To generate the virtual wall, the desired set-point position was read from a front panel potentiometer knob. A linear transform between the voltage across the potentiometer and the speaker's full-scale displacement is used to obtain the desired position in millimeters. The current position of the cone, x, is read from the IR sensor and converted to millimeters using its known calibration equation (Eqn. 33).

Error in position is then calculated as

$$e(t) = x - x_{sp} = y - r \qquad \text{(Eq. 37)}$$

where $e(t)$ is the error in position; and x_{sp} is the set-point, or desired position. In Eqn. 37, the generic variables y and r represent the process output and reference point, respectively, as

Table 4. Ziegler-Nichols and Tyreus-Luyben PID tuning parameters [171]

Ziegler-Nichols	K_c	τ_I	τ_D
P	$0.50\ K_u$	—	—
PI	$0.45\ K_u$	$P_u/1.2$	—
PID	$0.60\ K_u$	$P_u/2$	$P_u/8$
Tyreus-Luyben	K_c	τ_I	τ_D
PI	$0.31\ K_u$	$2.2\ P_u$	—
PID	$0.45\ K_u$	$2.2\ P_u$	$P_u/6.3$

labeled in **Figure 86**. The set-point is subtracted from the current position in the virtual wall because control over the speaker is "negative," due to the arrangement of our system electronics: the speaker voltage must be reduced to push upwards against a downwards external force.

PID controller gains were tuned using the open-loop Ziegler-Nichols (Z-N) method [172]. Briefly, the procedure is as follows. Initially, K_i and K_d are set to zero. K_p is then increased from zero until the system oscillates at some stable frequency. The gain at which the system went unstable, K_u, the "ultimate gain," is recorded, as well as the period of oscillation, P_u. The PID constants may then be directly calculated according to the relationships listed in **Table 4**.

The relationship between the Z-N parameters (K_u and P_u) and PID controller gains are defined with respect to the *parallel form* of the PID equation, where Eqn. 34 is re-expressed as

$$u(t) = K_c \left[e(t) + \frac{1}{\tau_I} \int_0^t e(\tau)\, d\tau + \tau_D \frac{de(t)}{dt} \right] \quad \text{(Eq. 38)}$$

where K_c is a dimensionless controller gain; τ_I is the *integral* or *reset* time, with units of time; and τ_D is the *derivative* time, also in units of time. Therefore we can use the following relationships to convert back and forth between these two forms and their associated constants

$$K_i = \frac{K_c}{\tau_I} \quad \text{(Eq. 39)}$$

$$K_d = K_c * \tau_D \quad \text{(Eq. 40)}$$

Z-N tuning was performed using a virtual slider from the GUI, which was represented as a 12-bit number, taking any value between 0 and 4095. Using a virtual slider is more accurate than a hardware knob, because there is unavoidable measurement noise when reading from a real potentiometer. Even small variations in the gain while tuning can cause the system to oscillate at lower values of K_u. Tuning with the virtual slider allows us to increase the gain one bit at a time, which provides excellent resolution for finding the "true" value of K_u.

Using the Z-N procedure, we found the ultimate gain to be $K_u = 1.50$, and the oscillation period, P_u, to be 0.025 seconds for the virtual wall mode. We found the most stable controller gains to be the Tyreus-Luyben PID settings, which are considered to be more conservative than the Ziegler-Nichols settings. The PID controller equation, with tuned constants, is then

$$u(t) = 0.681 * e(t) + 12.382 \int_0^t e(\tau)\, d\tau + 0.00271 \frac{de(t)}{dt} \quad \text{(Eq. 41)}$$

which corresponds to system behavior as depicted in **Figure 87**.

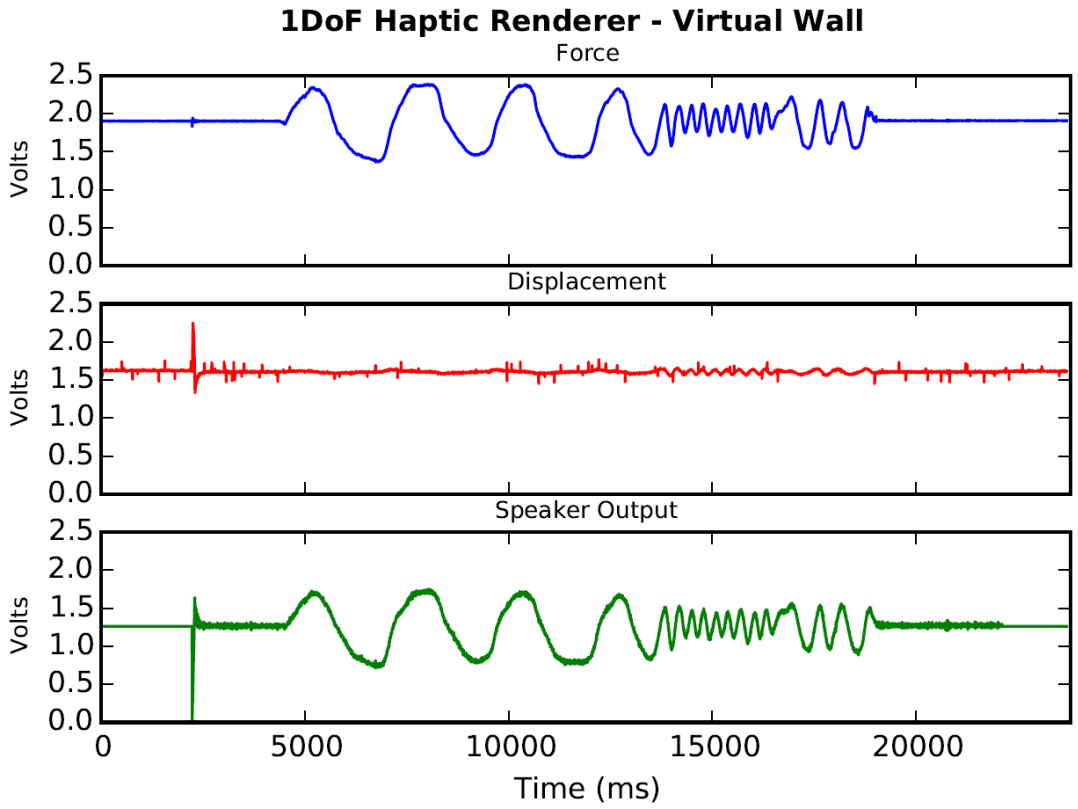


Figure 87. 1DoF haptic renderer in virtual wall mode with canonical PID control. Force (top), displacement (middle), and output (bottom) voltage data collected at 1 kHz through the Wixel interface.

Using this implementation of the canonical PID controller was associated with significant amounts of noise, which is rendered as vibration in the 1DoF speaker cone. This noise can be clearly seen in the output voltage trace in **Figure 87** (green). A typical source of this noise is the derivative term in the controller. Further, a large "kick" in the system is seen when initializing the virtual wall mode, manifesting as the large spikes in both the displacement (red) and output (green) voltage traces in **Figure 87**. These large transients are likely due to action of the proportional and derivative terms when initializing the error calculation. Overall, position was well controlled when the external force (blue) was applied slowly, but fast changes were not as well tolerated.

To address these issues, several advanced features were implemented to augment the PID controller [170]. Set point weights, β and γ, were introduced to scale the reference point, r, and the derivative of the reference point, respectively. These constants were applied to the proportional and derivative terms of the controller equation, such that, for positive control (i.e., when $e = r - y$) the PID controller output is now given by

$$u(t) = K_p * (\beta r - y) + K_i \int_0^t \big(r(\tau) - y(\tau)\big)\, d\tau + K_d \left(\gamma \frac{dr}{dt} - \frac{dy}{dt} \right) \qquad \text{(Eq. 42)}$$

where u(t) is the generic variable for the controller output; $\beta = [0, 1]$; and usually, $\gamma = 0$. When $\gamma = 0$, such action is called "taking the derivative on measurement" [170]. One major advantage to taking the derivative on the measured variable is that changes in the reference point do not result in large "kicks" in the controller output, as we saw with the canonical PID controller.

Additionally, a low-pass filter was implemented on the derivative term, so as to further reduce the noise associated with derivative action of the controller. This filter is given by the transfer function

$$G(s) = \frac{1}{1 + \tau_f s} \qquad \text{(Eq. 43)}$$

where τ_f is some small time constant. Combining the set point weighting and filtered derivative actions together, the revised proportional and derivative terms (P_k and D_k, respectively) for the controller in their discrete forms become

$$P_k = K_p * \left(x_{curr} - \beta x_{sp}\right) = K_p * (y - \beta r) \qquad \text{(Eq. 44)}$$

$$D_k = \frac{\tau_f}{\tau_f + \Delta t} D_{k-1} - \frac{K_d}{\tau_f + \Delta t}(y_k - y_{k-1}) \qquad \text{(Eq. 45)}$$

where, again, control is negative for the proportional term.

In our controller, we chose $\beta = 1$ and $\gamma = 0$, and set the time constant for the derivative term filter to be

$$\tau_f = \frac{\left(K_d / K_p\right)}{N} = \frac{\left(0.00271 / 0.681\right)}{20} = 1.9897 \times 10^{-4} \qquad \text{(Eq. 46)}$$

where $N = [2, 20]$ [170]. N was chosen to be 20, so τ_f was as small as possible.

Integral action is often associated actuator saturation, since errors sum to large values very quickly and the integral term in Eqn. 34 dominates the controller output. An additional error signal, e_s, was introduced to prevent actuator saturation due to this accumulation, a technique called "anti-windup reset." The e_s term works by reducing the controller output, u, if the commanded controller output, v, exceeds the actuator's output limits. Thus in discrete form, the integral term, I_k, is given by

$$I_{k+1} = I_k + K_i * \Delta t * e_k + K_t * (sat(v) - v)$$

$$= I_k + K_i \, \Delta t \, e_k + K_t \, e_s \qquad \text{(Eq. 47)}$$

where K_t is the anti-windup gain; e_s is the saturation error signal; and $sat(v)$ is the "saturation function" that simulates actuator saturation and enforces limits based on system knowledge. In our system, since the Analog Devices microprocessor can only generate up to 2.5 V, the saturation function is given by

$$sat(v) = \begin{cases} 2.5 & if\ v > 2.5 \\ 0 & if\ v < 0 \\ v & otherwise \end{cases} \quad \text{(Eq. 48)}$$

The final systems diagram of the controller with set point weighting, derivative filtering, and anti-windup reset is shown in **Figure 88** [170]. We chose the anti-windup gain to be $K_t = K_i/K_p$.

As shown in **Figure 89**, with these modifications to the canonical PID controller, noise in the speaker output is reduced, and position is better maintained, even with faster variations in force. The kick associated with initializing the virtual wall mode has also been eliminated, due to taking the derivative on measurement.

In addition to the PID controller with set point weighting and derivative filtering, further limits to the maximum change in controller output were implemented in software. This step was necessary because random noise in the optical sensor is amplified by the proportional term of the controller and is perceptible in the rendering device as vibration. Therefore, a maximum slew rate limit of 25 mV between actuation loops was enforced to cancel out this undesirable effect in

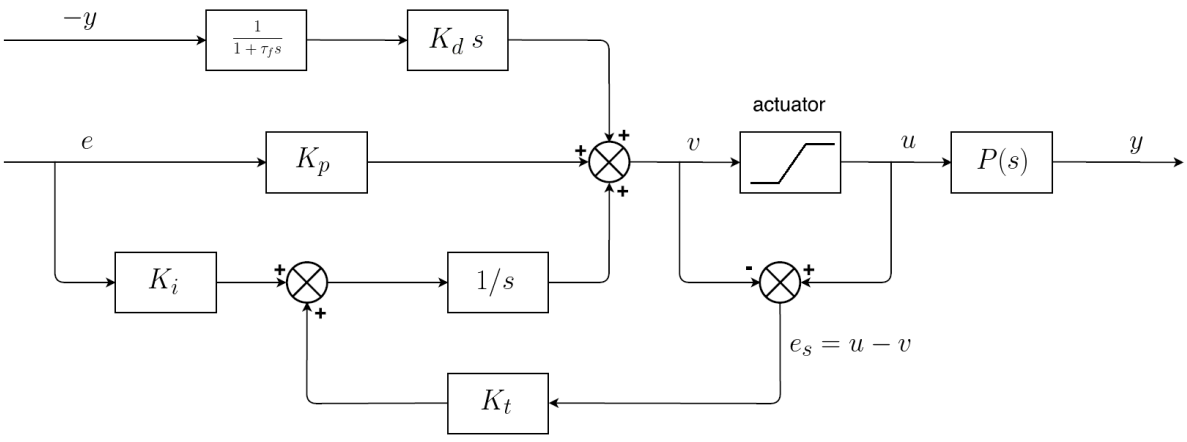


Figure 88. PID controller with set point weighting, anti-windup, and derivative filtering. [170]

both PID controller implementations. While this adds some "viscosity" to the response of the device – since now large voltage changes are forced to happen over longer timeframes – the device is much more stable. Since vibration may be perceived by users with exquisite sensitivity, this tradeoff in performance is acceptable. Our Z-N tuning was done with this slew-rate limitation enabled, so the controller gains take this aspect of the system into account.

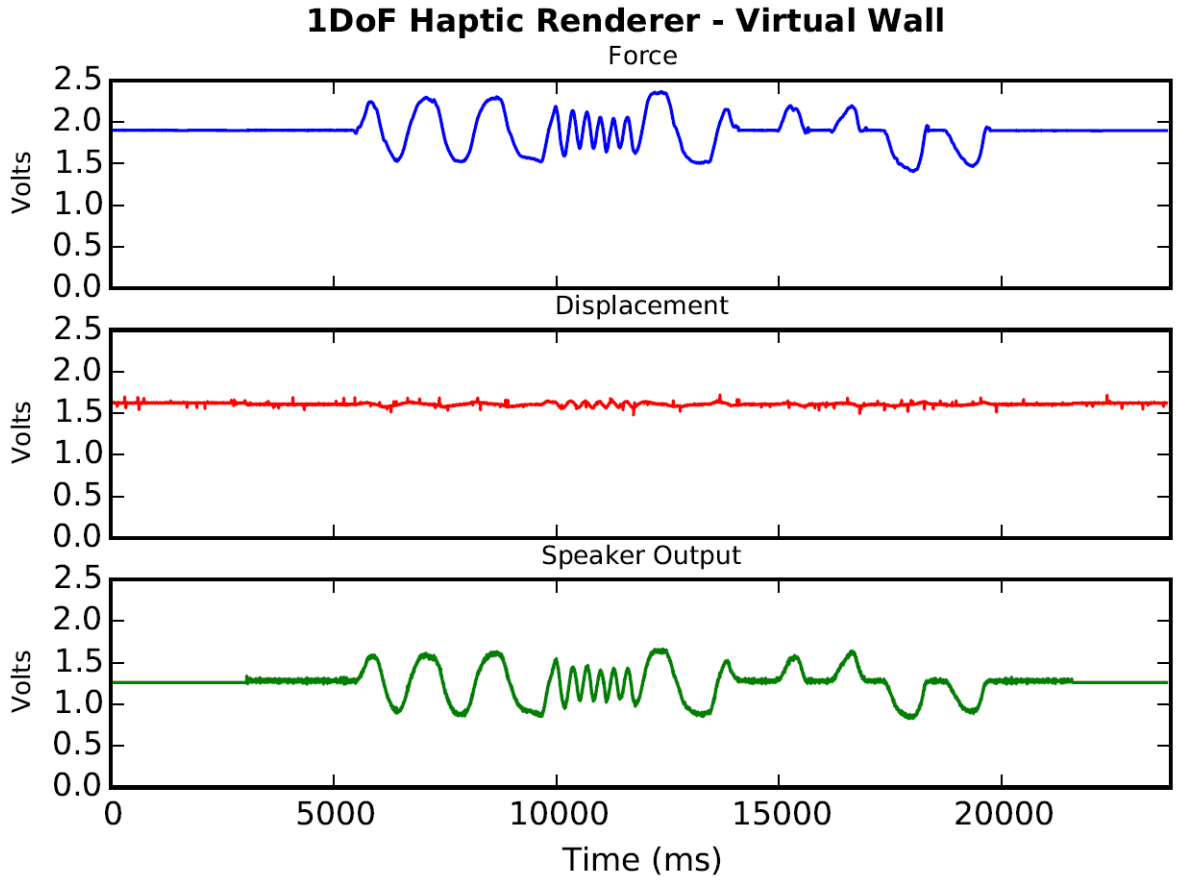


Figure 89. 1DoF haptic renderer in virtual wall mode with set-point weighting, anti-windup, and derivative filtering. Force (top), displacement (middle), and output (bottom) voltage data collected at 1 kHz through the Wixel interface.

6.2.2.2 Zero Stiffness Spring

A second simulation common in haptic rendering is to reproduce empty space, akin to rendering a mechanical spring with zero stiffness. In the context of our haptic renderer, this means that, in response to an externally applied force, the speaker cone should not resist its movement whatsoever. Indeed, the renderer should actively "help" the user move the cone, making the cone feel weightless.

To implement this behavior into the haptic renderer, a PID controller was implemented using the FS03 force sensor as its input. In this case, voltage on the preloaded sensor was sampled at initialization and this force value was stored as the set point. The output of the controller was then changed so as to eliminate any offset to the preload, where the error is defined as

$$e(t) = F_{sp} - F = r - y \qquad \text{(Eq. 49)}$$

where F_{sp} is the force set-point, the preload on the FS03 sensor; and F is the current force on the sensor. Error is so defined because the control is positive. For example, to relieve an increase in the preload (i.e., a push onto the sensor), the speaker must move downward, which requires an increase in the speaker voltage.

The Ziegler-Nichols tuning procedure was again utilized to tune the PID controller, as described previously. The ultimate gain and period of oscillation are expected to be different from the virtual wall implementation, because the two behaviors utilize different sensors to drive output behavior. Indeed, the ultimate gain was now found to be $K_u = 7.348$, corresponding to an oscillation period of 0.002 seconds. Again, the Tyreus-Luyben settings were utilized. The final PID output equation for this system is then

$$u(t) = 3.3066\, e(t) + 1670 \int_0^t e(t')\, dt' + 0.0023 \frac{de(t)}{dt} \qquad \text{(Eq. 50)}$$

The 1DoF in zero stiffness mode with a canonical PID controller is shown in **Figure 90**. While the zero stiffness action works very well in the middle of the displacement range, when the speaker reaches its extremes, additional force is required to overcome the buildup of error in the integral term, making the cone motion feel "sticky." Notice the small bumps in the force trace (blue, top) in **Figure 90** after the speaker had reached the extremes of its displacement.

Set-point weighting, anti-windup reset, and derivative filtering were added to the zero stiffness controller to improve its performance. As in the virtual wall mode, set point weights β and γ were set to 1 and 0, respectively. The N associated with calculating τ_f was again chosen to be 20, so $\tau_f = 3.477 \times 10^{-6}$ as dictated by Eqn. 46. **Figure 91** shows the zero stiffness behavior with these advanced features implemented. Anti-windup in particular was very effective, allowing the speaker to move in the opposite direction immediately upon reaching one extreme.

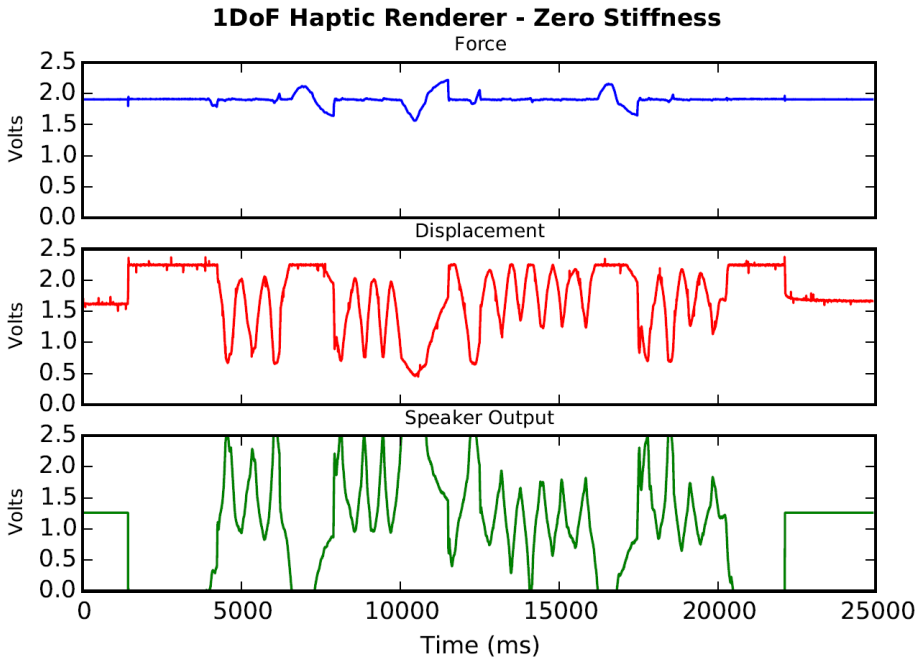


Figure 90. 1DoF haptic renderer in zero stiffness mode with canonical PID control. Force (top), displacement (middle), and output (bottom) voltage data collected at 1 kHz through the Wixel interface.

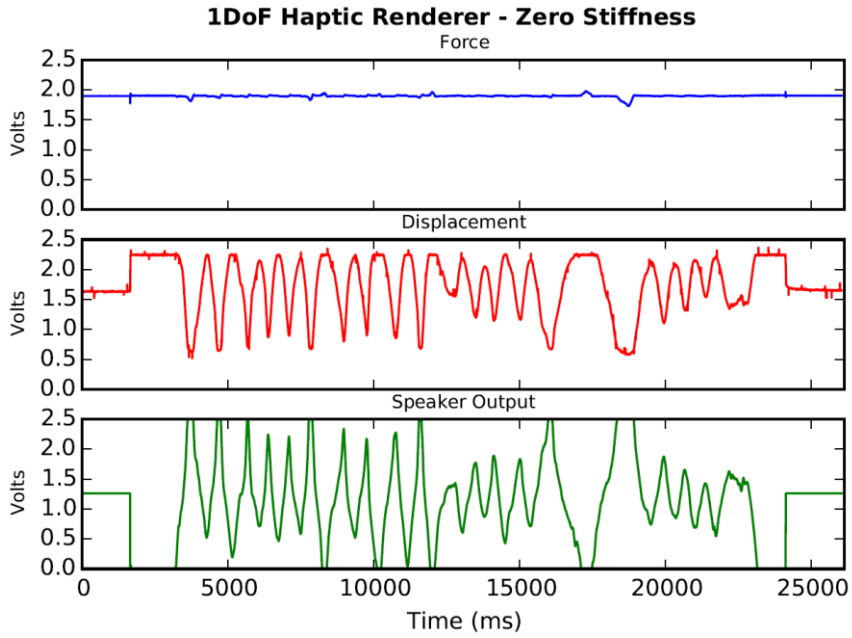


Figure 91. 1DoF haptic renderer in zero stiffness mode with set-point weighting, anti-windup, and derivative filtering. Force (top), Displacement (middle), and Output (bottom) voltage data collected at 1 kHz through the Wixel interface.

6.2.2.3 Membrane Puncture

By combining these two basic behaviors, the virtual wall and zero stiffness spring, we could simulate membrane puncture as was done on the MLHD. As detailed in Chapter 4, our original membrane puncture experiments were programmed by defining two regions with differing properties: the membrane, with a defined stiffness and thickness; and free space, the regions before and after the membrane in which the moving mass is made weightless. A simple simulation of membrane puncture was implemented using a threshold in the measurement from the force sensor as a surrogate for the critical puncture force. In this simulation, the zero stiffness spring simulation was implemented for two regions above and below the membrane. As soon as

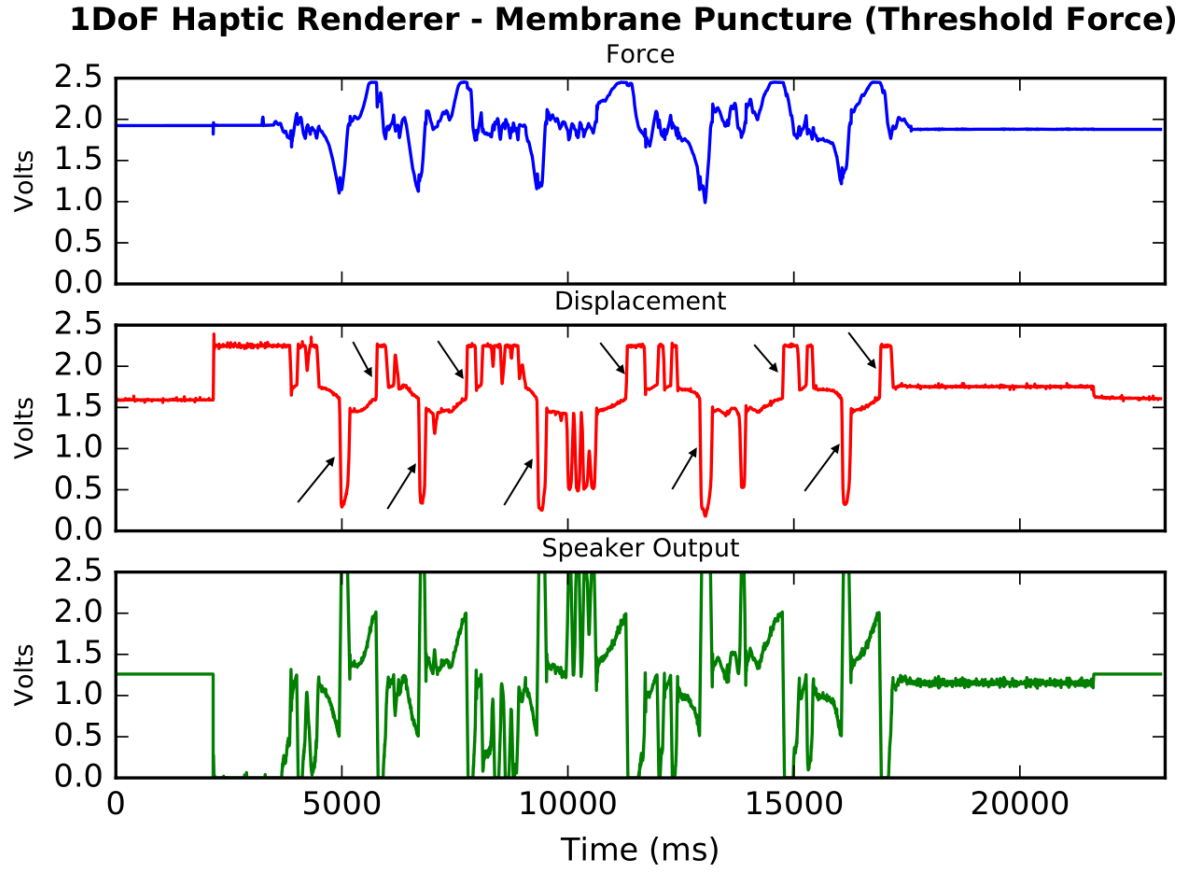


Figure 92. 1DoF membrane puncture simulation using threshold force. Force (top), Displacement (middle), and Output (bottom) voltage data collected at 1 kHz through the Wixel interface.

the cone reached some predetermined location for the membrane, the virtual wall mode was enabled. Force would build on the sensor as the user continued to push or pull, until a critical threshold of force measurement was reached. At this "fracture point," the zero stiffness spring was again rendered. **Figure 92** shows the data logged from punctures so implemented. In **Figure 92** the sharp spikes in the displacement (arrows) represent a single puncture.

This rendering is not strictly a true virtual membrane, because the user does not progress through the membrane by distending or deforming it, building in force with displacement following Hooke's law. In order to render a membrane of some arbitrary stiffness, it is imperative to calibrate the speaker's voltage – force relationship along its displacement range. Knowing these relationships, we could generate feedback forces as a function of position and velocity, as was done in the MLHD.

The virtual wall mode was used to determine the voltage – force relationship at 0.5 mm intervals along the displacement range. Knowing the force sensor calibration factor (40 mV/50 grams for the Honeywell FS03), we could then apply forces to the system and obtain the speaker's force – current relationship at those positions. The sensor and actuator voltages were then logged through the GUI. The raw data and calibration equations obtained from this process are depicted in **Figure 93**.

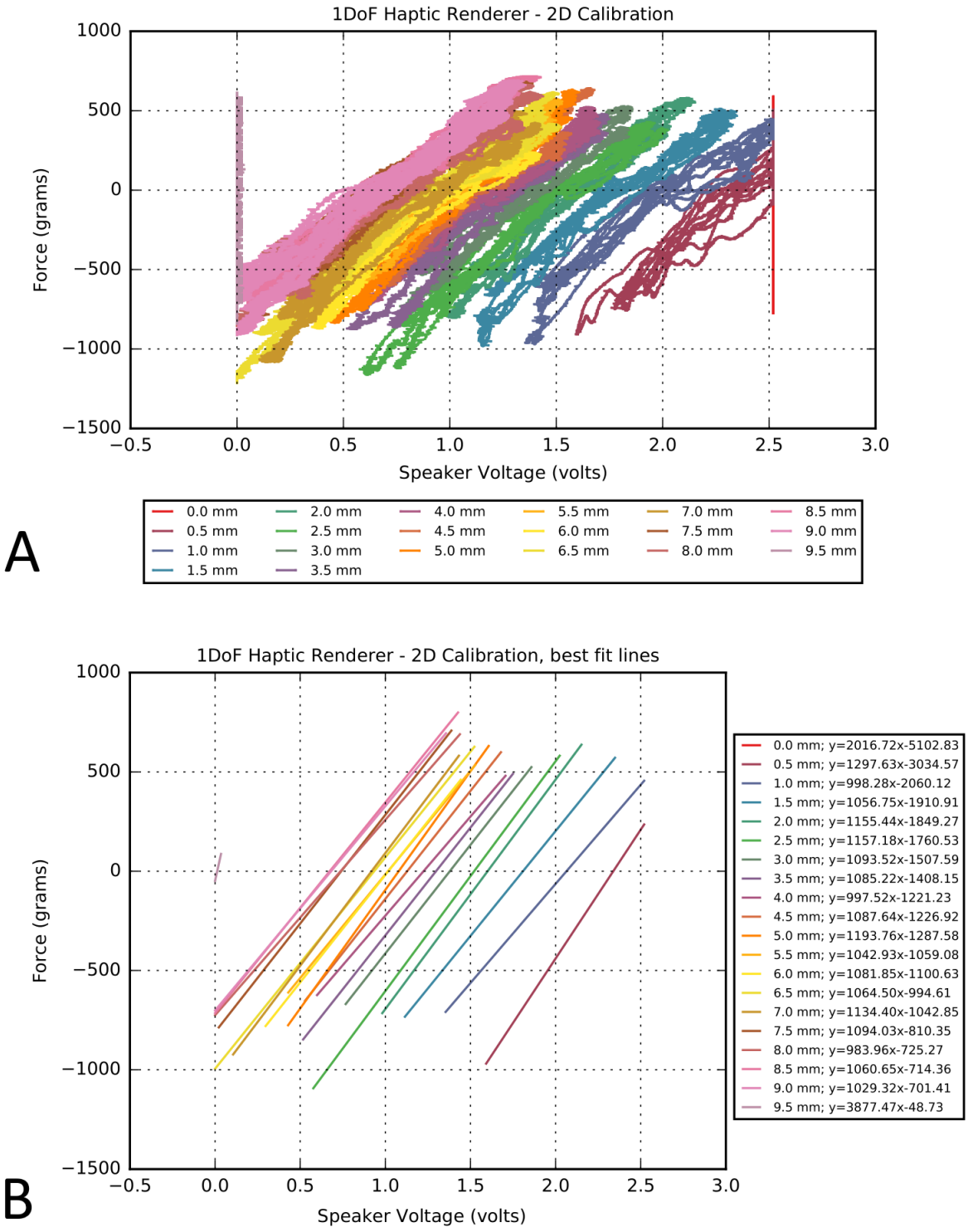


Figure 93. 1DoF Haptic Renderer 2D force calibration (A) raw data and (B) calibration best fit lines, obtained at various displacements by slowly applying force to the cone while in the virtual wall mode.

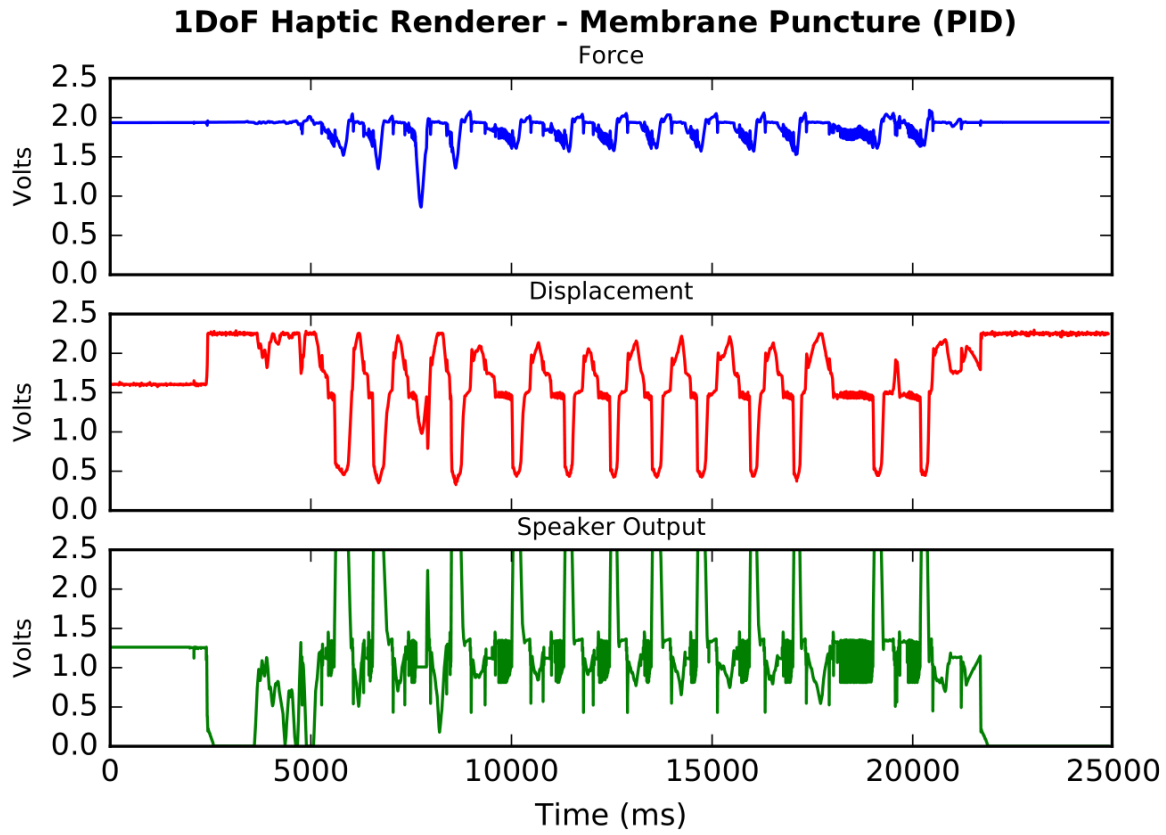


Figure 94. 1DoF membrane puncture simulation using PID and interpolated force, position. Force (top), Displacement (middle), and Output (bottom) voltage data collected at 1 kHz through the Wixel interface.

We found that, in this rendering formulation, the speaker cone would tend to "bounce" on the membrane, as shown in **Figure 94**. This bounce is rendered as a significant vibration at the interface between the membrane and free space regions. Part of the reason this may happen is due to noise in the IR sensor measuring displacement. Due to this inaccuracy, the system may unpredictably jump from the region of free space, to deep within the membrane, and back, between successive actuation loops. The slew-rate limiting imposed on the output voltage helps in this regard, but the interaction between membrane and free space is still affected by this noise.

6.3 DISCUSSION

In this chapter, we presented the development of an inexpensive, single-axis haptic renderer, and implementation of control strategies to produce several virtual environments. By choosing the appropriate sensor and set-points, varied behaviors may be implemented, such as the virtual wall or zero stiffness spring. PID control offers stability and reliability for these simple simulations, but sensor noise may not be well tolerated even with advanced features like set-point weighting and derivative filtering, in more complex simulations.

7.0 DISCUSSION AND FUTURE WORK

In this chapter, we conclude the dissertation by outlining potential next steps and discussing the overall significance of the work.

7.1 THE HAND-HELD FORCE MAGNIFIER

From our work, we are confident that HHFM prototypes induce significant benefits to the human sensorimotor loop. Yet, the feedback provided to the user is simple: feedback forces are only some multiple of the distal force. This makes the device easy to understand and use, but one consequence of this type of feedback is that large transients are seen following ubiquitous events such as membrane puncture. Under such simple implementation of *in-situ* magnification, membrane puncture may actually be made less safe as we increase the perceived stiffness of the membrane, thereby increasing the distance traveled post-puncture.

Several options are available to counteract this possibility. Magnification is currently accessed by using a foot switch, and surgeons can simply choose to forego magnification during the initial puncture. Another possibility is that some internal intelligence recognizes the characteristic rise and sharp fall in the distal force associated with membrane puncture. Rather than simply magnify the force, upon correctly identifying these signal features, the system would deliver a short vibrotactile cue to indicate to the user that they had passed through the membrane.

A final alternative is to magnify the forces throughout the passage of the membrane, but rather than allowing the feedback force to drop precipitously following puncture, the system could gradually reduce the feedback force over longer periods of time. As we know from our membrane puncture work, motor control follows exponential decay in force in the immediate period following puncture. A similar form of decay in the feedback force may work well to reduce the distance traveled post-puncture. In this paradigm, the user would benefit from faster reaction times due to the stronger afferent signal of the magnified membrane, while at the same time avoiding the large transients at puncture.

Interaction between the HHFM and other medical technologies in the operating room further expand the possibilities for computer assisted surgery. For example, using ultrasound or optical coherence tomography (OCT) images to drive feedback force can potentially replace the internal force sensor, simplifying the mechanical design of the tool. Image segmentation software would produce a measure of the strain resulting from force applied to the tissue, which could in turn drive the feedback force. This may be especially important in membrane puncture applications, where these images could be used to signal insipient puncture. Tracking the tool tip in the medical image also presents opportunities to create virtual fixtures: artificial barriers that use force feedback to prevent the user from reaching locations or contacting structures that may be dangerous. Virtual fixtures would, for example, be helpful in preventing "back-walling" in vascular surgery.

Sensorimotor benefits were seen with *in-situ* force feedback in only the axial direction, though we know that real surgeries are performed over multiple DoF. An intriguing idea is to investigate how these benefits translate to *in-vitro* and *ex-vivo* surgical analogues. Membrane peeling may be practiced using strips of adhesive tape, or the inner membrane of a chicken egg

shell, for example [115], [173]. Presenting only a substantial subset of the available information is a valuable technique in designing assistive technologies, and single DoF force magnification may be sufficient to improve performance in surgery. This would avoid the difficulties of assembling a complex multiple DoF HHFM, and at the same time make the HHFM more reliable.

As the HHFM works right now, control is "open-loop," meaning there is no sensor that measures the force generated by the feedback actuator. Instead the HHFM actuator is calibrated, and that calibration equation is used to precisely render the feedback force. This approach is justified because the actuator calibration equation does not change much over time, compared to the sensing calibration, which certainly changes over time due to material aging. Still, closed-loop control could offer some increased stability and reliability through PID control or other advanced techniques. Efforts to "close the loop" with the HHFM, however, is complicated due to the small size required of the feedback sensors. Feedback sensors to measure force directly would also need to be calibrated, equations for which would also change over time.

7.2 PSYCHOPHYSICS

The exploratory task in Experiment 6 (Low Force Matching, Section 4.2.3.1) briefly examines the complex interplay between visual and haptic feedback, an issue that will be crucial as augmented reality technologies are developed for use in the surgical theatre. Both visual and haptic feedback, for example, could potentially further lower the absolute force threshold past the improvements already seen from *in-situ* force feedback. The interplay between the totality of afferent information from our visual, haptic, and auditory senses, and the resultant efferent force generated by the human may be leveraged in interesting and surprising ways with these new robotic technologies.

Additional psychophysics experiments with the Minimum Contact task are warranted. In particular, we would repeat this experiment with medical professionals at different stages in their careers: from medical students, to surgical residents, fellows, and experienced attending surgeons. The motor control analysis performed in this experiment could potentially add to the growing literature that differentiates between medical practitioners at these various skill levels.

Indeed, the HHFM has been considered a potential surgical training tool, where students may use a high HHFM gain the first time they attempt a delicate surgery, gradually reducing the gain as they become more proficient. Identifying potential learning effects from first practicing on *in-vitro* simulators, such as the 1DoF haptic renderer, with the HHFM before progressing to a real surgery with traditional tools would also be of interest.

7.3 HAPTIC RENDERING

Difficulties persist in creating models of delicate membranes using our single-axis haptic renderer. Due to the limited processing power of the Analog Devices microprocessor, part of these difficulties may lie in the fact that more complex control is necessary to enhance stability. We know that the lower limit for realistic rendering is approximately 1 kHz, whereas the microprocessor can reliably actuate much faster, up to 5 kHz in the early HHFM prototypes. One possible way to render virtual environments more smoothly is to nest a fast PID loop within a slower loop. The "slow" loop updates the various set points in position or force as called for by the model, say, at 1 kHz. The "fast" PID loop would minimize the error against the set point, at 5 kHz. This nested control may help stabilize the interface between membrane and free space, for instance.

Haptic rendering may be useful for patients as well as surgeons. Accurate reproduction of simple sensorimotor tasks in an inexpensive single-axis haptic renderer would open up the possibility to using our modeling to diagnose or otherwise quantify motor control deficits, for example those related to Parkinson's disease. Due to the sensitivity and segmentation offered by our motor control model, it would be interesting to see how the parameters of our model change with disease states, or over time as a result of treatment.

Work is progressing in our laboratory to implement models of surface texture using the haptic renderer. Texture originates from small variations on a surface, and its perception will therefore depend on the force and velocity of travel along that surface. Simple models of texture may be represented as sinusoidal waves that depend on the travel position or velocity. Designs that place a linear potentiometer or miniature track pad on the speaker cone can therefore drive the motion in our 1DoF haptic renderer. The University of Pennsylvania has released an open-

source “Haptic Texture Toolkit” which synthesizes models of a variety of textures from accelerometer and force sensor data [174], [175]. As part of other work, I refactored the Haptic Texture Toolkit using the MLHD API, taking advantage of its much higher force capability compared to the Phantom Omni, for which the toolkit was originally programmed. In this implementation, stiff surfaces like marble or rock were perceived very well, even eliciting characteristic sounds of contact between the flotor and the virtual surface. Importing the data and rendering schemes from these models to the 1DoF haptic renderer may be a productive approach to rendering textures with our new platform, with some effort – the toolkit makes use of some C++ libraries that may not be easily available in C, much less ARM C – and may yield interesting psychophysics, given we are rendering textures with the absolute minimal degrees of freedom.

Work is also progressing on creating a new type of vibrotactile stimulator with the 1DoF haptic renderer, called the “Make-and-Break” tactor. The concept underlying this tactor is that the cutaneous sensors are extremely sensitive to the initiation and removal of contact. Therefore this stimulator will actively use force measurements to detect contact, adjusting the cone position such that contact is continually made and broken with the user. The Welch’s t-test contact detector developed as part of the Minimum Contact experiment may be applied to this stimulator prototype.

7.4 CONCLUSION

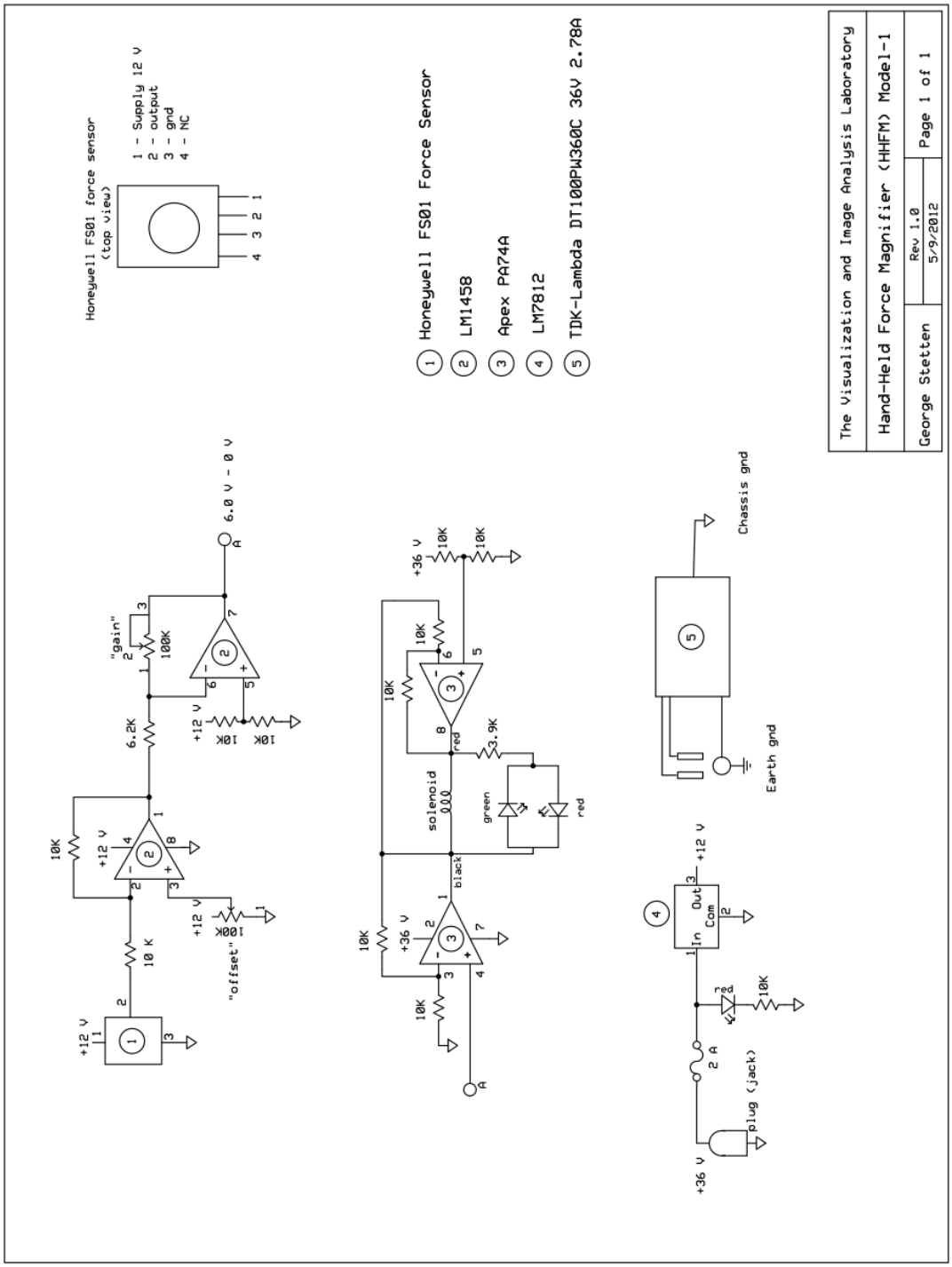
In this dissertation, we have presented the design and development of several hand-held surgical robots that augment the sense of touch. In using these prototypes, we have shown improvements to sensory and motor control abilities, beyond what is normally possible in humans. These improvements have been verified using the methods of psychophysics, advancing the state of surgical robotics on a firm scientific footing. Psychophysics generalizes and broadens the reach of *in-situ* force feedback: not only is this paradigm of force feedback useful in medicine and surgery, but it may well be useful in other applications.

APPENDIX A

HHFM MODEL-1 SYSTEM SCHEMATICS AND SOFTWARE

The HHFM Model-1 was driven by analog circuitry directly connected to the force sensor (Honeywell FS01).

A.1 SCHEMATICS

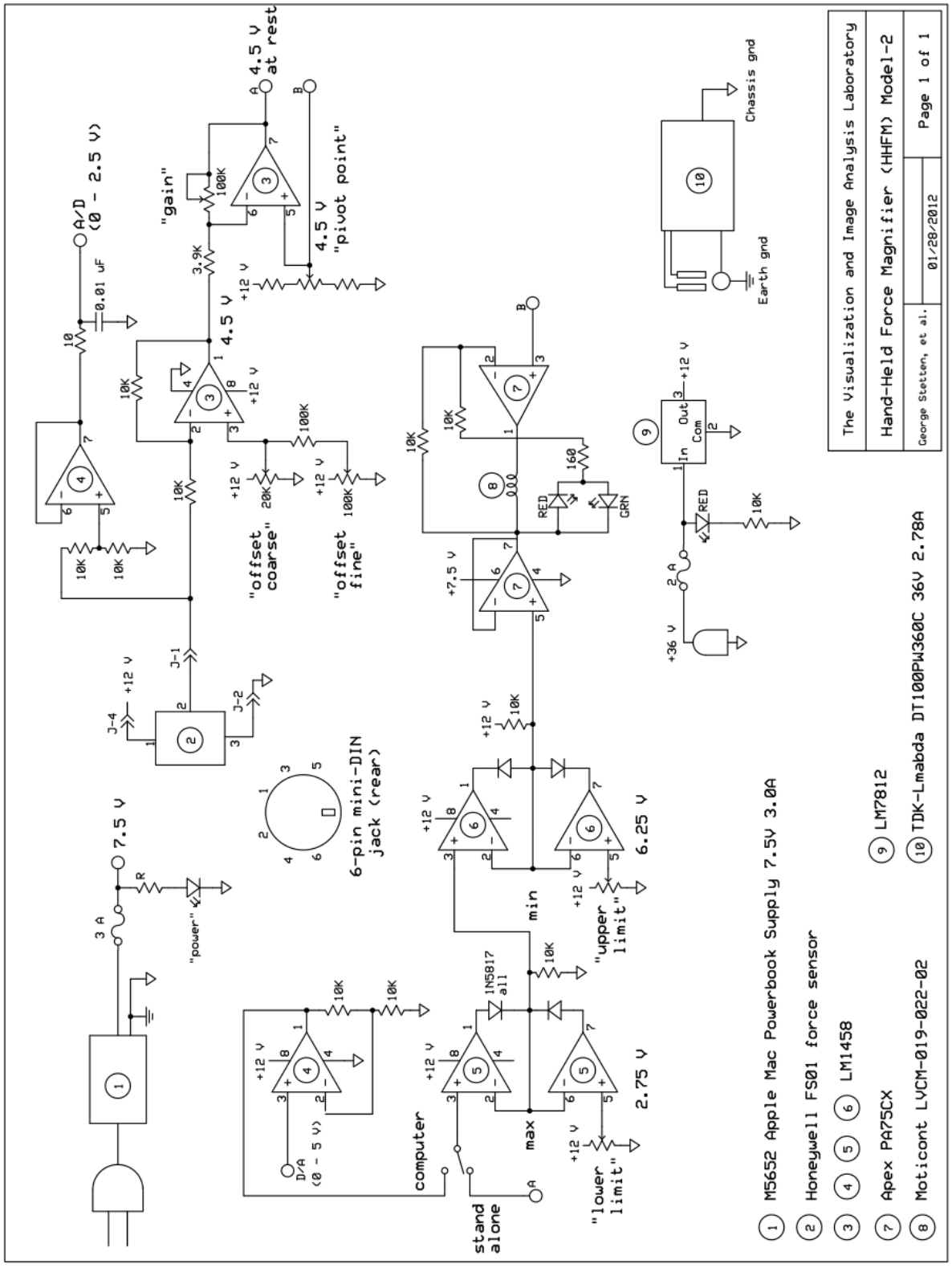


APPENDIX B

HHFM MODEL-2 SYSTEM SCHEMATICS AND SOFTWARE

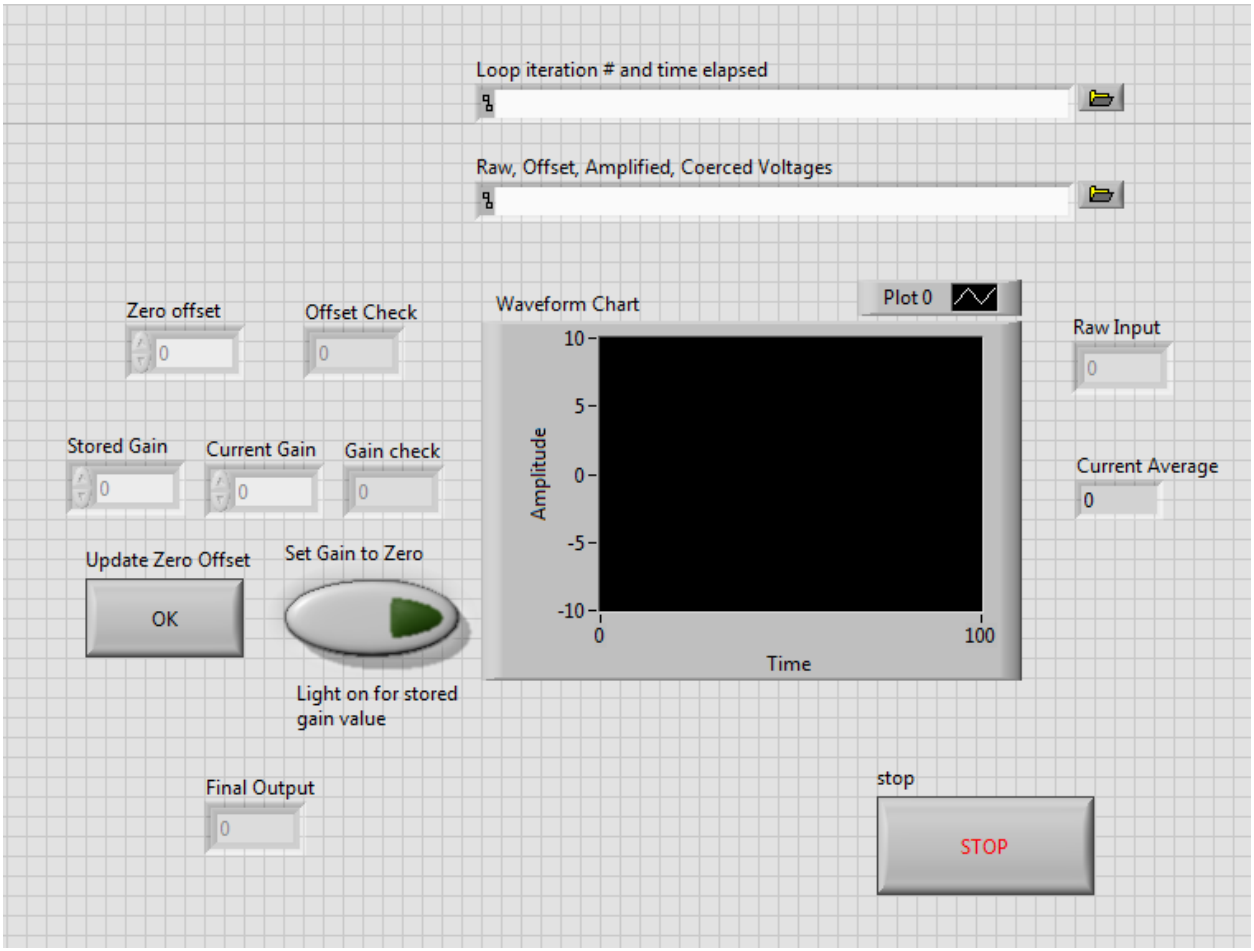
The HHFM Model-2 was powered with analog circuitry which was controlled with either a LabVIEW VI or an Analog Devices ADuC7026 microprocessor.

B.1 HHFM MODEL-2 SYSTEM SCHEMATICS

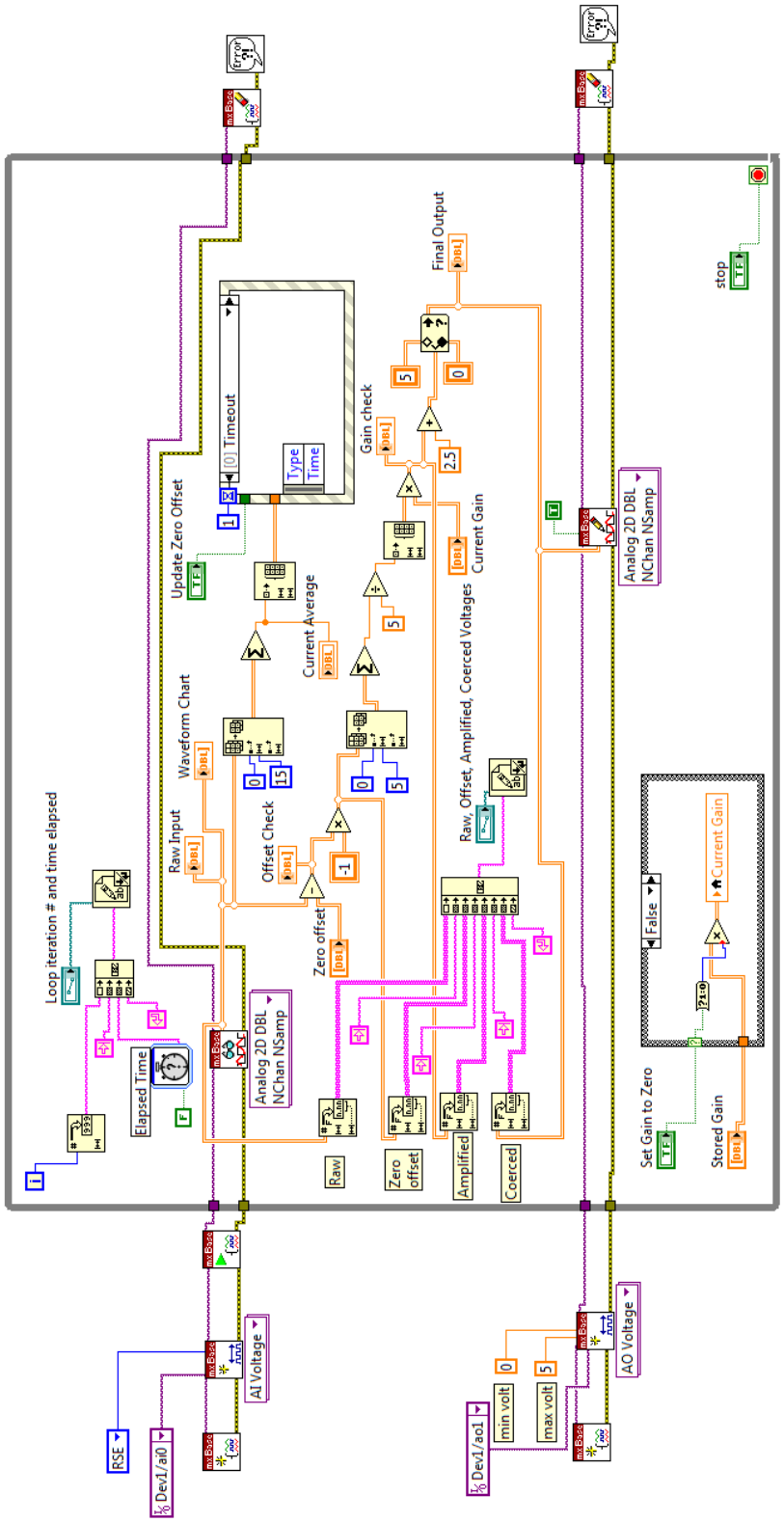


B.2 HHFM MODEL-2 LABVIEW VI

B.2.1 LabVIEW VI Front Panel



B.2.2 LabVIEW VI Back Panel



B.3 HHFM MODEL-2 SOURCE CODE

```
/*****************************************************************************
Randy Lee (ral63@pitt.edu)
VIA Lab
University of Pittsburgh
HHFM MCU Control

HHFM Model-2 (Shim Stock Cage)
Analog Devices ADUC7026
TotalPhase Aardvark I2C/SPI Host Adapter

Code version: v023

Description and Notes:
Analog force measurement taken from ADC0 channel on range 0-2.5V.
Signal subtracted from a constant which is calculated by taking
the average of a randomly sampled signal over one second. Zero-centered
signal obtained when device in use; multiplied by gain. Resultant DAC3
output centered around 1.5V, with rails at 2.08V and 0.92V (max displacement
from midpoint is 1.75/3 V).

Pin and Port Assignments:
GPIO P1.x -- I2C pins on .0/ .1
GPIO P2.x -- LED pins (not yet connected, 17 Dec 2012)
GPIO P3.x -- Front panel push buttons
        .1 = tare, .2 = mute, .3 = calibration
        .4 = ADUC7026 board reset

DAC1,DAC2 -- N/A
DAC3 -- Solenoid output

ADC0 -- Sensor measurement
ADC5 -- Gain knob measurement
ADC6, ADC7 -- N/A

timer0 -- 5/10 kHZ sampling rate regulation
timer1 -- us/ms delay functions
timer2 -- time since last tare (NOT YET IMPLEMENTED)

*****************************************************************************/
#include <aduc7026.h>
#include <math.h>
#include <stdio.h>

#define TRUE 1;
#define FALSE 0;
#define SLAVE_ADDRESS 0xA0;   // I2C slave address

/* Global Variable Declarations */
volatile unsigned char timer0_Flag, timer1_Flag = 0;
volatile unsigned char oneSecTare_Flag = 0;
//volatile unsigned char SPI_Tx_Flag = 0;
volatile unsigned char I2C_Tx_Flag = 0;
volatile unsigned char I2C_Tx_FlushFlag = 0;
volatile unsigned char randomSampleFlag = 0;
volatile unsigned char char1, char2 = 0;

char data[5] = 0;

int randomVoltageArrayCounter, randomWaitCounter, timeToWait, i;
/* Variables to hold raw sensor/actuator voltage char data */
int sensorVoltage_CHAR;
int transformedVoltage_CHAR;

/* Set of 100 random numbers, drawn from uniform distribution from 1 to 200,
 * the number of 100 us periods of wait between sensor sampling, to avoid
 * aliasing.
 * Avg = 109.24, obtained 14 Feb 2012 @ 2:05pm from random.org
 */
int randomDelayTimes[] = {132,163,186,124,177,7,91,112,122,102,145,63,
                          188,130,15,34,129,45,149,170,125,111,101,180,
                          153,140,53,125,171,166,36,57,68,128,72,95,140,
                          79,140,185,45,124,129,144,75,152,160,153,62,29,
                          1,40,133,157,84,68,139,157,24,192,12,178,24,92,
                          73,1,56,120,8,113,63,34,127,169,71,191,133,60,
                          139,193,141,139,184,51,167,154,175,80,176,123,
                          41,95,152,26,122,155,29,131,165,84};

/* Full scale "electronic gain */
float fullscale_gain = 60.00;
float midpoint = 1.50;
float sensor_voltage, transformed_voltage, offset_voltage, gain;
float offset_voltage_1s, offset_voltage_quick;
float storedRandomSamples[100];
float num1,num2;

/* Function Definitions */
/* converts hex ADC conversion result to workable decimal voltage */
```

```
float ADCinput(int ADC_hex_result);

/* converts voltage from potentiometer to a numerical gain */
float calculatePOTvoltage(int hex_POT_voltage, float fullscale_value);

/*  Converts desired decimal voltage to hex integer ready for DAC output */
int outputDAC(float desired_voltage);

/* Fetches state from switch on digital I/O port.pin; pushed state returns 1*/
int getSwitchSTATE(int port_number, int pin_number);

/* Converts num to unsigned char using known maximum */
unsigned char numToChar(float num, float max);

/* System Initialization */
void ADCpoweron(int);
void system_init(void);
void timer0_init(void);
void timer2_init(void);
void randomSample_init(void);
void I2C_init(void);

/* Timer functions */
void delay_sw(int time);
void delay_us(unsigned long time);
void delay_ms(unsigned long time);

/* Wait function to regulate sampling/actuation rate */
void waitForRestOfPeriod(void);

/* HHFM Functions */
void magnifyForces(void);
void tareRoutine(void);
//void autoTare(void);
void quickTare(void);
void storeSensorVoltageRandomly(void);
float calculateMeanFromStoredRandomSamples(void);
void calibrationProcedure(void);
void I2C_Tx(unsigned char data, unsigned char data2);

/* Interrupt routine for hardware timer; sets global timer flags */
void General_IRQ_Handler(void);

int main(void)
{
  /* Core clock and power state control
   * Clock set to 41.78 MHz, CD = 0, Vref = 2.5V,
   * DAC3 = 0-2.5V range, GPIO P2.x = output, GPIO P3.x = input
   */
  system_init();

  /* Power on ADC and DAC channels */
  ADCpoweron(20000);

  /* Enable timer0 and I2C Master interrupts, specify handler */
  IRQEN = 0x404;
  IRQ = General_IRQ_Handler;

  /* I2C Interface Initialization (NOT YET IMPLEMENTED) */
  I2C_init();           // I2C on GPIO P1; 400kHz bit rate
  //timer2_init();          // timer2 to see if recently tared

  //num1 = 50.0;
  //num2 = 50.0;

  while(1)
  {
    /* Start 200us timer on timer0 */
    timer0_init();

    /* Tare routine on P3.1 switch push */
    if(getSwitchSTATE(3,1) == 1)
    {
      tareRoutine();      /* Resets timer0 to 100us period for tare */
      timer0_init();      /* Reset timer0 to 200us period */
    }

    /* Active routine on P3.2 switch push */
    if(getSwitchSTATE(3,2) == 1)
    {
      delay_us(50);     /* to prevent contact bounce */
      quickTare();
      while(getSwitchSTATE(3,2) == 1)
      {
        magnifyForces();
        waitForRestOfPeriod();
      }
    }

    /* Step calibration function on P3.3 switch push */
    if(getSwitchSTATE(3,3) == 1)
    {
      calibrationProcedure();
    }
```

```
    /* I2C testing
    char1 = numToChar(num1,255.0);
    char2 = numToChar(num2,255.0);

    I2C_init();
    I2C_Tx(0xAA, 0x33);
    delay_us(10);
    I2C_Tx(char1, char2);
    I2C_Tx(0xBB, char1);

    num1++;
    num2++;
    */

    /* Default behavior = muted, if not switches pressed */
    DAC3DAT = outputDAC(1.50);

    /* Hardware-timed wait to pace out while loop at 5kHz (period = 200us) */
    waitForRestOfPeriod();
  }
}

void system_init(void)
{
  /* Configure CPU clock for 41.78 MHz, CD = 0 */
  POWKEY1 = 0x01;
  POWCON = 0x00;
  POWKEY2 = 0xF4;

  PLLKEY1 = 0xAA;
  PLLCON = 0x01;
  PLLKEY2 = 0x55;

  /* Connect internal 2.5V reference to Vref pin
   * Set DAC3 output to 0-Vref range and turn DAC3 on
   * Set digital I/O Port 2 as GPIO
   * Set all pins on Port 2 as output
   * Set digital I/O Port 3 as GPIO
   * Set all pins on Port 3 as input
   */
  REFCON = 0x01;
  DAC3CON = 0x12;
  GP2CON = 0x00;
  GP2DAT = 0xFF000000;
  GP3CON = 0x00;
  GP3DAT = 0x00;

  return;
}

void timer0_init(void)
{
  /* timer0 set to 200us period */
  timer0_Flag = 0;

  /* Re-initialize timer IRQ */
  IRQEN |= 0x04;
  /* Load timer0 counter with countdown value, given know operating freq.
   * 0x1053 = 100us period at 41.78MHz clock freq; 0x20A6 for 200us;
   * x30F9 for 300us
   */
  T0LD = 0x20A6;

  /* enable timer0, set in periodic mode with multiplier = 1 */
  T0CON = 0xC0;

  return;
}

void timer2_init(void)
{
  /* timer2 to see if tare happened recently (NOT IMPLEMENTED) */
  IRQEN |= 0x10;
  T2LD = 0xF00;       /* F00 = 15s; 1E00 = 30s; 20000 = 2 min; */

  /* 32kHz int. oscillator, HH:MM:SS format, periodic counting down */
  T2CON = 0xE8;

  return;
}

void randomSample_init(void)
{
  /* Function called in tareRoutine() */

  /* re-initialize counters/flags in case tare happens in the middle of loop*/
  randomWaitCounter = 0;
  randomVoltageArrayCounter = 0;
  randomSampleFlag = 0;
  timeToWait = randomDelayTimes[0];   /* Load first random delay time */


  /* Re-initialize timer0 to 100us sampling period for tare */
  timer0_Flag = 0;
  IRQEN |= 0x04;
```

```
  T0LD = 0x1053;      /* 0x1053 = 100us period at 41.78MHz clock freq */
  T0CON = 0xC0;       /* enable timer0, periodic mode w/ multiplier = 1 */

  return;
}

void I2C_init(void)
{
  /* Set P1.0 & P1.1 for I2C communication */
  GP1CON = 0x22;
  I2C0CFG = 0x82;       /* Set ADUC is I2C master device */
  //I2C0CFG = 0xC2;       /* Set ADUC as I2C master with loopback */

  /* Set I2C clock frequency
   * 0x283C for 400kHz bit rate; 0xCFCF for 100 kHz */
  I2C0DIV = 0x283C;
  return;
}

void I2C_Tx(unsigned char data, unsigned char data2)
{
  I2C0MTX = data;
  I2C0MTX = data2;
  I2C0ADR = 0x00;       // set I2C address (LSB = 0, Master write)
  return;
}

unsigned char numToChar(float num, float max)
{
  unsigned char result;
  result = floor((num*0xFF)/max);
  return result;
}

void ADCpoweron(int time)
{
  ADCCON = 0x20;          /* power-on the ADC */
  delay_sw(time);         /* software wait for ADC to be fully powered on */
  return;
}

void General_IRQ_Handler(void)
{
  if ((IRQSIG & 0x04) == 0x4)
  {
    /* timer0 IRQ clear routine */
    timer0_Flag = 1;
    T0CLRI = 0xFF;
    return;
  }

  if ((IRQSIG & 0x8) == 0x8)
  {
    /*  timer1 IRQ clear for us/ms delay */
    timer1_Flag = 1;
    T1CLRI = 0xFF;
    return;
  }

  if ((IRQSIG & 0x10) == 0x10)
  {
    /* timer2 IRQ clear for tare functions */
    //DAC3DAT = outputDAC(2.50);
    //delay_us(10);
    oneSecTare_Flag = 0;
    T2CLRI = 0xFF;
    return;
  }

  if((I2C0MSTA & 0x4) == 0x4)
  {
    /* Master I2C transmit IRQ */
    return;
  }

  if((I2C0MSTA & 0x8) == 0x8)
  {
    /* Master Recieve IRQ */
    return;
  }
}

void quickTare(void)
{
  /* Takes 5 measurements of sensor voltage to roughly tare system */
  int counter = 0;
  float currentSensorVoltage, sum = 0;

  for(counter = 0; counter < 5; counter++)
  {
    ADCCP = 0x00;
    ADCCON = 0x6A3;
    ADCCON &= ~(1 << 7);
    while(!ADCSTA){}
    currentSensorVoltage = ADCinput(ADCDAT);
```

```
    sum += currentSensorVoltage;
  }
  offset_voltage = sum/5.0;
  return;
}

void tareRoutine(void)
{
  /* sample from ADC5 for gain knob calculation */
  ADCCP = 0x05;
  ADCCON = 0x6A3;
  ADCCON &= ~(1 << 7);
  while(!ADCSTA){}
  gain = calculatePOTvoltage(ADCDAT, fullscale_gain);

  /* re-initialize variables and 100us timer0 for tare, set P2.x as output */
  randomSample_init();
  GP2DAT = 0xFF000000;

  while(randomSampleFlag == 0)
  {
    /* mute output during tare */
    DAC3DAT = outputDAC(1.50);
    GP2SET = 0x2000;          /* Turn on P2.1 LED to indicate tare */

    /* sample ADC0, HHFM Force Sensor */
    ADCCP = 0x00;
    ADCCON = 0x6A3;
    ADCCON &= ~(1 << 7);
    while (!ADCSTA){}
    sensor_voltage = ADCinput(ADCDAT);

    /* only store measurement if waited correct # of delay periods */
    storeSensorVoltageRandomly();

    /* hardware-timed wait to maintain 100us period */
    waitForRestOfPeriod();
  }
  offset_voltage_1s = calculateMeanFromStoredRandomSamples();
  GP2CLR = 0x2000;          /* turn off LED when done with tare */

  /* disable timer0, reset in timer0_init() */
  T0CON = 0x0;

  oneSecTare_Flag = 1;
  return;
}

int getSwitchSTATE(int port_number, int pin_number)
{
  int pinInputData, pinInputData_MASKED, pinBit;

  if (port_number == 0){pinInputData = GP0DAT;}
  else if (port_number == 1){pinInputData = GP1DAT;}
  else if (port_number == 2){pinInputData = GP2DAT;}
  else if (port_number == 3){pinInputData = GP3DAT;}
  else if (port_number == 4){pinInputData = GP4DAT;}

  /* Mask GPIO result to show only the corresponding pin number */
  pinBit = pow(2,pin_number);
  pinInputData_MASKED = (pinInputData &  pinBit);

  if (pinInputData_MASKED == 0)
  {
    /* bit LOW when pushed, HIGH when not pushed */
    return 1;
  }
  else
  {
    return 0;
  }
}

float ADCinput(int ADC_hex_result)
{
  float input_voltage;
  int ADC_hex_result_shifted;
  ADC_hex_result_shifted = ADC_hex_result >> 16;

  /* 2.52 is highest voltage on ADC when connected to voltmeter */
  input_voltage = ((ADC_hex_result_shifted)*2.52)/(0xFFF);
  return input_voltage;
}

int outputDAC(float desired_voltage)
{
  /* "unit conversion" from desired decimal voltage to hex code;
   * 2.4976 is voltage when 0xFFF applied to DAC
   */
  int output_integer = floor((desired_voltage*0xFFF)/2.497);

  if (output_integer <= 0xFFF && output_integer >= 0x200)
  {
    /* shift integer result 16 bits to left to conform to DACxDAT format
```

```
     * result subtracted from small integer as rough calibration to measured
     * values; bounded by maximum and minimum integer caps
     */
    output_integer = (output_integer - 0x2) << 16;
  }

  else if (output_integer > 0xFFF)
  {
    /* maximum integer cap */
    output_integer = 0xFFF;
    output_integer = output_integer << 16;
  }
  else
  {
    /* minimum integer cap */
    output_integer = 0x200;
    output_integer = output_integer << 16;
  }
  return output_integer;
}

void magnifyForces(void)
{
  /* select ADC0 for HHFM Force Sensor measurement */
  ADCCP = 0x00;
  ADCCON = 0x6A3;         /* software conv., single-ended, conv. enabled */
  ADCCON &= ~(1 << 7);    /* clear bit7 to stop additional conversions */
  while (!ADCSTA){
    /* wait for end of conversion */
  }
  /* convert ADC result to decimal voltage */
  sensor_voltage = ADCinput(ADCDAT);

  transformed_voltage = (sensor_voltage - offset_voltage)*gain + midpoint;
  DAC3DAT = outputDAC(transformed_voltage);
}

void waitForRestOfPeriod(void)
{
  if (timer0_Flag == 1)
  {
    timer0_Flag = 0;
    return;
  }
  else
  {
    while(timer0_Flag == 0){
      /* wait until interrupt flips timer0 flag */
    }
    timer0_Flag = 0;
    return;
  }
}

void delay_sw(int time)       // software delay. i = 10 --> ~12 us
{
  while (time >=0)
  time --;
}

void delay_us(unsigned long time)
{
  timer1_Flag = 0;
  T1CON = 0x0;

  /* enable timer1 IRQ, timer1 = bit3 */
  IRQEN |= 0x08;

  /* T1LD = time (us) * freq (MHz)
   * (e.g. 500us * 41.78 MHz = 20890-1) subtract 1 since includes zero
   */
  T1LD = ((time * 42782) >> 10) - 1;

  /* init timer1, 41.78MHz, down mode, periodic */
  T1CON = 0xC0;
  while(timer1_Flag == 0){
    /* wait for timer1 flag flip in IRQ handler */
  }
  timer1_Flag = 0;
  T1CON = 0x0;
  IRQCLR |= 0x08;

  return;
}

void delay_ms(unsigned long time)
{
  unsigned short i = 0;
  for(i = 0; i < time; i++)
  {
    /* wait for 1 ms each time through loop */
    delay_us(1000);
  }
}
```

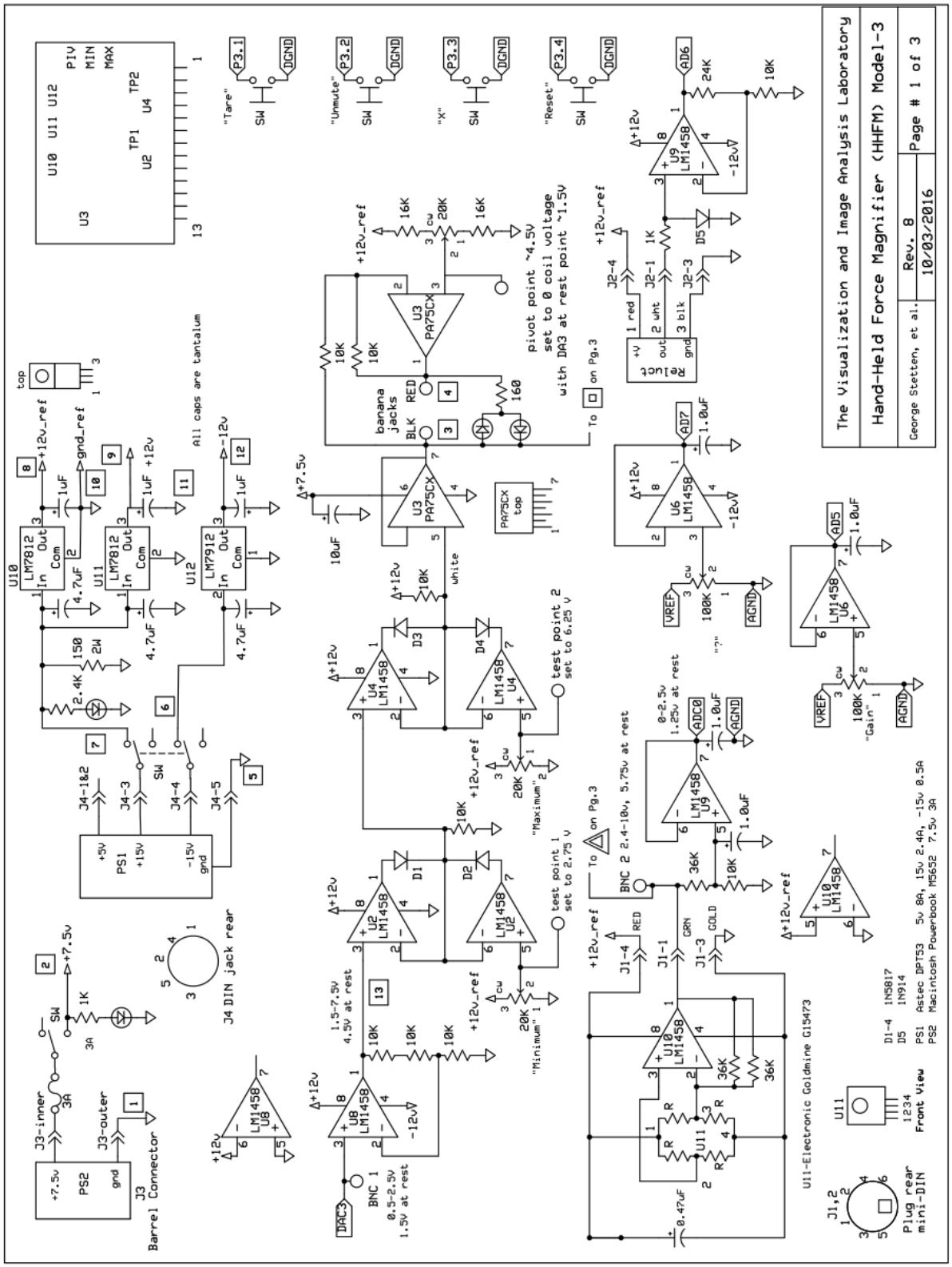
```
float calculatePOTvoltage(int POT_voltage, float fullscale_value)
{
  float calculatedVoltage;
  int POT_voltage_shifted = POT_voltage >> 16;
  calculatedVoltage = (POT_voltage_shifted / (float) 0xFFF) * (fullscale_value);
  return calculatedVoltage;
}

void storeSensorVoltageRandomly(void)
{
  if (randomWaitCounter >= timeToWait)
  {
    /* store sensor voltage if counter reached time to wait */
    storedRandomSamples[randomVoltageArrayCounter] = sensor_voltage;
    randomWaitCounter = 0;
    randomVoltageArrayCounter++;
    if (randomVoltageArrayCounter >= 100)
    {
      /* all 100 samples taken, flip flag and reset array counter */
      randomVoltageArrayCounter = 0;
      randomSampleFlag = 1;
    }
    /* load next time to wait from array */
    timeToWait = randomDelayTimes[randomVoltageArrayCounter];
  }
  else
  {
    randomWaitCounter++;
  }
}

float calculateMeanFromStoredRandomSamples(void)
{
  int i;
  float sum = 0.0;
  float mean = 0.0;
  for(i = 0; i < 100; i++)
  {
    sum += storedRandomSamples[i];
  }
  mean = sum/100.0;
  return mean;
}

void calibrationProcedure(void)
{
  float appliedVoltage = 1.50;
  float voltageStep = 0.05;

  while(1)
  {
    /* tare button forces positive voltage step (pull direction) */
    if(getSwitchSTATE(3,1) == 1) voltageStep = 0.05;

    /* mute button forces negative step (push direction) */
    else if(getSwitchSTATE(3,2) == 1) voltageStep = -0.05;

    /* X button to actuate the voltage change */
    else if(getSwitchSTATE(3,3) == 1)
    {
      /* timer delay to prevent contact bounce */
      delay_ms(500);

      appliedVoltage += voltageStep;

      if(appliedVoltage >= 2.50)
      {
        DAC3DAT = outputDAC(2.50);
        voltageStep = -0.05;
      }
      else if(appliedVoltage <= 0.0)
      {
        DAC3DAT = outputDAC(0.0);
        voltageStep = 0.05;
      }
    }
    else
    {
      DAC3DAT = outputDAC(appliedVoltage);
      waitForRestOfPeriod();
    }
  }
}
/* EOF ------------------------------------------------------------------- */
```

APPENDIX C

HHFM MODEL-3 SYSTEM SCHEMATICS AND SOFTWARE

The HHFM Model-3 was powered with analog circuitry and controlled by an Analog Devices ADuC7026 microprocessor. A secondary control box was added to control the device for psychophysics experiments.

C.1 HHFM MODEL-3 SYSTEM SCHEMATICS



C.2 HHFM MODEL-3 CONTROL SOFTWARE

```
/*****************************************************************************
 *  Randy Lee (ral63@pitt.edu)
 *  Visualization and Image Analysis Laboratory
 *  Department of Bioengineering
 *  University of Pittsburgh
 *
 *  HHFM MCU Control, for Low Force Experiments
 *
 *  HHFM Model-3 (SLA Direct Contact)
 *  Analog Devices ADuC7026
 *
 *  Code version: v16
 *
 *  Description and Notes:
 *      Analog force measurement taken from ADC0 channel on range 0-2.5V.
 *      Signal subtracted from a constant which is calculated by taking
 *  the average of a randomly sampled signal over one second. Zero-centered
 *  signal obtained when device in use; multiplied by gain. Resultant DAC3
 *  output centered around 1.5V, with rails at 2.08V and 0.92V (max
 *  displacement from midpoint is 1.75/3 V).
 *
 *  Pin and Port Assignments:
 *  GPIO P0.x -- Digital input from 'Experimental ADuC7026'
 *              .0 = currentlyInTrial (1 = before/after trial; 0 = in trial)
 *              .1 = magnification state bit0
 *              .2 = magnification state bit1
 *                  --> (bit0,bit1 -- 00 = 0x; 01 = 3x; 10 = 10x; 11 = 7x)
 *  GPIO P1.x -- I2C Tx/Rx pins on .0/ .1
 *  GPIO P2.x -- LED pins
 *              .0 = Yellow (NC), .1 = Red, .2 = Green (NC)
 *  GPIO P3.x -- Front panel push buttons
 *              .1 = tare, .2 = unmute/foot pedal,
 *              .3 = calibration/experiment, .4 = ADUC7026 board reset
 *  GPIO P4.x -- NC
 *
 *  DAC1,DAC2 -- NC
 *  DAC3 -- Solenoid output
 *
 *  ADC 0 -- Force sensor
 *  ADC 5 -- Force sensor gain knob
 *
 *  timer0 -- 5/10 kHz sampling rate regulation, init in timer0_init()
 *  timer1 -- us/ms delay functions, init in delay_us()
 *  timer2 -- time since last tare, init in timer2_init(); not implemented
 *
 * Last Updated 11 October 2016
 *****************************************************************************/

/* Libraries ------------------------------------------------------------- */
#include <aduc7026.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/* Defines --------------------------------------------------------------- */
#define TEMP_DISABLED 0
#define OBSOLETE 0
#define NOT_YET_IMPLEMENTED 0

/* I2C implementation defines */
#define SLAVE_ADDRESS 0xA0
#define FIFO_BAUDRATE 9600

#define TRUE 1
#define FALSE 0
#define DECIMAL 1
#define HEX 0

/* Global variables ------------------------------------------------------ */
volatile unsigned int timer0_Flag, timer1_Flag = 0;
volatile unsigned int oneSecTare_Flag = 0;
volatile unsigned int randomSampleFlag = 0;
volatile unsigned char char1, char2 = 0;

/* char data[5] = 0; */

int randomVoltageArrayCounter, randomWaitCounter, timeToWait, i;
float g_epsilon = 0.05;

/* variables to hold raw sensor/ actuator voltage char data */
int sensorVoltage_CHAR;
int transformedVoltage_CHAR;

/* set of 100 numbers drawn from a uniform distribution from 1 to 200,
 * avg = 109.24 -- Obtained 14 Feb 2012 @ 2:05pm from random.org
 */
```

```
int randomDelayTimes[] = {132,163,186,124,177,7,91,112,122,102,145,63,
                          188,130,15,34,129,45,149,170,125,111,101,180,
                          153,140,53,125,171,166,36,57,68,128,72,95,140,
                          79,140,185,45,124,129,144,75,152,160,153,62,29,
                          1,40,133,157,84,68,139,157,24,192,12,178,24,92,
                          73,1,56,120,8,113,63,34,127,169,71,191,133,60,
                          139,193,141,139,184,51,167,154,175,80,176,123,
                          41,95,152,26,122,155,29,131,165,84};

float storedRandomSamples_Force[100], storedRandomSamples_Accel[100];
float num1, num2;

/* struct of TYPE "offsets" to hold sensor/actuator offsets */
struct offsets{
    volatile float Force;
    volatile float Accel;
    volatile float Solenoid;
    };

/* struct of TYPE "gains" to hold sensor/actuator gains */
struct gains{
    volatile float Force;
    volatile float Accel;
    float FullscaleForce;
    float FullscaleAccel;
    };

/* struct of TYPE "voltages" to hold sensor/actuator hex&decimal voltages */
struct voltages{
    volatile float Force;
    volatile float Accel;
    volatile float Solenoid;
    volatile char ForceHex;
    volatile char AccelHex;
    volatile char SolenoidHex;
    };

#if OBSOLETE
/* struct of TYPE "voltageHex" to hold hex voltages */
struct voltageHex{
    volatile float Force;
    volatile float Accel;
    volatile float Output;
}

/* struct of TYPE "voltage history" holding ARRAYS of previous voltages */
struct voltageHistory{
    float Force;
    float Accel;
    float Output;
    };
#endif

struct offsets Offset, *pO;
struct gains Gain, *pG;
struct voltages Voltage, *pV;

/* Functions ---------------------------------------------------------------- */
/* HARDWARE INTERFACE FUNCTIONS */
/* converts hex ADC conversion result to decimal voltage */
float ADCinput(int ADC_hex_result);

/* converts voltage from potentiometer to a number that is a fraction of
 *  fullscaleValue
 */
float potVoltage(float voltage, float fullscaleValue);

/* reads sensor voltage from ADC channel # */
float readSensors(int channel);

/* converts desired decimal voltage output to hex integer */
int outputDAC(float desired_voltage);

/* fetches state from push switch on GPIO pin. Pushed state returns 1 */
int getSwitchSTATE(int port_number, int pin_number);

/* fetches state from an external trigger on a GPIO pin. HIGH state returns 1
 */
int getExtSwitchSTATE(int port_number, int pin_number);

/* @OBSOLETE: sends data to USBMOD5 via FIFO. Successful send returns 1 */
int sendData(int channel);

/* Quick tare -- 5 sample sensor zeroing. Returns pointer to offsets struct */
struct offsets *quickTare(struct voltages *, struct gains *,
                          struct offsets *);

/* Full 1 second tare -- 100 sample sensor zeroing. Returns ptr to offsets
 * struct
 */
struct offsets *tareRoutine(struct voltages *, struct gains *,
                            struct offsets *);

/* offset voltage update using mean of sotred sensor voltages */
struct offsets *calculateMeanFromStoredRandomSamples(struct offsets *);
```

```
/* converts number to unsigned char using known maximum */
unsigned char numToChar(float num, float max);

/* SYSTEM FUNCTIONS */
/* initialize ADuC7026 system -- ADC/DAC/GPIO P0,P2,P3 */
void system_init(void);

/* initialize ADC on ADuC7026 */
void ADCpoweron(int);

/* initilize timer0 on ADuC7026 */
void timer0_init(void);             // Initialize timer0 routine

/* @TODO: initialize timer2 for automatic tare */
//void timer2_init(void);

/* initialize random sampling variables & timer */
void randomSample_init(void);

/* software delay */
void delay_sw(int time);

/* us delay using timer1 */
void delay_us(unsigned long time);

/* ms delay using delay_us() */
void delay_ms(unsigned long time);

/* wait function to delay next loop until timer0 counts down */
void waitForRestOfPeriod(void);

/* @TODO: automatically re-tare sensor if minimal activity/ timer2 completes
 */
//void autoTare(void);

/* waits number of 100us loops equal to the element of randomDelayTimes to
 * store sensor voltage over 1 second
 */
void storeSensorVoltageRandomly(struct voltages *);

/* force magnification function */
void magnifyForces(struct voltages *, struct gains *, struct offsets *);

/* force magnification with external switches for gain */
void expMagnifyForces(struct voltages *, struct gains *, struct offsets *);

/* user actuated calibration using push button to increment force. Force
 * increment flips to other direction when max/min reached. Use SW reset on
 * P1.4 to escape
 */
void calibrationProcedure(void);

/* automatic calibration procedure, VCM goes through 2 cycles from 0-2.5V in
 *  0.02V steps
 */
void autoCalibrate(void);

/* function to generate specific force value */
void generateForce(float force, struct voltages *, struct gains *,
                   struct offsets *);

/* verify calibration by generating specified forces */
void verifyCalibration(void);

/* seperate experimental "mode" to control HHFM via external hardware */
void experimentMode(void);

/* interrupt handler for hardware timers, sets timer expiration flags */
void General_IRQ_Handler(void);

/* Start of Main-------------------------------------------------------------*/
int main(void)
{
    /* SW wait to avoid JTAG lock while prototyping */
    delay_sw(50000);

    /** Core clock and power state control **/
    /* clock set to 41.78MHz, CD = 0
     * Vref = 2.5V; DAC3 = 2.5V range
     * GPIO P0.x = input; GPIO P2.x = output; GPIO P3.x = input
     */
    system_init();

    /* Power on ADC */
    ADCpoweron(20000);

    /** IRQ Set-up **/
    /* enable timer0 interrupt */
    IRQEN = 0x404;

    /* specify interrupt handler */
    IRQ = General_IRQ_Handler;

    /** Struct, Variable, & Pointer Initialization **/
```

```
#if OBSOLETE
Gain.FullscaleForce = 60.0;
Gain.FullscaleAccel = 10.0;
#endif

/* offset updated from 10/2016 calibration - binned least squares */
Offset.Solenoid = 1.44759825;
Gain.Force = 5.0;
pV = &Voltage;
pG = &Gain;
pO = &Offset;

/* initial tare */
pO = tareRoutine(pV,pG,pO);

/* Main loop ------------------------------------------------------------- */
while (1)
{
    /* start 200us timer on timer0 */
    timer0_init();              //Start 200us timer on timer0

    /*** Tare routine on P3.1 switch push ***/
    if(getSwitchSTATE(3,1) == 1)
    {
        /* reset timer0 to 100us period for tare */
        pO = tareRoutine(pV,pG,pO);

        /* reset timer0 to 200us period after tare */
        timer0_init();
    }

    /*** Active force magnification routine on P3.2 switch push
     *** (via foot pedal)
     ***/
    else if(getSwitchSTATE(3,2) == 1)
    {
        /* short delay to prevent contact bounce */
        delay_us(50);


        /* 5 sample tare immediately before magnification, resets gain
         * and saves new offsets
         */
        //pO = quickTare(pV,pG,pO);

        while (getSwitchSTATE(3,2) == 1){
            magnifyForces(pV,pG,pO);
            waitForRestOfPeriod();
        }
    }

    /*** Stepwise calibration function on P3.3 switch push ***/
    else if(getSwitchSTATE(3,3) == 1)
    {
        /* seperate "mode" to increase/decrease coil in 50mV steps.
         * Actuated by push button press
         */
        //calibrationProcedure();

        /* seperate "mode" to increase/decrease coil in 20mV steps.
         * Automatically actuates 2 cycles between 0-2.5V
         */
        autoCalibrate();

        /* seperate "mode" to verify calibration by generating specified
         * forces
         */
        //verifyCalibration();
    }

    #if OBSOLETE
    /*** External control behavior on P0.1 (curr in trial) state change
     ***/
    else if(getExtSwitchSTATE(0,1) == 1)
    {
        delay_us(25);

        /* seperate mode that allows for HHFM control via external ADuC */
        experimentMode();
    }
    #endif

    /*** External control behavior on P0.0 (curr in trial) state change
     ***/
    else if(getSwitchSTATE(0,0) == 1)
    {
        delay_us(50);

        /* seperate mode that allows for HHFM control via external ADuC */
        experimentMode();
    }

    /*** Default behavior = no force on voice coil ***/
    else
    {
```

```
            /* "mute" the HHFM voice coil */
            DAC3DAT = outputDAC(1.50);

            /* hardware-timed wait to pace out while loop to 5kHz
             * (period = 200us)
             */
            waitForRestOfPeriod();
        }
    }
}
/* End of Main---------------------------------------------------------------- */


/* Functions------------------------------------------------------------------ */

/** General_IRQ_Handler() --
 **     Case structure to handle IRQ interrupts
 **/
void General_IRQ_Handler(void)
{
    /* timer0 IRQ clear routine */
    if ((IRQSIG & 0x04) == 0x4)
    {
        timer0_Flag = 1;
        T0CLRI = 0xFF;
        return;
    }

    /* timer1 IRQ clear for us/ms delay */
    if ((IRQSIG & 0x8) == 0x8)
    {
        timer1_Flag = 1;
        T1CLRI = 0xFF;
        return;
    }

    #if NOT_YET_IMPLEMENTED
    /* timer2 IRQ clear for automatic tare functions */
    if ((IRQSIG & 0x10) == 0x10)
    {
        //DAC3DAT = outputDAC(2.50);
        //delay_us(10);
        oneSecTare_Flag = 0;
        T2CLRI = 0xFF;
        return ;
    }
    #endif
}

/** tareRoutine() --
 **     Records and averages sensor readings over 1 second. Sets gain.
 **     Returns pointer to Offset struct
 **/
struct offsets *tareRoutine(struct voltages *pv, struct gains *pg,
                            struct offsets *po)
{
    float voltage;

    /* mute HHFM actuator during tare routine */
    DAC3DAT = outputDAC(1.50);

    #if OBSOLETE
    /* force gain on ADC5 (potentiometer knob) */
    //voltage = readSensors(5);     //Force gain on ADC5 (pot knob)
    //(*pg).Force = potVoltage(voltage, (*pg).FullscaleForce);

    /* force gain set as 1.8x, to mirror HHFM Model-1 experiments */
    //(*pg).Force = 10.80;

    /* perceptual gain: Kp = 4, so actuator gain Ka = Kp - 1 = 3 */
    //(*pg).Force = 3.00;
    #endif

    delay_us(10);

    #if OBSOLETE
    //voltage = readSensors(7);         //Accel gain on ADC7
    //(*pg).Accel = potVoltage(voltage, (*pg).FullscaleAccel);
    #endif

    /* re-initialize variables and 100us timer0 for tare over 1 second */
    randomSample_init();

    /* re-initialize P2.x as output */
    GP2DAT = 0xFF000000;

    while(randomSampleFlag == 0)
    {
        /* flip P2.0 HIGH to note currently calculating tare, turn on LED */
        GP2DAT |= 0x10000;

        /* force sensor on ADC0 */
        (*pv).Force = readSensors(0);
        delay_us(10);
```

```
        #if NOT_YET_IMPLEMENTED
        /* accelerometer on ADC10 */
        (*pv).Accel = readSensors(10);
        #endif

        /* only store if # of loops waited is equal to current element in
         * randomDelayTimes
         */
        storeSensorVoltageRandomly(pv);

        /* wait out rest of current 100us period */
        waitForRestOfPeriod();
    }
    po = calculateMeanFromStoredRandomSamples(po);

    /* flip P2.0 LOW, turn off LED */
    GP2DAT ^= 0x10000;

    /* temporarily disable timer0. Reset with timer0_init() */
    T0CON = 0x0;

    //oneSecTare_Flag = 1;
    return po;
}

/** quickTare() --
 **     Records and averages 5 successive sensor readings.
 **     Returns pointer to offsets.
 **/
struct offsets *quickTare(struct voltages *pv, struct gains *pg,
                          struct offsets *po)
{
    int counter, doneFlag = 0;
    float voltage, sumF, sumA = 0.0;

    DAC3DAT = outputDAC(1.50);

    while (doneFlag == 0){
        #if OBSOLETE
                                        /* force gain knob on ADC5 */
        //voltage = readSensors(5);

        //(*pg).Force = potVoltage(voltage, (*pg).FullscaleForce);

        /* gain = 1.8x to mirror HHFM Model-1 experiments */
        //(*pg).Force = 10.80;
        #endif

        delay_us(10);

        #if NOT_YET_IMPLEMENTED
        //voltage = readSensors(7);             //Accel gain on ADC7
        //(*pg).Accel = potVoltage(voltage, (*pg).FullscaleAccel);
        #endif

        for (counter = 0; counter < 5; counter++){
            /* sample from force sensor */
            (*pv).Force = readSensors(0);       //Sample from force sensor

            delay_us(10);

            #if NOT_YET_IMPLEMENTED
            /* sample from accelerometer */
            //(*pv).Accel = readSensors(10);    //Sample from accelerometer
            //sumA += (*pv).Accel;
            #endif

            sumF += (*pv).Force;

        }
        (*po).Force = sumF/5.0;
        //(*po).Accel = sumA/5.0;

        doneFlag = 1;
    }
    return po;
}

/** readSensors() --
 **     Initialize ADC transfer from specified channel and return decimal
 **     voltage
 **/
float readSensors(int channel)
{
    float sensorVoltage;

    ADCCP = channel;
    ADCCON = 0x6A3;

    /* clear bit 7 to stop additional conversions */
    ADCCON &= ~(1 << 7);

    while (!ADCSTA){
        /* wait until conversion completes */
    }
```

```
    sensorVoltage = ADCinput(ADCDAT);

    return sensorVoltage;
}

/** magnifyForces() --
 **     Routine to generate feedback forces using gathered voltage, gain,
 **     and offset data
 **/
void magnifyForces(struct voltages *pv, struct gains *pg, struct offsets *po)
{
    /*  Current Calibration Equations ----
     *  Force sensor (11 October 2016 -- binned least squares fit)
     *  ADC voltage --> force: y = 591.253227x -708.0596313
     *
     *  Voice Coil (11 October 2016)
     *  desired force --> DAC voltage: y = -0.008619688x + 1.44759825
     */

    float diffForceVoltage, sensorForceGrams, desiredForceSolenoid, knobVoltage;
    float curr_sensor_grams, offset_sensor_grams;

    /* set gain in software */
    (*pg).Force = 10.00;

    /* gain controlled by front panel knob */
    //knobVoltage = readSensors(5);
    //(*pg).Force = potVoltage(knobVoltage, 15.00);

    /** Force centered magnification: Measure force in grams from sensor
     ** using calibration eqns. Multiply by device gain to obtain desired
     ** force. Actuator voltage for this desired force obtained using
     ** calibration eqns.
     **/
    /* read force sensor on ADC0. readSensors() returns FLOAT */
    (*pv).Force = readSensors(0);

    /* new implementation (v014) -- calculate current and offset force
     * on sensor in grams, using calibration equations directly. then
     * subtract the two to get change.
     */
    curr_sensor_grams = 591.253227 * (pv->Force) - 708.0596313;
    offset_sensor_grams = 591.253227 * (po->Force) - 708.0596313;

    desiredForceSolenoid = (pg->Force) * (curr_sensor_grams - offset_sensor_grams);

    #if TEMP_DISABLED
    /* experimenting with previous voltage difference implementation */
    /* previous implementation (v013) -- calculate change in voltage,
     * then use that change to calculate the change in grams on
     * sensor
     */
    diffForceVoltage = (pv->Force) - (po->Force);

    /* calculate force in grams from voltage change, no intercept becuase
     * sensor offset @ 0 grams can drift, but slope should be the same
     */
    sensorForceGrams = 591.253227*(diffForceVoltage);

    /* desired force is the measured force multiplied by gain */
    desiredForceSolenoid = (pg->Force) * sensorForceGrams;
    #endif

    /* obtained actuator voltage from force in grams */
    (*pv).Solenoid = 0.008619688*(desiredForceSolenoid) + (po->Solenoid);

    /* generate voltage on actuator */
    DAC3DAT = outputDAC(pv->Solenoid);

    return;
}

void expMagnifyForces(struct voltages *pv, struct gains *pg, struct offsets *po)
{
    /*  Current Calibration Equations ----
     *  Force sensor (11 October 2016 -- binned least squares fit)
     *  ADC voltage --> force: y = 591.253227x -708.0596313
     *
     *  Voice Coil (11 October 2016)
     *  desired force --> DAC voltage: y = -0.008619688x + 1.44759825
     */

    float diffForceVoltage, sensorForceGrams, desiredForceSolenoid, knobVoltage;
    float curr_sensor_grams, offset_sensor_grams;

    /* check hardware pins, set gain in software */
    if (getExtSwitchSTATE(0,1) == 0 && getExtSwitchSTATE(0,2) == 0)
    {
        /* bit0 == 0, bit1 == 0 */
        /* (*pg).Force = 0.0; */
        DAC3DAT = outputDAC(1.50);
        return;
    }
    else if (getExtSwitchSTATE(0,1) == 1 && getExtSwitchSTATE(0,2) == 0)
    {
```

```
        /* bit0 == 1, bit1 == 0 */
        (*pg).Force = 2.0;
    }
    else if (getExtSwitchSTATE(0,1) == 0 && getExtSwitchSTATE(0,2) == 1)
    {
        /* bit0 == 0, bit1 == 1 */
        (*pg).Force = 4.0;
    }
    else if (getExtSwitchSTATE(0,1) == 1 && getExtSwitchSTATE(0,2) == 1)
    {
        /* bit0 == 1, bit1 == 1 */
        (*pg).Force = 3.0;
    }

    /** Force centered magnification: Measure force in grams from sensor
     ** using calibration eqns. Multiply by device gain to obtain desired
     ** force. Actuator voltage for this desired force obtained using
     ** calibration eqns.
     **/
    /* read force sensor on ADC0. readSensors() returns FLOAT */
    (*pv).Force = readSensors(0);

    /* new implementation (v014) -- calculate current and offset force
     * on sensor in grams, using calibration equations directly. then
     * subtract the two to get change.
     */
    curr_sensor_grams = 591.253227 * (pv->Force) - 708.0596313;
    offset_sensor_grams = 591.253227 * (po->Force) - 708.0596313;

    desiredForceSolenoid = (pg->Force) * (curr_sensor_grams - offset_sensor_grams);

    #if TEMP_DISABLED
    /* experimenting with previous voltage difference implementation */
    /* previous implementation (v013) -- calculate change in voltage,
     * then use that change to calculate the change in grams on
     * sensor
     */
    diffForceVoltage = (pv->Force) - (po->Force);

    /* calculate force in grams from voltage change, no intercept becuase
     * sensor offset @ 0 grams can drift, but slope should be the same
     */
    sensorForceGrams = 591.253227*(diffForceVoltage);

    /* desired force is the measured force multiplied by gain */
    desiredForceSolenoid = (pg->Force) * sensorForceGrams;
    #endif

    /* obtained actuator voltage from force in grams */
    (*pv).Solenoid = 0.008619688*(desiredForceSolenoid) + (po->Solenoid);

    /* generate voltage on actuator */
    DAC3DAT = outputDAC(pv->Solenoid);

    return;
}

/** generateForce() --
 **     Function to generate a particular force to test the current
 **     calibration
 **/
void generateForce(float force, struct voltages *pv, struct gains *pg,
                   struct offsets *po)
{
    (*pv).Solenoid = 0.01036*(force) + (po->Solenoid);
    DAC3DAT = outputDAC(pv->Solenoid);

    return;
}

/** potVoltage() --
 **     Calculates voltage on a potentiometer.
 **     Returns decimal scaled by fullscaleValue
 **/
float potVoltage(float voltage, float fullscaleValue)
{
    float calculatedVoltage;

    calculatedVoltage = (voltage / (float) 2.52) * (fullscaleValue);

    return calculatedVoltage;
}

/** storeSensorVoltageRandomly() --
 **     Store recorded voltage at pre-defined random points during 1 second
 **     sampling time. Builds arrays of sensor voltages
 **/
void storeSensorVoltageRandomly(struct voltages *pv)
{
    if (randomWaitCounter >= timeToWait)
    {
        /* store sensor voltages when required time reached */
        storedRandomSamples_Force[randomVoltageArrayCounter] = (*pv).Force;
        //storedRandomSamples_Accel[randomVoltageArrayCounter] = (*pv).Accel;
```

```
        /* reset wait counter */
        randomWaitCounter = 0;

        /* advance to next element in randomDelayTimes and storedRandomSamples
         * arrays
         */
        randomVoltageArrayCounter++;
        if (randomVoltageArrayCounter >= 100)
        {
            /* loop around random voltage array */
            randomVoltageArrayCounter = 0;

            /* flip flag to signify all 100 samples taken */
            randomSampleFlag = 1;
        }

        /* load next time to wait from randomDelayTimes array */
        timeToWait = randomDelayTimes[randomVoltageArrayCounter];
    }
    else
    {
        randomWaitCounter++;
    }

    return;
}

/** calculateMeanFromStoredRandomSamples() --
 **     Averages contents of stored sensor voltages
 **/
struct offsets *calculateMeanFromStoredRandomSamples(struct offsets *po)
{
    int i;
    float sum_Force, sum_Accel = 0.0;

    for (i=0; i<100; i++){
        sum_Force += storedRandomSamples_Force[i];
        sum_Accel += storedRandomSamples_Accel[i];
    }
    (*po).Force = sum_Force/100.0;
    (*po).Accel = sum_Accel/100.0;

    return po;
}

/** getSwitchSTATE() --
 **     Detects button push on Port.Pin. Returns 1 if pushed.
 **/
int getSwitchSTATE(int port_number, int pin_number)
{
    int pinInputData, pinInputData_MASKED, pinBit;

    if (port_number == 0){pinInputData = GP0DAT;}
    else if (port_number == 1){pinInputData = GP1DAT;}
    else if (port_number == 2){pinInputData = GP2DAT;}
    else if (port_number == 3){pinInputData = GP3DAT;}
    else if (port_number == 4){pinInputData = GP4DAT;}

    pinBit = pow(2,pin_number);

    /* masked result shows only bit corresponding to pin number */
    pinInputData_MASKED = (pinInputData & pinBit);

    /* bit is LOW when pushed, HIGH when not pushed */
    if (pinInputData_MASKED == 0)
    {
        return 1;
    }
    else
    {
        return 0;
    }
}

/** waitForRestOfPeriod() --
 **     Maintains 200us cycle using timer0 IRQ and global flags
 **/
void waitForRestOfPeriod(void)
{
    if (timer0_Flag == 1)
    {
        /* reset timer flag */
        timer0_Flag = 0;

        #if NOT_YET_IMPLEMENTED
        /* output LED light to signal device is operating slower than desired
         * frequency
         */
        #endif
        return;
    }
    else
    {
        while (timer0_Flag == 0){
            /* wait until interrupt request */
```

```
        }

        /* reset timer flag */
        timer0_Flag = 0;
        return;
    }
}

/** ADCinput() --
 **     Converts ADC conversion HEX to decimal voltage.
 **/
float ADCinput(int ADC_hex_result)
{
    float input_voltage;
    int ADC_hex_result_shifted;

    ADC_hex_result_shifted = ADC_hex_result >> 16;

    /* 2.52V chosen experimentally as max with ADC connected to voltmeter */
    input_voltage = ((ADC_hex_result_shifted)*2.52)/(0xFFF);
    return input_voltage;
}

/** outputDAC() --
 **     Converts decimal voltage to HEX for DAC output.
 **/
int outputDAC(float desired_voltage)
{
    #if OBSOLETE
    /* "unit conversion from desired decimal voltage to hex code. 2.4976V is
     * voltage when 0xFFF is applied to DAC
     */
    int output_integer = floor((desired_voltage*0xFFF)/2.497);

    if (output_integer <= 0xFFF && output_integer >= 0x200)
    {
        /* shift integer result 16 bits to left to conform to DACxDAT format
         * maximum and minimum integer cap result subtracted from small
         * integer as rough calibration to measured values
         */
        output_integer = (output_integer - 0x2) << 16;
    }
    else if (output_integer > 0xFFF)
    {
        /* maximum cap */
        output_integer = 0xFFF;
        output_integer = output_integer << 16;
    }
    else
    {
        /* minimum cap */
        output_integer = 0x200;
        output_integer = output_integer << 16;
    }
    return output_integer;
    #endif

    /* "unit conversion" from desired decimal voltage to hex code
     * 2.4976V is voltage when 0xFFF applied to DAC
     */
    int output_integer = floor((desired_voltage * 0xFFF) / 2.497);

    if (desired_voltage < 2.49 && desired_voltage >= 0.036)
    {
        /* shift integer 16 bits to left to conform to DACxDAT format, within
         * maximum and minimum voltage caps
         */
        output_integer = output_integer << 16;
    }
    else if (desired_voltage >= 2.49)
    {
        /* maximum possible voltage */
        output_integer = 0xFFF;
        output_integer = output_integer << 16;
    }
    else if (desired_voltage < 0.036 && desired_voltage >= 0.0065)
    {
        /* function obtained by linear fit for voltages below 0.036V */
        desired_voltage = 1.2 * desired_voltage - 0.0072733;
        output_integer = floor((desired_voltage * 0xFFF) / 2.497);
        output_integer = output_integer << 16;
    }
    else if (desired_voltage < 0.0065)
    {
        /* minimum possible voltage */
        output_integer = 0x00;
    }

    return output_integer;
}

/** system_init() --
 **     Configures CPU clock, ADC, DAC, GPIO P2.x, & P3.x
 **/
void system_init(void)
```

```
{
    //delay_sw(100000);

    POWKEY1 = 0x01;
    POWCON = 0x00;          /* Configures CPU Clock for 41.78MHz, CD=0 */
    POWKEY2 = 0xF4;

    PLLKEY1 = 0xAA;
    PLLCON = 0x01;
    PLLKEY2 = 0x55;

    /* Connect internal 2.5V reference on Vref pin
     * Set DAC3 output to 0-V_ref(2.5V) range and turn DAC3 on
     */
    REFCON = 0x01;
    DAC3CON = 0x12;

    /* Set digital I/O port 0 as GPIO on pins P0.0-0.7
     * Set all I/O pins on port 0 as *INPUT*, initialize LOW
     */
    GP0CON = 0x00;
    GP0DAT = 0x00;

    /* Set digital I/O port 2 as *GPIO* on pins P2.0-2.7
     * Set all pins on port 2 as *OUTPUT*, initialize LOW
     */
    GP2CON = 0x00;
    GP2DAT = 0xFF000000;

    /*Set digital I/O port 3 as GPIO on pins P3.0-3.7
     *Set all I/O pins on port 3 as *INPUT*, initialize LOW
     */
    GP3CON = 0x00;
    GP3DAT = 0x00;
}

/** timer0_init() --
 **     Configures timer0 for 200us period keeping
 **/
void timer0_init(void)
{
    /** timer0 currently set to 200us period (5 kHz operating freq) **/

    /* re-initialize flag */
    timer0_Flag = 0;

    /* re-initialize timer IRQ */
    IRQEN |= 0x04;

    /* 0x1053 = 100us period at 41.78MHz clock freq
     * 0x20A6 = 200us; 0x30F9 for 300us
     */
    T0LD = 0x20A6;

    /* enable timer0, set in periodic mode, with multiplier = 1 */
    T0CON = 0xC0;
}

/** timer2_init() --
 **     Configures timer2 for long term period keeping
 **     (@TODO: as of 08 Aug 2013)
 **/
void timer2_init(void)
{
    //timer2 to see if tare happened recently
    IRQEN |= 0x10;

    /* F00 = 15s; 1E00 = 30s; 20000 = 2 min */
    T2LD = 0xF00;

    /* 32 kHz internal oscillator, HH:MM:SS format, periodic, counting down */
    T2CON = 0xE8;
}

/** randomSample_init() --
 **     Configures timer0 and variables for random sampling
 **/
void randomSample_init(void)
{
    /** Function called by tareRoutine() **/

    /* re-initialize randomWaitCounter in case tare happens in middle of loop
     */
    randomWaitCounter = 0;
    randomVoltageArrayCounter = 0;

    /* load first random delay time */
    timeToWait = randomDelayTimes[0];

    /* flag to signify all 100 samples taken.
     * Flipped in storeSensorVoltageRandomly()
     */
    randomSampleFlag = 0;

    /** Re-initialize timer0 for 100us sampling period **/
```

```
    timer0_Flag = 0;

    /* Re-initialize timer IRQ */
    IRQEN |= 0x04;

    /* 0x1053 = 100us period at 41.78MHz clock freq.
     * 0x20A6 for 200us; 0x30F9 for 300us
     */
    T0LD = 0x1053;

    /* enable timer0, set in periodic mode with multiplier = 1 */
    T0CON = 0xC0;
}

/** numToChar() --
 **     Converts FLOAT to CHAR
 **/
unsigned char numToChar(float num, float max)
{
    unsigned char result;
    result = floor((num*0xFF)/max);

    return result;
}

/** ADCpoweron() --
 **     Power on ADC, software-timed wait
 **/
void ADCpoweron(int time)
{
    /* power-on ADC */
    ADCCON = 0x20;

    /* wait for ADC to be fully powered on */
    delay_sw(time);

    return;
}

/** delay_sw() --
 **     Software delay. time = 10 --> ~12us
 **/
void delay_sw(int time)
{
    while (time >=0){
        time --;
    }
    return;
}

/** delay_us() --
 **     Microsecond delay on timer1. Enables/disables timer1 within function
 **/
void delay_us(unsigned long time)
{
    /* reset globals and timer1 */
    timer1_Flag = 0;
    T1CON = 0x0;

    /* enable timer1 IRQ; timer1 = bit3 */
    IRQEN |= 0x08;

    /* T1LD = time (us) * CPU freq (in MHz), e.g. 500us * 41.78 MHz = 20890-1
     *      -> subtract 1 since count includes zero
     * Initialize timer1, 41.78MHz, down mode, periodic
     */
    T1LD = ((time * 42782) >> 10) - 1;
    T1CON = 0xC0;

    while (timer1_Flag == 0){
        /* wait for timer1 flag flip */
    }
    timer1_Flag = 0;

    /* disable timer1 and clear IRQ */
    T1CON = 0x0;
    IRQCLR |= 0x08;

    return;
}

/** delay_ms() --
 **     Millisecond delay on timer1 using delay_us().
 **/
void delay_ms(unsigned long time)
{
    unsigned long i = 0;
    /* disable timer0 to prevent interrupt confusion */
    T0CON = 0x00;

    /* Wait for 1 ms each time through loop */
    for (i=0; i<time; i++){
        delay_us(1000);
    }
```

```
    /* re-enable timer0 */
    timer0_init();

    return;
}

/** calibrationProcedure() --
 **     Alternate device behavior to step through solenoid voltages
 **     with front panel button push. Demo of blinking LED.
 **/
void calibrationProcedure(void)
{
    float appliedVoltage = 1.50;
    float voltageStep = 0.05;
    //float previousVoltage = 1.50;
    int counter = 1;

    while (1){
        /** Remap button functions **/
        /* tare button forces positive step (pull) */
        if(getSwitchSTATE(3,1) == 1) voltageStep = 0.05;

        /* mute button forces negative step (push) */
        else if(getSwitchSTATE(3,2) == 1) voltageStep = -0.05;

        /* X button to actuate voltage change */
        else if(getSwitchSTATE(3,3) == 1)
        {
            /* timer delay to eliminate contact bounce */
            delay_ms(500);

            appliedVoltage += voltageStep;

            if(appliedVoltage >= 2.50)
            {
                DAC3DAT = outputDAC(2.50);
                voltageStep = -0.05;
            }
            else if(appliedVoltage <= 0.0)
            {
                DAC3DAT = outputDAC(0.0);
                voltageStep = 0.05;
            }
        }
        /** "Default" behavior -- blink on and off with compliment function
         ** (i.e. ^=)
         **/
        else
        {
            if(counter % 2500 == 0)
            {
                #if OBSOLETE
                /*GP2_State = GP2DAT & (0xF00);
                    if(LED_Flag == 0)
                    {
                        GP2SET = 1 << 16;
                        LED_Flag = 1;
                    }
                    else if(LED_Flag == 1)
                    {
                        GP2CLR = 1 << 16;
                        LED_Flag = 0;
                    }
                */
                #endif
                GP2DAT ^= 0x10000;
            }
            counter++;

            DAC3DAT = outputDAC(appliedVoltage);
            //previousVoltage = appliedVoltage;
            waitForRestOfPeriod();
        }
    }
}

/** autoCalibrate() --
 **     Automatically cycle DAC output to solenoid. Each step is 20mV and
 **     lasts 100ms, or 500ms at extremes
 **/
void autoCalibrate(void)
{
    float appliedVoltage = 1.50;
    float voltageStep = 0.02;
    int maxReached = 0;

    DAC3DAT = outputDAC(appliedVoltage);
    delay_ms(250);

    while (maxReached < 2){

        delay_ms(100);

        appliedVoltage += voltageStep;
```

```
        if(appliedVoltage >= 2.50)
        {
            delay_ms(500);
            DAC3DAT = outputDAC(2.50);
            voltageStep = -0.02;
            maxReached += 1;
        }

        else if(appliedVoltage <= 0.0)
        {
            delay_ms(500);
            DAC3DAT = outputDAC(0.0);
            voltageStep = 0.02;
        }

        else
        {
            DAC3DAT = outputDAC(appliedVoltage);
        }
        //waitForRestOfPeriod();
    }

    /* Cycle down to 1.50V */
    while (appliedVoltage != 1.50){
        delay_ms(100);
        voltageStep = -0.02;

        /* Voltage step should be -0.02 after reaching max */
        appliedVoltage += voltageStep;
        DAC3DAT = outputDAC(appliedVoltage);
        //waitForRestOfPeriod();
    }
}

/** verifyCalibration() --
 **     Function to generate test forces to validate actuator calibration
 **/
void verifyCalibration(void)
{
    //generateForce(5, pV, pG, pO);
    //delay_ms(2000);
    generateForce(10, pV, pG, pO);
    delay_ms(1000);
    generateForce(30, pV, pG, pO);
    delay_ms(1000);
    generateForce(45, pV, pG, pO);

    generateForce(0, pV, pG, pO);
    delay_ms(1000);

    //generateForce(-5, pV, pG, pO);
    delay_ms(1000);
    generateForce(-10, pV, pG, pO);
    delay_ms(1000);
    generateForce(-30, pV, pG, pO);
    delay_ms(1000);
    generateForce(-45, pV, pG, pO);

    return;
}

/** experimentMode() --
 **     Alternate device behavior that allows control of HHFM device state
 **     from external Analog Devices ADuC7026
 **/
void experimentMode(void)
{
    while (1){
        /* Light middle red LED to show in 'Experimental Mode' */
        /* P2.1 for front panel LED (RED) */
        GP2DAT |= 1<<17;
        timer0_init();

        /*** Tare routine on P3.1 switch push ***/
        if (getSwitchSTATE(3,1) == 1)
        {
            /* turn off P2.1 during tare */
            GP2DAT &= ~(1<<17);

            /* reset timer0 to 100us period for 100-sample tare over 1s */
            pO = tareRoutine(pV,pG,pO);

            /* turn back on P2.1 */
            GP2DAT |= 1<<17;

            /* reset timer0 to 200us period */
            timer0_init();
        }

        /** Magnify forces if P0.0 goes LOW (via setMag() from exp. ADuC) ***/
        else if (getSwitchSTATE(0,0) == 1)
        {
            /* allow for stabilization of extern. 3.3V signal */
            delay_us(50);
```

```
            while (getSwitchSTATE(0,0) == 1){
                /* allow re-tare during trial block */
                                    if (getSwitchSTATE(3,1) == 1)
                                    {
                                              /* turn off P2.1 (RED Exp LED) during tare */
                                              GP2DAT &= ~(1<<17);

                                              /* initiate 1s tare routine, turn off timer0 within func */
                                              pO = tareRoutine(pV,pG,pO);

                                              GP2DAT |= 1<<17;

                                              timer0_init();
                }

                expMagnifyForces(pV,pG,pO);
                waitForRestOfPeriod();
            }
        }

        else
        {
            DAC3DAT = outputDAC(1.50);
            waitForRestOfPeriod();
        }
    }
}

/** getExtSwitchSTATE() --
 **     Get external pins' HIGH or LOW state.
 **/
int getExtSwitchSTATE(int port_number, int pin_number)
{
    int pinInputData, pinInputData_MASKED, pinBit;

    if (port_number == 0){pinInputData = GP0DAT;}
    else if (port_number == 1){pinInputData = GP1DAT;}
    else if (port_number == 2){pinInputData = GP2DAT;}
    else if (port_number == 3){pinInputData = GP3DAT;}
    else if (port_number == 4){pinInputData = GP4DAT;}

    pinBit = pow(2,pin_number);

    /* masked result shows only bit corresponding to pin number */
    pinInputData_MASKED = (pinInputData & pinBit);

    /* if bit is HIGH (3.3V from ext ADuC7026) return 1 */
    if (pinInputData_MASKED != 0)
    {
        return 1;
    }
    else
    {
        return 0;
    }
}
/* EOF ------------------------------------------------------------------------ */
```

C.3 MEMBRANE PUNCTURE MLHD SIMULATION SOURCE CODE

```
/*************************************************************************
    Randy Lee (ral63@pitt.edu)
    VIA Lab
    University of Pittsburgh
    HHFM MLHD Experiments

    Originally developed by Vikas Shivaprabhu

    Butterfly Haptics Magnetically Levitated Haptic Device

*************************************************************************/
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <math.h>
#include <string.h>
#include <time.h>
#include "..\ml\ml_api.h"
#include "GL\glut.h"
#include <fstream>

//#define MembraneVisualization 1

static ml_device_handle_t Device_Handle = NULL;
float Servo_Frequency = 1000.0f;    // servo frequency in Hz
float Boundary_Radius = 0.012f;     // the radius of the bounding sphere in meters
char* Default_IP = "192.168.0.2";   // default ip address to be connected

// Haptic rendering functions
int InitializeMLHD(char* Server_Address, ml_device_handle_t *device_hdl);
void SmoothTransitionTo(ml_device_handle_t dev_hdl, ml_position_t desiredPosition);

// GLUT callback functions
void Resize(int width, int height);
void DisplayFunction();
void ProcessNormalKeys(unsigned char key, int x, int y);

//Global variables
ml_forces_t Gravity[17][17];       // Gravity matrix obtained from calibration
static float ConstrainXAxisMin=-0.008;
static float ConstrainXAxisMax=0.008;
static float ConstrainZAxisMin=-0.008;
static float ConstrainZAxisMax=0.008;

//Callback Handler variables
static ml_position_t   curr_Position;
static ml_velocities_t curr_Velocity;
static ml_forces_t curr_Force;
static float curr_p_x,curr_v_x,curr_p_z,curr_v_z;
static float gravityX,gravityZ;
static ml_gain_pid_t   VecGains;
static int   ilower, iupper;
float SpringForce, SpringConstant=200, SpringDamping;
float SpringPosition = ConstrainXAxisMin;
float data_positionX[120000],data_velocityX[120000],data_SpringForceX[120000];
float data_positionZ[120000],data_velocityZ[120000],data_SpringForceZ[120000];
float data_AppliedForceX[120000],data_AppliedForceZ[120000];
float data_SpringStiffness[120000];
float SpringForceX, SpringForceZ, AppliedForceX, AppliedForceZ;
int data_TimeStamp[120000];
int DataIndex = 0;
int TickCounter = 0;
int CurrentFMagStatus,CurrentPushOrPull;
static int TrialNumber = 0;
char            SaveDataFileName[100]           =           "H:\\Documents           and           Settings\\Randy\\My
Documents\\ForceMagnifierExp\\Release\\Gravity_Exp_";
float ForceX,ForceZ;
float Radius;
int SigmoidConstant = 15000;
float MembraneStartingPosition = 0.002f;
float PopDistance,AlleyDistance;
int zindex,xindex;
char          GravityValuesXFileName[100]          =          "H:\\Documents          and          Settings\\Randy\\My
Documents\\ForceMagnifierExp\\GravityValuesX.txt";
char          GravityValuesZFileName[100]          =          "H:\\Documents          and          Settings\\Randy\\My
Documents\\ForceMagnifierExp\\GravityValuesZ.txt";
char str[1024];
int StartTaskFlag = 0;
time_t t1,t2,t3;               // System time variables
bool WithinMembrane = 0, MembranePenetrated = 0;
```

```
int TimeStamp;
int EndofTrialFlag = 0;
bool BeepCompleted = 0;

float SpringStiffness[100];
int FMagStatus[100];
int NumOfTrials;
float MembranePositionList[100];
int PushOrPull[100];

int main(int argc, char* argv[])
{
    char FileName[100] = "H:\\Documents and Settings\\Randy\\My Documents\\"
                         "ForceMagnifierExp\\Release\\FMagExperiment.txt";

    NumOfTrials = LoadExperimentParameters(FileName);
    for(int i=0;i<NumOfTrials;i++)
        printf("\nSpringStiffness[%d]=%f",i,SpringStiffness[i]);

    // Initialize the MLHD; InitializeMLHD() returns # of trials
    int Initializeflag;
    Initializeflag = InitializeMLHD(Default_IP, &Device_Handle);
    if(Initializeflag)
        return -1;

    //Constrain the flotor
    ml_ConstrainAxis(Device_Handle, ML_AXIS_X_POS, ConstrainXAxisMin,
                     ConstrainXAxisMax);
    ml_ConstrainAxis(Device_Handle, ML_AXIS_Z_POS, ConstrainZAxisMin,
                     ConstrainZAxisMax);

    /*
    // Calibrate & save gravity matrix
    ml_position_t pos;
    printf("\n\n\nCalibrating. Please wait......\n");
    for(int i=0;i<6;i++) pos.values[i] = 0.0f;

    for(int j=0;j<17;j++){
        pos.values[2] = ConstrainZAxisMin + 0.001f*j;
        printf("\n%d\t",j);
        for(int i=0;i<17;i++){
            pos.values[0] = ConstrainXAxisMin + 0.001f*i;
            SmoothTransitionTo(Device_Handle, pos);
            Sleep(250);
            ml_FindGravity(Device_Handle, &Gravity[j][i]);
            //printf("\nPosition %d,%d: %f\tGravity:%f",i,j,pos.values[0],Gravity[j][i].values[0]);
            printf("*");
        }
    }
    printf("\n\n\n");
    SaveGravityValues(GravityValuesXFileName,GravityValuesZFileName,Gravity);
        //Move the flotor to the center
    for(int i=0;i<6;i++)
        pos.values[i]=0.0;
    SmoothTransitionTo(Device_Handle, pos);
    */

    // Load and print gravity values
    LoadGravityValues(GravityValuesXFileName,GravityValuesZFileName,Gravity);

    printf("GravityX:\n");
    for(int j=0;j<(int)(1000*ConstrainZAxisMax*2 + 1);j++){
        printf("\n");
        for(int i=0;i<(int)(1000*ConstrainXAxisMax*2 + 1);i++){
            printf("%12.6f\t",Gravity[j][i].values[0]);
        }
    }

    printf("GravityZ:\n");
    for(int j=0;j<(int)(1000*ConstrainZAxisMax*2 + 1);j++){
        printf("\n");
        for(int i=0;i<(int)(1000*ConstrainXAxisMax*2 + 1);i++){
            printf("%12.6f\t",Gravity[j][i].values[2]);
        }
    }

    // Save data and initiate next trial
    NextTrial();

    if(ML_STATUS_OK != ml_RegisterCallbackTick(Device_Handle, tick_callback_handler)){
        printf("\n\tERROR: Failed to register the callback function\n");
        ml_Land(Device_Handle);
        ml_Disconnect(Device_Handle);
        return -1;
    }

    // Visualization using GLUT
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA | GLUT_DEPTH);
    glutInitWindowSize(800, 800);
    glutInitWindowPosition(0, 0);
    glutCreateWindow("Force Magnifier");

    // GLUT callbacks functions
    glutIdleFunc(NULL);
```

```
    glutReshapeFunc(Resize);
    glutDisplayFunc(DisplayFunction);
    glutKeyboardFunc(ProcessNormalKeys);
    glutMainLoop();

    //getch();
    return 0;
}

/************************** Haptic Functions ******************************/

int tick_callback_handler(ml_device_handle_t dev_hdl, ml_position_t* pos)
{
    // Function runs at servo frequency providing force feedback & position
    // reading

    if(StartTaskFlag == 0){
        ml_ResetGainVecAxes(Device_Handle,ML_GAINSET_TYPE_NORMAL);
        return 0;
    }
    TickCounter++;

    if(SpringConstant < 150)
          SpringConstant = 150;
    //else if(SpringConstant > 1000)
    //      SpringConstant = 1000;

    SpringDamping = 5.0;

    // read the current status of the flotor
    ml_GetActualPosition(dev_hdl, &curr_Position);
    ml_GetVelocity (dev_hdl, &curr_Velocity);
    ml_GetForces(dev_hdl, &curr_Force);

    curr_p_x = curr_Position.values[0];
    curr_v_x = curr_Velocity.values[0];
    curr_p_z = curr_Position.values[2];
    curr_v_z = curr_Velocity.values[2];

    // gravity calibration using the readings obtained previously in main()
    // X axis calibration
    if(curr_p_x <= ConstrainXAxisMin){
        // use the reading at x = -0.008m
        gravityX = Gravity[0][0].values[0];
    }
    else{
        if(curr_p_x >= ConstrainXAxisMax){
            //use the reading at x = +0.008m
            gravityX = Gravity[0][17].values[0];
        }
        else{
            // estimate the gravity reading by linear interpolation
            zindex = (int)(1000.0*(curr_p_z+ConstrainZAxisMax));
            ilower = (int)(1000.0*(curr_p_x+ConstrainXAxisMax));
            iupper = ilower + 1;
            gravityX = (1000.0*(curr_p_x+ConstrainXAxisMax)-ilower)*Gravity[zindex][ilower].values[0] +
                       (iupper-1000.0*(curr_p_x+ConstrainXAxisMax))*Gravity[zindex][iupper].values[0];
        }
    }

    // Z axis calibration
    if(curr_p_z <= ConstrainZAxisMin){
        //use the reading at z= -0.006m
        gravityZ = Gravity[0][0].values[2];
    }
    else{
        if(curr_p_z >= ConstrainZAxisMax){
            //use the reading at z= +0.006m
            gravityZ = Gravity[17][0].values[2];
        }
        else{
            // estimate the gravity reading by linear interpolation
            xindex = (int)(1000.0*(curr_p_x+ConstrainXAxisMax));
            ilower = (int)(1000.0*(curr_p_z+ConstrainZAxisMax));
            iupper = ilower + 1;
            gravityZ = (1000.0*(curr_p_z+ConstrainZAxisMax)-ilower)*Gravity[ilower][xindex].values[2] +
                       (iupper-1000.0*(curr_p_z+ConstrainZAxisMax))*Gravity[iupper][xindex].values[2];
        }
    }

    // Set the force feedback
    // Calculate the vector from the current position of the flotor to the
    // center of the XZ plane.
    Radius = sqrt(pow(curr_p_x,float(2.0)) + pow(curr_p_z,float(2.0)));

    // X-Axis
    // Negative X axis
    if(curr_p_x < 0.0 && Radius < PopDistance &&
        (Radius - MembraneStartingPosition) >= 0)
    {
        // Inside membrane
        VecGains.ff = gravityX +
                      ((Radius - MembraneStartingPosition) * SpringConstant -
                        SpringDamping * curr_v_x);
        VecGains.d  = 15.0f;
```

```
}
// Positive X axis
else if (curr_p_x > 0.0 && Radius < PopDistance &&
          (Radius - MembraneStartingPosition) >= 0)
{
    VecGains.ff = gravityX +
                  (-(Radius  - MembraneStartingPosition)* SpringConstant -
                    SpringDamping * curr_v_x);
    VecGains.d  = 15.0f;
}
else
{
    //Before And After the membrane, render weightlessness
    VecGains.ff = gravityX;
    VecGains.d  = 15.0f;
}
VecGains.p  = 0.0f;
VecGains.i  = 0.0f;

// Set calculated force feedback values in X axis
ml_SetGainVecAxis(Device_Handle, ML_GAINSET_TYPE_NORMAL, ML_AXIS_X,
                  VecGains);

// Save force applied by membrane to subject, and efferent force applied
// by subject
SpringForceX = VecGains.ff - gravityX;
AppliedForceX = curr_Force.values[0] - gravityX;

// Z-axis
// Negative Z axis
if(curr_p_z < 0.0 && Radius < PopDistance &&
    (Radius - MembraneStartingPosition) >= 0)
{
    // Inside membrane
    VecGains.ff = gravityZ +
                  ((Radius - MembraneStartingPosition) * SpringConstant -
                    SpringDamping * curr_v_z);
    VecGains.d  = 15.0f;
}
// Positive Z axis
else if(curr_p_z > 0.0 && Radius < PopDistance &&
        (Radius - MembraneStartingPosition)>=0)
{
    VecGains.ff = gravityZ +
                  (-(Radius - MembraneStartingPosition) * SpringConstant -
                     SpringDamping * curr_v_z);
    VecGains.d  = 15.0f;
}
else
{
    // Outside membrane, render weightlessness
    VecGains.ff = gravityZ;
    VecGains.d  = 15.0f;
}
VecGains.p  = 0.0f;
VecGains.i  = 0.0f;

// Set calculated force feedback values in Z axis
ml_SetGainVecAxis(Device_Handle, ML_GAINSET_TYPE_NORMAL, ML_AXIS_Z,
                  VecGains);

// Save force applied by membrane to subject, and efferent force applied
// by subject
SpringForceZ = VecGains.ff - gravityZ;
AppliedForceZ = curr_Force.values[2] - gravityZ;

// Timestamp and conditions set
if((Radius < MembraneStartingPosition))
{
    // Before membrane
    TimeStamp = 0;
    WithinMembrane = 0;
    MembranePenetrated = 0;
}
else if ((Radius > MembraneStartingPosition) && (Radius < PopDistance))
{
    // Inside membrane
    TimeStamp = 1;
    if(!WithinMembrane)
    {
        // time() function stores current system time into t1 pointer
        (void) time(&t1);
        WithinMembrane = 1;
    }

}
else if ((Radius > PopDistance))
{
    // Outside membrane
    // continuously store time outside of membrane in t3
    (void) time(&t3);
    TimeStamp = 2;
    if(!MembranePenetrated)
    {
        // Save initial time of membrane break
```

```
            (void) time(&t2);
            MembranePenetrated = 1;
        }
    }

    if(0==(TickCounter%5))
    {
        // save data into temp array only once every 5 tickback loops
        data_positionX[DataIndex]=curr_p_x;
        data_velocityX[DataIndex]=curr_v_x;
        data_AppliedForceX[DataIndex] = AppliedForceX;
        data_SpringForceX[DataIndex]=SpringForceX;

        data_positionZ[DataIndex]=curr_p_z;
        data_velocityZ[DataIndex]=curr_v_z;
        data_AppliedForceZ[DataIndex] = AppliedForceZ;
        data_SpringForceZ[DataIndex]=SpringForceZ;

        data_SpringStiffness[DataIndex]=SpringConstant;
        data_TimeStamp[DataIndex] = TimeStamp;
        if (DataIndex < 120000)
            DataIndex++;
        else
            DataIndex = 120000;
    }

    if((int)(t3-t2) > 2){
        // End trial 2s after puncturing membrane
        // TrialFlag = 0 to release handle
        EndofTrialFlag = 0;
    }

    return 0;
}


/*****************************************************************************
 * Function:     To initialize the Maglev device
 *****************************************************************************/
int InitializeMLHD(char* Server_Address, ml_device_handle_t *device_hdl)
{

    ml_fault_t      fault;
    int flag;   //Used to Defy Gravity

    // to create a connection between the program and the MLHI controller
    if(ML_STATUS_OK != ml_Connect(device_hdl, Server_Address)) {
        fprintf(stderr, "\nError!!!! Unable to connect to the server (%s).\n",
                Server_Address);
        return -1;
    }

    // to take off the flotor
    do{
        do{
            ml_ResetFault(*device_hdl);
            Sleep(500);
            ml_GetFault(*device_hdl, &fault);
        }while(ML_STATUS_OK != fault.value);
        flag = ml_Takeoff(*device_hdl);
    }while(ML_STATUS_OK != flag);

    // Set the device parameters
    ml_SetServoFrequency(*device_hdl, Servo_Frequency);
    ml_SetBoundaryRadius(*device_hdl, Boundary_Radius);

    return 0;
}


/*****************************************************************************
Function:     To disconnect the client application from the Maglev device
*****************************************************************************/
void DisconnectMLDH(ml_device_handle_t device_hdl)
{
    // unregister the previously registered callback function
    ml_UnregisterCallbackTick(device_hdl);
    Sleep(25);
    // unlock all axes
    ml_UnlockAxis (device_hdl, ML_AXIS_X_ROT);
    ml_UnlockAxis (device_hdl, ML_AXIS_Y_ROT);
    ml_UnlockAxis (device_hdl, ML_AXIS_Z_ROT);

    ml_UnlockAxis (device_hdl, ML_AXIS_Y_POS);

    ml_UnlockAxis (device_hdl, ML_AXIS_X_POS);
    ml_UnlockAxis (device_hdl, ML_AXIS_Z_POS);
    // reset the device to the default status
    ml_Land (device_hdl);
    // disconnect from server
    ml_Disconnect(device_hdl);

    return ;

}
```

```
/*****************************************************************************
Function:     To smoothly move the flotor to a specified location
*****************************************************************************/
void SmoothTransitionTo(ml_device_handle_t dev_hdl,
                        ml_position_t desiredPosition)
{
    int i,j,Steps = 125;
    ml_position_t   curPosition, tgtPosition;

    ml_GetActualPosition(dev_hdl, &curPosition);
    for(i=0;i<=Steps;i++){
        for(j=0;j<6;j++)
            tgtPosition.values[j]= curPosition.values[j] +
                                   (desiredPosition.values[j]-curPosition.values[j])*(float)i/(float)Steps;
        ml_SetDesiredPosition(dev_hdl, tgtPosition);
        // wait for 2ms until the action is done
        Sleep(2);
    }
    return;
}

/*****************************************************************************
Function:       To save data to file
*****************************************************************************/
int SaveData(char *fn)
{
    int i,flag;
    FILE *fp;
    // to save the data
    flag = fopen_s(&fp,fn, "wt");
    if(flag) return -1;
    else{
        fprintf_s(fp,"Trial %d\nPush/Pull: %d\t Membrane Start Pos: %12.6f",
                  TrialNumber, CurrentPushOrPull, MembraneStartingPosition);

fprintf_s(fp,"SpringStiffness\tPositionX\tPositionZ\tVelocityX\tVelocityZ\tAppliedForceX\tAppliedForceZ\tSpringForceX\tSpringForc
eZ\tTimeStamp\n");
        for(i=0;i<DataIndex;i++){
            fprintf_s(fp,"%f\t%12.6f\t%12.6f\t%12.6f\t%12.6f\t%12.6f\t%12.6f\t%12.6f\t%d\n",
                      data_SpringStiffness[i], data_positionX[i],
                      data_positionZ[i], data_velocityX[i], data_velocityZ[i],
                      data_AppliedForceX[i], data_AppliedForceZ[i],
                      data_SpringForceX[i], data_SpringForceZ[i],
                      data_TimeStamp[i]);
        }
        fprintf_s(fp,"Time Within Membrane = %f",(float)t2-t1);
        fclose(fp);
    }

    return 0;
}

/*****************************************************************************************
Function:       To load experimental parameters from file
*****************************************************************************************/
int LoadExperimentParameters(char *Filename)
{
    FILE *fp;
    char temp[4096];
    int NumOfParameters = 4;
    int counter = 0;

    if(0==fopen_s( &fp,Filename,"rt"))
    {
        do{
            temp[0]='\0';
            fgets(temp,4095,fp);
            if(strlen(temp)>3){
                //Lines starting with #: comments
                if('#'!=temp[0]){
                    if(NumOfParameters ==
                       sscanf_s(temp,"%d %f %f %d",&FMagStatus[counter],
                                &MembranePositionList[counter],
                                &SpringStiffness[counter],
                                &PushOrPull[counter])){
                        counter++;
                    }
                }
            }
        }while(!feof(fp));
        fclose(fp);
    }
    return counter;
}

/*****************************************************************************
Function:       Save trial data and load parameters for next trial
*****************************************************************************/
int NextTrial()
{
    // Reset OpenGL visualisation flags
    StartTaskFlag = 0;
    EndofTrialFlag = 2;    // Flag = 2 to initiate start of next trial graphic
```

```
    BeepCompleted = 0;

    // Reset to center
    ml_position_t ResetPosition;
    for(int i=0;i<6;i++)
        ResetPosition.values[i]=0.0;
    ml_UnregisterCallbackTick(Device_Handle);
    ml_ResetGainVecAxes(Device_Handle,ML_GAINSET_TYPE_NORMAL);
    Sleep(10);
    SmoothTransitionTo(Device_Handle, ResetPosition);
    if(ML_STATUS_OK != ml_RegisterCallbackTick(Device_Handle, tick_callback_handler))
        return -1;

    // Save data from previous trial
    sprintf_s(str,"%s%02dData.txt",SaveDataFileName,TrialNumber);
    SaveData(str);
    DataIndex = 0;
    TickCounter  =-1;

    // Load parameters for current trial
    if (TrialNumber < NumOfTrials){
        SpringConstant = SpringStiffness[TrialNumber];
        CurrentFMagStatus = FMagStatus[TrialNumber];
        CurrentPushOrPull = PushOrPull[TrialNumber];
        MembraneStartingPosition = MembranePositionList[TrialNumber];
        PopDistance = MembraneStartingPosition + 0.0005;
        AlleyDistance = PopDistance + 0.002f;
        TrialNumber++;
    }
    else{
        DisconnectMLDH(Device_Handle);
        // release the device handle
        Device_Handle = NULL;
        exit(0);
    }

    /* Constraints for 1 = push; 2 = pull */
    if(CurrentPushOrPull == 1){
        ml_ConstrainAxis (Device_Handle, ML_AXIS_X_POS, ConstrainXAxisMin, 0);
        ml_ConstrainAxis (Device_Handle, ML_AXIS_Z_POS, ConstrainZAxisMin, -0.0001);
    }
    else if(CurrentPushOrPull == 2){
        ml_ConstrainAxis (Device_Handle, ML_AXIS_X_POS, 0, ConstrainXAxisMax);
        ml_ConstrainAxis (Device_Handle, ML_AXIS_Z_POS, 0.0001, ConstrainZAxisMax);
    }

    Sleep(10);
    t1=0;t2=0;t3=0;

    return 0;
}

/*****************************************************************************
Function:       Save grvity values in space of flotor
*****************************************************************************/
int SaveGravityValues(char *fn,char *fn2,ml_forces_t Gravity[17][17])
{
    int i,flag;
    FILE *fp;
    // to save the data
    flag = fopen_s(&fp,fn, "wt");
    if(flag) return -1;
    else{
        for(int j=0;j<(int)(1000*ConstrainZAxisMax*2 + 1);j++){
            fprintf_s(fp,"\n");
            for(int i=0;i<(int)(1000*ConstrainXAxisMax*2 + 1);i++){
                fprintf_s(fp,"%f\t",Gravity[j][i].values[0]);
            }
        }
        fclose(fp);
    }

    flag = fopen_s(&fp,fn2, "wt");
    if(flag) return -1;
    else{
        for(int j=0;j<(int)(1000*ConstrainZAxisMax*2 + 1);j++){
            fprintf_s(fp,"\n");
            for(int i=0;i<(int)(1000*ConstrainXAxisMax*2 + 1);i++){
                fprintf_s(fp,"%f\t",Gravity[j][i].values[2]);
            }
        }
        fclose(fp);
    }

    return 0;
}

/*****************************************************************************
Function:       Load gravity values from file
*****************************************************************************/
int LoadGravityValues(char *Filename, char *Filename2,
                      ml_forces_t Gravity[17][17])
{
    FILE *fp;
    char temp[4096];
```

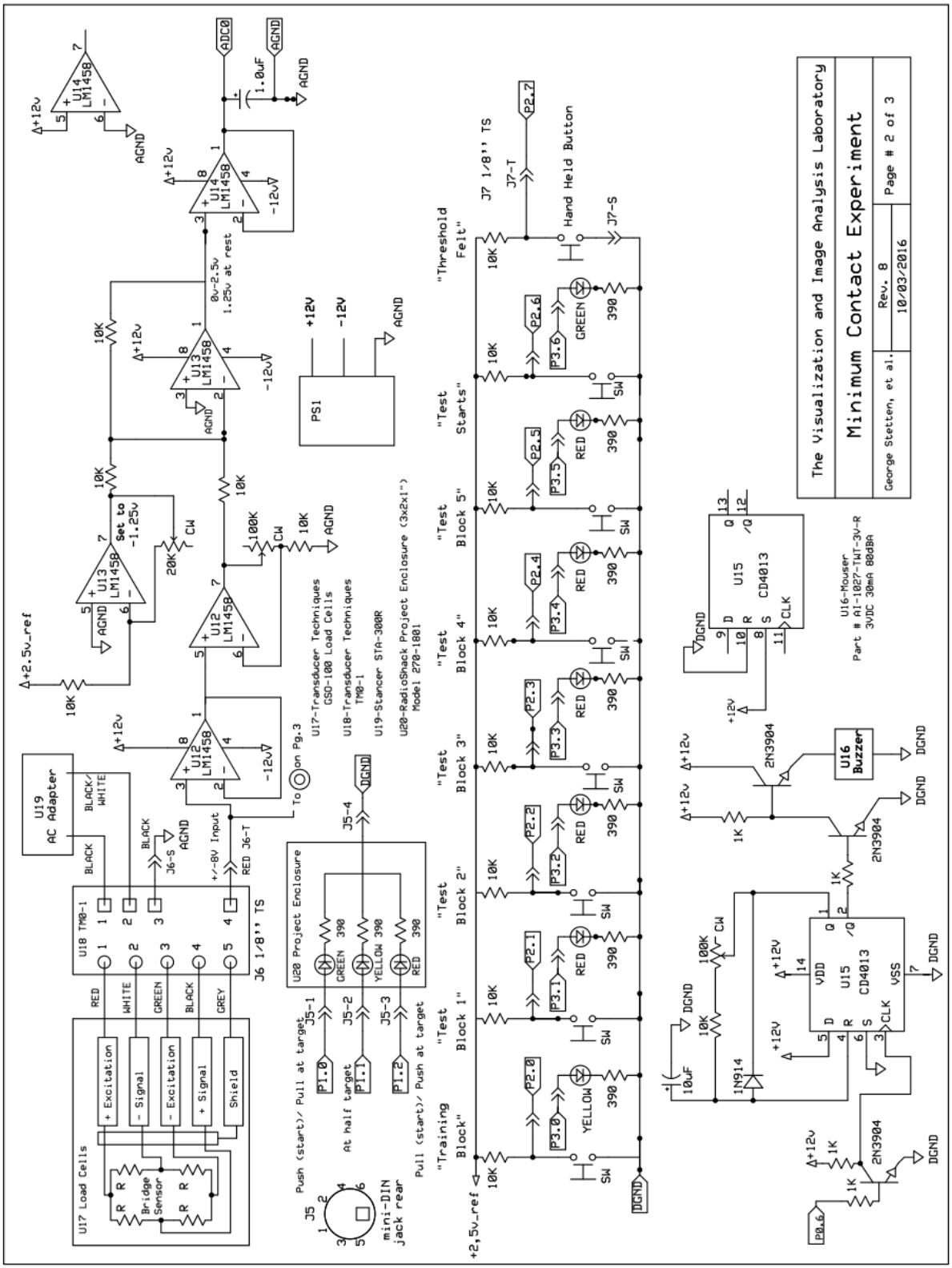
```
    std::ifstream f(Filename);
    for (int i = 0; i < (int)(1000*ConstrainZAxisMax*2 + 1); ++i){
        for (int j = 0; j < (int)(1000*ConstrainXAxisMax*2 + 1); ++j){
            if (!(f >> Gravity[i][j].values[0]))
                return false;
        }
    }
    std::ifstream f2(Filename2);
    for (int i = 0; i < (int)(1000*ConstrainZAxisMax*2 + 1); ++i){
        for (int j = 0; j < (int)(1000*ConstrainXAxisMax*2 + 1); ++j){
            if (!(f2 >> Gravity[i][j].values[2]))
                return false;
        }
    }
    return 1;
}


/***************************** GLUT callback functions *****************************/
/* Callback function for resizing the display window */
void Resize(int width, int height)
{
    glViewport(0,0,width,height);
    return;
}

/* Callback function for processing keyboard events */
void ProcessNormalKeys(unsigned char key, int x, int y)
{
    switch(key) {
        // To end the program
        case 0x1b: //ESC
        case 'Q':
        case 'q':
            if(Device_Handle){
                DisconnectMLDH(Device_Handle);
                // release the device handle
                Device_Handle = NULL;
            }
            //SaveData(SaveDataFileName);
            exit(0);
        case '+'://Increase stiffness
            SpringConstant += 50;
            break;
        case '-'://Decrease Stiffness
            SpringConstant -= 50;
            break;
        case 0xd: //Carriage Return
            if(0 != NextTrial())
                DisconnectMLDH(Device_Handle);
            EndofTrialFlag = 2;
            BeepCompleted = 0;
            break;
        case 0x20: //Space
            StartTaskFlag=1;
            EndofTrialFlag = 1;         // Flag = 1 for currently in trial
            BeepCompleted = 0;
            break;

        default:
            break;
    }

    return;
}

char str2[50];


void DisplayFunction()
{
    int i;
    float scaling;
    float pos_flotor,pos_spring,pos_flotorZ;
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(-1, 1, -1, 1, -1, 1);
    /* the visible range of flotor displacement is set to (-0.008,+0.008) */
    scaling = 1.0f/0.008f;
    glMatrixMode(GL_MODELVIEW);
    //glClearColor(0,0,0,0);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    pos_flotor = curr_Position.values[0];
    pos_flotorZ = curr_Position.values[2];

#ifdef MembraneVisualization
    //Alley, region the subject should not cross after popping
    glLoadIdentity();
    glTranslatef( 0.0f,0.0f, 0.0f);
    glColor3f(1.0f,0.0f,0.0f);
    glBegin(GL_TRIANGLE_FAN);
    glVertex2f(0, 0);
    for(double angle = 0.0; angle<=360.0;angle++)
```

```
    {
        glVertex2f(0 + sin(angle) * AlleyDistance*scaling, 0 +
                   cos(angle) * AlleyDistance*scaling);
    }
    glEnd();
    glLoadIdentity();
    glTranslatef( 0.0f,0.0f, 0.0f);
    glColor3f(0.0f,0.0f,0.0f);
    glBegin(GL_TRIANGLE_FAN);
    glVertex2f(0, 0);
    for(double angle = 0.0; angle<=360.0;angle++)
    {
        glVertex2f(0 + sin(angle) * (AlleyDistance-0.0002)*scaling, 0 +
                   cos(angle) * (AlleyDistance-0.0002)*scaling);
    }
    glEnd();

    //Circular membrane
    glLoadIdentity();
    glTranslatef( 0.0f,0.0f, 0.0f);
    glColor3f(0.0f,1.0f,0.0f);
    glBegin(GL_TRIANGLE_FAN);
    glVertex2f(0, 0);
    for(double angle = 0.0; angle<=360.0;angle++)
    {
        glVertex2f(0 + sin(angle) * PopDistance*scaling, 0 +
                   cos(angle) * PopDistance*scaling);
    }
    glEnd();

    //Black Circle
    glLoadIdentity();
    glTranslatef( 0.0f,0.0f, 0.0f);
    glColor3f(0.0f,0.0f,0.0f);
    glBegin(GL_TRIANGLE_FAN);
    glVertex2f(0, 0);
    for(double angle = 0.0; angle<=360.0;angle++)
    {
        glVertex2f(0 + sin(angle) * MembraneStartingPosition*scaling, 0 +
                   cos(angle) * MembraneStartingPosition*scaling);
    }
    glEnd();

    // show the flotor as a blue dot
    glLoadIdentity();
    glTranslatef( pos_flotor*scaling,pos_flotorZ*scaling, 0.0f);
    glColor3f(1.0f,0.0f,1.0f);
    glBegin(GL_TRIANGLE_FAN);
    glVertex2f(0, 0);
    for(double angle = 0.0; angle<=360.0;angle++)
    {
        glVertex2f(0 + sin(angle) * 0.01, 0 + cos(angle) * 0.01);
    }
    glEnd();

    // display the prompt
    glLoadIdentity();
    sprintf(str2,"Press:");
    glColor3f(1.0f,1.0f,1.0f);
    glRasterPos3f(-1,1.0-60.0/600.0, 0.0);
    for(i = 0; i<strlen(str2); i++)
        glutBitmapCharacter(GLUT_BITMAP_9_BY_15, str2[i]);

    sprintf(str2,"+ Increase Spring Stiffness");
    glRasterPos3f(-1,1.0-120.0/600.0, 0.0);
    for(i = 0; i<strlen(str2); i++)
        glutBitmapCharacter(GLUT_BITMAP_9_BY_15, str2[i]);

    sprintf(str2,"- Decrease Spring Stiffness");
    glRasterPos3f(-1,1.0-180.0/600.0, 0.0);
    for(i = 0; i<strlen(str2); i++)
        glutBitmapCharacter(GLUT_BITMAP_9_BY_15, str2[i]);

    sprintf(str2,"ESC or q Quit");
    glRasterPos3f(-1,1.0-240.0/600.0, 0.0);
    for(i = 0; i<strlen(str2); i++)
        glutBitmapCharacter(GLUT_BITMAP_9_BY_15, str2[i]);


    float F1,F2;
    //if(0==(TickCounter%500)){
        F1=curr_Force.values[0];
        F2=curr_Force.values[2];
    //}
    sprintf(str2,"Force= %f",sqrt(pow(ForceX,float(2.0)) + pow(ForceZ,float(2.0))));
    glRasterPos3f(-1,1.0-300.0/600.0, 0.0);
    for(i = 0; i<strlen(str2); i++)
        glutBitmapCharacter(GLUT_BITMAP_9_BY_15, str2[i]);

    sprintf(str2,"SpringConstant, StartTaskFlag = %f, %d",SpringConstant, StartTaskFlag);
    glRasterPos3f(-1,1.0-360.0/600.0, 0.0);
    for(i = 0; i<strlen(str2); i++)
        glutBitmapCharacter(GLUT_BITMAP_9_BY_15, str2[i]);
#endif
```

```
    glLoadIdentity();

    // Visualisation DURING trial
    if(EndofTrialFlag == 1)
    {
        //Display Push/Pull
        if (CurrentPushOrPull == 1){
            sprintf(str2,"PUSH");
            glClearColor(1.0f,0,0,0);}
        else if (CurrentPushOrPull == 2){
            sprintf(str2,"PULL");
            glClearColor(0.0f,0,1.0f,0);}
        else{
            sprintf(str2,"PUSH/PULL Undefined");
            glColor3f(0.0f,0.0f,1.0f);}

        glColor3f(1.0f,1.0f,1.0f);
        glRasterPos3f(0.0,1.0-240.0/600.0, 0.0);
        for(i = 0; i<strlen(str2); i++)
            glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24, str2[i]);
    }

    // Visualisation at END of trial
    else if(EndofTrialFlag == 0){
        if(!BeepCompleted){
            printf("\a\a");
            BeepCompleted = 1;
        }
        sprintf(str2,"Please release the handle");
        glClearColor(0.0f,0,0,0);
        glLoadIdentity();
        glColor3f(1.0f,1.0f,1.0f);
        glRasterPos3f(0.0,1.0-240.0/600.0, 0.0);
        for(i = 0; i<strlen(str2); i++)
            glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24, str2[i]);
    }

    // Visualisation at START of trial
    else if(EndofTrialFlag == 2){
        if(!BeepCompleted){
            printf("\a");
            BeepCompleted = 1;
        }
        sprintf(str2,"SET");
        glLoadIdentity();
        glColor3f(1.0f,1.0f,1.0f);
        glRasterPos3f(0.0,1.0-240.0/600.0, 0.0);
        for(i = 0; i<strlen(str2); i++)
            glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24, str2[i]);
    }

    // display the scene
    glFlush();
    glutSwapBuffers();
    // trigger the display function again
    glutPostRedisplay();

}


#ifdef junk
char KeyPress;
    do
    {
        KeyPress = getch();
        switch(KeyPress)
        {
            case '+'://Increase stiffness
                SpringConstant += 50;
                break;
            case '-'://Decrease Stiffness
                SpringConstant -= 50;
                break;
            default:
                break;
        }
        if(SpringConstant < 50)
            SpringConstant = 50;
        else if(SpringConstant > 1000)
            SpringConstant = 1000;


    }while(KeyPress != 'q');

        //Horizontal Bar
    glLoadIdentity();
    glTranslatef( float(-0.002*scaling),float(0.0005*scaling),0.0f );
    glColor3f(0.0f,0.0f,1.0f);
    glBegin(GL_QUADS);
        glVertex3f(0.0f,0.0f,0.0f);//Top right
        glVertex3f(-(0.001*scaling),0.0f, 0.0f);//Top Left
        glVertex3f(-(0.001*scaling),-(0.002*scaling), 0.0f);//Bottom Left
        glVertex3f(0.0f,-(0.002*scaling), 0.0f);//Bottom right
    glEnd();
```

```
    //Sweet Spot
    glLoadIdentity();
    glTranslatef( float(-0.0025*scaling),-(0.00025*scaling),0.0f );
    glColor3f(0.1f,1.0f,0.1f);
    glBegin(GL_QUADS);
        glVertex3f(0.0f,0.0f,0.0f);//Top right
        glVertex3f(-(0.0005*scaling),0.0f, 0.0f);//Top Left
        glVertex3f(-(0.0005*scaling),-(0.0005*scaling), 0.0f);//Bottom Left
        glVertex3f(0.0f,-(0.0005*scaling), 0.0f);//Bottom right
    glEnd();
#endif
/* EOF ************************************************************************************/
```

C.4 HHFM PSYCHOPHYSICS PLATFORM SCHEMATICS



C.5 HHFM PSYCHOPHYSICS PLATFORM CONTROL SOFTWARE

```
/*****************************************************************************
 *  Randy Lee
 *  Visualization and Image Analysis Lab
 *  Department of Bioengineering
 *  University of Pittsburgh
 *
 *  Adapted from
 *  Cindy Wong (cyw5@pitt.edu)
 *  VIA Lab, University of Pittsburgh
 *
 *  Minimum Force Contact Experiment -- Force Sensor Control
 *
 *  Code version: v01
 *
 *  Description and notes:
 *      Controlling an Analog Devices ADuC7026 microprocessor that is connected
 *  to a precise force sensor (Transducer Techniques GS0-100) and the HHFM
 *  ADuC7026 microprocessor. In response to applied forces to the GS0-100, LEDs
 *  are illuminated. The experiment asks subjects to ramp up to some
 *  suprathreshold force, and then ramp down to the smallest possible force they
 *  can generate.
 *      This code is adapted from the Isometric Low Force Control code.
 *
 *
 *  Pin and Port Assignments (identical to Isometric Low Force Experiment):
 *      GPIO P0.0 - 0.4:    Current trial status (output)
 *      GPIO P1.0 - 1.2:    Trial Feedback LEDs (output)
 *      GPIO P2.0 - 2.6:    Front panel push buttons (input)
 *      GPIO P3.0 - 3.6:    Front panel block LEDs (output)
 *      GPIO P4.0 - 4.7:    N/C
 *
 *      ADC0: Force sensor (GS0-100)
 *
 *      timer0:     200us time-keeping period (5kHz base operating frequency)
 *      timer1:     variable us & ms delays
 *
 *  ============================================================================
 *  Detailed Pin/Port assignments --
 *
 *  Experiment ADUC7026 Analog INPUT:
 *      - ADC0 : FS0-100 Force sensor; from ext. hardware
 *
 *  Experiment ADUC7026 Analog OUTPUT:
 *      - DAC0 : N/C
 *
 *  Experiment ADUC7026 Digital INPUT:
 *      - P2.0 : Trial block TRAINING; from ext. hardware (push button, YLW LED)
 *      - P2.1 : Trial block 1; from ext. hardware (push button, red LED)
 *      - P2.2 : Trial block 2; from ext. hardware (push button, red LED)
 *      - P2.3 : Trial block 3; from ext. hardware (push button, red LED)
 *      - P2.4 : Trial block 4; from ext. hardware (push button, red LED)
 *      - P2.5 : Trial block 5; from ext. hardware (push button, red LED)
 *      - P2.6 : Block start; from ext. hardware (black push button)
 *      - P2.7 : Threshold felt; from ext. hardware (external push button)
 *
 *  Experiment ADUC7026 Digital OUTPUT:
 *      - P0.0 : Trial number bit 0; to NI-6009 AI0.3
 *      - P0.1 : Trial number bit 1; to NI-6009 AI0.4
 *      - P0.2 : Trial number bit 2; to NI-6009 AI0.5
 *      - P0.3 : Magnification bit0; to HHFM ADUC7026 P0.1
 *      - P0.4 : Magnification bit1; to HHFM ADUC7026 P0.2
 *                          (i.e. bit0,bit1 -- 00 = 0x; 01 = 3x; 10 = 10x)
 *      - P0.5 : Currently in trial; to NI-6009 AI0.7 & HHFM ADUC7026 P0.0
 *                          (i.e. HIGH = before/after trial; LOW = currently in trial)
 *      - P0.6 : Trial start/stop buzzer; to ext. hardware
 *
 *      - P1.0 : Trial direction LED1 (pull, RED); to ext. hardware
 *      - P1.1 : Trial direction LED2 (in-range, YEL); to ext. hardware
 *      - P1.2 : Trial direction LED3 (push, GRN); to ext. hardware
 *
 *      - P3.0 : Trial block TRAINING LED; to ext. hardware (YLW)
 *      - P3.1 : Trial block 1 LED; to ext. hardware (RED)
 *      - P3.2 : Trial block 2 LED; to ext. hardware (RED)
 *      - P3.3 : Trial block 3 LED; to ext. hardware (RED)
 *      - P3.4 : Trial block 4 LED; to ext. hardware (RED)
 *      - P3.5 : Trial block 5 LED; to ext. hardware (RED)
 *      - P3.6 : Trial block START LED; to ext. hardware (GRN)
 *
 *  ============================================================================
 *  Change log:
 *
 *  v01 -   Branched from ForceSensor.c (v22)
 *          Chnaged experiment to minimum contact
 *                          Added timer0 disable/re-enable during delay_ms() to avoid conflicts
 *
 *
 *  Last updated 02 October 2016
```

```
 ******************************************************************************/

/* Libraries ------------------------------------------------------------------ */
#include <stdio.h>
#include <aduc7026.h>
#include <math.h>

/* Defines -------------------------------------------------------------------- */
#define OBSOLETE 0
#define TEMP_DISABLED 0
#define NOT_YET_IMPLEMENTED 0

#define NUM_TRIAL 6

/* Global variables ----------------------------------------------------------- */
double g_calibration_slope = (0xFFF/200);

int g_init_FSVoltage;
int g_init_FSVoltageHex;

int g_curr_FSVoltage;
int g_curr_FSVoltageHex;

int g_target_FSVoltage;
int g_target_FSVoltageHex;

int g_targetHalf_FSVoltage;
int g_targetHalf_FSVoltageHex;

/* Flags ---------------------------------------------------------------------- */
volatile unsigned int timer0_Flag, timer1_Flag = 0;

int g_flag = 1;

/* Randomized trial blocks ---------------------------------------------------- */
/* Each trial permutation is given by an ID number, which is recorded in
 * binary by the digital output pins of the experimental system.
 *
 * Conditions: HHFM gains (0x, 3x, 10x); Direction (push/pull)
 */
int block = 0;

int block1[6] = {5,2,1,3,6,4};
int block2[6] = {4,6,2,3,5,1};
int block3[6] = {6,3,2,5,4,1};
int block4[6] = {2,6,1,4,3,5};
int block5[6] = {1,2,5,6,3,4};

int blockTRAIN[6] = {4,2,3,6,5,1};

/* Functions ------------------------------------------------------------------ */
/* system initialization */
void program_init(void);
void timer0_init(void);
void General_IRQ_Handler(void);

/* hardware interface functions */
void beep(void);
float ADC_hex_to_float(int ADC_hex);
int DAC_float_to_hex(float desired_voltage);
int tare_channel(int channel);
int read_ADC_channel(int channel);

/* experiment functions */
int checkTrialButton(void);
void startTrial(int status);
void trial_block(int block);
int trialSetUp(int trial_ID);
void set_GPIO(int trial_ID);
void set_direction(int direction);
void set_mag(int gain);
int waitForStart(void);
int checkCurrentVoltage(void);
void min_contact_trial(void);

/* timing functions */
void delay_us(unsigned long time);
void delay_ms(unsigned long time);
void wait(void);
void waitForRestOfPeriod(void);


/* Main program --------------------------------------------------------------- */
int main(void)
{
    /* initial software wait to avoid JTAG lock */
    wait();

    /* initialize CPU and hardware pins */
    program_init();

    /* enable timer0 and timer1 interrupts, specify interrupt handler */
    IRQEN = 0x04;
    IRQ = General_IRQ_Handler;
```

```
    wait();

    /* pulse currentlyInTrial LOW to force HHFM into experimentMode() */
    startTrial(1);

    /* 100 sample mean, calculated as mean of hex values from GS0-100 */
    g_init_FSVoltageHex = tare_channel(0);
    delay_ms(2000);
    startTrial(0);

    /* turn off all trial LEDs */
    GP1DAT = 0xFF000000;
    GP3DAT = 0xFF000000;

    /* Main loop ------------------------------------------------------------ */
    while(1)
    {
        /* select trial based on front panel switch press */
        block = checkTrialButton();

        trial_block(block);

        /* reset flags and other global variables after block done*/
        block = 0;
    }
}
/* End of Main ---------------------------------------------------------- */

/* Function definitions ------------------------------------------------- */

/* experiment functions */
void startTrial(int status)
{
    if (status == 0)
    {
        /* two beeps to signify the end of the trial */
        beep();
        delay_ms(500);
        beep();

        /* flip P0.5 HIGH at end of trial */
        /* flip after audio feedback since HHFM now uses currInTrial to control
         * magnification
         */
        GP0DAT |= 1<<21;
    }
    else if (status == 1)
    {
        /* flip P0.5 LOW at beginning of trial */
        GP0DAT &= ~(1<<21);

        /* one beep to signify the start of the trial */
        beep();
    }
    else if (status == 3)
    {
        /* three beeps to signify the end of the block */
        beep();
        delay_ms(500);
        beep();
        delay_ms(500);
        beep();
    }

    return;
}

int checkTrialButton(void)
{
    int blockChosen = 0;

    while (blockChosen == 0){
        if ((GP2DAT & 32) == 0)
        {
            /* Exp P2.5 button pressed, light P3.5 LED (RED) */
            GP3DAT = 0xFF200000;
            delay_ms(500);
            blockChosen = 5;
        }
        else if ((GP2DAT & 16) == 0)
        {
            /* Exp P2.4 button pressed, light P3.4 LED (RED) */
            GP3DAT = 0xFF100000;
            delay_ms(500);
            blockChosen = 4;
        }
        else if ((GP2DAT & 8) == 0)
        {
            /* Exp P2.3 button pressed, light P3.3 LED (RED) */
            GP3DAT = 0xFF080000;
            delay_ms(500);
            blockChosen = 3;
        }
        else if ((GP2DAT & 4) == 0)
        {
```

```
            /* Exp P2.2 button pressed, light P3.2 LED (RED) */
            GP3DAT = 0xFF040000;
            delay_ms(500);
            blockChosen = 2;
        }
        else if ((GP2DAT & 2) == 0)
        {
            /* Exp P2.1 button pressed, light P3.1 LED (RED) */
            GP3DAT = 0xFF020000;
            delay_ms(500);
            blockChosen = 1;
        }
        else if ((GP2DAT & 1) == 0)
        {
            /* Exp P2.0 button pressed, light P3.0 LED (YLW) */
            GP3DAT = 0xFF010000;
            delay_ms(500);
            blockChosen = 99;
        }
    }

    return blockChosen;
}

void trial_block(int block)
{
    int i = 0;
    int currentTrial;
    int numTrial;
    int trial_set[6] = {0};

    /* copy trial IDs */
    if (block == 1)
    {
        for (i=0;i<NUM_TRIAL;i++)
          trial_set[i] = block1[i];
    }
    else if (block == 2)
    {
        for (i=0;i<NUM_TRIAL;i++)
          trial_set[i] = block2[i];
    }
    else if (block == 3)
    {
        for (i=0;i<NUM_TRIAL;i++)
          trial_set[i] = block3[i];
    }
    else if (block == 4)
    {
        for (i=0;i<NUM_TRIAL;i++)
          trial_set[i] = block4[i];
    }
    else if (block == 5)
    {
        for (i=0;i<NUM_TRIAL;i++)
          trial_set[i] = block5[i];

    }
    else if (block == 99)
    {
        for (i=0;i<NUM_TRIAL;i++)
          trial_set[i] = blockTRAIN[i];
    }

    /* waitForStart() includes while loop to wait until confirmation/escape
     * button is pressed
     */
    if (waitForStart())
    {
        for (numTrial=0; numTrial<NUM_TRIAL; numTrial++){
            /* turn off LEDs at beginning of trial */
            GP1DAT = 0xFF000000;
            GP3DAT = 0xFF000000;

            /* set mag/direction & get trial ID from randomized block array */
            currentTrial = trialSetUp(trial_set[numTrial]);

            /* set trial number in binary representation on GPIO P0.0-0.2 */
            set_GPIO(currentTrial);
            delay_ms(500);

            /* START TRIAL -- beep once, set currently in trial pin LOW */
            startTrial(1);

            /* restart 200us device timer, start low contact trial
             * turns on LEDs based on direction indicated, until subject hits 5g
             * turns off LEDs after target force reached
             */
            min_contact_trial();

            /* wait an additional 6s after visual feedback period, collecting
             * more data
             */
            delay_ms(6000);
```

```
            /* beep twice at end of the trial, sets "currently in trial"
             * pin HIGH
             */
            startTrial(0);

            /* all LEDs on briefly to show end of trial */
            GP1DAT = 0xFFFF0000;
            GP3DAT = 0xFFFF0000;

            /* wait 5 seconds between trials */
            delay_ms(5000);
        }

        /* three beeps to signify the end of trial block */
        startTrial(3);
    }

    /* clear LEDs in preparation for the next trial block */
    GP1DAT = 0xFF000000;
    GP3DAT = 0xFF000000;

    return;
}

void min_contact_trial(void)
{
    int current_status, target_reached = 0;

    /* start 200us timer */
    timer0_init();

    while (target_reached == 0){
          /* check current force sensor voltage against target */
        current_status = checkCurrentVoltage();

        /* target reached when current status is 2 */
        if (current_status == 2) target_reached = 1;

        waitForRestOfPeriod();
    }

    if (target_reached == 1)
    {
        delay_ms(250);
        beep();

        /* when target reached, turn off all LEDs & continue collecting data */
        GP1DAT &= ~(1<<16);
        GP1DAT &= ~(1<<17);
        GP1DAT &= ~(1<<18);
    }

    return;
}

int checkCurrentVoltage(void)
{
    /* status = 0, below half way;
     * status = 1, at or above half way;
     * status = 2, at or above full target;
     */
    int status = 0;

    g_curr_FSVoltageHex = read_ADC_channel(0);

    /* push direction */
    if (g_init_FSVoltageHex > g_target_FSVoltageHex)
    {
        if (g_curr_FSVoltageHex > g_targetHalf_FSVoltageHex)
        {
            /* below half way point, leave RED LED @ P1.4 on */
            GP1DAT &= ~(1<<16);
            GP1DAT &= ~(1<<17);
            GP1DAT |= 1<<18;
            status = 0;
        }
        else if (g_curr_FSVoltageHex <= g_targetHalf_FSVoltageHex &&
                 g_curr_FSVoltageHex > g_target_FSVoltageHex)
        {
            /* above half way point, but not at full target
             * turn on RED (1.4) and YLW (1.3) LEDs
             */
            GP1DAT &= ~(1<<16);
            GP1DAT |= 1<<17;
            GP1DAT |= 1<<18;
            status = 1;
        }
        else if (g_curr_FSVoltageHex <= g_target_FSVoltageHex)
        {
            /* at or above target, turn on all LEDs */
            GP1DAT |= 1<<16;
            GP1DAT |= 1<<17;
            GP1DAT |= 1<<18;
            status = 2;
        }
```

```
    }

    /* pull direction */
    else if (g_init_FSVoltageHex < g_target_FSVoltageHex)
    {
        if (g_curr_FSVoltageHex < g_targetHalf_FSVoltageHex)
        {
            /* below half way point, leave GRN LED @ P1.2 on */
            GP1DAT |= 1<<16;
            GP1DAT &= ~(1<<17);
            GP1DAT &= ~(1<<18);
            status = 0;
        }
        else if (g_curr_FSVoltageHex >= g_targetHalf_FSVoltageHex &&
                 g_curr_FSVoltageHex < g_target_FSVoltageHex)
        {
            /* above half way point but not at full target
             * turn on GRN (P1.2) & YLW (P1.3) LEDs
             */
            GP1DAT |= 1<<16;
            GP1DAT |= 1<<17;
            GP1DAT &= ~(1<<18);
            status = 1;
        }
        else if (g_curr_FSVoltageHex >= g_target_FSVoltageHex)
        {
            /* at or above target, turn on all LEDs */
            GP1DAT |= 1<<16;
            GP1DAT |= 1<<17;
            GP1DAT |= 1<<18;
            status = 2;
        }
    }

    return status;
}

int trialSetUp(int trial_ID)
{
    int returnGPIO = 0;

    /* reset magnification before loading each trial */
    set_mag(0);

    /* set trial parameters based on ID number.
     * direction: -1 = push; +1 = pull
     * magnification: 0x, 3x, 10x
     */
    if (trial_ID == 1)
    {
        set_direction(-1);
        set_mag(0);

        returnGPIO = 0x00;
    }
    else if (trial_ID == 2)
    {
        set_direction(-1);
        set_mag(2);

        returnGPIO = 0x01;
    }
    else if (trial_ID == 3)
    {
        set_direction(-1);
        set_mag(4);

        returnGPIO = 0x02;
    }
    else if (trial_ID == 4)
    {
        set_direction(1);
        set_mag(0);

        returnGPIO = 0x03;
    }
    else if (trial_ID == 5)
    {
        set_direction(1);
        set_mag(2);

        returnGPIO = 0x04;
    }
    else if (trial_ID == 6)
    {
        set_direction(1);
        set_mag(4);

        returnGPIO = 0x05;
    }

    /* training block trial parameters */
    else if (trial_ID == 7)
    {
          set_direction(-1);
```

```
          set_mag(7);

          returnGPIO = 0x06;
    }
    else if (trial_ID == 8)
    {
          set_direction(1);
          set_mag(7);

          returnGPIO = 0x07;
    }

    return returnGPIO;
}

void set_direction(int direction)
{
    /* direction = -1, push; direction = 1, pull */
    g_target_FSVoltageHex = (int)(g_calibration_slope * direction * 15.0) +
                            g_init_FSVoltageHex;

    g_targetHalf_FSVoltageHex = (int)(g_calibration_slope * direction * 7.5) +
                                g_init_FSVoltageHex;

    return;
}

void set_mag(int gain)
{
    /* Exp P0.3 = bit0, P0.4 = bit1; connect to HHFM P0.1 & P0.2 */

    if (gain == 0)
    {
        /* set P0.3 & P0.4 == 0 */
        GP0DAT &= ~(1<<19);
        GP0DAT &= ~(1<<20);
    }
    else if (gain == 2)
    {
        /* set P0.3 == 1, P0.4 == 0 */
        GP0DAT |= 1<<19;
        GP0DAT &= ~(1<<20);
    }
    else if (gain == 4)
    {
        /* set P0.3 == 0, P0.4 == 1 */
        GP0DAT &= ~(1<<19);
        GP0DAT |= 1<<20;
    }
    /* training block gain */
    else if (gain == 7)
    {
          /* set P0.3 == 1, P0.4 == 1 */
          GP0DAT |= 1<<19;
          GP0DAT |= 1<<20;
    }

    return;
}

void set_GPIO(int trial_ID)
{
          /* clear P0.0 - 0.2 */
          GP0DAT &= ~(0x7 << 16);

          /* mast INT trial_ID with 3 bits (0x7), left shift input by 16 bits
           * REMEMBER: trial_ID = actualTrialNum - 1
           */
          GP0DAT |= (trial_ID & 0x7) << 16;

          return;
}

int waitForStart(void)
{
    while (1){
        /* if P2.6 pressed, light GRN LED and start trial */
        if ((GP2DAT & 64) == 0)
        {
            GP3DAT |= 0xFF400000;
            delay_ms(500);
            return 1;
        }

        /* if P2.0 pressed, return 0 and reset to main */
        if ((GP2DAT & 1) == 0)
        {
            GP3DAT = 0xFF010000;
            delay_ms(500);
            return 0;
        }
    }
}

/* system initialization */
```

```
void program_init(void)
{
    /* configure CPU clock for 41.78MHz, CD = 0 */
    POWKEY1 = 0x01;
    POWCON = 0x00;
    POWKEY2 = 0xF4;

    PLLKEY1 = 0xAA;
    PLLCON = 0x01;
    PLLKEY2 = 0x55;

    /* Connect internal 2.5V reference to V_ref pin
     * set DAC0 output range to 0-V_ref and turn on
     */
    REFCON = 0x01;
    DAC0CON = 0x12;

    /* Trial Data Outputs (Trial #, Magnification State) */
    /* set Port 0 as GPIO OUTPUT on all pins, initialize all pins HIGH */
    GP0CON = 0x00;
    GP0DAT = 0xFFFF0000;

    /* Initialize Visual Stimulus LEDs */
    /* set Port 1 as GPIO OUTPUT on all pins, initialize all pins HIGH */
    GP1CON = 0x00;
    GP1DAT = 0xFFFF0000;

    /* Front Panel trial block swtich inputs */
    /* set Port 2 as GPIO INPUT on all pins, initialize all pins HIGH */
    GP2CON = 0x00;
    GP2DAT = 0x00;

    /* Front Panel trial block LEDs */
    /* set Port 3 as GPIO OUTPUT on all pins, initialize all pins HIGH */
    GP3CON = 0x00;
    GP3DAT = 0xFFFF0000;

    /* HHFM State outputs */
    /* set Port 4 as GPIO OUTPUT on all pins, initialize all pins HIGH */
    GP4CON = 0x00;
    GP4DAT = 0xFFFF0000;

    /* Initialize ADC, wait to allow components to power on */
    ADCCON = 0x20;
    wait();

    return;
}

void timer0_init(void)
{
    /* timer0 currently set to 200us period */

    /* (re)-initialize flag & IRQ */
    timer0_Flag = 0;
    IRQEN |= 0x04;

    /* load timer0 counter
     * 0x1053 = 100us period @ 41.78MHz clock freq (set in program_init())
     * 0x20A6 = 200us; 0x30F9 = 300us
     */
    T0LD = 0x20A6;

    /* start timer0, set in periodic mode with multiplier = 1 */
    T0CON = 0xC0;

    return;
}

void General_IRQ_Handler(void)
{
    /* timer0 IRQ clear for 200us period */
    if ((IRQSIG & 0x4) == 0x4)
    {
        timer0_Flag = 1;
        T0CLRI = 0xFF;
        return;
    }

    /* timer1 IRQ clear for us/ms delay functions */
    else if ((IRQSIG & 0x8) == 0x8)
    {
        timer1_Flag = 1;
        T1CLRI = 0xFF;
        return;
    }
}

/* ADuC hardware interface functions */
void beep(void)
{
    /* buzzer on P0.6, drop LOW to sound */
    GP0DAT &= ~(1<<22);

    /* buzzer on for 300ms */
```

```
    delay_ms(300);

    /* flip P0.6 HIGH to end buzzer */
    GP0DAT ^= 1<<22;

    return;
}

float ADC_hex_to_float(int ADC_hex)
{
    /* HHFM function -- ADCinput(). Convert ADC hex result to decimal voltage */
    float input_voltage;
    int ADC_hex_shifted = ADC_hex >> 16;

    input_voltage = ((ADC_hex_shifted * 2.52)/0xFFF);

    return input_voltage;
}

int DAC_float_to_hex(float desired_voltage)
{
    /* HHFM/1DoF function -- outputDAC(). Convert desired voltage to DAC hex */

    /* "unit conversion" from desired decimal voltage to hex
     * 2.4976V is voltage when 0xFFF applied to DAC
     */
    int output_integer = floor((desired_voltage*0xFFF)/2.497);

    /* add max & min integer caps, linear calibration function for low voltages
     */
    if (desired_voltage < 2.49 && desired_voltage >= 0.036)
    {
        /* shift integer result 16 bits to left to conform to DACxDAT format */
        output_integer = output_integer << 16;
    }
    else if (desired_voltage >= 2.49)
    {
        output_integer = 0xFFF;
        output_integer = output_integer << 16;
    }
    else if (desired_voltage < 0.036 && desired_voltage >= 0.0065)
    {
        /* voltage under 0.036V need a seperate function for increased accuracy
         */
        desired_voltage = 1.2 * desired_voltage - 0.0072733;
        output_integer = floor((desired_voltage * 0xFFF)/2.497);
        output_integer = output_integer << 16;
    }
    else if (desired_voltage < 0.0065)
    {
        /* the minimum voltage we can get */
        output_integer = 0x00;
    }

    return output_integer;
}

int tare_channel(int channel)
{
    int i, voltageHex, resultHex = 0;
    unsigned long sum = 0;

    for (i=0; i<100; i++){
        voltageHex = read_ADC_channel(channel);
        sum += voltageHex;
        waitForRestOfPeriod();
    }

    resultHex = floor(sum/100.0);

    return resultHex;
}

int read_ADC_channel(int channel)
{
    /* ForceSensor.c function -- adc_voltage() */
    int voltage_hex = 0;

    /* select ADC channel */
    ADCCP = channel;

    /* start conversion, no 2nd conversion */
    ADCCON = 0x6A3;
    ADCCON &= ~(1 << 7);
    while (!ADCSTA){
        /* wait until conversion complete */
    }

    /* shift voltage to right 16 bits, so full scale is 0xFFF */
    voltage_hex = ADCDAT >> 16;

    return voltage_hex;
}

/* timing functions */
```

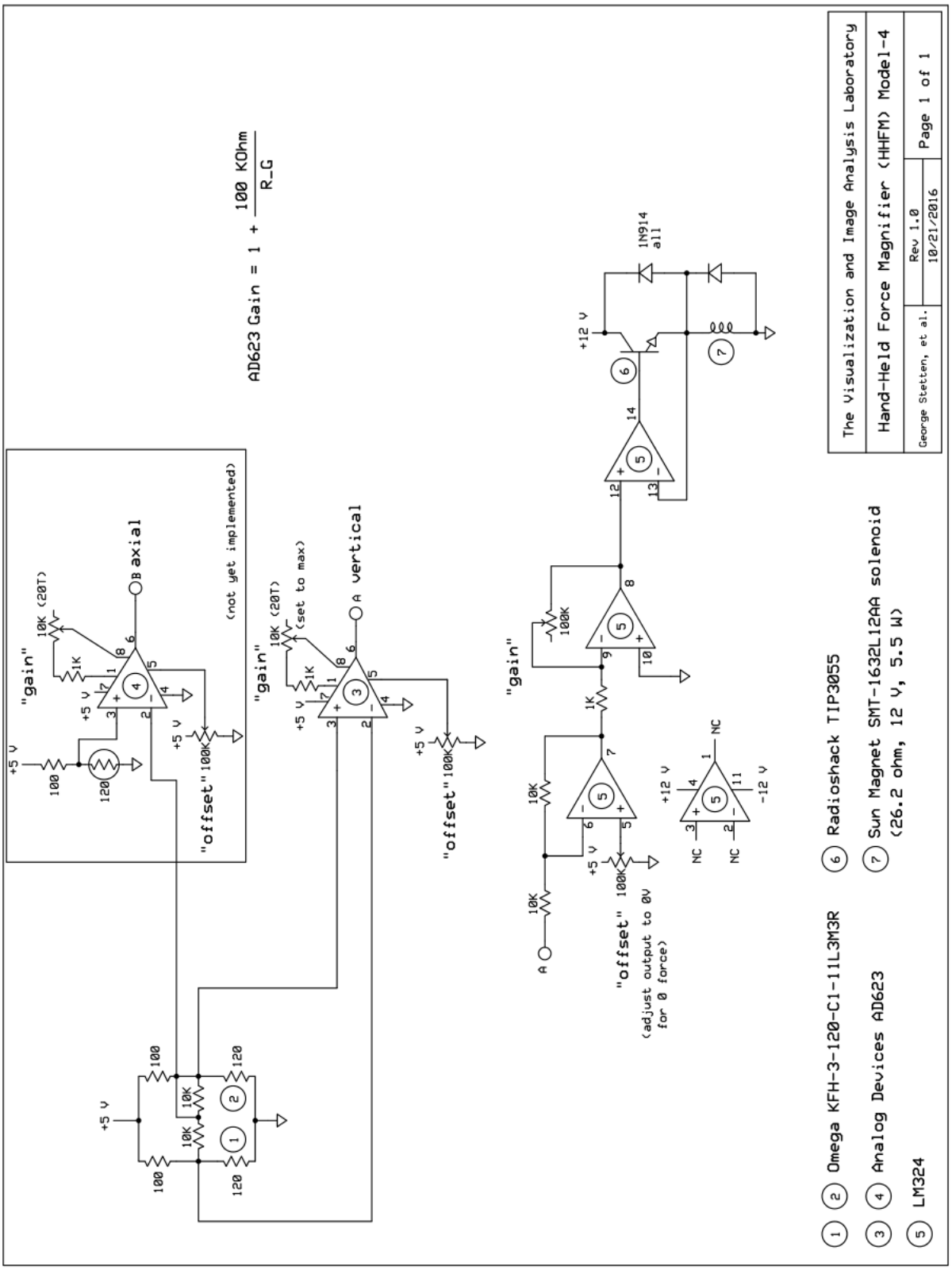
```
void wait(void)
{
    int t;
    for (t=0; t<1000000; t++){
        /* software wait */
    }

    return;
}

void delay_us(unsigned long time)
{
          /* reset globals and timer1 */
    timer1_Flag = 0;
    T1CON = 0x00;

    /* enable timer1 IRQ */
    IRQEN |= 0x08;

    /* T1LD = time(us) * CPU freq (MHz), (e.g. 500us * 41.78 MHz = 20890 - 1)
     *      -> subtract 1 since count includes zero
     * Initialize timer1, 41.78MHz, down mode, periodic
     */
    T1LD = ((time * 42782) >> 10) - 1;
    T1CON = 0xC0;

    while (timer1_Flag == 0){
        /* wait for timer1 flag flip */
    }
    timer1_Flag = 0;

    /* clear interrupt and turn off timer1 */
    IRQCLR |= 0x08;
    T1CON = 0x00;

    return;
}

void delay_ms(unsigned long time)
{
    unsigned long i = 0;

    /* disable timer0 to prevent interrupt confusion */
    T0CON = 0x00;

    /* wait for 1ms each time through loop */
    for (i=0; i<time; i++){
        delay_us(1000);
    }

    /* re-enable timer0 */
    timer0_init();

    return;
}

void waitForRestOfPeriod(void)
{
          /* maintains a strict operating frequency according to timer0 */
    if (timer0_Flag == 1)
    {
        /* timer0 has already finished, reset flag */
        timer0_Flag = 0;
        return;
    }

    else
    {
        while (timer0_Flag == 0){
            /* wait until interrupt request */
        }
        timer0_Flag = 0;
        return;
    }
}
/* EOF ------------------------------------------------------------------------ */
```

APPENDIX D

HHFM MODEL-4 SYSTEM SCHEMATICS

The HHFM Model-4 was powered and controlled by analog circuitry.

D.1 SCHEMATICS

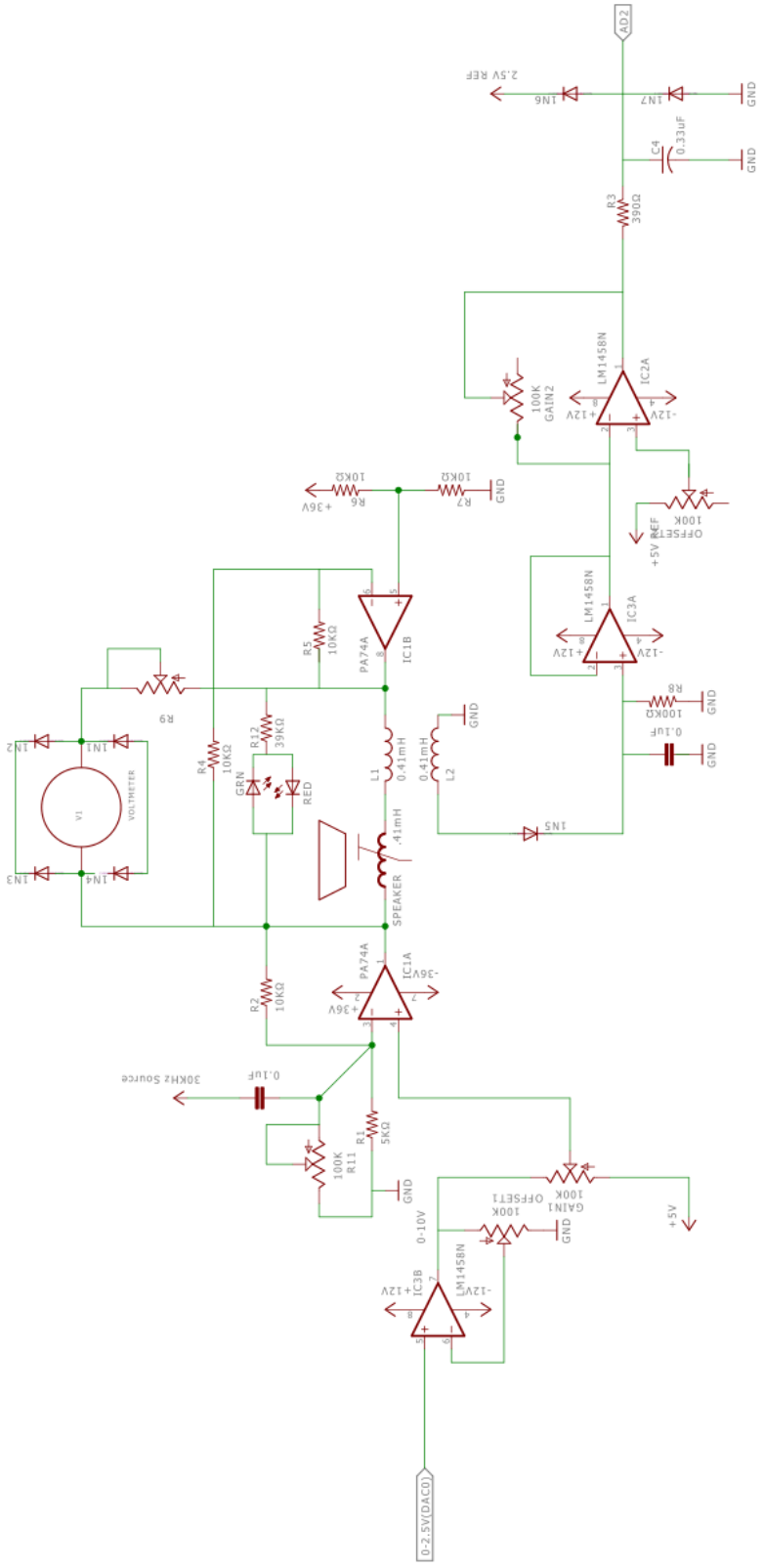


APPENDIX E

ONE-DIMENSIONAL HAPTIC RENDERER SYSTEM SCHEMATICS AND CONTROL SOFTWARE

The One-Dimensional Haptic Renderer ("1DoF") was powered by analog circuitry and controlled by an Analog Devices ADuC7026 microprocessor. The microprocessor interfaces with a Wixel programmable interface through SPI, which transmits and receives data between the microprocessor and a USB connected laptop.

E.1 ONE-DIMENSIONAL HAPTIC RENDERER SYSTEM SCHEMATICS



E.2 ONE-DIMENSIONAL HAPTIC RENDERER CONTROL SOFTWARE

E.2.1 ADuC7026 “Master” Source Code

```
/*****************************************************************************
Randy Lee (ral63@pitt.edu)
Avi Marcovici (avi.marcovici@gmail.com)
Avin Khera (avinkhera@gmail.com)
Visualization and Image Analysis Laboratory
Department of Bioengineering
University of Pittsburgh

1 DoF Haptic Renderer MCU Control

Equipment:
- Analog Devices ADuC7026
- FaitalPRO 5FE120 speaker
- Pololu Wixel programmable USB module

------------
Description and notes:

Controlling a commercial speaker in force and position to create a haptic
renderer. Virtual membranes of variable stiffness are created by combining
two functionalities: (1) a zero stiffness spring -- defies gravity by keeping
force on the speaker drum at zero; (2) an infinite stiffness spring -- virtual
wall to control both force and position. Infinite stiffness spring will also
enable position dependent calibration for force and current

-----------
Pin and Port Assignments:

ADC0 -- Honeywell FS01 Force Sensor
ADC1 -- Front panel knob 1
ADC2 -- Front panel knob 2
ADC3 -- Front panel knob 3
ADC4 -- IR proximity (optical) sensor
ADC5 -- Speaker impedence circuit output
ADC6 -- Desired speaker position, from 10 turn pot

DAC0 -- Speaker output

GPIO P0.x -- Digital output
GPIO P1.x -- SPI communication
GPIO P2.x -- Digital output


Last updated: 19 September 2016
*****************************************************************************/

/* Libraries ------------------------------------------------------------- */
#include <aduc7026.h>
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <string.h>

/* Defines --------------------------------------------------------------- */
#define OBSOLETE 0
#define NOT_YET_IMPLEMENTED 0
#define TEMP_DISABLED 0
#define PI 3.14159265358979323846264338327950288

/* Constants ------------------------------------------------------------- */
const float g_sample_time = 0.001;    /* sample time in seconds */
const float g_epsilon = 0.02;         /* arbitrary epsilon */


/* Global variables ------------------------------------------------------ */
/* 1DoF device data */
float force_volts, force_volts_avg, force_preload;
float opticalPos_volts, opticalPos_mm;
float inductPos_volts, inductPos_mm;
int speaker_voltageHex;
float speaker_volts;
float desiredSpeaker_volts, lastDesiredSpeaker_volts;
float desiredSpeaker_mm, lastDesiredSpeaker_mm;
float rest_pos_volts, rest_pos_mm;

unsigned char g_speaker_mode = 0x99;
int g_virtual_knob1, g_virtual_knob2, g_virtual_knob3 = 0;

/* each message received from Wixel/GUI is 1 byte in size */
unsigned char SPI_received_data[14];
```

```
unsigned char *pSPI_received;

/* linear fit variables for zero stiffness spring
 * @TODO: Confirm -- OBSOLETE AS OF VXX (?)
 */
float forceSlope, intersectionPoint;

/* counter for SPI comm testing */
int counterTest_message = 0xA00;

/* testing for 8 data comms */
char g_loop_count = 0x00;

/* define struct of three 12 bit messages to experiment with SPI sending */
struct Wixel_test{
    int Comp_1;
    int Comp_2;
    int Comp_3;
};

/* structure of 3 integers for testing the Wixel */
struct Wixel_test test_messages, *pTest_msg;

/* define struct for actual messages to be sent through wixel.
 * members are integers beause they are saved in read_store_sensors() a
 * shifted format from ADC/DAC raw results
 */
struct Wixel_msg{
    int forceSensor;
    int opticalSensor;
    int inductanceSensor;
    int speakerOut;
    int knob1;
    int knob2;
    int knob3;
    int tenTurnPot;
};

/* structure of 8 data points to send through Wixel */
struct Wixel_msg curr_data, *pCurr_data;
struct Wixel_msg init_data, *pInit_data;
struct Wixel_msg test_data, *pTest_data;

/* define struct for PID constants */
struct PID_const{
    float pGain;
    float iGain;
    float dGain;
    float K_u;
    float T_u;
    float setPoint;
    float errorSum;
    float T_f;
    float K_t;
    float D_1;
    float D_2;
    float lastInput;
    float lastOutput;
    float lastError;
    float lastErrorDerivative;
    float lastPos;
    float speakerBias;
};

/* PID constants for various simulations*/
struct PID_const PID_zeroStiff, *pPID_zero;
struct PID_const PID_wall, *pPID_wall;
struct PID_const PID_membrane, *pPID_mem;
struct PID_const PID_ratchet, *pPID_ratchet;

/* define struct for wave generators */
struct Wave_const{
    int curr_theta;
    int delta_theta;
    int theta_counter;
    float amplitude;
    float frequency;
};

/* structure of wave constants */
struct Wave_const Sine_const, *pSine_const;


/* define general structure for calibration equations */
struct calibration{
    float x;
    float slope;
    float intercept;
};

/* calibration constants */
float cal_x[20] = {0.0, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0,
                   5.5, 6.0, 6.5, 7.0, 7.5, 8.0, 8.5, 9.0, 9.5};

float cal_m_DACtoForce[20] = {2016.72133, 1297.62685, 998.28008, 1056.75399,
```

```
                            1155.43665, 1157.18334, 1093.51827, 1085.22422,
                            997.5204, 1087.64422, 1193.76108, 1042.93087,
                            1081.84914, 1064.50462, 1134.40323, 1094.02751,
                            983.95872, 1060.6518, 1029.32366, 3877.46988};

float cal_b_DACtoForce[20] = {-5102.83037, -3034.56697, -2060.11869,
                            -1910.91469, -1849.27433, -1760.53447,
                            -1507.5878, -1408.14884, -1221.23293,
                            -1226.91712, -1287.58321, -1059.07548,
                            -1100.62929, -994.60905, -1042.84995,
                            -810.34989, -725.26999, -714.35619, -701.41134,
                            -48.72909};

float cal_m_ForcetoDAC[20] = {2.92E-33, 7.21E-04, 9.34E-04, 8.98E-04, 8.46E-04,
                            8.27E-04, 8.88E-04, 8.80E-04, 9.78E-04, 8.85E-04,
                            8.17E-04, 9.32E-04, 8.75E-04, 9.11E-04, 8.22E-04,
                            8.96E-04, 9.82E-04, 9.28E-04, 9.39E-04, 1.68E-07};

float cal_b_ForcetoDAC[20] = {2.52, 2.3332, 2.05504, 1.80478, 1.5992, 1.51599,
                            1.37641, 1.29529, 1.22336, 1.12816, 1.07798,
                            1.0154, 1.01193, 0.93082, 0.90955, 0.74088,
                            0.7376, 0.67283, 0.68085, 0.00035};

/* array of structs to hold calibration equations */
struct calibration DAC_to_Force[20], *pDAC_to_Force;
struct calibration Force_to_DAC[20], *pForce_to_DAC;
struct calibration interpolated, *pInterpolated;


/* Flags, counters, and misc ------------------------------------------------ */
/* timer flags are volatile because they are flipped by interrupt routines */
volatile signed int timer0_Flag = 0;
volatile signed int timer1_Flag = 0;
int flag = 0;
int counter = 0;        /* loop counter */
int i = 0;              /* global loop counter */
//int message = 0;        /* SPI comm incoming message */

int g_membrane_punctured = 0;   /* flag for membrane puncture function */
int g_in_membrane_flag = 0;       /* currently in membrane flag */
int g_membrane_entry = 0;         /* 0=not entered, 1=entered top, 2=entered bot */
int g_ratchet_punctured = 0;    /* flag for ratchet function */
//int g_ratchet_level = 1;    /* counter for ratchet level */

/* array of status flags to see if a sensor has been polled this loop */
char sensor_status[8];


/* Function declarations ---------------------------------------------------- */
/* system and components init */
void General_IRQ_Handler(void);
void system_init(void);
void timer0_init(void);
void SPI_init(void);
void SPI_write(unsigned char data);
unsigned char SPI_read(void);
void ADCpoweron(int software_delay_time);

/* data update and formatting functions for 3 channel SPI comm testing */
void dataUpdate_testmsg(struct Wixel_test *);
void dataUpdate_testformat(struct Wixel_test *);

/* data update and formatting functions for 8 channel SPI comms testing */
void dataUpdate_msg(struct Wixel_msg *data);
void send_test_data(struct Wixel_test *);
void parse_SPI_data(unsigned char *received_data);

/* 1DoF device sensor data read/store/format for Wixel */
void initial_read_store_sensors(struct Wixel_msg *data);
float read_store_sensors(int channel, struct Wixel_msg *data);
void store_DAC_data(int rawDAC_hex, struct Wixel_msg *data);
void send_curr_data(struct Wixel_msg *data, unsigned char *received_data);
void check_sensor_status(void);

/* system behavior functions */
void testmode(int times);
float PID_zeroStiffness(struct PID_const *);
float PID_virtualWall(struct PID_const *);
float membrane_puncture(struct PID_const *, struct PID_const *,
                        struct PID_const *);
float ratchet_simulation(struct PID_const *, struct PID_const *,
                         struct PID_const *);
float sine_wave(struct Wave_const *);

/* sensor calibration/ manipulation functions */
float speaker_DACtoDisplacement(float volts);
float optical_ADCtoDisplacement(float volts);
float displacement_OpticalToADC(float displacement_mm);
float displacement_SpeakerToDAC(float displacement_mm);
float force_VoltsToGrams(float volts);
void interp_cal_constants(float curr_pos, struct calibration *cal_eqns,
                          struct calibration *result);
float sensor_avg(int channel);
```

```
/* Imported HHFM functions ------------------------------------------------ */
/* timer settings and delay functions */
/* initialize main loop period time keeping */
void timer0_init(void);

/* wait function at end of main loop to limit loop duration using timer0 */
void waitForRestOfPeriod(void);

/* variable software delay using decreasing for-loop */
void delay_sw(int time);

/* variable us delay using timer1 */
void delay_us(unsigned long time);

/* variable ms delay using timer1 */
void delay_ms(unsigned long time);

/* sensor and output functions */
/* converts raw hex result from ADC conversion to decimal voltage */
float ADCinput(int ADC_hex_result);

/* converts desired decimal voltage to hex integer for DAC output */
int outputDAC(float desired_voltage);

/* read voltage on desired channel (ADC#). returns decimal voltage */
float read_sensors(int channel);

/* scales potentiometer voltage by some fullscale value */
float pot_voltage(float voltage, float fullscaleValue);


/* Main program ----------------------------------------------------------- */
int main(void)
{
    /* software delay to avoid JTAG lock while prototyping */
    delay_sw(50000);

    /* Core clock and power state control */
    system_init();

    /* Power on ADC channels */
    ADCpoweron(20000);      // power on ADC

    /* Enable timer0 interrupt for computation-actuation loop timing */
    IRQEN = 0x04;               /* enable timer0 interrupt */
    IRQ = General_IRQ_Handler;  /* specify interrupt handler */

    /* Set up SPI on GPIO P1.4 - 1.7
     * SPI in master mode, clock pulses on xfer init
     * transmit on write to TX register
     * interrupt when TX empty
     */
    SPI_init();

                /* SPI received data pointer */
                pSPI_received = &SPI_received_data[0];

    /* testmode() generates a triangular increase/decrease in speaker voltage
     */
    //testmode(5);

    /* Initialize struct pointers */
    /* pTest_msg for updating test_msg struct using dataUpdate_testmsg() */
    pTest_msg = &test_messages;
    pTest_data = &test_data;

    /* ADC/DAC values that will be sent to Wixel or received from Wixel */
    pCurr_data = &curr_data;

    /* initial sensor data */
    pInit_data = &init_data;

    /* PID controller values for zero stiffness and virtual wall modes */
    pPID_zero = &PID_zeroStiff;
    pPID_wall = &PID_wall;
    pPID_mem = &PID_membrane;
    pPID_ratchet = &PID_ratchet;

    /* calibration equation pointers */
    pDAC_to_Force = &DAC_to_Force[0];
    pForce_to_DAC = &Force_to_DAC[0];
    pInterpolated = &interpolated;

    /* load calibration values into arrays of structs */
    for (counter = 0; counter < 20; counter++)
    {
        DAC_to_Force[counter].x = cal_x[counter];
        DAC_to_Force[counter].slope = cal_m_DACtoForce[counter];
        DAC_to_Force[counter].intercept = cal_b_DACtoForce[counter];

        Force_to_DAC[counter].x = cal_x[counter];
        Force_to_DAC[counter].slope = cal_m_ForcetoDAC[counter];
        Force_to_DAC[counter].intercept = cal_b_ForcetoDAC[counter];
    }
```

```
/* wave constants */
pSine_const = &Sine_const;
pSine_const->theta_counter = 0; /* initialize theta_counter */

/* initialize speaker output at 1.25 so sensors have a baseline */
speaker_voltageHex = outputDAC(1.25);
DAC0DAT = speaker_voltageHex;
store_DAC_data(speaker_voltageHex, pCurr_data);

/* one second delay to allow sensors to stabilize after power on/ reset */
delay_ms(1000);

/* initial read/store of all sensors */
initial_read_store_sensors(pInit_data);

/* calculate and save average voltage from the optical sensor */
rest_pos_volts = sensor_avg(4);
rest_pos_mm = optical_ADCtoDisplacement(rest_pos_volts);
//pInit_data->opticalSensor = rest_pos_volts;
pInit_data->opticalSensor = rest_pos_mm;

/* calculate and save the average voltage from the force sensor on ADC0 */
force_volts_avg = sensor_avg(0);
pInit_data->forceSensor = force_volts_avg;
force_preload = force_VoltsToGrams(force_volts_avg);


/* initialize K_u, T_u, and PID constants for virtual wall */
/*Tyreus Luyben PID control for virtual wall w/ slew rate limit
 * k_p = K_u/2.2; k_i = k_p/(2.2*T_u); k_d = (k_p*T_u)/6.3
 */
pPID_wall->K_u = 1.50;
pPID_wall->T_u = 0.025;
pPID_wall->pGain = pPID_wall->K_u/2.2;
pPID_wall->iGain = pPID_wall->pGain/(2.2*pPID_wall->T_u);
pPID_wall->dGain = (pPID_wall->pGain*pPID_wall->T_u)/6.3;
/* scale PID constants by sample time */
pPID_wall->iGain *= g_sample_time;
//pPID_wall->dGain /= g_sample_time;

/* initialize reference point and integral */
pPID_wall->setPoint = rest_pos_mm;
pPID_wall->errorSum = 0.0;
/*Added to old version, not necessary for PID control*/
/*pPID_wall->speakerBias = 1.25;*/


/* initialize K_u, T_u, and PID constants for zero stiffness */
/* Tyreus Luyben PID control for zero stiffness spring w/ slew rate limit
 */
pPID_zero->K_u = 7.348;
pPID_zero->T_u = 0.002;
pPID_zero->pGain = pPID_zero->K_u/2.2;
pPID_zero->iGain = pPID_zero->pGain/(2.2*pPID_zero->T_u);
pPID_zero->dGain = (pPID_zero->pGain*pPID_zero->T_u)/6.3;

/* compensate integral and derivative gains with sample time */
pPID_zero->iGain *= g_sample_time;
//dGain = dGain / sample_time;

/* initialize reference point and integral */
pPID_zero->setPoint = force_volts_avg;
pPID_zero->errorSum = 0.0;


/* initialize K_u, T_u, and PID constants for membrane puncture */
pPID_mem->K_u = pPID_zero->K_u;
pPID_mem->T_u = pPID_zero->T_u;
pPID_mem->pGain = pPID_zero->pGain;
pPID_mem->iGain = pPID_zero->iGain;
pPID_mem->dGain = pPID_zero->dGain;

pPID_mem->setPoint = rest_pos_mm;
pPID_mem->errorSum = 0.0;


/* Main loop ------------------------------------------------------------- */
while(1) {
    /* start timer0 to time while loop duration
     * timer0 currently set for 1000 us --> 1 kHz operating frequency
     */
    timer0_init();

    /* flip P2.0 HIGH to measure duration of computation - actuation loop
     * P2.0 flipped LOW in waitForRestOfPeriod()
     */
    GP2DAT ^= 0x10000;

    /* reset sensor status array */
    for (i=0; i<8; i++)
    {
        sensor_status[i] = 0;
    }

    /* Update mode of 1DoF depending on message received from Wixel */
```

```
    if (g_speaker_mode == 0xAA)
    {
        /* sine wave */
        speaker_volts = sine_wave(pSine_const);
        speaker_voltageHex = outputDAC(speaker_volts);
        DAC0DAT = speaker_voltageHex;
        store_DAC_data(speaker_voltageHex, pCurr_data);
    }
    else if (g_speaker_mode == 0xBB)
    {
        /* virtual wall behavior */
        speaker_volts = PID_virtualWall(pPID_wall);
        speaker_voltageHex = outputDAC(speaker_volts);
        DAC0DAT = speaker_voltageHex;
        store_DAC_data(speaker_voltageHex, pCurr_data);
    }
    else if (g_speaker_mode == 0xCC)
    {
        /* zero stiffness behavior */
        speaker_volts = PID_zeroStiffness(pPID_zero);
        speaker_voltageHex = outputDAC(speaker_volts);
        DAC0DAT = speaker_voltageHex;
        store_DAC_data(speaker_voltageHex, pCurr_data);
    }
    else if (g_speaker_mode == 0xDD)
    {
        /* membrane puncture */
        speaker_volts = membrane_puncture(pPID_wall, pPID_zero, pPID_mem);
        speaker_voltageHex = outputDAC(speaker_volts);
        DAC0DAT = speaker_voltageHex;
        store_DAC_data(speaker_voltageHex, pCurr_data);
    }
    else if (g_speaker_mode == 0xEE)
    {
        #if TEMP_DISABLED
        speaker_volts = ratchet_simulation(pPID_wall, pPID_zero,
                                           pPID_ratchet);
        speaker_voltageHex = outputDAC(speaker_volts);
        DAC0DAT = speaker_voltageHex;
        store_DAC_data(speaker_voltageHex, pCurr_data);
        #endif

        /* control speaker voltage w/ knob 1 */
        speaker_volts = read_store_sensors(1, pCurr_data);
        speaker_voltageHex = outputDAC(speaker_volts);
        DAC0DAT = speaker_voltageHex;
        store_DAC_data(speaker_voltageHex, pCurr_data);
    }
    else
    {
        /* mute if incorrect/undefined mode received */
        speaker_voltageHex = outputDAC(1.25);
        DAC0DAT = speaker_voltageHex;
        store_DAC_data(speaker_voltageHex, pCurr_data);
    }

    /* check to see if any sensors have not yet been read */
    check_sensor_status();

    /* dataUpdate_msg() is used to test SPI comms w/ 8 12-bit messages */
    //dataUpdate_msg(pTest_data);
    //send_curr_data(pTest_data);

    /* send data to Wixel over SPI. Data already formatted to 12-bit
     * [0 - FFF] range. Split eight 12-bit messages into twelve 8-bit
     * messages. Save received data into an array to be parsed later
     */
    GP2DAT ^= 0x20000; /*flip P2.1 to examine time for each send/ read*/
    send_curr_data(pCurr_data, pSPI_received);
    GP2DAT ^= 0x20000;

    #if OBSOLETE
    /* Read data from Wixel */
    while (!((SPISTA & 0x8) == 0x8)){
        /* wait while nothing in SPIRX */
    }
    message = SPI_read();

    /* only update mode if message falls within particular values */
    if (message == 0xAA || message == 0xBB || message == 0xCC ||
        message == 0xDD || message == 0xEE || message == 0x99)
    {
        g_speaker_mode = message;
    }
    #endif

    /* parse recieved SPI data for control keys and incoming GUI info
     * global variables are updated in this function
     */
    parse_SPI_data(pSPI_received);

    counter++;
    waitForRestOfPeriod();
}
```

```
    return 0;
} /* end of Main loop ------------------------------------------------------ */


/* Functions ------------------------------------------------------------- */

/**  PID_virtualWall() --
 **     PID control of position to resist speaker movement by an extern. force
 **/
float PID_virtualWall(struct PID_const *PID)
{
    #if TEMP_DISABLED
    float iMax = 1.25;
    float iMin = -1.25;
    #endif
    /* actuator limits */
    float outMax = 2.5;
    float outMin = 0.0;

                    float output_voltage, calc_output;
    float currOptical_volts, currOptical_mm;
    float pState, iState, dState;
    float pGain, iGain, dGain;

                    /* set point weights */
                    float beta = 1.0;
                    float gamma = 0.0;

    float error = 0.0;
    float slew_rate_limit = 0;
    float sample_time = 0.001; // sample (inter-loop) time in seconds
                    float T_f, K_t;

    //#if NOT_YET_IMPLEMENTED
    /* retreive PID constants, which already compensate for sample time
     * @TODO: remove NOT_YET_IMPLEMENTED blockers when prototyping done */
    pGain = PID->pGain;
    iGain = PID->iGain;
    dGain = PID->dGain;

    /* derivative filter constant and anti-kick gains */
    T_f = (pGain/dGain)/20.0;      /* T_f = (k_d/k_p)/N, N=[2,20] */
    K_t = iGain/pGain;             /* K_t = 1/T_i, where T_i = K_p/K_i */
    //#endif


    #if TEMP_DISABLED
    /* Obtain PID gains from knobs 1-3. Only update if knob has changed by
     * more than epsilon
     */
    //pGain = read_store_sensors(1, pCurr_data);
    //pGain = 1.0 + read_store_sensors(6, pCurr_data); -- V16
    //pGain = pot_voltage(pGain, 5.0);

    if (abs(pGain - PID->pGain) >= g_epsilon) PID->pGain = pGain;
    else pGain = PID->pGain;

    iGain = read_store_sensors(2, pCurr_data);
    if (abs(iGain - PID->iGain) >= g_epsilon) PID->iGain = iGain;
    else iGain = PID->iGain;

    dGain = read_store_sensors(3, pCurr_data);
    if (abs(dGain - PID->dGain) >= g_epsilon) PID->dGain = dGain;
    else dGain = PID->dGain;

    /* Ziegler Nichols tuning using virtual knob */
    pGain = ((float)g_virtual_knob1)/4096.0 + 1.0;

    /* set dGain & iGain to zero for Ziegler Nichols tuning */
    iGain = 0.0;
    PID->iGain = 0.0;
    dGain = 0.0;
    PID->dGain = 0.0;
    #endif

    /* Speaker Setpoint Manual Control tuning. Adjusts setpoint over based
     * on user control of position via knob 1. Setpoint changes due to while
     * function's placement in while loop (in main)
     */
    rest_pos_volts = read_store_sensors(1, pCurr_data);
    rest_pos_mm = pot_voltage(rest_pos_volts, 8.4);
    //rest_pos_mm = opticalADCtoDisplacement(rest_pos_volts);
    PID->setPoint = rest_pos_mm;

    /* read current position and calculate error */
    currOptical_volts = read_store_sensors(4, pCurr_data);
    currOptical_mm = optical_ADCtoDisplacement(currOptical_volts);

    /* error calculated in mm. setPoint is set at system initialization */
    /* error is curr - setPoint becuase the control is negative,
     *  i.e., to push up against external force, reduce speaker voltage
     */
    error = currOptical_mm - PID->setPoint;
    //PID->errorSum += error;
```

```
    /* Proportional term */
    //pState = (pGain * error);
                    pState = pGain * (currOptical_mm - beta*PID->setPoint);

    /* Integral term */
    //iState = iGain * (PID->errorSum);
                    iState = PID->errorSum;
    if (iState >= outMax) iState = outMax;
    else if (iState <= outMin) iState = outMin;

    /* Derivative term */
    //dState = dGain * (error - PID->lastError);
                    //dState = dGain * (currOptical_mm - PID->lastInput);

                    /* filtered derivative & derivative on measurement */
    dState = ((T_f/(T_f + g_sample_time)) * PID->lastErrorDerivative) -
             (dGain/(T_f + g_sample_time)) * (currOptical_mm - PID->lastInput);

    /* increasing voltage on speaker pushes down
     * optical position sensor increases as speaker pushes down
     */
    calc_output = pState + iState + dState;

                    /* check saturation */
    if (calc_output >= outMax) output_voltage = outMax;
    else if (calc_output <= outMin) output_voltage = outMin;
    else output_voltage = calc_output;

    /* limiter to block single loop impulses, checks diff between last
     * output and current output
     *      - noise acceptable @ 25mV limit, speaker can zero out in
     *        virtual wall
     */
    slew_rate_limit = 0.025;
    if (output_voltage - PID->lastOutput >= slew_rate_limit)
    {
        /* if new output > last by more than limit, only increment by limit */
        output_voltage = PID->lastOutput + slew_rate_limit;
    }
    else if (output_voltage - PID->lastOutput <= -1.0 * slew_rate_limit)
    {
        /* if new output < last by more than limit, decrement by limit */
        output_voltage = PID->lastOutput - slew_rate_limit;
    }

    /* update integral using saturation and integral of k_i*e to reduce
     * kick related to changing gains
     */
    PID->errorSum += (iGain * error) + K_t * (output_voltage - calc_output);

                    PID->lastInput = currOptical_mm;
    PID->lastOutput = output_voltage;
                    PID->lastErrorDerivative = dState;
    PID->lastError = error;
    return output_voltage;
}

/** PID_zeroStiffness() --
 **     PID control of force to keep force on sensor at its preload point
 **/
float PID_zeroStiffness(struct PID_const *PID)
{
    #if TEMP_DISABLED
    /* @TODO: delete after prototyping completed */
    float iMax = 1.25;
    float iMin = -1.25;
    #endif
    /* actuator limits */
    float outMax = 2.5;
    float outMin = 0.0;

    float calc_output, output_voltage;
    float pState, iState, dState;
    float pGain, iGain, dGain;
                    float T_f, K_t;

    float curr_pos_volts, curr_pos;
    float error = 0.0;

    /* set point weighting parameters */
    float beta = 1.0;       /* reference point weight */
    float gamma = 0.0;      /* delta reference point weight */

    float slew_rate_limit = 0.0;
    float sample_time = 0.001;  // sample (inter-loop) time in seconds

    //#if NOT_YET_IMPLEMENTED
    /* retreive PID constants, which already compensate for sample time
     * @TODO: remove NOT_YET_IMPLEMENTED blockers when prototyping done */
    pGain = PID->pGain;
    iGain = PID->iGain;
    dGain = PID->dGain;

    /* derivative filter constant and anti-kick gains */
    T_f = (pGain/dGain)/20.0;      /* T_f = (k_d/k_p)/N, N=[2,20] */
```

```
K_t = iGain/pGain;            /* K_t = 1/T_i, where T_i = K_p/K_i */
//#endif

#if TEMP_DISABLED
/* @TODO: DISABLE PID constant settings when prototyping completed. */

/* read gains from knobs and only save if different from last loop */
/* first knob -- proportional gain */
//pGain = 2.0 + read_store_sensors(6, pCurr_data);
//pGain = pot_voltage(pGain, 10.0);
pGain = 1.0 + 10.0*(((float)g_virtual_knob1)/4096.0);

if (abs(pGain - PID->pGain) > g_epsilon) PID->pGain = pGain;
else pGain = PID->pGain;

/* second knob -- integral gain */
iGain = read_store_sensors(2, pCurr_data);
if (abs(iGain - PID->iGain) > g_epsilon) PID->iGain = iGain;
else iGain = PID->iGain;

/* third knob -- differential gain */
dGain = read_store_sensors(3, pCurr_data);
if (abs(dGain - PID->dGain) > g_epsilon) PID->dGain = dGain;
else dGain = PID->dGain;

/* set dGain & iGain to zero during Ziegler Nichols tuning */
iGain = 0.0;
PID->iGain = 0.0;
dGain = 0.0;
PID->dGain = 0.0;
#endif

/* push --> voltage drop; pull --> voltage increase */
force_volts = read_store_sensors(0, pCurr_data);
curr_pos_volts = read_store_sensors(4, pCurr_data);
curr_pos = optical_ADCtoDisplacement(curr_pos_volts);

/* Voltage-based error implementation */
/* error calculated as setPoint - curr because control is positive,
 *   i.e., with push on sensor, increase speaker voltage to "run away"
 */
error = PID->setPoint - force_volts;

/* Integral Term */
iState = PID->errorSum;
/* limit integral value to reduce effect of windup */
if (iState >= outMax) iState = outMax;
else if (iState <= outMin) iState = outMin;

/* Differential Term */
                //dState = dGain * (force_volts - PID->lastInput);
//dState = dGain * (error - PID->lastError);
/* filtered derivative & derivative on measurement */
                //#if TEMP_DISABLED
dState =  ((T_f/(T_f + g_sample_time)) * PID->lastErrorDerivative) -
          (dGain/(T_f + g_sample_time)) * (force_volts - PID->lastInput);
                //#endif

/* Proportional Term */
//pState = pGain * error;
pState = pGain * (beta*PID->setPoint - force_volts);

/* PID Controller Equation */
//output_voltage = pState + iState + dState + 1.25;
calc_output = pState + iState + dState;

/* check saturation */
if (calc_output >= outMax) output_voltage = outMax;
else if (calc_output <= outMin) output_voltage = outMin;
else output_voltage = calc_output;

/* Slew rate limiter implemented to minimize speaker drift to ends of
 * displacement range. Prevents large changes in voltage
 */
slew_rate_limit = 0.25;
if (output_voltage - PID->lastOutput >= slew_rate_limit)
{
    /* if new output > last by more than limit, only increment by limit */
    output_voltage = PID->lastOutput + slew_rate_limit;
}
else if (output_voltage - PID->lastOutput <= -1.0 * slew_rate_limit)
{
    /* if new output < last by more than limit, decrement by limit */
    output_voltage = PID->lastOutput - slew_rate_limit;
}

/* update integral using saturation and integral of k_i*e to reduce
 * kick related to changing gains
 */
PID->errorSum += (iGain * error) + K_t * (output_voltage - calc_output);

/* save data for next loop */
PID->lastInput = force_volts;
PID->lastOutput = output_voltage;
PID->lastError = error;
```

```
    PID->lastErrorDerivative = dState;

    return output_voltage;
}

/** membrane_puncture() --
 **     Simulate membrane puncture by rendering virtual wall until a certain
 **     increase or decrease in force on the force sensor. Then render
 **     a zero stiffness membrane.
 **/
float membrane_puncture(struct PID_const *PID_w, struct PID_const *PID_z,
                        struct PID_const *PID_m)
{
    float curr_pos, curr_pos_volts;
    float curr_force_grams, curr_force_volts;
    float applied_force, membrane_force;
    float spring_stiff, spring_damping;

    float interp_m, interp_b;

    /* membrane is 1mm thick, centered around init point */
    float membrane_top = PID_m->setPoint + 0.5;
    float membrane_bot = PID_m->setPoint - 0.5;
    float membrane_center = PID_m->setPoint;

    float pState, iState, dState;
    float pGain, iGain, dGain;
    float T_f, K_t;
    float error;
    float calc_output, output_voltage;
    float slew_rate_limit;
    float outMax = 2.5;
    float outMin = 0.0;
                    float beta = 1.0;

    /* critical force/voltage puncture implementation */
    float iMax = 2.5;
    float iMin = -2.5;
    float critical_force;
    int in_membrane_flag = 0;

    /* read position and force sensors */
    curr_pos_volts = read_store_sensors(4, pCurr_data);
    curr_pos = optical_ADCtoDisplacement(curr_pos_volts);

    curr_force_volts = read_store_sensors(0, pCurr_data);
    curr_force_grams = force_VoltsToGrams(curr_force_volts);
    applied_force = curr_force_grams - force_preload;

    /* spring stiffness in grams/mm,
     * @TODO: in [N/m] = [101.97162 grams/1000 mm]
     * spring damping in [N*s/m]
     */
    spring_stiff = 50.0;
    spring_damping = 5.0;

    #if TEMP_DISABLED
    /* interpolate slope and intercept from current pos */
    interp_cal_constants(curr_pos, pForce_to_DAC, pInterpolated);
    interp_m = pInterpolated->slope;
    interp_b = pInterpolated->intercept;

    /* calculate haptic forces depending on position w/r/t membrane */
    if (curr_pos > membrane_top || curr_pos < membrane_bot)
    {
        /* above and below membrane, render zero stiffness */
        pGain = PID_z->pGain;
        iGain = PID_z->iGain;
        dGain = PID_z->dGain;

        /* PID control based on force sensor voltage */
        error = PID_z->setPoint - curr_force_volts;

                    if (g_in_membrane_flag == 1)
                    {
                              /* was in membrane, no longer in membrane */
                              PID_m->errorSum = 0.0;

                              g_membrane_punctured = 0;
                              g_in_membrane_flag = 0;
                              g_membrane_entry = 0;
                    }
    }
    else if (curr_pos <= membrane_top && curr_pos >= membrane_bot)
    {
        /* on first entry into membrane, flip inside_flag and set entry
         * indicator
         */
        if (g_in_membrane_flag == 0 && curr_pos > membrane_center)
        {
            g_in_membrane_flag = 1;

            /* membrane entry: 0 = not entered, 1 = entered from top,
             * 2 = entered from bot
             */
```

```
        g_membrane_entry = 1;
    }
    else if (g_in_membrane_flag == 0 && curr_pos < membrane_center)
    {
        g_in_membrane_flag = 1;
        g_membrane_entry = 2;
    }

    /* calculate membrane force with respect to where user first entered */
    /* make note to calculate velocity in [m/s] */
    if (g_membrane_entry == 1)
    {
        /* entered from top */
        membrane_force = spring_stiff * (membrane_top - curr_pos);
        #if TEMP_DISABLED
        membrane_force = spring_stiff * (membrane_top - curr_pos) +
                         spring_damping * (PID_m->lastPos - curr_pos);
        #endif
    }
    else if (g_membrane_entry == 2)
    {
        /* entered from bottom */
        membrane_force = spring_stiff * (membrane_bot - curr_pos);
        #if TEMP_DISABLED
        membrane_force = spring_stiff * (membrane_bot - curr_pos) +
                         spring_damping * (PID_m->lastPos - curr_pos);
        #endif
    }

}

if (g_in_membrane_flag >= 1)
{
    /* calculate force if within membrane */
    output_voltage = interp_m * membrane_force + interp_b;
}
else if (g_in_membrane_flag == 0)
{
    /* PID zero stiffness control */
    pState = pGain * (beta*PID_z->setPoint - curr_force_volts);
    iState = PID_m->errorSum;
    if (iState >= outMax) iState = outMax;
    else if (iState <= outMin) iState = outMin;
    dState = ((T_f/(T_f + g_sample_time)) * PID_m->lastErrorDerivative) -
             (dGain/(T_f + g_sample_time)) * (curr_force_volts - PID_m->lastInput);

    /* calculate PID output */
    calc_output = pState + iState + dState;

    /* check for saturation */
    if (calc_output >= outMax) output_voltage = outMax;
    else if (calc_output <= outMin) output_voltage = outMin;
    else output_voltage = calc_output;

    /* slew rate limiter implemented to minimize speaker drift to ends of
     * displacement range. Prevents large changes in voltage
     */
    slew_rate_limit = 0.25;
    if (output_voltage - PID_m->lastOutput >= slew_rate_limit)
    {
        /* if new output > last by more than limit, only increment by limit */
        output_voltage = PID_m->lastOutput >= slew_rate_limit;
    }
    else if (output_voltage - PID_m->lastOutput <= -1.0 * slew_rate_limit)
    {
        /* if new output < last by more than limit, only reduce by limit */
        output_voltage = PID_m->lastOutput - slew_rate_limit;
    }

    T_f = (pGain/dGain)/20.0;
    K_t = iGain/pGain;

    PID_m->errorSum += (iGain * error) + K_t * (output_voltage - calc_output);

    /* save calculated values */
    PID_m->lastError = error;
    PID_m->lastInput = curr_force_volts;
    PID_m->lastErrorDerivative = dState;
}

PID_m->lastPos = curr_pos;
PID_m->lastOutput = output_voltage;
#endif

//#if OBSOLETE
/* membrane puncture by critical force/voltage change */
applied_force = PID_z->setPoint - curr_force_volts;

/* make critical force 1.50 V from preload */
critical_force = 1.00;
if (curr_pos >= membrane_top || curr_pos <= membrane_bot)
{
    /* above & below membrane, render zero stiffness */
    pGain = PID_z->pGain;
    iGain = PID_z->iGain;
```

```
    dGain = PID_z->dGain;

    /* outside membrane, user applied force is "error" */
    error = applied_force;

    /* "reset" error sum for membrane before/after puncture */
    PID_m->errorSum = 0.0;

    /* reset puncture flag */
    g_membrane_punctured = 0;
    in_membrane_flag = 0;
}
else if (curr_pos < membrane_top && curr_pos > membrane_bot)
{
    if (abs(applied_force) <= critical_force && g_membrane_punctured == 0)
    {
        in_membrane_flag = 1;

        /* use knob to adjust stiffness of spring. Stop at 0.90
         * because @ 1.0*K_u, speaker oscillates
         */
        //spring_stiff = pot_voltage(2, 0.95);

        /* render virtual spring using only proportional control */
        //pGain = PID_w->K_u * spring_stiff;
        pGain = PID_w->K_u * 0.50;  /* Z-N tuning for P control */
        iGain = 0.0;
        dGain = 0.0;

        /* inside membrane, user position w/r/t membrane is "error" */
        /* error is negative because virtual wall control is negative
         *   i.e. increasing voltage on speaker pushes down
         */
        if (curr_pos < membrane_top && curr_pos > membrane_center)
        {
            /* error < 0 so speaker pushes up as user pushes down */
            error = curr_pos - membrane_top;
        }
        else if (curr_pos > membrane_bot && curr_pos < membrane_center)
        {
            /* error > 0 so speaker pushes down as user pulls up */
            error = curr_pos - membrane_bot;
        }

        PID_m->errorSum += error;
    }
    else
    {
        /* if applied force greater than critical value, render zero
         * stiffness membrane and flip punture flag
         */
        pGain = PID_z->pGain;
        iGain = PID_z->iGain;
        dGain = PID_z->dGain;

        error = applied_force;
        PID_m->errorSum = 0.0;

        g_membrane_punctured = 1;
        in_membrane_flag = 0;
    }
}

/* save gains into membrane puncture struct */
PID_m->pGain = pGain;
PID_m->iGain = iGain;
PID_m->dGain = dGain;

/* Proportional Term */
pState = pGain * error;

/* Integral Term */
iState = iGain * PID_m->errorSum;
if (iState > iMax) iState = iMax;
else if (iState < iMin) iState = iMin;

/* Differential Term */
dState = dGain * (error - PID_m->lastError);

/* PID output equation */
//output_voltage = pState + iState + dState + 1.25;

/* check for saturation */
//if (calc_output >= outMax) output_voltage = outMax;
//else if (calc_output <= outMin) output_voltage = outMin;
//else output_voltage = calc_output;

/* PID output equation with position dependent speaker bias */
//#if TEMP_DISABLED
if (in_membrane_flag)
{
    output_voltage = pState + iState + dState + 1.25;
}
else
{
```

```
        PID_m->speakerBias = displacement_SpeakerToDAC(curr_pos);
        output_voltage = pState + iState + dState + PID_m->speakerBias;
    }
    //#endif

    /* limiter to block single loop impulses, checks diff between last
     * output and current output
     *      - noise acceptable @ 25mV limit, viscosity noticeable in
     *        zero stiffness regions
     *      - noise acceptable @ 50mV limit, viscosity reduced compared
     *        to limit @ 25mV
     */
    slew_rate_limit = 0.050;
    if (output_voltage - PID_m->lastOutput >= slew_rate_limit)
    {
        /* if new output > last by 50mV only increment by 50mV */
        output_voltage = PID_m->lastOutput + slew_rate_limit;
    }
    else if (output_voltage - PID_m->lastOutput <= -1.0 * slew_rate_limit)
    {
        /* if new output < last by more than 50mV, decrement by 50mV*/
        output_voltage = PID_m->lastOutput - slew_rate_limit;
    }

    PID_m->lastOutput = output_voltage;
    PID_m->lastError = error;
    //#endif

    return output_voltage;
}

/** ratchet_simulation() --
 **     Simulate the tactile feeling of inserting/ removing linear ratchet.
 **     Determine if current position is within a membrane, if so, set a
 **     global flag and membrane gains. Else, set zero stiffness gains.
 **/
float ratchet_simulation(struct PID_const *PID_w, struct PID_const *PID_z,
                         struct PID_const *PID_r)
{
    float iMax = 2.5;
    float iMin = -2.5;
    float pGain, iGain, dGain;
    float pState, iState, dState;
    float output_voltage;
    float curr_pos, curr_pos_volts;
    float critical_force, applied_force, curr_force_volts;
    float slew_rate_limit;
    float error;

    /* membrane parameters in mm */
    float membrane_thickness = 0.50;
    float membrane_center[4] = {1.50, 3.50, 5.50, 7.50};
    float curr_center = 0.0;
    float curr_mem_top = 0.0;
    float curr_mem_bot = 0.0;
    int in_membrane = 0;

    curr_pos_volts = read_store_sensors(4, pCurr_data);
    curr_pos = optical_ADCtoDisplacement(curr_pos_volts);

    curr_force_volts = read_store_sensors(0, pCurr_data);
    applied_force = PID_z->setPoint - curr_force_volts;

    //critical_force = pot_voltage(1, 0.50);
    critical_force = 0.50;

    /* determine if current position is within a membrane */
    for (i=0;i<4;i++)
    {
        if (curr_pos <= (membrane_center[i] + membrane_thickness/2.0) &&
            curr_pos >= (membrane_center[i] - membrane_thickness/2.0))
        {
            in_membrane = 1;

            curr_center = membrane_center[i];
            curr_mem_top = membrane_center[i] + membrane_thickness/2.0;
            curr_mem_bot = membrane_center[i] - membrane_thickness/2.0;
        }
    }

    /* set PID gains based on whether we are currently in a membrane */
    if (in_membrane)
    {
        if (abs(applied_force) < critical_force && g_ratchet_punctured == 0)
        {
            /* while in membrane, and force is less than critical, error
             * given by PID
             */
            pGain = PID_w->K_u *0.50;
            iGain = 0.0;
            dGain = 0.0;

            if (curr_pos > curr_center && curr_pos <= curr_mem_top)
            {
                error = curr_pos - curr_mem_top;
```

```
            }
            else if (curr_pos < curr_center && curr_pos >= curr_mem_bot)
            {
                error = curr_pos - curr_mem_bot;
            }
            PID_r->errorSum += error;
        }
        else
        {
            /* force has exceeded critical, render zero stiffness spring */
            g_ratchet_punctured = 1;

            pGain = PID_z->pGain;
            iGain = PID_z->iGain;
            dGain = PID_z->dGain;

            error = applied_force;
            PID_r->errorSum = 0.0;
        }

    }
    else
    {
        /* not inside any membrane, so set zero stiffness gains & reset
         * pucnture flag
         */
        pGain = PID_z->pGain;
        iGain = PID_z->iGain;
        dGain = PID_z->dGain;

        error = applied_force;
        PID_r->errorSum = 0.0;

        g_ratchet_punctured = 0;
    }

    PID_r->pGain = pGain;
    PID_r->iGain = iGain;
    PID_r->dGain = dGain;

    /* Proportional Term */
    pState = pGain * error;

    /* Integral Term */
    iState = iGain * PID_r->errorSum;
    if (iState > iMax) iState = iMax;
    else if (iState < iMin) iState = iMin;

    /* Differential Term */
    dState = dGain * (error - PID_r->lastError);

    /* PID output equation */
    //output_voltage = pState + iState + dState + 1.25;

    /* PID output equation with position dependent speaker bias */
    if (in_membrane)
    {
        output_voltage = pState + iState + dState + 1.25;
    }
    else
    {
        PID_r->speakerBias = displacement_SpeakerToDAC(curr_pos);
        output_voltage = pState + iState + dState + PID_r->speakerBias;
    }

    /* limiter to block single loop impulses, checks diff between last
     * output and current output
     *      - noise acceptable @ 25mV limit, viscosity noticeable in
     *        zero stiffness regions
     *      - noise acceptable @ 50mV limit, viscosity reduced compared
     *        to limit @ 25mV
     */
    slew_rate_limit = 0.050;
    if (output_voltage - PID_r->lastOutput >= slew_rate_limit)
    {
        /* if new output > last by 50mV only increment by 50mV */
        output_voltage = PID_r->lastOutput + slew_rate_limit;
    }
    else if (output_voltage - PID_r->lastOutput <= -1.0 * slew_rate_limit)
    {
        /* if new output < last by more than 50mV, decrement by 50mV*/
        output_voltage = PID_r->lastOutput - slew_rate_limit;
    }

    PID_r->lastOutput = output_voltage;
    PID_r->lastError = error;
    return output_voltage;
}

/** sine_wave() --
 **     Generate sine wave at specified amplitude and frequency, determined
 **     by knobs 1 and 2
 **/
float sine_wave(struct Wave_const *constants)
{
```

```
    float amplitude = 0.0;
    float frequency = 0.0;
    float output = 0.0;
    float radians = 0.0;
    float voltage = 0.0;

    int curr_theta = 0;
    int delta_theta = 0;
    int fullscale = pow(2.0, 16);   /* fullscale number of bits */

    /* read knob 1 to get amplitude, centered around 1.25 */
    voltage = read_store_sensors(1, pCurr_data);
    amplitude = pot_voltage(voltage, 1.0);

    /* read knob 2 to get frequency & calculate the desired change in theta */
    voltage = read_store_sensors(2, pCurr_data);
    frequency = pot_voltage(voltage, 50.0);

    /* update current theta and convert to radians */
    /* delta_theta = fullscale / operating freq (hz), the change in theta to
     * obtain a 1 Hz sine wave. For 16 bits @ 1333 Hz update, delta = 49
     */
    //delta_theta = 49 * frequency; /* del_theta for 1333 Hz, loop = 750us */
    delta_theta = 65 * frequency;   /* del_theta for 1000 Hz, loop = 1000us */
    constants->theta_counter += delta_theta;
    curr_theta = constants->theta_counter % fullscale;
    radians = curr_theta * (2.0 * PI) / fullscale;

    /* calculate output voltage, centered around midpoint */
    output = amplitude*sin(radians) + 1.25;

    return output;
}

/** speakerDACtoDisplacement() --
 **     Calculate expected speaker displacement from a reference voltage
 **/
float speaker_DACtoDisplacement(float speaker_volts)
{
    float speaker_mm = 0.0;

    /* calibration from 28 March 2016, displacement = 0 when DAC0 = 2.5 */
    speaker_mm = 2.0994*pow(speaker_volts, 3.0) -
                 7.5992*pow(speaker_volts, 2.0) + 2.8832*speaker_volts +
                 8.0796;

    return speaker_mm;
}

/** opticalADCtoDisplacement() --
 **     Calculate speaker displacement from output voltage
 **/
float optical_ADCtoDisplacement(float optical_volts)
{
    float optical_mm = 0.0;

    /* calibration from 28 Mar 2016, displacement = 0 when DAC0 = 2.5 */
    optical_mm = 2.1677*pow(optical_volts, 2.0) - 1.2489*optical_volts + 0.32;

    return optical_mm;
}

/** displacementToOpticalADC() --
 **     Calculate expected optical ADC voltage from speaker displacement
 **/
float displacement_OpticalToADC(float displacement_mm)
{
    float optical_ADC = 0.0;

    /* calibration from 28 Mar 2016, displacement = 0 when DAC0 = 2.5 */
    optical_ADC = -0.0194*pow(displacement_mm, 2.0) + 0.3633*displacement_mm +
                   0.4993;

    return optical_ADC;
}

/** displacementToSpeakerDAC() --
 **     Calculate the expected speaker bias for a given desired displacement
 **/
float displacement_SpeakerToDAC(float displacement_mm)
{
    float speaker_DAC;

    /* calibration from 28 Mar 2016, displacement = 0 when DAC0 = 2.5 */
    speaker_DAC = -0.0161*pow(displacement_mm, 3.0) +
                   0.2130*pow(displacement_mm, 2.0) - 0.9592*displacement_mm +
                   2.6129;

    return speaker_DAC;
}

/** force_VoltsToGrams() --
 **     Convert force sensor voltage to grams
 **/
float force_VoltsToGrams(float force_volts)
```

```
{
    return (50.0*force_volts)/0.04;
}

/** interpolate_cal_constants() --
 **     Interpolate calibration constants from current position
 **/
void interp_cal_constants(float curr_pos, struct calibration *cal_eqns,
                          struct calibration *result)
{
    float part_int, part_frac;
    float interp_m, interp_b;
    float w;
    int inside_positions = 0;

                  float lower_pos;
    int lower_index, upper_index;

    part_int = modff(curr_pos, &part_frac);

    if (part_frac >= 0.500)
    {
        lower_pos = part_int + 0.500;

                                       lower_index = (int)(lower_pos * 2.0);
        upper_index = lower_index + 1;
    }
    else
    {
        lower_pos = part_int;

                                       lower_index = (int)(lower_pos * 2.0);
        upper_index = lower_index + 1;
    }

    /* check to make sure curr_pos is within the two indices just identified */
    while (inside_positions == 0)
    {
        if (curr_pos >= cal_eqns[lower_index].x && curr_pos <= cal_eqns[upper_index].x)
        {
            /* correct */
            inside_positions = 1;
        }
        else
        {
            if (curr_pos - cal_eqns[lower_index].x < 0)
            {
                /* curr_pos somehow lower than lower position */
                lower_index -= 1;
                upper_index = lower_index + 1;
            }
            else if (curr_pos - cal_eqns[upper_index].x > 0)
            {
                /* curr_pos somehow higher than upper position */
                lower_index += 1;
                upper_index = lower_index + 1;
            }
        }
    }

    /* calculate weight calibration factor w/r/t left index, on scale [0,1] */
    w = (curr_pos - cal_eqns[lower_index].x) / 0.50;

    /* interp = w * val[i] + (1-w) * val[i+1] */
    interp_m = w * cal_eqns[lower_index].slope +
               (1.0 - w) * cal_eqns[upper_index].slope;

    interp_b = w * cal_eqns[lower_index].intercept +
               (1.0 - w) * cal_eqns[upper_index].intercept;

    result->slope = interp_m;
    result->intercept = interp_b;

    return;
}

/** dataUpdate_msg() --
 **     Update and format test messages sent to Wixel
 **/
void dataUpdate_msg(struct Wixel_msg *data)
{
    /* Create test messages which follow similar format to
     * dataUpdate_testmsg().
     * Start with [800, 901, A02, B03, C04, D05, E06, F07] -->
     * [8FF, 900, A01,... etc]
     * the global counter g_loop_count is used to keep track btw loops
     */

    data->forceSensor = 0x800 + g_loop_count;
    data->opticalSensor = 0x900 + (g_loop_count + 1);
    data->inductanceSensor = 0xA00 + (g_loop_count + 2);
    data->speakerOut = 0xB00 + (g_loop_count + 3);
    data->knob1 = 0xC00 + (g_loop_count + 4);
    data->knob2 = 0xD00 + (g_loop_count + 5);
    data->knob3 = 0xE00 + (g_loop_count + 6);
```

```
    data->tenTurnPot = 0xF00 + (g_loop_count + 7);

    /* g_loop_count is char, so it returns to 00 after FF */
    g_loop_count++;

    return;
}

/** dataUpdate_testmsg() --
 **     Loop HEX integer from A00 to CFF amongst members of Wixel_testmsg
 **/
void dataUpdate_testmsg(struct Wixel_test *test_msg)
{
    /* Creates 3 test msgs which cycle, to test Wixel data logging interface
     * Messages are patterned like: A00, B01, C02 --> AFF, B00, C01
     * A global counter counterTest_message is used to keep track btw loops
     */

    if (counterTest_message == 0xB00)
    {
        counterTest_message = 0xA00;
    }

    test_msg->Comp_1 = counterTest_message;

    if (counterTest_message == 0xAFE)
    {
         test_msg->Comp_2 = counterTest_message + 0x101;
         test_msg->Comp_3 = 0xC00;
    }
    else if (counterTest_message == 0xAFF)
    {
         test_msg->Comp_2 = 0xB00;
         test_msg->Comp_3 = 0xC01;
    }
    else
    {
         test_msg->Comp_2 = counterTest_message + 0x101;
         test_msg->Comp_3 = counterTest_message + 0x202;
    }

    counterTest_message += 1;

    return;
}

/** dataUpdate_testsend() --
 **     Modify message members to fit 12 bit format
 **/
void dataUpdate_testformat(struct Wixel_test *curr_msg)
{
    /*  Modify messages to fit 12 bit format */
    curr_msg->Comp_1 = curr_msg->Comp_1 >> 16;
    curr_msg->Comp_2 = curr_msg->Comp_2 >> 16;
    curr_msg->Comp_3 = curr_msg->Comp_3 >> 16;

    return;
}

/** send_test_data() --
 **     Splits 3 messages of 12 bits into 5 8-bit messages and send
 **/
void send_test_data(struct Wixel_test *msg)
{
    int msg1, msg2, msg3;
    char split_msg[5];

    msg1 = msg->Comp_1;
    msg2 = msg->Comp_2;
    msg3 = msg->Comp_3;

    /* split three 12 bit messages into 5 using bit masking & shifting
     * msg #5 is shifted left 4 bits so last nibble is 000
     */
    split_msg[0] = (msg1 & 0xFF0) >> 4;
    split_msg[1] = (msg1 & 0xF) << 4 | (msg2 & 0xF00) >> 8;
    split_msg[2] = (msg2 & 0xFF);
    split_msg[3] = (msg3 & 0xFF0) >> 4;
    split_msg[4] = (msg3 & 0xF) << 4;

    /* send messages with delay between each to accomodate slower Wixel
     * operating freq
     */
    SPI_write(0xAA);
    delay_us(10);
    SPI_write(split_msg[0]);
    delay_us(10);
    SPI_write(split_msg[1]);
    delay_us(10);
    SPI_write(split_msg[2]);
    delay_us(10);
    SPI_write(split_msg[3]);
    delay_us(10);
    SPI_write(split_msg[4]);
    //delay_us(5);
```

```
    //SPI_write(0xFF);

    return ;
}

/** send_curr_data() --
 **     split ADC/DAC data into individual byte messages and send data via SPI
 **/
void send_curr_data(struct Wixel_msg *data, unsigned char *received_data)
{
    int msg0, msg1, msg2, msg3, msg4, msg5, msg6, msg7 = 0;
    char split_msg[12];

    msg0 = data->forceSensor;
    msg1 = data->opticalSensor;
    msg2 = data->inductanceSensor;
    msg3 = data->speakerOut;
    msg4 = data->knob1;
    msg5 = data->knob2;
    msg6 = data->knob3;
    msg7 = data->tenTurnPot;

    /* split each 1.5 byte piece of data into one byte segments */
    split_msg[0] = (char)((msg0 & 0xFF0) >> 4);
    split_msg[1] = (char)((msg0 & 0xF) << 4 | (msg1 & 0xF00) >> 8);
    split_msg[2] = (char)(msg1 & 0xFF);
    split_msg[3] = (char)((msg2 & 0xFF0) >> 4);
    split_msg[4] = (char)((msg2 & 0xF) << 4 | (msg3 & 0xF00) >> 8);
    split_msg[5] = (char)(msg3 & 0xFF);
    split_msg[6] = (char)((msg4 & 0xFF0) >> 4);
    split_msg[7] = (char)((msg4 & 0xF) << 4 | (msg5 & 0xF00) >> 8);
    split_msg[8] = (char)(msg5 & 0xFF);
    split_msg[9] = (char)((msg6 & 0xFF0) >> 4);
    split_msg[10] = (char)((msg6 & 0xF) << 4 | (msg7 & 0xF00) >> 8);
    split_msg[11] = (char)(msg7 & 0xFF);

    /* send messages w/ 10us delay so Wixel buffer does not overflow
                     * start/stop handshakes 0x11 and 0x22 so they do not match any
                     * command bytes
                     */
    /*SPI_write(0xAA);*/
                    SPI_write(0x11);
    /* Read data from Wixel */
    *received_data = SPI_read();
    for (i=0;i<12;i++)
    {
        delay_us(10);
        SPI_write(split_msg[i]);

        /* Read data from Wixel */
        *(received_data + i + 1) = SPI_read();
    }
    delay_us(10);
    /*SPI_write(0xBB);*/
                    SPI_write(0x22);
    /* read data from Wixel */
    *(received_data + 13) = SPI_read();

    return;
}

/** parse_SPI_data() --
 **     Read stored SPI data from Wixel. Return the control key if contained
 **     in the received data
 **/
void parse_SPI_data(unsigned char *received_data)
{
    int message = 0;
    unsigned int rec1, rec2, rec3 = 0;
    int loop_count, i = 0;
    unsigned int decoded_msg[8] = {0,0,0,0,0,0,0,0};

    /* received data is command = 1 byte, followed by 7 1.5 byte data, with
     * 4bits left over. 8th decoded point should be 0x0BB
     */
  /* received data is organized as:
   * rec_data[0] = ignored (0x11)
   * rec_data[1] = command mode
   * rec_data[2:12] = data from GUI
   * rec_data[13] = 0x22
   */
    for (i=0; i<4; i++)
    {
        /* extract three byte sections from rec_data to decode into two 1.5
         * byte data points, i.e. [A0],[1B],[02] --> [A01],[B02]
         */
        rec1 = *(received_data + i*3 + 2);
        rec2 = *(received_data + i*3 + 3);
        rec3 = *(received_data + i*3 + 4);

        /* mask extracted bytes to obtain 1.5 byte data */
        decoded_msg[i*2] = (rec1 << 4)  | ((rec2 & 0xF0) >> 4);
        decoded_msg[i*2+ 1] = ((rec2 & 0xF)<< 8) | (rec3);
    }
```

```
  /* speaker mode is first entry after 0x11 handshake */
    //g_speaker_mode = *(received_data + 1);
                    //g_speaker_mode = received_data[1];

                    /* loop through all received messages for command key */
                    for (loop_count = 0; loop_count<14; loop_count++)
                    {
                              message = received_data[loop_count];
                              if (message == 0xAA || message == 0xBB || message == 0xCC ||
          message == 0xDD || message == 0xEE || message == 0x99)
      {
                                        /* at most one control key in each batch of received data
                                 * so we can exit the loop immediately after identifying it
                                 */
         g_speaker_mode = message;
                                          break;
      }
                    }

    /* loop through all messages and assign globals depending on function
   * decoded_msg starts with virtual knob 1
   */
    for (loop_count=0; loop_count<8; loop_count++)
    {
        message = decoded_msg[loop_count];

        if (loop_count == 0)
        {
            g_virtual_knob1 = message;
        }
        else if (loop_count == 1)
        {
            g_virtual_knob2 = message;
        }

        else if (loop_count == 2)
        {
            g_virtual_knob3 = message;
        }
    }
}

/** SPI_read() --
 **     Read from SPI receive register
 **/
unsigned char SPI_read(void)
{
    unsigned char message = 0;
    while (!((SPISTA & 0x8) == 0x8)){
        /* wait while nothing in SPIRX */
    }
    message = SPIRX;
    return message;
}

/** SPI_write() --
 **     Send a single 8 bit message over SPI on GP1
 **/
void SPI_write(unsigned char data)
{
    SPITX = data;
    while((SPISTA & 0x1) == 0x1) {
        /* wait while SPI is transmitting */
    }
    return;
}

/** read_sensors() --
 **     Initialize ADC transfer from specified channel and return decimal
 **     voltage
 **/
float read_sensors(int channel)
{
    float sensorVoltage;

    ADCCP = channel;
    ADCCON = 0x6A3;
    ADCCON &= ~(1 << 7);   /* clear bit 7 to stop additional conversions */
    while (!ADCSTA){
        /* wait until conversion completes */
    }
    sensorVoltage = ADCinput(ADCDAT);
    return sensorVoltage;
}

/** read_store_sensors() --
 **     Initialize ADC transfer from specified channel and store results into
 **     curr_data.
 **/
 float read_store_sensors(int channel, struct Wixel_msg *data)
 {
    /* -----------
    Pin and Port Assignments:

    ADC0 -- Honeywell FS01 Force Sensor
```

```
    ADC1 -- Front panel knob 1
    ADC2 -- Front panel knob 2
    ADC3 -- Front panel knob 3
    ADC4 -- IR proximity (optical) sensor
    ADC5 -- Speaker inductance circuit output
    ADC6 -- 10 turn pot

    DAC0 -- Speaker output
    */

    float sensorVoltage = 0.0;
    int voltageHex = 0;
    int voltage_int = 0;

    ADCCP = channel;
    ADCCON = 0x6A3;
    ADCCON &= ~(1 << 7);   /* clear bit 7 to stop additional conversions */
    while (!ADCSTA){
        /* wait until conversion completes */
    }
    voltageHex = ADCDAT;
    sensorVoltage = ADCinput(voltageHex);

    /* right shift ADC data 16 bits to obtain result on [0, FFF] range
                    * mask with FFF to ensure data is 12-bits
                    */
    voltage_int = (voltageHex >> 16) & 0xFFF;

    /* store raw ADC results into curr_data and update sensor status array */
    if (channel == 0)
    {
        data->forceSensor = voltage_int;
        sensor_status[0] = 1;
    }
    if (channel == 1)
    {
        data->knob1 = voltage_int;
        //data->knob1 = g_virtual_knob1;
        sensor_status[4] = 1;
    }
    if (channel == 2)
    {
        data->knob2 = voltage_int;
        //data->knob2 = g_virtual_knob2;
        sensor_status[5] = 1;
    }
    if (channel == 3)
    {
        data->knob3 = voltage_int;
        //data->knob3 = g_virtual_knob3;
        sensor_status[6] = 1;
    }
    if (channel == 4)
    {
        data->opticalSensor = voltage_int;
        sensor_status[1] = 1;
    }
    if (channel == 5)
    {
        data->inductanceSensor = voltage_int;
        sensor_status[2] = 1;
    }
    if (channel == 6)
    {
        data->tenTurnPot = voltage_int;
        sensor_status[7] = 1;
    }

    return sensorVoltage;
 }

/** initial_read_store_sensors() --
 **     read/store sensor data so struct members contain some data, even if
 **     sensor is not read in the loop
 **/
void initial_read_store_sensors(struct Wixel_msg *data)
{
    int i = 0;

          for (i = 0; i < 7; i++)
    {
      read_store_sensors(i, data);
    }

    #if TEMP_DISABLED
    read_store_sensors(0, data);
    read_store_sensors(1, data);
    read_store_sensors(2, data);
    read_store_sensors(3, data);
    read_store_sensors(4, data);
    read_store_sensors(5, data);
    read_store_sensors(6, data);
    #endif

    return;
```

```
}

/** store_DAC_data() --
 **     store DAC data and flip sensor_status[] entry for speakerOut
 **/
void store_DAC_data(int rawDAC_hex, struct Wixel_msg *data)
 {
    int voltage_int = 0;

    voltage_int = (rawDAC_hex >> 16) & 0xFFF;
    /* store speakerOut voltage and set sensor status */
    data->speakerOut = voltage_int;
    sensor_status[3] = 1;

    return;
}

/** check_sensor_status() --
 **     check if sensor has been read this loop. If not, read/store result
 **/
 void check_sensor_status(void)
 {
    /* sensor_status array mirrors the order of Wixel_msg struct
     * [0 force_sensor, 1 optical_sensor, 2 inductance_sensor, 3 DAC_output,
     *  4 knob1, 5 knob2, 6 knob3, 7 10-turn knob]
     */
    for (i=0; i<8; i++)
    {
        if (!sensor_status[i])
        {
            /* read sensor if it hasn't yet been read */
            if (i == 0) read_store_sensors(0, pCurr_data);
            else if (i == 1) read_store_sensors(4, pCurr_data);
            else if (i == 2) read_store_sensors(5, pCurr_data);
            else if (i == 3) {/*do nothing for speakerOut voltage */}
            else if (i == 4) read_store_sensors(1, pCurr_data);
            else if (i == 5) read_store_sensors(2, pCurr_data);
            else if (i == 6) read_store_sensors(3, pCurr_data);
            else if (i == 7) read_store_sensors(6, pCurr_data);
        }
        delay_us(25);
    }

    return;
}

/** system_init() --
 **     Initialize ADuC7026 CPU, DAC, GPIOs
 **/
void system_init(void)
{
    POWKEY1 = 0x01;
    POWCON = 0x00;        /* Configures CPU clock for 41.78 MHz, CD=0 */
    POWKEY2 = 0xF4;

    PLLKEY1 = 0xAA;
    PLLCON = 0x01;
    PLLKEY2 = 0x55;

    /* Connect internal 2.5V reference on Vref pin */
    REFCON = 0x01;

    /* Set DAC0 output to 0-V_ref(2.5V) range and turn DAC0 on */
    DAC0CON = 0x12;

    /* Set DAC0 output to 0-V_ref(2.5V) range and turn DAC0 on */
    DAC1CON = 0x12;

    /* Set DAC3 output to 0-V_ref(2.5V) range and turn DAC2 on */
    //DAC2CON = 0x12;

    /* Set DAC3 output to 0-V_ref(2.5V) range and turn DAC3 on */
    //DAC3CON = 0x12;

    /* Set digital I/O port 0 as GPIO on pins P0.0-0.7
     * Set all I/O pins on port 0 as *OUTPUT*, init LOW
     */
    GP0CON = 0x00;
    GP0DAT |= 0xFF000000;

    /* GPIO Port 1 is used for SPI, set in SPI_init() */

    /* Set I/O port 2 as GPIO on pins P2.0-2.7
     * Set all pins on port 2 as *OUTPUT*, init LOW
     */
    GP2CON = 0x00;
    GP2DAT = 0xFF000000;

    /* Set digital I/O port 3 as GPIO on pins P3.0-3.7
     * Set all I/O pins on port 3 as *INPUT*, initialize LOW
    GP3CON = 0x00;
    GP3DAT = 0x00;
    */
}
```

```
/** ADCpoweron() --
 **     Power on ADC, software-timed wait
 **/
void ADCpoweron(int time)
{
    ADCCON = 0x20;        /* Power-on the ADC */
    delay_sw(time);       /* Wait for ADC to be fully powered on */
}

/** SPI_init() --
 **     Initialize ADuC7026 SPI system on GPIO Port 1
 **/
void SPI_init(void)
{
    /* Set GPIO P1.x as SPI */
    /* P1.4 - 1.7 = SCLK, MISO, MOSI, CS, respectively */
    GP1CON = 0x22220000;
    SPIDIV = 0x05;

    /* SPI enabled, master mode, serial clock pulses at beginning of xfer,
     * serial clock idles high, MSB transmitted first, transfer on write to
     * SPITX, interrupt when TX empty (SPI interrupt not enabled in IRQEN)
     * overwrite SPIRX when new serial byte received
     */
    SPICON = 0x14F;
}

/** timer0_init() --
 **     Configures timer0 for while loop period keeping
 **/
void timer0_init(void)
{
    /* timer0 currently set to 1000 us period */
    timer0_Flag = 0;    /* Re-initialize flag */
    IRQEN |= 0x04;      /* Re-initialize timer IRQ */

    /* 0x1053 = 100us period at 41.78MHz clock freq (set in system_init());
     * 0x20A6 for 200us; x30F9 for 300us; x519A for 500us; x7A67 for 750us;
     * 0xA334 for 1000us; 0x14668 for 2000us;
     * 0x829 for 50us; 0x414 for 25us; 0x20A for 12.5us
     */
    T0LD = 0xA334;

    /* enable timer0, set in periodic mode with multiplier = 1 */
    T0CON = 0xC0;
}

/** General_IRQ_Handler() --
 **     specifies behavior when an IRQ condition is reached
 **/
void General_IRQ_Handler(void)
{
    if ((IRQSIG & 0x04) == 0x4)
    {
        /* timer0 IRQ clear routine */
        timer0_Flag = 1;
        T0CLRI = 0xFF;
        return;
    }

    else if ((IRQSIG & 0x8) == 0x8)
    {
        /* timer1 IRQ clear for us/ms delay */
        timer1_Flag = 1;
        T1CLRI = 0xFF;
        return;
    }

    #if TEMP_DISABLED
    if ((IRQSIG & 0xD) == 0xD)
    {
        /* SPI IRQ clear on transmit */
        //SPITX_flag = 1;
        return ;
    }
    #endif

}

/** pot_voltage() --
 **     Calculates voltage on a potentiometer. Returns decimal scaled
 **     by fullscaleValue
 **/
float pot_voltage(float voltage, float fullscaleValue)
{
    float calculatedVoltage;
    calculatedVoltage = (voltage / (float) 2.52) * (fullscaleValue);
    return calculatedVoltage;
}

/** waitForRestOfPeriod() --
 **     Maintains 200us cycle using timer0 IRQ and global flags
 **/
void waitForRestOfPeriod(void)
{
```

```
    /* flip P2.0 low, to know when waitForRestOfPeriod() begins */
    GP2DAT ^= 0x10000;

    if (timer0_Flag == 1)
    {
        timer0_Flag = 0;    /* Reset timer flag */
        return;
    }
    else
    {
        while(timer0_Flag == 0){
            /* Wait until interrupt request */
        }
        timer0_Flag = 0;
        return;
    }
}

/** ADCinput() --
 **     Converts ADC conversion HEX to decimal voltage.
 **/
float ADCinput(int ADC_hex_result)
{
    float input_voltage;
    int ADC_hex_result_shifted;

    /* shift to right 16 bits to obtain base integer */
    ADC_hex_result_shifted = ADC_hex_result >> 16;

    /* "unit conversion" using 2.52V with ADC connected to voltmeter */
    input_voltage = ((ADC_hex_result_shifted)*2.52)/(0xFFF);

    return input_voltage;
}

/** outputDAC() --
 **     Converts decimal voltage to HEX for DAC output.
 **/
int outputDAC(float desired_voltage)
{
    /* "unit conversion" from desired decimal voltage to hex code
     *  2.4976 is voltage when 0xFFF applied to DAC
     */
    int output_integer = floor((desired_voltage * 0xFFF) / 2.497);

    /* majority of voltage range is [0.036, 2.49] */
    if (desired_voltage < 2.49 && desired_voltage >= 0.036)
    {

        /* Shift integer result 16 bits to left to conform to DACxDAT format*/
        output_integer = output_integer << 16;
    }
    else if (desired_voltage >= 2.49)
    {
        output_integer = 0xFFF;
        output_integer = output_integer << 16;
    }
    /* Voltages under 0.036V need a function to be more accurate */
    else if (desired_voltage < 0.036 && desired_voltage >= 0.0065)
    {
        /* Linear function obtained by excel calibration for small voltages */
        desired_voltage = 1.2 * desired_voltage - 0.0072733;
        output_integer = floor((desired_voltage * 0xFFF) / 2.497);
        output_integer = output_integer << 16;
    }
    /* The minimum voltage we can get */
    else if (desired_voltage < 0.0065)
    {
        output_integer = 0x00;
    }

    return output_integer;
}

/** delay_sw() --
 **     Software delay. time = 10 --> ~12us
 **/
void delay_sw(int time)
{
    while (time >=0)
    time --;
}

/** delay_us() --
 **     Microsecond delay on timer1. Enables/disables timer1 within function
 **/
void delay_us(unsigned long time)
{
    timer1_Flag = 0;
    T1CON = 0x0;

    /* enable timer1 IRQ, timer1 = bit3 */
    IRQEN |= 0x08;

    /* T1LD = time (us) * clock freq (MHz)
```

```
     * (e.g. 500us * 41.78 MHz = 20890 - 1)
     * subtract 1 since timer load includes zero
     */
    T1LD = ((time * 42782) >> 10) - 1;

    /* initialize timer1, 41.78MHz, down mode, periodic */
    T1CON = 0xC0;
    while(timer1_Flag == 0){
        /* Wait for timer flag flip */
    }
    timer1_Flag = 0;
    T1CON = 0x0;            /* Disable timer1 */
    IRQCLR |= 0x08;
    return;
}

/** delay_ms() --
 **     Millisecond delay on timer1 using delay_us().
 **/
void delay_ms(unsigned long time)
{
    unsigned short i = 0;

    /* Wait for 1 ms each time through loop */
    for (i=0; i<time; i++)
    {
        delay_us(1000);
    }
}

/** testmode() --
 **     Cycle DAC output up and down n times in a triangular wave
 **/
void testmode(int times)
{
    int n;
    float voltage;

    for (n=1; n<=times; ++n)
    {
        /* Cycle up to 2.49V */
        for (voltage=1.25; voltage<=2.5; voltage=voltage+0.001)
        {
            DAC0DAT = outputDAC(voltage);
            delay_ms(2);
        }

        delay_ms(50);

        /* Cycle down to 0.0065V */
        for (voltage=2.5; voltage>=0.0065; voltage=voltage-0.001)
        {
            DAC0DAT = outputDAC(voltage);
            delay_ms(2);
        }

        delay_ms(50);

        /* Cycle up to 2.49V */
        for (voltage=0.0065; voltage<=1.25; voltage=voltage+0.001)
        {
            DAC0DAT = outputDAC(voltage);
            delay_ms(2);
        }

        /* Return to midpoint */
        DAC0DAT = outputDAC(1.25);

        delay_ms(50);
    }

    #if OBSOLETE
    /* ALTERNATE/OBSOLETE Implementation */
    float voltage=0.0065,step=0.001;

    DAC2DAT = outputDAC(1.25);

    while ( times >= 0 )
    {

        delay_ms(2);
        voltage += step;

        if ( voltage >= 2.5 ) // Cycle down to 0.0065V
        {
            DAC2DAT = outputDAC(2.5);
         step = -0.001;
         delay_ms(100);
        }
        if ( voltage <= 0.0064) // Cycle up to 2.49V
        {
            DAC2DAT = outputDAC(0.0065);
            step = 0.001;
            --times;
            delay_ms(50);
```

```
        }
        DAC2DAT = outputDAC(voltage);
    }
    #endif
}

/** sensor_avg() --
 **     Takes 10 sample average of indicated channel
 **/
float sensor_avg(int channel)
{
    int i;
    float avg_voltage, add_voltage = 0;
    float voltage[10];

    for (i=0; i<10; i++)
    {
        voltage[i] = read_store_sensors(channel, pCurr_data);
        add_voltage += voltage[i];
        delay_us(10);
    }
    avg_voltage = add_voltage / 10.0;

    return avg_voltage;
}

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

E.2.2 Wixel “Slave” SPI to USB Interface Source Code

```
#include <wixel.h>
#include <usb.h>
#include <usb_com.h>
#include <stdio.h>
#include <spi.h>

#define SPI0

#define INTERRUPT_PRIORITY_GROUP    2

#define URXNIF                      URX0IF
#define URXNIE                      URX0IE
#define UNGCR                       U0GCR
#define UNBAUD                      U0BAUD
#define UNDBUF                      U0DBUF
#define spiNSlaveInit               spi0SlaveInit
#define spiNSlaveSetFrequency       spi0SlaveSetFrequency
#define spiNSlaveSetClockPolarity   spi0SlaveSetClockPolarity
#define spiNSlaveSetClockPhase      spi0SlaveSetClockPhase
#define spiNSlaveSetBitOrder        spi0SlaveSetBitOrder

void spi0SlaveInit(void);
void spi0SlaveSetFrequency(uint32 freq);
void spi0SlaveSetClockPolarity(BIT polarity);
void spi0SlaveSetClockPhase(BIT phase);
void spi0SlaveSetBitOrder(BIT bitOrder);
void spiByteFinished() __interrupt 2 __using 0;

// txbuffer is used for bytes sent from upper computer.
static volatile uint8 XDATA txbuffer[256] = {0xAA, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
0xff, 0XBB};

// txPointer points to the next byte in txbuffer that need to send to SPI.
static volatile const uint8 XDATA * DATA txPointer;

// bytesLeft is the number of bytes we still need to send to SPI.
static volatile uint8 DATA bytesLeft = 0;

BIT yellowLedOn = 0;
BIT redLedOn = 0;

void spiNSlaveInit(void)
{
    P2DIR &= ~0xC0;  // P2DIR.PRIP0 (7:6) = 00 : USART0 takes priority over USART1.
    PERCFG &= ~0x01; // PERCFG.U0CFG (0) = 0 (Alt. 1) : USART0 uses alt. location 1.

    // The MODE bit in U0CSR/U1CSR is 0(SPI mode) by default.
    U0CSR |= 0x20;   // Set the mode of the USART to SPI Slave mode.

          // Set the mode of the SCK and MOSI pins to "peripheral function".
          P0SEL |= 0x3C; // P0SEL.SELP0_5 = P0SEL.SELP0_4 = P0SEL.SELP0_3 = P0SEL.SELP0_2 = 1

    IP0 |= (1<<INTERRUPT_PRIORITY_GROUP);
    IP1 &= ~(1<<INTERRUPT_PRIORITY_GROUP);

    URXNIF = 0; // Clear RX flag.
    EA = 1;     // Enable interrupts in general.
}

void spiNSlaveSetFrequency(uint32 freq)
{
    uint32 baudMPlus256;
    uint8 baudE = 0;

    // max baud rate is 3000000 (F/8); min is 23 (baudM = 1)
    if (freq < 23 || freq > 3000000)
        return;

    // 495782 is the largest value that will not overflow the following calculation
    while (freq > 495782)
    {
        baudE++;
        freq /= 2;
    }

    // calculate baud rate - see datasheet 12.14.3
    // this is derived from (baudM + 256) = baud * 2^28 / 24000000
    baudMPlus256 = (freq * 11) + (freq * 8663 / 46875);

    // get baudMPlus256 into the range 256-511 (so BAUD_M is in the range 0-255)
    while (baudMPlus256 > 0x1ff)
    {
        baudE++;
        baudMPlus256 /= 2;
```

```
        }
        UNGCR &= 0xE0; // preserve CPOL, CPHA, ORDER (7:5)
        UNGCR |= baudE; // UNGCR.BAUD_E (4:0)
        UNBAUD = baudMPlus256; // UNBAUD.BAUD_M (7:0) - only the lowest 8 bits of baudMPlus256 are used, so this is
effectively baudMPlus256 - 256
    }

    void spiNSlaveSetClockPolarity(BIT polarity)
    {
        if (polarity == SPI_POLARITY_IDLE_LOW)
        {
            UNGCR &= ~(1<<7);   // SCK idle low (negative polarity)
        }
        else
        {
            UNGCR |= (1<<7);    // SCK idle high (positive polarity)
        }
    }

    void spiNSlaveSetClockPhase(BIT phase)
    {
        if (phase == SPI_PHASE_EDGE_LEADING)
        {
            UNGCR &= ~(1<<6);   // data centered on leading (first) edge - rising for idle low, falling for idle high
        }
        else
        {
            UNGCR |= (1<<6);    // data centered on trailing (second) edge - falling for idle low, rising for idle high
        }
    }

    void spiNSlaveSetBitOrder(BIT bitOrder)
    {
        if (bitOrder == SPI_BIT_ORDER_LSB_FIRST)
        {
            UNGCR &= ~(1<<5);   // LSB first
        }
        else
        {
            UNGCR |= (1<<5);    // MSB first
        }
    }

    /*
     * RxReceiveByte2txbuffer - Get bytes from upper computer and put them into txbuffer.
     */
    void RxReceiveByte2txbuffer(/*uint8 XDATA * txBuffer*/)
    {
              /* Detect how many bytes upper computer has sent */
              if (usbComRxAvailable() >= 14 && bytesLeft == 0) {
            usbComRxReceive(txbuffer, 14);
            yellowLedOn = 1;
    //        strncpy(txbuffer2, txbuffer, 14);
        }

        /* Read those availalbe bytes and store them to txbuffer */
        yellowLedOn = 0;
    }

    void updateLeds()
    {
        usbShowStatusWithGreenLed();
        LED_YELLOW(yellowLedOn);
        LED_RED(redLedOn);
    }


    void main()
    {
              txPointer = txbuffer;

              systemInit();
        usbInit();
              spi0SlaveInit();
              spi0SlaveSetFrequency(250000);
              spi0SlaveSetClockPolarity(SPI_POLARITY_IDLE_HIGH);
              spi0SlaveSetClockPhase(SPI_PHASE_EDGE_TRAILING);
              spi0SlaveSetBitOrder(SPI_BIT_ORDER_MSB_FIRST);
              URXNIE = 1; // Enable RX interrupt.

              while(1)
        {
                        redLedOn = 0;
                        boardService();
                        usbComService();
                        updateLeds();
                        RxReceiveByte2txbuffer();
    //                  URXNIE = 1;
              }
    }


    /*
     * spiByteFinished - interrupt routine. It will be invoked every time a whole SPI byte has arrived.
     */
```

```
void spiByteFinished() __interrupt 2 __using 0
{
//        URXNIE = 0;
//        URXNIF = 0;

    /* Mark whether AD device is ready to receive message from wixel */
    static int flag = 0;

    /* If txbuffer of usbcom still has space, the arrived SPI byte(stored in register UNDBUF) will
     * be sent to that buffer. Bytes sent to that buffer could be read by program on upper computer. */
    if (usbComTxAvailable())
        usbComTxSendByte(UNDBUF); //read byte from UNDBUF

    /* If we are not sending meaningful data to AD device and the arrived data from AD device now is start
     * byte 0xAA, then we set the flag to be 1, which marked the start of message sending to AD device*/
    if (UNDBUF == 0xAA && flag == 0) {
        flag = 1;
        bytesLeft = 14;
    }

    /* If the 14-byte message sending has not completed and flag is 1, then we send the next byte to AD device */
    if (flag == 1) {
        UNDBUF = *txPointer;      //write byte to UNDBUF
        txPointer++;
        bytesLeft--;
        /* If the last byte of the message has been sent, we reset the txPointer and set flag to be 0 */
        if(!bytesLeft) {
            txPointer = txbuffer;
            flag = 0;
        }
    }
    /* Otherwise error has happened and we just put 0xFF into register UNDBUF */
    else {
                    UNDBUF = 0xFF;
        redLedOn = 1;
          }

//        URXNIE = 1;
}
```

E.2.3 Haptic Renderer GUI Source Code

```
// SerialDlg.cpp : implementation file
//

#include "stdafx.h"
#include "Serial.h"
#include "SerialDlg.h"
#include "afxdialogex.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

// CAboutDlg dialog used for App About
class CAboutDlg : public CDialogEx
{
public:
          CAboutDlg();

// Dialog Data
          enum { IDD = IDD_ABOUTBOX };

          protected:
          virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support

// Implementation
protected:
          DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialogEx(CAboutDlg::IDD)
{
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
          CDialogEx::DoDataExchange(pDX);
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialogEx)
END_MESSAGE_MAP()


// CSerialDlg dialog
CSerialDlg::CSerialDlg(CWnd* pParent /*=NULL*/)
          : CDialogEx(CSerialDlg::IDD, pParent)
          //, m_strReceive(_T(""))
          //, m_strSend(_T(""))
          //, m_autoSend(FALSE)
          //, m_repeatInterval(_T(""))
          , m_pause(FALSE)
          , m_log(FALSE)
          , errcnt(0)
          , m_logOri(FALSE)
          , m_strR1(_T(""))
          , m_strR2(_T(""))
          , m_strR3(_T(""))
          , m_strR4(_T(""))
          , m_strR5(_T(""))
          , m_strR6(_T(""))
          , m_strR7(_T(""))
          , m_strR8(_T(""))
          , m_strV1(_T(""))
          , m_strV2(_T(""))
          , m_strV3(_T(""))
          , m_strV4(_T(""))
          , m_strV5(_T(""))
          , m_strV6(_T(""))
          , m_strV7(_T(""))
          , m_strV8(_T(""))
          , m_strM1(_T(""))
          , m_radioMode(0)
{
          m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
          cnt = 0;
          //m_hAcce1 = ::LoadAccelerators(AfxGetInstanceHandle(), MAKEINTRESOURCE(IDR_ACCELERATOR1));
          //m_hAcce2 = ::LoadAccelerators(AfxGetInstanceHandle(), MAKEINTRESOURCE(IDR_ACCELERATOR2));
}

void CSerialDlg::DoDataExchange(CDataExchange* pDX)
{
          CDialogEx::DoDataExchange(pDX);
          DDX_Control(pDX, IDC_MSCOMM1, m_mycom);
          DDX_Control(pDX, IDC_ChartCtrl1, m_ChartCtrl1);
          DDX_Control(pDX, IDC_ChartCtrl2, m_ChartCtrl2);
```

```
	DDX_Control(pDX, IDC_ChartCtrl3, m_ChartCtrl3);
	//DDX_Text(pDX, IDC_EDIT1, m_strReceive);
	//DDX_Text(pDX, IDC_EDIT2, m_strSend);
	//DDX_Check(pDX, IDC_CHECK1, m_autoSend);
	//DDX_Text(pDX, IDC_EDIT3, m_repeatInterval);
	DDX_Check(pDX, IDC_CHECK2, m_pause);
	DDX_Check(pDX, IDC_CHECK3, m_log);
	DDX_Text(pDX, IDC_EDIT7, errcnt);
	DDX_Check(pDX, IDC_CHECK4, m_logOri);
	//DDX_Control(pDX, IDC_SPIN1, m_Spin1);
	//DDX_Control(pDX, IDC_SLIDER1, m_Slider1);

	DDX_Text(pDX, IDC_EDIT1, m_strKnob1);
	DDX_Text(pDX, IDC_EDIT2, m_strKnob2);
	DDX_Text(pDX, IDC_EDIT32, m_strKnob3);

	DDX_Text(pDX, IDC_EDIT4, m_strR1);
	DDX_Text(pDX, IDC_EDIT5, m_strR2);
	DDX_Text(pDX, IDC_EDIT6, m_strR3);
	DDX_Text(pDX, IDC_EDIT11, m_strR4);
	DDX_Text(pDX, IDC_EDIT12, m_strR5);
	DDX_Text(pDX, IDC_EDIT13, m_strR6);
	DDX_Text(pDX, IDC_EDIT14, m_strR7);
	DDX_Text(pDX, IDC_EDIT15, m_strR8);
	DDX_Text(pDX, IDC_EDIT16, m_strV1);
	DDX_Text(pDX, IDC_EDIT17, m_strV2);
	DDX_Text(pDX, IDC_EDIT18, m_strV3);
	DDX_Text(pDX, IDC_EDIT19, m_strV4);
	DDX_Text(pDX, IDC_EDIT20, m_strV5);
	DDX_Text(pDX, IDC_EDIT21, m_strV6);
	DDX_Text(pDX, IDC_EDIT22, m_strV7);
	DDX_Text(pDX, IDC_EDIT23, m_strV8);

	DDX_Text(pDX, IDC_EDIT25, m_strM1);

	DDX_Control(pDX, IDC_SPIN1, m_Spin1);
	DDX_Control(pDX, IDC_SLIDER1, m_Slider1);
	DDX_Control(pDX, IDC_SPIN2, m_Spin2);
	DDX_Control(pDX, IDC_SLIDER2, m_Slider2);
	DDX_Control(pDX, IDC_SPIN3, m_Spin3);
	DDX_Control(pDX, IDC_SLIDER4, m_Slider3);

	DDX_Radio(pDX, IDC_RADIO5, m_radioMode);
	DDV_MinMaxInt(pDX, m_radioMode, 0, 5);
}

BEGIN_MESSAGE_MAP(CSerialDlg, CDialogEx)
	ON_WM_SYSCOMMAND()
	ON_WM_PAINT()
	ON_WM_QUERYDRAGICON()
	ON_BN_CLICKED(IDC_BUTTON1, &CSerialDlg::OnBnClickedButton1)
	ON_BN_CLICKED(IDC_BUTTON2, &CSerialDlg::OnBnClickedButton2)
	//ON_BN_CLICKED(IDC_BUTTON5, &CSerialDlg::OnBnClickedButton5)
	//ON_BN_CLICKED(IDC_BUTTON3, &CSerialDlg::OnBnClickedButton3)
	ON_WM_TIMER()
	//ON_BN_CLICKED(IDC_CHECK1, &CSerialDlg::OnBnClickedCheck1)
	ON_BN_CLICKED(IDC_CHECK3, &CSerialDlg::OnBnClickedCheck3)
	ON_BN_CLICKED(IDC_CHECK4, &CSerialDlg::OnBnClickedCheck4)
	//ON_NOTIFY(UDN_DELTAPOS, IDC_SPIN1, &CSerialDlg::OnDeltaposSpin1)
	//ON_NOTIFY(NM_CUSTOMDRAW, IDC_SLIDER1, &CSerialDlg::OnNMCustomdrawSlider1)
	ON_BN_CLICKED(IDC_BUTTON4, &CSerialDlg::OnBnClickedButton4)
	ON_BN_CLICKED(IDC_RADIO5, &CSerialDlg::OnBnClickedRadio5)
	ON_BN_CLICKED(IDC_RADIO6, &CSerialDlg::OnBnClickedRadio6)
	ON_BN_CLICKED(IDC_RADIO7, &CSerialDlg::OnBnClickedRadio7)
	ON_BN_CLICKED(IDC_RADIO8, &CSerialDlg::OnBnClickedRadio8)
	ON_BN_CLICKED(IDC_RADIO9, &CSerialDlg::OnBnClickedRadio9)
	ON_BN_CLICKED(IDC_RADIO10, &CSerialDlg::OnBnClickedRadio10)
	//ON_COMMAND(IDR_ACCELERATOR1, &CSerialDlg::OnIdrAccelerator1)
	ON_NOTIFY(UDN_DELTAPOS, IDC_SPIN1, &CSerialDlg::OnDeltaposSpin1)
	ON_NOTIFY(NM_CUSTOMDRAW, IDC_SLIDER1, &CSerialDlg::OnNMCustomdrawSlider1)
	ON_NOTIFY(UDN_DELTAPOS, IDC_SPIN2, &CSerialDlg::OnDeltaposSpin2)
	ON_NOTIFY(UDN_DELTAPOS, IDC_SPIN3, &CSerialDlg::OnDeltaposSpin3)
	ON_NOTIFY(NM_CUSTOMDRAW, IDC_SLIDER2, &CSerialDlg::OnNMCustomdrawSlider2)
	ON_NOTIFY(NM_CUSTOMDRAW, IDC_SLIDER4, &CSerialDlg::OnNMCustomdrawSlider4)
END_MESSAGE_MAP()

BEGIN_EVENTSINK_MAP(CSerialDlg, CDialogEx)
	ON_EVENT(CSerialDlg, IDC_MSCOMM1, 1, CSerialDlg::OnCommMscomm1, VTS_NONE)
END_EVENTSINK_MAP()

// CSerialDlg message handlers

BOOL CSerialDlg::OnInitDialog()
{
	CDialogEx::OnInitDialog();

	// Add "About..." menu item to system menu.

	// IDM_ABOUTBOX must be in the system command range.
	ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
	ASSERT(IDM_ABOUTBOX < 0xF000);

	CMenu* pSysMenu = GetSystemMenu(FALSE);
	if (pSysMenu != NULL)
	{
```

```
		BOOL bNameValid;
		CString strAboutMenu;
		bNameValid = strAboutMenu.LoadString(IDS_ABOUTBOX);
		ASSERT(bNameValid);
		if (!strAboutMenu.IsEmpty())
		{
			pSysMenu->AppendMenu(MF_SEPARATOR);
			pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
		}
	}

	// Set the icon for this dialog.  The framework does this automatically
	//  when the application's main window is not a dialog
	SetIcon(m_hIcon, TRUE);			// Set big icon
	SetIcon(m_hIcon, FALSE);		// Set small icon

	// TODO: Add extra initialization here
	CButton* radio;
	//radio = (CButton*)GetDlgItem(IDC_RADIO1);
	//radio->SetCheck(1);
	//
	//radio = (CButton*)GetDlgItem(IDC_RADIO3);
	//radio->SetCheck(1);

	//OutputMsg initialization
	OutputMsg.SetSize(14);
	OutputMsg[0] = 0xAA;
	OutputMsg[13] = 0xBB;
	for (int i = 1; i < 13; i++)
		OutputMsg[i] = 0;
	modeCmd = 0x99;
	knob1 = 0;
	knob2 = 0;
	knob3 = 0;
	radio = (CButton*)GetDlgItem(IDC_RADIO5);
	radio->SetCheck(1);

	//Knob
	m_Spin1.SetBuddy(GetDlgItem(IDC_SLIDER1));
	//m_Spin1.SetRange(0, 100);
	m_Slider1.SetRange(0,4095);
	m_Slider1.SetTicFreq(100);
	m_Slider1.SetLineSize(1);
	m_Slider1.SetPageSize(10);

	m_Spin2.SetBuddy(GetDlgItem(IDC_SLIDER2));
	m_Slider2.SetRange(0, 4095);
	m_Slider2.SetTicFreq(100);
	m_Slider2.SetLineSize(1);
	m_Slider2.SetPageSize(10);

	m_Spin3.SetBuddy(GetDlgItem(IDC_SLIDER4));
	m_Slider3.SetRange(0, 4095);
	m_Slider3.SetTicFreq(100);
	m_Slider3.SetLineSize(1);
	m_Slider3.SetPageSize(10);


	//Graph
	CChartAxis *pAxis = NULL;
	pAxis = m_ChartCtrl1.CreateStandardAxis(CChartCtrl::BottomAxis);
	pAxis->SetAutomatic(true);
	pAxis = m_ChartCtrl1.CreateStandardAxis(CChartCtrl::LeftAxis);
	pAxis->SetMinMax(0, 2.5);

	pAxis = m_ChartCtrl2.CreateStandardAxis(CChartCtrl::BottomAxis);
	pAxis->SetAutomatic(true);
	pAxis = m_ChartCtrl2.CreateStandardAxis(CChartCtrl::LeftAxis);
	pAxis->SetMinMax(0, 2.5);

	pAxis = m_ChartCtrl3.CreateStandardAxis(CChartCtrl::BottomAxis);
	pAxis->SetAutomatic(true);
	pAxis = m_ChartCtrl3.CreateStandardAxis(CChartCtrl::LeftAxis);
	pAxis->SetMinMax(0, 2.5);

	TChartString str;
	str = _T("Force Sensor");
	m_ChartCtrl1.GetTitle()->AddString(str);
	str = _T("Optical Displacement");
	m_ChartCtrl2.GetTitle()->AddString(str);
	str = _T("Speaker Output");
	m_ChartCtrl3.GetTitle()->AddString(str);

	//initialize dataset
	for (int i = 0; i<10000; i++) {
		x[i] = i;
		y1[i] = 0;
		y2[i] = 0;
		y3[i] = 0;
	}

	m_hAcce1 = ::LoadAccelerators(AfxGetInstanceHandle(), MAKEINTRESOURCE(IDR_ACCELERATOR1));

	return TRUE;  // return TRUE  unless you set the focus to a control
}
```

```
void CSerialDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
	if ((nID & 0xFFF0) == IDM_ABOUTBOX)
	{
		CAboutDlg dlgAbout;
		dlgAbout.DoModal();
	}
	else
	{
		CDialogEx::OnSysCommand(nID, lParam);
	}
}

// If you add a minimize button to your dialog, you will need the code below
//  to draw the icon.  For MFC applications using the document/view model,
//  this is automatically done for you by the framework.

void CSerialDlg::OnPaint()
{
	if (IsIconic())
	{
		CPaintDC dc(this); // device context for painting

		SendMessage(WM_ICONERASEBKGND, reinterpret_cast<WPARAM>(dc.GetSafeHdc()), 0);

		// Center icon in client rectangle
		int cxIcon = GetSystemMetrics(SM_CXICON);
		int cyIcon = GetSystemMetrics(SM_CYICON);
		CRect rect;
		GetClientRect(&rect);
		int x = (rect.Width() - cxIcon + 1) / 2;
		int y = (rect.Height() - cyIcon + 1) / 2;

		// Draw the icon
		dc.DrawIcon(x, y, m_hIcon);
	}
	else
	{
		CDialogEx::OnPaint();
	}
}

// The system calls this function to obtain the cursor to display while the user drags
//  the minimized window.
HCURSOR CSerialDlg::OnQueryDragIcon()
{
	return static_cast<HCURSOR>(m_hIcon);
}


/*
 *  Open button handler
 */
void CSerialDlg::OnBnClickedButton1()
{
	// TODO: Add your control notification handler code here
	m_mycom.put_CommPort(3);
	m_mycom.put_PortOpen(TRUE);
	m_mycom.put_Settings(_T("1200,n,8,1"));
	m_mycom.put_InputMode(1);
	m_mycom.put_RThreshold(1);
//	m_mycom.put_InBufferSize(10000);
	m_mycom.put_InputLen(0);
	m_mycom.get_Input();
//	logfile = fopen(".\logfile.txt", "w");
	fopen_s(&log, "log.txt", "w");
	SetTimer(2, 1, NULL);
	SetTimer(3, 0, NULL);
	MessageBox(_T("COM is open"));
}


/*
 *  Close button handler
 */
void CSerialDlg::OnBnClickedButton2()
{
	// TODO: Add your control notification handler code here
	KillTimer(1);
	KillTimer(2);

	m_mycom.put_PortOpen(FALSE);
	fclose(log);
	MessageBox(_T("COM is closed"));
}


/*
 *  Mscomm event handler
 */
void CSerialDlg::OnCommMscomm1()
{
	// TODO: Add your message handler code here
	COleSafeArray safearray_input;
```

```
	LONG len, byteleft, k = 0;
	BYTE bt0;
	int i;

	UpdateData(TRUE);

	if (m_mycom.get_CommEvent() == 2) //Event number 2 means there is characters in rxbuffer
	{
		//if pause, we simply discard all the input data in receive buffer
		if (m_pause) {
			m_mycom.get_Input();
			return;
		}
		safearray_input = (COleSafeArray)m_mycom.get_Input();
		len = safearray_input.GetOneDimSize();
		//record the length of input date received this time
		fprintf(log, "length: %d\n", len);

		//Check whether the last message constructed in last loop is finished
		if (cnt) {
			//Continue constructing left part of the message
			for (i = cnt; i < 13; i++) {
				safearray_input.GetElement(&k, Msg + i);
				k++;
			}
			SendMsg(Msg, &k);   //Send message
			cnt = 0;  //clear cnt
		}

		for (k; k < len; k++)
		{
			//Detect start symbol of message
			safearray_input.GetElement(&k, &bt0);
			if (bt0 == 0xAA) {
				byteleft = len - k - 1;
				if (byteleft >= 13) {         //Enough byte left to form up a message
					for (i = 0; i < 13; i++) {
						k++;
						safearray_input.GetElement(&k, Msg + i);
					}
					SendMsg(Msg, &k); //Send message
				}
				else {    //Less than 5 bytes left
					//just put them into message arrary and set cnt
					cnt = byteleft;
					for (i = 0; i < cnt; i++) {
						k++;
						safearray_input.GetElement(&k, Msg + i);
					}
				}
			}
			//Some error happend, enter next loop to detect next byte
			else
				continue;
		}
	}
}

void CSerialDlg::SendMsg(BYTE* msg, LONG* pk) {
	float v[8];
	int raw[8];
	int i;

	if (msg[12] != 0xBB) {        // Verify the message
		// If the last byte is not 0xBB,  we treat it as a wrong message
		*pk = *pk - (13-cnt);
		errcnt++;
		for (i = 0; i < 8; i++) {
			v[i] = -1;
			raw[i] = -1;
		}
	}
	else
		Parse(msg, v, raw); // Parse the message to three real voltages

	// Update GUI and several logs
	Update(msg, v, raw);
}

void Parse(BYTE* msg, float* v, int* raw)
{
	// Attention: big endian and little endian
	WORD w, x;
	int i;

	for (i = 0; i < 4; i++) {
		w = *(WORD*)(msg + i*3);
		x = (w >> 12) + (w << 4) & 0x0fff;
		raw[2*i] = x;
		v[2*i] = 2.52 / 4095 * x;

		w = *(WORD*)(msg + i*3 + 1);
		x = (w >> 8) + (w << 8) & 0x0fff;
		raw[2*i+1] = x;
		v[2*i+1] = 2.52 / 4095 * x;
```

```
		}
	}

	void CSerialDlg::Update(BYTE* msg, float* v, int* raw)
	{
		CString strtemp;
		CString time;
		CTime tm;

		//Construct time segment
		SYSTEMTIME st;
		GetSystemTime(&st);
		time.Format(_T("%d/%d/%d %d:%d:%d.%d"), st.wMonth, st.wDay, st.wYear,
			st.wHour, st.wMinute, st.wSecond, st.wMilliseconds);
		//Alternate way(but less accurate): tm = CTime::GetCurrentTime(); time = tm.Format("%Y %m %d %X");

		//Update raw data values on GUI
		m_strR1.Format(_T("%d"), raw[0]);
		m_strR2.Format(_T("%d"), raw[1]);
		m_strR3.Format(_T("%d"), raw[2]);
		m_strR4.Format(_T("%d"), raw[3]);
		m_strR5.Format(_T("%d"), raw[4]);
		m_strR6.Format(_T("%d"), raw[5]);
		m_strR7.Format(_T("%d"), raw[6]);
		m_strR8.Format(_T("%d"), raw[7]);

		//Update voltage data values on GUI
		m_strV1.Format(_T("%.3f"), v[0]);
		m_strV2.Format(_T("%.3f"), v[1]);
		m_strV3.Format(_T("%.3f"), v[2]);
		m_strV4.Format(_T("%.3f"), v[3]);
		m_strV5.Format(_T("%.3f"), v[4]);
		m_strV6.Format(_T("%.3f"), v[5]);
		m_strV7.Format(_T("%.3f"), v[6]);
		m_strV8.Format(_T("%.3f"), v[7]);

		//Update meaning data values on GUI
		//m_strM1.Format(_T("%.3f"), m[1]);

		//Update the y axis array
		for (int i = 1; i < 10000; i++)
		{
			y1[i - 1] = y1[i];
			y2[i - 1] = y2[i];
			y3[i - 1] = y3[i];
		}
		y1[9999] = v[0];
		y2[9999] = v[1];
		y3[9999] = v[3];


		//Print the translated voltage data on logfile.txt
		if (m_log) {
			strtemp.Format(_T("\t%.3f\t %.3f\t %.3f\t %.3f\t %.3f\t %.3f\t %.3f\t %.3f"), v[0], v[1], v[2],
v[3], v[4], v[5], v[6], v[7]);
			USES_CONVERSION;
			fprintf(logfile, "(%s) %s\n", T2A(time),T2A(strtemp));
		}

		//Print the original hexidecimal data on logfileOri.txt
		if (m_logOri) {
			strtemp.Format(_T("%02X %02X %02X %02X %02X %02X %02X %02X %02X %02X %02X %02X"), msg[0], msg[1],
msg[2], msg[3], msg[4], msg[5], msg[6], msg[7], msg[8], msg[9], msg[10], msg[11]);
			USES_CONVERSION;
			fprintf(logfileOri, "(%s) %s\n", T2A(time), T2A(strtemp));
		}

		UpdateData(FALSE);
	}

	int String2Hex(CString str, CByteArray &senddata)
	{
		int hexdata, lowhexdata;
		int hexdatalen = 0;
		int len = str.GetLength();
		senddata.SetSize(len / 2);
		for (int i = 0; i<len;) {
			char lstr, hstr = str[i];
			if (hstr == ' ') {
				i++;
				continue;
			}
			i++;
			if (i >= len) break;
			lstr = str[i];
			hexdata = ConvertHexChar(hstr);
			lowhexdata = ConvertHexChar(lstr);
			if ((hexdata == -1) || (lowhexdata == -1))
				break;
			else
				hexdata = hexdata * 16 + lowhexdata;
			i++;
			senddata[hexdatalen] = (char)hexdata;
			hexdatalen++;
		}
```

```
		senddata.SetSize(hexdatalen);
		return hexdatalen;
}

char ConvertHexChar(char ch)
{
		if ((ch >= '0') && (ch <= '9'))
			return ch - 0x30;
		else if ((ch >= 'A') && (ch <= 'F'))
			return ch - 'A' + 10;
		else if ((ch >= 'a') && (ch <= 'f'))
			return ch - 'a' + 10;
		else return -1;
}

/*
*  Log button handler
*/
void CSerialDlg::OnBnClickedCheck3()
{
		// TODO: Add your control notification handler code here
		UpdateData(TRUE);

		if (m_log)
			fopen_s(&logfile, "logfile.txt", "w");
		else
			fclose(logfile);
}


void CSerialDlg::OnBnClickedCheck4()
{
		// TODO: Add your control notification handler code here
		UpdateData(TRUE);

		if (m_logOri)
			fopen_s(&logfileOri, "logfileOri.txt", "w");
		else
			fclose(logfileOri);
}

void CSerialDlg::OnBnClickedButton4()
{
		// TODO: Add your control notification handler code here
		errcnt = 0;
		UpdateData(FALSE);
}


void CSerialDlg::OnBnClickedRadio5()
{
		// TODO: Add your control notification handler code here
		modeCmd = 0x99;
		m_radioMode = 0;
}


void CSerialDlg::OnBnClickedRadio6()
{
		// TODO: Add your control notification handler code here
		modeCmd = 0xAA;
		m_radioMode = 1;
}


void CSerialDlg::OnBnClickedRadio7()
{
		// TODO: Add your control notification handler code here
		modeCmd = 0xBB;
		m_radioMode = 2;
}


void CSerialDlg::OnBnClickedRadio8()
{
		// TODO: Add your control notification handler code here
		modeCmd = 0xCC;
		m_radioMode = 3;
}


void CSerialDlg::OnBnClickedRadio9()
{
		// TODO: Add your control notification handler code here
		modeCmd = 0xDD;
		m_radioMode = 4;
}


void CSerialDlg::OnBnClickedRadio10()
{
		// TODO: Add your control notification handler code here
		modeCmd = 0xEE;
		m_radioMode = 5;
}
```

```
BOOL CSerialDlg::PreTranslateMessage(MSG* pMsg)
{
	if (WM_KEYFIRST <= pMsg->message && pMsg->message <= WM_KEYLAST) {
		if (m_hAcce1) {
			if (::TranslateAccelerator(m_hWnd, m_hAcce1, pMsg)) {
				UpdateData(FALSE);
				return(TRUE);
			}
		}
	}
	return CDialog::PreTranslateMessage(pMsg);
}

void CSerialDlg::OnDeltaposSpin1(NMHDR *pNMHDR, LRESULT *pResult)
{
	LPNMUPDOWN pNMUpDown = reinterpret_cast<LPNMUPDOWN>(pNMHDR);
	// TODO: Add your control notification handler code here
	knob1 = m_Slider1.GetPos();
	if (pNMUpDown->iDelta == -1 && knob1 > 0) {
		m_Slider1.SetPos(--knob1);
	}
	else if (pNMUpDown->iDelta == 1 && knob1 < 4095) {
		m_Slider1.SetPos(++knob1);
	}
	m_strKnob1.Format(_T("%d"), knob1);
	UpdateData(false);
	*pResult = 0;
}


void CSerialDlg::OnNMCustomdrawSlider1(NMHDR *pNMHDR, LRESULT *pResult)
{
	LPNMCUSTOMDRAW pNMCD = reinterpret_cast<LPNMCUSTOMDRAW>(pNMHDR);
	// TODO: Add your control notification handler code here
	knob1 = m_Slider1.GetPos();
	m_strKnob1.Format(_T("%d"), knob1);
	UpdateData(false);

	*pResult = 0;
}


void CSerialDlg::OnDeltaposSpin2(NMHDR *pNMHDR, LRESULT *pResult)
{
	LPNMUPDOWN pNMUpDown = reinterpret_cast<LPNMUPDOWN>(pNMHDR);
	// TODO: Add your control notification handler code here
	knob2 = m_Slider2.GetPos();
	if (pNMUpDown->iDelta == -1 && knob2 > 0) {
		m_Slider2.SetPos(--knob2);
	}
	else if (pNMUpDown->iDelta == 1 && knob2 < 4095) {
		m_Slider2.SetPos(++knob2);
	}
	m_strKnob2.Format(_T("%d"), knob2);
	UpdateData(false);
	*pResult = 0;
}


void CSerialDlg::OnDeltaposSpin3(NMHDR *pNMHDR, LRESULT *pResult)
{
	LPNMUPDOWN pNMUpDown = reinterpret_cast<LPNMUPDOWN>(pNMHDR);
	// TODO: Add your control notification handler code here
	knob3 = m_Slider3.GetPos();
	if (pNMUpDown->iDelta == -1 && knob3 > 0) {
		m_Slider3.SetPos(--knob3);
	}
	else if (pNMUpDown->iDelta == 1 && knob3 < 4095) {
		m_Slider3.SetPos(++knob1);
	}
	m_strKnob3.Format(_T("%d"), knob3);
	UpdateData(false);
	*pResult = 0;
}


void CSerialDlg::OnNMCustomdrawSlider2(NMHDR *pNMHDR, LRESULT *pResult)
{
	LPNMCUSTOMDRAW pNMCD = reinterpret_cast<LPNMCUSTOMDRAW>(pNMHDR);
	// TODO: Add your control notification handler code here
	knob2 = m_Slider2.GetPos();
	m_strKnob2.Format(_T("%d"), knob2);
	UpdateData(false);
	*pResult = 0;
}


void CSerialDlg::OnNMCustomdrawSlider4(NMHDR *pNMHDR, LRESULT *pResult)
{
	LPNMCUSTOMDRAW pNMCD = reinterpret_cast<LPNMCUSTOMDRAW>(pNMHDR);
	// TODO: Add your control notification handler code here
	knob3 = m_Slider3.GetPos();
	m_strKnob3.Format(_T("%d"), knob3);
```

```
	UpdateData(false);
	*pResult = 0;
}


void CSerialDlg::OnTimer(UINT_PTR nIDEvent)
{
	// TODO: Add your message handler code here and/or call default
	switch (nIDEvent) {
		//case 1:
		//	OnBnClickedButton5();
		//	CDialogEx::OnTimer(nIDEvent);//???
		//	break;
	case 2:
		OutputMsg[1] = modeCmd;
		OutputMsg[2] = (BYTE)((knob1 & 0xFF0) >> 4);
		OutputMsg[3] = (BYTE)((knob1 & 0xF) << 4 | (knob2 & 0xF00) >> 8);
		OutputMsg[4] = (BYTE)(knob2 & 0xFF);
		OutputMsg[5] = (BYTE)((knob3 & 0xFF0) >> 4);
		OutputMsg[6] = (BYTE)((knob3 & 0xF) << 4);

		m_mycom.put_Output(COleVariant(OutputMsg));
		CDialogEx::OnTimer(nIDEvent);
		break;
	case 3:
		CChartLineSerie *pLineSerie;

		m_ChartCtrl1.EnableRefresh(false);
		m_ChartCtrl1.RemoveAllSeries();//clear
		pLineSerie = m_ChartCtrl1.CreateLineSerie();
		pLineSerie->SetColor(RGB(255, 0, 0));
		pLineSerie->SetSeriesOrdering(poNoOrdering);//set to disorder
		pLineSerie->AddPoints(x, y1, 10000);
		pLineSerie->SetName(_T("Force Sensor"));
		m_ChartCtrl1.EnableRefresh(true);

		m_ChartCtrl2.EnableRefresh(false);
		m_ChartCtrl2.RemoveAllSeries();//clear
		pLineSerie = m_ChartCtrl2.CreateLineSerie();
		pLineSerie->SetColor(RGB(0, 0, 255));
		pLineSerie->SetSeriesOrdering(poNoOrdering);
		pLineSerie->AddPoints(x, y2, 10000);
		pLineSerie->SetName(_T("Optical Displacement"));
		m_ChartCtrl2.EnableRefresh(true);

		m_ChartCtrl3.EnableRefresh(false);
		m_ChartCtrl3.RemoveAllSeries();//clear
		pLineSerie = m_ChartCtrl3.CreateLineSerie();
		pLineSerie->SetColor(RGB(0, 255, 0));
		pLineSerie->SetSeriesOrdering(poNoOrdering);
		pLineSerie->AddPoints(x, y3, 10000);
		pLineSerie->SetName(_T("Speaker Output"));
		m_ChartCtrl3.EnableRefresh(true);
	}
}
```

BIBLIOGRAPHY

[1] I. Rutkow, "History of Surgery," in *Sabiston Textbook of Surgery: The Biological Basis of Modern Surgical Practice*, 19th ed., C. M. Townsend, R. D. Beauchamp, B. M. Evers, and K. L. Mattox, Eds. Philadelphia: Elsevier Saunders, 2012, pp. 2–18.

[2] L. D. Cameron and L. Jago, "Cognitive Mediators," *Encyclopedia of Behavioral Medicine*, vol. 451, no. 1700. Springer, New York, 2013.

[3] H. Heuer and A. M. Wing, "Doing Two Things at Once: Process Limitations and Interactions," in *The Psychology of Human Movement*, Academic Press, 1984.

[4] J. M. Loomis, R. L. Klatzky, and N. A. Guidice, "Sensory substitution of vision : Importance of perceptual and cognitive processing," in *Assistive Technology for Blindness and Low Vision*, R. Manduchi and S. Kurniawan, Eds. Boca Raton, FL: CRC Press, 2012.

[5] C. Richards, J. Rosen, B. Hannaford, C. Pellegrini, and M. Sinanan, "Skills evaluation in minimally invasive surgery using force / torque signatures," *Surg. Endosc.*, vol. 14, pp. 791–798, 2000.

[6] S. L. Cremers, J. B. Ciolino, Z. K. Ferrufino-Ponce, and B. A. Henderson, "Objective Assessment of Skills in Intraocular Surgery (OASIS)," *Ophthalmology*, vol. 112, no. 7, pp. 1236–1241, Jul. 2005.

[7] S. L. Cremers, A. N. Lora, and Z. K. Ferrufino-Ponce, "Global Rating Assessment of Skills in Intraocular Surgery (GRASIS)," *Ophthalmology*, vol. 112, no. 10, pp. 1655–1660, Oct. 2005.

[8] D. G. Ezra, R. Aggarwal, M. Michaelides, N. Okhravi, S. Verma, L. Benjamin, P. Bloom, A. Darzi, and P. Sullivan, "Skills Acquisition and Assessment after a Microsurgical Skills Course for Ophthalmology Residents," *Ophthalmology*, vol. 116, no. 2, pp. 257–262, 2009.

[9] J. Rosen, M. Sinanan, and B. Hannaford, "Objective Assessment of Surgical Skills," in *Surgical Robotics: Systems Applications and Visions*, J. Rosen, B. Hannaford, and R. M. Satava, Eds. Boston, MA: Springer, 2011, pp. 619–649.

[10] MedlinePlus, "Laparoscopic surgery," 2009. [Online]. Available: https://www.nlm.nih.gov/medlineplus/ency/presentations/100166_1.htm.

[11] N. J. Soper, L. M. Brunt, and K. Kerbl, "Laparoscopic General Surgery," *N. Engl. J. Med.*, vol. 330, no. 6, pp. 409–419, 1994.

[12] L. Boni, A. Benevento, F. Rovera, G. Dionigi, M. D. I. Giuseppe, C. Bertoglio, and R. Dionigi, "Infective Complications in Laparoscopic Surgery," *Surg. Infect. (Larchmt).*, vol. 7, no. Suppl 2, pp. 109–112, 2006.

[13] B. Jonsson and N. Zethraeus, "Costs and Benefits of Laparoscopic Surgery -- a Review of the Literature," *Eur. J. Surg.*, vol. 166, no. Suppl 585, pp. 48–56, 2000.

[14] U. Berggren, T. Gordh, D. Grama, U. Haglund, J. Rastad, and D. Arvidsson, "Laparoscopic versus open cholecystectomy: hospitalization, sick leave, analgesia and trauma responses," *Br. J. Surg.*, vol. 81, pp. 1362–1365, 1994.

[15] O. A. J. van der Meijden and M. P. Schijven, "The value of haptic feedback in conventional and robot-assisted minimal invasive surgery and virtual reality training: a current review," *Surg. Endosc.*, vol. 23, pp. 1180–1190, 2009.

[16] E. P. Westebring-Van Der Putten, J. J. Van Den Dobbelsteen, R. H. M. Goossens, J. J. Jakimowicz, and J. Dankelman, "Effect of laparoscopic grasper force transmission ratio on grasp control," *Surg. Endosc. Other Interv. Tech.*, vol. 23, no. 4, pp. 818–824, 2009.

[17] E. P. Westebring-van der Putten, J. J. van den Dobbelsteen, R. H. M. Goossens, J. J. Jakimowicz, and J. Dankelman, "Force feedback requirements for efficient laparoscopic grasp control," *Ergonomics*, vol. 52, no. 9, pp. 1055–1066, 2009.

[18] L. H. Yap and C. E. Butler, "Principles of Microsurgery," in *Grabb and Smith's Plastic Surgery*, 6th ed., C. H. Thorne, Ed. Philadelphia: Lippincott Williams & Wilkins, 2007, pp. 66–72.

[19] A. Portincasa, G. Gozzo, D. Parisi, L. Annacontini, A. Campanale, and A. Maiorella, "Microsurgical Treatment of Injury to Peripheral nerves in Upper and Lower Limbs: A Critical Review of the Last 8 Years," *Microsurgery*, vol. 27, no. February, pp. 455–462, 2007.

[20] A. D. Jagtap and C. N. Riviere, "Applied Force during Vitreoretinal Microsurgery with Handheld Instruments," in *Proc 26th Annual International Conference IEEE EMBS*, 2004, pp. 2771–2773.

[21] L. A. Jones, "Kinesthetic Sensing," in *Human and Machine Haptics*, M. R. Cutkosky, R. D. Howe, K. Salisbury, and M. A. Srinivasan, Eds. MIT Press, 2000.

[22] M. Bohan, D. S. McConnell, A. Chaparro, and S. G. Thompson, "The effects of visual magnification and physical movement scale on the manipulation of a tool with indirect vision.," *J. Exp. Psychol. Appl.*, vol. 16, no. 1, pp. 33–44, 2010.

[23] MedlinePlus, "Anastomosis," 2014. [Online]. Available: https://www.nlm.nih.gov/medlineplus/ency/article/002231.htm.

[24] M. S. Alghoul, C. R. Gordon, R. Yetman, G. M. Buncke, M. Siemionow, A. M. Afifi, and W. K. Moon, "From simple interrupted to complex spiral: A systematic review of various suture techniques for microvascular anastomoses," *Microsurgery*, vol. 31, no. 1, pp. 72–80, Jan. 2011.

[25] R. C. Danczyk, J. Coleman, J. Allensworth, A. F. Azarbal, E. L. Mitchell, T. K. Liem, G. J. Landry, and G. L. Moneta, "Incidence and outcomes of intraoperative vascular surgery consultations," *J. Vasc. Surg.*, vol. 62, no. 1, pp. 177–182, 2014.

[26] H. Haimovici, “Exposure of the Carotid Artery,” in *Haimovici’s Vascular Surgery*, 6th ed., E. Ascher, F. J. Veith, and P. Gloviczki, Eds. West Sussex, UK: Wiley & Sons, 2012, pp. 317–319.

[27] MedlinePlus, “Abdominal aortic aneurysm,” 2013. [Online]. Available: https://www.nlm.nih.gov/medlineplus/ency/article/000162.htm.

[28] M. C. Tracci and K. J. Cherry, “The Aorta,” in *Sabiston Textbook of Surgery: The Biological Basis of Modern Surgical Practice*, 19th ed., C. M. Townsend, R. D. Beauchamp, B. M. Evers, and K. L. Mattox, Eds. Philadelphia: Elsevier Saunders, 2012, pp. 1697–1724.

[29] R. M. Fairman and G. J. Wang, “Abdominal Aortic Aneurysms,” in *Rutherford’s Vascular Surgery*, 8th ed., J. L. Cronenwett and K. W. Johnston, Eds. Philadelphia: Elsevier, 2014, pp. 2046–2061.

[30] P. J. E. Holt and M. M. Thompson, “Abdominal Aortic Aneurysms: Evaluation and Decision Making,” in *Rutherford’s Vascular Surgery*, 8th ed., J. L. Cronenwett and K. W. Johnston, Eds. Philadelphia, 2014, pp. 1999–2023.

[31] J. L. De Bruin, A. F. Baas, J. Buth, M. Prinssen, E. L. G. Verhoeven, P. W. M. Cuypers, M. R. H. M. van Sambeek, R. Balm, D. E. Grobbee, and J. D. Blankensteijn, “Long-term outcome of open or endovascular repair of abdominal aortic aneurysm.,” *N. Engl. J. Med.*, vol. 362, no. 20, pp. 1881–1889, 2010.

[32] R. Brahmbhatt, J. Gander, Y. Duwayri, and R. R. Rajani, “Improved trends in patient survival and decreased major complications after emergency ruptured abdominal aortic aneurysm repair,” *J. Vasc. Surg.*, vol. 63, no. 1, pp. 39–48, 2011.

[33] ECRI Institute, “Scleral and Corneal Burns during Phacoemulsification,” 1996. [Online]. Available: http://www.mdsr.ecri.org/summary/detail.aspx?doc_id=8219.

[34] National Eye Institute, “Macular Pucker,” 2012. [Online]. Available: https://nei.nih.gov/health/pucker/pucker.

[35] L. Wickham and Z. Gregor, “Epiretinal Membranes,” in *Retina*, 5th ed., S. J. Ryan, Ed. China: Elsevier Saunders, 2013, pp. 1954–1961.

[36] National Eye Institute, “Facts about Vitreous Detachment,” 2009. [Online]. Available: https://nei.nih.gov/health/vitreous/vitreous.

[37] P. K. Gupta, P. S. Jensen, and E. de Juan Jr., “Surgical Forces and Tactile Perception During Retinal Surgery,” *LNCS*, vol. 1679, pp. 1218–1225, 1999.

[38] American Academy of Ophthalmology, “Preferred Practice Pattern Guidelines: Cataract in the Adult Eye,” 2011. [Online]. Available: www.aao.org/ppp.

[39] J. D. Stein, D. S. Grossman, K. M. Mundy, A. Sugar, and F. A. Sloan, “Severe Adverse Events after Cataract Among Medicare Beneficiaries,” *Ophthalmology*, vol. 118, no. 9, pp. 1716–1723, 2011.

[40] National Eye Institute, “Facts About Cataract.” [Online]. Available: https://www.nei.nih.gov/health/cataract/cataract_facts.asp.

[41] The Eye Disease Prevalence Research Group, “Prevalence of Cataract and Pseudophakia/Aphakia Among Adults in the United States,” *Arch Ophthalmol*, vol. 122, pp. 487–494, 2004.

[42] H. E. Gollogly, D. O. Hodge, J. L. S. Sauver, and J. C. Erie, “Increasing incidence of cataract surgery : Population-based study,” *J. Cataract Refract. Surg.*, vol. 39, no. 9, pp. 1383–1389, 2013.

[43] H. V Gimbel, “Capsulorrhexis,” in *Achieving Excellence in Cataract Surgery: A Step-by-Step Approach*, D. M. Colvard, Ed. 2009, pp. 19–26.

[44] S. Arshinoff, “Mechanics of capsulorhexis,” *J. Cataract Refract. Surg.*, vol. 18, pp. 623–628, 1992.

[45] F. F. Marques, D. M. V Marques, R. H. Osher, and J. M. Osher, “Fate of anterior capsule tears during cataract surgery.,” *J. Cataract Refract. Surg.*, vol. 32, no. 10, pp. 1638–42, Oct. 2006.

[46] B. C. Little, J. H. Smith, and M. Packer, “Little capsulorhexis tear-out rescue,” *J. Cataract Refract. Surg.*, vol. 32, no. 9, pp. 1420–1422, 2006.

[47] R. P. Coelho, J. S. Paula, J. M. Rosatelli, and A. M. V. Messias, “Capsulorhexis rescue after peripheral radial tear-out: Quick-pull technique,” *J. Cataract Refract. Surg.*, vol. 38, no. 5, pp. 737–738, 2012.

[48] P. B. Greenberg, V. L. Tseng, W.-C. Wu, J. Liu, L. Jiang, C. K. Chen, I. U. Scott, and P. D. Friedmann, “Prevalence and predictors of ocular complications associated with cataract surgery in United States veterans.,” *Ophthalmology*, vol. 118, no. 3, pp. 507–514, 2011.

[49] R. B. Vajpayee, N. Sharma, T. Dada, V. Gupta, A. Kumar, and V. K. Dada, “Management of Posterior Capsule Tears,” *Surv. Opthalmology*, vol. 45, no. 6, pp. 473–488, 2001.

[50] A. M. Murray, R. L. Klatzky, and P. K. Khosla, “Psychophysical Characterization and Testbed Validation of a Wearable Vibrotactile Glove for Telemanipulation,” *Presence*, vol. 12, no. 2, pp. 156–182, 2003.

[51] E. O. Doebelin, *Measurement Systems: Application and Design*, 4th ed. New York: McGraw-Hill, 1990.

[52] J. Fraden, *Handbook of Modern Sensors*, 4th ed. New York: Springer, 2010.

[53] National Instruments, “How Is Temperature Affecting Your Strain Measurement Accuracy?,” 2016. [Online]. Available: http://www.ni.com/white-paper/3432/en/.

[54] W. Tong, *Mechanical Design of Electric Motors*. Boca Raton, FL: CRC Press, 2014.

[55] J. R. Brauer, *Magnetic Actuators and Sensors*, 2nd ed. Somerset, NJ: John Wiley & Sons, 2014.

[56] E. P. Gardner and K. O. Johnson, “Sensory Coding,” in *Principles of Neural Science*, 5th ed., E. R. Kandel, J. H. Schwartz, T. M. Jessell, S. A. Siegelbaum, and A. J. Hudspeth, Eds. New York: McGraw-Hill, 2013.

[57] L. A. Jones and H. Z. Tan, “Application of Psychophysical Techniques to Haptic Research,” *IEEE Trans. Haptics*, vol. 6, no. 3, pp. 268–284, 2012.

[58] J. M. Wolfe, K. R. Kluender, D. M. Levi, L. M. Bartoshuk, R. S. Herz, R. L. Klatzky, S. J. Lederman, and D. M. Merfeld, *Sensation and Perception*, 3rd ed. Sunderland, MA: Sinauer Associates, Inc, 2012.

[59] S. Hecht, S. Shlaer, and M. H. Pirenne, "Energy, Quanta, and Vision," *J. Gen. Physiol.*, vol. 25, no. 6, pp. 819–840, 1942.

[60] S. S. Stevens, *Psychophysics: Introduction to Its Perceptual, Neural, and Social Prospects*. New York: John Wiley, 1975.

[61] F. J. Clark and K. W. Horch, "Kinesthesia," in *Handbook of Perception and Human Performance, Vol I: Sensory Processes and Perception*, K. R. Boff, L. Kaufman, and J. P. Thomas, Eds. Toronto: Wiley & Sons, 1986.

[62] E. P. Gardner and K. O. Johnson, "The Somatosensory System: Receptors and Central Pathways," in *Principles of Neural Science*, 5th ed., E. R. Kandel, J. H. Schwartz, T. M. Jessell, S. A. Siegelbaum, and A. J. Hudspeth, Eds. New York: McGraw-Hill, 2013.

[63] S. Weinstein, "Intensive and extensive aspects of tactile sensitivity as a function of body part, sex, and laterality," in *The skin senses*, D. R. Kenshalo, Ed. Springfield, IL: Thomas, 1968, pp. 195–222.

[64] E. P. Gardner and K. O. Johnson, "Touch," in *Principles of Neural Science*, 5th ed., E. R. Kandel, J. H. Schwartz, T. M. Jessell, S. A. Siegelbaum, and A. J. Hudspeth, Eds. New York: McGraw-Hill, 2013.

[65] R. L. Klatzky and S. J. Lederman, "Touch," in *Handbook of Psychology*, vol. 4, A. F. Healy and R. W. Proctor, Eds. New York: Wiley, 2002, pp. 147–176.

[66] C. E. Sherrick and R. W. Cholewiak, "Cutaneous Sensitivity," in *Handbook of Perception and Human Performance, Vol I: Sensory Processes and Perception*, K. R. Boff, L. Kaufman, and J. P. Thomas, Eds. Toronto: Wiley & Sons, 1986.

[67] S. Hendry and S. Hsiao, "Somatosensory System," in *Fundamental Neuroscience*, 3rd ed., L. Squire, D. Berg, F. Bloom, S. du Lac, A. Ghosh, and N. Spitzer, Eds. Burlington, MA: Elsevier, 2008.

[68] U. Proske and S. C. Gandevia, "The kinaesthetic senses," *J. Physiol.*, vol. 587, no. 17, pp. 4139–4146, 2009.

[69] T. A. Woolsey, J. Hanaway, and M. H. Gado, *The Brain Atlas: A Visual Guide to the Human Central Nervous System*, 3rd ed. Hoboken, NJ: John Wiley & Sons, 2008.

[70] T. Harnden, "Central Nervous System." [Online]. Available: http://www2.highlands.edu/academics/divisions/scipe/biology/faculty/harnden/2121/notes/cns.htm.

[71] OpenStax, "Central Processing," *Anatomy & Physiology*, 2014. [Online]. Available: http://cnx.org/contents/FPtK1zmh@6.27:KcreJ7oj@5/Central-Processing.

[72] D. M. Wolpert, K. G. Pearson, and C. P. J. Ghez, "The Organization and Planning of Movement," in *Principles of Neural Science*, 5th ed., E. R. Kandel, J. H. Schwartz, T. M. Jessell, S. A. Siegelbaum, and A. J. Hudspeth, Eds. New York: McGraw-Hill, 2013.

[73] A. G. Witney, A. Wing, J. Thonnard, and A. M. Smith, "The cutaneous contribution to adaptive precision grip," *Trends Neurosci.*, vol. 27, no. 10, pp. 637–643, 2004.

[74] B. Wu, R. Klatzky, R. Lee, V. Shivaprabhu, J. Galeotti, M. Siegel, J. S. Schuman, R. Hollis, and G. Stetten, "Psychophysical Evaluation of Haptic Perception Under Augmentation by a Handheld Device," *Hum. Factors*, vol. 57, no. 3, pp. 523–537, May 2015.

[75] J. Dosher and B. Hannaford, "Human Interaction with Small Haptic Effects," *Presence Teleoperators Virtual Environ.*, vol. 14, no. 3, pp. 329–344, 2005.

[76] E. Mesa-Munera, J. F. Ramirez-Salazar, P. Boulanger, W. F. Bischof, and J. W. Branch, "Estimation of Vibration and Force Stimulus Thresholds for Haptic Guidance in MIS Training.pdf," *Rev. Ing. Biomed.*, vol. 5, no. 10, pp. 17–22, 2011.

[77] S. J. Lederman and R. L. Klatzky, "Haptic perception: A tutorial," *Attention, Perception, Psychophys.*, vol. 71, no. 7, pp. 1439–1459, 2009.

[78] B. Wu, R. L. Klatzky, and R. L. Hollis, "Force, Torque, and Stiffness: Interactions in Perceptual Discrimination," *IEEE Trans. Haptics*, vol. 4, no. 3, pp. 221–228, 2011.

[79] C. B. Novak, "Evaluation of Hand Sensibility : A Review," *J. Hand Ther.*, vol. 14, no. 4, pp. 266–272, 2001.

[80] K. Salisbury, F. Conti, and F. Barbagli, "Haptic Rendering: Introductory Concepts," *IEEE Computer Graphics and Applications*, no. April, pp. 24–32, 2004.

[81] A. M. Okamura, C. Richard, and M. R. Cutkosky, "Feeling is Believing: Using a Force-Feedback Joystick to Teach Dynamic Systems," *J. Eng. Educ.*, vol. 91, no. 3, pp. 345–350, 2002.

[82] R. Gassert, J. C. Metzger, K. Leuenberger, W. L. Popp, M. R. Tucker, B. Vigaru, R. Zimmermann, and O. Lambercy, "Physical student-robot interaction with the ETHZ haptic paddle," *IEEE Trans. Educ.*, vol. 56, no. 1, pp. 9–17, 2013.

[83] R. B. Gillespie, M. B. Hoffinan, and J. Freudenberg, "Haptic interface for hands-on instruction in system dynamics and embedded control," *Proc. - 11th Symp. Haptic Interfaces Virtual Environ. Teleoperator Syst. HAPTICS 2003*, pp. 410–415, 2003.

[84] M. Solazzi, A. Frisoli, and M. Bergamasco, "Design of a novel finger haptic interface for contact and orientation display," in *Proc IEEE Haptics Symposium*, 2010, pp. 129–132.

[85] F. Chinello, M. Malvezzi, C. Pacchierotti, and D. Prattichizzo, "A three DoFs wearable tactile display for exploration and manipulation of virtual objects," in *Proc IEEE Haptics Symposium*, 2012, pp. 71–76.

[86] J. Brown, M. Ibrahim, E. D. Z. Chase, C. Pacchierotti, and K. J. Kuchenbecker, "Data-Driven Comparison of Four Cutaneous Displays for Pinching Palpation in Robotic Surgery," *Proc IEEE Haptics Symp.*, 2016.

[87] P. B. McBeth, D. F. Louw, P. R. Rizun, and G. R. Sutherland, "Robotics in neurosurgery," *Am. J. Surg.*, vol. 188, no. 4A Suppl, p. 68S–75S, Oct. 2004.

[88] J. A. Smith, J. Jivraj, R. Wong, and V. Yang, "30 Years of Neurosurgical Robots: Review and Trends for Manipulators and Associated Navigational Systems," *Ann. Biomed. Eng.*, 2015.

[89] S. Dimaio, M. Hanuschik, and U. Kreaden, “The da Vinci Surgical System,” in *Surgical Robotics: Systems Applications and Visions*, J. Rosen, B. Hannaford, and R. M. Satava, Eds. Boston, MA: Springer, 2011, pp. 199–217.

[90] M. J. H. Lum, D. C. W. Friedman, G. Sankaranarayanan, H. King, K. Fodero, R. Leuschke, B. Hannaford, J. Rosen, and M. N. Sinanan, “The RAVEN: Design and Validation of a Telesurgery System,” *Int. J. Rob. Res.*, vol. 28, no. 9, pp. 1183–1197, 2009.

[91] J. Rosen, B. Hannaford, and R. M. Satava, “Raven: Developing a Surgical Robot from a Concept to a Transatlantic Teleoperation Experiment,” in *Surgical Robotics: Systems Applications and Visions*, J. Rosen, B. Hannaford, and R. M. Satava, Eds. Boston, MA: Springer, 2011, pp. 159–197.

[92] B. Hannaford, J. Rosen, D. W. Friedman, H. King, P. Roan, L. Cheng, D. Glozman, J. Ma, S. N. Kosari, and L. White, “Raven-II: An open platform for surgical robotics research,” *IEEE Trans. Biomed. Eng.*, vol. 60, no. 4, pp. 954–959, 2013.

[93] A. Murali, S. Sen, B. Kehoe, A. Garg, S. McFarland, S. Patil, W. D. Boyd, S. Lim, P. Abbeel, and K. Goldberg, “Learning by observation for surgical subtasks: Multilateral cutting of 3D viscoelastic and 2D Orthotropic Tissue Phantoms,” *Proc IEEE Int. Conf. Robot. Autom.*, no. June, pp. 1202–1209, 2015.

[94] B. Kehoe, G. Kahn, J. Mahler, J. Kim, A. Lee, A. Lee, K. Nakagawa, S. Patil, W. D. Boyd, P. Abbeel, and K. Goldberg, “Autonomous multilateral debridement with the Raven surgical robot,” *Proc. - IEEE Int. Conf. Robot. Autom.*, pp. 1432–1439, 2014.

[95] N. Diolaiti, G. Niemeyer, F. Barbagli, and J. K. Salisbury, “Stability of haptic rendering: Discretization, quantization, time delay, and Coulomb effects,” *IEEE Trans. Robot.*, vol. 22, no. 2, pp. 256–268, 2006.

[96] C. R. Wagner, N. Stylopoulos, and R. D. Howe, “The Role Of Force Feedback In Surgery : Analysis Of Blunt Dissection,” 2002.

[97] C. E. Reiley, T. Akinbiyi, D. Burschka, D. C. Chang, A. M. Okamura, and D. D. Yuh, “Effects of visual force feedback on robot-assisted surgical task performance,” *J. Thorac. Cardiovasc. Surg.*, vol. 135, no. 1, pp. 196–202, 2008.

[98] Z. F. Quek, S. B. Schorr, I. Nisky, W. R. Provancher, and A. M. Okamura, “Sensory Substitution of Force and Torque using 6-DoF Tangential and Normal Skin Deformation Feedback,” 2015.

[99] D. Prattichizzo, C. Pacchierotti, and G. Rosati, “Cutaneous Force Feedback as a Sensory Subtraction Technique in Haptics,” *IEEE Trans. Haptics*, vol. 5, no. 4, pp. 289–300, 2012.

[100] L. Meli, C. Pacchierotti, and D. Prattichizzo, “Sensory Subtraction in Robot-Assisted Surgery : Fingertip Skin Deformation Feedback to Ensure Safety and Improve Transparency in Bimanual Haptic Interaction,” *IEEE Trans. Biomed. Eng.*, vol. 61, no. 4, pp. 1318–1327, 2014.

[101] C. Pacchierotti, D. Prattichizzo, and K. J. Kuchenbecker, “Cutaneous Feedback of Fingertip Deformation and Vibration for Palpation in Robotic Surgery,” *IEEE Trans. Biomed. Eng.*, vol. 63, no. 2, pp. 278–287, 2016.

[102] R. A. MacLachlan, B. C. Becker, J. Cuevas Tabares, G. W. Podnar, L. A. Lobes Jr., and C. N. Riviere, "Micron: an Actively Stabilized Handheld Tool for Microsurgery," *IEEE Trans. Robot.*, vol. 28, no. 1, pp. 195–212, 2012.

[103] B. C. Becker, S. Voros, R. A. Maclachlan, G. D. Hager, and C. N. Riviere, "Visual Guidance of an Active Handheld Microsurgical Tool," in *Surgical Robotics: Systems Applications and Visions*, J. Rosen, B. Hannaford, and R. M. Satava, Eds. Boston, MA: Springer, 2011, pp. 339–344.

[104] C. N. Riviere and P. S. Jensen, "A study of instrument motion in retinal microsurgery," *Proc 22nd Annu. Int. Conf. IEEE Eng. Med. Biol. Soc.*, vol. 1, pp. 59–60, 2000.

[105] B. C. Becker, S. Voros, L. A. Lobes, J. T. Handa, G. D. Hager, and C. N. Riviere, "Retinal Vessel Cannulation with an Image-Guided Handheld Robot," in *Proc of IEEE Engineering in Medicine and Biology Conference*, 2010, pp. 5420–5423.

[106] B. C. Becker, R. A. MacLachlan, L. A. Lobes, and C. N. Riviere, "Position-based retinal membrane peeling with a handheld robot," in *2012 IEEE International Conference on Robotics and Automation*, 2012, pp. 1075–1080.

[107] D. Y. Choi, R. Sandoval, R. A. MacLachlan, L. Ho, L. A. Lobes Jr., and C. N. Riviere, "Test of Tracing Performance with an Active Handheld Micromanipulator," in *Proc 29th Annual Int'l Conf IEEE EMBS*, 2007, pp. 3638–3641.

[108] S. Yang, "Handheld Micromanipulator for Robot- Assisted Microsurgery," Carnegie Mellon University, 2015.

[109] H. Yao, V. Hayward, and R. E. Ellis, "A Tactile Enhancement Instrument for Minimally Invasive Surgery," *Comput. Aided Surg.*, vol. 10, no. 4, pp. 233–239, 2004.

[110] H. Yao, "Touch Magnifying Instrument Applied to Minimally Invasive Surgery," McGill University, 2004.

[111] M. Nambi, W. R. Provancher, and J. J. Abbott, "On the Ability of Humans to Apply Controlled Forces to Admittance-Type Devices," *Adv. Robot.*, vol. 25, pp. 629–650, 2011.

[112] R. Taylor, P. Jensen, L. Whitcomb, A. Barnes, R. Kumar, D. Stoianovici, P. Gupta, Z. Wang, E. de Juan, and L. Kavoussi, "A Steady-Hand Robotic System for Microsurgical Augmentation," *Int. J. Rob. Res.*, vol. 18, no. 12, pp. 1201–1210, 1999.

[113] B. Mitchell, J. Koo, I. Iordachita, P. Kazanzides, A. Kapoor, J. Handa, R. H. Taylor, and G. Hager, "Development and Application of a New Steady-Hand Manipulator for Retinal Surgery," in *Proc IEEE International Conference on Robotics and Automation*, 2007, pp. 623–629.

[114] I. Fleming, M. Balicki, J. Koo, I. Iordachita, B. Mitchell, J. Handa, G. Hager, and R. Taylor, "Cooperative Robot Assistant for Retinal Microsurgery," in *Proc 11th International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, 2008, pp. 543–550.

[115] T. Leng, J. M. Miller, K. V Bilbao, D. V Palanker, P. Huie, and M. S. Blumenkranz, "The chick chorioallantoic membrane as a model tissue for surgical retinal research and simulation," *Retina*, vol. 24, no. 3, pp. 427–434, 2004.

[116] M. Balicki, A. Uneri, I. Iordachita, J. Handa, P. Gehlbach, and R. Taylor, “Micro-force Sensing in Robot Assisted Membrane Peeling for Vitreoretinal Surgery,” in *LNCS*, 2010, vol. 6363, no. 3, pp. 303–310.

[117] A. Uneri, M. Balicki, J. Handa, P. Gehlbach, and R. H. Taylor, “New Steady-Hand Robot with Micro-Force Sensing for Vitreoretinal Surgery,” in *Proc IEEE RAS EMBS Int Conf Biomed Robot Biomechatron*, 2011, pp. 814–819.

[118] P. J. Berkelman, L. L. Whitcomb, R. H. Taylor, and P. Jensen, “A Miniature Microsurgical Instrument Tip Force Sensor for Enhanced Force Feedback During Robot-Assisted Manipulation,” *IEEE Trans. Robot. Autom.*, vol. 19, no. 5, pp. 917–922, 2003.

[119] I. Iordachita, Z. Sun, M. Balicki, J. U. Kang, S. J. Phee, J. Handa, P. Gehlbach, and R. Taylor, “A sub-millimetric, 0.25 mN resolution fully integrated fiber-optic force-sensing tool for retinal microsurgery,” *Int. J. Comput. Assist. Radiol. Surg.*, vol. 4, no. 4, pp. 383–390, Jun. 2009.

[120] X. He, P. Gehlbach, J. Handa, R. Taylor, and I. Iordachita, “Development of A Miniaturized 3-DOF Force Sensing Instrument for Robotically Assisted Retinal Microsurgery and Preliminary Results,” 2014.

[121] A. M. Okamura, L. N. Verner, T. Yamamoto, J. C. Gwilliam, and P. G. Griffiths, “Force Feedback and Sensory Substitution for Robot-Assisted Surgery,” in *Surgical Robotics: Systems Applications and Visions*, J. Rosen, B. Hannaford, and R. M. Satava, Eds. Boston, MA: Springer, 2011, pp. 419–448.

[122] V. Nitsch and B. Farber, “A Meta-Analysis of the Effects of Haptic Interfaces on Task Performance with Teleoperation Systems,” *IEEE Trans. Haptics*, vol. 6, no. 4, pp. 387–398, 2013.

[123] M. O. Culjat, C. King, M. L. Franco, C. E. Lewis, J. W. Bisley, E. P. Dutson, and W. S. Grundfest, “A Tactile Feedback System for Robotic Surgery,” in *30th Int’l IEEE EMBS Conference*, 2008, pp. 1930–1934.

[124] C. King, M. O. Culjat, M. L. Franco, J. W. Bisley, G. P. Carman, E. P. Dutson, and W. S. Grundfest, “A Multielement Tactile Feedback System for Robot-Assisted Minimally Invasive Surgery,” *IEEE Trans. Haptics*, vol. 2, no. 1, pp. 52–56, 2009.

[125] M. O. Culjat, J. W. Bisley, C.-H. King, C. Wottawa, R. E. Fan, E. P. Dutson, and W. S. Grundfest, “Tactile Feedback in Surgical Robotics,” in *Surgical Robotics: Systems Applications and Visions*, J. Rosen, B. Hannaford, and R. M. Satava, Eds. Boston, MA: Springer, 2011, pp. 449–468.

[126] C. King, M. O. Culjat, M. L. Franco, J. W. Bisley, E. Dutson, and W. S. Grundfest, “Optimization of a Pneumatic Balloon Tactile Display for Robot-Assisted Surgery Based on Human Perception,” *IEEE Trans. Biomed. Eng.*, vol. 55, no. 11, pp. 2593–2600, 2008.

[127] C. J. Payne, W. T. Latt, and G.-Z. Yang, “A New Hand-Held Force-Amplifying Device for Micromanipulation,” in *IEEE International Conference on Robotics and Automation*, 2012, pp. 1583–1588.

[128] C. J. Payne, K. Kwok, and G. Yang, “An Ungrounded Hand-Held Surgical Device Incorporating Active Constraints with Force-Feedback,” in *IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2013, pp. 2559–2565.

[129] C. J. Payne, H. J. Marcus, and G.-Z. Yang, “A Smart Haptic Hand-Held Device for Neurosurgical Microdissection,” *Ann. Biomed. Eng.*, vol. 43, no. 9, pp. 2185–2195, 2015.

[130] G. Stetten, B. Wu, R. Klatzky, J. Galeotti, M. Siegel, R. Lee, F. Mah, A. Eller, J. Schuman, and R. Hollis, “Hand-Held Force Magnifier for Surgical Instruments,” *LNCS*, vol. 6689, pp. 90–100, 2011.

[131] J. M. Loomis, “Distal Attribution and Presence,” *Presence Teleoperators Virtual Environ.*, vol. 1, no. 1, pp. 113–119, 1991.

[132] J. H. Siegle and W. H. Warren, “Distal attribution and distance perception in sensory substitution,” *Perception*, vol. 39, no. 2, pp. 208–223, 2010.

[133] R. Lee, “The Hand-Held Force Magnifier for Surgical Instruments,” Carnegie Mellon University, 2012.

[134] P. Berkelman and R. L. Hollis, “Dynamic Performance of a Magnetic Levitation Haptic Device,” in *SPIE Proceedings*, 1997, no. 3206.

[135] C. Kaernbach, “Adaptive threshold estimation with unforced-choice tasks,” *Percept. Psychophys.*, vol. 63, no. 8, pp. 1377–1388, 2001.

[136] R. Lee, B. Wu, R. Klatzky, V. Shivaprabhu, J. Galeotti, S. Horvath, M. Siegel, J. S. Schuman, R. Hollis, and G. Stetten, “Hand-Held Force Magnifier for Surgical Instruments: Evolution toward a Clinical Device,” *LNCS*, vol. 7815, pp. 77–89, 2013.

[137] H. Z. Tan, M. A. Srinivasan, B. Eberman, and B. Cheng, “Human Factors for the Design of Force-Reflecting Haptic Interfaces,” in *Dynamic Systems and Control*, vol. 55–1, C. J. Radcliffe, Ed. The American Society of Mechanical Engineers, 1994.

[138] R. L. Klatzky, P. Gershon, V. Shivaprabhu, R. Lee, B. Wu, G. Stetten, and R. H. Swendsen, “A model of motor performance during surface penetration: from physics to voluntary control,” *Exp. Brain Res.*, vol. 230, no. 2, pp. 251–260, 2013.

[139] Y.-C. Fung, *Biomechanics: Mechanical Properties of Living Tissues*, 2nd ed. New York: Springer, 1993.

[140] Y.-C. Fung, *A First Course in Continuum Mechanics*, 3rd ed. Englewood Cliffs, N.J.: Prentice Hall, 1993.

[141] A. Nambu, H. Tokuno, and M. Takada, “Functional significance of the cortico-subthalamo-pallidal ‘hyperdirect’ pathway,” *Neurosci. Res.*, vol. 43, pp. 111–117, 2002.

[142] A. R. Aron and R. A. Poldrack, “Cortical and subcortical contributions to stop signal response inhibition: role of the subthalamic nucleaus,” *J. Neurosci.*, vol. 26, pp. 2424–2433, 2006.

[143] C. D. Chambers, M. A. Bellgrove, M. G. Stokes, T. R. Henderson, H. Garavan, I. H. Robertson, A. P. Morris, and J. B. Mattingley, “Executive ‘brake failure’ following deactivation of human frontal lobe,” *J. Cogn. Neurosci.*, vol. 18, pp. 444–455, 2006.

[144] F. X. Neubert, R. B. Mars, E. R. Buch, E. Olivier, and M. F. Rushworth, “Cortical and subcortical interactions during action reprogramming and their related white matter pathways,” *Proc. Natl. Acad. Sci.*, vol. 107, pp. 13240–13245, 2010.

[145] B. B. Zandbelt, M. Bloemendaal, J. M. Hoogendam, R. S. Kahn, and M. Vink, "Transcranial magnetic stimulation and functional MRI reveal cortical and subcortical interactions during stop-signal response inhibition," *J. Cogn. Neurosci.*, vol. 25, no. 157–174, 2013.

[146] P. Gershon, R. L. Klatzky, and R. Lee, "Handedness in a virtual haptic environment: Assessments from kinematic behavior and modeling," *Acta Psychol. (Amst).*, vol. 155, pp. 37–42, 2015.

[147] J. A. Bernard, S. F. Taylor, and R. D. Seidler, "Handedness, dexterity, and motor cortical representations," *J. Neurophysiol.*, vol. 105, no. 1, pp. 88–99, 2011.

[148] G. Hammond, "Correlates of human handedness in primary motor cortex: A review and hypothesis," *Neurosci. Biobehav. Rev.*, vol. 26, no. 3, pp. 285–292, 2002.

[149] P. K. Mutha, K. Y. Haaland, and R. L. Sainburg, "Rethinking motor lateralization: Specialized but complementary mechanisms for motor control of each arm," *PLoS One*, vol. 8, no. 3, p. e58582, 2013.

[150] R. L. Sainburg, "Evidence for a dynamic-dominance hypothesis of handedness," *Exp. Brain Res.*, vol. 142, no. 2, pp. 241–258, 2002.

[151] R. L. Sainburg, "Lateralization of goal-directed movement," in *Vision and goal-directed movement: Neurobehavioral perspectives*, D. Elliott and M. Khan, Eds. Champaign, IL, 2010, pp. 219–238.

[152] D. J. Serrien, R. B. Ivry, and S. P. Swinnen, "Dynamics of hemispheric specialization and integration in the context of motor control," *Nat. Rev. Neurosci.*, vol. 7, no. 2, pp. 160–166, 2006.

[153] C. J. Winstein and P. S. Pohl, "Effects of unilateral brain damage on the control of goal-directed hand movements," *Exp. Brain Res.*, vol. 105, no. 1, pp. 163–174, 1995.

[154] R. C. Oldfield, "The assessment and analysis of handedness: The Edinburgh inventory," *Neuropsychologia*, vol. 9, no. 1, pp. 97–113, 1971.

[155] S. M. Ross, *Introduction to Probability and Statistics for Engineers and Scientists*, 4th ed. Burlington, MA: Elsevier Academic Press, 2009.

[156] "Kolmogorov-Smirnov test," *Wikipedia*. [Online]. Available: https://en.wikipedia.org/wiki/Kolmogorov%25E2%2580%2593Smirnov_test.

[157] L. Neumyer and N. Ghalyaie, "Principles of Preoperative and Operative Surgery," in *Sabiston Textbook of Surgery: The Biological Basis of Modern Surgical Practice2*, 20th ed., C. M. Townsend, R. D. Beauchamp, B. M. Evers, and K. L. Mattox, Eds. Philadelphia: Elsevier, 2017, pp. 201–240.

[158] Aberdeen Endometrial Ablation Trials Group, "A randomised trial of endometrial ablation versus hysterectomy for the treatment of dysfunctional uterine bleeding: outcome at four years," *Br. J. Obstet. Gynaecol.*, vol. 106, pp. 360–366, 1999.

[159] M. A. Srinivasan and J. Chen, "Human Performance in Controlling Normal Forces of Contact with Rigid Objects," *Adv. Robot. Mechatronics, Haptic Interfaces*, vol. 49, pp. 119–125, 1993.

[160] D. Wang, J. Jiao, G. Yang, and Y. Zhang, “Force Maintenance Accuracy Using a Tool: Effects of Magnitude and Feedback,” *IEEE Trans. Haptics*, vol. 9, no. 3, pp. 432–436, 2016.

[161] A. M. Taylor, E. A. Christou, and R. M. Enoka, “Multiple Features of Motor-Unit Activity Influence Force Fluctuations During Isometric Contractions,” *J. Neurophysiol.*, vol. 90, pp. 1350–1361, 2003.

[162] A. F. de C. Hamilton, K. E. Jones, and D. M. Wolpert, “The scaling of motor noise with muscle strength and motor unit number in humans,” *Exp. Brain Res.*, vol. 157, pp. 417–430, 2004.

[163] J. Timmer, M. Lauk, and G. Deuschl, “Quantitative analysis of tremor time series,” *Electroencephalogr. Clin. Neurophysiol.*, vol. 101, pp. 461–468, 1996.

[164] B. Safwat, E. L. M. Su, R. Gassert, C. L. Teo, and E. Burdet, “The Role of Posture, Magnification, and Grip Force on Microscopic Accuracy,” *Ann. Biomed. Eng.*, vol. 37, no. 5, pp. 997–1006, 2009.

[165] N. J. Seo, T. J. Armstrong, and J. G. Young, “Effects of handle orientation, gloves, handle friction, and elbow posture on maximum horizontal pull and push forces,” *Ergonomics*, vol. 53, no. 1, pp. 92–101, 2010.

[166] J. Di Domizo and P. G. Keir, “Forearm posture and grip effects during push and pull tasks,” *Ergonomics*, vol. 53, no. 3, pp. 336–343, 2010.

[167] “Scalpel,” *Wikipedia*. [Online]. Available: https://en.wikipedia.org/wiki/Scalpel. [Accessed: 12-Dec-2016].

[168] A. Khera, R. Lee, A. Marcovici, Z. Yu, R. Klatzky, M. Siegel, S. Shroff, and G. Stetten, “One-Dimensional Haptic Rendering Using Audio Speaker with Displacement Determined by Inductance,” *Machines*, vol. 4, no. 1, p. 9, 2016.

[169] M. Dodd, W. Klippel, and J. Oclee-Brown, “Voice Coil Impedance as a Function of Frequency and Displacement,” *Audio Eng. Soc. Conv. 119*, pp. 1–17, 2004.

[170] K. J. Åström and R. M. Murray, *Feedback Systems: An Introduction for Scientists and Engineers*, 2.11b. Princeton, NJ: Princeton University Press, 2008.

[171] D. E. Seborg, T. F. Edgar, and D. A. Mellichamp, *Process Dynamics and Control*, 2nd ed. Hoboken, NJ: John Wiley & Sons, 2004.

[172] J. G. Ziegler and N. B. Nichols, “Optimum settings for Automatic Controllers,” *Trans. ASME*, pp. 759–768, 1942.

[173] B. C. Becker, R. A. MacLachlan, L. A. Lobes, and C. N. Riviere, “Vision-based retinal membrane peeling with a handheld robot,” in *2012 IEEE International Conference on Robotics and Automation*, 2012, pp. 1075–1080.

[174] H. Culbertson, J. Unwin, B. E. Goodman, and K. J. Kuchenbecker, “Generating Haptic Texture Models From Unconstrained Tool-Surface Interactions,” in *IEEE World Haptics Conference*, 2013, pp. 295–300.

[175] H. Culbertson, J. J. Lopez Delgado, and K. J. Kuchenbecker, “One Hundred Data-Driven Haptic Texture Models and Open-Source Methods for Rendering on 3D Objects,” in *IEEE Haptics Symposium*, 2014, pp. 319–325.