Electric Powered Wheelchair Control with a Variable Compliance Joystick: Improving Control of Mobility Devices for Individuals with Multiple Sclerosis

by

Karl Walter Brown

Bachelor of Science Aerospace Engineering Sciences, University of Colorado at Boulder, 2001

Submitted to the Graduate Faculty of

the School of Health and Rehabilitation Sciences in partial fulfillment

of the requirements for the degree of

Master of Science

University of Pittsburgh

2007

UNIVERSITY OF PITTSBURGH

SCHOOL OF HEALTH AND REHABILITATION SCIENCES

This thesis was presented

by

Karl Walter Brown

It was defended on

December 19, 2006

and approved by

Rory Cooper, Ph. D., Rehabilitation Science and Technology

Dan Ding, Ph. D., Rehabilitation Science and Technology

Thesis Director: Donald Spaeth, Ph. D., Rehabilitation Science and Technology

Electric Powered Wheelchair Control with a Variable Compliance Joystick: Improving Control of Mobility Devices for Individuals with Multiple Sclerosis

Karl Brown, M.S.

University of Pittsburgh, 2007

While technological developments over the past several decades have greatly enhanced the lives of people with mobility impairments, between 10 and 40 percent of clients who desired powered mobility found it very difficult to operate electric powered wheelchairs (EPWs) safely because of sensory impairments, poor motor function, or cognitive deficits [1]. The aim of this research is to improve control of personalized mobility for those with multiple sclerosis (MS) by examining isometric and movement joystick interfaces with customizable algorithms.

A variable compliance joystick (VCJ) with tuning software was designed and built to provide a single platform for isometric and movement, or compliant, interfaces with enhanced programming capabilities.

The VCJ with three different algorithms (basic, personalized, personalized with fatigue adaptation) was evaluated with four subjects with MS (mean age 58.7±5.0 yrs; years since diagnosis 28.2±16.1 yrs) in a virtual environment. A randomized, two-group, repeated-measures experimental design was used, where two subjects used the VCJ in isometric mode and two in compliant mode.

While still too early to draw conclusions about the performance of the joystick interfaces and algorithms, the VCJ was a functional platform for collecting information. Inspection of the data shows that the learning curve may be long for this system. Also, while subjects may have low trial times, low times could be related to more deviation from the target path.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

PREFACE

Numerous people have supported me and my research over these past few years. I would like to thank the Department of Veterans Affairs for supporting this research and the subjects who participated for your patience and feedback. I would like to thank my committee for your patience, encouragement, and advice; Dr. Songfeng Guo, for your encouragement and wisdom; Dr. Diane Collins (a.k.a. “Mama Di”) for your strong support; Jeremy Puhlman and Mark McCartney for putting up with me in the shop and providing your technical expertise; Steve Hayashi and Walter Sinegal for your electronics support and patience; Dr. Cameron Riviere for your support with the WFLC; Emily Teodorski and Annmarie Kelleher for helping with recruiting subjects and understanding of the mind of a slightly stressed engineer; Megan Yarnall, the Vicon Saint; Erica Authier and Harshal Mahajan for helping in times of need; and the folks at HERL and RST who have provided collegial support and friendship over the years. I would also like to thank my extended family in Pittsburgh: Uncle Jake and Aunt Vera and the Godin, Wehar, and Slack families. You guys have been an incredible blessing. Mom and Dad, you have been supportive all the way. And am... I’ll be home soon.

1.0 INTRODUCTION

Technological developments over the past several decades have greatly enhanced the lives of people with mobility impairments. Manual wheelchairs are becoming lighter and easier to propel, and power wheelchairs give individuals with limited mobility and upper body strength the ability to interact better with their environment. However, clinicians have reported that between 10 and 40 percent of their clients who desired powered mobility found it very difficult to operate electric powered wheelchairs (EPWs) safely because of sensory impairments, poor motor function, or cognitive deficits [1]. Individuals with movement disorders are particularly difficult to fit to wheelchairs because the device most often used to command a wheelchair is a hand-operated joystick. While innovative ways for individuals with movement disorders to control wheelchairs have been explored [2]-[6], there is still a lack of research in this area for individuals with multiple sclerosis (MS) [7]. Particular concerns for individuals with MS include tremor in the upper extremities and fatigue. Beginning with an overview of MS, mobility, and wheelchair user interfaces, this thesis will explore several novel methods to improve control of mobility for individuals with MS. Additionally, it will present initial results of a study in which individuals with MS used movement and isometric joystick user interfaces with personally-fitted input control algorithms to control a virtual EPW.

1.1 MULTIPLE SCLEROSIS AND THE NEED FOR PERSONAL MOBILITY

Approximately 211,000 people are living in the United States with MS [8]. MS is a progressive disease that attacks the central nervous system. As such, the disease shows itself in a variety of formats, including sensory, motor, and cognitive difficulties. Though highly variable among individuals, the progressive nature of MS means that conditions will worsen. Little is known

about the exact cause of the disease, and a cure does not exist. Current research trends are aimed at understanding the disease and finding ways to alleviate symptoms, delay its progression, and restore lost function. This chapter will examine some of the medical aspects of MS focusing on fatigue and tremor because of their relevance to wheelchair control; and, second, it will examine the need for personal mobility, its influence, and how this study addresses the specific needs of individuals with MS.

1.1.1 Medical Aspects of Multiple Sclerosis

1.1.1.1 Epidemiology

Multiple sclerosis is an acquired disease characterized by lesions in the central nervous system (CNS). Little is known about the exact cause of the disease, but its main feature is that the immune system attacks proteins on the myelin that surrounds neurons [7]. As a result, the ability for the axons to transmit electrical impulses deteriorates. Conduction becomes slow, blocked, or may be short circuited.

The etiology of MS involves a combination of environmental contacts and genetic susceptibility. Individuals living farther from the equator, such as in North America, southern Australia, and northern Europe, are more likely to have MS, suggesting environmental factors. But, higher risk rates for family members (20 to 40 times the risk of the general population) and identical twins (31% risk) of individuals with MS suggest genetic influences [9]. There is approximately twice the number of women with MS as men [9].

MS is generally progressive and can be classified into four types based on its progression. Eighty percent of individuals with MS have the *relapsing-remitting type* [9], where the person will undergo a relapse (also called an attack or exacerbation, with new neurological abnormalities) followed by periods of remissions or inactivity of the disease. The *primary progressive type* is defined by a slow progression of the disease without relapses. The *secondary progressive type* has a relapsing onset with progression between attacks. Lastly, the *progressive relapsing type*, occurring in less than one percent of those affected, begins with a progressive onset with a later relapse. Relapses are not to be confused with pseudoexacerbations, where symptoms may be worse but are not accompanied by new lesions in the CNS.

The prognosis for an individual diagnosed with MS can range from little impact to decreasing one's life expectancy to just a few months [9], with a majority anticipating a normal life expectancy [7]. Women and individuals with sensory or visual onset generally have a better outcome, while men, a later onset, and early, motor disability are generally predictive of a more severe clinical course [9],[10]. Because the disease attacks the CNS, a wide range of symptoms may occur.

Symptoms of MS vary between individuals, over days, and over hours [7]. Table 1 highlights some of its symptoms. Furthermore, individuals with MS may experience cognitive changes or depression [9]. Ataxia of the upper extremities is common in the later stages of MS [7].

1.1.1.2 Fatigue in Multiple Sclerosis

Fatigue is a problem for 75% to 90% of individuals with MS, where 50% to 60% report fatigue as being the worst symptom of the disease [11]. Fatigue is defined as "a state of reduced capacity for work following a period of mental or physical activity" [12] and has three primary components: motor (physical involvement), cognitive (mental involvement), and lassitude (perception of fatigue) [12].

Scales have been developed to measure fatigue; but each has unique properties and, as such, does not fully encompass fatigue in MS [13]. Performance-based measures do not correlate

Table 1: Common symptoms associated with MS.

Category	Symptoms
Sensory	Numbness and tingling, pain, a tight or constricting sensation, or Lhermittes phenomenon
Visual	Optic neuritis, nystagmus, or oscillopsia
Motor	Hemiparesis, paraparesis, spacticity, hyperreflexia, Babinski's sign, intention tremor, rubral tremor, or ataxia
Autonomic	Neurogenic bladder, neurogenic bowel, or sexual dysfunction
Brainstem	Dysarthria, dysphagia, vertigo, trigeminal neuralgia, or nystagmus

to self-report measures [14], and the definition of fatigue varies in each of the surveys, if defined at all [12]. Some of the most common surveys include the Fatigue Severity Scale (FSS), the Fatigue Assessment Instrument (FAI), and the Fatigue Impact Scale (FIS). The FSS emphasizes the physical involvement aspect of fatigue [13] and distinguishes between fatigue experienced between patients with an illness and healthy controls [12]. The FAI looks at the severity of the fatigue, consequences of fatigue, and the subject's responsiveness to rest [12]. And the FIS assesses fatigue's impact on the subject's life [12]. The FSS and FIS both have good test-retest reliability when there is little change in patients' fatigue status [15]. Schwid *et al*. [12] and Krupp [14], however, view surveys as inadequate measures of fatigue because of surveys' reliance on self-assessment and their retrospective nature, which confounds results because of recall bias. Schwid *et al*. [16] examined several performance-based measures of motor fatigue and found that their Fatigue Index (FI) based on the area between the maximal force level and the subject's force over the last 25 seconds provided the best test-retest reliability and was more sensitive to motor fatigue. Here, subjects pulled against a strap attached to an adjustable frame using knee extensors, elbow extensors, or ankle dorsiflexors while maintaining predefined limb positions and joint angles. Hand grip strength was assessed with a digital grip dynamometer. The FI is calculated as in equation **1**,

$$1: \qquad FI = 100 - \frac{\int_{5}^{30} F dt}{F_{\max}^{0-5} \cdot 25} \times 100\%,$$

where F is the subject's maximal voluntary isometric contraction force (MVIC) during the test and $F_{\max}^{0-5}$ is the maximum force applied in the first five seconds of the test. In order of test-retest reliability, the FI was calculated using MVIC force for hand grip, knee extensors, elbow extensors, and ankle dorsiflexor with interclass correlation coefficients (ICC) of 0.96, 0.83, 0.73, and 0.71, respectively. In the same study, Schwid *et al*. attempted to measure a dynamic motor fatigue – the dominant hand grip strength during a series of brief maximal contractions for 30 seconds – but found difficulties with this test as a measure of fatigue because some subjects found the frequency of contractions (1 Hz) to be too fast.

Fatigue has been shown to impact MS patients' mental health negatively because of its disruption on their lives [17]. Krupp and Elkins [18] found that MS patients' verbal memory, conceptual planning, and visual memory declined following tasks requiring continuous cognitive

effort. Between 48% and 60% of individuals with MS report fatigue as aggravating other MS symptoms [17],[19]. While fatigue is usually worsened with heat, the impact of fatigue seems to be independent of the patients' age, duration of MS, and Expanded Disability Status Score (EDSS) [17]. An individual's FI as calculated by Djaldetti *et al*. [20] – a slight variation of equation 1 – may or may not be worse during a relapse. They found if the pyramidal tract is involved, the FI will be worse; but no significant difference was found during a relapse if the pyramidal tract is not involved. Muscular weakness does not appear to be correlated to motor fatigue in patients with MS [16].

Current practices in treating fatigue involve both behavioral and pharmacological treatments. Behavioral treatments include education, support, exercise, rest periods, and avoidance of heat. And pharmacological treatments include regimens of amantadine, modafinil, or pemoline [14]. The coping mechanisms often used by individuals with MS to combat fatigue are pacing and reducing physical activity [19].

1.1.1.3 Tremor in Multiple Sclerosis

Tremor is defined as the involuntary oscillations of a body part and is another disabling symptom of MS. Approximately 75% of individuals with MS have some form of tremor [21]. About 10% of MS patients describe their tremor as "incapacitating" [22]. Tremor is located in the arms of approximately 56% of individuals with MS [22]. Specific types include

- Rest Tremor – Tremor in a body part that is not voluntarily active and is supported against gravity
- Postural Tremor – Tremor in a body part while maintaining a position against gravity
- Kinetic Tremor – Tremor in a body part during any voluntary movement
- Intention Tremor – Excluding postural tremor that may occur and the beginning or end of a movement, tremor in a body part during a target-directed movement [22] and worsens as the precision requirements increase [21].

Alusi, Worthington, and Glickman [22] studied these types of tremor in the arms of 100 patients with MS and found 16% with distal postural tremor and 18% with distal postural and kinetic tremor. Six percent had isolated intention tremor, and no subjects in their study showed signs of rest tremor. Eight percent had proximal postural and kinetic tremor.

Tremor frequencies associated with MS tend to be much lower than those associated with physiological tremor, which typically operates at 8 to 12 Hz [22]. Alusi, Worthington, and Glickman [22], Liu *et al*. [23], and Liu *et al*. [24] found postural, kinetic, and intention tremor frequencies for the upper arms to be between 3.5 and 5 Hz. Examining intention tremor more closely, Liu *et al*. [24] found peaks at 1 to 2 Hz when subjects could visually track targets that disappeared when the visual feedback was removed. A spike at 4 Hz was present with or without visual feedback for subjects with MS. The spike was much smaller for healthy control subjects.

Treating disability associated with tremor has been difficult. Mechanical or robotic treatments are not widespread, and medicinal treatments usually have side effects that counterbalance the benefits of alleviating the tremor [21]. Because the arm has more degrees of freedom than necessary to perform various movements, patients with tremor often develop compensatory strategies that use the non-afflicted part of the limb [23]. Alternatively, they may use both arms to perform a task, brace an arm, restrain movement altogether, or ensure other parts of the body are stabilized [21],[25]. Surgical measures such as thalamotomy have been attempted with a low benefit to side-effect ratio, and deep brain stimulation is a new treatment not yet proven to be superior [21].

1.1.2 Personal Mobility

Personal mobility is a necessary component for not only assisting individuals with MS but also maintaining an acceptable quality of life (QOL). Pittock *et al*. [10] studied a cohort of individuals with MS and found a general trend toward needing assistance with mobility. One in three patients worsened to the point of needing a cane or wheelchair in the study's 10-year period, and 51% of patients with an Expanded Disability Status Score (EDSS) of 6 or 7 in 1991 required "a wheelchair or worse" after the 10-year study period. Noseworthy reported that half of MS patients will require assistance walking within 15 years of the disease's onset [9]. Those more likely to deteriorate with regard to mobility include males, older individuals, and individuals with brainstem or cerebellar involvement [10]. As a person's mobility decreases (e.g., inability to climb stairs or increased susceptibility to fatigue), so does his or her QOL [7]. Advancements in materials, electronics, controls, and clinical techniques applied to wheeled

mobility during the past several decades have given mobility back to many people with severe physical impairments [26].

Not only can personal mobility devices such as scooters and EPWs restore mobility, they have important secondary effects as well. They can reduce fatigue, allow people to interact with their environment and society [7], and allow people to carry out daily activities [27], all of which are closely related to QOL. A recent development, for example, is the pushrim-activated, power-assisted wheelchair (PAPAW). Several studies have shown reduced energy demands, favorable ergonomic ratings during activities of daily living (ADL), and reduced joint range of motion with a PAPAW compared to individuals' manual wheelchairs [28]-[30]. Also, EPW drivers reported improved quality of life, mobility, and pain/discomfort ratings after four months of using an EPW [31]. Just as decreased mobility can lower one's QOL, restoring mobility or activity with assistive technologies has the ability to improve it [7],[32]. While EPWs have been shown to help individuals with severe physical impairments, few research studies exist related to the assistive technology needs of individuals with MS [7].

1.2 THE WHEELCHAIR USER INTERFACE

The user interface plays a pivotal role in the success or failure of an EPW as a mobility device [7],[27]. It is the means by which the user supplies commands to the wheelchair's controller, which in turn sends commands to the motors. While the most common control interface for EPWs is the movement joystick (MJ) [26],[33], researchers have been reporting success with isometric joysticks [4],[5],[34],[35], exploring alternative interfaces [2],[3],[6],[36]-[41] and control enhancers [42]-[56], and finding success with personalized controls [42],[49],[57],[58].

A MJ consists of a hand-sized stick, or shaft, that pivots at its base. The desired speed and turning rate correspond to the angle of deflection, and the desired heading corresponds to the direction of the deflection. An IJ, alternatively, consists of a rigid beam and force sensors. The desired speed corresponds to the magnitude of the applied force, and desired heading corresponds to the direction of the input.

1.2.1 Current Practices in Wheelchair User Interfaces

User interfaces have three main aspects: the control interface, the selection set, and the selection method [33]. The control interface is the physical means by which commands are provided to the wheelchair controller. It defines the input domain, or the set of motions or forces a user must produce to generate a signal for the controller. Examples of control interfaces include a MJ, a sip-and-puff switch array, a head array, or a touch pad; and examples of input domains include arm and wrist movements while holding the stick of a MJ or head movements to activate switches in a head array. The selection set includes the items available for control at a given time. Members of the selection set may include speed and direction commands for the motors, commands for tilt and recline actuators on the EPW, and inputs to electronic aids to daily living (EADL). The selection method describes how commands are selected. With direct selection, any command is available to the user at any time; and with indirect selection, commands are selected through scanning or coded access. Direct selection requires fine, controlled movements; but it is cognitively less challenging and produces quicker results than indirect selection, thus making it the preferred method for control.

The most common user interface for EPWs is the MJ [26],[33]. The MJ employs not only direct selection, but it allows graded input, where virtually an unlimited number of speed and direction commands are available to the user. There are two significant regions for the input domain of a MJ:

- The dead zone: the region near the center where slight stick movements do not cause any output signals.
- The proportional region: the region where displacing the stick farther from center causes a progressive increase in the output signal in the direction selected.

The ratio of the output to input signals within the proportional region defines the joystick's gain. The higher the gain, the more sensitive to movements the joystick will be. If the user has difficulty controlling the wheelchair with the standard MJ, control enhancers may be added.

Control enhancers enable users to make the most out of the selection method(s) that are available. Control enhancers may be physical, such as a larger handle on a joystick to assist users' grips, or be embedded in software, such as a low-pass filter to remove unintended signals

from being sent to the EPW controller. Commonly used control enhancers with MJs include templates, low pass filters, modified handles, and posturing techniques.

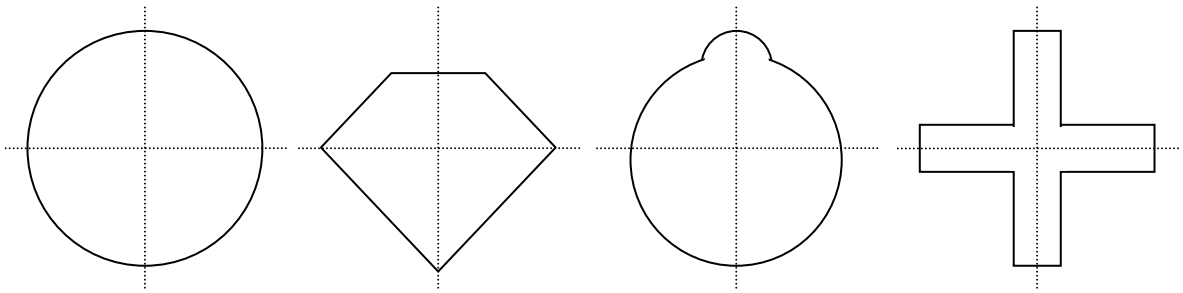
A template is a restraining outer boundary that limits the maximum deflection of the stick and helps "steer" the users' unintended movement while permitting a level of direct selection. "For some individuals, the use and type of joystick template means the difference between success [and] failure in the operation of a powered wheelchair" [33]. Examples of shapes the template may possess are depicted in **Figure 1**.

A low pass filter is a conditioning circuit that removes high frequency signals such as vibration or noise (usually greater than 8 Hz) from being sent to the EPW controller. Two common low pass filters, a moving average [59] and a Butterworth filter [47], may be employed in software and hardware, respectively. Using time-averaged data, the frequency response of a low pass filter may be calculated with equation 2,

$$2: \qquad H(e^{j\omega}) = \frac{1}{N+1} e^{j\omega(N/2)} \frac{\sin\left(\omega(N+1)/2\right)}{\sin(\omega/2)}$$

where N is the number of samples to average and ω is the input signal's frequency [60]. A second-order Butterworth filter exhibits low ripple in the pass band frequencies [61] and is depicted in **Figure 2**. The ideal transfer function for the circuit is given by equation **3**,

$$3: \qquad H(f) = \frac{\frac{R_3 + R_4}{R_3}}{(j2\pi f)^2 (R_1 R_2 C_1 C_2) + (j2\pi f)\left(R_1 C_2 + R_2 C_2 + R_1 C_1 \left(\frac{-R_4}{R_3}\right)\right) + 1},$$



(a) Circle *(b) Diamond* *(c) Notch* *(d) Cross*

Figure 1: Common template boundary shapes include (a) circle, (b) diamond, (c) notch, and (d) cross [62],[33].

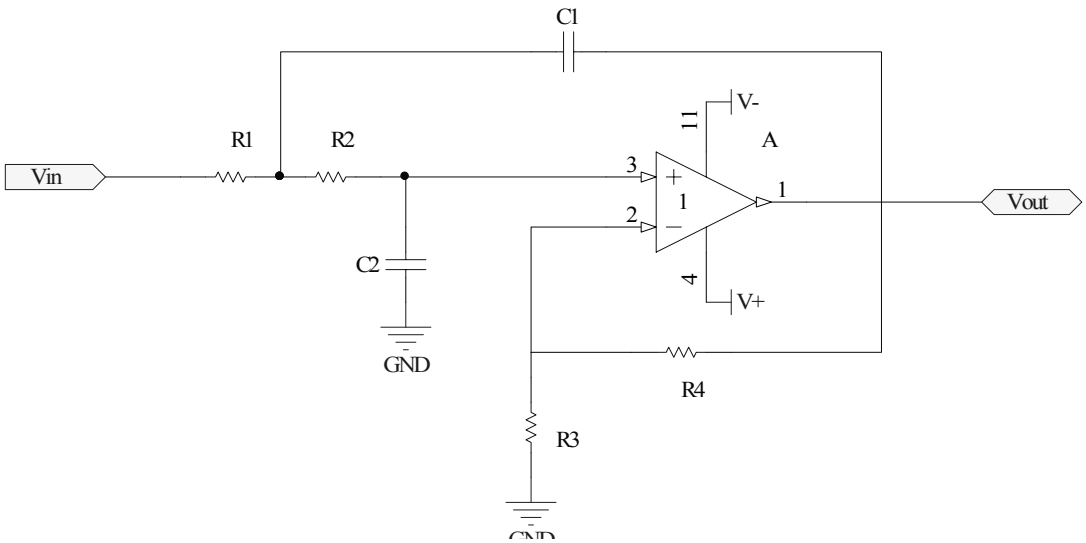


Figure 2: Schematic diagram of a second-order Butterworth filter [63].

where R_i and C_i are resistance and capacitance values, respectively, given in **Figure 2**; the numerator defines the filter's gain in its bandpass frequencies; and the denominator characterizes the filter's frequency response [63].

The handle's size and shape may be altered to enhance control. For example, a larger handle or a U-shaped handle that supports the sides of a user's hand may be helpful.

Stabilizing proximal body parts also assists with EPW control [33]. Gillen [25] described how providing full head and trunk support, adaptive seating, volar wrist support, and an adapted armrest to support the entire forearm to a 40-year-old male with MS enhanced his control of an EPW. Nonetheless, tremor, spasms, weakness, and inadequate range of motion (ROM) inhibit a person's ability to control an EPW with a MJ [34]. User interfaces such as switch control, head control, or sip-and-puff control are available; but they are much less efficient and more cognitively challenging to use [33]. Likewise, low-pass filters cause a delay in the wheelchair, increase instability, are not adequate when the tremor frequency is near the intentional frequency, and reduce performance for individuals with low tremor powers [41]-[43],[47],[58],[64]. Therefore, researchers have been exploring a variety of alternative methods for improving access to powered mobility.

1.2.2 Research Trends in Wheelchair User Interfaces

Methods to improve EPW control that researchers have been exploring are quite diverse. Almost serendipitously, as a result of these different studies researchers have been finding that the determining factor may not necessarily be on a universal control interface or universal control enhancer that significantly improves control but on the act of personalization. That is, providing a user interface that is tunable to the client's personal characteristics may provide the best increase in control. The methods researchers have been exploring, described in more detail below, fall into the general categories of control interfaces and control enhancers.

1.2.2.1 Alternative Control Interfaces

Researchers have explored a variety of alternative interfaces for EPW control. Pellegrini *et al.* [36] gave 47 individuals with Duchenne muscular dystrophy (DMD) and restricted EPW driving abilities the opportunity to use alternative interfaces to the standard joystick. Interfaces included a mini-joystick, an isometric mini-joystick, a finger joystick, and a tactile pad. Of the 18 who chose to switch devices and used them, all regained their ability to drive unrestricted. Powell and Inigo [6] found positive qualitative results with a hybrid force- and movement-sensing joystick. Their joystick was designed for front wheel drive EPWs which are sensitive to changes in heading. The joystick used hydraulics and pressure sensors for left-right steering and position sensing for forward and reverse. Repperger and others who developed a force reflecting joystick for Air Force pilots found performance improvements in individuals with spasticity who used the joystick to complete tracking tasks [37],[38]. The joystick was able to distinguish between spasms and normal movement and activate a damping circuit when a spasm occurred. Rosen and students at MIT explored a series of joysticks with a mechanically damped handle [2],[39],[40]. The joysticks reduce intention tremor at the hand and have the benefit of removing the DC offset that occurs when users makes contact with the template or cross the dead zone [2],[40]. In a study with three subjects, they found a significant reduction in tremor while subjects used the medium damped setting compared to no damping [2]. Pledgie *et al.* [41] used the PHANToM, a small robotic arm used in haptic interfaces, to attenuate tremor through impedance control. Modeling the hand-forearm movement with a linear second-order time-invariant transfer function, they altered rate and acceleration feedback coefficients to reduce tremor by 10 dB and

20 dB at the subjects' tremor frequencies. They found this technique not to be suitable for individuals whose tremor frequency is near the frequency of their normal voluntary movement and noted the constant coefficients to be restrictive in potentially changing environments. An alternative interface that has had a particular bit of success is the IJ.

IJs may prove to be a viable control interface alternative to MJs because a number of studies have shown similar or improved performance in EPW driving with an IJ vs. a MJ. Unlike MJs, IJs remain in the same position when a user applies forces to them; thus, they require virtually no ROM to operate. They operate by transforming the user's force input to a signal usable by a controller and are thus said to be "force sensing." Transducers typically used with IJs include pressure sensors or strain gages. Cooper *et al.* [34] compared an IJ with a MJ by asking 10 experienced EPW drivers to navigate a short course. They found that the subjects drove faster with the IJ while moving forward around curves and had a lower error when driving reverse. Controlling both speed and heading is a complex task that appears to be easier with an IJ, they claimed. Cooper *et al.* [4] found no difference between an IJ and MJ in root mean square (RMS) tracking error and completion time while novice and experienced EPW drivers drove an EPW over a 60-m course. They did find, however, that subjects had a lower RMS tracking error driving straight forward and driving in a circle with an IJ but a higher RMS tracking error during turning tasks with an IJ. Subjects reported that they experienced fatigue or soreness in the hand while driving the IJ. While driving in a virtual environment with isometric and movement joysticks, subjects showed no significant difference in driving time and root mean square error (RMSE) between joysticks [35]. Riley and Rosen [42] compared an IJ and MJ with and without filter algorithms using an on-screen target tracking task. They found that velocity control with an IJ was generally more successful than position control with an IJ. Guo, Grindle, and Cooper [5] constructed a force-sensing joystick that allowed users to control an EPW with their chin. They noted that its sensitivity and tremor rejection algorithms could be customized in software to fit individual users' needs.

Two studies have shown MJs to be superior to IJs and have thus revealed some interesting features of IJs. Rao, Seliktar, Rahman [65] compared the performance of individuals with and without cerebral palsy (CP) during computer tracking tasks with IJs and MJs. Here, the authors found longer movement paths for the IJ to be a result of the IJ being more sensitive to tremor. Alternatively, MJs may allow tremor to be dampened by inertial loads of the hand and

arm and possibly by friction inside the MJ. They also cited lack of somatosensory feedback from joint receptors to be a probable cause for reduced performance while using an IJ. Stewart, Noble, and Seeger [66] compared the performance of children with CP and one adult in a computer tracking task using a modified MJ and an IJ. While a majority of the subjects performed better with the MJ, the authors noted that the subjects had difficulty holding the IJ because of the handle's size; and they noted that the subject who had had no prior experience with either type of interface performed as well with the IJ as the MJ. Therefore, tremor filtering; ensuring a proper, ergonomic interface to the joystick; and training all are important features to consider when using an IJ as a control input device.

1.2.2.2 Alternative Control Enhancers

In addition to alternative interfaces, signal processing techniques, orthotics, and robotics may enhance wheelchair control. When Riley and Rosen compared IJs and MJs with and without tremor filtering, rather than recommending a superior input device, they stated that filtering and control algorithms are more important than the type of interface [42]. They also said that decreasing the gain improves the signal to noise ratio, while tremor power and tracking error increased with increasing gain. Gonzalez *et al*. [43] used a linear finite impulse response (FIR) filter with what they called pulled-optimization to filter tremor during a target tracking device. Their goal was to minimize the filtered mean-square error with delay correction, F-mse$_d$, a performance measure they developed that is not biased by tracking-delay problems, overshoots, undershoots, and shortcuts. Riviere *et al.* [44]-[46] investigated adaptive notch filters and developed a weighted Fourier linear combiner (WFLC) to model and filter tremor. Notch filters have the advantage of suppressing only the tremor frequency and reducing distortion of intentional signals [47]. Adaptive filters, those that self-adjust their parameters, have an added benefit because tremor is not always constant [7],[47]. Riviere's WFLC adjusts its parameters based on the input signal's noise frequency and amplitude and was found to be effective for handwriting and microsurgical tool applications. Nho [47] further developed Riviere's WFLC by adding instantaneous bandwidth information to the filter inputs. He found the modified WFLC performed similarly and better than Riviere's WFLC and a 3-Hz low-pass filter, respectively, in a virtual environment for wheelchair driving. Phillips, Repperger, and Chelette [48] developed an algorithm based on the acceleration-velocity relationship to detect spasms during a target

tracking task with a MJ. On a force-reflecting joystick, the algorithm allowed subjects with spasticity to perform at levels near subjects without spasticity [37].

Researchers at MIT developed several prototype orthoses for tremor suppression [49]-[51], all of which may enhance wheelchair control. The controlled-energy-dissipation orthosis (CEDO) and modulated-energy-dissipation manipulator (MED Arm) generate resistive viscous loads with computer-controlled magnetic particle brakes. The orthoses attach to patients' forearms and are mounted on a standard wheelchair. While CEDO can operate in three degrees of freedom, MED Arm operates in six degrees, allowing the resistive forces at the tip to be directly opposite the tremor force. They found that the damping loads can attenuate tremor but that too much damping was not effective for some subjects. Subjects also reported fatigue, but the researchers suggested that it may have been a testing effect. The Viscous Beam, also developed at MIT, is a wearable orthosis that attaches to the forearm and cuff to attenuate tremor in wrist flexion and extension. The orthosis helped reduce tremor and improve control in five subjects performing a set of functional tasks.

Techniques in robotics may enhance EPW control through the idea of shared control. Shared control devices augment a user's ability to drive a wheelchair through a variety of internal and external sensors and an automatic control system. Internal sensors may monitor the state of the wheel speed and position, power stage current, bearing failure, motor temperature, and the primary processor. External sensors monitor potential hazards (e.g. stairs, curbs, steep hills) and include infrared detectors, ultrasound transducers, video, structured light, magnetic sensors, and telemetry systems. The automatic control system may compute a desired trajectory based on sensor readings and the user's input with a variety of algorithms (e.g. probabilistic models (e.g. partially observable Markov decision process (POMDP)), human decision or human behavior models, control loops) [52]. For example, Brienza and Angelo [3] constructed a force feedback joystick and developed two modes for its operation. One mode alters the compliance of the joystick handle based on the wheelchair's proximity to an obstacle, while the other mode causes the joystick to move, steering the EPW away from an obstacle. Examples of ongoing shared control projects include SIAMO [53],[54] and the Smart Wheelchair Component System (SWCS) [55],[56].

1.2.2.3 Benefit of Personalized Controls

Other research has shown a benefit in personalized controls. Trispel, Rau, and Brudermann found reduced error and effort for EPW control when the force characteristics on a MJ were tuned to the individual's motor abilities [57]. A study related to computer access for individuals with MS and tremor [58] showed that those with more severe tremor (indicated by lower Nine Hole Peg test scores) benefited from personalized software for PC cursor control. The researchers also found that no single configuration for PC cursor control satisfied everybody, suggesting a need for personalized control. Similarly, Aisen *et al*.'s study [49] showed that no single configuration for damping arm tremor benefited every subject. And, while some subjects showed improvements with the IJ, Riley and Rosen [42] concluded that control algorithms and interfaces that are adaptable to the individual users are more important than the specific type of interface.

1.3 OVERVIEW OF THESIS

Assistive technology has the remarkable capability of restoring the quality of life for millions of individuals. Though the outcome is variable among individuals with MS, a trend exists toward needing progressive assistance with mobility. When this happens, problems with tremor and fatigue may inhibit the use of mobility devices. The aim of this thesis, therefore, is to develop a wheelchair user interface that mitigates these problems and allows users to control an EPW successfully and independently.

The remainder of this thesis is divided into eight chapters. Chapter two describes the technical development of the novel EPW user interface, dubbed the variable compliance joystick (VCJ), from mechanical and electrical perspectives. Chapter three details the experiments used to validate the static and dynamic responses of the VCJ. Chapter four covers the software and procedures for tuning the VCJ to the subject. Chapter five describes the research methods for evaluating the VCJ while individuals with MS use it to drive a virtual wheelchair. Chapter six highlights the results from a study with 4 subjects with MS. Chapter seven provides the author's insights into the overall performance of the VCJ. Chapter eight summarizes lessons learned in addition to suggesting future research with the VCJ.

2.0 THE VARIABLE COMPLIANCE JOYSTICK

Given the impairments associated with MS, popularity of the conventional MJ, benefit of control enhancers, recent success with the IJ, and desire for personalized controls, we began thinking of a user interface that could incorporate the best of all worlds to meet the needs of individuals with MS. We developed the following design criteria for the user interface:

1. The user interface should be capable of operating as a conventional MJ, where displacement of the joystick results in a proportional output signal.
2. The amount of force required to deflect the joystick fully should be adjustable.
3. The user interface should employ the following control enhancers
    a. Dead zone,
    b. Templates,
    c. Bias axis,
    d. Non-orthogonal axes,
    e. Optimized handle, and
    f. Tremor filter.
4. The user interface should be capable of being configured as an IJ.
5. The user interface should be readily tunable to the user's preferences.
6. To compensate for fatigue in MS, the sensitivity of the joystick should be dependent upon the user's fatigue status with the option to use a gain schedule.

A detailed list of technical requirements (e.g. force to deflection ratios, strength characteristics, and interface requirements) is provided in Appendix A. Because the force required to deflect the joystick is designed to be adjustable, the proposed user interface has been named the variable compliance joystick (VCJ).

The VCJ was built using the IJ developed at the Human Engineering Research Laboratories (HERL) [4],[67] as a baseline (see **Figure 3**). The HERL IJ features a 3/8-inch

diameter tool steel post, full Wheatstone strain gage bridges for forward-reverse and left-right directions, instrument amplifiers, second-order Butterworth filters, a 24-bit A/D converter, a MC68HC11 microcontroller, 8 Kb of ROM and 32 Kb of RAM, a quad 12-bit D/A converter, and a DB-9 serial connector. Operator controls such as speed control, horn signal, power switch, and battery status bar graph are included at the rear of the joystick enclosure. It is capable of driving a Quickie P300 EPW as well as transmitting interlaced x- and y-axis data samples at 83 Hz with 12 bits of resolution. The joystick was used in several real world and virtual driving studies comparing isometric controls with conventional MJs [4],[34],[35],[67]-[69]; it was used as a platform to test a novel adaptive filter algorithm [47]; and it is currently being used in two studies to determine the effects of personally-fitted algorithms for EPW driving for individuals with traumatic brain injury [70] and cerebral palsy [71]. While the core concepts of the HERL IJ remain the same, hardware and software components have been updated to expand its capabilities.

The VCJ is mechanically more versatile, provides for variable compliance including a full isometric mode, can produce x- and y-axis data samples at up to 250 Hz with 12 bits of resolution, and can be expanded to drive any EPW with analog inputs between ±10 V. The remainder of this chapter will describe the technical development of the VCJ, first describing the system as a whole and then its three main components.

Figure 3: HERL IJ developed by Spaeth [67].

2.1 OVERVIEW

Signals travel through the VCJ in three separate stages: the joystick post, the signal conditioning unit (SCU), and the signal processing unit (SPU). A diagram depicting the signal flow is provided in **Figure 4**. The joystick post is the control interface: it is the physical means by which commands are supplied to the wheelchair. The SCU converts the strain that results from the force to electrical signals, which are then amplified and filtered. The SPU converts the analog signals to digital values and applies various signal processing algorithms (e.g. dead zone, templates, bias axis, etc) to the signals. The readings are then available for the Virtual Driving Simulation (VDS) (section 5.4.2 describes the VDS in more detail) to use or for other routines to convert to analog signals for EPW driving. When driving an EPW, the output voltage of the VCJ is software-selectable within the ranges of ±10 V to mimic that of a conventional MJ (some joysticks output 6±1 V, and others output 2.5±1 V) thus mitigating the need to tap into the EPW controller. The interface between the SCU and SPU is designed to be modular, where the SPU can accept any two-dimensional, differential input that varies between ±500 mV.

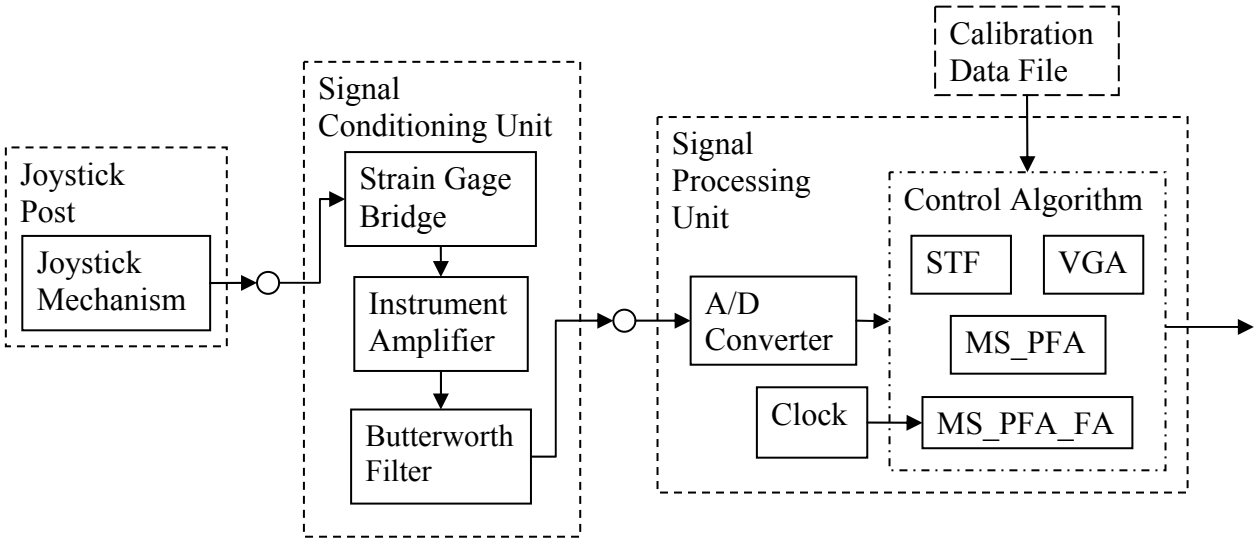


Figure 4: Signal flow of the VCJ.

2.2 JOYSTICK POST

2.2.1 Initial Design Concepts

Based on the requirements, we developed several design concepts and prototypes for the joystick post before choosing the final version. An initial concept was to dampen tremor mechanically with a spring-mass-damper [72]. However, the necessary spring constant would cause the joystick to behave very differently from a conventional MJ: while a conventional MJ requires about 4 N to produce full deflection [67], the force required to push the joystick to full deflection with mechanical tremor isolation would be more than 220 N. The first prototype, illustrated in **Figure 5**, featured an elastomer and open center gimbal on top of a load cell. As the amount of elastomer that was free to bend decreased, so did the amount of compliance, until the joystick behaved isometrically [73]. However, while testing the joystick, we found that the handle would not return to the center position after being deflected for extended periods of time. The second prototype, depicted in **Figure 6**, was designed to be more robust. To solve the recentering problem, closed-coil and open-coil springs replaced the elastomer. To improve the robustness of the joystick, aluminum and stainless steel parts replaced Si-40 resin, the plastic used for the rapid prototyping process. However, we found that this joystick was difficult to use as an input device because the wheelchair did not always drive in the intended direction. The causes for this are likely related to the buckling spring and the pivot mechanism.

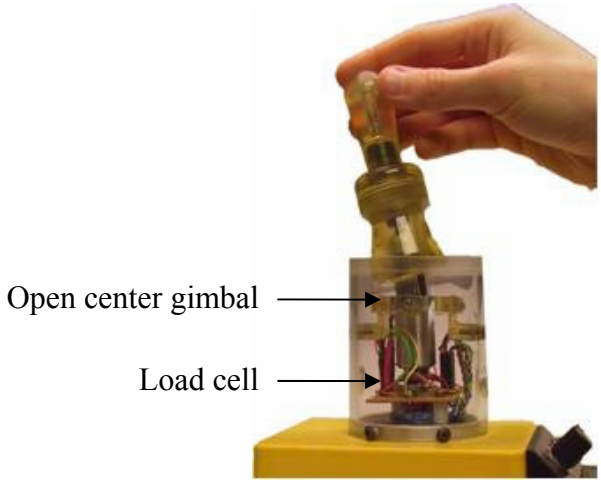


Figure 5: The first prototype variable compliance joystick [73].

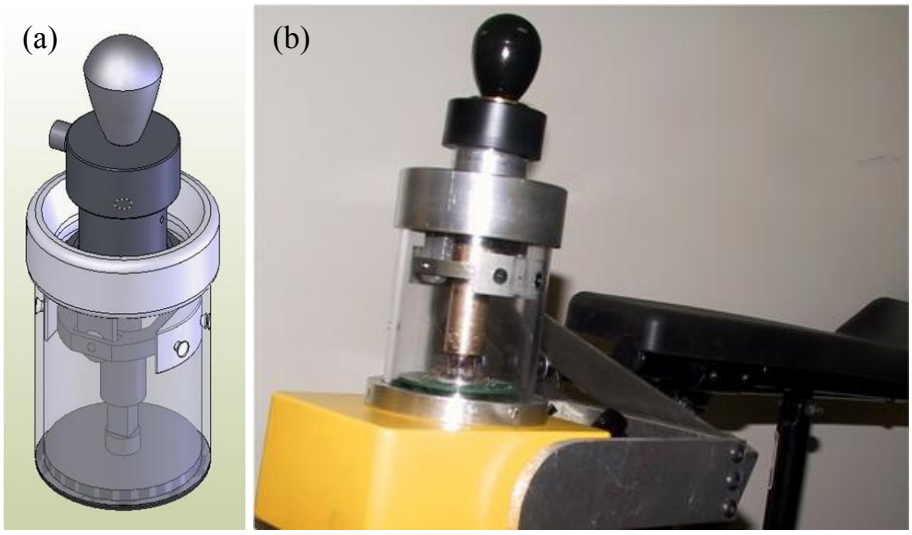


Figure 6: Second prototype of the variable compliance joystick. (a) solid model (b) completed prototype.

2.2.2 Proposed Solution for the Joystick Post

The final design concept uses two different configurations for EPW driving and fatigue characterization, both of which use a load cell to convert force input into an electrical signal. In driving mode, depicted in **Figure 7**, the VCJ uses a similar pivot and spring mechanism to that of a conventional MJ. The amount of force required to deflect the joystick can be adjusted by selecting different springs or adjusting the pretension in the spring with a nut. Mechanical templates provide a physical boundary for joystick deflection, and a variety of handles provide ergonomic grips for the driver. While the joystick can behave like a MJ because of the similar compliance mechanism, it can also behave like an IJ if the spring is replaced with a solid piece of metal. In fatigue characterization mode, shown in **Figure 8**, an aluminum shaft may be inserted on top of the load cell instead of the pivoting mechanism. The aluminum shaft provides the added robustness for fatigue characterization, where we expect loads to reach 300 N [4]. Loads while driving with the joystick, conversely, are not expected to exceed 35 N. Because of their significance, components discussed in detail include the load cell, pivoting mechanism, templates, handles, and aluminum shaft. Technical drawings for all of the joystick post's parts are included in Appendix B.

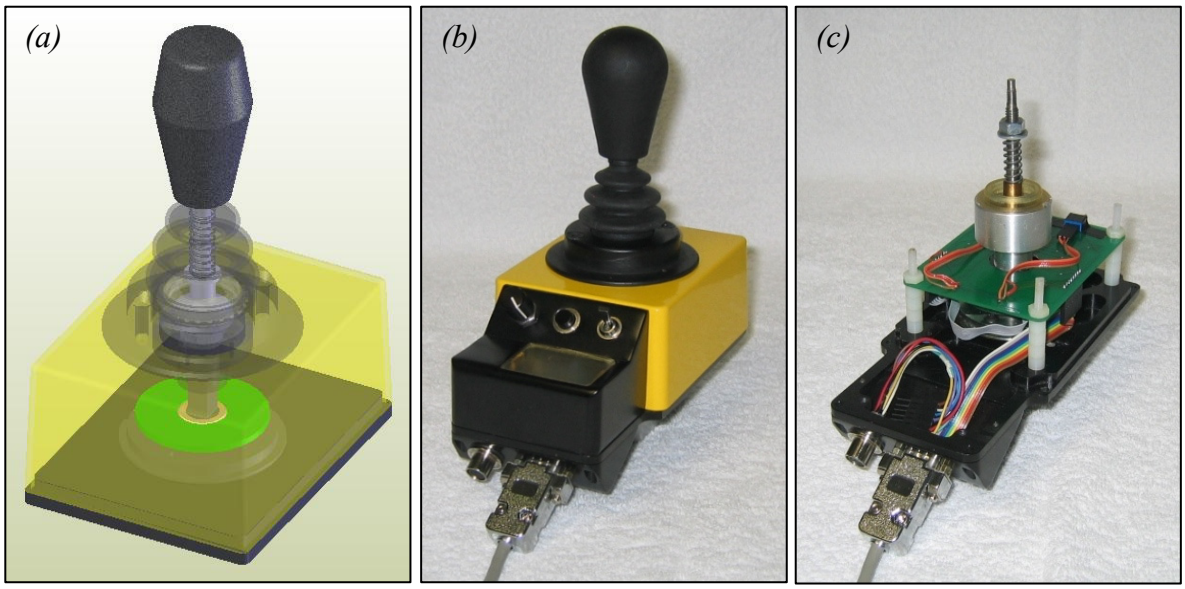


Figure 7: The VCJ (a) as a solid model to illustrate more inner components; (b) fully assembled; and (c) with the box cover, bellows, and handle removed to reveal its inner components. The MiniDIN-9 connector provides an interface directly to the strain gage bridge.

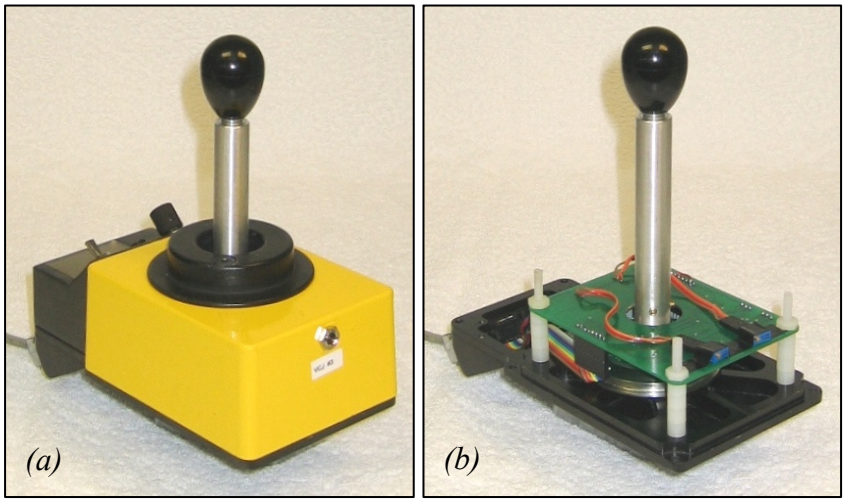


Figure 8: The VCJ with the solid insert for fatigue characterization (a) fully assembled and (b) with the top cover removed.

2.2.2.1 Load Cell

The dimensions and material of the HERL IJ's load cell were modified to improve its robustness. The original dimensions of the post were 0.375 in., 0.25 in., 3.4 in., and 2.56 in., for the diameter, width of strain gage mounting surface, total length, and distance between the strain gage mounting surface and top, respectively. While Widman showed that a 300-N force would not cause the strain gages to deform beyond their limits [74], a finite element analysis with SolidWorks' ® COSMOSXpress™ revealed weaknesses on the strain gage mounting surface and at the transition between the square cross-section of the strain gage mounting surface to the circular cross-section of the shaft. SolidWorks calculated the safety factor (SF) to be 0.23, where an SF below 1.0 indicates that the part has begun to yield under the modeled loading conditions. Therefore, we increased the shaft diameter and width of strain gage mounting surface to 0.50 in. and 0.32 in., respectively, and selected Nitronic 50 as the material. The resulting SF is 1.11. The stress concentrations of the two designs are depicted in **Figure 9**. The yield strength in the original design is 2.068×10^8 N/m^2 and is equivalent to the aqua color **Figure 9a**. Thus, shades above aqua (e.g. green, yellow, and red) indicate places where the part is likely to have begun to yield. The yield strength for the new design is 3.93×10^8 N/m^2, which is above any color on the color gradient in **Figure 9b**. Looking at Widman's criterion for the length to width cubed ratio,

$$4: \qquad \frac{L}{w^3} \leq 1.723 \times 10^3 \, in^{-2},$$

to ensure the strain gages are not stressed beyond their limits, we have

$$5: \qquad \frac{L}{w^3} = \frac{4.52125}{0.32^3} = 138.0 in^{-2} < 1.723 \times 10^3 \, in^{-2},$$

where L is derived from the length of the aluminum shaft for fatigue characterization and the center of the strain gage mounting surface, which satisfies the condition.

Furthermore, we modified the attachment of the load cell to the base of the joystick. Originally, a shaft collar connected the post to the base, but this design was a bit cumbersome to machine and did not provide a mate secure enough to withstand some of the torques applied to the joystick. Therefore, we tapered the bottom of the isometric post 15° and added a 10-32 tap, making sure the tap did not extend beyond the bottom plane of the mounting surface. A flanged button screw is then used to cinch the joystick post into the base, as shown in **Figure 10**.

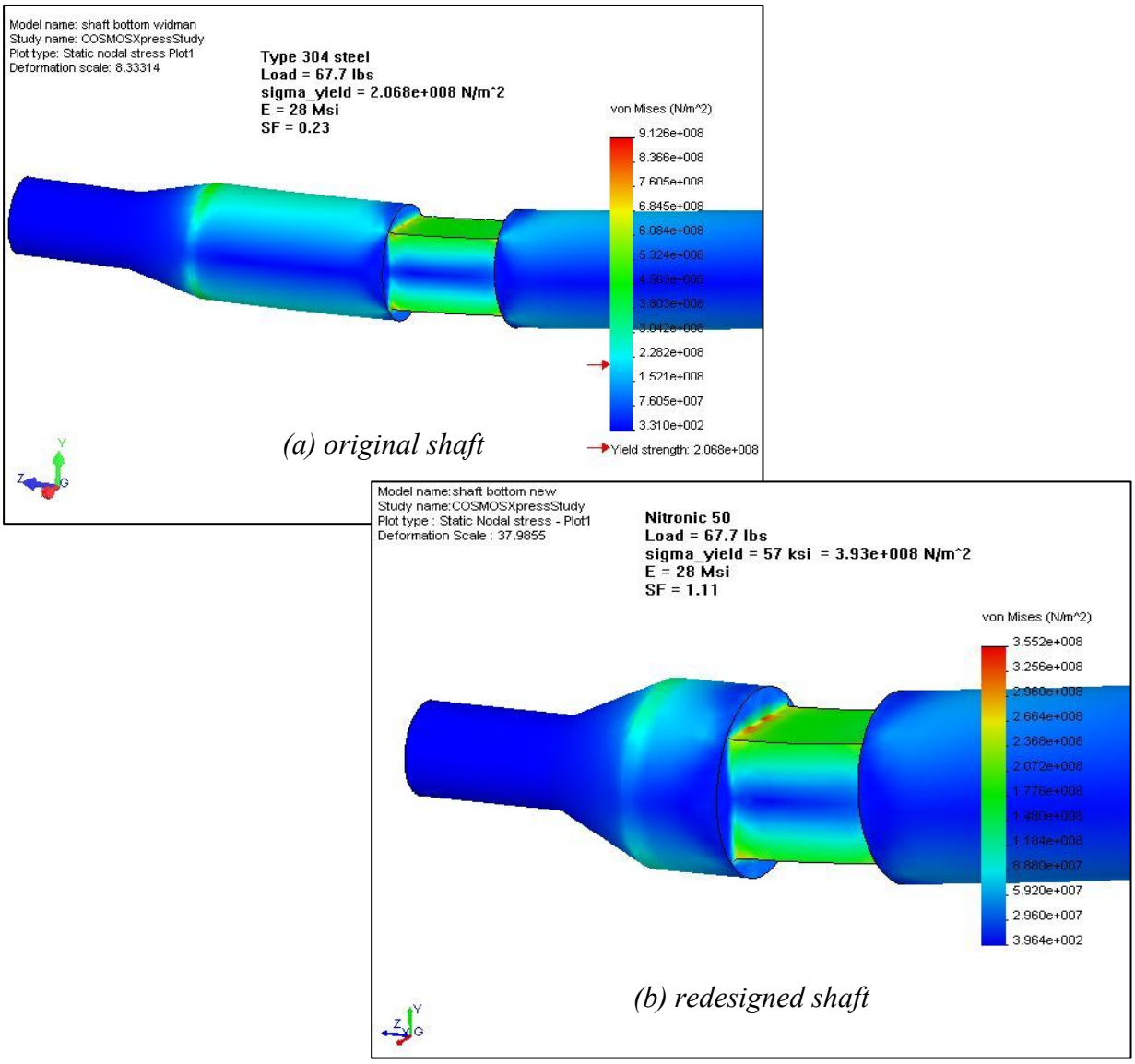


Figure 9: Stress concentrations of (a) original and (b) modified shaft designs.

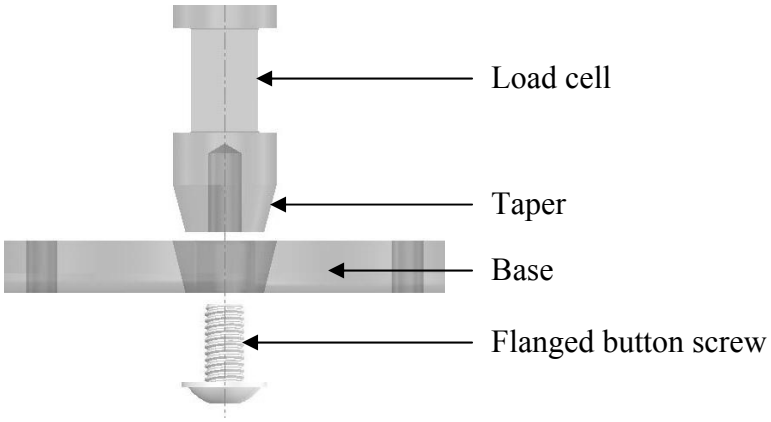


Figure 10: Exploded, semi-transparent view of the taper lock connecting the load cell to the base.

2.2.2.2 Pivot Mechanism

The first version of the pivot for the final design concept used a gimbal with pins (**Figure 11**). However, a static loads analysis, shown in Appendix C, revealed that the pins would not be strong enough for repeated use. Therefore, we used a steel spherical bearing in the gimbal's place (Figure 12). The joystick shaft is press fit into the bearing, and the bearing is tied down with a stainless steel enclosure and machine screws. On top of the stainless steel enclosure is a 0.040-in. thick aluminum ring to prevent the boot from catching on the heads of the machine screws. The boot is composed of oilite, and EP/Red bearing grease (Lawson) was rubbed into the bearing to minimize friction and wear in the moving parts. The pivot mounts to the load cell with a sliding press fit and is held in place with two brass, 6-32 set screws. Type 309 stainless steel was selected for the joystick shaft for its good machinability and corrosion resistance.

2.2.2.3 Mechanical Templates

Four templates, shown in **Figure 13**, were constructed to enhance control while in movement mode. Even though the templates would not effect the joystick signal, mechanical templates provide feedback to the driver about the location the joystick. Notches at the edge of the templates allow for bias axis angles in 15° increments. This allows the "forward" direction for the user to be altered without needing to rotate the entire VCJ or the isometric post. Since the

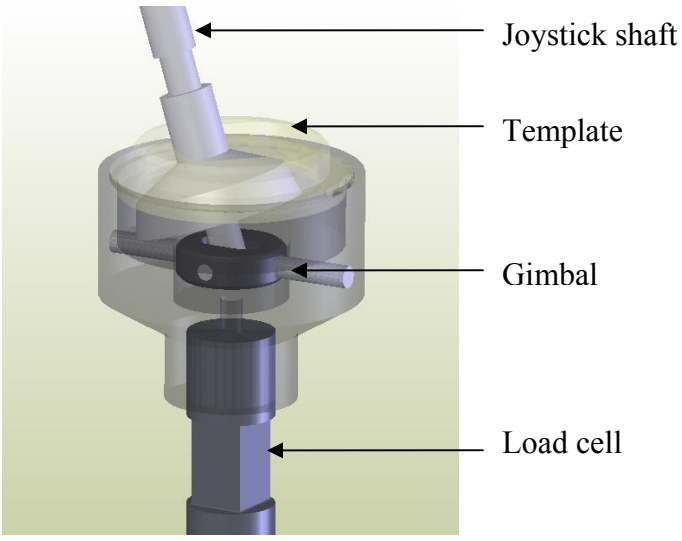


Figure 11: Gimbal pivoting mechanism design concept.

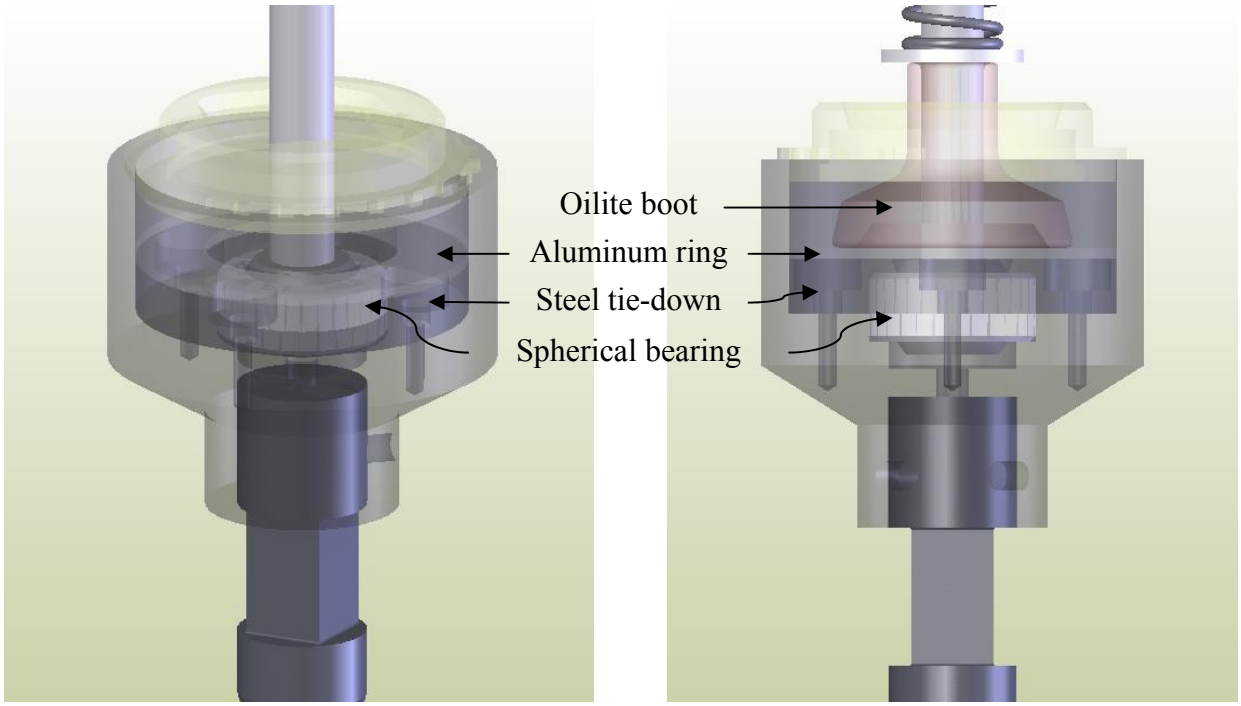


Figure 12: The spherical bearing and its tie-down pieces.

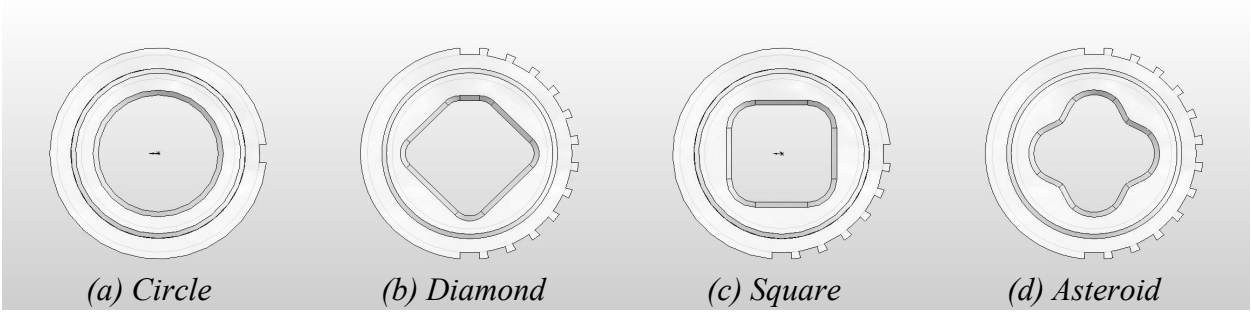


Figure 13: Templates constructed include (a) circle, (b) diamond, (c) square, and (d) asteroid.

isometric post is not rotated, the appropriate bias angle is applied in software to correspond to the physical orientation of the template.

2.2.2.4 Handles

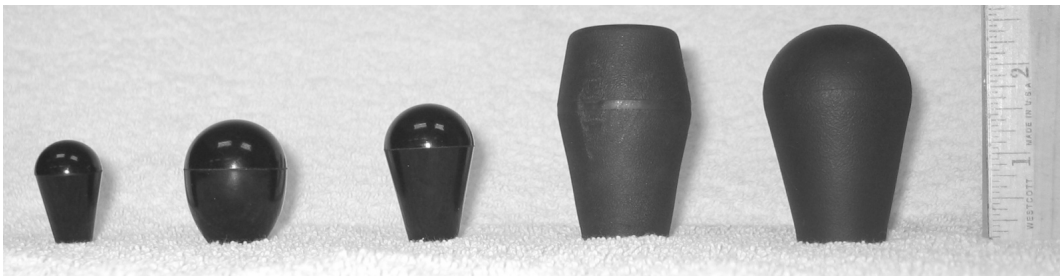
Because the physical interface plays an important role for IJs, a variety of handles, both in size and shape, are available for the user to grasp. The handle on the HERL IJ has a 0.375 in. radius, is 1.125 in. long, and screws onto the top of the joystick post. However, a handle with a small radius has potential to cause a pressure point in the user's hand, while a larger radius distributes the force [75] and consequently improves comfort. Furthermore, different shapes, such as cylindrical and T-shaped, or field goal posts, are currently used with standard MJs as means to enhance control. Therefore, a selection of joystick handles, descriptions and photographs of which are provided in Table 2 and **Figure 14**, respectively, are available to use with the VCJ.

2.2.2.5 Aluminum Shaft

Since designing the pivot mechanism to handle 35 N proved to be a challenge and the pivot mechanism is not permanently bonded to the load cell, we opted for a more robust approach for the fatigue characterization process. During this phase, as much as 300 N may be applied to the joystick, and we added a 50-N margin of error. Therefore, we selected a solid, 5/8-in. aluminum shaft to be inserted on top of the load cell during fatigue characterization. The diameter was chosen to ensure a 1/8-inch wall thickness at the load cell interface (the reader is referred to

Table 2: Properties of handles available to use with the VCJ.

ID Number	Shape	Mass (g)	Radius (in.)	Height (in.)	Material
1	Dome top, Tapered	10	0.375	1.22	Phenol
2	Oval	34	0.565	1.47	Phenol
3	Dome top, Tapered	28	0.50	1.635	Phenol
4	Flat top, Tapered	56	0.73	2.48	Rubber
5	Dome top, Tapered	62	0.81	2.51	Rubber



(1) *(2)* *(3)* *(4)* *(5)*

Figure 14: Photographs of each of the handles.

Appendix D for relevant calculations). Like the pivot, it mounts to the load cell with a sliding press fit and is secured with similar brass, 6-32 set screws. Once the hardware for the joystick post was complete, the electronics were examined to ensure optimum signal acquisition from the strain gage bridge and to prepare the signals for the SPU.

2.3 SIGNAL CONDITIONING UNIT

2.3.1 Circuit Design

While the primary roles of the SCU are to convert the force input to an electrical signal, amplify the signal, and filter it, accessories such as a gain switch for the amplifiers and a voltage inverter were also required. A similar set of full Wheatstone bridge configurations as in the HERL IJ were used with the VCJ. Four metal foil strain gages convert force to a usable signal for the speed and direction axes. Five components in the VCJ prepare the signals for the SPU: a voltage inverter, instrument amplifiers (one for each axis), bias potentiometers (one for each axis), a switch, and Butterworth filters (one for each axis).

2.3.1.1 Voltage Inverter

While the A/D converter on the HERL IJ accepted only positive voltages, the NI data acquisition card (DAQCard) used in the SPU accepts ranges of ±10 V, ±5 V, ±0.5 V, and ±0.05 V depending on its software-selectable, internal gain. We considered using only the positive range of the NI-DAQcard to keep the original electronics, but this would have reduced the signal's resolution by

one half. Therefore, supported by the +5 V supply from the NI-DAQCard, we added a MAX660 chip to the circuit to provide a negative voltage supply. With a target range of ±0.5 V and an altered load cell, a new set of gains for the instrument amplifiers was derived analytically.

2.3.1.2 Amplifiers

Simple beam bending was assumed for the load cell [76], as given by equation 6

$$6: \qquad \varepsilon = \frac{6PL}{Ebt^2},$$

where ε is the strain; P is the input force; L is the length of the joystick post; E is the post's modulus of elasticity; and b and t are the post's width and thickness, respectively. The strain is converted to a voltage signal E_o with the strain gage bridge as given by equation 7

$$7: \qquad E_o = E_i F \varepsilon + \varepsilon_{bridge},$$

where E_i is the bridge's excitation voltage, F is the gage factor for the given strain gages, and ε_{bridge} is an adjustment to correct for human error during the strain gage mounting process. The instrument amplifiers multiply the signal by their gain factor G_{INA}, which is determined by the gain resistor R_G as shown in equation 8

$$8: \qquad G_{INA} = 1 + \frac{50k\Omega}{R_G}.$$

The Butterworth filter's gain is provided in the numerator of equation **3**. Therefore, the output JP_{out} of the joystick post's electronics is summarized in equation 9

$$9: \qquad JP_{out} = \left(\frac{R_3 + R_4}{R_3} \right) G_{INA} E_o.$$

Solving for R_G in terms of the joystick post's fundamental components and input force yields

$$10: \qquad R_G = 50\ k\Omega \left(\frac{JP_{out} \left(\frac{R_3}{R_3 + R_4} \right)}{E_i F \frac{6PL}{Ebt^2} + \varepsilon_{bridge}} - 1 \right)^{-1}.$$

Since the joystick post is expected to act in two modes, one for driving and one for measuring the user's MVIC, two different input forces were selected as cutoff forces for the joystick post. The cutoff force is defined as the force above which the output signal of the joystick post's electronics does not increase. The cutoff forces to operate the joystick while

driving and measuring the MVIC were selected to be 25 N and 350 N, respectively. The driving cutoff force is consistent with the HERL IJ, which has a cutoff force of approximately 27 N [67]. While a lower number would improve the signal to noise ratio (average forces seen while using the HERL IJ are 5.5±2.4 N [69]), prior experience with the HERL IJ has shown some subjects to apply more than 27 N while operating the joystick. Keeping a high cutoff force minimizes the risk of clipping, which can disrupt digital tremor filters [2]. The 350-N cutoff force is derived from adding a 50-N margin of error to the maximum input force on a joystick measured by Cooper *et al.* [4]. Therefore, with the parameters listed in Table 3 and using equation 10, the gain resistors for driving and measuring MVIC were designed to be 327.24 Ω and 5.423 kΩ, respectively.

2.3.1.3 Bias Potentiometers

To account for inconsistencies in mounting strain gages, two bias potentiometers were added to the V_{REF} inputs of the instrument amplifiers. By using a multimeter to measure the output of the

Table 3: Parametric values used for determining the gain resistor.

Parameter	Driving	MVIC
JP_{out}	0.500 V	0.500 V
R_3	22 kΩ	22 kΩ
R_4	22 kΩ	22 kΩ
E_i	5 V	5 V
F	2.105	2.105
P	25 N = 5.620 lbs	350 N = 78.68 lbs
L	4.2025 in	4.52125 in
E	28000000 psi	28000000 psi
b, t	0.32 in	0.32 in
ε_{bridge}	0 V	-2.0e-5 V

circuit and a small, flathead screwdriver, tuning the potentiometers allows the circuit output to be within 5.0 mV of zero when there is no force input.

2.3.1.4 Gain Switch

The original concept for switching between gain resisters for driving and MVIC modes of the VCJ involved manually replacing axial resistors on the circuit board. But, this method was found to be time-consuming and cumbersome. A single pull double throw (SPDT) switch, therefore, was included in the circuit to simplify the substitution. The SPDT switch allows the gain resistors for both axes to be switched with one toggle. Since power is supplied through the NI-DAQCard and not controlled via the joystick enclosure, the power switch from the HERL IJ assumed the role of toggling between gain resisters for the current design. The toggle was rotated 90° both to prevent confusion with the switch's purpose on the HERL IJ and to facilitate ease of use, where the up position indicates high gain and the down position indicates low gain.

Interestingly, the output bias of the circuit depends upon which gain resistor is selected. If the circuit is tuned in high gain mode (such that zero force input results in less than 5.0 mV output), the output bias after switching to low gain mode becomes 22.8 mV and -62.5 mV for the direction and speed axes, respectively. Finding a way to isolate the components seems problematic because internal schematics of the instrument amplifier reveal the V_{REF} input and R_G inputs to be connected. Therefore, software will need to recalibrate itself after switching between driving and MVIC operating modes. And while it is possible to tune the bias to a midpoint where either equal force or equal signal will need to be applied to achieve zero output in both modes (e.g., tune the output to be -11.4 mV for the direction signal in high gain mode), we decided to tune the joystick to 0 mV (specifically, within 5.0 mV of 0 mV) while in high gain mode since this mode will be used for the majority of the testing.

2.3.1.5 Butterworth Filter

The cutoff frequency for the second-order Butterworth filter in the HERL IJ was originally 100 Hz. However, the fastest sampling rate that could be achieved in the VDS was 60 Hz. Since the sampling rate must be greater than or equal to 2 times any frequency produced in the system [60], the cutoff frequency for the Butterworth filter was adjusted to 30 Hz. The cutoff frequency is defined as the frequency at which the signal output is 3 dB below that of the maximum output

in the bandpass frequency range. New resistor and capacitor values in the Butterworth filter are as follows:

$R_1 = 174\ k\Omega$ $C1 = 82\ nF$

$R_2 = 226\ k\Omega$ $C_2 = 10\ nF$

$R3 = R4 = 22\ k\Omega$

Substituting these values into equation **3** provides a cutoff frequency of about 33 Hz, as shown in the Bode diagram in **Figure 15**, where the maximum magnitude response is 6.02 dB at 1.12 Hz.

2.3.2 Hardware Implementation

2.3.2.1 Printed Circuit Boards (PCBs)

Two printed circuit boards (PCBs) provide a platform for passing the signals from the load cell to the SPU and mounting the signals' power, amplification, and filtering components. The first PCB, given in **Figure 16**, mounts directly onto the load cell to gather signals from the strain gages. This board provides access to the load cell that is independent of the conditioning circuit. The second PBC, given in **Figure 17** and **Figure 18**, mounts to the VCJ enclosure to condition the signal before it is sent to the SPU.

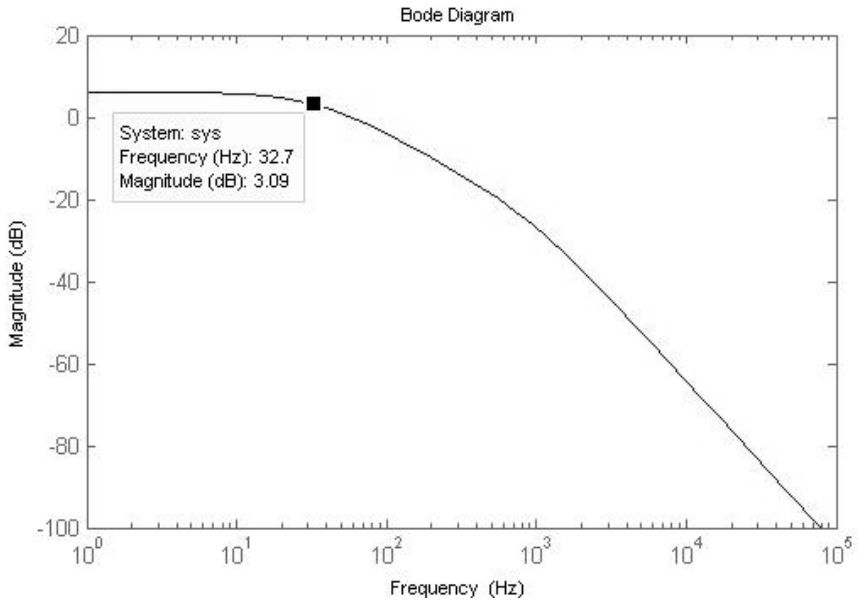


Figure 15: Bode plot for updated Butterworth filter shows a cutoff frequency of about 30 Hz.

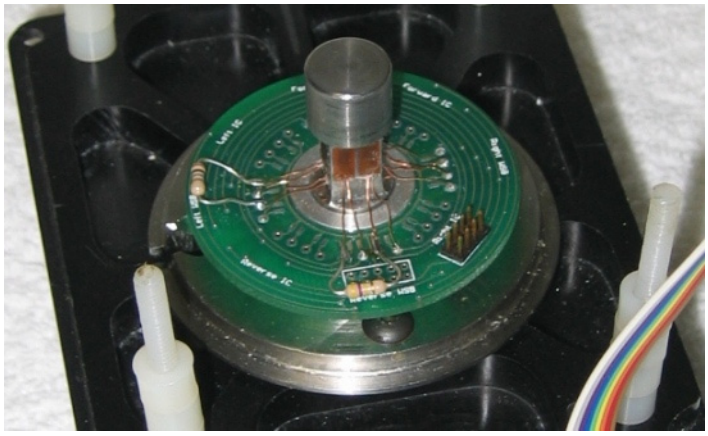
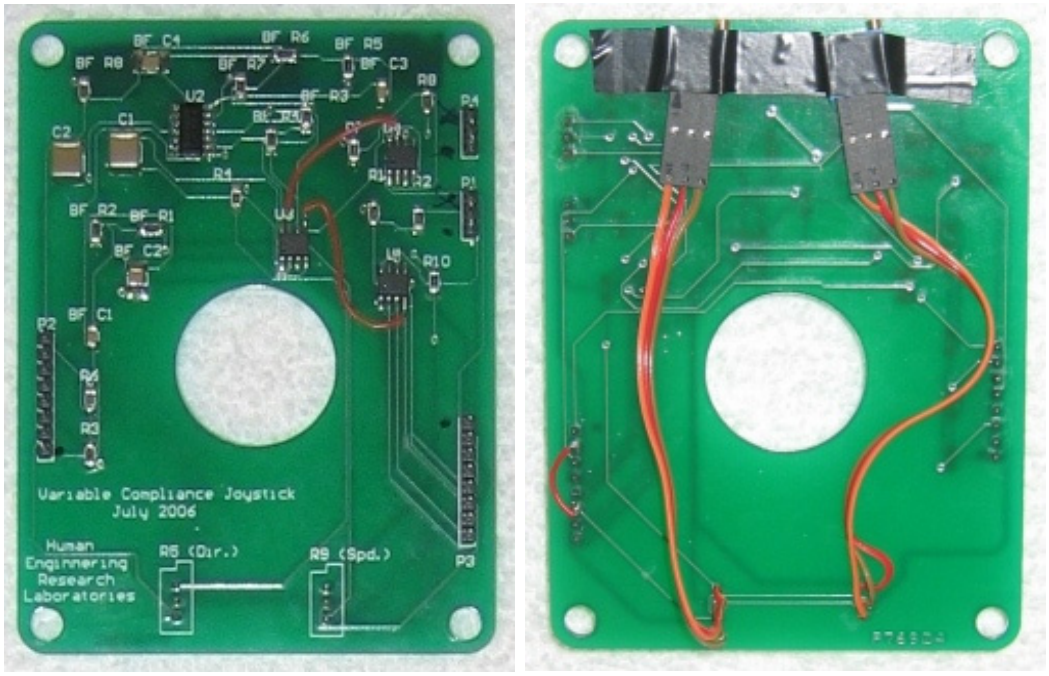


Figure 16: Photograph of the PCB that collects the signals from the strain gages.



(a) *(b)*

Figure 17: Photograph of the VCJ's PCB (a) top view and (b) bottom view.

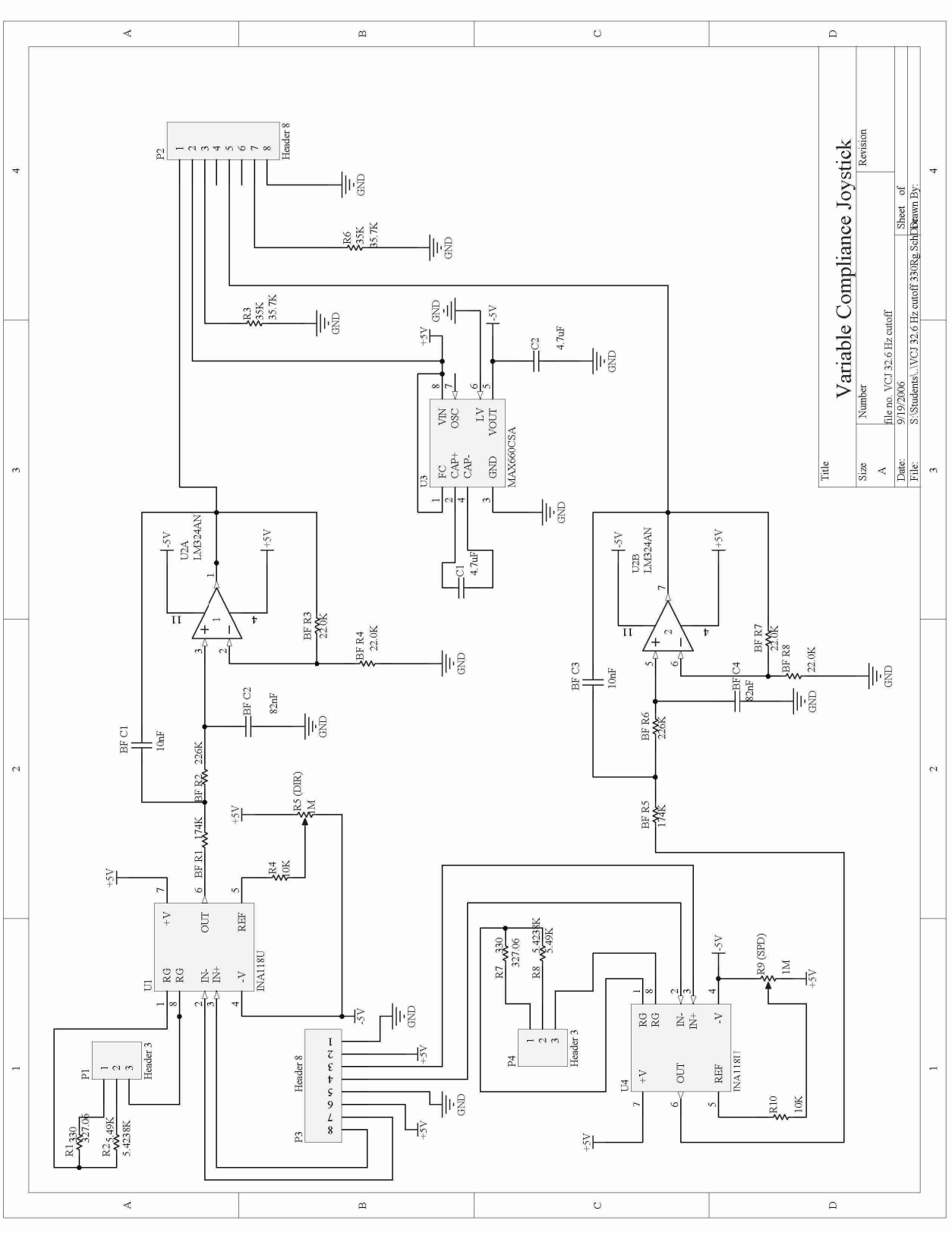


Figure 18: Schematic diagram of the VCJ's conditioning circuit.

P3 in **Figure 18** is the attachment point for the ribbon connecting the two PCBs together. P1 and P4 are the attachment points for the ribbon connecting to the SPDT switch. Pins 3 for P1 and P4 connect to pins 2 and 5 on the switch, pins 1 connect to pins 3 and 4 on the switch, and pins 2 connects to pins 1 and 6. Ideally, the pins on the potentiometers would slide into their respective holes on the PCB. However, since pin numbers were not specified in the circuit schematic, Altium Protel ® automatically generated its own pin order when its auto-route tool created the circuit layout on the PCB. The resulting order was not consistent with the order on the potentiometers. Therefore, an extra set of leads and ribbon were inserted to connect the PCB with the potentiometer.

2.3.2.2 Integrated Circuits (ICs)

Three different integrated circuits (ICs) support the functions of the conditioning circuit. Two INA118U ICs amplify the signals from the strain gages for the direction and speed channels. An LM324 quad operational amplifier IC is used to support both Butterworth filters. While the LM324 eliminates the need for multiple taps to the power supply, its maximum current rating is 50 mA. The current rating provided design parameters for the third IC, the MAX660CSA Complementary-symmetry/metal-oxide semiconductor (CMOS) monolithic voltage converter, which provides a -5 V supply to the circuit. To maintain an output current of approximately 50 mA, the MAX660 data sheet recommended external capacitor values of 4.7 to 10 μF. Low capacitor values can increase the total output source resistance (TOSR), however, which has the negative affect of increasing the voltage drop from the ideal value. Increasing the pump frequency mitigates the problem and is achieved by tying the V+ pin to the FC pin. This configuration multiplies the pump frequency eight-fold from the typical operating rate of 10 kHz. Capacitor values of 4.7 μF were ultimately chosen because the lower value ensures less output current and the output resistance does not appear to increase significantly based on the TOSR vs. Capacitance curve (**Figure 19**).

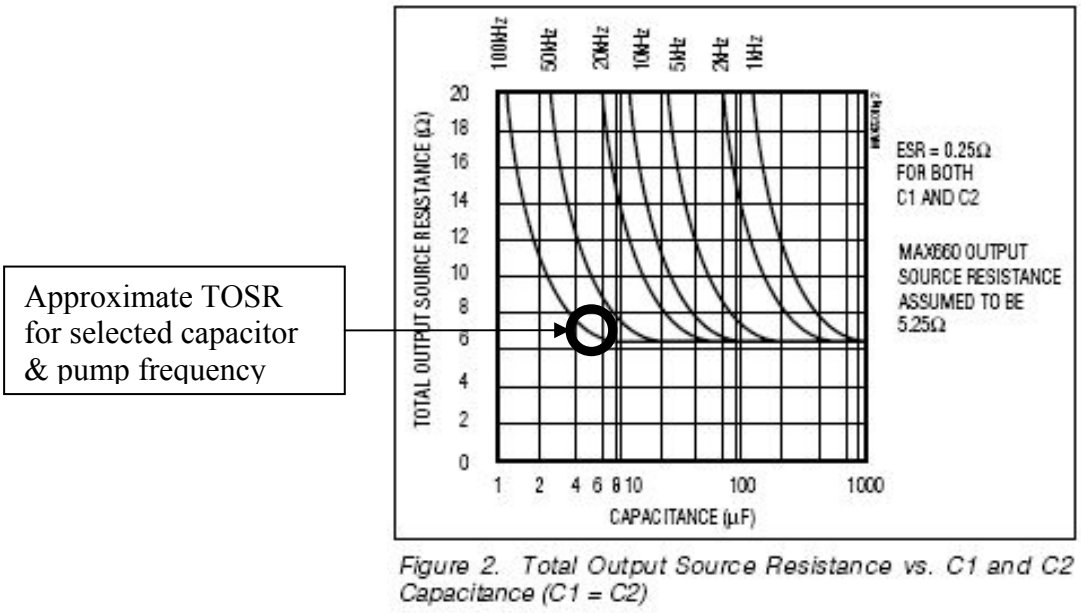


Figure 2. Total Output Source Resistance vs. C1 and C2 Capacitance (C1 = C2)

Figure 19: Total output source resistance for current configuration.

2.4 SIGNAL PROCESSING UNIT

After the analog signal leaves SCU, the SPU converts the signal to a digital value and applies the signal processing algorithms that personalize the joystick to the user. The key steps in the designing the SPU included selecting a platform, configuring the NI-DAQCard's electrical and software settings, and defining the basic and personalizable algorithms.

2.4.1 Selecting a Platform

The driving component for the SPU is the processor. Specifications for the processor determine the number and type of inputs and outputs, their resolution, the programming capacity, and the processing speed. We considered three options for interpreting and processing signals from the joystick post: the existing HERL IJ microcontroller, a Tattletale, or a laptop and DAQCard (a digital signal processor (DSP) framework was also briefly considered but felt to require too long of a development time). A brief description and features of each of the options are presented in Table 4.

Table 4: Key features of the three candidate solutions for the SPU processor.

	HERL IJ Microcontroller	Tattletale (Model 8v2)	Laptop and DAQCard
Description	The current processor is robust and adequately carries out calculations required for the VGA algorithm.	A Tattletale can be likened to a microcontroller with memory for programs and data storage included.	A Sony VAIO computer, Intel® Pentium® M, 1.20 GHz processor, operating on Microsoft Windows XP with an NI-DAQCard 6024E.
Clock	Clock needs to be added. Can do so with the DS1744	Clock included	Clock included
Programming Memory	32 KB with above clock	248 KB	+1 GB
Flash Memory	None	256 KB	0.99 GB
Digital I/O	None	Up to 25 lines	8 lines
A/D Conversion	4 Channels, 24 Bit, 0-5 V	8 Channels, 12 Bit, 100 kHz	16 single-ended or 8 differential inputs with S/W selectable gain, 12 Bit, up to 200 kHz
Output	3 to 9 V D/A Converter, 12 Bit	See Digital I/O	Two ±10 V channels, 12 Bit, plus Digital I/O
Output Format	Serial Port	Serial and Parallel Ports	PCMCIA Bus

The microcontroller on the HERL IJ and its supporting electronics would have required a substantial number of modifications to meet the study's requirements. Proposed modifications to the electronics included replacing the EPROM with a real time clock/calendar (RTC) and 32k non-volatile SRAM IC (Dallas Semiconductor's DS1744), adapting the D/A converter to provide for interfaces to joysticks with 2.5±1 V voltage swings, incorporating several layers of bank switching to increase the memory for data logging purposes, and purchasing a programmer for the clock. Furthermore, given that the microcontroller was already being operated near its maximum capacity, taxing it with more operations did not seem feasible.

The Tattletale was a good candidate and had been used with an isometric, chin joystick designed and built at HERL [5]. It includes its own microcontroller and clock, has sufficient memory for software and data logging, and has a serial interface to communicate with the PC running the VDS. A D/A converter and digital potentiometer would need to have been added to accommodate interfaces with MJs, and it does not include a Butterworth filter. While the Tattletale would probably have been sufficient with a few additions, it does not compare with the capabilities provided by a laptop and DAQCard.

The laptop and DAQCard configuration was the final choice because it allowed for reduced development time and reduced number of components while offering better performance than the alternatives. The Sony VAIO VGN-T370P laptop features a 1.2 GHz, Intel® Pentium® M processor and virtually unlimited memory for software and data logging. Its size is tiny compared to most laptops, weighing 3.0 lbs and measuring 1.2 in. by 8.0 in. by 10.5 in., giving it easy portability onto multiple EPWs. This solution would require an additional viewing screen for virtual driving, however, because the screen size of the laptop is very small, only 10.6 in. While the alternatives would need a serial port to communicate with the computer running the VDS, here the VDS and VCJ software may be executed on the same platform. While amplifiers and Butterworth filters would need to be added to collect signals from the strain gages, analog input and output and digital conversion is housed in one component, the NI-DAQCard 6024E. The NI-DAQCard 6024E supports up to 16 single-ended or 8 differential analog inputs with user-defined, bipolar input voltage ranges. Its maximum sampling rate is twice that of the Tattletale and provides similar resolution to the alternatives. The NI-DAQCard 6024E also provides two analog output signals spanning ±10 V with 12 bits of resolution. Thus, with the

data acquisition and processing platform selected, their specific configuration and the SPU software needed to be developed.

2.4.2 Data Acquisition

The NI-DAQCard 6024E offers excellent flexibility for acquiring information about the environment. It accepts analog input sources in multiple electrical configurations, the range of which is programmable. Sample rate, the rate at which a single channel is sampled, and scan rate, the rate at which all channels are sampled in one loop, are programmable, in addition to the order in which channels are scanned. And it allows the user to select the method for transferring data across the PCMCIA bus. The DAQCard provides a +5 V signal source for low power electronics. Guiding us through the DAQCard's electrical and software configurations were the design goals of keeping noise low while optimizing performance. A high-level view of the source code implementation is also presented.

2.4.2.1 Electrical Configuration

Support documentation for the DAQCard suggested three different electrical configurations for its analog inputs[1] as depicted in **Figure 20**: differential (DIFF), single-ended—ground referenced (RSE), or single-ended—nonreferenced (NRSE). Support documentation further recommended that the DIFF configuration be used if the input signal is less than 1 V to reduce noise pickup and increase common-mode noise rejection. In the case of DIFF configuration, the negative input should also be connected to ground through a resistor R_{ext} that is about 100 times the equivalent source impedance if the source impedance is greater than 100 Ω. This allows about the same amount of noise to couple on both positive and negative connections thereby putting the signal path nearly in balance. Thus, the R_{ext} value becomes 35 kΩ since the resistance across the strain gages is 350 Ω.

In practice, however, tying the negative input to the signal source did not appear to work well. First the negative input was tied to the reference signal from the instrument amplifiers on

[1] Technically, there are six configurations; but we did not consider ground-referenced signal sources because we could not assume that the joystick would be connected to the ground at all times.

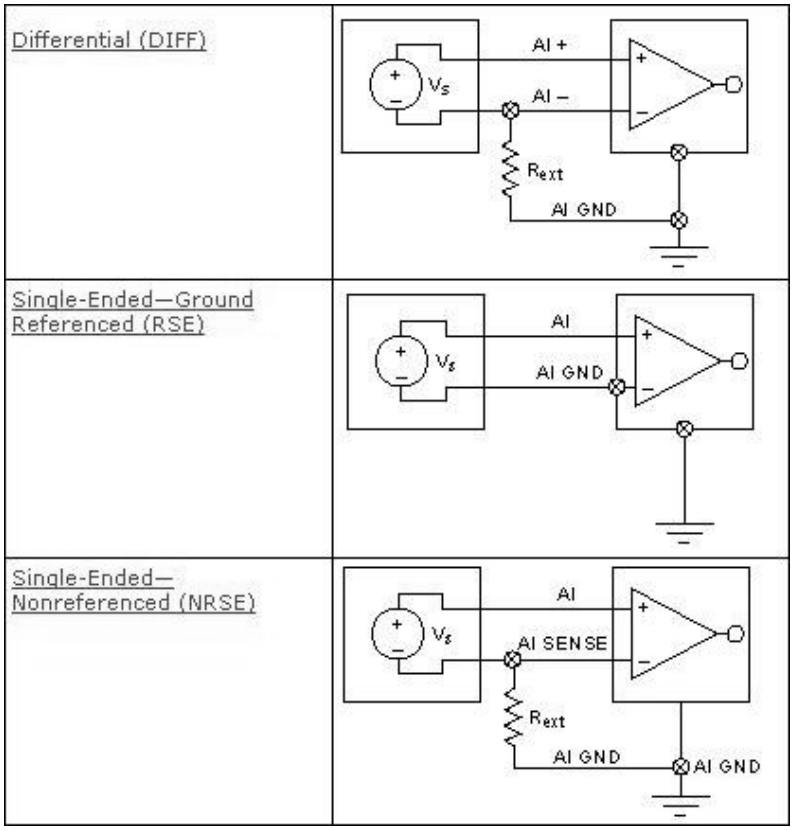


Figure 20: Recommended configurations for source signal.

the conditioning circuit and ground through R_{ext}, but there was too much noise. Second, the negative input of the DAQCard was tied to the negative side of the strain gage bridge, but this also produced too much noise. Instead, connecting the negative input only to ground through a resistor R_{ext} while still configuring the DAQCard to read in DIFF mode seemed to provide a steady signal. Variations in the readings were further improved with the number of samples and sample rate of the DAQCard.

2.4.2.2 Software Configuration

Though the NI-DAQCard can scan multiple channels at high rates and digitize them accurately, its performance is limited by the settling time of its internal programmable gain instrumentation amplifier (PGIA). The settling time is the time it takes the PGIA to amplify the signal to the desired accuracy before the DAQCard's A/D converter samples it, where longer settling times may lead to more error in the reading. Factors that increase the settling time include a high source impedance, poor quality cabling connecting the source, suboptimal channel scanning

order, and a high scanning rate. Mitigating settling time error may be achieved through increasing the sample size. To optimize the settling time with channel scan order, channels should be scanned such that the difference between the signals is minimized. In the case of the VCJ, which has only two input channels whose signals may or may not be close, accuracy could be improved by sampling one channel 50 times and then sampling the other channel 50 times, for example, instead of interleaving the two channels. But, this would cause the two readings to have different time values and therefore was not used. Increasing the scanning rate has the benefit of increasing the number of samples, but it also may decrease the performance. Therefore, several different configurations were examined to see which offered the best performance. Before describing this analysis, however, a brief moment needs to be taken to compare scan rate with sample rate and sample with reading.

The scan rate is the rate all channels are scanned in one loop, while the sample rate is the rate at which a single channel is sampled within a scan sequence. Ideally, the scan rate is equal to the sample rate divided by the number of channels; but the PGIA for the NI-DAQCard 6024E requires a minimum delay of about 5 μs between scans. Rather than requiring the user to calculate the best scan rate for a given sample rate, the DAQCard's driver will automatically calculate the best scan rate for a given sample rate if 0 is entered in its place. A sample is a discrete measurement by the DAQCard, and a reading is the average of all the samples acquired during a data acquisition operation for a given channel.

A variety of configurations for sample rate and sample size were examined to see which offered the best performance both in reduced standard error and reading period. The configurations consisted of a range of sample rates from 10 kS/s to 100 kS/s with sample sizes chosen such that the estimated reading period would be less than 2 ms. The estimated reading period, *dt*, is

$$11: \qquad dt = \frac{n_{samples} \times n_{channels}}{R_{sample}},$$

where $n_{samples}$ is the number of samples per channel, $n_{channels}$ is the number of channels, and R_{sample} is the sample rate. For example, if we would like to sample at 50 kS/s for 2 channels, the maximum number of samples we can acquire in 2 ms would be 50. To compare the various configurations, 1000 readings were acquired with no force input on the joystick three times. The average of the three trials' standard deviations in the speed and direction channels were then

compared graphically. The standard deviation of the readings offers a measure of the variance, or noise, in a constant signal while not being heavily influenced by outliers, as would be the case if comparing ranges. The average time to acquire a reading, or the actual reading period, was also examined.

Results from the analysis, depicted in **Figure 21**, confirm the DAQCard's support documentation that low sample size and high sample rates increase variance. Trials with 10 samples had very high standard deviations even though the sample rate was low. When the sample rate was consistent but the sample size varied, configurations with higher sample sizes had lower standard deviations but longer reading periods. While increasing the sample rate allowed for more samples, the standard deviation increased when the sample rate grew above 75 kS/s. Therefore, since a sample rate below 75 kS/s offered slightly worse performance even with the same sample size, we selected 75 kS/s as the sample rate. We selected 40 samples for the sample size because it offers comparable standard deviation with a better reading period than 50 samples.

It is not entirely clear why the direction channel consistently has larger standard deviations than the speed channel. It may have been a result of the scanning order; the direction

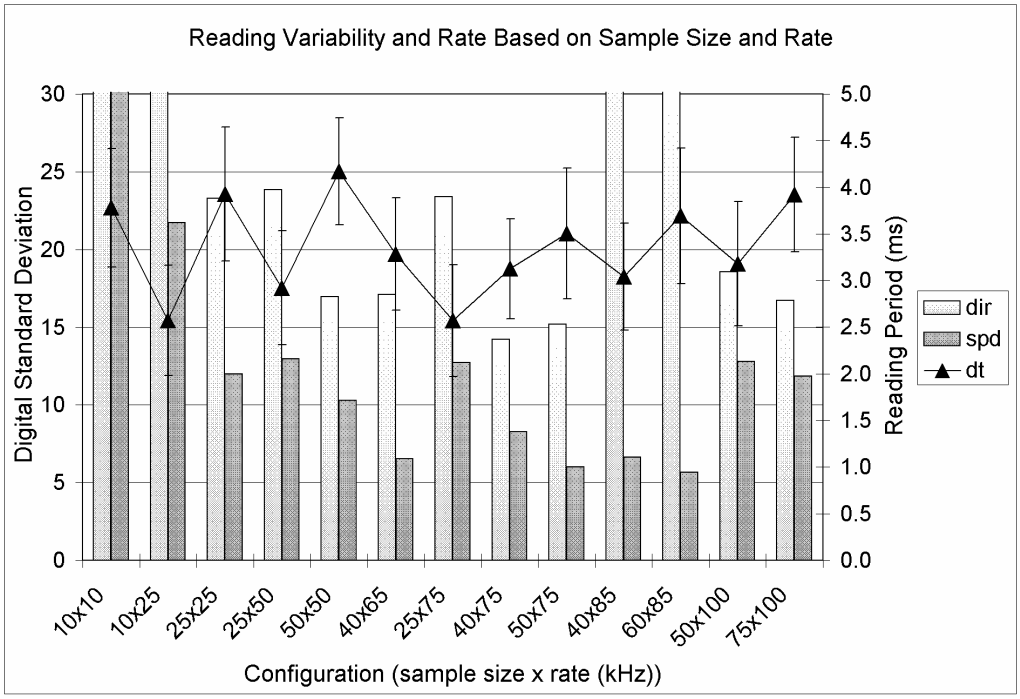


Figure 21: Comparison of various data acquisition sample rate and size configurations.

channel is always scanned first. But, when the scanning order was reversed, the direction channel still had the larger variance. The actual reading period is also 50% larger than the design goal. This is likely a result of the reading being obtained during a synchronous operation. Each time a reading is requested, the DAQCard needs to configure itself for the new scan; and the data needs to be transferred across the PCMCIA bus. The estimated reading time also does not take into account the minimum delay between scans.

The input range of ±0.5 V for the DAQCard provides maximum flexibility for source signals. Input ranges available to the user include ±0.05 V, ±0.5 V, ±5.0 V, and ±10.0 V. While a range of ±0.05 V would have been ideal for a strain gage bridge configuration, we had also considered letting a conventional MJ (e.g. FlighLinks JC200) act as a signal source. The FlightLinks JC200 has an output signal range based on its supply voltage V_s. Specifically, its voltage swing is ±10% V_s [62]. Since the DAQCard supplies +5 V and if the JC200's reference signal is connected to the DAQCard's negative input, the voltage swing from the JC200 becomes ±0.5 V. Once a reading has been collected from the DAQCard, the SPU transforms the data to suit the needs of the user.

2.4.2.3 Source Code Implementation

The software to configure and collect readings from the VCJ's conditioning circuit was written in C++ in the Microsoft Visual Studio .NET environment. While NI's most recent application programming interface (API) is the NI-DAQmx driver, support documentation was not clear regarding the development of applications in Visual Studio with the NI-DAQmx when APIs were being investigated. NI's Traditional NI-DAQ (Legacy) API was chosen because it did offer data acquisition examples for Visual Studio and supported the NI-DAQCard 6024E.

Four functions were written to configure the NI-DAQCard and collect readings from the VCJ's conditioning circuit. The reader is referred to Appendix E for their source code implementation. `DAQ_initialize()` configures the DAQCard's input mode and sets the timeout limit if there is an error. Because an offset may exist on each axis, `DAQ_find_offset2()` acquires several readings at once for both channels and takes an average. The resulting values are subtracted from susbsequent read operations. `DAQ_get_data()` sets up the read operation for each channel, collects interleaved x- and y-axis data from the conditioning

circuit, and takes averages for both channels. Lastly, `DAQ_close_device()` resets the DAQCard to its original configuration.

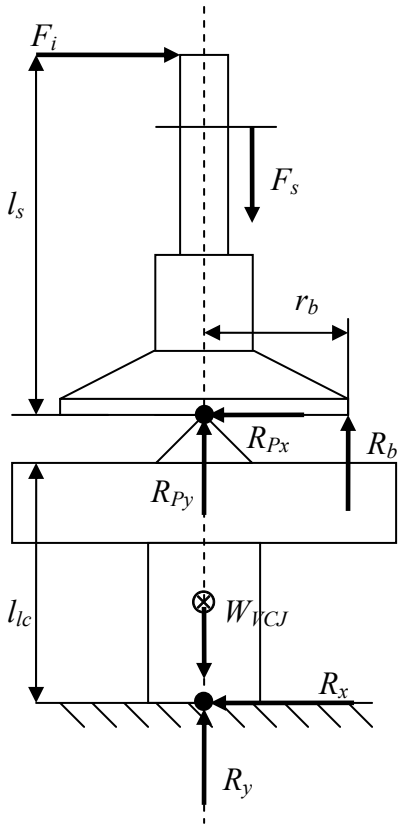
2.4.3 Input Control Algorithms

The SPU contains four algorithms for processing the user's input:

- *The Standard Transfer Function* (STF) is intended to behave similarly to the signal processing on a conventional MJ. But since the joystick is built upon a force transducer, the algorithm applies a dead zone and template. It also applies a gain to interface with the VDS's expected input range.
- *The Variable Gain Algorithm* (VGA) has been modified slightly since its development in [67], but the core concept of mimicking the response of a MJ with an IJ is the same. It differs from the STF only in parameters for the dead zone, gain, and templates.
- *The Multiple Sclerosis – Personally Fitted Algorithm* (MS_PFA) incorporates a tremor filter, bias axes, a dead zone, independent gains for each axis, and adjustable template sizes and shapes.
- *The Multiple Sclerosis – Personally Fitted Algorithm with Fatigue Adaptation* (MS_PFA_FA) includes the same features as the MS_PFA in addition to a gain schedule based upon the user's activity and personal characteristics.

2.4.3.1 The Standard Transfer Function (STF)

The STF features a dead zone, template, and gain. The dead zone is required because the VCJ's joystick post will output a signal even if the input force is within the range of a typical MJ's dead zone. The free body diagram in **Figure 22** shows that the stick will not pivot if the pretension in the spring is large enough. Furthermore, the free body diagram illustrates that there is no external horizontal force to act against the input force other than the reaction force at the base of the joystick. Therefore, the force will be transmitted through the strain gages. It follows, then, that since there is no external reaction force other than at the base, the signal will increase proportionally even if the input force is beyond the template force for a conventional MJ,



Key
F_i – force input
l_s – length of the stick
F_s – force of spring along axis
R_b – reaction at boot
r_b – radius of boot
R_{px} – reaction at pivot, x-direction
R_{Py} – reaction at pivot, y-direction
R_x – reaction at base, x-direction
R_y – reaction at base, y-direction
R_M – reaction moment at base
l_{lc} – length of the load cell
W_{VCJ} – weight of VCJ

Sum of the forces above the pivot
$\Sigma M_p = -F_i \times l_s + R_b \times r_b$
$\Sigma F_y = -F_s + R_b = 0 \Rightarrow F_s = R_b$ (see note)
$\Sigma F_x = F_i - R_{Px} = 0 \Rightarrow F_i = R_{Px}$
Sum of the forces below the pivot
$\Sigma M_O = -R_{Px} \times l_{lc} - R_b \times r_b$
$\Sigma F_y = -R_{Py} + R_y - W_{VCJ} = 0$
$\Sigma F_x = R_{Px} - R_x = 0$
Substitutions
$\Sigma M_p = -F_i \times l_s + F_s \times r_b$
$\Sigma F_x = R_x = F_i$
Conclusions
If $F_s > (F_i \times l_s) / r_b$, the resulting moment is counter-balanced by the opposite side of the boot, and the stick does not rotate.
Since $R_x = F_i$, the strain gages will sense a force.

Note: The boot and stick are different masses. Therefore, $R_b = R_{Py}$.

Figure 22: Free body diagram of the VCJ demonstrating that a dead zone force will be sensed, even though the stick will not move. Friction is ignored.

necessitating a software template. The primary purpose of the gain is to translate the force input signal from the DAQCard to the appropriate range for the VDS.

The dead zone and template parameters were chosen based on the best fit force vs. angle curves for the VCJ with two different handles. For handles 3 and 4, which were chosen to represent the two different classes of small and large handles, the y-intercepts were at 0.8821 N and 1.1539 N, respectively. The 18°-intercepts were at 4.176 N and 3.788 N, respectively. The average values for the two handles' dead zones and templates were used to allow the software to be independent of the handle. Therefore, the dead zone force is 1.018 N and the template force is 3.982 N.

The flowchart for the STF is depicted in **Figure 23**. The inputs are converted to force first because the force to digital reading ratio in not consistent for the four directions for the VCJ, as described in section 3.1.1. For example, while mechanically about 3.98 N will deflect the joystick to 18°, the digital values for 3.98 N in the forward and reverse directions are 397 and 326, respectively. Using dead zone and template parameters in units of force allows there to be one set of numbers for all directions. Next, if the input is within the dead zone, the input is set to zero, and the dead zone flag is set. If the input is outside of the dead zone, the dead zone force is subtracted from the input to ensure a smooth transition out of the dead zone. The dead zone force is also subtracted from the template force because of the preceding linear translation applied to the inputs. The algorithm then checks to see if the input is outside the template. If so, the inputs are set to the corresponding template value, and the template flag is set. Lastly, the gain is applied to the inputs to convert the force to the expected digital input from the VDS. The gain was computed such that an input force that corresponds to the template force would result in the maximum speed of the wheelchair, or

$$12: \qquad Gain = \frac{V_{\max-dig}}{F_{template} - F_{dead-zone}},$$

where $V_{max\text{-}dig}$ is the maximum wheelchair velocity in digital units, $F_{template}$ is the template force, and $F_{dead\text{-}zone}$ is the dead zone force. Therefore,

$$13: \qquad Gain = \frac{2048}{3.982 - 1.018} = 691.0.$$

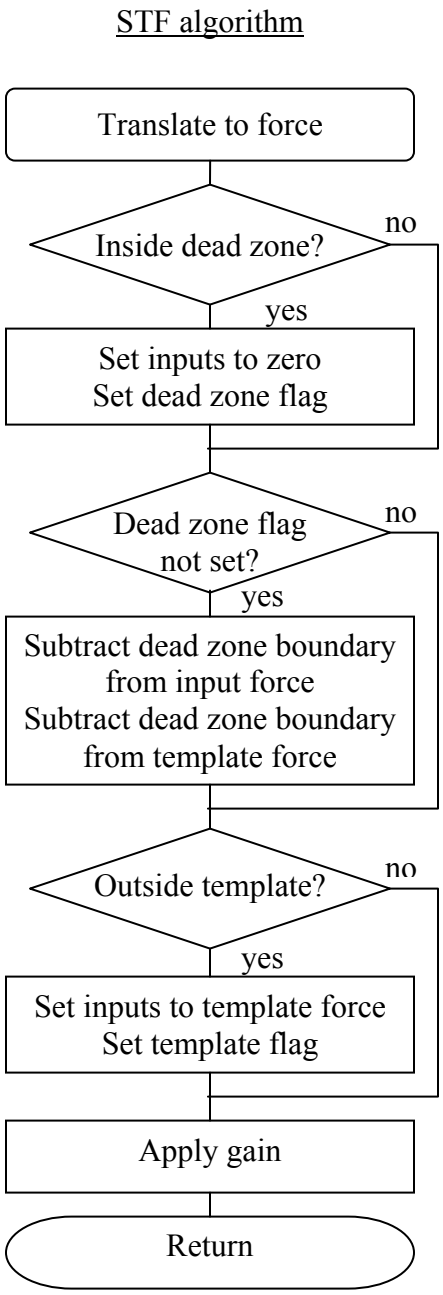


Figure 23: Flowchart for the standard transfer function (STF) and variable gain algorithm (VGA).

2.4.3.2 The Variable Gain Algorithm (VGA)

While the purpose of the VGA remains the same as originally conceived, it differs from the original implementation in its computation method, organization, and working units. With the faster speed permitted by the laptop's processor, computations for trigonometric and exponential functions can be performed while the joystick is operating instead of referencing look-up tables. The organization has also been altered to improve readability. The VGA follows the same flowchart as shown in **Figure 23**. Lastly, since the force to digital signal ratio is not consistent in the four directions, comparisons for dead zone crossings and template violations are performed in units of force instead of the digital representation.

To determine the force equivalencies for the dead zone and template, an equation was derived from the force to signal and signal to digital relationships for the HERL IJ. A linear regression of the force, F_i, to signal, S, data reveals the following relationship: $S = 0.10613F_i + 0.03194$ or

$$14: \qquad F_i = \frac{S - 0.03194}{0.10613}.$$

The signal to digital reading, *dig*, is derived from an input range of +3 V to +6 V mapping to a digital range of 0 to 4096 or

$$15: \qquad S = \left(\frac{3}{2048} dig + 3\right) - 6.$$

Substituting equation 15 into 14 results in

$$16: \qquad F_i = \frac{\left(\frac{3}{2048} dig - 3\right) - 0.03196}{0.10613}.$$

Digital values for the VGA dead zone and template were 2150 and 2550, respectively. Therefore, using equation 16 their equivalent force values are 1.107 N and 6.628 N, respectively.

2.4.3.3 The MS Personally Fitted Algorithm (MS_PFA)

The MS_PFA applies a number of transformations to the input signal including a tremor filter, bias axes, dead zone, independent gains for each axis, and adjustable template sizes and shapes. The flowchart for the MS_PFA is depicted in **Figure 24**. It is important that the tremor filter be applied first because the dead zone or template would clip the data. While we had considered

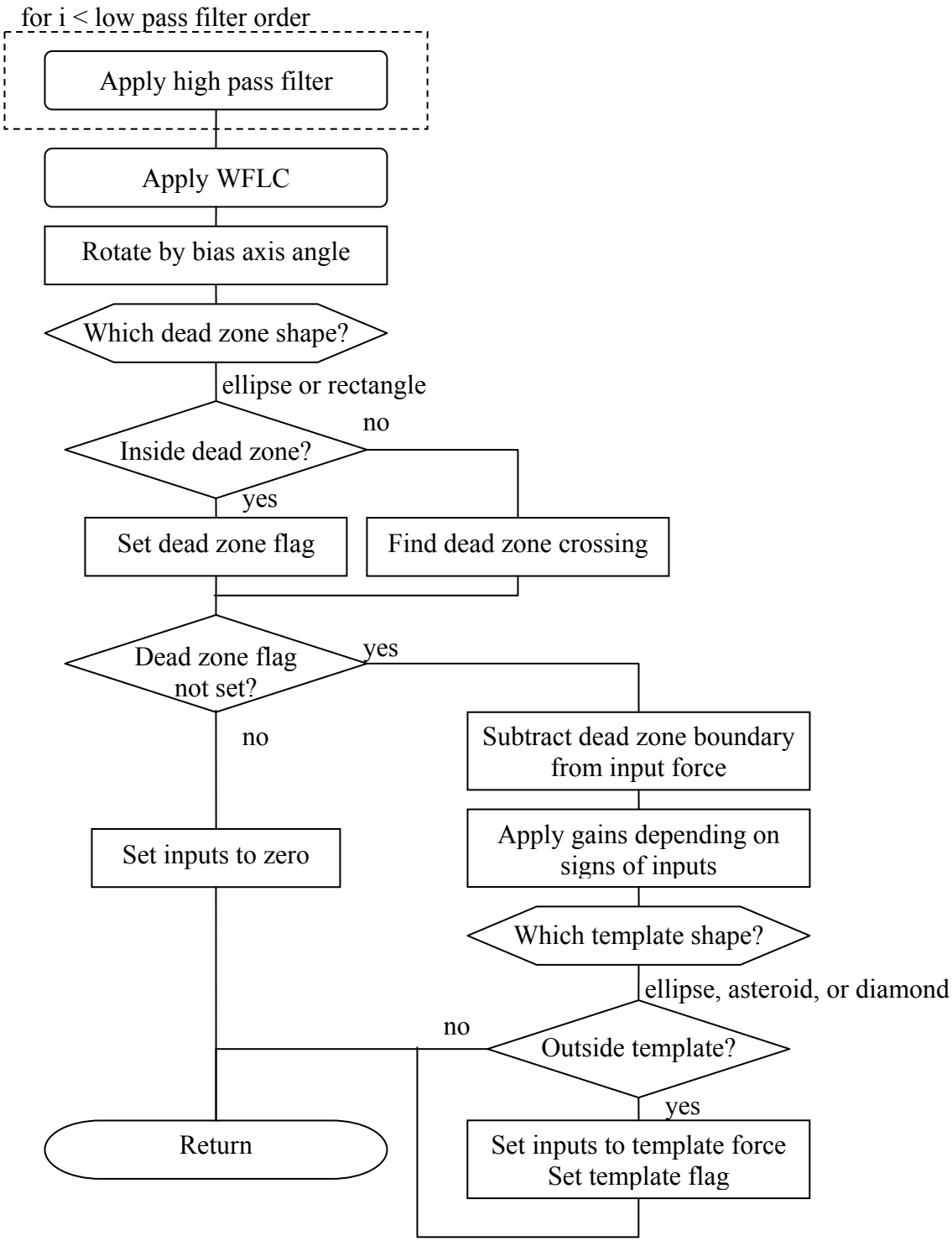


Figure 24: Simplified flowchart for the MS_PFA algorithm.

using a simple low-pass filter to mitigate tremor, a WFLC filter had been shown to offer superior performance compared to a low-pass filter in driving a virtual wheelchair [47]. While the modified adaptive filter Nho developed performed similarly to the original WFLC, the adaptive filter as originally developed is used here because the extra complexity did not seem necessary. To prevent the WFLC from removing the intentional movements that typically occur at less than 2 Hz, a second order infinite impulse response (IIR) high-pass filter executes before the WFLC. Rotating the input by the bias angle also needs to occur early in case the dead zone or template is not circular.

The algorithm determines dead zone crossings and template violations based on the shape of the dead zone and template, respectively. The algorithm for processing dead zone crossings and template violations originated as part of tuning software developed by Ding, Cooper, and Spaeth [77]. Dead zone shapes include circular, elliptical, and rectangular; and template shapes include circular, elliptical, diamond, and asteroid. In their implementation, however, circular shapes are a subset of elliptical shapes.

Gains are applied depending along which axes the force is applied. The input does not need to be translated into units of force because the default gains are calculated based on the raw digital input during joystick tuning.

2.4.3.4 The MS Personally Fitted Algorithm with Fatigue Adaptation (MS_PFA_FA)

In addition to providing custom settings to the user, the purpose of the MS_PFA_FA is to provide a joystick interface that tunes itself to the user as he or she becomes fatigued. As fatigue sets in, it becomes increasingly more difficult to apply the force necessary for maintaining a desired velocity. The MS_PFA_FA will increase the gain gradually thus reducing the amount of force needed to produce the desired output. Likewise, the gain will decrease back to the original setting if the joystick is not in use. Specifically, the gain is governed by equation 17

$$17: \qquad K = \begin{cases} K_1 = K_{\max} - (K_{\max} - K_2)e^{-\alpha T} & F > F_{dead-zone} \\ K_2 = K_{\min} + (K_1 - K_{\min})e^{-\beta T} & F \le F_{dead-zone} \end{cases},$$

where K is the current gain, K_1 and K_2 are the last updated gains while in adaptation or recovery modes, $K_{\max}$ is the maximum possible gain for the user, $K_{\min}$ is the default gain for the user, α is the fatigue adaptation parameter that governs the rate at which the gain increases, β is the fatigue recovery parameter that governs the rate at which the default gain is regained, T is the amount of

time in ms elapsed since F crossed $F_{dead\text{-}zone}$, F is the user's input force, and $F_{dead\text{-}zone}$ is the force required to cross the dead zone boundary. To demonstrate how the gain varies with time, a simulation was executed in MATLAB (The Mathworks, Inc, Natick, MA) where the input force was switched in and out of the dead zone. The results are depicted in **Figure 25**, where $K_{\min}$ and $K_{\max}$ are 1.0 and 3.0, respectively, and α and β are 1.0e-5 and 1.5e-5, respectively.

The flowcharts for the MS_PFA and MS_PFA_FA are similar with the exception that the Apply gain box in **Figure 24** is replaced with the flowchart in **Figure 26**. Since the function governing the gain depends on whether or not the input force is within the dead zone, a mode variable is set first depending on the status of the dead zone flag. If the new mode does not equal the previous mode, the start time for the current mode is saved, the previous mode is updated to the new mode, and the last gains are saved in the K_1 and K_2 variables. Next, if the current force is within the dead zone – that is, the function is in gain recovery mode – the second part of equation 17 is used. Otherwise, the first part of equation 17 is used. Lastly, the gains are applied if the input is outside of the dead zone.

2.4.3.5 Implementing the Algorithms

The reader is referred to Appendix F for the algorithms' source code implementation. Because of the algorithms' similarities, the STF and VGA algorithms were combined into one routine, `ICD_VGA`; and the MS_PFA and MS_PFA_FA algorithms were combined into another routine,

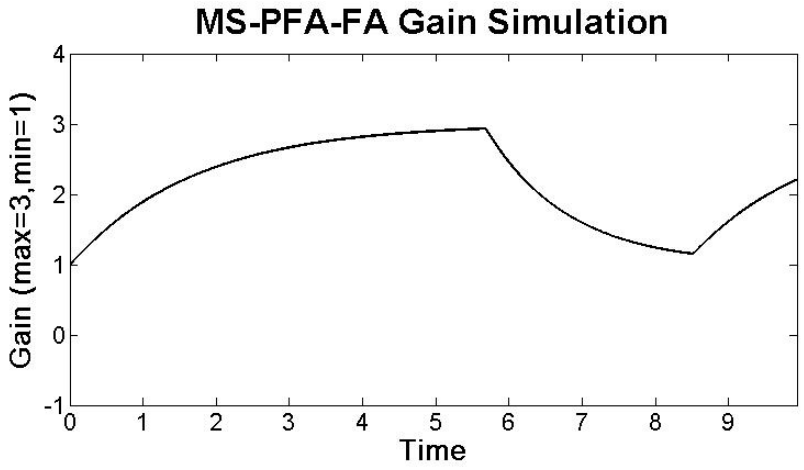


Figure 25: Simulation demonstrating how the gain varies over time based on minimum and maximum gains and adaptation and recovery rates.

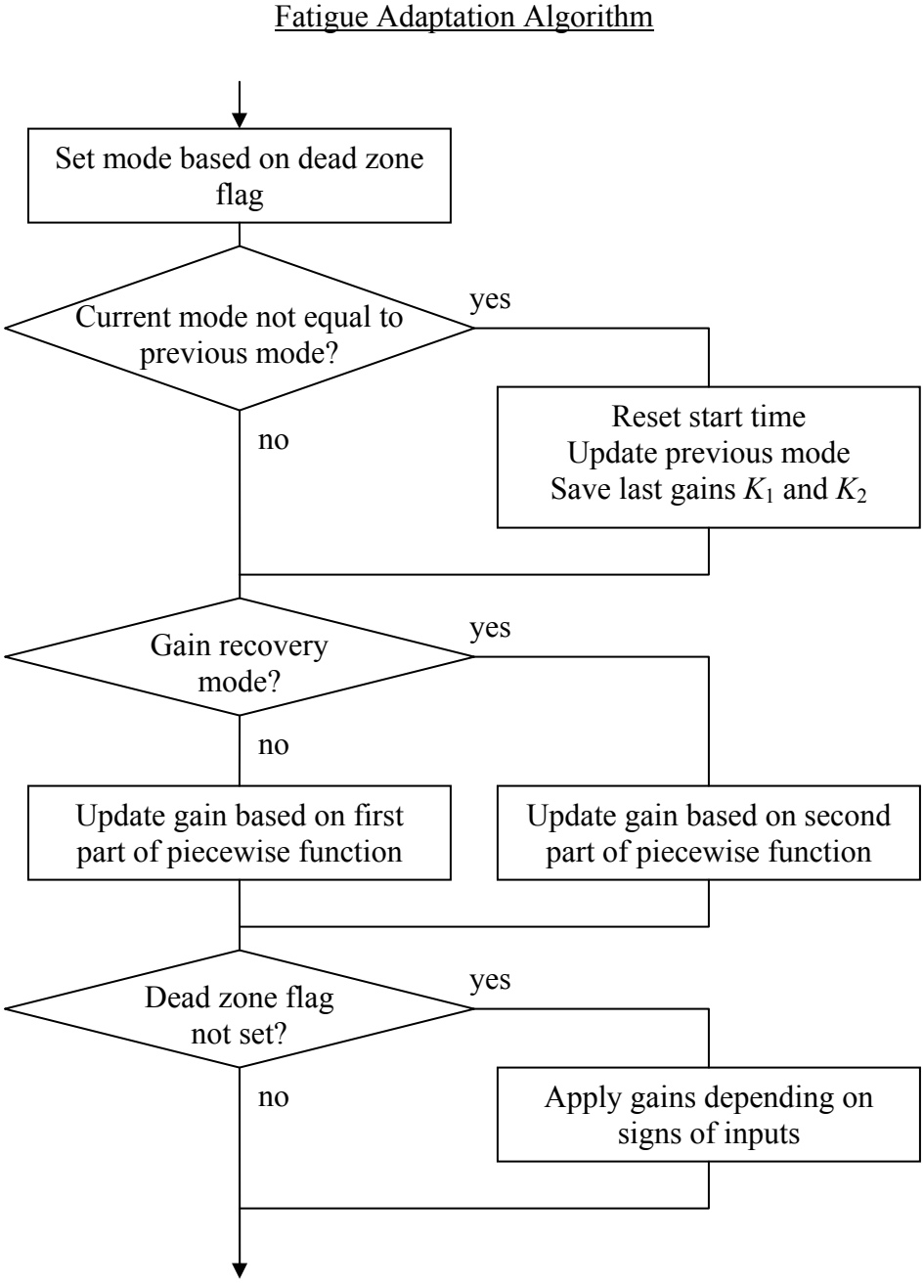


Figure 26: Flowchart for fatigue adaptation algorithm (not including tremor filter and personalization features).

ICD_MS_PFA. Logic at the beginning of ICD_VGA sets the dead zone, gain, and template parameters depending on which algorithm is specified with the do_vga input parameter. Likewise, logic before the point where gain is applied checks the do_fatigue input parameter before applying gain in ICD_MS_PFA. Helper functions were also written to load the custom settings from the setup files (Getsettings(), which are described in sections 4.3.1.5 and 4.3.2.2), initialize parameters (MS_PFA_initialize(), ICD_calibration_constants(), and FA_load_defaults()), convert digital readings to force (ICD_dig2force()), protect against software crashes (FA_load_last_saved()), and interpolate missing data points (neville()).

3.0 VALIDATION OF THE VCJ

A potential limitation to ensuring that the input control algorithms presented in this thesis can improve driving in any wheelchair configuration is the novelty of the VCJ. Conventional MJs use different sensing methods and thus different transducers to collect user input, and they generally use a pivot mechanism with pins rather than a swivel bearing. Therefore, to prevent the novelty of the VCJ from confounding the performance of the algorithms, the VCJ needed to behave similarly to a conventional MJ. An additional feature of the VCJ, nonetheless, is that its compliance level can span a wide range of settings. The influence of different springs and different pretensions on the springs is not totally understood, especially in a dynamic environment. Therefore, validation of the VCJ consisted of ensuring that it could mimic the static and dynamic responses of a conventional MJ and characterizing its behavior analytically and with a variety of spring sizes and pretensions. First the static responses of the VCJ in isometric and compliant modes will be discussed followed by its dynamic response in compliant mode.

3.1 STATIC RESPONSE

The purposes of validating and characterizing the static response of the VCJ are to ensure that the VCJ provides the expected digital values for given force inputs in both high and low gain modes, to ensure that the force to deflection response is consistent with a standard MJ, and to provide insight into the influence of different springs and their pretensions on the force to deflection response. Therefore several experiments in isometric and compliant modes were performed with the VCJ, where known forces were applied to the joystick. Characterizing the joystick in isometric mode provides insight into the exact response of the VCJ. Based on a simple beam bending model, known force inputs can be applied and compared with the

measured result. If there is deviation from the model, regression analysis may be used to determine the actual input to output relationship. Characterizing the joystick in compliant mode provides the opportunity to compare the force response of the VCJ with that of a standard MJ. The influence of different springs and their pretensions may also be ascertained.

3.1.1 Isometric Mode

3.1.1.1 Test Procedures and Setup

The procedure for applying known loads to the joystick included the following steps:

1. For the respective modes of VCJ operation, place the force meter and jig in the quill of a computer numerically controlled (CNC) mill and mount the VCJ to the milling table. In high gain mode use the Ameteck AccuForce II (Sellersville, PA) force gage, and in low gain mode use the GSE 355 (Allen Park, MI) force gage. The Amtech AccuForce II force gage has a precision of about 0.01 N up to 50.0 N, and the GSE 355 force gage has a precision of 0.5 lbs up to 200 lbs.
2. Apply a set of increasing forces perpendicular to the tip of the joystick along the positive y-axis by moving the VCJ into the force gage probe with the mill table. In high gain mode, force increments should be about 1 N when the force is below 8 N and about 2 N when the force is above 8 N. Bifurcating the step size allows for more sensitivity in the driving range of operation. In low gain mode, force increments should be about 9 lbs until just before the force reaches the limit of the inputs, when force increments should be between 3 and 5 lbs. Here, bifurcation improves the estimation for where the DAQCard no longer measures increases in force.
3. Record and average the digital reading from the DAQCard for 10 s. Record the input force from the respective force gage.
4. Repeat the above steps for the negative y-, positive x-, and negative x-axes.

Instead of using the conventional handle at the tip of each joystick post, the mill applied forces to a small, aluminum cuff that had a flat contact pad machined into it. The center of pressure was located at the center of the machined pad. The same jig described in section 3.1.2 was used in high gain mode. However, the lock at the pivot would not hold when forces exceeded about 35

lbs. Therefore, a new jig without a pivot was designed to provide an interface between the mill and the GSE 355 force gage.

3.1.1.2 Analytical Model

Equations 6 through 9 served as the basis for the analytical model of the VCJ in isometric mode. Furthermore, since the DAQCard is bipolar and operates with 12 bits of resolution, an input of 0.5 V results in a digital reading of 2048. Or,

$$18: \qquad dig = \frac{2048}{0.5\ V} JP_{out} = 4096 \times JP_{out}$$

Solving for the digital reading in terms of the VCJ's fundamental components, therefore, yields

$$19: \qquad dig = 4096 \left(\frac{50000}{R_G} + 1 \right) \frac{E_i F \dfrac{6\,PL}{Ebt^2} + \varepsilon_{bridge}}{\left(\dfrac{R_3}{R_3 + R_4} \right)}.$$

With the exception of the length because of the addition of the aluminum cuff, the same parameters as in Table 3 were used. The new lengths for the high gain and low gain modes were 4.20 in. and 4.989 in., respectively.

3.1.1.3 Results

The results for the experiments in high gain and low gain modes are presented in **Figure 27** and **Figure 28**, respectively. The experimental measurements are consistently smaller in magnitude than the analytical model causing the cutoff force to be much lower than expected. Therefore, the experiment for low gain mode was repeated but with a new gain resistor of 330 Ω, the results for which are presented in **Figure 29**. The approximate cutoff forces are presented in Table 5.

Because the experimental data did not match the analytical model well, best fit curves were determined for each axis and gain resistor. It was necessary to determine the best fit curve for each axis because of their different responses. Left and reverse inputs (when the force was along the negative axis) resulted in smaller signals than right and forward inputs (when the force was along the positive axis). Results are provided in **Figure 30** through **Figure 33**.

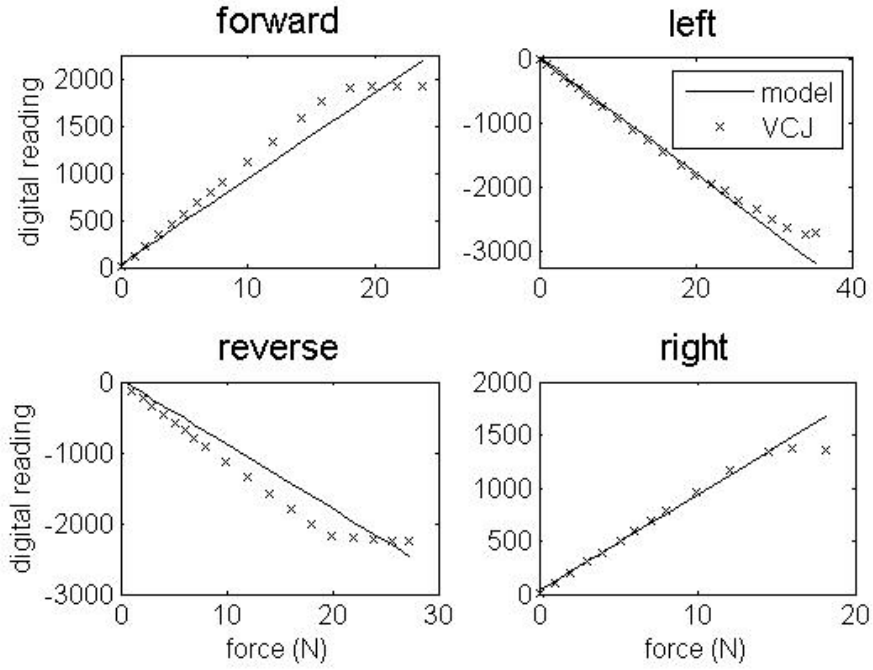


Figure 27: Force response of VCJ in high gain mode with the designed gain resistor in the four directions.

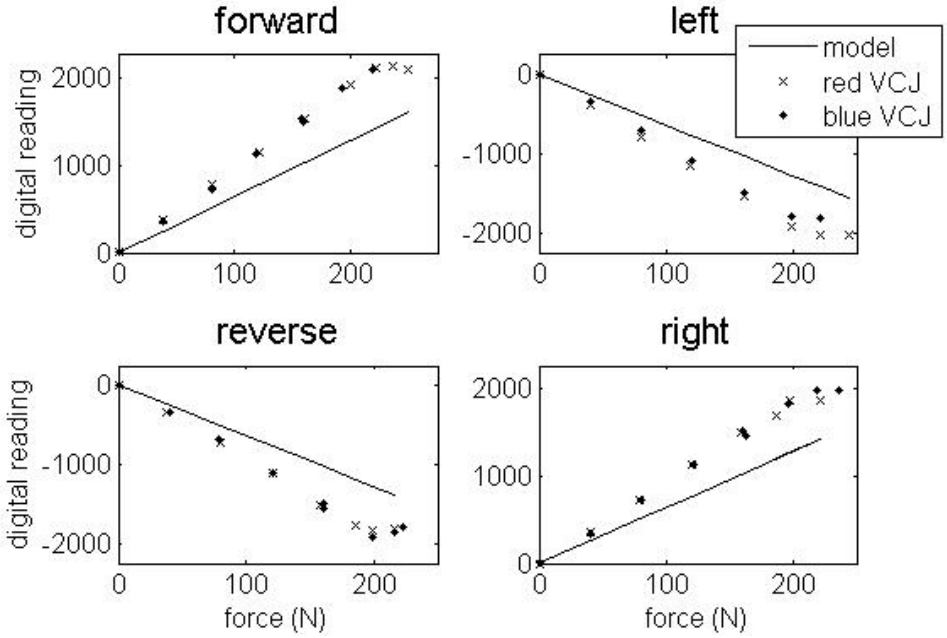


Figure 28: Force response of VCJ in low gain mode with the designed gain resistor in the four directions. Results from both circuit boards are presented.

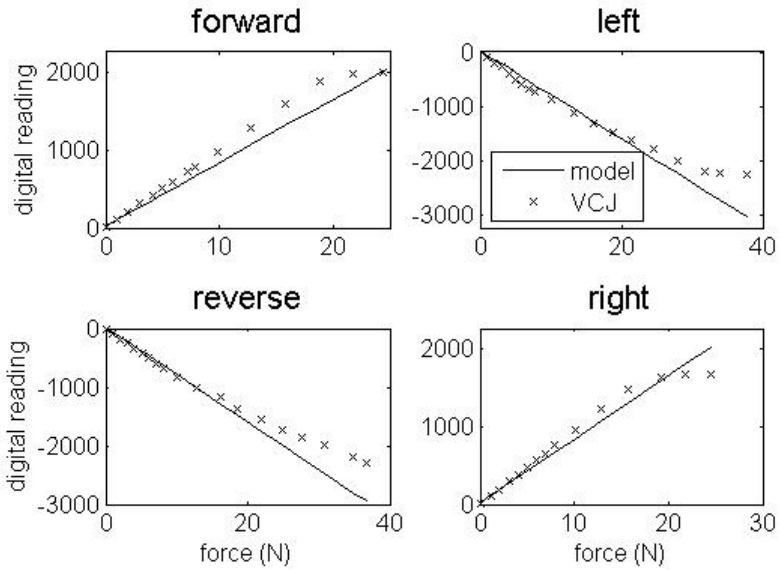


Figure 29: Force response of VCJ in high gain mode with the new gain resistor in the four directions.

Table 5: Cutoff forces based on gain resister and direction. Rg = 292 Ω for High Gain 1, 330 Ω for High Gain 2, and 5.42 kΩ for Low Gain.

	High Gain 1 (N)	**High Gain 2 (N)**	**Low Gain (N) [red; blue]**
Forward	18	19.5	219; 210
Reverse	>36	36.5	199; 199
Left	34	32	199; 197
Right	14.5	18	219; 210

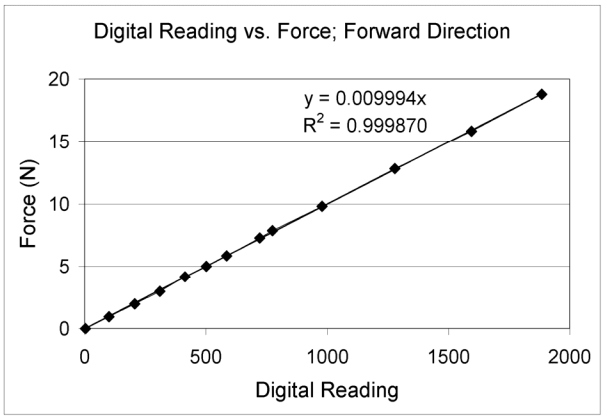


Figure 30: Forward direction best fit curve.

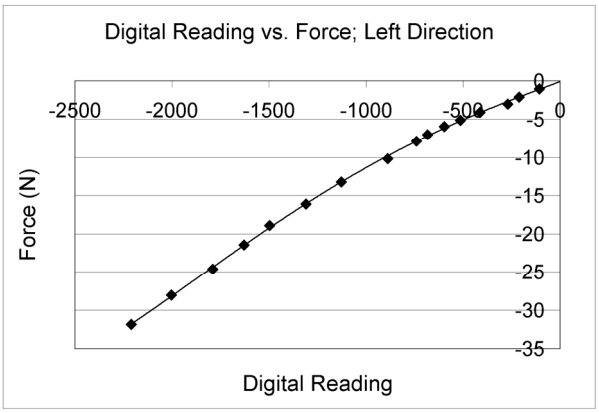


Figure 31: Left direction best fit curve. $y = 6.3121E\text{-}13x^4 + 2.3122E\text{-}09x^3 - 2.5613E\text{-}07x^2 + 9.3217E\text{-}03x - 6.9416E\text{-}02$; $R^2 = 0.99969$.

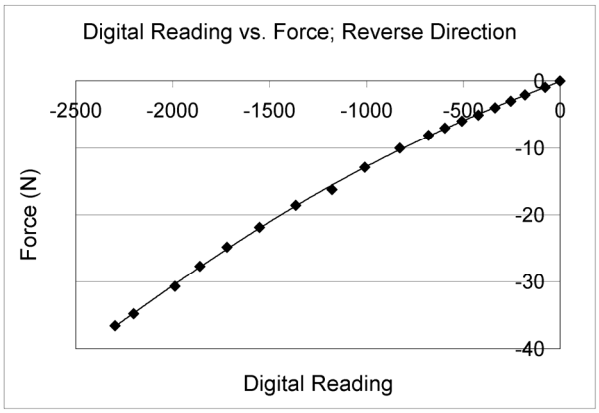


Figure 32: Reverse direction best fit curve. $y = 3.2298E\text{-}13x^4 + 1.4706E\text{-}09x^3 - 4.1389E\text{-}07x^2 + 1.1160E\text{-}02x - 5.1782E\text{-}02$; $R^2 = 0.99966$.

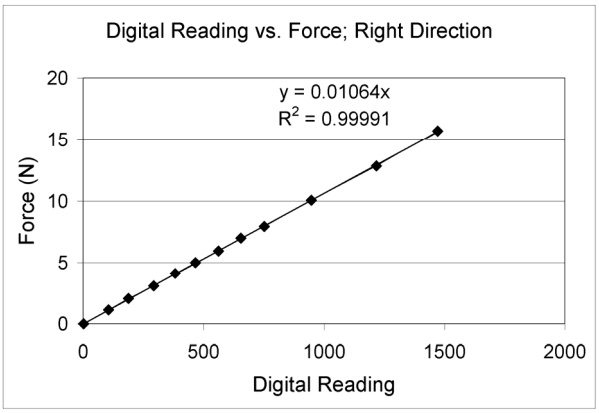


Figure 33: Right direction best fit curve.

3.1.1.4 Discussion

It is not entirely clear why the digital reading was lower than the expected results, nor why the negative side did not produce as much output as the positive side. It may be a result of the potentiometer in the input circuit. Nonetheless, if the cutoff values were not consistent with the design, their ranges may have been appropriate. For example, while the right turn cutoff force is 32 N and the left cutoff force is 18 N, this equates to a total range of 50 N or ±25 N if the offset is zeroed properly. The offset is the digital reading when no external force is applied to the joystick. During the experiments, the offset values were also recorded and showed biases in the positive direction, making is possible to exceed the designed cutoff force in the negative direction.

While the output signal was much lower than expected for the circuit in low gain mode as well as the high gain mode, modifying the low gain resistor did not seem to be critical because forces will very likely not reach 300 N. The 300-N requirement is based on a maximum input force in the forward direction. However, to characterize fatigue, the FI is based on the MVIC in medial direction. That is, if the subject is right-handed, the subject will apply a maximum force in the left direction. If the subject is left-handed, the subject will apply a maximum force in the right direction. In Windam's study of maximum input forces, the maximum force in the left direction (all subjects used their right hand) was 44.4 lbs [74] or 197 N.

While the positive axes (i.e. forward and right directions) showed linear responses up until the cutoff force, the response was not as linear as expected for the negative axes (i.e. the reverse and left directions). A fourth-order polynomial was chosen to approximate the relationship because it provides a very good fit to the data ($R^2 = 0.9997$).

3.1.2 Compliant Mode

3.1.2.1 Test Procedures and Setup

The method used for characterizing the compliant mode of the VCJ was similar to Spaeth's method for a MJ, where the force gage and joystick shaft were repositioned to precise locations for each measurement [67]. Unlike Spaeth's method, forces were applied to two different handles rather than an aluminum cuff. The elevation of the force was located where the diameter of the handle is the greatest. Different handles were included because as the handle tilts, the

weight of the handle will influence the amount of torque at the load cell. Likewise, the distance between the load cell and the plane in which the force is applied will influence the amount of torque at the load cell. Handles 3 and 4 described in Table 2 were used as representatives for the small and large classes of handles, respectively. Since the radius of the handle needed to be included when calculating the mill table's translations, a new set of translations was derived.

The table displacements were derived by defining the jig's pivot point as the origin **O** and tracking the position of the VCJ's pivot **P** through an angle θ. The table displacement is the difference between the new position $\mathbf{P_2}$ and its position $\mathbf{P_1}$ when the stick is upright and the force probe is barely making contact with the joystick handle. The free body diagram and equations in **Figure 34** explicate the method in more detail. The tables for mill positions for handles 3 and 4 are provided in Appendix G. The photograph in **Figure 35** depicts the VCJ and jig in the mill with the AccuForce II force gage and the Pro 360 Digital Protractor.

The procedure for comparing the force response of the VCJ with the MJ involved selecting a spring and pretension and measuring the input force and output signals for varying degrees of deflection. Spring selection was performed by choosing one from an assortment of springs that produced a similar, subjective feel to a conventional joystick. To set the pretension, the VCJ and calibration jig were configured to deflect the joystick post 8.0° as shown in **Figure 35**, where 8.0° represents an approximate midpoint for the full range of deflection. The position of the pretension nut was then adjusted until the force gage read the same force as that required to deflect the MJ 8°, or 2.2 ± 0.05 N [67]. With the pretension set, a second measurement was taken with the deflection set to 16° to verify that the spring's stiffness was adequate. A force too low would suggest that the spring is not stiff enough, and a force too high would suggest that it is too stiff. Force measurements were then recorded from the VCJ for 10 s each and from the AccuForce force gage at 1°, 2° through 16° in 2° increments, 17°, and 18°. Measurements were taken in only the forward direction. Special care when making contact between the joystick handle and force gage probe was needed to produce consistent results. The method that produced the most repeatability involved maneuvering the jig and mill table to their correct positions while the handle was pulled away from the probe and then very slowly releasing the handle until the force gage supplied all the input force.

Key

θ – deflection angle

ϕ – angle by which to rotate

D_v – vertical distance from jig pivot

D_h – horizontal distance from jig pivot

r_{jh} – radius of handle

L_{jp} – distance between VCJ pivot and handle's largest diameter

dx – horizontal translation

dz – vertical translation

$\mathbf{Q}_\phi$ – rotation matrix

Method

$$\mathbf{P_1} = \begin{bmatrix} (-D_h - r_{jh}) & (-D_v - L_{jp}) \end{bmatrix}$$

$$\phi = -\theta$$

$$\mathbf{Q}_\phi = \begin{bmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{bmatrix}$$

$$\mathbf{P_2} = \mathbf{P_1}\mathbf{Q}_\phi$$

$$\Delta = \mathbf{P_2} - \mathbf{P_1} = \begin{bmatrix} dx & dz \end{bmatrix}$$

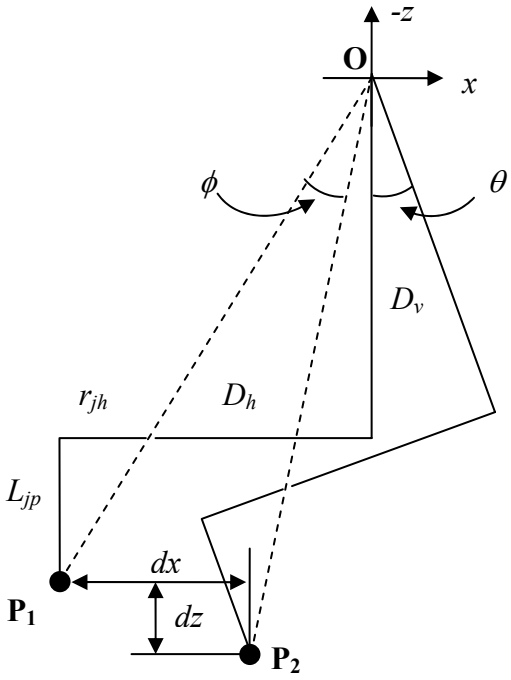
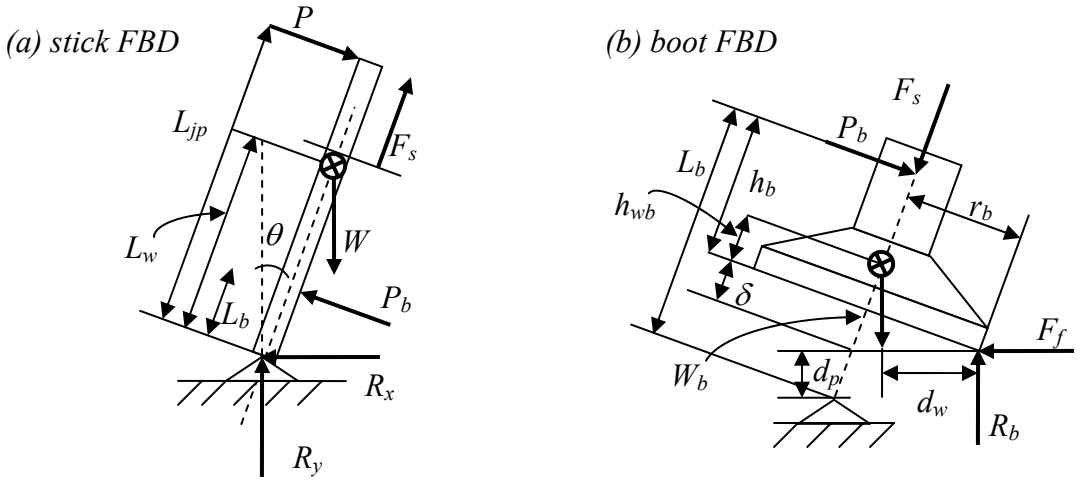


Figure 34: Analytical method for determining milling table displacement.

Figure 35: Photograph of test setup in compliant mode.

3.1.2.2 Analytical Model

An analytical model of the static response of the VCJ was derived to provide insight into the influence of different spring rates and pretensions without testing many different configurations. The free body diagrams in **Figure 36** served as the basis for the model. Application of the laws of statics to both masses produced a system of equations that can be solved using linear algebra. A MATLAB script (Appendix H) was then written to determine what input force P would result in a deflection of the stick θ degrees for handles 3 and 4. Locations for the centers of masses were computed with SolidWorks models of the joystick and boot, and the distance L_{jp} between the pivot and the input force was measured with a digital caliper. While a perfect mate between the stick and the boot would result in a uniform distribution of force between the stick's and boot's surfaces, the boot's inner diameter is slightly larger than the stick's outer diameter resulting in some play between the two surfaces. Therefore, the center of pressure h_b between the



Key		
d_p – depth of pivot below sliding surface	L_b – distance from pivot to boot reaction force	R_x – reaction at pivot, x-direction
d_w – horizontal distance to boot center of mass	L_{jp} – length of joystick post	R_y – reaction at pivot, y-direction
F_f – frictional force	L_w – distance from pivot to post-handle center of mass	W – weight of joystick post and handle
F_s – spring force	P – input force	W_b – weight of boot
h_b – height of P_b	P_b – force between boot and post	δ – axial deflection of boot
h_{wb} – height of boot center of mass	r_b – radius of boot base	θ – deflection angle
	R_b – reaction at boot, y-direction	

Figure 36: Free body diagram of (a) joystick shaft and (b) boot.

boot and stick was assumed to be about 85% high along the boot's axis rather than 50%. Friction between the surfaces was assumed to be negligible.

3.1.2.3 Results

Four different springs were available to choose from for the VCJ, with spring rates *K* of 3.00, 8.40, 15.0, and 28.05 lbs/in. The first spring that was selected, *K*=15.0 lbs/in, was not stiff enough. Therefore, the 28.05 lbs/in spring was used. The number of turns of the pretension nut from the top of the threads of the joystick shaft for calibration were 8.5 and a little over 9.0 for handles 3 and 4, respectively.

A comparison of the static responses for a MJ and the VCJ with handles 3 and 4 is provided in **Figure 37**. The digital force to deflection relationship is provided in **Figure 38**. Linear regressions were also performed with the digital force to deflection relationship and are provided in the same figure. Comparisons between the analytical model and experimental results for handles 3 and 4 are provided in **Figure 39**.

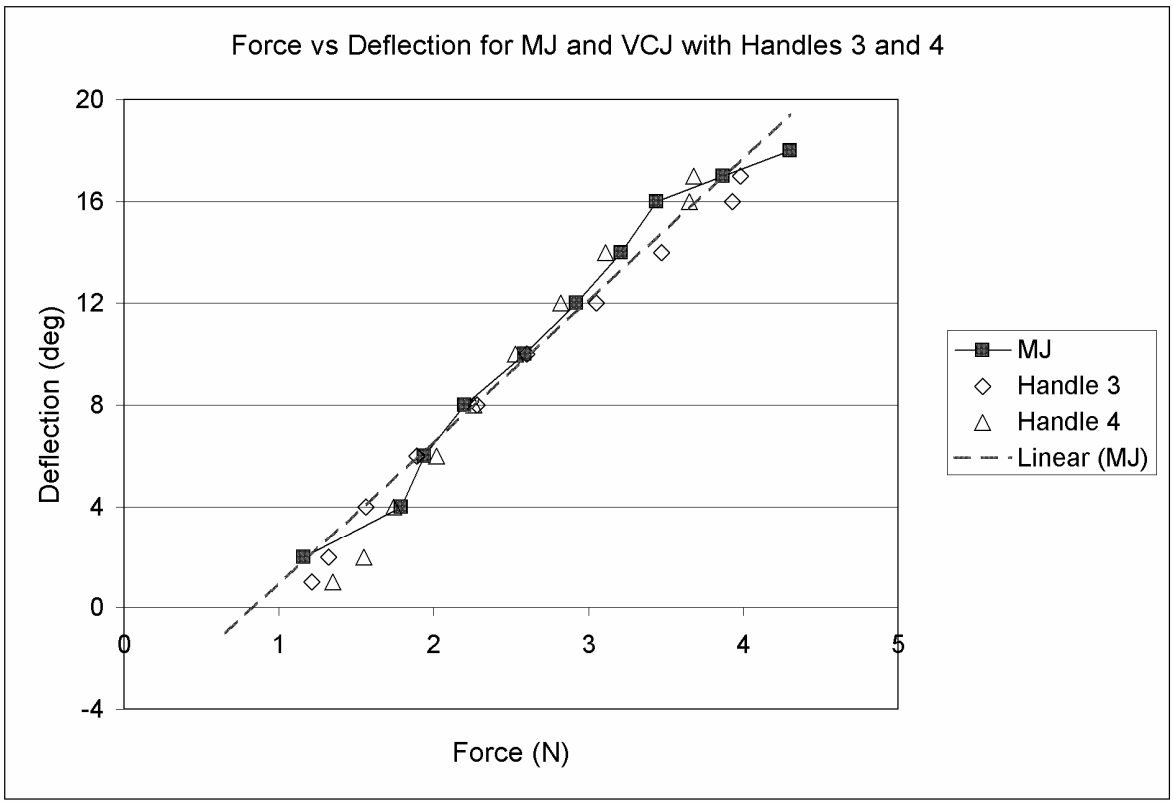


Figure 37: Force vs. deflection for MJ and VCJ wit handles 3 and 4.

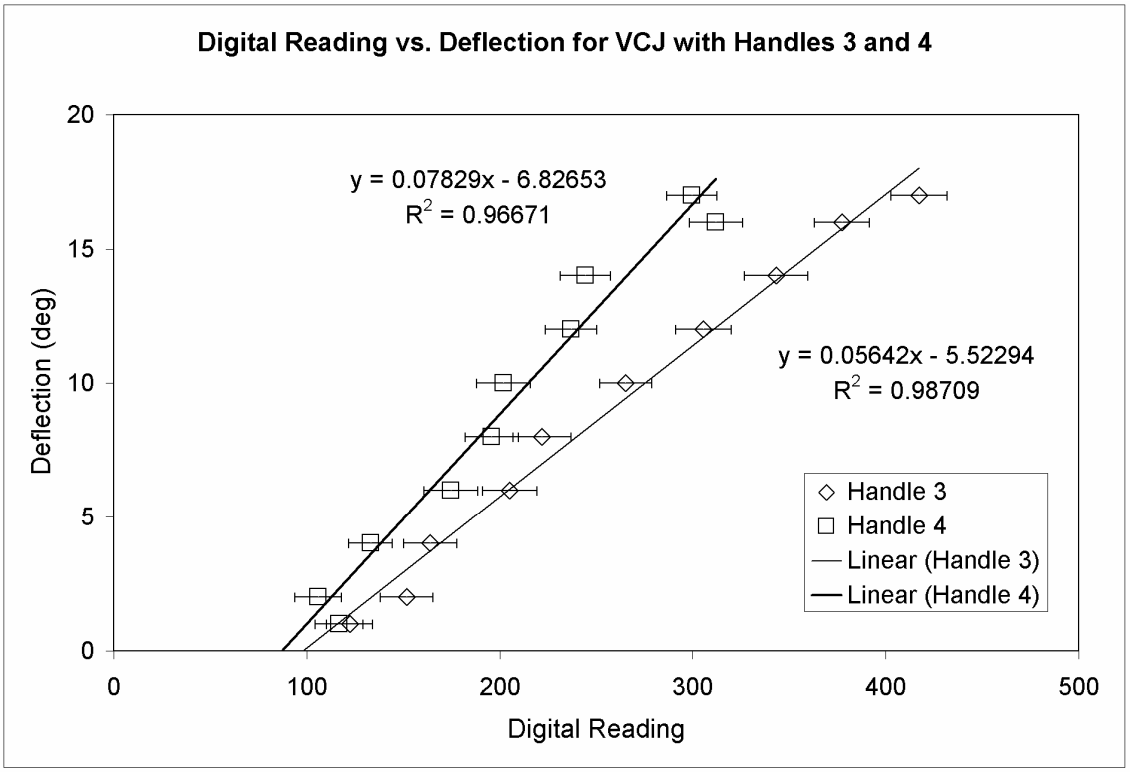


Figure 38: Digital reading vs. deflection for VCJ with handles 3 and 4.

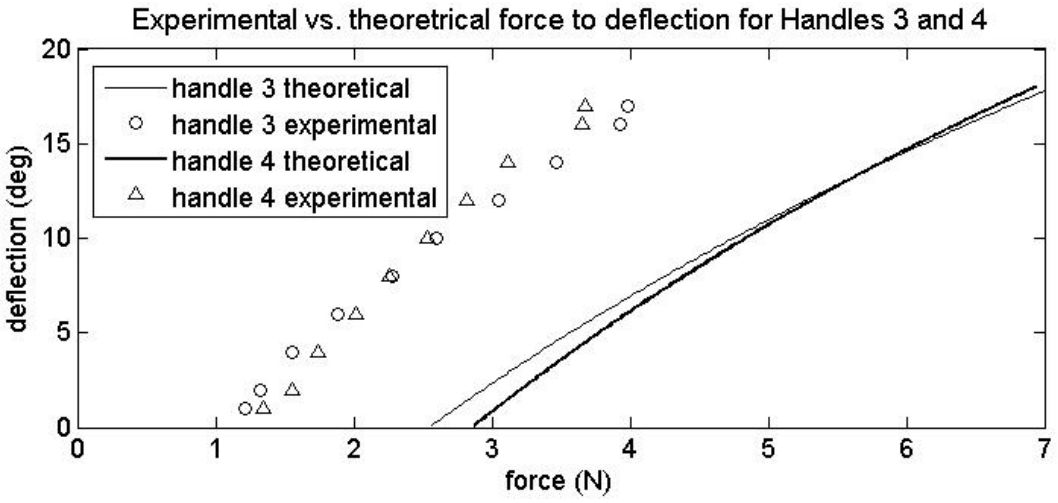


Figure 39: Experimental vs. theoretical force to deflection for handles 3 and 4.

While the VCJ responded very similarly to the MJ with the large and small handles, slight differences exist in their dead zone and peak forces. Table 6 summarizes the dead zone and peak forces for the MJ and VCJ with handles 3 and 4.

The digital readings are fairly linear, where the slope of the force response is greater for the VCJ with handle 4. This is consistent with the trend shown in Table 6, where not as much force is required to achieve maximum deflection and the dead zone forces are similar.

Lastly, there appears to be an offset between the experimental data and the analytical model. The analytical model suggests that about two to three times as much force is needed to deflect the joystick than what was experimentally determined. The slope is greater for handle 4, which would be expected as a result of the greater moment arm. Likewise, the dead zone force is greater for handle 4, which would be expected as a result of the greater pretension in the spring.

3.1.2.4 Discussion

The static response of the VCJ appears to match that of a conventional MJ very well. At the extremes, dead zone forces are not different by more than 0.5 N, and the peak force varies by less than 15%. The responses are very similar in the mid proportional range, which is where a majority of the driving forces are typically located during short driving tasks with a MJ [69].

While the digital data is noisy, distinctions can still be made within a few degrees for the VCJ with each handle. With a smaller lever arm, more force is required to deflect the joystick, thus increasing the digital range and improving sensitivity.

The analytical model and experimental data did not match very well. While the model curves have about the same slopes for both handles as the experimental data, the curves are offset by about 1.5 to 3.0 N. Originally, the curves matched very well until the author realized an error in the number of turns of the pretension nut to its axial displacement ratio just before

Table 6: Approximate dead zone and peak forces for the MJ and VCJ.

	MJ	**VCJ Handle 3**	**VCJ Handle 4**
Dead zone force (N)	0.9	1.1	1.25
Peak force (N)	4.4	4.0	3.75

submitting this thesis. If one turn of the nut results in 0.025 in. of axial displacement for a ¼-20 thread, the data matches very well. But, one turn actually results in 0.05 in. of axial displacement. The author double checked physical parameters and verified that the spring rate is what the manufacturer claimed but still cannot identify the source(s) of the discrepancy. This matter will be investigated further as deemed necessary.

Nonetheless, if there is some validity to the model, the curves in **Figure 40** through **Figure 42** depict how the force response would vary for increasing lever arms, spring rates, and pretensions. The results are consistent with what one would intuitively expect:

- longer shaft lengths decrease the amount of force needed to deflect the joystick because of the increased lever arm,
- increasing spring rates will increase the amount of force needed to deflect the joystick because of the increased stiffness,
- and greater preloading of the spring will increase the dead zone force because of the increased reaction for R_b.

Thus, to improve the consistency of the force response with handle 4 to a conventional MJ, lowering the location of the handle on the shaft and decreasing the pretension of the spring would probably help. However, the handle cannot be lowered any farther in the present configuration without physically altering it because it contacts the pretension nut when it is screwed onto the shaft.

The analytical results suggest a potentially exciting use for the VCJ operating as an IJ. As Rao, Seliktar, Rahman [65] mentioned that lack of somatosensory feedback may reduce performance with an IJ, a related complaint with the IJ is that the operator does not know when he or she is applying the force required to drive at maximum speed. Thus, drivers may have the tendancy to apply too much force [69]. To solve this problem, a configuration with the VCJ could be created where the dead zone force is greater than force typically required to reach the template. This could be achieved with a sufficient preload on the spring. The VCJ would operate as an IJ in normal driving circumstances; but when the applied force is too great, the pivot will buckle, thus providing feedback to the user that too much force has been applied.

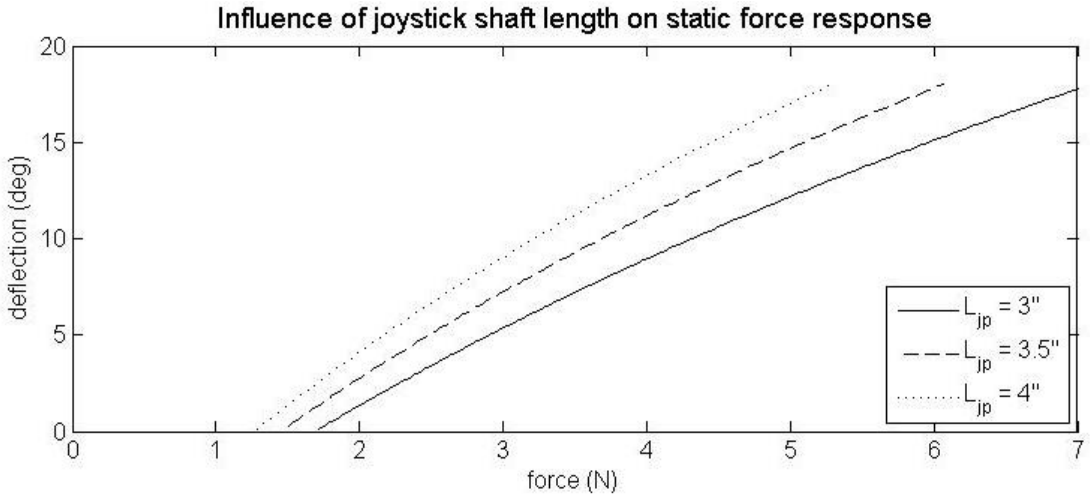


Figure 40: Influence of joystick shaft length on static force response of VCJ.

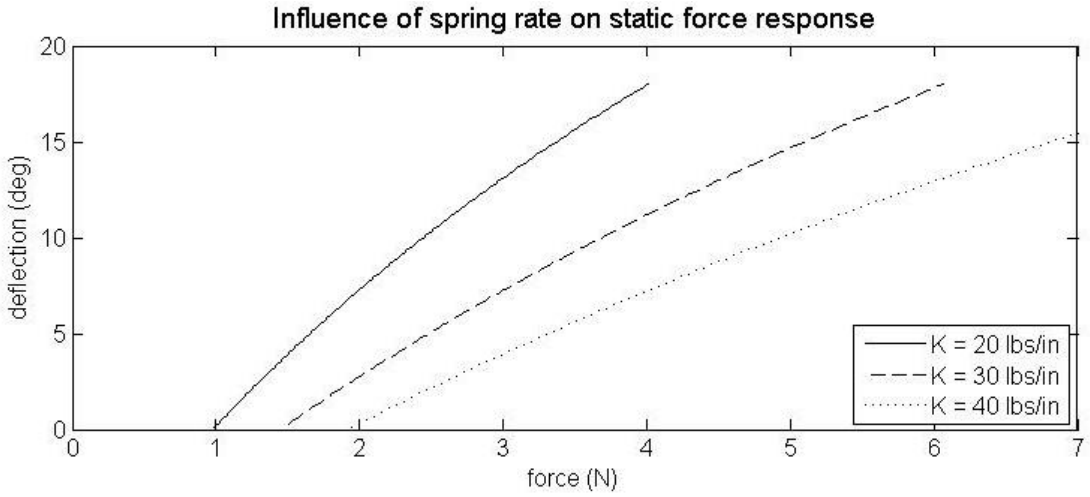


Figure 41: Influence of spring rate on static force response of VC.

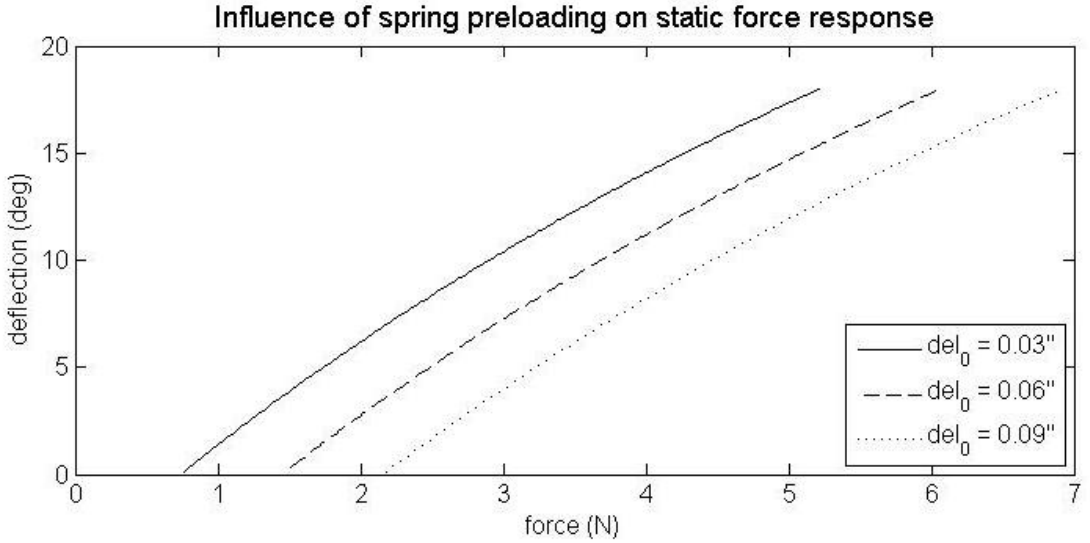


Figure 42: Influence of spring preloading on static force response of VCJ. del_0 is the spring's initial displacement.

3.2 DYNAMIC RESPONSE

The purposes of validating and characterizing the dynamic response of the VCJ are to ensure that its dynamic behavior is consistent with a standard MJ and to provide insight into the influence of different handles, springs, and the springs' pretensions on its movement. While static characterization will provide insight into how "stiff" the joystick is, or how difficult it is to deflect it to a certain angle, dynamic characterization will provide insight into how it "feels," or how it responds to the motions of the hand. While there have not been previous studies comparing the "feel" of a joystick with driving performance, Dicianno *et al.* [69] have suggested that EPW drivers rely on positional feedback for EPW control while using a MJ. Brown, Spaeth, and Cooper [78] previously characterized the impulse response of a conventional MJ in terms of its rise time, peak overshoot, and settling time, to which the behavior of the VCJ will be compared. While an analytical model was attempted, the resulting equations of motion were non-linear and subsequently oversimplified during the linearization process. Pursuing the model further to find a solution was deemed out of the scope of this thesis project. Instead, to provide insight into the influence of parameters such as handle size, spring rate, and spring pretension, similar impulse response experiments but with different configurations were performed with the VCJ, where its tip was released from an angle and its position over time recorded.

3.2.1 Test Procedure and Setup

The same test setup as described in Brown, Spaeth, and Cooper [78] was used to capture the motion of the VCJ. A small reflecting sphere was taped to the joystick tip, and a Vicon MX motion capture system (ViconPeak, Lake Forest, CA) recorded its position as the stick was released from the forward, right, and reverse directions. The diamond-shaped template (**Figure 13b**) was used to provide consistent start points for the three directions. In addition to the configurations used for emulation of a MJ (see first paragraph of section 3.1.2.3), two other pretension settings with the spring were used with each handle as well as two different springs. Table 7 provides a summary of the configurations used. The student *t*-test was used determine whether results were significantly different, with the significance level set at 0.05.

Table 7: Spring settings used to test dynamic response of the VCJ.

Spring ID	Rate (lbs/in)	Handle	Number of lock nut turns
1	28.05	3	8, 8.5, 10.5
		4	8, 9.1, 10.5
2	15.0	3	7.5, 8.5, 10.5
		4	7.5, 9, 10.5
3	3.00	3	12, 15
		4	15, 17

MATLAB was used to import the data; interpolate the data with a cubic spline; and find the rise time, peak overshoot, and settling time. The rise time was defined as the time it took the joystick tip to be within 10% of the resting position. The peak overshoot was defined as the maximum deflection relative to the initial deflection past the resting position after it was released. And the settling time was defined as the time it took the joystick tip to remain within 5% of the resting position relative to the initial deflection.

3.2.2 Results

While computing the characteristic parameters for the VCJ, an error was found in the algorithm to compute settling time in [78]. Rather than being within 5% of the joystick's final position, the settling time was computed to be within 5% of zero. This alters the results reported in [78] slightly since the position marker would rotate with respect to the joystick occasionally. Comparisons between the VCJ with handles 3 and 4 to the MJ are provided in Table 8. Results were analyzed based on the direction of impulse because of differences in characteristics for a conventional MJ for the various directions. Five of nine characteristics are not significantly different for handle 3, and three of nine are not significantly different for handle 4.

Parametric analyses of the influence of handle size, spring rate, and spring pretension are provided in **Figure 43** through **Figure 51**. Averages with standard deviation error bars for the dynamic characteristics are plotted against the pretension in terms of the number of turns of the

Table 8: Comparisons between rise time, peak overshoot, and settling time for the VCJ and handles 3 and 4 with a conventional MJ.

		Forward		**Reverse**		**Lateral**	
Rise Time (sec)	MJ	0.022 (0.004) n=30		0.027 (0.004) n=31		0.026 (0.003) n=26	
	VCJ Handle 3	0.018 (0.003) n=6	*p=0.02*	0.026 (0.002) n=11	p=0.29	0.026 (0.002) n=12	p=1.0
	VCJ Handle 4	0.027 (0.003) n=12	*p=0.0001*	0.033 (0.003) n=11	*p<0.0001*	0.032 (0.002) n=10	*p<0.0001*
Peak Overshoot (%)	MJ	57.3 (16.8) n=30		46.5 (5.1) n=31		52.0 (9.3) n=26	
	VCJ Handle 3	78.8 (13.5) n=6	*p=0.009*	59.3 (7.44) n=11	*p=0.0001*	55.4 (2.59) n=12	p=0.09
	VCJ Handle 4	64.4 (6.75) n=12	p=0.10	57.1 (3.82) n=11	*p<0.0001*	64.1 (1.12) n=10	*p<0.0001*
Settling Time (sec)	MJ	0.134 (0.041) n=30		0.146 (0.044) n=31		0.158 (0.020) n=26	
	VCJ Handle 3	0.141 (0.002) n=6	p=0.36	0.131 (0.013) n=11	p=0.10	0.132 (0.016) n=12	*p=0.0002*
	VCJ Handle 4	0.149 (0.010) n=12	p=0.07	0.172 (0.040) n=11	p=.087	0.205 (0.002) n=10	*p<0.0001*

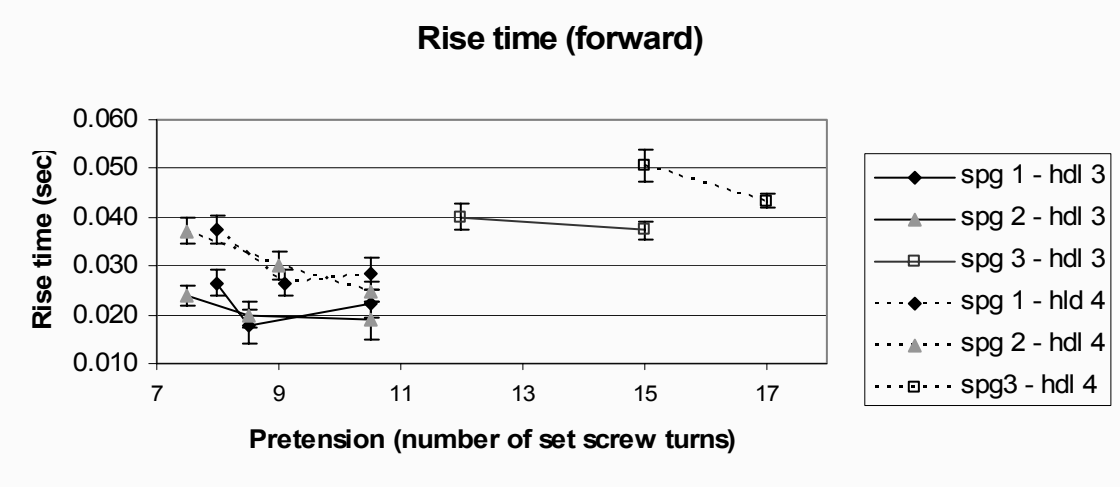


Figure 43: Influence of handle size, spring rate, and spring pretension on rise time for the forward direction.

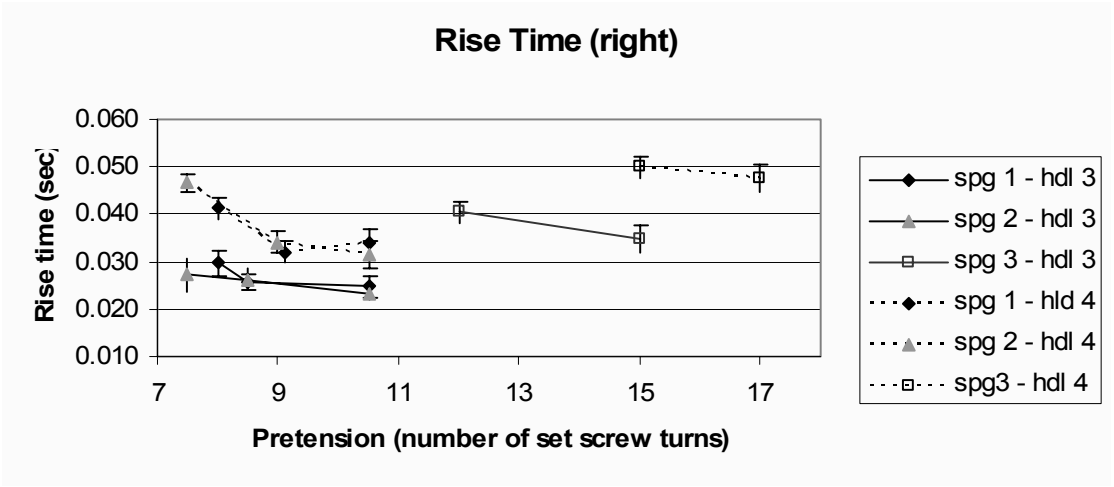


Figure 44: Influence of handle size, spring rate, and spring pretension on rise time for the right direction.

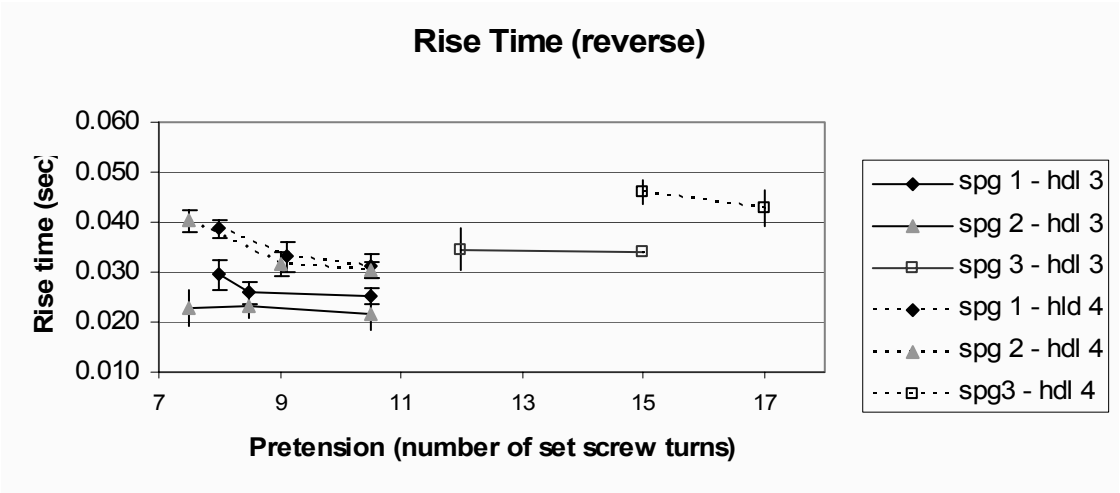


Figure 45: Influence of handle size, spring rate, and spring pretension on rise time for the reverse direction.

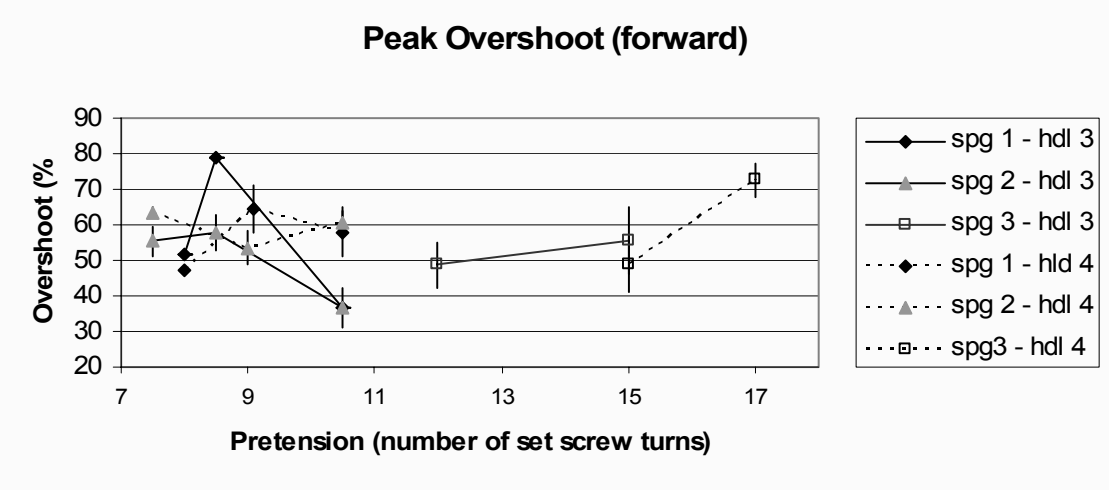


Figure 46: Influence of handle size, spring rate, and spring pretension on overshoot for the forward direction.

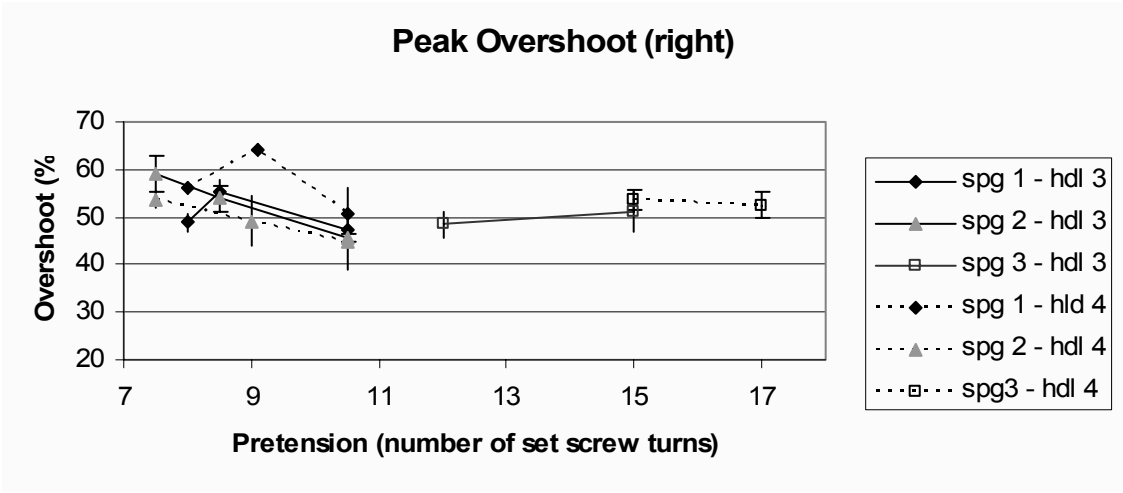


Figure 47: Influence of handle size, spring rate, and spring pretension on overshoot for the right direction.

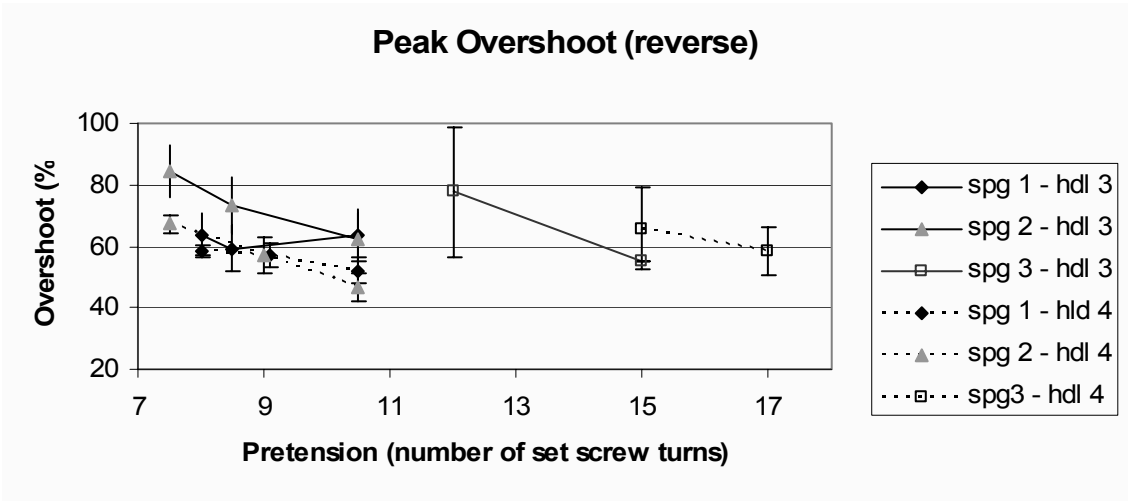


Figure 48: Influence of handle size, spring rate, and spring pretension on overshoot for the reverse direction.

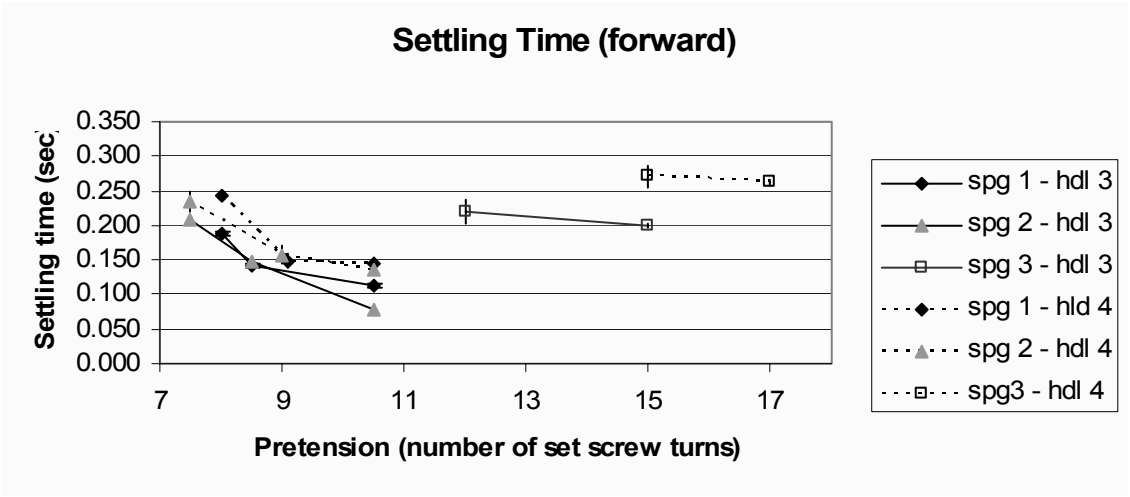


Figure 49: Influence of handle size, spring rate, and spring pretension on settling time for the forward direction.

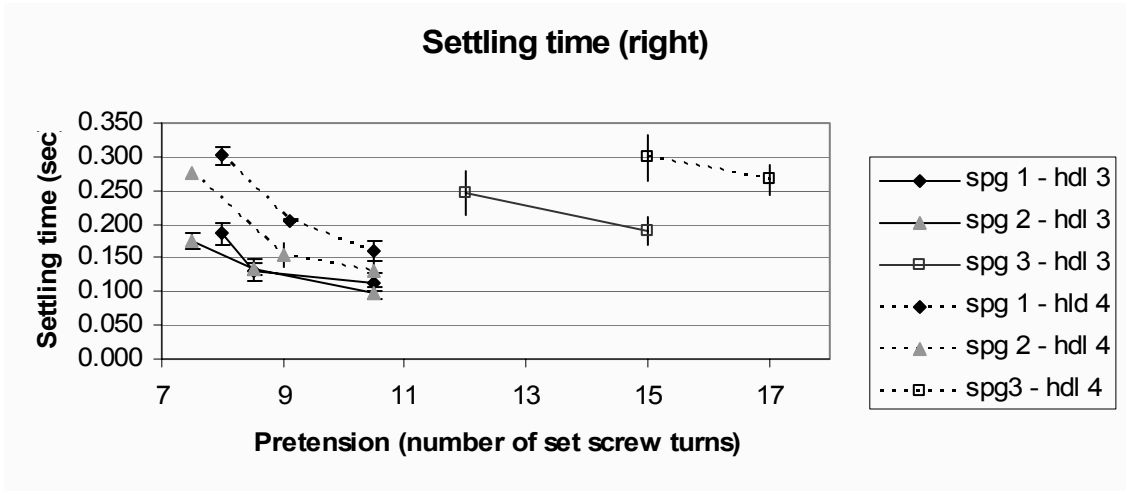


Figure 50: Influence of handle size, spring rate, and spring pretension on settling time for the right direction.

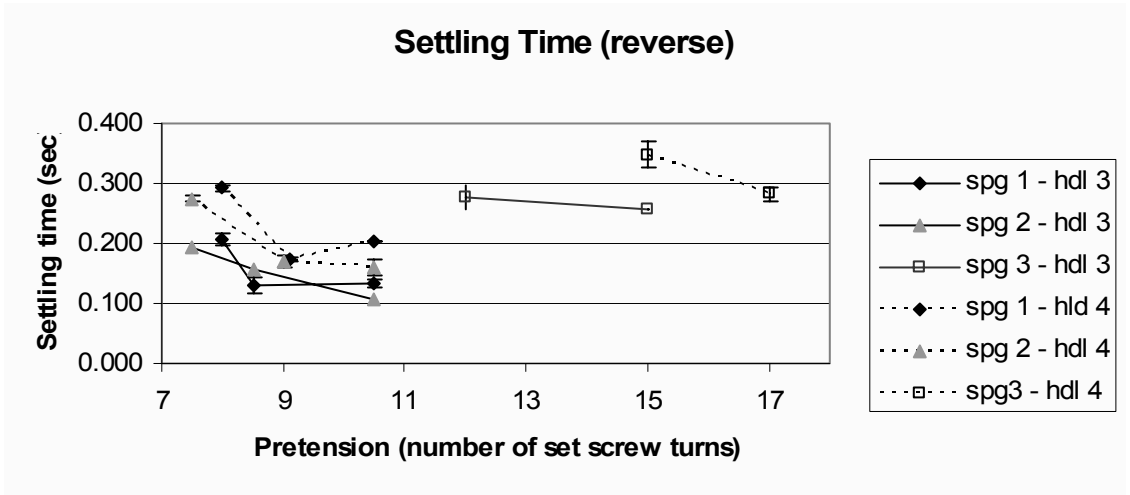


Figure 51: Influence of handle size, spring rate, and spring pretension on settling time for the reverse direction.

set screw from the top of the threads. Dashed lines indicate handle 4, while solid lines indicate handle 3.

3.2.3 Discussion

3.2.3.1 Comparison of VCJ with MJ Characteristics

The VCJ marginally matches the dynamic response of a conventional MJ. The VCJ with handle 3 satisfied more than 50% of the criteria, while the VCJ with handle 4 satisfied only 33%. The VCJ with handle 3 differed in peak overshoot the most, which could indicate input signals will be amplified slightly more. The VCJ with handle 4 tended to have longer rise times, greater overshoot, and longer settling times, all of which are very likely related to its larger mass. The larger mass has more inertia, which would result in a general resistance to change, both from a resting position to moving and from moving to resting. Thus, the VCJ with handle 4 may have a tendency to feel "heavy." These said, this experiment had a few limitations.

While results were statistically significantly different, it is unclear to what extent the joystick will feel different. It seems unlikely that an operator would be able to distinguish between settling times that differ by 0.026 s, for example, as was the case for the lateral deflection. Further investigation is needed to quantify what amount of change in mass and dynamic properties would be perceptible and influential during pointing tasks with a MJ.

Threats to the validity of the test include human error and appropriateness of the test setup. Human error may induce variance in the start time of a trial because the joysticks were released by hand from the deflected position. While effort was made to release the joysticks as instantaneously and consistently as possible, friction between the fingertip and joystick handle could slow down the response of the motion. Likewise, the release method also means that the release time and the Vicon's start time were not synchronized, resulting in the need for graphical analysis to judge the start time. The start time was thus defined as the time before which the tip moved by more than 0.05 in., with exceptions made for particularly fast responses. Because the same release method and the method for judging the start times were consistent for all trials, for both MJ and VCJ characterization, results should not be biased for any particular testing configuration.

The test setup threatens test validity because the testing configuration consisted simply of the joystick handle and the small reflecting sphere responding to an impulse. In real driving, however, the joystick is coupled with the driver's hand. The 5th percentile female has a hand/forearm mass of 470 g, while the 95th percentile male has a hand/forearm mass of 2,160 g [72]. The mass is an important aspect of dynamic response, and masses of 20 to 100 times that of the experimental conditions could significantly affect the actual behavior of the joystick. But, EPW drivers use different grips while holding the joystick: some use just their first three fingers while others use their whole hand. The important aspect of this experiment was that conditions were consistent for both the MJ and VCJ. Perhaps in the future it may be wise to add mass to the joystick handle, but this adds many extra questions such as how much, where it should be located, and what shape it should possess.

3.2.3.2 Parametric Analysis of Handle Size, Spring Rate, and Spring Pretension

The handle size had perhaps the most effect on the VCJ's rise time. In nearly every case, the rise time was longer with handle 4 than handle 3 when controlling for spring rate and pretension. The spring rate did not appear to have as much of an influence on rise time as handle size, especially between springs 1 and 2; but spring 3 generally had longer rise times than the springs with higher rates. This may also be related to the initial loading condition of the springs in the fully deflected positions. Greater pretension indicates that the force to deflect the joystick to its full position is also greater. Even though the pretension on spring 3 was great, its low rate means that the force to deflect it fully is less than that of the other springs. Therefore, with less force pushing back, it will take longer to return to the center position.

Trends do not appear to exist consistently for the handle size, spring rate, and spring pretension as they relate to the peak overshoot. In some cases greater pretension indicates more overshoot; but in other cases, just the opposite is true. Overshoots based and handle size seem to overlap too much to note any significant differences.

Handle size and pretension appear to have the most influence on the VCJ's setting time. Handle 4 has a longer settling time very likely a result of its larger mass. Since it possesses more inertia, it will take it longer to come to a resting position. Greater pretension seems to indicate a shorter settling time. While spring 3 differed from springs 1 and 2, this may again be related to

the initial loading of the spring, or the amount of force required to deflect the joystick fully. With less preloading, there is not as much energy for the joystick to return itself to the center position.

3.2.4 Conclusions

The mass of the handle influences the dynamic behavior of the joystick since it slows down both the rise time and settling time of an impulse response when controlling for spring rate and pretension. While the larger handle also has a larger radius which should be ergonomically more appealing [75], its extra inertia would require more power from the user to operate. This would have a negative affect for EPW drivers with muscular dystrophy, for example, whose muscle strength gradually decrease over time. Alternatively, a large enough mass may have potential to help mitigate the effects of tremor since the impulses are slowed down. Implementing such a handle, however, would probably require a larger spring rate to ensure that the joystick can return to its center position quickly after the driver lets go. If not corrected in software, the extra power requirements may be a precursor for fatigue.

The spring rate and pretension will also influence the "feel" of the joystick since a smaller rate will slow down the response and a greater pretension will increase it. The slower response with a smaller spring rate does not mean that more power is required to operate the joystick as described above for larger masses. Rather, with less preloading, its tendency to return to center will be decreased. A low spring rate may be very appropriate for a person with limited strength; and a MJ with a sufficiently low spring rate (<1 lb/in) would truly be isotonic, where the same amount of force is necessary for any deflection angle. The exact amount of force to operate the joystick would simply be adjusted with the pretension. One must take care, however, when selecting such a spring because it would not be good for a MJ based on force transducers, such as the VCJ. There needs to be enough gradation in force for the sensors to be able to interpret the driver's commands sufficiently.

4.0 TUNING THE VCJ

4.1 INTRODUCTION

The purpose of tuning the VCJ is to adjust its features to match the characteristics of its operator for improved EWP control. Tunable features can be categorized as hardware or software and are listed in Table 9. The procedure for tuning the VCJ involves systematically asking the user to apply forces to the joystick and getting subjective feedback about its feel. The steps, which are discussed in more detail in this chapter, are provided in Table 10. Dead zone shape and size, minimum gain, and bias axis angle are derived directly from forces the user applies the joystick. Maximum gain, template size, template shape (hardware and software), handle selection, and VCJ position are derived from feedback from the subject and an evaluation of driving skill in a virtual environment. Fatigue adaptation and recovery rates are derived from three measurements

Table 9: Tunable features of the VCJ.

Hardware	**Software**
Handle size and shape	Dead zone size and shape
Template shape	Minimum Gain (left, right, forward, reverse)
VCJ position	Maximum Gain (left, right, forward, reverse)
	Template size and shape
	Bias axis
	Fatigue adaptation and recovery rate
	High-pass filter order and cut-off frequency
	WFLC initial configuration and adaptation rates

Table 10: Procedural steps for tuning the VCJ.

Step ID	Description
1	Joystick positioning
2	Handle selection
3	Introduction to calibration software
4	Dead zone determination
5	FI 1
6	Bias axes and default gain determination
7	Strength analysis
8	FI 2
9	Max gain determination
10	Data analysis with MATLAB (fine tune α and β and set filter parameters)
11	FI 3
12	Virtual driving on track

of the user's FI: baseline, after activity, and after a rest period. This chapter will describe how to configure the joystick's mechanical properties and define its software parameters.

4.2 ADJUSTING HARDWARE FEATURES

Before tuning any of the software parameters, the clinician should position the joystick so that the user has easy access to its handle. Height adjustment may be performed by loosening the restraining bolts and sliding the joystick up or down along the slotted arm (see **Figure 52**). With the restraining bolts loose, lateral adjustment may be performed by moving the joystick forward or backward meanwhile rotating the VCJ's mounting bracket to keep the joystick roughly level. Set screws in the mounting bracket prevent the joystick from rotating after the final position has been selected and restraining bolts tightened. The larger dial at the base of the slotted arm allows for 15° increments, and the dial on at the top of the slotted arm allows for 30° increments.

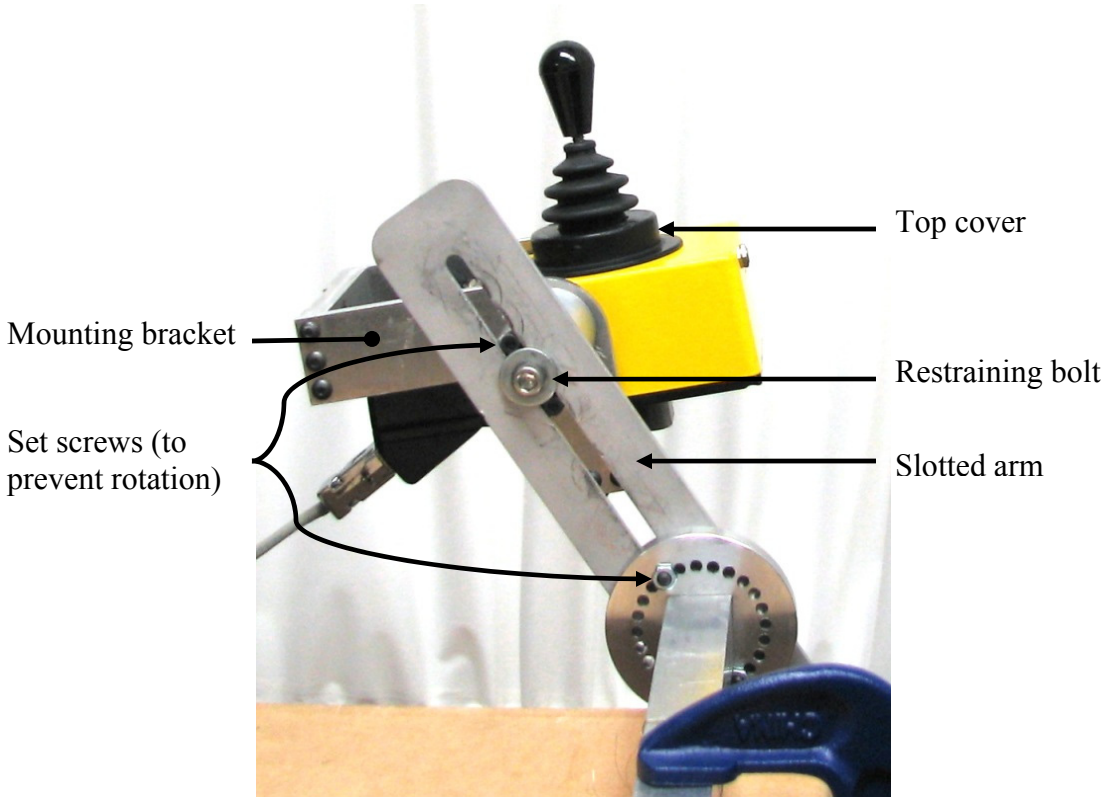


Figure 52: Photograph of VCJ in adjustable mounting bracket.

Next, the subject should select a handle that feels comfortable. If the subject selects one of the two large handles while in compliant mode, the lock screw should be turned a little over 9 rotations, or about 9.1, from the top of the threads. If the subject selects one of the three smaller handles, the lock screw should be turned 8.5 rotations from the top of the threads. This may be performed after removing the handle and then the black top cover from the top of the yellow box. The top cover is attached to the yellow box with two 10-32 flat head machine screws located on its top surface. When emulating an IJ, the spring should be replaced with the Aluminum insert and the lock nut tightened over it.

If necessary, the mechanical template may be adjusted after removing the top cover. Either an alternate template may be inserted, or the existing template may be rotated such that the notch matches the key in the bearing mount.

4.3 TUNING SOFTWARE PARAMETERS

Two applications support tuning software parameters for the VCJ: *MS Study (MSS) Tuning* and *MSS Input Analysis*. *MSS Tuning* is a Windows-based executable designed to collect input from the VCJ and to help personalize the features of the joystick. *MSS Input Analysis* runs in the MATLAB command window and allows the clinician to tune tremor filter parameters.

4.3.1 MSS Tuning

Developed in Borland C++ Builder version 5, *MSS Tuning* collects user data to compute the dead zone, fatigue adaptation parameters, bias axis, gains, and template shape. It is based on the tuning interface developed by Ding, Cooper, and Spaeth [77] and has been expanded to meet the needs of the study.

4.3.1.1 Dead Zone

The first screen with which the clinician is presented is the Personalized Deadzone window (see **Figure 53**). Here, the subject places his or her hand on the joystick in a resting position, and the clinician clicks the **Record Data** button. The software collects data from the joystick for 30 s to get a feel for isolated hand motions when the subject is not intending to drive the EPW. While in driving mode, signals within the dead zone will be set to zero to prevent the chair from moving.

After clicking **Show Data**, the clinician has the option to choose the shape and size of the dead zone. When clicking on a shape – circular, square, or elliptical – the tuning software will automatically compute a size such that the selected shape encapsulates all data points. Moving the slider bars adjusts its size; and maximum values in the direction and speed axes are displayed in the text boxes for `xvalue` and `yvalue`, respectively. As a rule of thumb, typical MJs have a dead zone of about 1 N, which corresponds to roughly 100 points in the text boxes.

The clinician should enter the subject ID number in the text box at the bottom right of the screen. Clicking on **Click to next step** takes the clinician to the first screen for determining fatigue adaptation parameters.

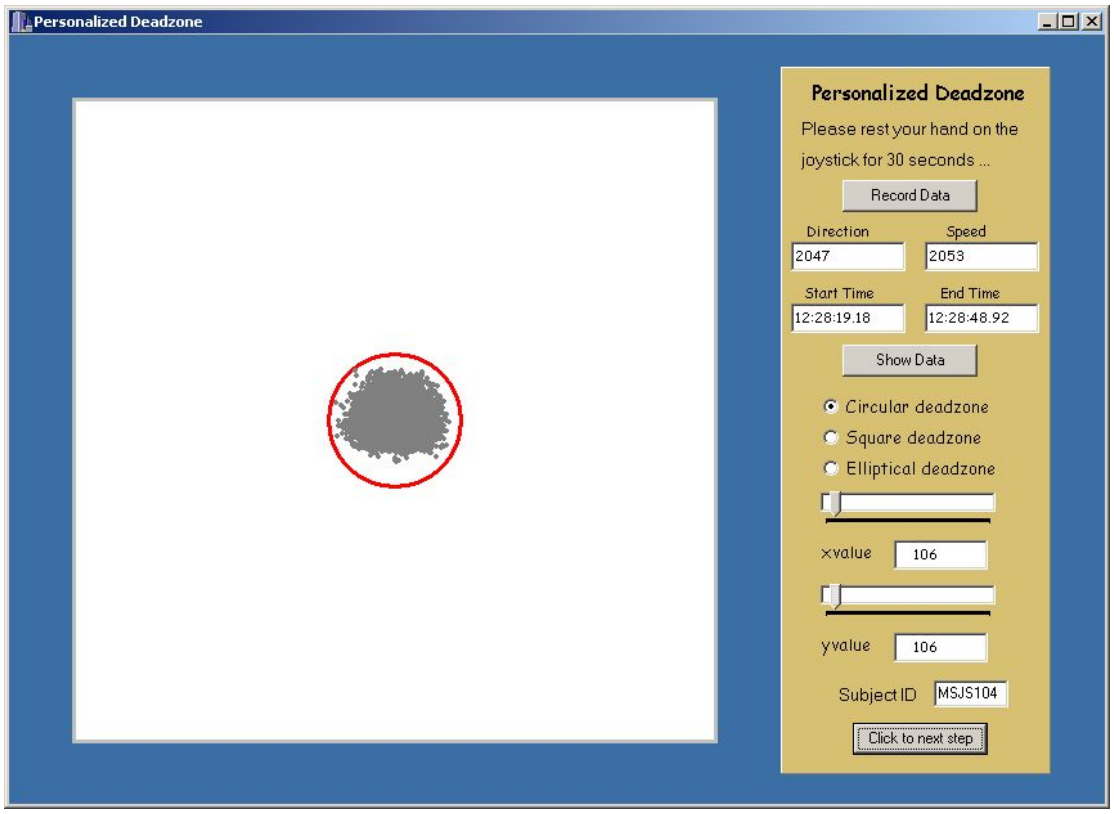


Figure 53: Personalized Dead Zone window.

4.3.1.2 Fatigue Adaptation

The Gain Adjustment window (see **Figure 54**) appears on three occasions to tune the fatigue adaptation parameters, which are based on the subject's FI. The FI is based on the subject's MVIC over 30 s and computed with equation 1. The first instance collects a baseline FI; the second instance assumes that the subject has fatigued during the bias axis and gain determination steps; and the third instance, performed after *MSS Input Analysis* has been completed, assumes the subject has recovered to some extent. The relationship between the first and second FI controls the fatigue adaptation parameter α. Specifically,

$$20: \qquad \alpha = \frac{-1}{t}\ln\left(\frac{FI_2 - FI_{Min}}{FI_1 - FI_{Min}}\right),$$

where t is the time between readings in milliseconds and FI_{Min} is an assumed minimum FI for all people. The relationship between the first and third FI determines the fatigue recovery parameter β. Specifically,

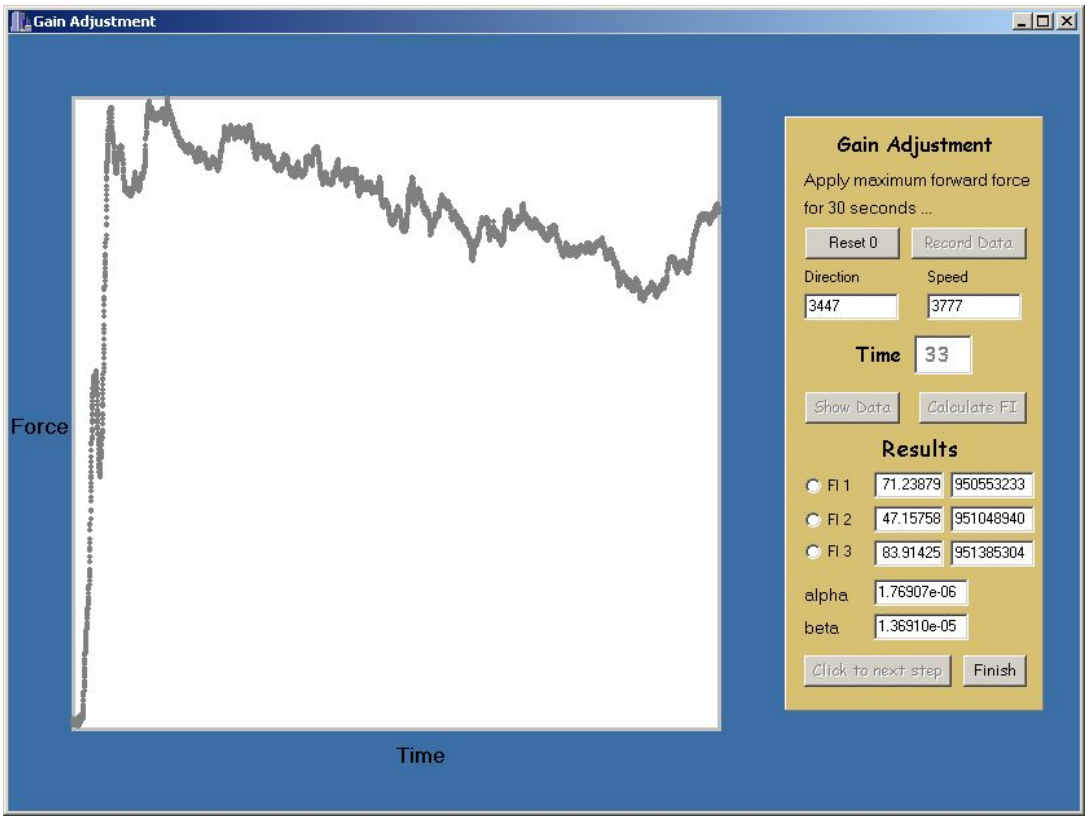


Figure 54: Gain Adjustment window.

$$21: \qquad \beta = \frac{-1}{t}\ln\left(\frac{-(FI_3 - FI_{Min})}{FI_1 - FI_{Min}}\right)$$

We had also considered simply asking the subject how tired he or she is on a scale of 1 to 5 between readings. The numbers would then correspond to percent fatigued (i.e. 1 = rested = 20%, 2 = moderately tired = 40%, 3 = tired = 60%, 4 = very tired = 80%, and 5 = extremely tired = 99%). But lassitude – a person's perception of fatigue – does not correlate with his or her motor fatigue [12]. While the final method has a couple limitations noted below, its benefit is that it provides an objective way of estimating the α and β parameters based on the subject's change in FI and the time between readings. Larger differences between FI_1 and FI_2, which would indicate a large increase in the person's fatigue, result in less time to increase to the higher gain. But if the readings were taken far apart from each other, it may not mean that the subject fatigues quickly. Larger differences between FI_1 and FI_3, indicating that the person has not yet reached his or her original fatigue state, result in more time to regain the default gain. But if the

two readings were taken close to each other, it may not mean that the subject recovers slowly. Limitations are a result of the novelty of this method.

The method for determining the α and β parameters has a couple limitations because the FI is still a relatively new scale. While the FI can detect differences between MS subjects and healthy controls [12] and between an exacerbation and remission [20], the literature does not compare the FI before and after physical activity. Likewise, average values for FI in MS subjects are provided [16], but the data does not specifically mention the existence of an absolute minimum. To mitigate these limitations, the clinician may adjust the α and β parameters in their respective text boxes after reviewing data presented in *MSS Input Analysis* (the meaning of which is discussed in section 4.3.2.1).

The clinician must adjust the VCJ in three ways before collecting data from the joystick for fatigue characterization. The first step is to remove the joystick post with the pivot mechanism and replace it with the isometric insert. The switch mounted on the rear of the VCJ enclosure should be switched to the low position. Only after the switch has been flipped low the **Reset 0** button should be clicked. These allow the subject to press up to 350 N (~75 lbs) on the joystick without it breaking and the software to measure a force that high.

Before collecting data, the clinician should instruct the subject to press as hard as possible in the left or right direction (whichever is medial for the subject) and hold it for a little more than 30 s. It is also important that he or she reaches his or her maximum possible force within the first five seconds. Before the first test, the clinician should ask the subject to try pressing on the joystick as hard as possible before collecting any data for the subject to become familiar with the response of the isometric post. The subject's elbow should be resting comfortably on the armrest, and the subject should not change his or her grip during the reading period. The task will be taxing, and the subject may need encouragement to maintain his or her MVIC for the full length of time. While only the first 30 s are used to determine the FI, data is recorded for 33 s to mitigate any errors if the subject chooses to let go before the full 30 s has elapsed. Clicking **Record Data** begins the data collection process. The timer does not count down on the screen, but the clinician will see that data collection has stopped when the `Direction` and `Speed` values stop updating. The next steps are to click on **Show Data**; **Calculate FI**; and, depending on which FI is computed, **Click to next step** or **Finish**.

If it appears that the subject did not follow instructions, it is possible to redo a measurement by clicking on the radio button for the erroneous measurement and clicking **Record Data** again. Potential errors include not reaching the maximum force within the first five seconds or letting go before 30 s had elapsed. The new readings will overwrite the previous measurement.

4.3.1.3 Bias Axes, Default Gain, and Template

After exiting the Gain Adjustment window for the first time, the Bias Axes, Gain, and Template Settings window opens (see **Figure 55**). While the primary goals are to compute these three settings, this screen is also designed to induce some fatigue in the subject's hand. The subject may rest at any time but should be notified of the fatiguing affect beforehand and how it will be used to tune parameters for fatigue compensation later in the visit. Also before collecting data, the clinician should replace the isometric insert with the pivoting VCJ post, switch the toggle on

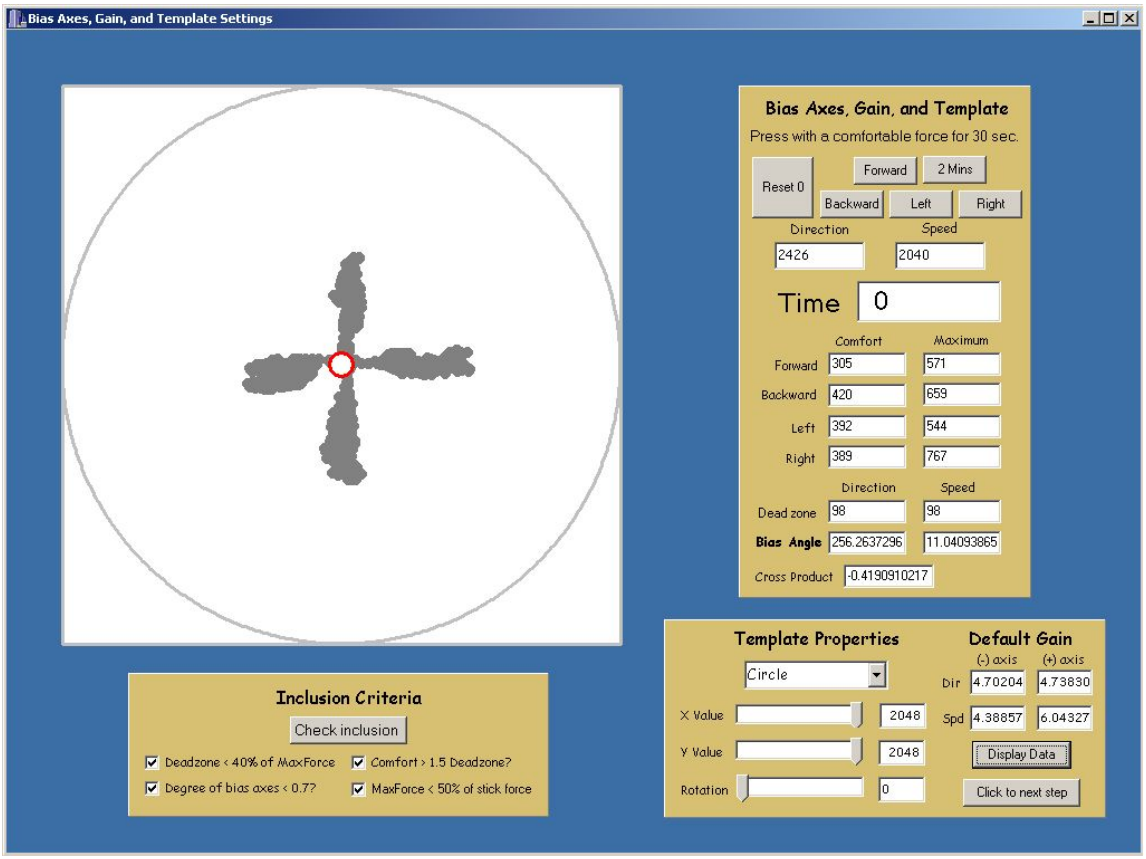


Figure 55: Bias Axes, Gain, and Template Settings window.

the rear of the enclosure back to the high position, and click on the **Reset 0** button after the switch has been flipped. These steps reset the VCJ to its driving mode.

To compute the bias axes and gain, the subject should press on the joystick for 30 s in the four directions corresponding to the button names (**Forward**, **Backward**, **Left**, and **Right**). The subject should imagine he or she is driving down a long corridor or turning for an extended period of time and press with a force that seems reasonable to him or her for such a task. Once data collection has begun, a new window will appear to provide some feedback to the subject. So long as the subject presses with enough force to keep the bar moving, the VCJ will be able to interpret grades between unintended movements and full speed.

To collect data for evaluating endurance, subject should press comfortably on the joystick in any direction for 2 min. Clicking on the **2 Mins** button begins the data collection process. Again, a window will appear indicating whether or not the subject is pressing with enough force.

Once data collection is complete, the clinician should ensure that inclusion criteria are met (a feature left over from a prior study) and select a template. Though the inclusion criteria do not need to be followed rigidly in this study, they are good to follow to ensure that the VCJ can operate with sufficient efficiency. Perhaps the most significant criterion is the degree of bias axes. A value less than 0.7 allows the VCJ to distinguish between inputs applied along the subject's direction and speed axes.

Unless the clinician knows that the subject has driving limitations, the shape and size should be set to circular and the maximum setting in each direction, respectively. Clicking on **Display Data** shows the data with the template superimposed and automatically computes the default (rested) gain parameters. The gains are derived from average values of the second half of data collection and assumed to be 90% of the subject's maximum driving force.

Adjusting the template can be likened to setting a governor in a vehicle or imposing stops on a steering wheel. It will limit the speed at which the wheelchair can drive or turn. For example, a value of 2048 for the y-axis will permit the user to command the chair to drive 7 ft/s at top speed. Setting it to 1024, however, will cause the max speed the person can drive to be 3.5 ft/s. The template shape should be adjusted if the subject is having difficulty maintaining a heading with the joystick. In order of most to least freedom, possible templates include circle, ellipse, diamond, and asteroid. The templates may be rotated for graphical purposes, but the

Rotation slider bar is not used in VCJ software. Zero degrees bias is located along the +y-axis, increasing in the direction of the +x-axis.

4.3.1.4 Secondary Gain

After exiting the Gain Adjustment window for the second time, the clinician may select the maximum gain for the VCJ (i.e., the gain the joystick will have when the subject is assumed to be totally fatigued) in the Maximum Gain window (see **Figure 56**). But, since this step follows the fatigue characterization, the VCJ must again be reset to driving mode (replace post, flip switch to high position, click **Reset 0** button). The first step is to give the subject a feel for the default gains. The subject may apply a force to the joystick in the forward, reverse, left, or right directions depending on which radio button is selected in Adjusted Gains. Data is displayed in 45-s intervals, and the subject should attempt to keep the meter within the red target area of the progress bar. This is the force the subject will need to apply to the joystick to drive at maximum speed. Selecting alternative forces (e.g., 90% default, 80% default, etc) indicate that the subject needs to press with that much less force (e.g., 9/10 or 4/5 times the default force, respectively) to

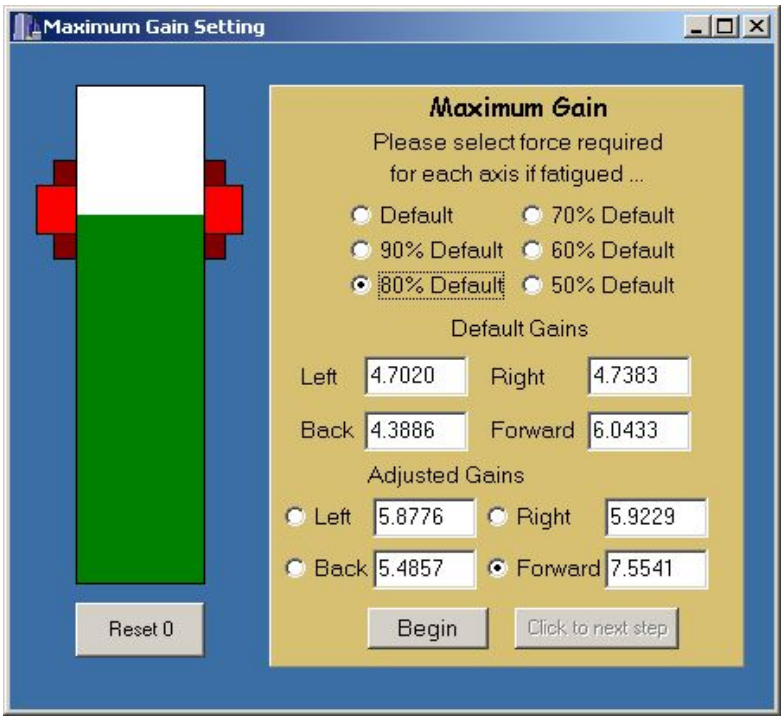


Figure 56: Maximum Gain Setting window.

reach the maximum possible speed with the VCJ. Again, the subject should attempt to keep the status bar within the red target area. If it appears too difficult to stay within bounds, it may be necessary to limit the maximum gain of the VCJ. While the default gains cannot be adjusted in the text window at this time, it may be an included feature in future versions.

Once the maximum gains have been determined, the clinician should minimize the program and begin *MSS Input Analysis*. This will give the subject some time to rest his or her hand before the third FI is calculated. Once the third FI has been determined, the clinician should note the fatigue adaptation parameters α and β for use with the other tuning routine. Clicking on **Finish** will write the personalized VCJ parameters to the file `C:\Settings\setup.txt`.

4.3.1.5 Setup File

The setup file, `setup.txt`, is the interface between the tuning application and the VCJ software. Parameters are contained in the first column, and descriptions are listed in the second column. Tabs separate the columns. If the VCJ is operating in compliant mode and the subject requires a diamond or square template, the bias angle should be adjusted to the nearest 15° increment (0, 15, 30... 330, 345, 360). Likewise, the physical template should be rotated to match. An example setup file is included in Box 1.

```
MSJS104     Subject identifier
1     Dead zone shape: 0 = no dead zone; 1 = ellipse; 2 = rectangle
98    Dead zone x-axis: a number from 0 to 500 (ignored if DZ shape = zero)
98    Dead zone y-axis: a number from 0 to 500 (ignored if DZ shape = zero)
1     Template shape: 0 = no template; 1 = ellipse; 2 = asteroid; 3=diamond
2048  Template x-axis: 100 to 2048 (ignored if template shape = zero)
2048  Template y-axis: 100 to 2048 (ignored if template shape = zero)
1     Bias axis status: 0 = none; 1 = active
11.040939   Bias axis angle: 0 to 360 degrees (ignored if bias axis status
= 0)
1     Gain status: 0 = off; 1 = on
4.702041    Gain -X: 1 to 20 (if no or overlimit, 1X will be used)
4.738303    Gain +X: 1 to 20 (if no or overlimit, 1X will be used)
4.388571    Gain -Y: 1 to 20 (if no or overlimit, 1X will be used)
6.043279    Gain +Y: 1 to 20 (if no or overlimit, 1X will be used)
5.877551    Max Gain -X: 1 to 20 (if no or overlimit, 1X will be used)
5.922879    Max Gain +X: 1 to 20 (if no or overlimit, 1X will be used)
5.485714    Max Gain -Y: 1 to 20 (if no or overlimit, 1X will be used)
7.554098    Max Gain +Y: 1 to 20 (if no or overlimit, 1X will be used)
1.250000    Gain Multiplier: 1, 10/9, 5/4, 10/7, 5/3, 2
1.769070e-06      alpha: 0 to ~1
1.369100e-05      beta: 0 to ~1
```

Box 1: Example setup file.

4.3.1.6 Source Code Implementation

The reader is referred to Appendix I for *MSS Tuning's* source code implementation. `Tuning_MS_Study.cpp` is the main function that initializes the SPU (formerly called the Input Control Device (ICD)) and the program's screens. The software retains the capability to read from the serial port and can be modified to collect data for shorter periods of time for debugging purposes. `Unit1` creates and controls the functions of the Personalized Deadzone window (**Figure 53**). `Unit2` creates and controls the functions of the Bias Axes, Gain, and Template Settings window (**Figure 55**). `Unit3` creates and controls the functions of the Gain Adjustment window (**Figure 54**). `Unit4` creates a small bar graph that provides feedback to the subject that enough force is being applied to the joystick during default gain determination and endurance analysis. `Unit5` creates and controls the functions of the Maximum Gain Setting window (**Figure 56**). `ICD_functions.cpp` interfaces to the NI-DAQCard to collect readings from the VCJ, where methods are similar – if not identical – to those described in section 2.4.2.3, with the exception that the development environment was Borland C++ Builder version 5 and functions are prefixed with `ICD*` instead of `DAQ*`.

4.3.2 MSS Input Analysis

The MATLAB routine *MSS Input Analysis* is designed to interpret data from previous driving sessions with the VDS software to determine optimum parameters for the tremor filter and to fine tune fatigue adaptation parameters. The program is executed by setting the current directory to the one that contains the analysis routines, typing

```
>>MSS_input_analysis();
```

at the command prompt, and following on-screen instructions.

The first question asks which trials should be examined after a list of available trials is printed in the command window. The clinician should select files with the same ID number as the subject's and should have "VGA" or "STF" in the file name depending on if the subject's joystick is in the isometric or compliant mode, respectively. Files are presented alphabetically and have a numerical identifier listed next to them, where the clinician can select the files by entering their ID numbers in array format in MATLAB (e.g., 5, 1:24, [1:3, 5]) (a warning about not finding an exact match for 'DB' may come up after pressing **Enter**, but it can be ignored).

The following steps allow the clinician to gain a better understanding of the subject's strength performance and potential tremor filter performance.

4.3.2.1 Strength Performance

Data from the baseline trials and the two-minute strength test from the tuning software are presented in four different formats for the clinician to gain a better understanding of the subject's strength performance and fatiguing characteristics. The first format is a bar graph of the subject's average force on the joystick during each virtual driving task. The bar graph is plotted after the clinician enters `1` (for yes) for the question, Should average forces be plotted. An example is presented in **Figure 57**. Here, the clinician should look for trends in the average force. While a decreasing average force could be evidence that the subject is learning how to use the joystick more efficiently, it may also be an indication of fatigue. Alternatively, the frequency spectrum may provide better insight into the subject's fatigue because increasing tremor power is an indicator of fatigue [79],[80].

The second format presents the data transformed into the frequency spectrum and creates a movie and a 3D plot of the spectrum for each trial. The movie and 3D plot are created after the clinician types `1` for yes after the question, Should individual episodes be plotted. Since tremor with MS patients typically exists within 3.5 to 5 Hz, the clinician should observe the tremor

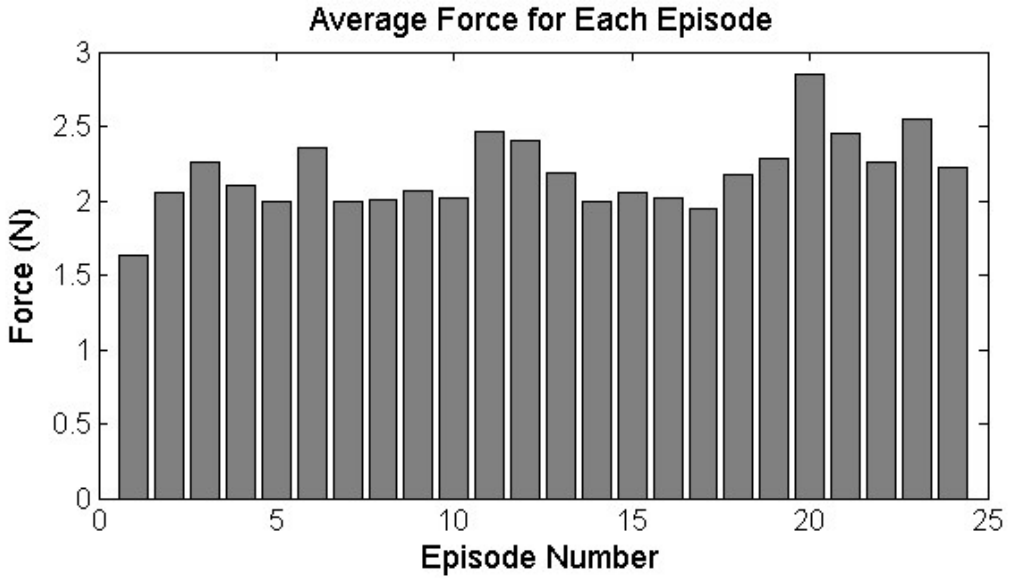


Figure 57: Average force applied to joystick for baseline virtual driving tasks.

power in the 3.5 to 5 Hz range as the movie plays. The clinician should also be mindful of the 8 to 12 Hz range because this is typically where a person may experience physiological tremor related to fatigue [79],[80]. The clinician may be able to observe a rate at which the tremor power increases. The 3D plot that replaces the movie, an example of which is presented in Figure 58, may be rotated with the mouse to help the clinician identify powers and rates over the course of the baseline data collection procedure. To watch the movie again, after exiting the *MSS Input Analysis* routine, type

```
>>load 'psd_movie.mat';
>>movie(F, n, fps);
```

at the command prompt, where `n` and `fps` indicate the indicate number of loops and frame rate of the movie, respectively.

The last format consists of plots of the applied force and the cumulated frequency spectrum from the subject's hand during the two-minute strength test. The plots are generated after typing `1` after the question, Should strength data be analyzed. An example is depicted in **Figure 59**. Here, the clinician may look for changes in force magnitude and the power in the frequency range of 3.5 to 5 Hz.

After tremor filter parameters have been selected and the third FI collected, impressions from the strength analysis may be used to update the adaptation and recovery parameters for fatigue adaptation. If it seems like the subject fatigues or recovers at different rates than those proposed in *MSS Tuning*, new values may be entered in their respective text boxes in *MSS Tuning*. The curve in **Figure 60** compares values for α and β with the time it would take to go from the default gain to 90% of the maximum gain and vice versa. For example, if it appears that the subject takes about 19 min to fatigue fully, a good value for α would be 2.0e-6. Likewise, if it seems like it would take 30 min for the subject's hand to recover fully, a good value for β would be 1.25e-6.

4.3.2.2 Selecting Tremor Filter Parameters

The VCJ uses two filters to mitigate the effects of tremor. The first filter is a high-pass filter and is intended to be a safety measure so that the second filter, the WFLC, does not remove intentional signals from the subject's input. The input analysis software creates three different plots to help the clinician tune the filter parameters.

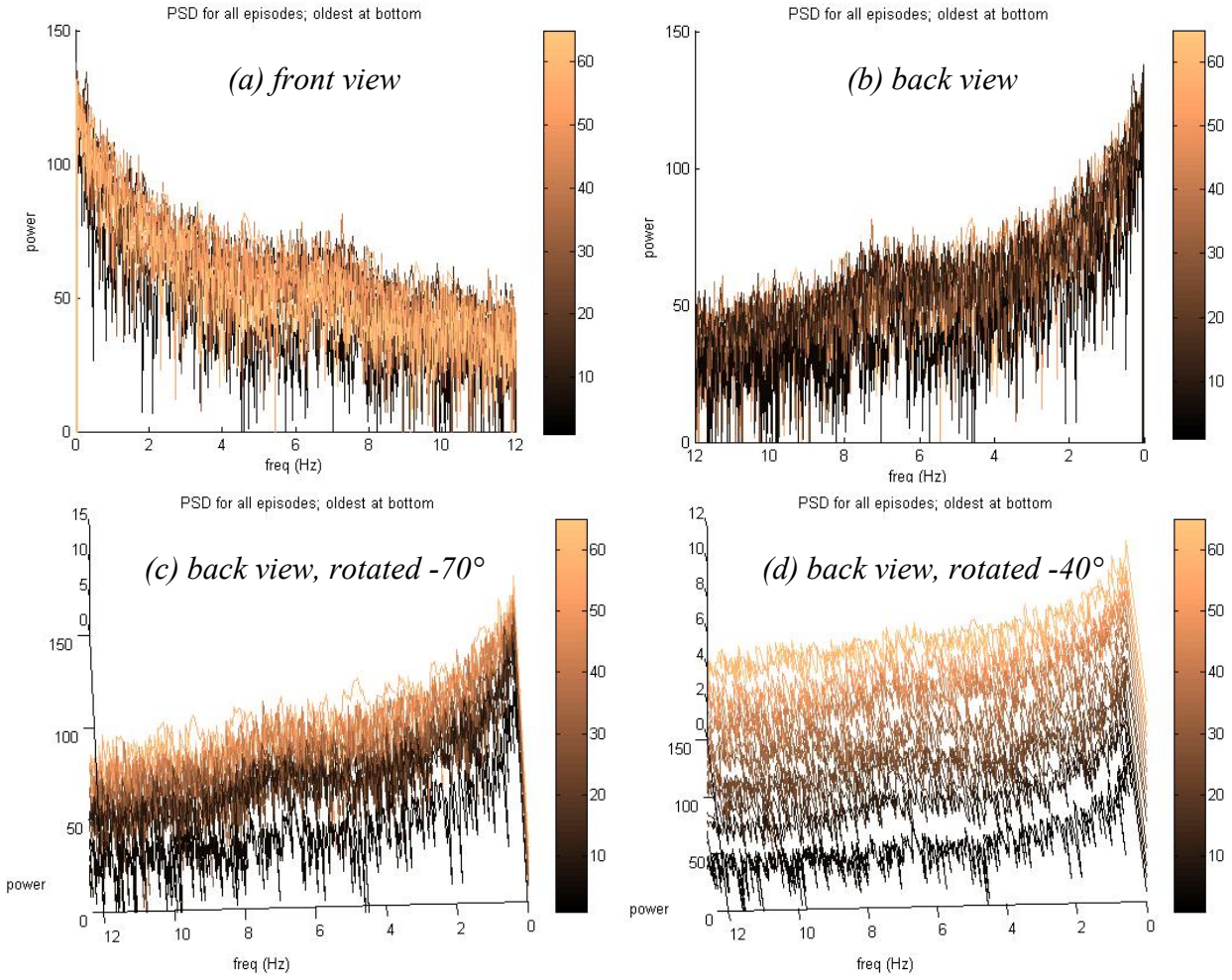


Figure 58: Multiple views of the power spectral density plots for the baseline virtual driving tasks.

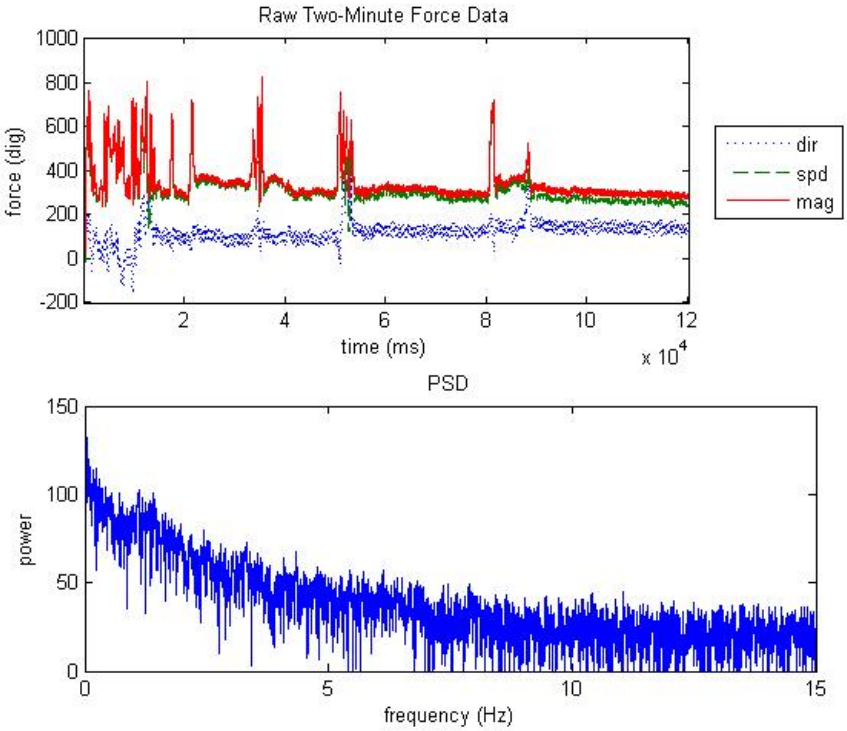


Figure 59: Strength data from 2-minute strength task.

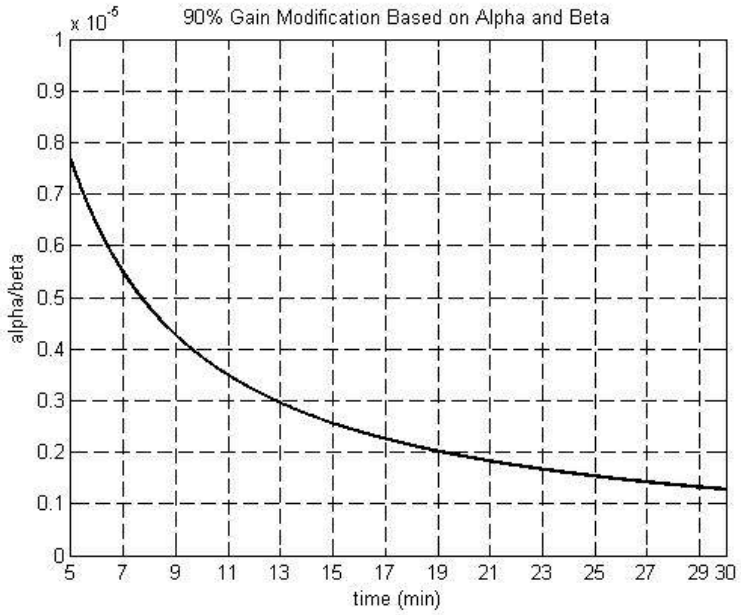


Figure 60: Relationship between alpha and beta and time to be within 90% of target gain.

The first two plots illustrate the subject's signal before and after the filters have been applied for the direction and speed axes. An example of the first plot, depicting the direction channel's signal, is provided in **Figure 61**. Blue lines indicate the subject's original signal, and red lines indicate what the signal would look like had the tremor filter been implemented. The clinician may click on each graph and drag the plots to see other segments of the driving data. The window represents 10 s worth of data. Frequency spectrum data is also presented. The third plot, an example of which is provided in **Figure 62**, illustrates what the WFLC observed as the tremor frequency over the course of the baseline driving procedures.

When analyzing the first two figures, the clinician should look at the amplitude of the tremor and the overshoot when the subject's force moves to a constant value. Ideally, the tremor amplitude in the red curve should be much smaller than in the blue curve. It should almost look like the red curve is a moving average of the blue curve, taking averages of points before and after the given point. Also, the overshoot when transitioning from a continuous force to a steady force should be minimized to the extent possible.

At the command prompt, the input analysis software asks the clinician if the filter parameters should be updated. If yes, the program prompts for the parameters in sequential order, with the high pass and direction axis parameters first. Values from the most recent iteration of the analysis are also provided for comparison. If the clinician miss-enters a value, parameters for each axis may be entered again.

Perhaps the first parameter to look at is the initial estimated tremor frequency. This may be re-evaluated based on an approximate average value or late value in the WFLC Frequency vs. time plot, **Figure 62**, or based on peak tremor frequencies observed in the strength analysis plots.

The remaining parameters are described in Table 11. While other parameters to the WFLC exist, changing them does not appear to improve the performance of the filter. The clinician may update parameters as many times as felt necessary in the time allotted.

Upon completing filter parameter analysis, the clinician is prompted to close all figures if desired and whether the filter parameters should be written to the filter's setup file. If the clinician approves of the parameters, they are written to the file `c:\settings\wflc_setup.txt`. Like the tuning setup file, this file is the link between the input analysis software and the VCJ's software. An example of the filter setup file is provided in Box 2.

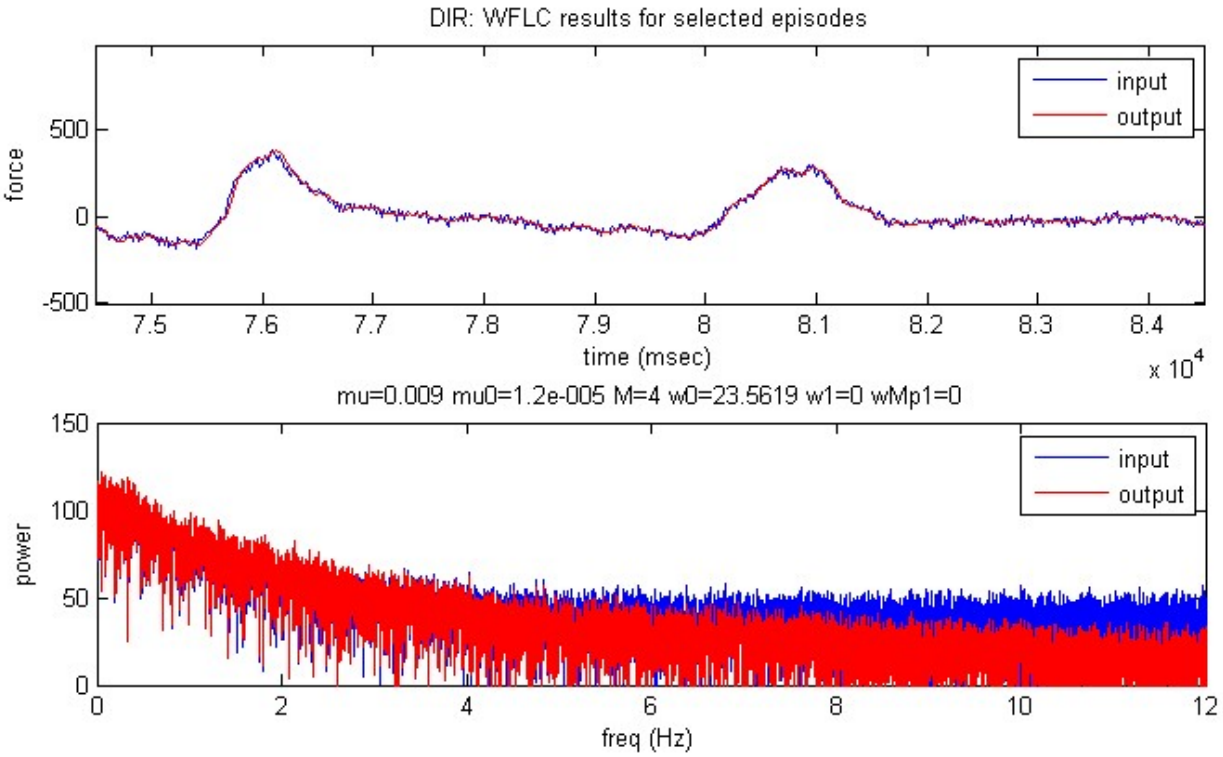


Figure 61: WFLC result for selected direction channel.

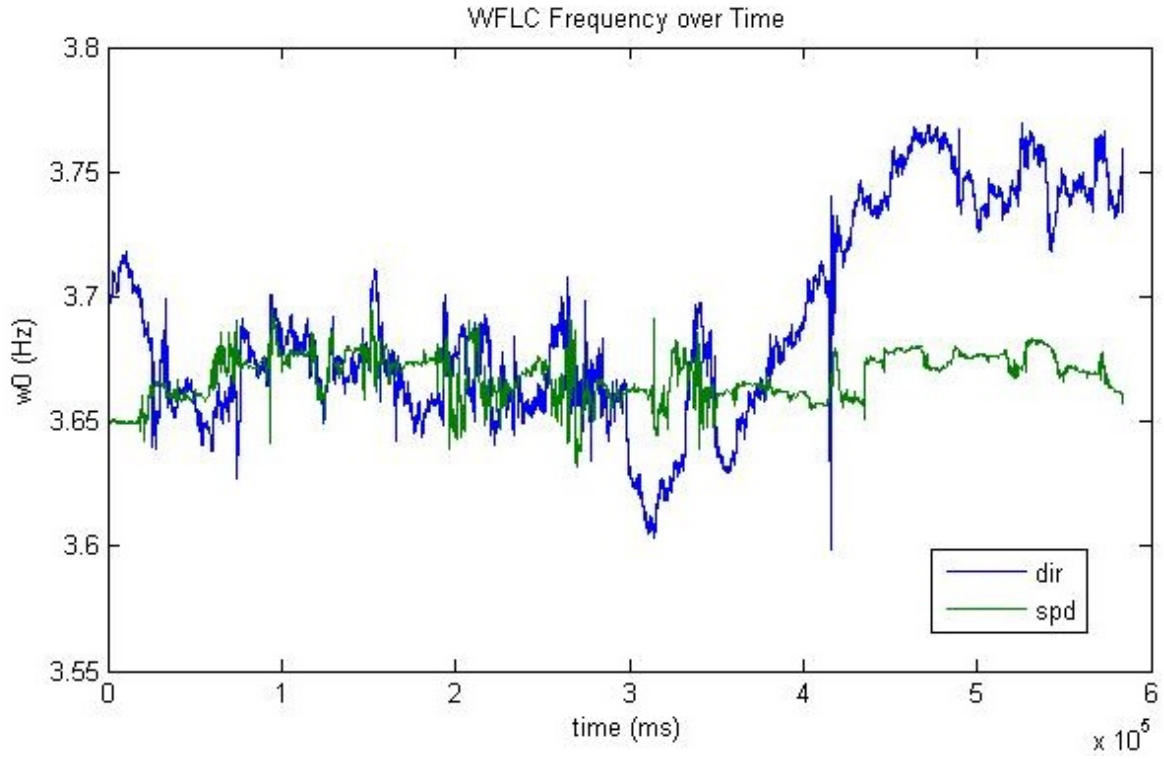


Figure 62: WFLC frequency vs. time.

Table 11: Definitions and acceptable ranges of the tremor filter's tunable parameters.

Parameter	Description
Order	Order of the high pass filter. Two or three is good.
Corner freq	Cutoff frequency for the high pass filter, frequencies below which should not be seen by the WFLC filter. If the initial estimated tremor frequency increases above 4.5 Hz, it may be wise to increase this value as well. It probably should be more than 2 Hz below the WFLC estimated frequency. If the subject does not appear to have tremor, set this number and the initial WFLC estimated frequency to a high value (>10 Hz but <30 Hz).
mu	Mu relates to the gain and bandwidth of the WFLC. If the subject has significant tremor, high values (>1.2e-2) may help, but overshoot may become a problem.
mu0	Mu0 is a secondary parameter on the frequency adaptation weight. This number should be kept low (between 1.0e-7 and 1.5e-5).
mub	This should be kept 0.
M	This is the order of the combiner. Four seems to work, but it may help to change it if changing other parameters are not effective. It should be kept below 7.
w0	This is the initial estimated tremor frequency. It should be located near the peak tremor frequency in frequency spectrum curves.

```
2           direction high-pass filter order
2.00000     direction high-pass cutoff frequency
59.39380    direction sampling frequency
1.000       direction high-pass filter gain
0.009000000 direction mu
0.000012000 direction mu0
0.000000000 direction mub
 4          direction M
22.99646    direction w0
0.000       direction w1
0.000       direction wMp1
 2          speed high-pass filter order
2.00000     speed high-pass cutoff frequency
59.39380    speed sampling frequency
1.000       speed high-pass filter gain
0.010000000 speed mu
0.000012000 speed mu0
0.000000000 speed mub
 4          speed M
25.13274    speed w0
0.000       speed w1
0.000       speed wMp1
```

Box 2: Example filter setup file wflc_setup.txt. Text descriptions are not included in the actual setup file.

4.3.2.3 Source Code Implementation

The reader is referred to Appendix J for the source code implementation of *MSS Input Analysis*. The main function is `MSS_input_analysis()`, which gathers the data, computes the forces and power spectral densities of the input data, displays the data, plots the results of the adaptive filter analysis, and writes the settings file for the high-pass and WFLC filters. The accessory functions, `MSS_get_trial_indexes()`, `driving_times`, and `MSS_get_new_filt_params()` allow the clinician to select which trials to analyze, truncate the data sets to only times when the chair was moving, and prompt the clinician to input new filter parameters, respectively. `MSS_wflc()` and `MSS_highpass_filter()` apply the WFLC and high-pass filters, respectively, to the input data.

5.0 METHODS

The research study "Application of Isometric Controls to Assist Individuals with Multiple Sclerosis in Driving Electric Powered Wheelchairs" [81] was developed to improve control of personalized mobility for those with MS by filtering tremor and compensating for fatigue. The study consists of two phases. Phase I takes place in a virtual environment, and Phase II takes place on an indoor driving course. A virtual environment provides a safe setting for training [82] and evaluation [35], and full control of the environment allows easy access to modifying joystick parameters. Because of similarities to performance in real world driving [35], driving performance in only the virtual environment will be considered for this thesis.

5.1 HYPOTHESES

The goal of the study is to provide preliminary analysis of the following hypotheses:

A. In a virtual driving environment while using the VCJ in compliant mode, within-subjects performance scores will be ranked based upon the control algorithm installed. The order in which they will be ranked, from significantly best to poorest, is as follows: MS_PFA_FA > MS_PFA > STF.
B. In a virtual driving environment while using the VCJ in isometric mode, within-subjects performance scores will be ranked based upon the control algorithm installed. The order in which they will be ranked, from significantly best to poorest, is as follows: MS_PFA_FA > MS_PFA > VGA.
C. Final performance scores by subjects using the IJ will be significantly better than those assigned the MJ.

Driving performance is a culmination of three different measures: mean time to complete a track, mean number of boundary violations, and RMSE off the target path. Mean time is defined as the time the subject applies a threshold force to the joystick to initiate chair motion to the time the

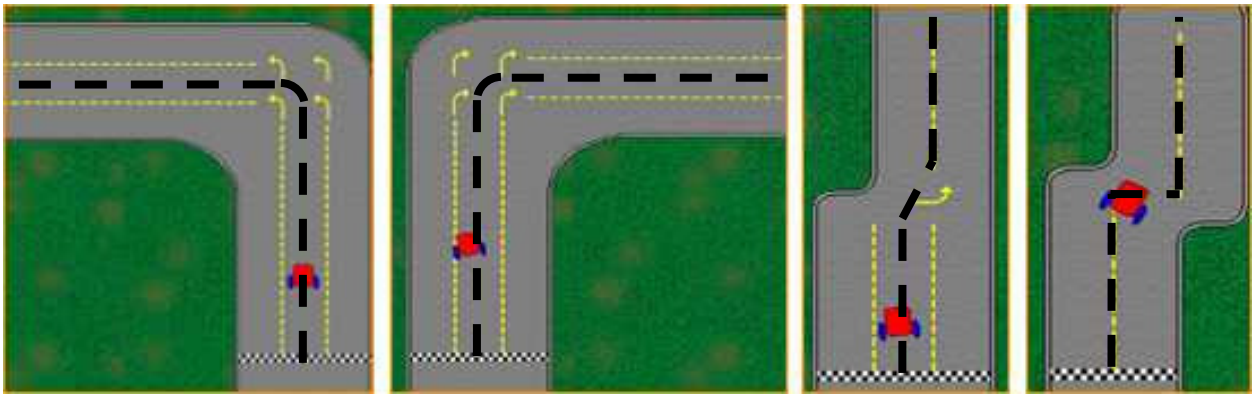
chair reaches the end of the track. Boundary violations are defined as instances in which the virtual chair makes contact with the wall causing the VDS to beep and the chair to be moved back to a previous location. RMSE is defined as the deviation from the target paths illustrated in **Figure 63**. Specifically,

$$22: \qquad RMSE = \frac{\sum_{i=1}^{n} \sqrt{x_i^2}}{n},$$

where x_i is the distance between the wheelchair's path and target path, and n is the number of data samples. The error, x_i, was measured every time the virtual wheelchair had traversed five pixels. Low mean times, low number of boundary violations, and small RMSEs are indicative of better performance.

5.2 STUDY DESIGN

Comparison of the joysticks (MJ vs. IJ) is a randomized, two-group, repeated-measures experimental study design. Subjects were randomly assigned to a group, where one group used the VCJ in isometric mode, and the other used the VCJ in its compliant mode, mimicking a MJ. To evaluate the personalization features, a repeated measures, within subjects scheme was used,



(a) left turn *(b) right turn* *(c) forward* *(a) docking*

Figure 63: The four virtual driving tasks: (a) left turn, (b) right turn, (c) straight forward, and (d) docking. The correct path is superimposed with thick, black lines.

where randomization and a washout period limited the order effect. Within each group, subjects first used the basic input control algorithm and then one of the personalized algorithms. The order in which the personalized algorithms were presented was randomized. After at least 2 days but no more than 10, subjects used the basic input control algorithm again and then the other of the personalized algorithms. Testing the basic input control algorithm twice served the purposes of checking the learning effect and controlling the amount of physical activity the subject underwent prior to using the joystick with personalized algorithms. Subjects were blinded to which personalization algorithm they were using during testing.

5.3 EXPERIMENTAL PROTOCOL

The research study was carried out in two testing sessions, where each session was broken into 30-minute segments. Driving performance was evaluated in a virtual driving environment, described in more detail in section 5.4.2. Subjects were reimbursed $40.00 for each testing visit.

5.3.1 Visit One

The first visit consisted of intake, baseline measurements, joystick personalization, and measurements with the personalized algorithms. After informed consent, the subjects completed a demographic questionnaire, included in Appendix K. Prior to their arrival, subjects were randomly assigned to one of four intervention options as provided in Table 12.

Table 12: Intervention path options.

Intervention Option	Joystick Type	First Algorithm	Second Algorithm
1	Isometric	MS_PFA	MS_PFA_FA
2	Isometric	MS_PFA_FA	MS_PFA
3	Compliant	MS_PFA	MS_PFA_FA
4	Compliant	MS_PFA_FA	MS_PFA

To obtain a baseline measure of performance, the subjects drove in the virtual environment with either the STF or VGA depending on which joystick mode they were assigned.

The virtual environment consisted of four tracks – left turn, right turn, forward maneuver, and docking maneuver – depicted in **Figure 63**. The correct paths are superimposed on the tracks with the black, dashed lines. The procedure for executing a task was explained, and the subjects drove each track three times for practice. The subjects were instructed that driving accuracy is more important than completion speed but that they should perform each maneuver as fast as practical without letting the onscreen chair stray over a boundary. The baseline data was then collected from driving the four tracks six times each, as presented in a balanced randomized scheme; every track appeared once before it was repeated.

Tuning of the joystick and its parameters was executed as described in chapter 4.0, with the exception that two separate joysticks were used for tuning parameters and characterizing fatigue. During mock trials, we discovered that switching the joystick post repeatedly was cumbersome and time-consuming. Therefore, one joystick was set up for driving and another for characterizing fatigue. The clinician still needed to click on the **Reset 0** button between driving and fatigue characterization modes since the bias voltages of the two joysticks may not have been the same. Following the tuning procedure, the subjects were given the opportunity to drive on a rectangular track to confirm that the settings were appropriate and to change them if necessary.

After the personalized settings were finalized, the subjects drove each of the four tracks again, six times each, as presented in a new balanced, randomized order. Depending on the intervention option to which they were assigned, subjects drove with either the MS_PFA or MS_PFA_FA algorithm installed on the joystick.

5.3.2 Visit Two

Visit two was conducted exactly as in visit one with the exceptions that the intake procedure was not performed and the alternate personalized algorithm was used for the second data collection battery.

5.4 INSTRUMENTATION

5.4.1 Testing Facility and Experimental Setup

Testing took place in the HERL Virtual Reality (VR) lab. See **Figure 64** for a photograph of the testing setup. The lab is equipped with a Matrix 3500 projector (Christie Digital Systems, Cypress, CA) and 6-ft by 8-ft projection screen. The subjects were positioned approximately 4.5 ft from the screen. Sound was provided with two Gateway 2000 (Altec Lansing Technologies, Inc., Milford, PA) speakers. The joystick was mounted to a sturdy frame with an apparatus that could position the joystick near the wheelchair's armrest in place of the subjects' joysticks. If the armrest on the subjects' wheelchairs needed to be raised or removed to position the VCJ, a substitute armrest was mounted on the rear of the mounting apparatus.

The Sony VAIO VGN-T370 laptop computer recorded the virtual chair's position and the forces applied to the joysticks. The time required to complete each maneuver and the number of boundary violations were also recorded. The same laptop was used to run the VDS.

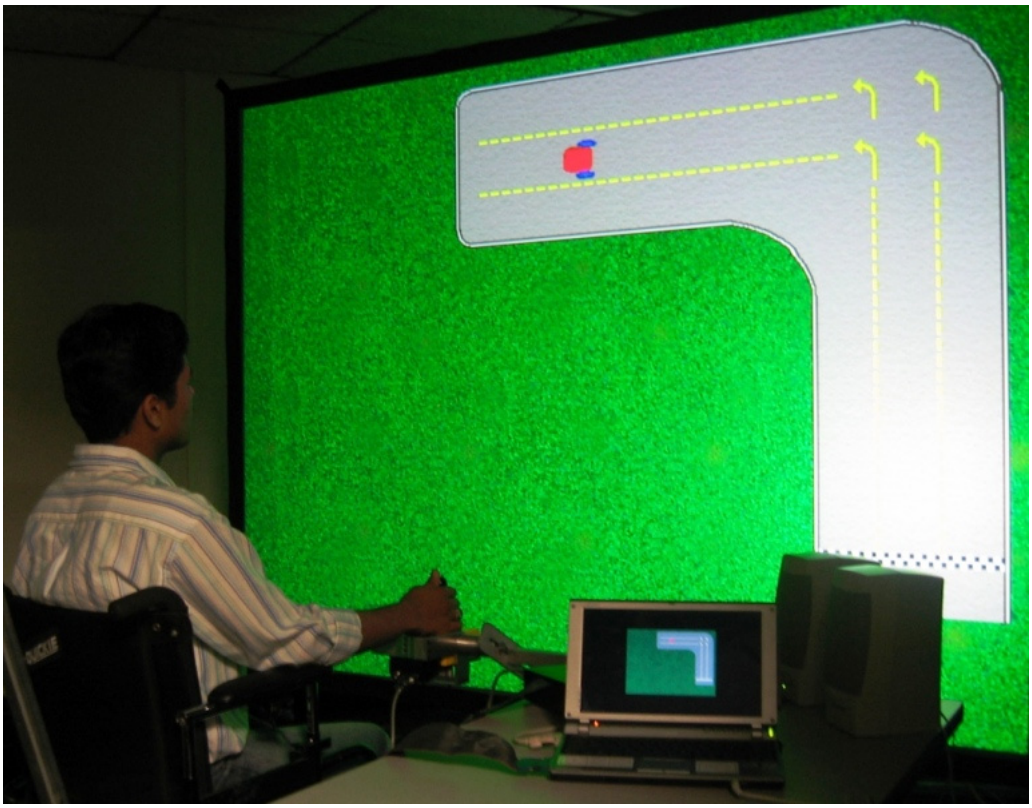


Figure 64: Photograph of the experimental setup.

5.4.2 Virtual Driving Simulation (VDS)

The VDS is based on the simulator developed for people with low visual attention span [70],[81]. Its dynamics are based on the acceleration profiles of a Quickie P300. Algorithm parameters may be updated while the program is running, and it features five different tracks: left turn; right turn; straight forward; docking; and track, which is in the shape of a rectangle. The first four driving maneuvers incorporate a variety of motions a person would perform during everyday driving. The left turn and right turn tracks involve gross motor movements including supination and pronation. The forward maneuver includes a gross motor movement with some fine tuning. And the docking maneuver incorporates fine motor movements including supination and pronation. While it may be possible for a subject to need to drive in reverse, the task is not called for specifically in this protocol. The fifth track involves gross motor movements and provides an environment for testing the personalized settings of the algorithms. Boundary detection may be turned off during this procedure. The tracks for the left and right turns are 120 pixels wide, the forward track is 120 pixels wide at the beginning and 80 pixels wide at the end, and the docking track is 80 pixels wide throughout. The chair is 40 pixels by 40 pixels, and 14 pixels represents about 1 ft.

While file naming conventions and techniques and data outputs were modified to suit the needs of the study, the most significant modifications revolved around controlling the loop rate of the VDS. The reason for controlling the loop rate was to ensure that the inputs to the digital filters, both the high-pass and WFLC, would have equal weights in time. Restructuring the loop, controlling the time at which data is acquired, and handling events when data acquisition was delayed ultimately produced a consistent loop period for the VDS software (17.0 ± 2 ms). The procedures introduce a few limitations such as glitches, reduced performance, and limited multi-tasking; but they do not seem catastrophic. A detailed analysis of the timing problem, alternative solutions, and the proposed solution is provided in Appendix L.

5.5 TEST SUBJECTS

The following inclusion criteria applied to those interested in participating in the study:

1. 18 to 80 years of age.
2. Diagnosis of MS. (Subject self-report)
3. Drive an EPW.
4. Able to reach and activate an armrest-mounted joystick.
5. Willing to travel to HERL.
6. Able to follow simple verbal instructions to complete study trials.

The following exclusion criteria also applied:

1. Not able to tolerate sitting for 2 hours.
2. Have any open pressure sores (self-report).
3. Unable to understand the procedures and provide informed consent.

Four subjects participated in the study. Their average age was 58.7 ± 5.0 yrs, and their average years since diagnosis was 28.2 ± 16.1 yrs. Table 13 provides demographic information about the subjects who participated in this study, and Table 14 lists which intervention option from Table 12 each subject received as well as driving characteristics such as EPW driving ability and driving hand. There was some ambiguity regarding the type of MS the subjects had. Subjects either had not heard the terminology used for this study, or were not sure because the "names keeps changing." Subjects then self-selected diagnoses after definitions were provided.

Table 13: Demographic information for subjects who participated in this study.

Subject ID	Gender	Ethnicity	Veteran	Years since Diagnosis	MS Type	EPW Drive Type
1	Female	Caucasian	No	24	Primary Progressive	Front-wheel
2	Female	Caucasian	No	47	Primary Progressive	Front-wheel
3	Male	Caucasian	Yes	8.5	Primary Progressive	Mid-wheel
4	Female	Caucasian	No	33	Primary Progressive	Mid-wheel

Table 14: Intervention option and driving characteristics of each subject.

Subject ID	Intervention Option	Driving Skill	Driving Hand
1	2	4	Right
2	3	4	Right
3	1	4	Right
4	4	4	Right

5.6 STATISTICAL ANALYSIS

Since four subjects had participated in the study, descriptive statistics were used to provide a preliminary glimpse of the data and identify possible trends. Means and standard deviations were calculated for the completion times and RMSEs, and the number of boundary violations per track and algorithm type were summed for each subject.

6.0 RESULTS

6.1 ALGORITHM SETTINGS

Personalized settings for the custom algorithms are provided in Table 15.

6.2 COMPLETION TIME

Mean completion times for all subjects are provided in **Figure 65** and broken out by track and algorithm type. Standard deviations are presented as error bars in the figures. Lower scores indicate improved performance. The VGA and STF algorithms are grouped together in the same algorithm since they provided the same basic programming for the isometric and movement joysticks. Visits one and two remain separated for each basic algorithm to provide insight to the existence of a learning curve.

To provide further insight into a learning curve, completion times for all trials with the Right Turn track are provided in the order in which they were presented in **Figure 66** and **Figure 67**. Joystick types, IJ and MJ, are provided in separate figures. Driving bouts consisted of driving the track six times with a given joystick configuration. While data was collected with a resolution of milliseconds, units of time in the figures are in seconds.

Table 15: Program settings for each of the subjects. Units for dead zone size are in N; and units for the template size are in terms of the maximum speed of the virtual EPW, where 2048 corresponds to 6.8 ft/s and 0.385 rad/s for the speed and direction axes, respectively.

Subject ID		**1**		**2**		**3**		**4**	
Program	**1=MS_PFA 2=MS_PFA_FA**	**1**	**2**	**1**	**2**	**1**	**2**	**1**	**2**
Dead Zone	Shape	ellipse	ellipse	rectangle	ellipse	ellipse	ellipse	ellipse	ellipse
	Size for Direction Axis	0.65	0.81	1.63	1.06	1.72	1.06	1.31	1.63
	Size for Speed Axis	0.49	0.81	0.65	1.06	1.72	0.98	1.22	2.53
Template	Shape	ellipse	ellipse	ellipse	ellipse	ellipse	ellipse	diamond	diamond
	Size for Direction Axis	1377	2048	2048	2048	2048	2048	1500	2048
	Size for Speed Axis	2048	2048	2048	2048	2048	2048	2048	2048
Bias Axis Angle (°)		none	none	347	none	none	none	none	none
Default Gain	Left	2.8	3	3.2	1.2	1.393	1.797	3.82	2.383
	Right	5.5	6	2.827	1.4	1.194	2.9	4.95	2.414
	Reverse	5.2	3.445	3.1	1.6	2.166	1.925	5.87	3.458
	Forward	6.55	5.009	3.93	1.2	2.167	1.367	5.49	2.008
Max Gain	Left	N/A	3.56	N/A	3.8	N/A	3.795	N/A	5.306
	Right	N/A	6.8	N/A	4.2	N/A	4.555	N/A	6.024
	Reverse	N/A	4.922	N/A	4.8	N/A	4.008	N/A	5.764
	Forward	N/A	7.155	N/A	4.2	N/A	3	N/A	3.346
Alpha		N/A	1.50E-06	N/A	3.00E-06	N/A	2.95E-06	N/A	2.00E-06
Beta		N/A	2.00E-06	N/A	2.25E-06	N/A	2.18E-06	N/A	3.13E-06

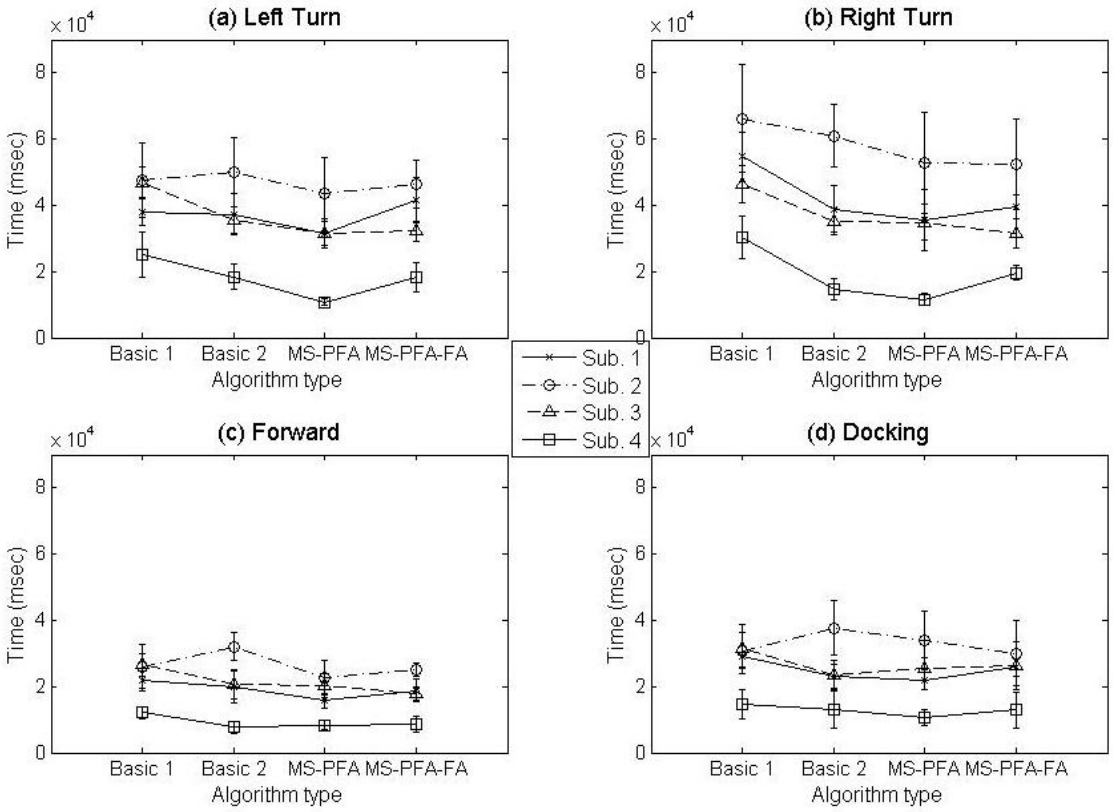


Figure 65: Mean time to complete each track for each subject and algorithm type per track.

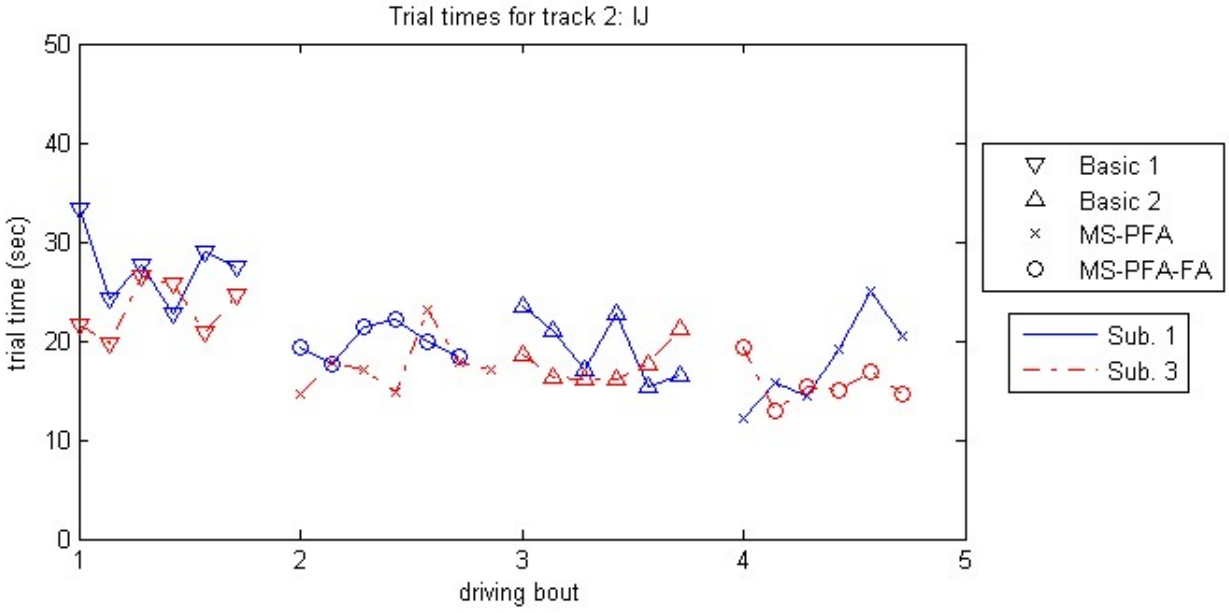


Figure 66: Trial times for each driving episode for the right turn track using the IJ.

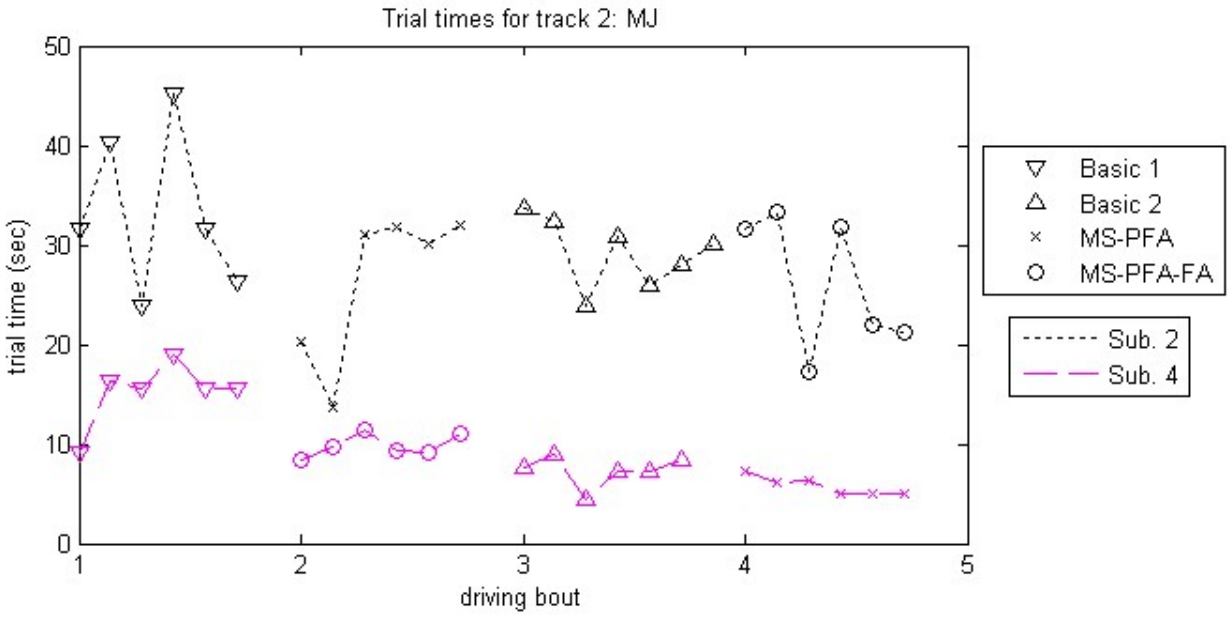


Figure 67: Trial times for each driving episode for the right turn track using the MJ.

6.3 RMSE

RMSEs for all subjects are provided in **Figure 68** are broken out by track and algorithm type. Again, standard deviations are presented as error bars in the figures, and lower scores indicate improved performance.

RMSEs for all trials with the Right Turn track are provided in the order in which they were presented in **Figure 69** and **Figure 70**. Joystick types, IJ and MJ, are provided in separate figures.

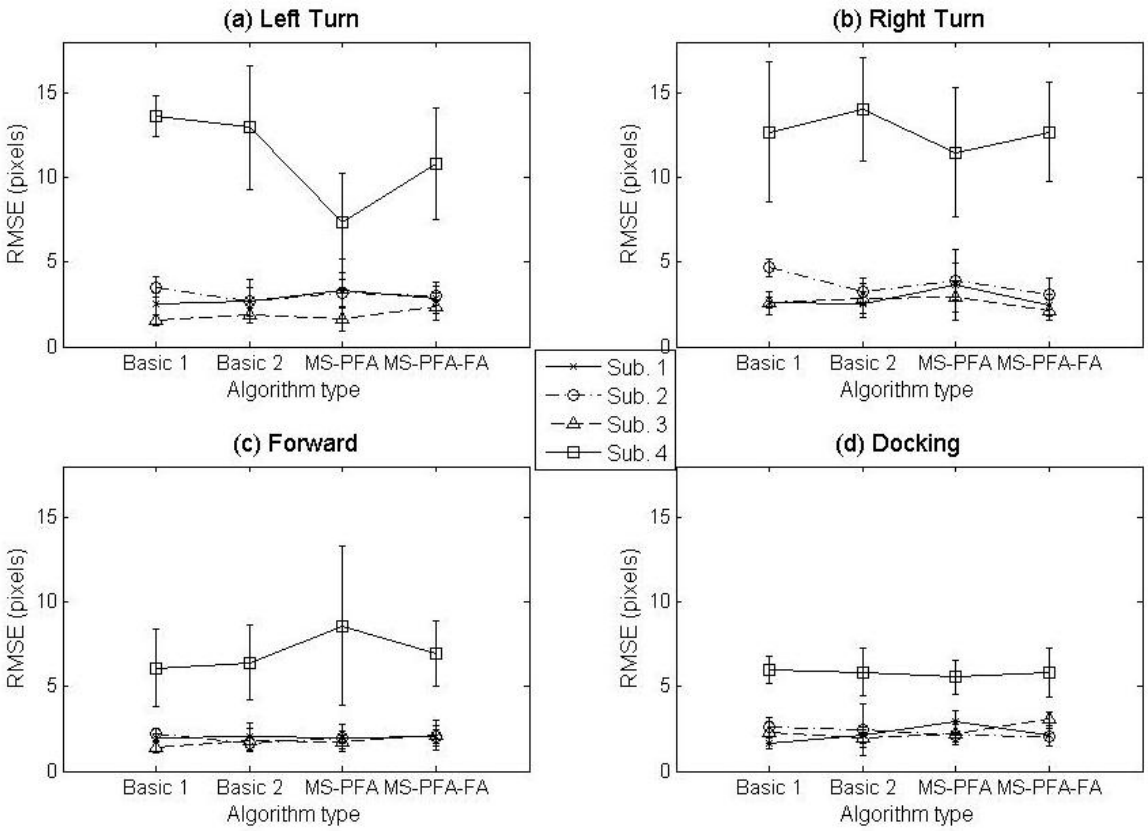


Figure 68: Mean RMSE for each subject and algorithm type per track.

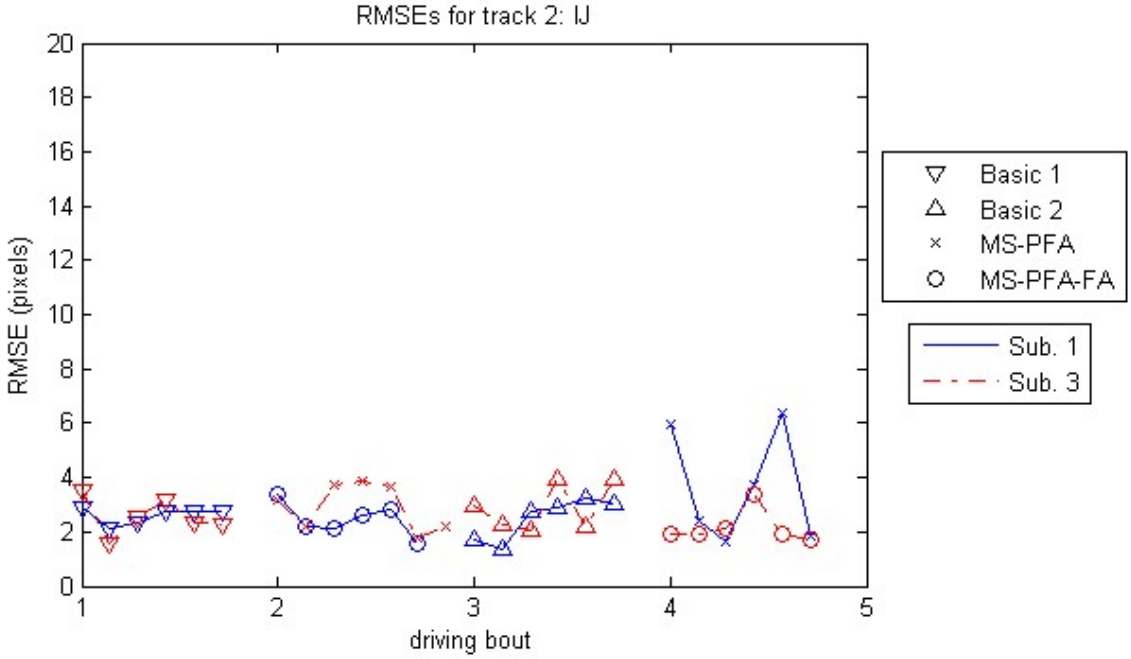


Figure 69: RMSEs for each driving episode for the right turn track using the IJ.

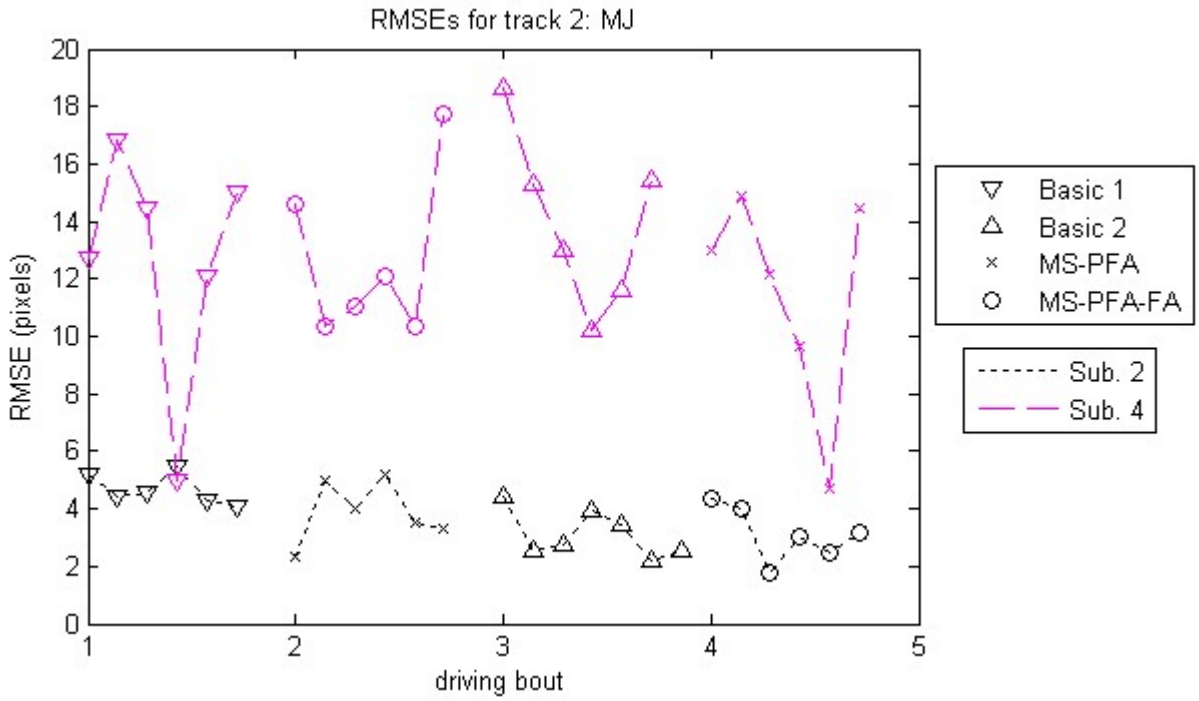


Figure 70: RMSEs for each driving episode for the right turn track using the MJ.

6.4 BOUNDARY VIOLATIONS

Boundary violations for all subjects per track and algorithm type are provided in **Figure 71**. Low scores indicate better performance.

The number of boundary violations for trials with the Right Turn track are provided in the order in which they were presented in **Figure 72**.

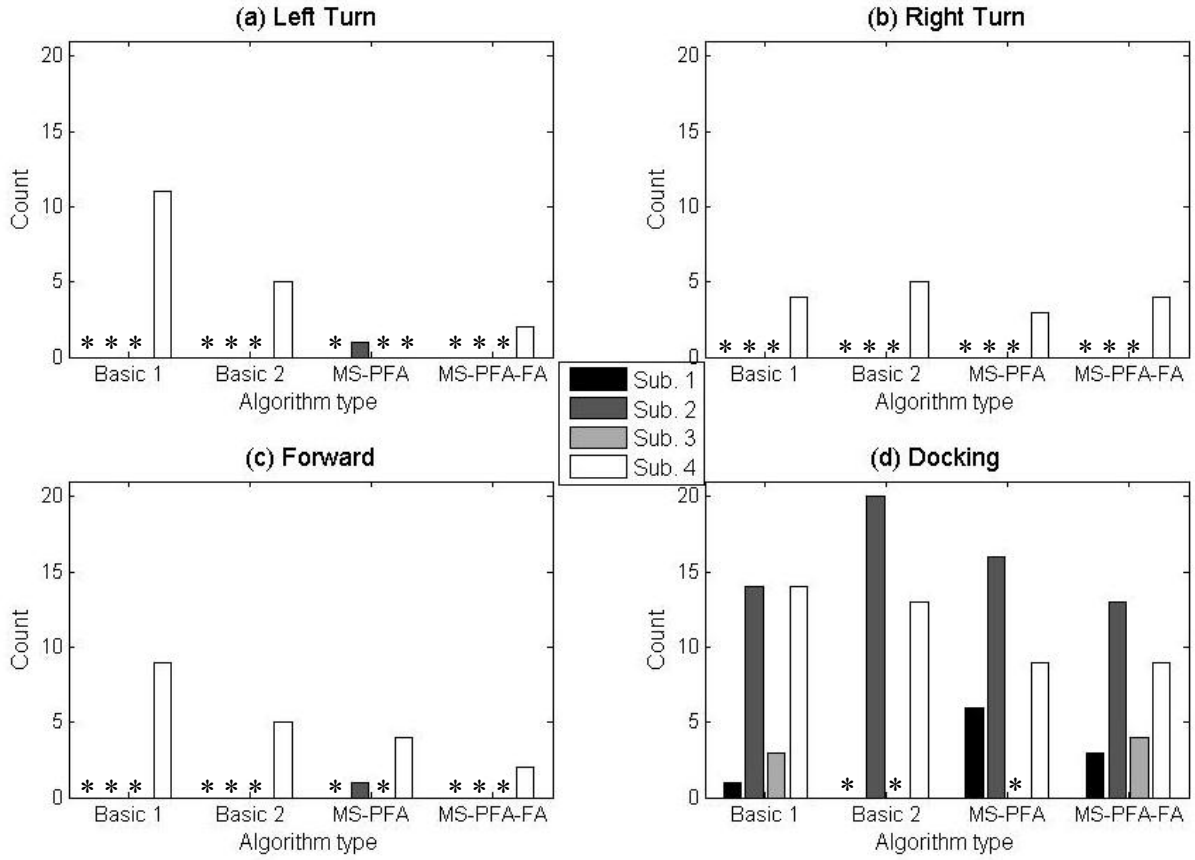


Figure 71: Total number of boundary violations for each subject and algorithm type per track. The violation for subject 2 for the left turn track is a false positive since this occurred at the very beginning of the trial, before the chair had started driving. Stars (*) indicate that no boundary violations were present for the given algorithm.

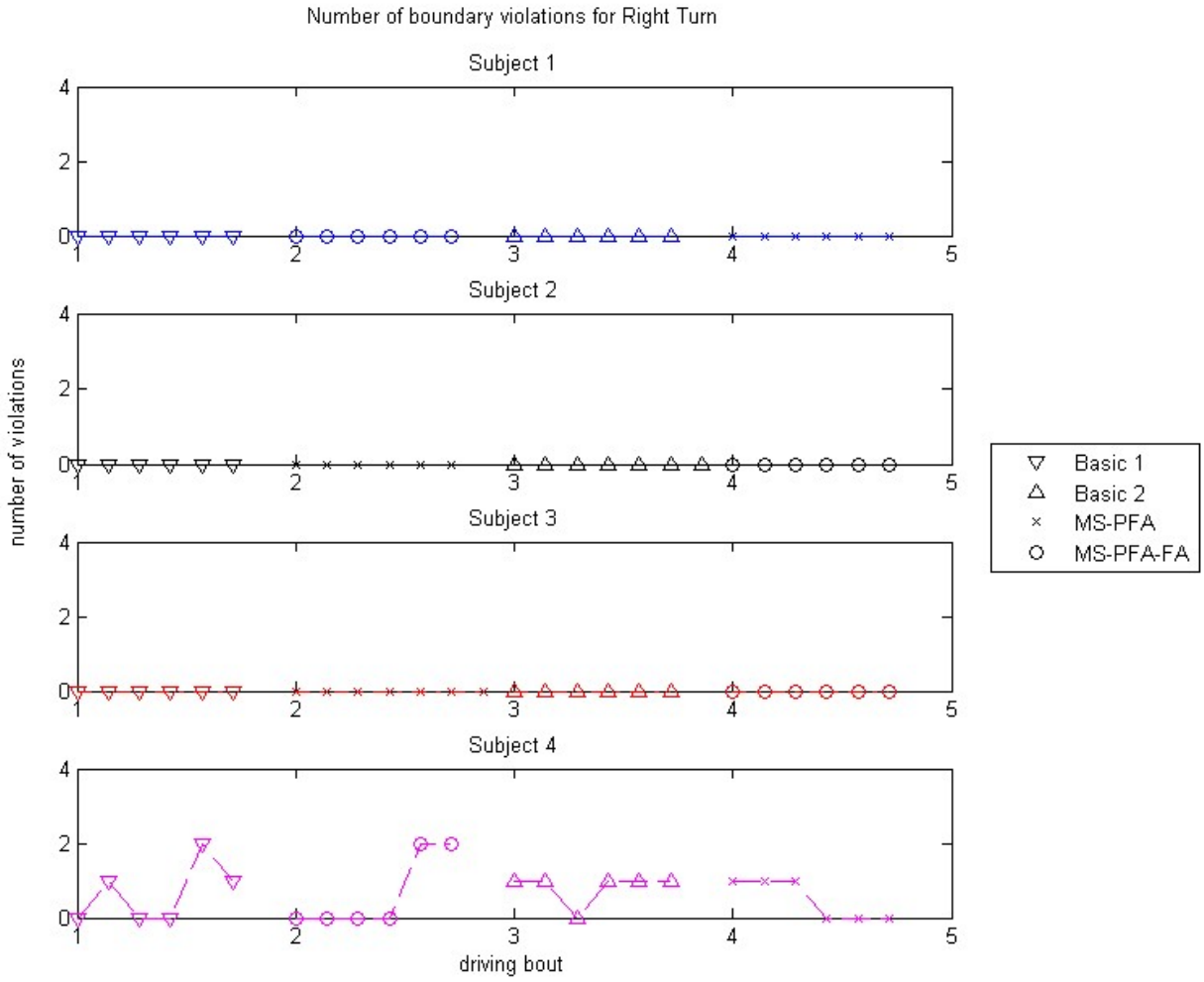


Figure 72: Number of boundary violations for each driving episode for the right turn track.

7.0 DISCUSSION

With only four subjects, drawing conclusions about the performances of the joystick types and the algorithms is difficult. However, few subjects also provides the opportunity to look at the data in more detail, observe trends that may become significant with more subjects, and provide initial feedback about the performance of the instrumentation.

7.1 ALGORITHM SETTINGS

As expected, no single configuration for the custom programs applied to everyone. Default gains ranged from very low (1.2) to high (6.55). The gain in the right direction (wrist extensors) was higher than that in the left direction (wrist flexors) in 75% of all cases. Elliptical dead zone and template shapes were chosen most often, though a rectangular shape for the dead zone and diamond shape for the template were also used. Settings were also not consistent between algorithms for individual subjects, suggesting that optimal settings for an individual could vary from day to day. Gains for the MS_PFA program, for example, ranged between -69 to 143% different from those for the MS_PFA_FA program (average ± standard deviation was -21 ± 52%).

7.2 PRELIMINARY ANALYSIS OF TEST MEASURES

7.2.1 Trial Time

Preliminary observations of the time to complete each trial indicate that the learning curve may be longer than anticipated. The mean time to complete a task with the basic algorithm on the second visit was less than that of the first visit for 75% of all cases based on inspection of **Figure**

65. There also appears to be a gradual decrease in completion time over time in **Figure 66** and **Figure 67**, especially for subjects 1, 3, and 4. Factors that could increase learning time include the novelty of the virtual environment, the dynamics of the virtual chair, and the novelty of the control interface. Since the VDS is a bird's-eye-view of the wheelchair, the subjects needed to make a mental transformation of the chair, especially while driving it across the screen, which would require some familiarization time. It would thus be interesting to compare the time to travel up the screen with the time to travel across the screen for the left and right turn tracks. Also, two subjects' EPWs were mid-wheel drive, and two subjects' EPWs were front-wheel drive. The virtual wheelchair, however, is intended to simulate a rear-wheel drive EPW, where its turning rate decreases as its speed increases. A difference arises because mid-wheel drive chairs are more responsive than front- and rear-wheel drive chairs [33]. Indeed, subject 4, who uses a mid-wheel drive EPW, commented that the virtual wheelchair "overcompensates." That is, the chair would overshoot the target trajectory after coming out of a turn. Lastly, three of the four subjects commented that the joysticks were very sensitive, where a small input would result in a large output velocity for the virtual chair. Thus, they needed to adjust the forces they typically would have applied to their personal chairs for the virtual chair. Subject 3 commented that he typically relies on the template for turning tasks such as in the left and right turn tracks, and that he needed to find a new method with the isometric interface.

Trial times for the left and right turn tracks are typically longer than those of the forward and docking tracks, which can be expected since those tracks are about twice as long. Trial times for the right turn appear longer than those for the left turn, especially during the first presentation of the basic algorithm for all subjects and all algorithm presentations for subject 2. In Jonker *et al.*'s study of muscle activity for individuals with MS during EPW driving [84], they found wrist flexors to experience more fatigue than other muscle groups activated during EPW driving. Since all subjects were right handed, the right turn would require using wrist flexor and would thus cause the activity to be more difficult.

7.2.2 RMSE

With the exception of subject 4's left turn, RMSEs for the subjects do not appear to improve with the personalized algorithms. Closer inspection of subjects 1 through 3's data show that some

may have improved scores with the MS_PFA, and others have better scores with the basic algorithms. Average RMSEs appear to be the lowest for the straight forward track, and subject 4 has much higher RMSEs for the left and right turn tracks than the straight forward and docking tracks. These differences could be related to the track widths. An average RMSE of 11 pixels amounts to 9.2% of the track width for the left and right turn tracks, and an average RMSE of 6 pixels amounts to 7.5% of the track width for the docking track. If the track was wider, the subject may have felt more free to vary from the target path.

7.2.3 Boundary Violations

While the personalized algorithms had fewer total boundary violations than the basic algorithms, the most notable feature is that the docking track had much more boundary violations than the other three tracks. This very likely may be related to the width of the track, the deceptive nature of the boundaries, and the tendency for violations to string together at low velocities. A narrow track with turns increases the need for fine motor control, otherwise the chair would run into a boundary. A high number of boundary violations for the docking track is consistent with Jonkers *et al*. [84], who found that patients with MS perceived tasks that require fine motor control to be more difficult.

The deceptive nature lies in the fact that the VDS software treats graphics such as the track surfaces or wheelchair sprite as square tiles instead of the contoured shapes they may appear to possess. For example, though the corner boxed in **Figure 73** appears rounded, the VDS software treats it as a right angle. Also, if the wheelchair sprite is rotated as in **Figure 73**, the sprite's boundaries are not located at its wheels, but along the edges of its tile. Thus, a collision may occur even if it does not appear like it will happen.

On several occasions, the VDS perpetuated boundary violations because of its collision handling routine. When a collision occurs, VDS software moves the sprite to a position the chair had held a fixed number of loop iterations prior to the collision. If the chair was driving fast prior to a collision, it would be repositioned far from the boundary. But, if the chair was moving sufficiently slow, the collision routine may not move the sprite far enough backwards, causing a single boundary violation to perpetuate. On two occasions, subjects were instructed how to

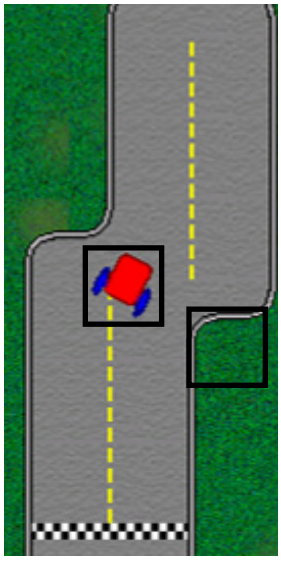


Figure 73: The VDS representation of virtual wheelchair and track boundaries is often wider than they appear.

command the chair to escape a loop; and on three occasions, trials were repeated because the chair was "stuck in the weeds."

7.2.4 Across the Measures

Since the trial time, RMSE, and number of boundary violations are all intended to be indicators of driving performance, one would expect some cross-talk between the measures. This is especially the case with subject 4, who had the fastest times but also had the most deviation from the target path. This is consistent with The Steering Law proposed by Accot and Zhai [85], where completion time T_C is roughly inversely proportional with the track, or path, width W. In the case of straight paths, for example, the completion time would be

$$T = a + b\frac{l}{W}, \qquad (23)$$

where a and b are constants and l is the length of the path. Thus, for subject 4, who had high deviation, or a large W, she also had low times T. Performance measures for subject 2, however, seemingly contradict the Law. While subject 2 had the longest trial times, the subject also had high RMSEs compared to subjects 1 and 3. This, however, may be related to the progression of her MS because subject 2 had been diagnosed the longest.

While conclusions regarding the type of interface, MJ vs. IJ, cannot be drawn at this stage, subject 3 raised an interesting point regarding the isometric interface. Specifically, he mentioned that the joystick was difficult to use because the lower part of his right hand was numb (i.e. his medial hand and digits four and five). He further commented that the numbness was worse following the FI measurements and that it took some time for feeling to return. Numbness is a typical symptom of MS (see Table 1), which should be noted when considering isometric interfaces for individuals with MS, especially when researchers have previously commented that lack of proprioceptional feedback may diminish control with an IJ [65],[69]. Therefore, joystick interfaces that offer at least some level of compliance may be superior devices with MS.

Subject 4 appeared to have significantly improved performance with the MS_PFA, especially for the left turn track, compared to the other algorithms. For the left turn, the subject had the lowest average trial time, lowest RMSE, and fewest number of boundary violations. While this may be a result of the learning effect – the MS_PFA was the last algorithm the subject used – more aggressive settings were installed on the joystick for the MS_PFA than MS_PFA_FA. The template shape for the MS_PFA was a diamond. And, the maximum value for the template along the direction axis was also limited to 1500 with the MS_PFA, while it was not changed with the MS_PFA_FA. The diamond template should limit signals with both high speed and turning rates, and the decreased template size should reduce the virtual wheelchair's turning rate by about 25%. Together, these changes could be the source for the improved control because the chair would not have been able to deviate from the straight path as easily.

7.3 PERFORMANCE OF THE INSTRUMENTATION

Four different and relatively novel technologies supported this research study: the Variable Compliance Joystick, the tuning applications *MSS Tuning* and *MSS Input Analysis*, and the Virtual Driving Simulation. While the instrumentation performed successfully – all the necessary variables for testing the hypotheses were collected without any adverse events – we experienced a few hiccups along the way, and there may be areas for improvement to facilitate consistency and reliability during the remainder of data collection. The rest of this chapter will address

specific concerns regarding the performance of each technology raised during testing with the first four subjects.

7.3.1 The VCJ

Prior to testing one subject, both channels of the VCJ in compliant mode and one channel of the VCJ in fatigue characterization mode stopped responding. Instead of outputting approximately 0 mV with no input, their output was a constant positive or negative 4.6 V. Tuning the potentiometers helped to some extent, but the input to output ratio was very low, and the resulting V_{REF} voltage for the instrument amplifiers was approximately positive or negative 3.8 V. Replacing the instrument amplifiers on the affected channels fixed the symptom and restored the original voltage range of 0 ± 500 mV. The root problem was very likely static discharges when the joysticks were connected to the mounting bracket. Connecting a wire between the armature of the mounting bracket with a ground pin on the NI-DAQCard seems to have fixed the problem.

However, a few days later, the chair became difficult to drive in the left and reverse directions with the STF and VGA algorithms. It was not clear whether the problem was software or hardware because the joystick would still output voltages in the expected ranges, and no software changes had been made. In software, computing the force for left and reverse signals involved evaluating a fourth order polynomial, for which a special function was written (`ICD_polyeval()`). While the relationship between the digital reading and force input may have been closer with a higher order polynomial, the mere fact of increased complexity may have contributed to the problem. Therefore, the polynomial was replaced with a linear best fit regression for both directions, where `force = 0.0113×(digital reading)` and `force = 0.0129×(digital reading)` for the left and reverse directions, respectively. The chair then began to drive within expectations after the change.

One tedious feature of the VCJ is that it needs to be reconfigured before every subject who uses it to emulate a MJ. That is, it needs to be set up in the mill and the lock nut adjusted such that an 8° tilt corresponds with 2.20 ± 0.05 N. This may cause some concern for future subject testing, especially since the electrical connection of the digital force meter is becoming less reliable.

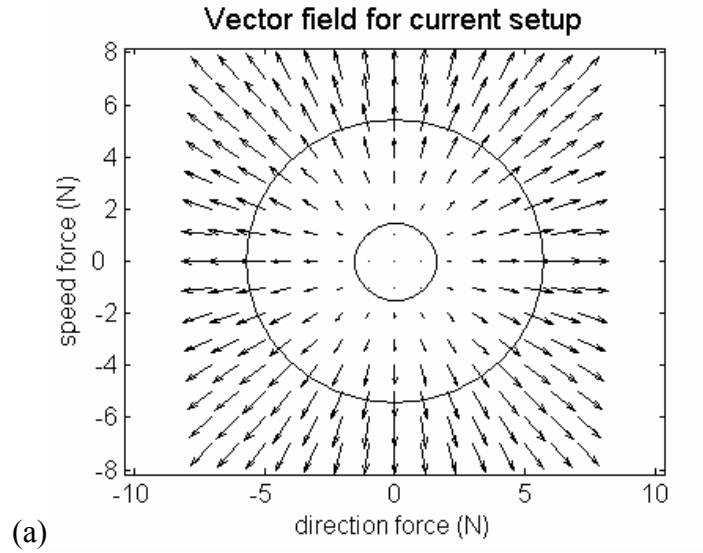
7.3.2 Personalized Algorithms

While the custom algorithms performed as they were intended for the purposes of this study, alternatives exist to how to adapt to fatigue and how to enhance control with the the VCJ in isometric mode. In this study, software adjusted the joystick's gain to adapt to fatigue, where increasing the gain resulted in lower force requirements to command the chair. Consequently, the template parameter was used to control the speed and turning rate of the wheelchair. This method, however, is conceptually inconsistent with the definitions of user interfaces and target devices. Since the VCJ is a user interface and not a target device, the template parameter, for example, should describe the force the user needs to apply to reach maximum speed rather than the actual maximum speed of the wheelchair. With fatigue adaptation, software should then control the the template boundary directly to alter the force requirements to command the chair. A second parameter would be used to define the maximum speed or turning rate of the wheelchair, and the gain becomes a derived parameter that maps the user interface's input domain to the target device's selection set.

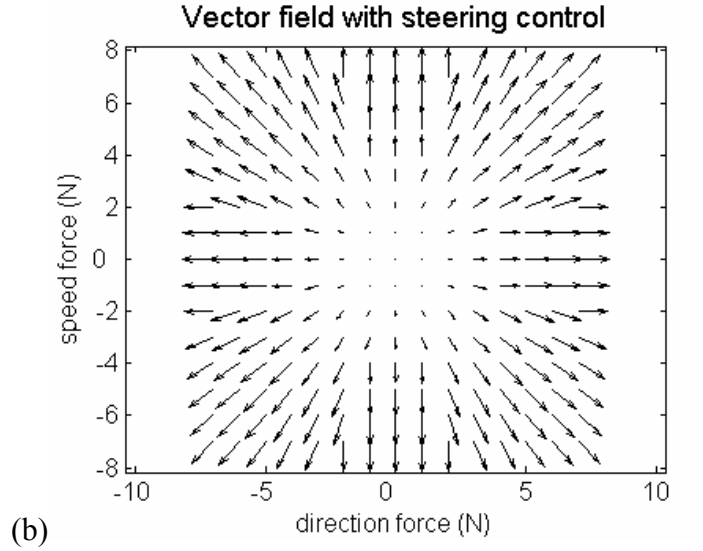
Just as mechanical templates help clients steer the wheelchair to desired heading with a MJ, vector fields may be used to help clients steer to a desired heading with an IJ. Though typically used to describe solution curves for differential equations [86], a vector field simply demonstrates how a set of inputs map to their respective outputs. Presently, the vector field associated with the IJ can be defined with the following function:

$$24: \qquad V(x,y) = \begin{cases} (0,0) & F < F_{dz} \\ \left((F - F_{dz})\cos\theta, (F - F_{dz})\sin\theta\right) & F_{dz} < F < F_{template} \\ \left((F_{template} - F_{dz})\cos\theta, (F_{template} - F_{dz})\sin\theta\right) & F_{template} < F \end{cases},$$

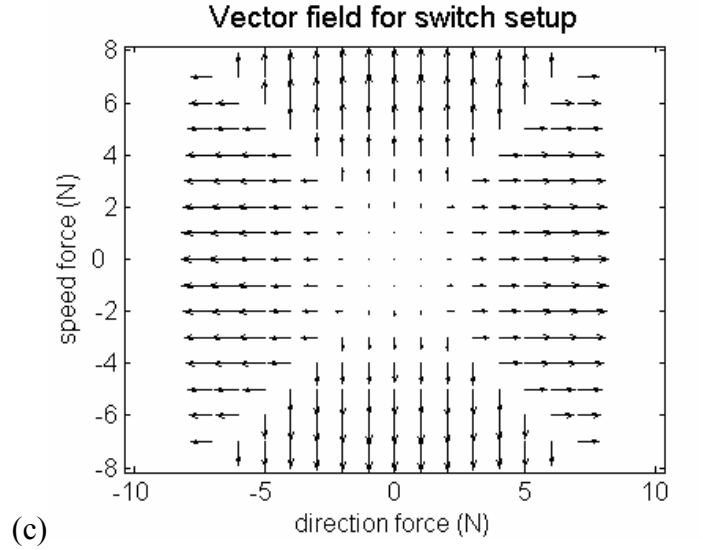
where F is the magnitude of the input force, θ is the input angle, F_{dz} is the dead zone force, and $F_{template}$ is the force required to reach the template. The vector field is graphically depicted in **Figure 74a**. Forces below the dead zone force result in null output vectors, and forces outside the template result in output vectors that have the same magnitude as those at the template. Additionally, it seems possible to define regions where the directions of the output vectors are the same. For example, it may be easier to steer the chair straight if all signals within ±15º of each axis resulted in the direction of that axis (further investigation would reveal if the transition



(a)



(b)



(c)

Figure 74: Vector fields for (a) current setup, (b) steering control, and (c) switch control.

out of the “straight” region should be smooth). At the extreme, the joystick could be used as a switch device, where only forward, reverse, left, and right commands are permitted, though with varying magnitudes. Graphical depictions of such vector fields are provided in **Figure 74b** and **Figure 74c**, respectively.

7.3.3 Tuning Applications

The tuning routine was intended to last approximately 30 min. However, the average ± standard deviation time to complete VCJ tuning was 61.5 ± 8.9 min. The segments of the tuning routine that took significantly longer than expected were the fine tuning of the maximum gain parameters, optimizing the WFLC, and switching between joysticks for driving and fatigue characterization modes. Fine tuning the maximum gain took longer than expected because more time was required for subjects to become familiar with the display screen, and we double-checked the gain adjustment for all four directions. Usually, the left and right directions were much more sensitive to increases in gain than the forward and reverse directions. When optimizing the tremor filters, it was originally configured that the WFLC would be applied to the complete set of 24 trials with the basic algorithm. However, the processing time for this step was substantial; and the program was modified to make it easier to change data sets. Since tremor is thought to be more significant in a fatigued state [79],[80], only the last 12 trials were analyzed with *MSS Input Analysis*. Still, it takes about two to three minutes to process the filter each time parameters are altered. Even though we had used two separate joysticks to facilitate switching between driving and fatigue characterization modes, switching the joysticks still was a little cumbersome and was subject to human error, where we did not reconnect the serial connector to the VCJ after switching joysticks on a couple of occasions. One instance simply required restarting the VDS program but another required a repeated FI measurement to be taken.

Even though it lasts a little over 30 s, measuring the FI three times seems to be a costly measure to determine fatigue adaptation and recovery rates. Subject 1 mentioned that her arm was sore two days after the first visit, and subject 3 mentioned that it took some time for his hand to recover from each test. With such a cost, its benefit comes into question. And with the first four subjects, only 25% of the rates that the tuning software calculated were used because the computed rates were either negative or seemed too slow. Also, the pattern that was expected – a

high FI followed by a low FI followed by a medium FI – does not appear evident based on graphical analysis of the first four subjects' FIs, nor does the method seem very repeatable (see **Figure 75** through **Figure 78**). Four subjects are still too few to make conclusions, and the trends associated with the three measurements of the FI should be investigated further as more subjects are tested. Alternative methods for determining fatigue adaptation and recovery rates could include using only one FI measurement or taking the second of two measurements since subjects appeared to take the test differently after the first measurement.

7.3.4 The VDS

Subjects noted that the joysticks – both IJ and MJ – were very sensitive when the VGA or STF algorithms were installed. These comments may be a reflection of the virtual environment, or they could be a reflection of differences in their chairs with the virtual wheelchair. The visual resolution of the wheelchair sprite was increased for this study from 10° to 3°, but the subjects still have only visual feedback to judge the inertial properties of the chair. Thus, the chair may

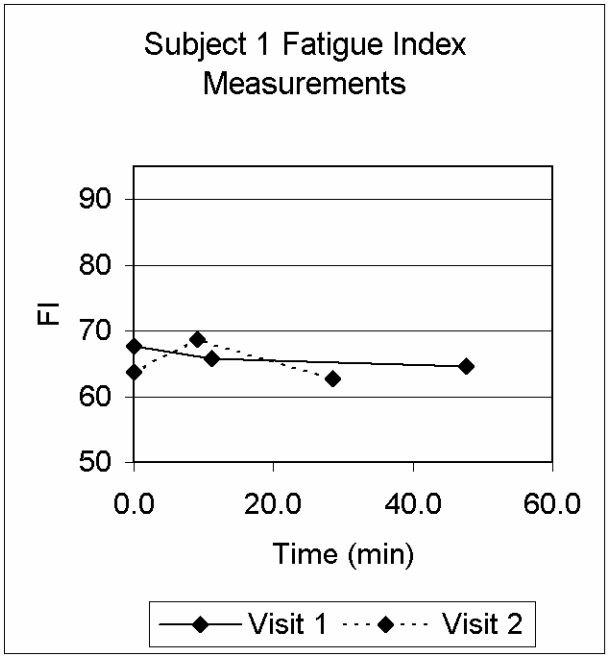


Figure 75: FI measurements for subject 1's first and second visits.

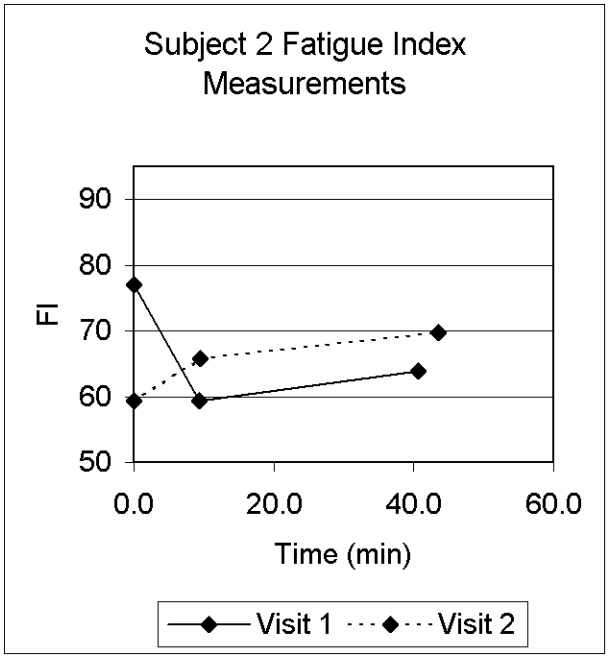


Figure 76: FI measurements for subject 2's first and second visits.

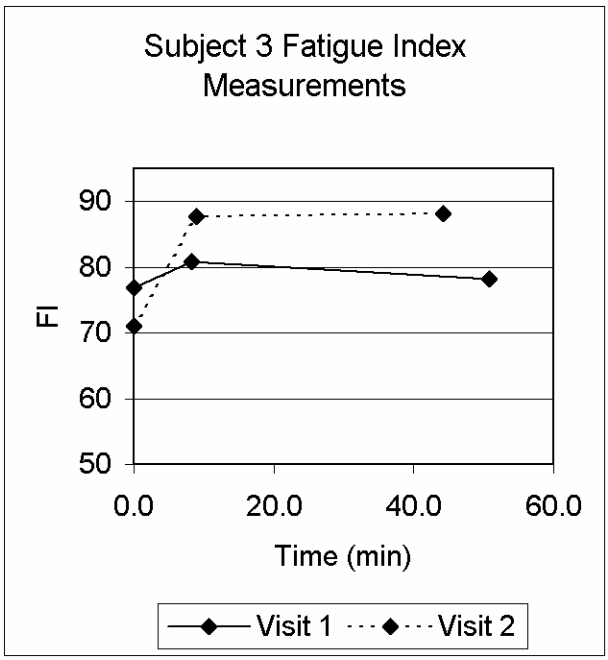


Figure 77: FI measurements for subject 3's first and second visits.

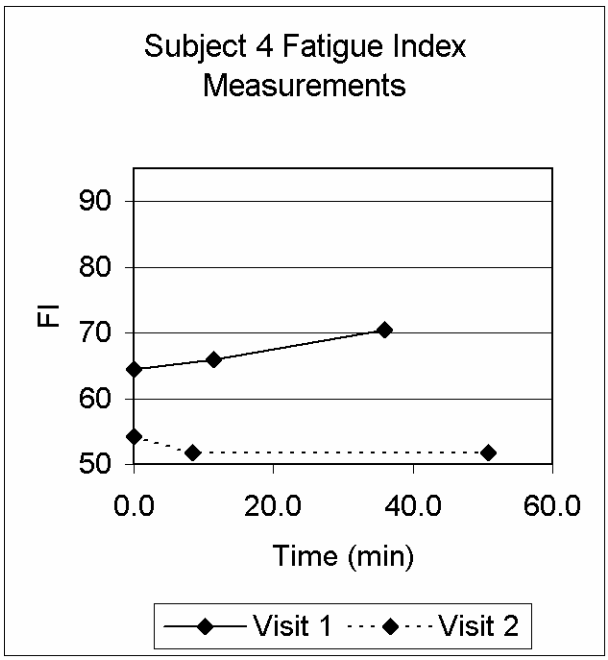


Figure 78: FI measurements for subject 4's first and second visits.

have more momentum by the time they notice that it has veered off path, thus making it difficult to drive back to the target path. Also, we observed while walking between hallways that the maximum speeds on the subjects' chairs were much lower than that of the virtual wheelchair. Subjects may not have been familiar with the speed of the virtual chair, causing difficulty.

Testing with the VDS software allows for some human error. On one occasion, the VDS mysteriously changed its algorithm for several trials. The subject gave permission to redo them. The program would also stop responding about once or twice during subjects' visits. This could very likely be related to the timing intervention described in Appendix L, and may become an issue if settings had been altered during the virtual driving phase of joystick tuning and not re-entered when restarting the VDS program.

8.0 CONCLUSIONS

The VCJ provides a functional platform for comparing isometric with compliant control interfaces with customizable algorithms. The current prototype of the VCJ also offers great flexibility in use, such as

- permitting a wide variety of control enhancers,
- spanning the full range of compliant to isometric settings, and
- measuring maximum voluntary isometric contractile forces of the arm.

The extra channels on the DAQCard permit easy expansion to drive real EPWs and collect signals from other sources such as a speed potentiometer or an encoder. The clinician has direct access to the joystick's programming to tune its parameters for the user. While its dynamic response matched a conventional MJ in only five of nine criteria for one handle and three of nine for another, the VCJ's static responses were very similar. Differences in the dynamic response are very likely a result of the different masses of the handles.

While it is difficult to make conclusions about the performance of the isometric and compliant interfaces with the various algorithms because of the sample size, preliminary inspection of the data shows that the learning curve may be long for this system. Also, while subjects may have low trial times, they could be related to higher RMSEs and number of boundary collisions.

A few changes to the system may improve testing with future subjects. Chiefly, the order of measuring FI 2 and determining the maximum gain could be switched. This would have the effect of eliminating one swap of the VCJ in driving mode with the VCJ in fatigue characterization mode, thus reducing the time to complete the personalization setup and reducing the chance of human error. Automatic updates to the settings file any time changes are made to the joystick parameters may also reduce the chance for human error in case the VDS software stops responding. Implementing alternative wheelchair dynamics to simulate drive types such as

mid- or front-wheel drive may reduce the familiarization time for subjects who do not typically use a rear-wheel drive EPW. Lastly, modifications to the boundary violation routine could reduce the risk of the virtual chair becoming "stuck in the weeds." If the chair were repositioned to the beginning of the track to restart the trial, for example, this would have the added effect of reminding the subject that accuracy is more important than speed.

While collecting multiple measurements of subjects' FIs could increase understanding of fatigue in MS, its use as a tool for determining fatigue adaptation and recovery rates should be monitored because of its risk to benefit ratio. Though it is only a little more than 30 s, one subject noted that it affects the feelings in his hands, and another subject mentioned arm soreness a couple days after testing. MS is already a highly variable condition and factors not related to muscle activity (e.g. the weather or time of day) may have an influence on a patient's motor performance [7].

If it seems that the personalized algorithm with fatigue adaptation could improve driving skill, a next step could be to do a controlled trial to see if people using the MS_PFA_FA are less fatigued using the EPW than the standard algorithm. The VCJ may also be used to examine how much force EPW drivers prefer to use while with their EPWs, "transitional" interfaces between movement and isometric joysticks [68], or improved computer access.

APPENDIX A

COMPLETE SET OF TECHNICAL REQUIREMENTS FOR THE VCJ

1. Joystick Post (JP) Requirements
   1.1. General Requirements
      1.1.1. The JP couples with the operator's body to provide a mechanical interface between the operator and the Input Control Device.
      1.1.2. The JP shall be ergonomically appealing.
   1.2. Functional Requirements
      1.2.1. Electronics Enclosure (box)
         1.2.1.1. The box shall provide a robust platform for the joystick electronics and the joystick post.
         1.2.1.2. The size of the housing shall be minimized.
         1.2.1.3. The enclosure shall be watertight.
         1.2.1.4. The enclosure shall accommodate the following configurations:
            1.2.1.4.1. Hand control,
            1.2.1.4.2. *To be considered for future generations.* Foot control.
         1.2.1.5. The box shall accommodate for a mini-DIN interface.
         1.2.1.6. The box shall have an access panel for maintenance of the circuit boards.
      1.2.2. Joystick post. The joystick post shall support three modes of operation (compliant mode, isometric mode, and tremor isolation mode) and provide a platform for the strain gauges.
         1.2.2.1. Compliant Mode

1.2.2.1.1. The post shall be variably compliant, where compliance ranges from 0° to 18° from null position towards the forward direction as a result of maximal force input. Maximal force input can range from 4.3 N [67] to 300 N [4].

1.2.2.1.2. The post shall allow for repeatability of its configuration.

1.2.2.1.3. The post shall have the capability to move with two degrees of freedom to distinguish between speed and direction.

1.2.2.1.4. A variety of mechanical templates shall be available, including the standard diamond shape as well as square, cross, circle, and oval. The template size shall be scaleable.

1.2.2.2. Emulation of a commercially available joystick.

1.2.2.2.1. The post shall have a length of 3.0" ± 0.5" (distance between pivot point and tip)

1.2.2.2.2. The post shall have a mechanical template with the diamond pattern (forward tip angle is 18°).

1.2.2.2.3. A force of 4.3 N applied in the forward direction shall result in a full deflection of 18°.

1.2.2.2.4. The post shall have the following dynamic characteristics when released from full deflection in the forward and reverse directions.

Table 16: Dynamic Characteristics of a conventional position sensing joystick [78].

	Forward	**Reverse**
Rise Time (sec)	0.024 (0.002)	0.026 (0.003)
Peak Overshoot (%)	45.284 (1.948)	41.781 (2.324)
Settling Time (sec)	0.123 (0.038)	0.128 (0.038)

1.2.2.3. Isometric Mode

1.2.2.3.1. Forces applied to the joystick shall not result in perceivable movement of the post.

1.2.2.3.2. While in driving mode, the joystick post shall not yield under 35 N.

1.2.2.3.3. While in fatigue characterization mode, the joystick post shall not yield unter 350 N.

1.2.2.4.*To be considered for future generations.* Tremor Isolation Mode

1.2.2.4.1. The post shall include a mass-spring-damper system to reduce the effect of tremor. Tremor frequencies are assumed to be a minimum of 4 Hz [72]. Hand/forearm masses are assumed to be between 0.47 kg and 2.16 kg [72].

1.2.2.4.2. The effective spring constant shall be between 50 lbs/in to 175 lbs/in [72].

1.2.2.4.3. The effective damping coefficient shall be between 12.4 lb-sec/in and 40.7 lb-sec/in [72].

1.2.2.4.4. The overshoot when returning to the null position after an applied force is withdrawn shall be minimized.

1.2.2.4.5. The time required for the post to return to the null position shall be minimized.

1.2.2.5.Strain Gauges

1.2.2.5.1. The post shall provide a platform for the strain gauges. The faces of the platform shall be orthogonal to each other and shall each support a Micro-Measurements 125BZ strain gauge.

1.2.2.5.2. The ratio of the beam length to the cube of its thickness shall be less than 1.723x103 in-2 [74].

1.2.2.5.3. The dimensions of each face shall be at least 0.29 inches long by 0.2765 inches wide, as these are the dimensions of two 125BZ strain gauges placed next to each other.

1.2.2.5.4. Maximum force on the post is assumed to be 67.676 lbf [74].

1.2.3. Mounting hardware. The joystick will be mounted to the armrest of a wheelchair. This depends on the solution for the joystick post.

1.2.3.1.The size of the mounting hardware shall be minimized.

1.2.3.2.The mounting hardware shall include a clamping mechanism to attach to the end of mounting tube with a diameter of 1”.

1.2.3.3. The mounting mechanism shall allow the joystick to move with two degrees of freedom to customize the position for the subject and then locked down while in use.

1.2.3.3.1. The allowable range of motion in the lateral direction shall be 3 in.

1.2.3.3.2. The allowable range of motion in the longitudinal direction shall be 6 in.

1.3. Interface Requirements

1.3.1. Input

1.3.1.1. The user shall be able to apply forces to the joystick indicating desired speed and direction of the wheelchair.

1.3.1.2. The user shall be able to adjust the level of compliance.

1.3.1.3. The user shall be able to adjust the position of the VCJ to fit the subject.

1.3.2. Output

1.3.2.1. The JP shall output analog signals proportional to the amount of force applied to the JP that the ICD can process.

1.4. Safety Requirements

1.4.1. The outside corners and edges shall be rounded.

1.4.2. The mass-spring-damper system shall be completely enclosed within the electronics enclosure.

1.5. Miscellaneous

1.5.1. A design in SolidWorks shall be provided for review.

1.5.2. The SolidWorks design shall be parts-based and assembled.

1.5.3. A parts list for the JP shall be provided.

2. Input Control Device (ICD) Requirements

2.1. General Requirements

2.1.1. The ICD processes the user input which gets sent to the wheelchair controller as analog inputs in place of the native joystick.

2.1.2. The ICD shall be capable of detecting template violations when in movement sensing emulation mode.

2.1.3. The VCJ shall be capable of communicating with the virtual driving software, the subject's wheelchair, and a lab wheelchair. Subject's wheelchair controllers include Dynamic, Invacare, and Penny & Giles.

2.1.4. *Not essential.* The algorithms must utilize less than 48 Kb of memory. What will really happen is that there will be 8 Kb of ROM available and 32 Kb of RAM.

2.1.5. *Not essential.* The VCJ shall accommodate PS/2 mouse emulation for interfacing with the virtual reality system.

2.1.6. *Not essential.* EEPROM "flash" memory shall be used instead of the current EPROM chips.

2.2. Functional Requirements

2.2.1. Variable Gain Algorithm (VGA)

2.2.1.1. The ICD shall be capable of implementing the VGA algorithm as described in [67].

2.2.2. Multiple Sclerosis-Personally Fitted Algorithm (MS_PFA)

2.2.2.1. A dead zone shall be applied, where forces below which results in zero output.

2.2.2.2. The ICD shall offset the speed and direction axes to fit the subject's definitions of "straight" and "left." "Backwards" and "right" shall be 180º off straight and left, respectively.

2.2.2.3. The limits for forward and reverse speed shall have a range of 10% to 150% of the initial value.

2.2.2.4. The limits for left and right turning rates shall have a range of 10% to 150% of the initial value.

2.2.2.5. A template shall be applied, where a force beyond which is mapped to the nearest location on the template in the same direction as the intended force.

2.2.2.5.1. Possible template shapes shall include square, circle, ellipse, and diamond.

2.2.2.5.2. Template shapes are defined with super quadratics.

2.2.2.5.3. The orientation of the template can be rotated.

2.2.2.5.4. The template is downloaded to the custom algorithm's memory after joystick calibration.

2.2.3. Multiple Sclerosis-Personally Fitted Algorithm with Fatigue Adaptation (MS_PFA_FA)

2.2.3.1. Functional requirements for the MS_PFA_FA shall include those listed for the MS_PFA in addition to those listed below.

2.2.3.2. The ICD shall log the accumulated driving activity and gradually adjust its gain parameters to compensate for reduced hand strength.

2.2.3.2.1. While force is being applied to the joystick, the gain shall be adjusted according to equation 1

$$K_1 = K_{\max} - (K_{\max} - K_2)e^{-\alpha A}, \qquad (1)$$

where K_1 is the new gain, $K_{\max}$ is the maximum gain set for the individual, $K_{\min}$ is the baseline gain, α is a parameter based on the subject's motor performance graph, and A is the accumulated force-time integral.

2.2.3.2.2. While force on the joystick is under the dead zone, the gain shall be adjusted according to equation 2.

$$K_2 = K_{\min} + (K_1 - K_{\min})e^{-\beta T}, \qquad (2)$$

where K_2 is the new gain, β is a parameter based on the subject's motor performance graph, and T is the duration the joystick is inactive.

2.2.3.2.3. After 8 hours of inactivity, the gain shall be reset to the nominal (baseline) value.

2.2.4. Filter Algorithm. A filter algorithm shall be coupled with the MS-PFA algorithms to reduce the effects of tremor via any of the following options.

2.2.4.1. The ICD shall reduce the effects of tremor via a second order digital Butterworth Filter. (As described in the grant, though the grant was more of a layout and need not be followed if there is a better filter)

2.2.4.2. The ICD shall be capable of employing an adaptive filter, pole placement scheme, or moving average to enhance the signal quality to the wheelchair.

2.2.4.2.1. Assuming the pole-placement scheme, poles shall be placed at -13.5 ± 13.6i (assuming cutoff frequency of 4 Hz).

2.3. Interface Requirements

2.3.1. Inputs

2.3.1.1. Operational Inputs

2.3.1.1.1. The ICD shall accept 12-bit digital inputs for speed and direction.

2.3.1.1.2. The ICD shall accept template boundary violations.

2.3.1.1.3. The ICD shall accept clock inputs.

2.3.1.2. Programming Inputs

2.3.1.2.1. Axis orientations for speed and direction (radians).

2.3.1.2.2. Nominal gains in speed and direction.

2.3.1.2.3. Maximum allowable gains in speed and direction (for gain scheduling).

2.3.1.2.4. Dead zone size and shape. Potential shapes include circular, square, oval, and rectangle.

2.3.1.2.5. Template size and shape. Potential shapes include circular, square, oval, rectangle, starburst, and diamond.

2.3.1.2.6. Motor performance parameters α and β for gain scheduling.

2.3.1.2.7. Cut-off frequencies for filter.

2.3.2. Outputs. Output signals go to virtual driving SW, the subject's WC, and a lab WC.

2.3.2.1. Direction

2.3.2.2. Speed

2.3.2.3. Error signal: Outside dead zone at start-up.

2.4. Safety Requirements

2.4.1. If the stick forces are outside the dead zone upon start-up of the controller, the wheelchair will not move and an error signal shall be provided.

APPENDIX B

TECHNICAL DRAWINGS FOR THE VCJ

B.1 OVERVIEW

The diagram in **Figure 79** depicts an exploded view of how the VCJ's components are assembled. A handle ① screws on top of the stick ③, which is press fit into the swivel bearing ⑧. The swivel bearing is pinned into the bearing mount ⑨ with the bearing stopper ⑦. The boot ⑤ slides along the sliding surface ⑥ whenever the joystick is deflected. The template ④ (four of which are available) prevents the stick from deflecting more than 18°. The bearing mount uses a sliding press fit with set screws to mate to the load cell ⑩, which is cinched to the base with an acorn nut (**Figure 10**). Not depicted in **Figure 79** are the yellow box in which the VCJ is housed nor the isometric insert for fatigue characterization.

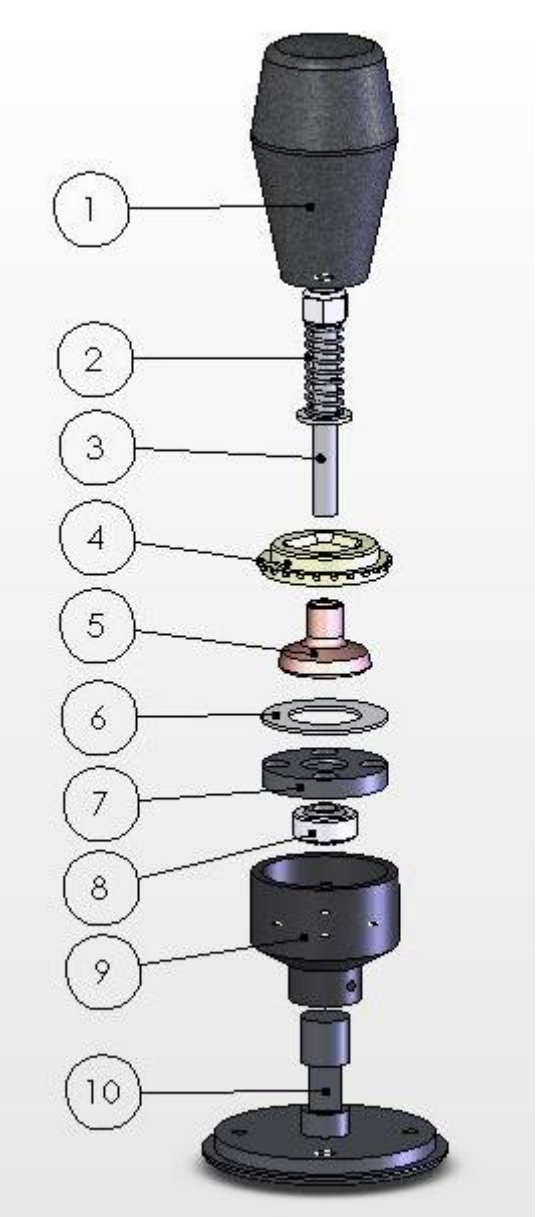


Figure 79: Exploded view of VCJ with pivoting mechanism.

B.2 VCJ COMPONENT DRAWINGS

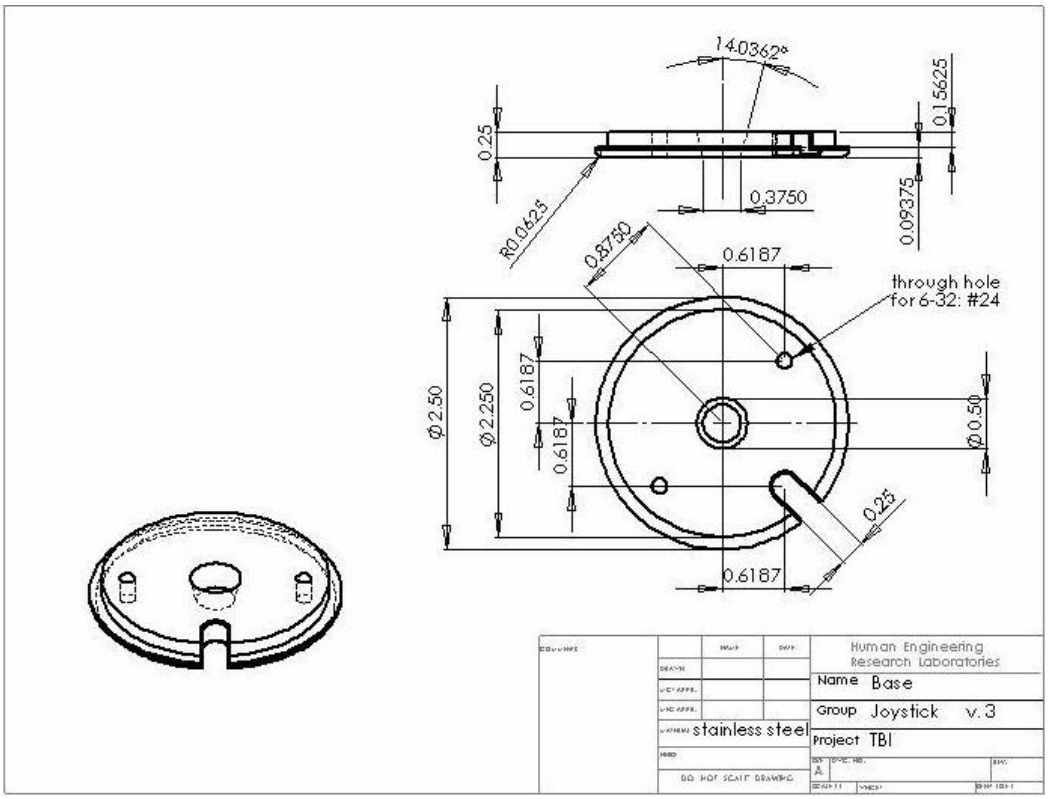


Figure 80: Base.

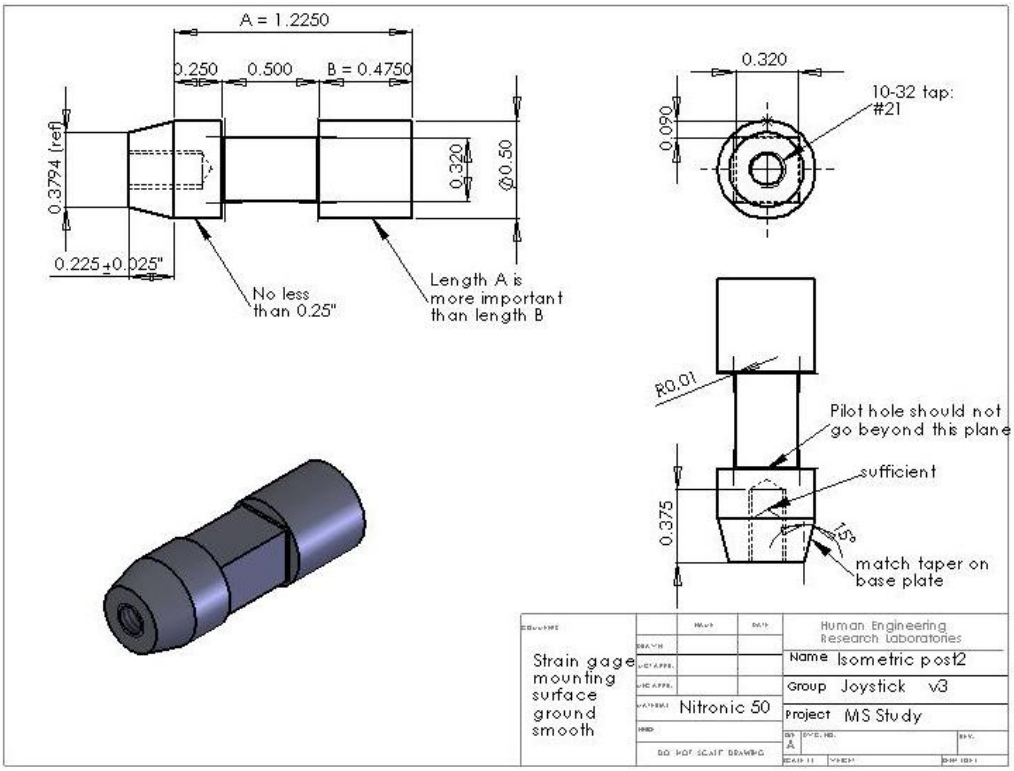


Figure 81: Load cell.

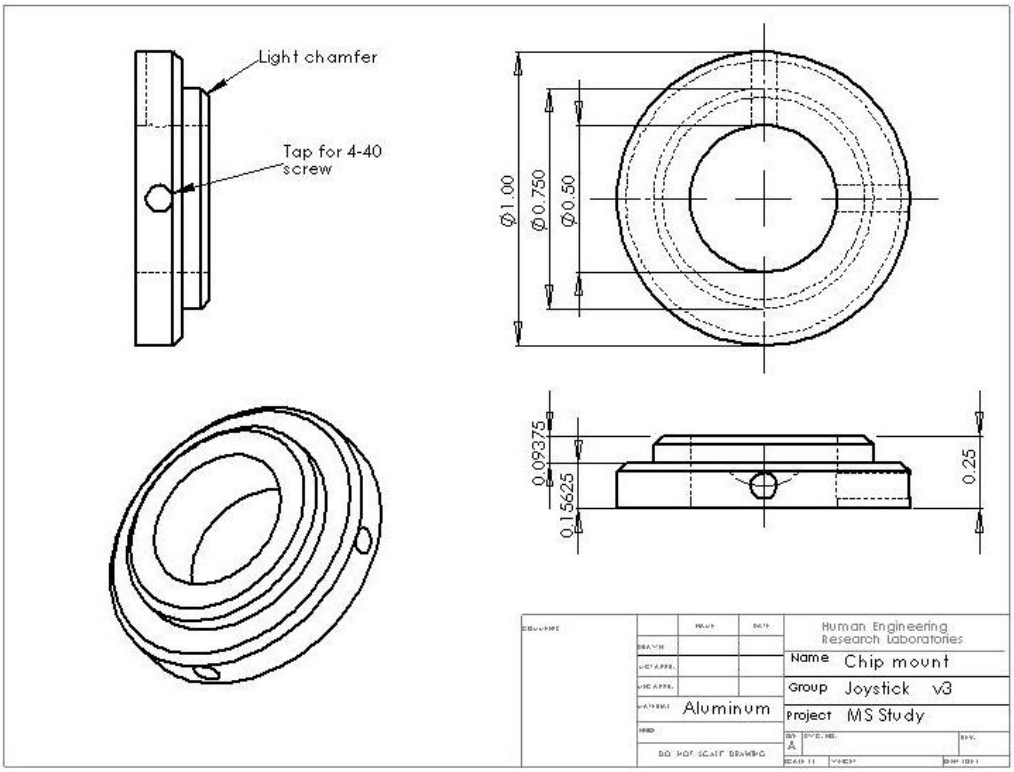


Figure 82: Chip mount.

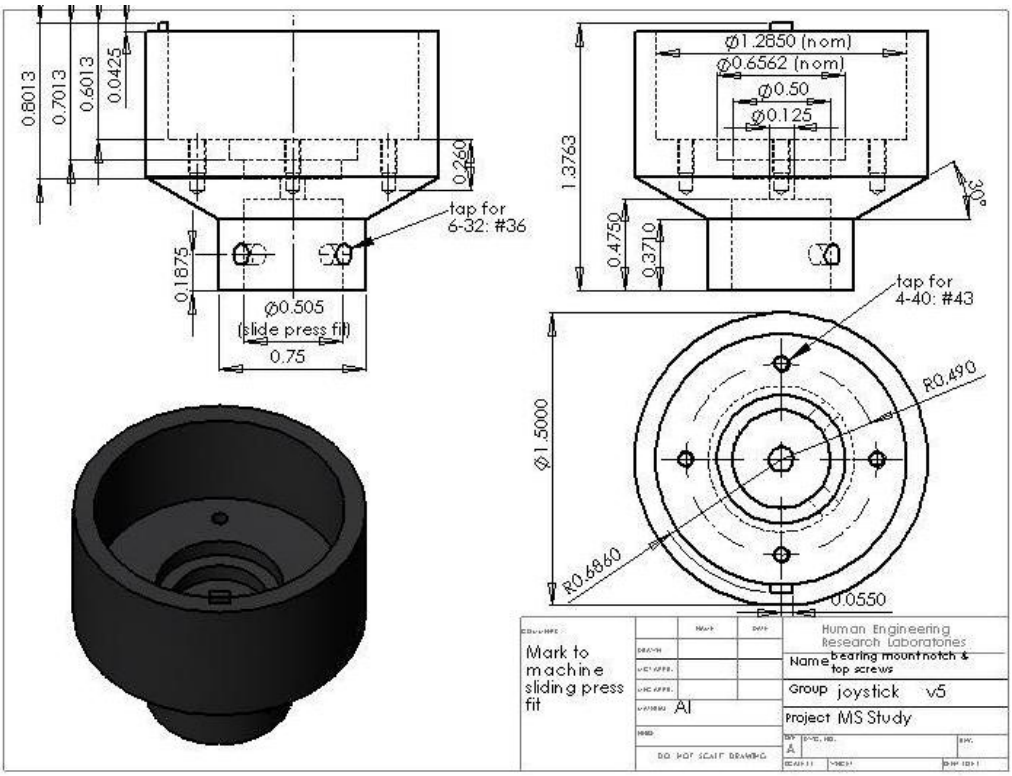


Figure 83: Bearing mount.

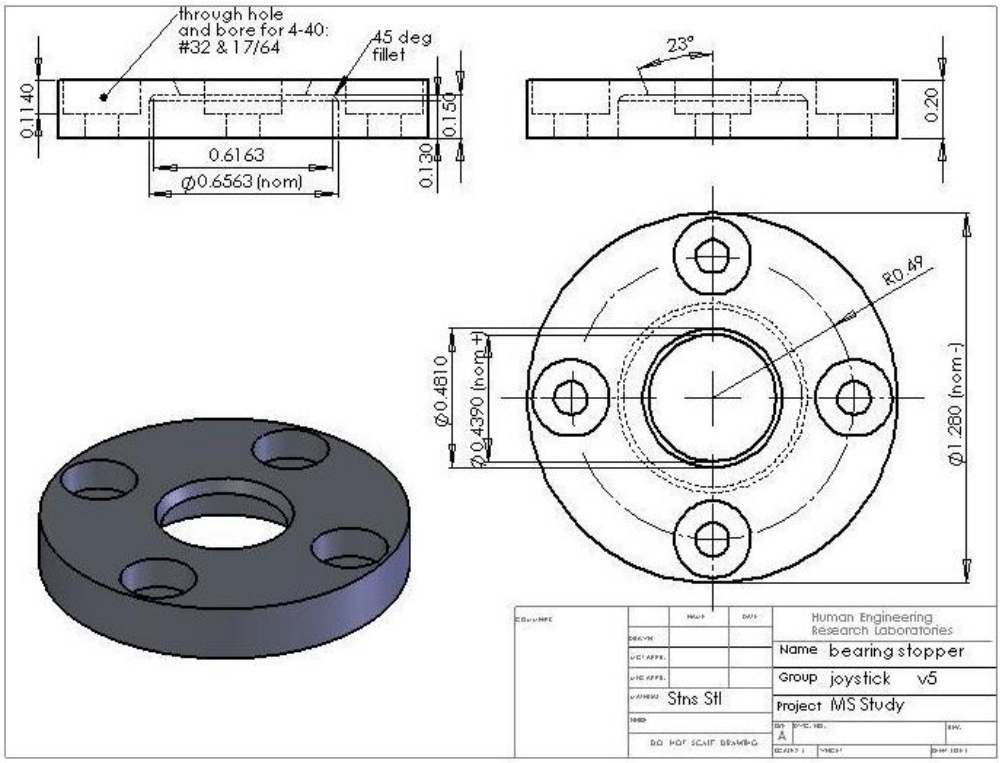


Figure 84: Bearing stopper.

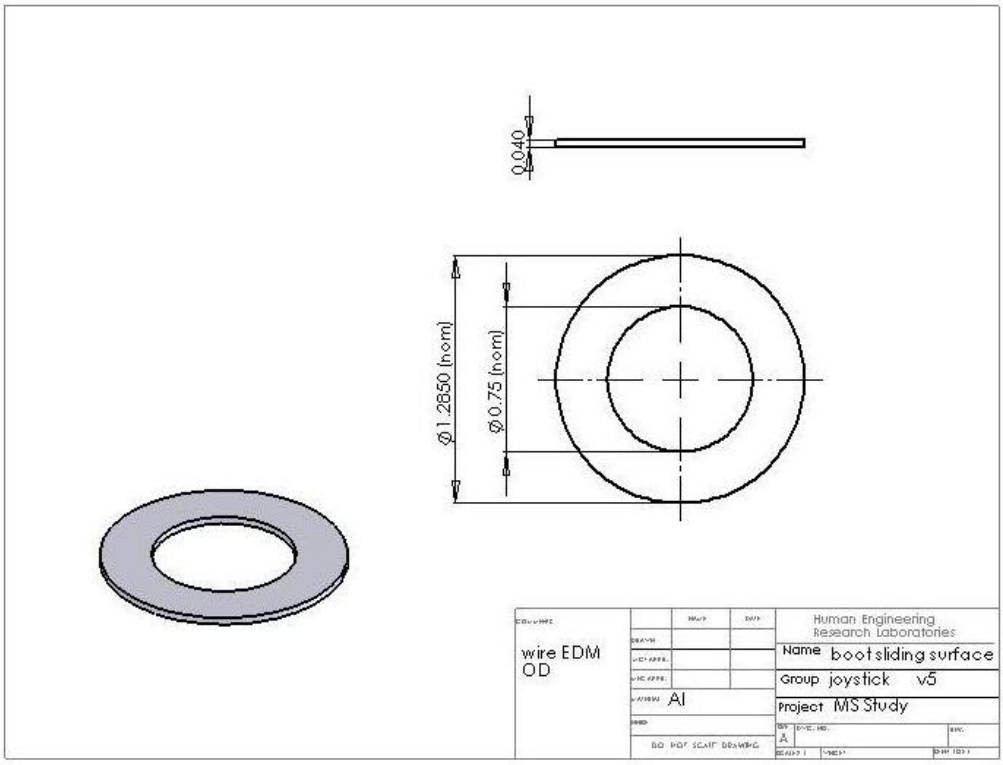


Figure 85: Boot sliding surface.

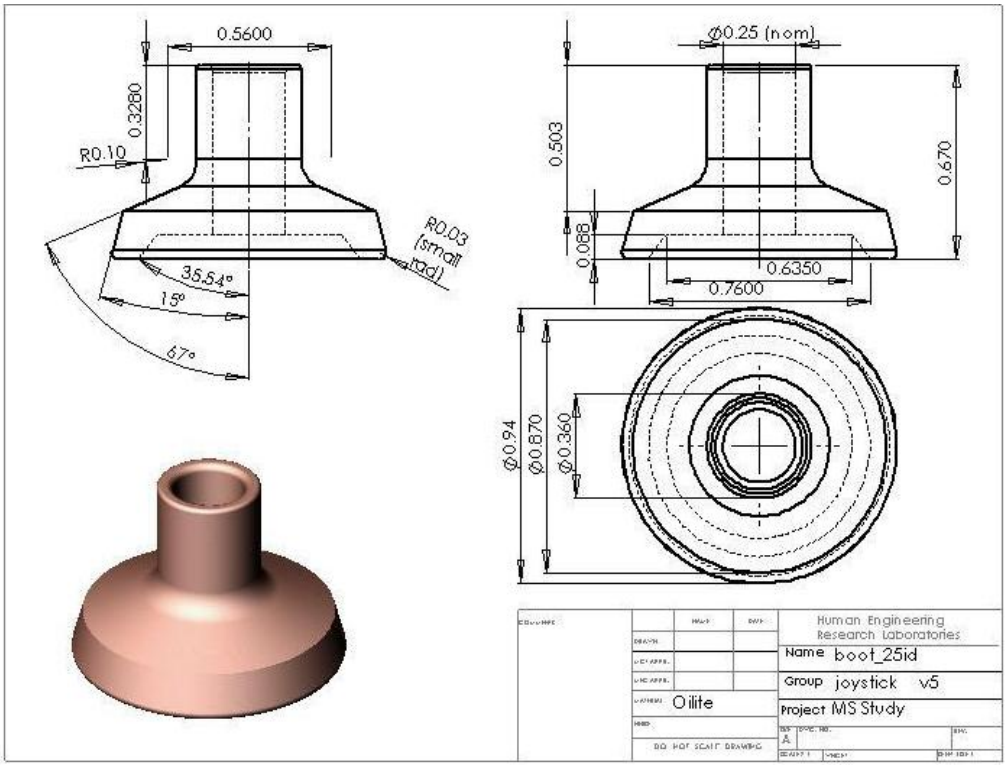


Figure 86: Boot.

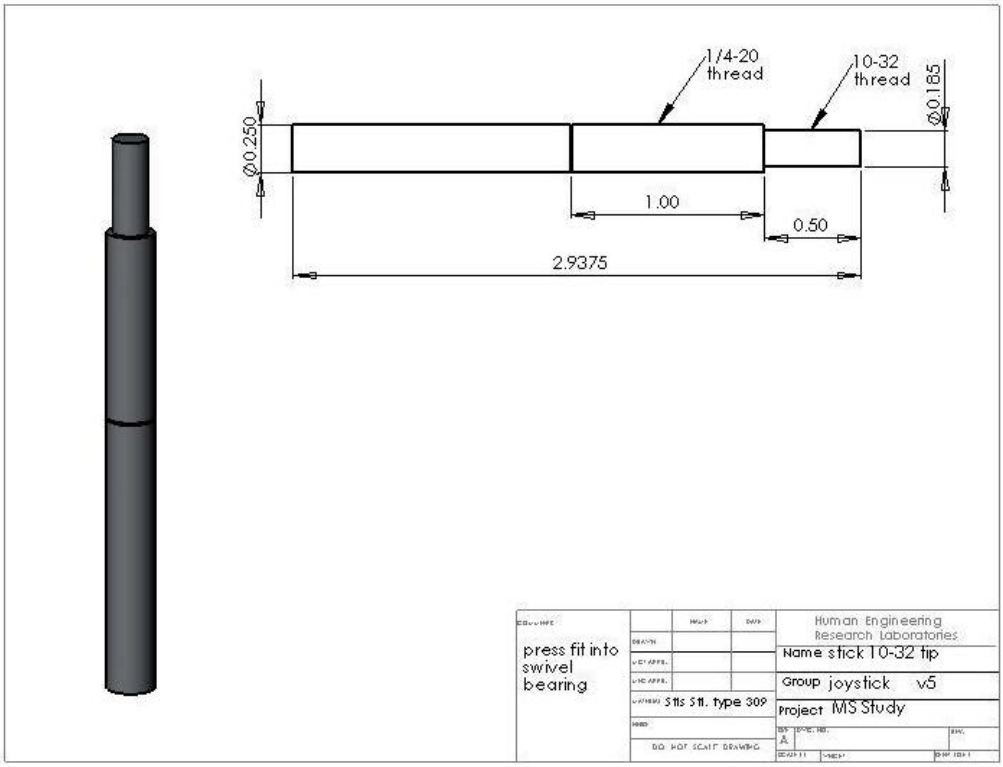


Figure 87: Joystick stick/shaft.

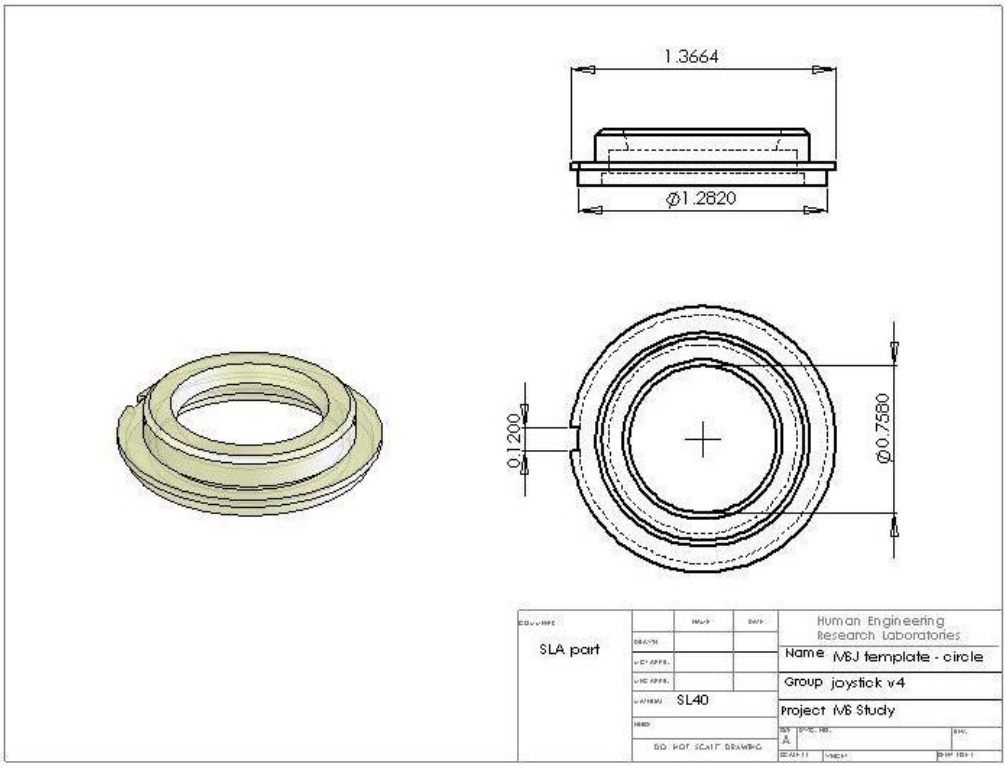


Figure 88: MJ template – circle.

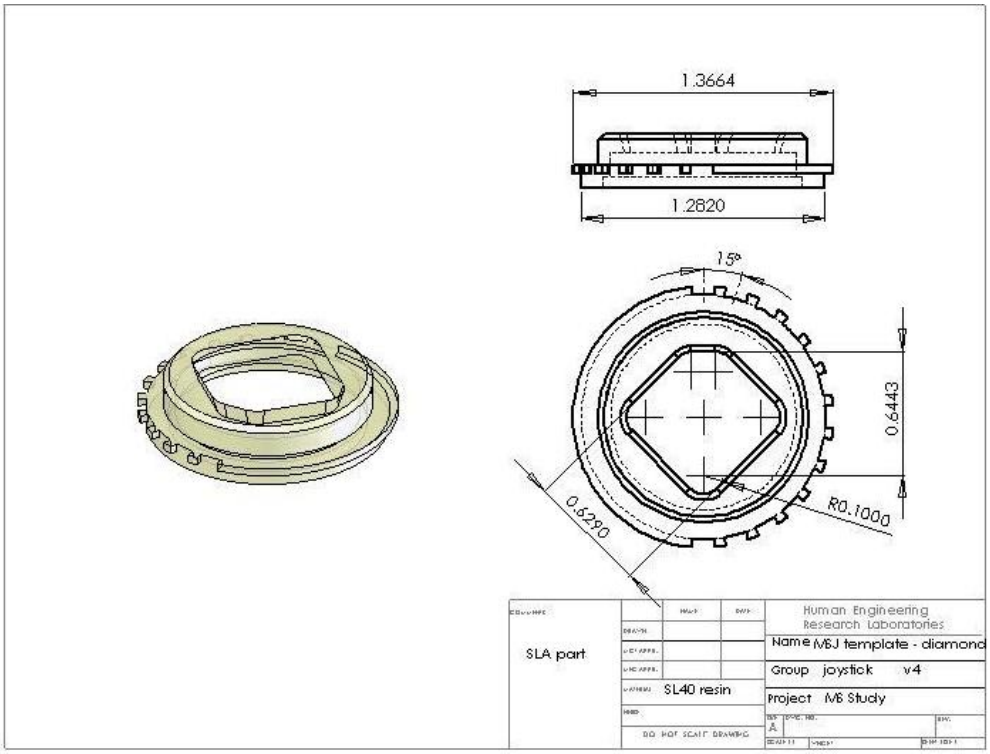


Figure 89: MJ template – diamond.

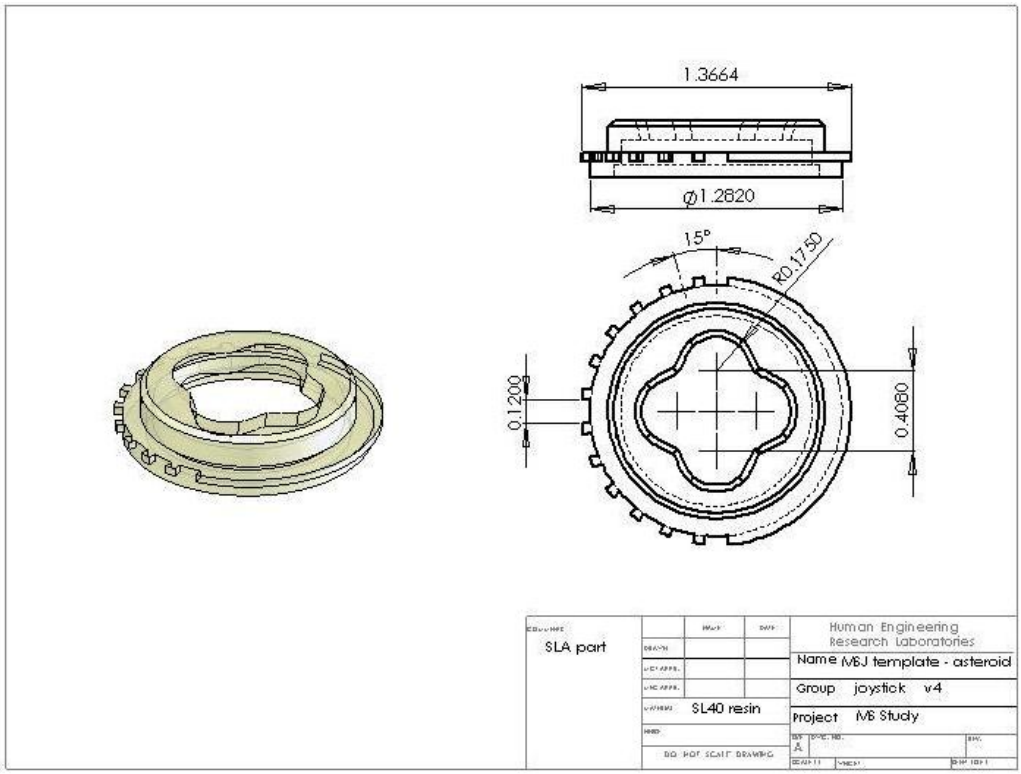


Figure 90: MJ template – asteroid.

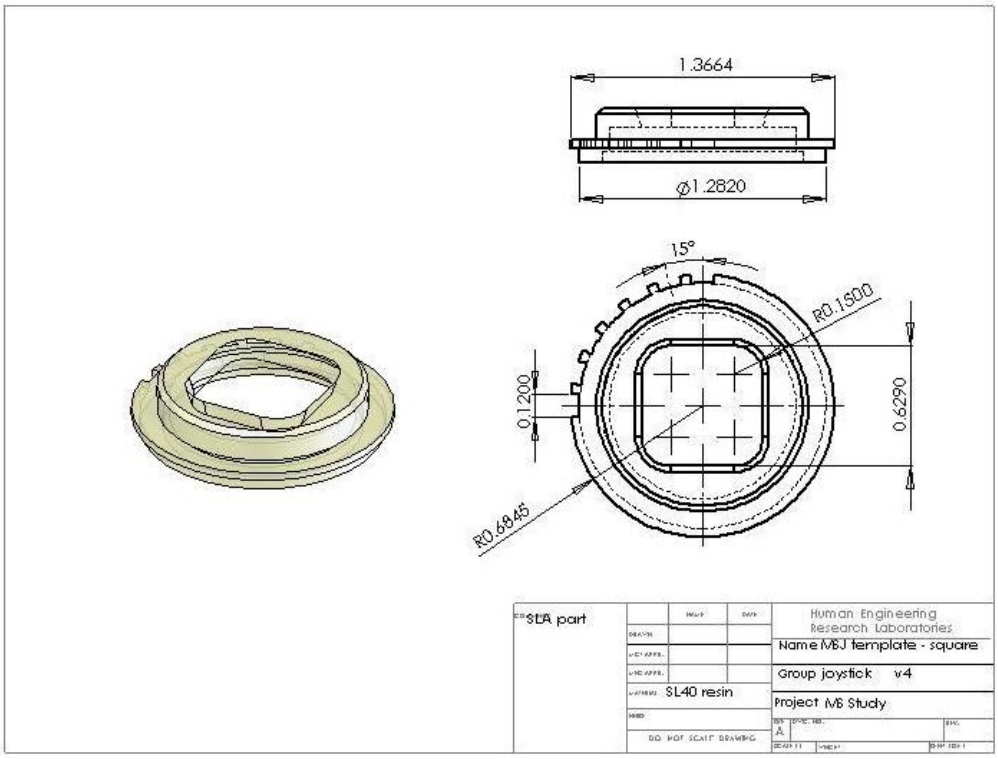


Figure 91: MJ template – square.

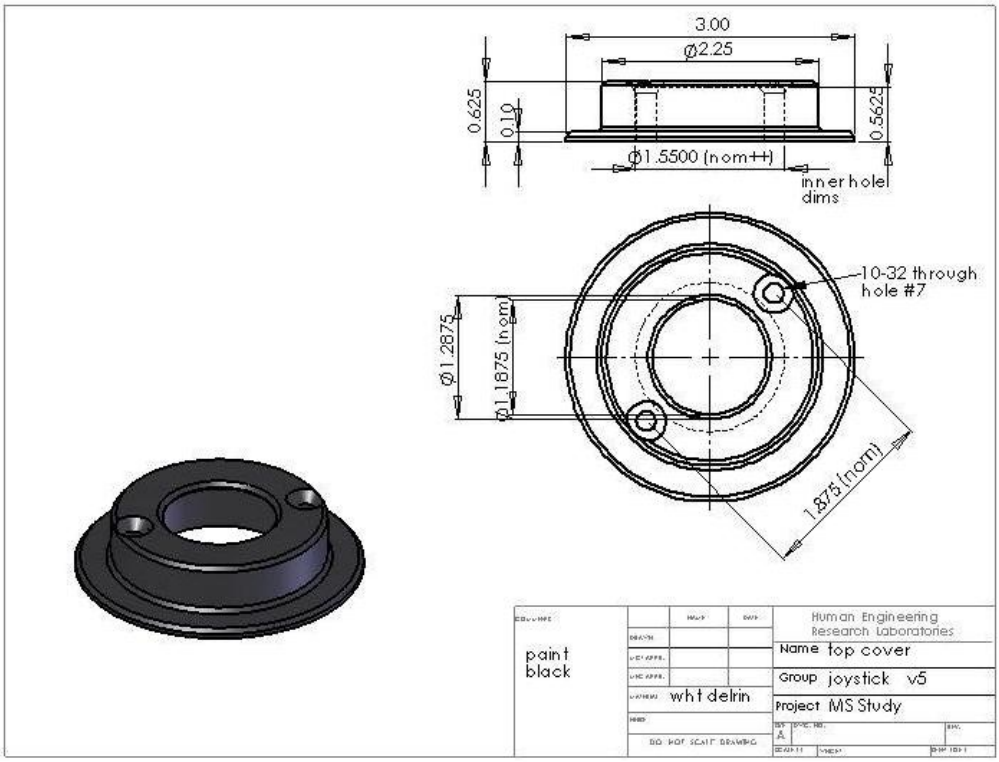


Figure 92: Top cover.

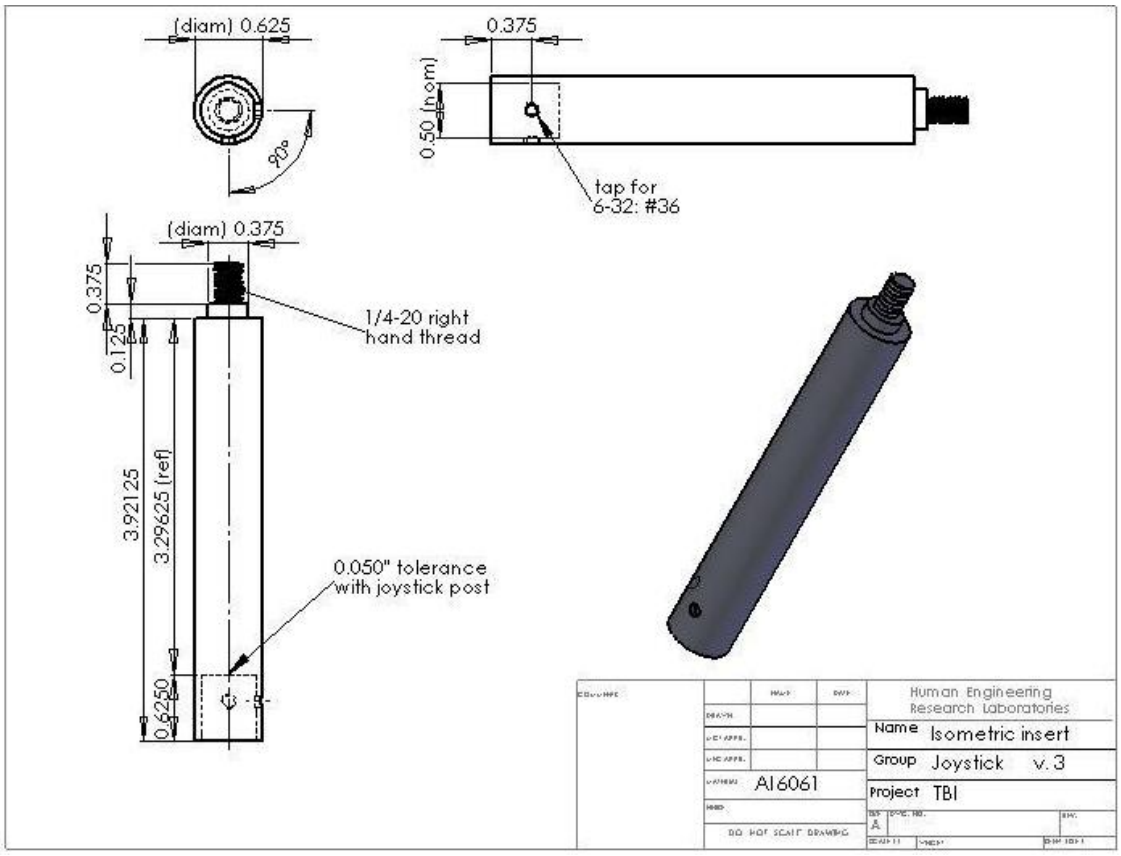


Figure 93: Isometric insert.

APPENDIX C

STATIC LOADS ANALYSIS OF PIN MECHANISM

C.1 ANALYSIS

Verifying whether the pins would be able to withstand repeated use involved a two step process of determining the resulting input force and ensuring that the maximum stresses were below the yield strength for the pins on both axes. The same free body diagram as depicted in **Figure 22** can be used with the analysis with the exception that the spring F_s is replaced with a solid piece of aluminum. Thus, the stick and boot can be viewed as a solid body. Applying the laws of statics, we have the following relationships.

C1: $\Sigma M_P = 0 \Rightarrow -F_i \times l_s + R_b \times r_b = 0 \Rightarrow R_b = F_i \times l_s / r_b$

C2: $\Sigma F_y = 0 \Rightarrow R_b + R_{Py} = 0 \Rightarrow R_{Py} = -R_b$

C3: $\Sigma F_x = 0 \Rightarrow F_i + R_{Px} = 0 \Rightarrow R_{Px} = -F_i$

The resulting input force R_P is square root of the sum of the squares of its reaction forces. After substitution,

C4: $$R_P = \sqrt{(-F_i)^2 + \left(\frac{-F_i l_s}{r_b}\right)^2} \ .$$

The free body diagrams for the pins are depicted in **Figure 94**. Maximum stress occurs when the input force is normal to the pins' axes. Thus, loading conditions needed to be considered individually for each axis. Since the pins are press fit into the gimbal and gimbal mount, fixed end conditions may be assumed. The moment at an end $M_{P,i}$ is given by equation C5 [76].

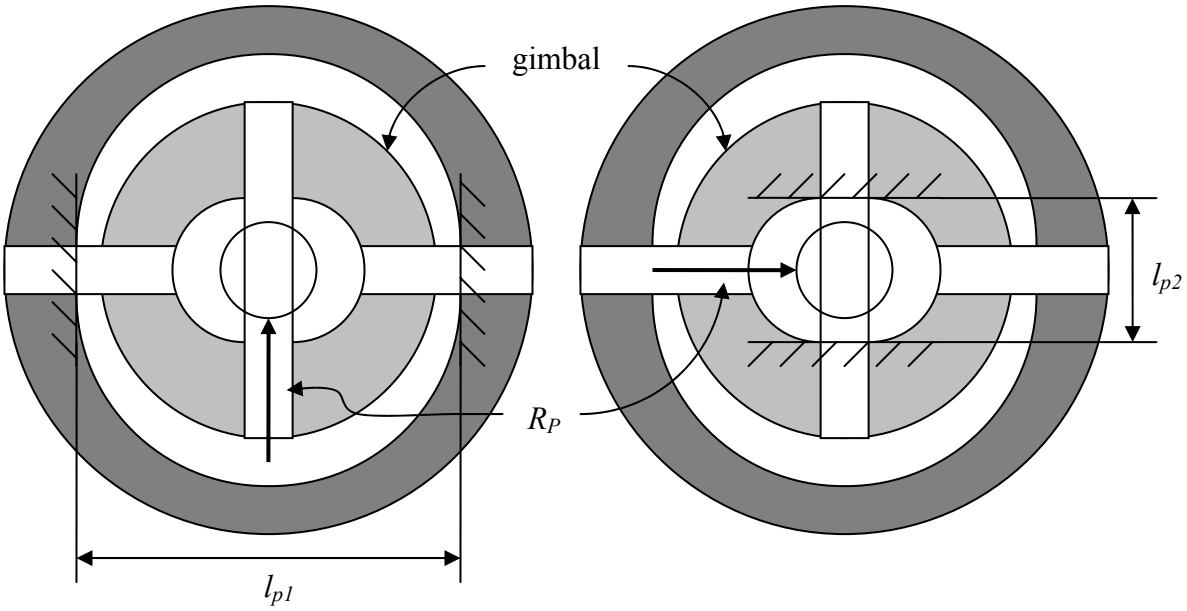


Figure 94: Free body diagram for reaction force at pin and gimbal.

$$\text{C5:} \qquad M_{P,i} = \frac{R_P \frac{l_{P,i}}{2}\left(\frac{l_{P,i}}{2}\right)^2}{l_{P,i}} \Rightarrow \frac{R_P l_{P,i}}{8},$$

where l_{Pi} is the length of pin exposed for bending, and i is the designator for the pin (1 or 2). The maximum stress σ_{max} is thus given by equation C6.

$$\text{C6:} \qquad \sigma_{\max} = \frac{M_{P,i} r_{P,i}}{I_{P,i}},$$

where $r_{P,i}$ is the radius of the given pin, and $I_{P,i}$ is its area moment of inertia.

C.2 RESULTS

A MATLAB script was written to determine the resulting maximum stresses for each pin and is included in section C.4. To account for repeated use, a safety factor of 3 was applied to the design input force of 35 N. Physical parameters were based on a prototype of the VCJ that used

the gimbal with pins. The maximum stresses are 75.8 ksi and 54.3 ksi for pins 1 and 2, respectively.

C.3 DISCUSSION

The yield strength of aluminum 6061 and stainless steel type 416 are approximately 35.0 ksi and 39.9 ksi, respectively. Since the maximum stress on pin 1 is greater than 39.9 ksi, there is potential for it to fail. While increasing the radii of the pins would decrease maximum stresses, the resulting through hole in the stick would be too large.

The strength characteristics of the bearing stopper were analyzed with COSMOSXpress in SolidWorks. Using the same pivot reaction force as an input force on the bearing stopper, the resulting safety factor is 2.31. The analysis output is provided in **Figure 95**.

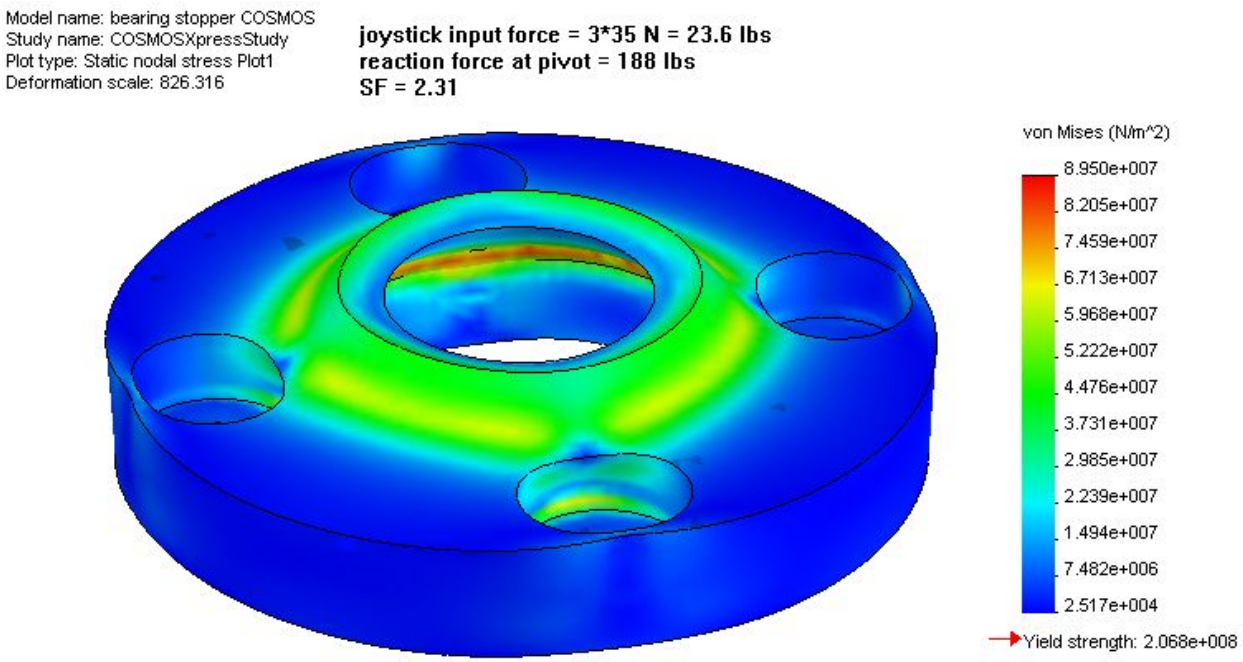


Figure 95: COSMOSXpress deformation model of bearing stopper under modeled loading conditions.

C.4 MATLAB CODE

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Version: 1.0
% Last Modified: 6/19/06
% By: kwb
%
% vcj_static_model.m
%
% >> PURPOSE <<
%
%   the purpose of this program is to calculate maximum stresses in the
%   VCJ.
%
% >> NOMENCLATURE <<
%
% >> HISTORY <<
%
%   Version 1.0         Karl Brown          22 October 2005
%   Original version. See notes from lab book 6/6/06.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%physical characteristics
l = 3.75;   %length of stick, in
l_b = 1.75; %length/height of boot, in
w_b = 0.95/2;   %radius of boot, in
l_p1 = 0.1876;  %length of pin that goes through the stick, in
l_p2 = 0.62;    %length of pin that goes through the gimbol, in
r_stick = 0.19/2;   %radius of the stick, in
r_p1 = 0.0938/2;    %radius of pin that goes through the stick (pin 1), in
r_p2 = 0.125/2; %radius of pin that goes through the gimbol (pin 2), in

% for reference only
% E_AL = 10000000;    %modulus of elasticity for aluminum, psi
% E_SS = 28000000;    %modulus of elasticity for stainless steel, psi

%yield strength of materials
sig_max_AL = 35e3;  %yield strength of aluminum, psi
sig_max_SS = 39.9e3;%yield strength of stainless steel type 416, psi

%force input
F_i = 7.86*3;   %lbs = default + safety factor of 3

%derived characteristics
I_stick = (pi*r_stick^4)/4; %area moment of inertia of stick, in^4
I_p1 = (pi*r_p1^4)/4;   %area moment of inertia of pin 1, in^4
I_p2 = (pi*r_p2^4)/4;   %area moment of inertia of pin 2, in^4

%%%%%%%
% determine max stress in stick
sig_stick = F_i*(l-l_b)*r_stick/I_stick;
```

```
%%%%%%%
% determine global reaction forces
R_2y = F_i*l/w_b;
R_1y = -R_2y;
R_1x = -F_i;
Rp = sqrt(R_1y^2+R_1x^2);

%%%%%%%
% determine reaction forces and moments for pins
%   only one side needs to be calculated because of symmetry
%pin 1
R_p1 = Rp/2;
M_p1 = Rp*l_p1/8;

%pin 2
R_p2 = Rp/2;
M_p2 = Rp*l_p2/8;

%%%%%%%
% determine max stress in pins
sig_pin1 = (M_p1*r_p1)/I_p1;
sig_pin2 = (M_p2*r_p2)/I_p2;

%%%%%%%
% display results
fprintf('\nInput Force:\t%4.2f\n', F_i);
fprintf('\nReaction Forces (lbs)\n');
fprintf('R_1x:\t%3.2f\n', R_1x);
fprintf('R_1y:\t%3.2f\n', R_1y);
fprintf('Rp:\t%3.2f\n', Rp);
fprintf('R_p1:\t%3.2f\n', R_p1);
fprintf('R_p2:\t%3.2f\n', R_p2);
fprintf('\nReaction Moments (in-lbs)\n');
fprintf('M_p1:\t%3.2f\n', M_p1);
fprintf('M_p2:\t%3.2f\n', M_p2);
fprintf('\nMax Stresses (Al=%3.2e\tSS=%3.3e) (psi)\n', sig_max_AL, ...
    sig_max_SS);
fprintf('stick:\t%3.2e\n', sig_stick);
fprintf('pin 1:\t%3.2e\n', sig_pin1);
fprintf('pin 2:\t%3.2e\n', sig_pin2);
```

APPENDIX D

FINDING THE ISOMETRIC INSERT'S WALL THICKNESS

D.1 ANALYSIS

The free body diagram in **Figure 96** was used to determine how thick the wall needed to be on the aluminum insert at the load cell interface. Maximum stress is assumed to be located at the bottom of the interface hole on the isometric insert since a gap may exist here between the load cell and the isometric insert if the mate is not perfect and the isometric insert cocks a small angle.

The maximum stress $\sigma_{\max}$ as defined in equation D1,

$$\text{D1:} \qquad \sigma_{\max} = \frac{Mr_2}{I},$$

where r_2 is the outer radius and I is the area moment of inertia given by equation D2,

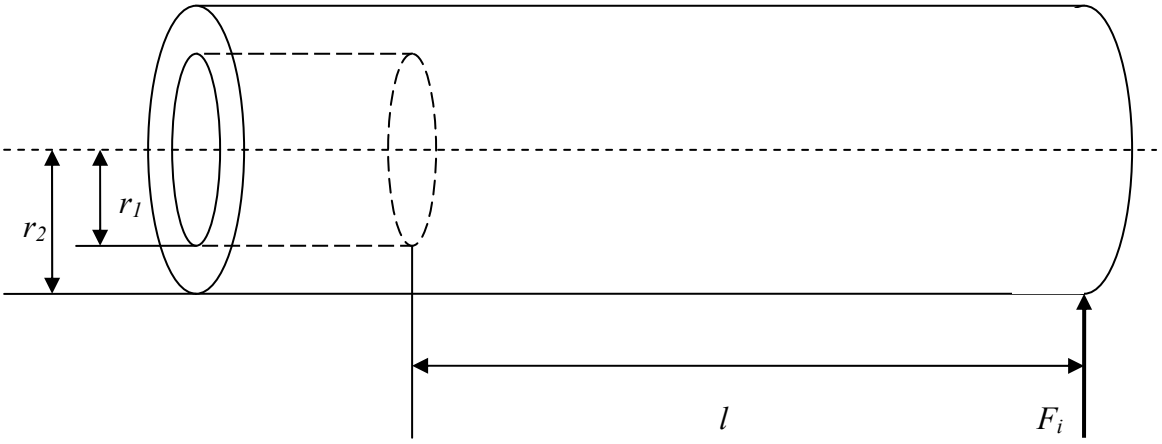


Figure 96: Free body diagram of isometric insert.

D2: $$I = \frac{\pi r_2^4}{4} - \frac{\pi r_1^4}{4} = \frac{\pi}{4}\left(r_2^4 - r_1^4\right),$$

where r_1 is the inner radius. Therefore, the maximum stress is

D3: $$\sigma_{\max} = \frac{4F_i l r_2}{\pi\left(r_2^4 - r_1^4\right)}.$$

The yield strength for aluminum 6061 is approximately 35 ksi. With an inner diameter of 0.5 in., input force of 350 N (78.7 lbs), and lever arm of 3.29 in. the outer radius r_2 is 0.285 in. The minimum wall thickness and outer diameter are thus 0.085 in. and 0.57 in., respectively.

APPENDIX E

SOURCE CODE IMPLEMENTATION OF DATA ACQUISITION SOFTWARE

E.1 JOYDAQ TRADITIONAL NIDAQ.H

```
/*************************************************************************
    modified: 9/12/06
    by: kwb

    JoyDAQ traditional NIDAQ.h

    PURPOSE
    This sets up the functions for JoyDAQ traditional NIDAQ.cpp

    HISTORY
    kwb [7/13/06]: original version.
    kwb [9/12/06]: add DAQ_find_offset2() as another method for finding
*************************************************************************/

/* function prototypes */
int DAQ_initialize(void);
int DAQ_find_offset(void);
int DAQ_find_offset2(void);
int DAQ_get_data(float *, float *); /* pass by address direction and speed */
int DAQ_close_device(void);

/*** globals ***/
/* address values for input and output channels */
#define ACH0        0    /* direction */
#define ACH1        1    /* speed */
#define DAC0OUT     0    /* direction */
#define DAC1OUT     1    /* speed */
#define DAQ_DEVICE  1
```

E.2 JOYDAQ TRADITIONAL NIDAQ.CPP

```
/*************************************************************************
     last modified: 11/05/06
     by: kwb

     JoyDAQ traditional NIDAQ.cpp

     PURPOSE
     This code interfaces to the NI-DAQcard 6024e. Its purposes are to
          * initialize the NI-DAQcard,
          * read analog inputs, and
          * send outputs [digital signal to software or analog outputs in
            place of a movement sensing joystick].
     The flow of the program is modeled after the suggested flow charts in the
     DAQ: Traditional NI-DAQ (Legacy) User Manual: Version 7.x, pgs 3-25++.

     NOMENCLATURE
     variable definitions provided in line with code

     HISTORY
     kwb [7/13/06]: original version. based on version 1.3 of ICD_functions.c
          sent algorithm functions and initialization to JoyAlgorithms.cpp,
          rename functions to have DAQ as a prefix rather than ICD
     kwb [8/17/06]: major overhaul=> switch to NI-DAQmx.
     kwb [8/21/06]: unless Jesse O can provide a solution, this method doesn't
          work. the problem appears to be that the events are generated too
          quickly for windows to handle. slowing the rate down pushes
          performance way below what i can tolerate. so, switch back to
          traditional NI-DAQ.
     kwb [9/12/06]: add DAQ_find_offset2() as improved method for finding the
          offset, update SCAN parameters to read faster and more points in
          DAQ_get_data()
     kwb [11/05/06]: update DAQ sample size and rate
*************************************************************************/

/* Include files */
#include <stdio.h>
#include <windows.h>
#include <mmsystem.h>
#include <math.h>

#include "nidaqex.h"
#include "nidaq.h"
#include "JoyDAQ Traditional NIDAQ.h"
#include "JoyAlgorithms.h"

#define PI 3.141592653589

/* Unit variables */
float X_Offset;            /* x-offset value */
```

```
float Y_Offset;            /* y-offset value */

/*
* PURPOSE:
*    this function initializes the NI-DAQcard. it also gets the start time
*    for the MS_PFA_FA.
*/
int DAQ_initialize(void)
{
/***** DECLARATIONS *****/

     /* Local Variables */

     /* Device Settings */
     i16 iInputMode = 0;         /* 1 for RSE, 0 for DIFF */
     i16 iInputRange = 0;        /* ignored */
     i16 iPolarity = 0;          /* 0=bipolar, 1=unipolar */
     i16 iDriveAIS = 1;          /* ignored */
     i32 lTimeout = 90;

     /* Debugging */
     i16 iStatus = 0;
     i16 iRetVal = 0;
     i16 iIgnoreWarning = 0;
     i16 iStatusReturn = 0;

     /***** CONFIGURE INPUT SETTINGS *****/
     /* this needs to be done only once */

     /* turn on dither */
     /* this doesn't seem to help - kwb - 10/12 */
     //iStatus = MIO_Config(DAQ_DEVICE, 1, 0);
     //iRetVal = NIDAQErrorHandler(iStatus, "MIO_Config", iIgnoreWarning);
     //if (DEBUG_lo) printf("iStatus, MIO_Config = %d\n", iStatus);

     /* channel 0 */
     iStatus = AI_Configure(DAQ_DEVICE, ACH0, iInputMode, iInputRange,
          iPolarity, iDriveAIS);
     if (iStatus) iStatusReturn = iStatus;
     iRetVal = NIDAQErrorHandler(iStatus, "AI_Configure", iIgnoreWarning);
     if (DEBUG_lo) printf("iStatus, AI_Configure channel %d = %d\n", ACH0,
          iStatus);
     /* channel 1 */
     iStatus = AI_Configure(DAQ_DEVICE, ACH1, iInputMode, iInputRange,
          iPolarity, iDriveAIS);
     if (iStatus) iStatusReturn = iStatus;
     iRetVal = NIDAQErrorHandler(iStatus, "AI_Configure", iIgnoreWarning);
     if (DEBUG_lo) printf("iStatus, AI_Configure channel %d = %d\n", ACH1,
          iStatus);

     /* timeout */
     /* This sets a timeout limit (#Sec * 18ticks/Sec) so that if there
        is something wrong, the program won't hang on the SCAN_Op call. */
     iStatus = Timeout_Config(DAQ_DEVICE, lTimeout);
     if (iStatus) iStatusReturn = iStatus;
     iRetVal = NIDAQErrorHandler(iStatus, "Timeout_Config", iIgnoreWarning);
     if (DEBUG_lo) printf("iStatus, Timeout_Config = %d\n", iStatus);
```

```
    /* initialize offset (probably not needed, but just in case) */
    X_Offset = 0.0;
    Y_Offset = 0.0;

    if (DEBUG_hi)
        printf("\nThe time for opening the ICD is %d\n", timeGetTime());

    return(iStatusReturn);
}

/*
* PURPOSE:
*   this function finds the offset so that zero input results in zero
*   output
*/
int DAQ_find_offset(void)
{
    /***** DECLARATIONS *****/
    /* Local Variables */
    int i;

    /* Device Settings */
    i16 numChan = 2;       /* number of channels to read */
    i16 iReadings = 100;   /* number of readings per channel */
    static i16 iGain[2] = {10, 10}; /* input gain */
    static i16 iChan[2] = {ACH0, ACH1}; /* which channel(s) to read */
    i16 iCount = numChan*iReadings; /* number of readings in one go */
    f64 iSampleRate = 25000;
    f64 iScanRate = 0.0;   /* this should get the maximum scanning rate */

    /* Device Readings */
    static i16 AI_buffer[200] = {0};
    static i16 x_buffer[100] = {0};
    static i16 y_buffer[100] = {0};
    double loc_x_reading, loc_y_reading; /* local copies of the readings */

    /* Debugging */
    i16 iStatus = 0;
    i16 iRetVal = 0;
    i16 iIgnoreWarning = 0;
    i16 iStatusReturn = 0;

    /***** START SCAN *****/
    /* high level function scans selected channel(s) */

    /* read the channels specified by iChan */
    iStatus = SCAN_Op(DAQ_DEVICE, numChan, iChan, iGain, AI_buffer,
        iCount, iSampleRate, iScanRate);
    if (iStatus) iStatusReturn = iStatus;
    iRetVal = NIDAQErrorHandler(iStatus, "SCAN_op", iIgnoreWarning);
    if (DEBUG_lo) printf("iStatus, SCAN_op = %d\n", iStatus);

    /***** CHECKING *****/
    /* not needed if we do high level function scans */

    /***** SIGNAL PROCESSING *****/
```

```
    /* this isn't in the manual, but this is needed so that there is only
   one output from the function */

    /* demuliplex the AI_buffer */
    iStatus = SCAN_Demux(AI_buffer, iCount, numChan, 0);
    if (iStatus) iStatusReturn = iStatus;
    iRetVal = NIDAQErrorHandler(iStatus, "SCAN_Demux", iIgnoreWarning);
    if (DEBUG_lo) printf("iStatus, SCAN_Demux = %d\n", iStatus);

    for (i=0; i<iReadings; i++) {
        x_buffer[i] = AI_buffer[ACH0*iReadings + i];
        y_buffer[i] = AI_buffer[ACH1*iReadings + i];
    }

    /* take means to mitigate the noise, hopefully */
    iRetVal = NIDAQMean(x_buffer, iReadings, WFM_DATA_I16, &loc_x_reading);
    iRetVal = NIDAQMean(y_buffer, iReadings, WFM_DATA_I16, &loc_y_reading);

    if (DEBUG_lo) {
        /* display results */
        printf("\nx-reading in DAQ_get_data: %lf\n", loc_x_reading);
        printf("y-reading in DAQ_get_data: %lf\n", loc_y_reading);
    }

    /* Output Result */
    X_Offset = (float)loc_x_reading;
    Y_Offset = (float)loc_y_reading;

    return(iStatusReturn);
}

/*
* PURPOSE:
*   this is an alternate method to get the offset reading
*/
int DAQ_find_offset2(void)
{
    int i, iStatus;
    const int num_samps = 40;
    float tmp_x_offset[50];
    float tmp_y_offset[50];
    double x_offset, y_offset;

    double tmp_x = 0;
    double tmp_y = 0;

    X_Offset = 0.0;
    Y_Offset = 0.0;
    for (i=0; i<num_samps; i++)
    {
        iStatus = DAQ_get_data(&tmp_x_offset[i], &tmp_y_offset[i]);
    }

    for (i=0; i<num_samps; i++)
    {
        tmp_x += tmp_x_offset[i];
        tmp_y += tmp_y_offset[i];
```

```
    }
    x_offset = tmp_x/num_samps;
    y_offset = tmp_y/num_samps;

    X_Offset = (float)x_offset;
    Y_Offset = (float)y_offset;

    return (iStatus);
}

/*
* PURPOSE:
*   this function will read the data from the NI-DAQcard
*
* OUTPUT:
*   x_reading         dithered direction
*   y_reading         dithered speed
*
*/
int DAQ_get_data(float *x_reading, float *y_reading)
{
    /***** DECLARATIONS *****/
    /* Local Variables */
    int i;            /* for loop counter */

    /* Device Settings */
    i16 numChan = 2;       /* number of channels to read */
    i16 iReadings =40;     /* number of readings per channel */
    static i16 iGain[2] = {10, 10}; /* input gain */
    static i16 iChan[2] = {ACH0, ACH1}; /* which channel(s) to read */
    i16 iCount = numChan*iReadings; /* number of readings in one go */
    f64  iSampleRate = 75000;
    f64  iScanRate =   0.0; /* this should get the maximum scanning rate */
    f64  dGainAdjust = 1.0;
    f64  dOffset =     0.0;

    /* Device Readings */
    f64 dVoltage1 = 0.0;
    i16 dReading1 = 0;
    static i16 AI_buffer[150] = {0};
    static f64 Volt_buffer[150] = {0.0};
    static i16 ICD_x_data[75] = {0};
    static i16 ICD_y_data[75] = {0};
    double loc_x_reading, loc_y_reading;/* local copies of the readings */

    /* Debugging */
    i16 iStatus = 0;
    i16 iRetVal = 0;
    i16 iIgnoreWarning = 0;
    i16 iStatusReturn = 0;

    /***** START SCAN *****/
    /* high level function scans selected channel(s) */

    /* read the channels specified by iChan iCount number of times */
    iStatus = SCAN_Op(DAQ_DEVICE, numChan, iChan, iGain, AI_buffer,
        iCount, iSampleRate, iScanRate);
```

```
    if (iStatus) iStatusReturn = iStatus;
    iRetVal = NIDAQErrorHandler(iStatus, "SCAN_op", iIgnoreWarning);
    if (DEBUG_lo) printf("iStatus, SCAN_op = %d\n", iStatus);

    /***** CHECKING *****/
    /* not needed if we do high level function scans */


    /***** SIGNAL PROCESSING *****/
    /* this isn't in the manual, but this is needed so that there is only
        one output from the function */

    /* demuliplex the AI_buffer */
    iStatus = SCAN_Demux(AI_buffer, iCount, numChan, 0);
    if (iStatus) iStatusReturn = iStatus;
    iRetVal = NIDAQErrorHandler(iStatus, "SCAN_Demux", iIgnoreWarning);
    if (DEBUG_lo) printf("iStatus, SCAN_Demux = %d\n", iStatus);

    for (i=0; i<iReadings; i++) {
        ICD_x_data[i] = AI_buffer[ACH0*iReadings + i];
        ICD_y_data[i] = AI_buffer[ACH1*iReadings + i];
    }

    /* take means to mitigate the noise, hopefully */
    iRetVal = NIDAQMean(ICD_x_data, iReadings, WFM_DATA_I16,
        &loc_x_reading);
    iRetVal = NIDAQMean(ICD_y_data, iReadings, WFM_DATA_I16,
        &loc_y_reading);

    /* adjust for offset */
    loc_x_reading = loc_x_reading - X_Offset;
    loc_y_reading = loc_y_reading - Y_Offset;

    if (DEBUG_lo) {
        /* display results */
        printf("\nx-reading in DAQ_get_data: %lf\n", loc_x_reading);
        printf("y-reading in DAQ_get_data: %lf\n", loc_y_reading);
    }

    /* return results */
    *x_reading = (float)loc_x_reading;
    *y_reading = (float)loc_y_reading;
    return(iStatusReturn);
}



/* this function closes the NI-DAQcard */
int DAQ_close_device(void)
{
    /***** DECLARATIONS *****/
    /* Local Variables */

    /* Debugging */
    i16 iStatus = 0;
    i16 iStatusReturn = 0;
```

```
    /***** CLEAN UP *****/
    /* this goes at the very end, i think */

    /* turn off any DAQ operation */
    /* oddly, DAQ_clear is undefined. so comment it out for the moment
    iStatus = DAQ_clear(DAQ_DEVICE);
    iRetVal = NIDAQErrorHandler(iStatus, "DAQ_clear", iIgnoreWarning);
    if (DEBUG_lo) printf("iStatus, DAQ_clear = %d\n", iStatus);
    */

    /* disable timeouts */
    iStatus = Timeout_Config(DAQ_DEVICE, -1);
    if (iStatus) iStatusReturn = iStatus;
    if (DEBUG_lo) printf("iStatus, Timeout_Config = %d\n", iStatus);

    /* Get and display current time */
    if (DEBUG_hi)
        printf("\nThe time of closing the ICD is %d\n", timeGetTime());

    return(iStatusReturn);
}
```

APPENDIX F

SOURCE CODE IMPLEMENTATION OF INPUT CONTROL ALGORITHMS

F.1 JOYALGORITHMS.H

```
/*************************************************************************
     last modified: 11/06/06
     by: kwb

     JoyAlgorithms.h

     DESCRIPTION
     this is the header file for functions used to process user input signals

     HISTORY
     kwb [7/13/06]: original version. based on version 1.0 of VCJ_ICD project
     kwb [7/17/06]: updated ICD_VGA to implement MJ STF
     kwb [7/21/06]: moved Getsettings() to here
     kwb [8/1/06]: added neville() to interpolate missing points
     kwb [8/2/06]: added algorithm wrapper ICD_apply_algorithm(), added
          ICD_algorithms from Save_Data.h
     kwb [9/26/06]: added functions to support the VGA and STF algorithms
     kwb [10/10/06]: make (+) and (-) axis gains different for each axis by...
          adding an extra dimension to Gain, k1, and k2;
     kwb [11/06/06]: added functions for loading FA params
*************************************************************************/

/*** type definitions ***/
typedef struct {
     DWORD start_time;      /* start time of current phase */
     int previous_mode;     /* 0=active phase, 1=recovery phase */
     int current_mode;      /* 0=active phase, 1=recovery phase */
     float Gain[2][2];      /* current gain; i=dir,spd; j=neg/pos */
     float k1[2][2];        /* last active and recovery phase gains, */
     float k2[2][2];        /*  respectively; i=dir,spd; j=neg/pos */
} fatigue_params;

enum ICD_algorithms {
     VGA,
```

```
    MS_PFA,
    MS_PFA_FA,
    MJ_STF
};

/*** prototypes ***/
int MS_PFA_initialize(void);
void Getsettings(void);
int FA_load_defaults(void);
int FA_load_last_saved(void);
int ICD_apply_algorithm(enum ICD_algorithms, float, float);
int ICD_VGA(bool, float, float);
double ICD_dig2force(int, float);
double ICD_polyeval(double, double [], int);
int ICD_calibration_constants(void);
float neville(float, int, DWORD [], float []);
int ICD_MS_PFA(char, float, float);

/*** globals ***/
/* debugging characteristics */
#define DEBUG_ALL 0
#define DEBUG_hi  1    /* hi-level debugging */
#define DEBUG_lo  0    /* lo-level debugging */
/* math */
#define Pi            3.141592653589
```

F.2 JOYALGORITHMS.CPP

```
/*****************************************************************************
    modified: 11/05/06
    by: kwb

    JoyAlgorithms.cpp

    PURPOSE
    This file contains the function that execute the algorithm transfer
    functions

    INPUTS
    ddirection   direction component of joystick input [0-4096]
    dspeed       speed component of joystick input [0-4096]

    OUTPUTS
    data         global array containing joystick input parameters

    HISTORY
    kwb [07/12/06]: added VGA and MS_PFA_FA algorithms, based on version 1.3
         of ICD_functions.c, renamed to JoyAlgorithms.cpp, replaced
         DataTransform() with ICD_MS_PFA()
    kwb [7/15/06]: replace Gain_* with Gain_min[]
    kwb [7/16/06]: fix bug with VGA algorithm
```

```
     kwb [7/17/06]: added standard transfer function for MJ to VGA algorithm
     kwb [7/21/06]: implement adaptive tremor filter, set values for VGA and
          MJ_STF dead zone and gain
     kwb [7/28/06]: added highpass params to cWFLC class
     kwb [8/2/06]: added ICD_apply_algorithm() as a wrapper for the real
          algorithm functions
     kwb [8/22/06]: make it so adaptive filter is used only while the race
          track screen is active (in case data acq is continuous)
     kwb [9/12/09]: fix bug with Subject_ID, replace function call to
          DAQ_find_offset() with DAQ_find_offset2()
     kwb [9/26/06]: modify VGA and STF to take into account asymmetry in the
          strain gage bridges' responses, replace Pi with math.h's M_PI
     kwb [10/10/06]: make (+) and (-) axis gains different for each axis by...
          adding an extra dimension to Gain_min, Gain_max, FA.Gain, FA.k1;
          computing dir and spd input based on their signs
     kwb [11/05/06]: update MJ_STF params, subtract DZ value from TMPT value
          in MJ_STF and VGA algorithms, update initialization of Subject_ID
     kwb [11/06/06]: protect against crashes by saving and loading MS_PFA_FA
          parameters to and from fa_params.txt file.
     kwb [11/28/06]: the -x and -y axes weren't working though the joystick
          was providing signals. change the equations to be linear rather than
          exponential.
*************************************************************************/

#define _USE_MATH_DEFINES  1
#include <math.h>
#include <stdio.h>
#include <direct.h>
#include <windows.h>
#include <mmsystem.h>
#include "JoyAlgorithms.h"
#include "cWFLC.h"
#include "VirtualDriving_demo.h"    //contains ActiveScreen definitions

/* globals */
extern unsigned int M_dir; //used in save_data_sampele()
extern unsigned int M_spd;
extern unsigned int ActiveScreen;   //defined in VirtualDriving_demo.cpp
int DEBUG_FILTS = 0;
int OUTPUT_FA_PARAMS = 1;  //whether FA params should be saved to data file
FILE *fp_fa;               //file pointer for fa_params.txt
fpos_t fa_start_pos;       //position in fp_fa for writing fa params

/* unit variables */
char Subject_ID[10];// = 'MSJS102';
float DIG_zero = 0.0;

/* filter parameters */
bool DO_WFLC = true;
bool DO_HPF = true;
cWFLC wflc[2];

/* MS_PFA parameters */
int Dead_Zone_Shape_Status;  //if zero none, 1 =  elliptical includes circle,
2 = rectangular and square
int Dead_Zone_Rad_X;  //Dead radius, 0 to 2048
int Dead_Zone_Rad_Y;  //Dead radius, 0 to 2048
```

```
int Template_Shape_Status;  //0 = none, 1 = elliptical, 2 = astroid, 3 =
diamond
int Template_Rad_X;
int Template_Rad_Y;

int Bias_Axis_Enabled_Status;       /* 0=not used, 1=used */
int Bias_Axis_Angle;                /* degrees */

int Gain_Status;
float Gain_multiplier;
float Gain_min[2][2];  /* mininmum (baseline) gain; i=dir,spd; j=neg,pos */
float Gain_max[2][2];  /* maximum gain; i=dir,spd; j=neg,pos axis */
float Alpha;           /* rate of increase while active */
float Beta;            /* recovery rate while inactive */
fatigue_params FA;

/* VGA parameters */
float DZ_boundary;     /* VGA Dead Zone */
float Gain;            /* VGA Gain */
float TMPT_boundary;   /* VGA Template Boundar */

/* output parameter */
extern float data[];

/* unit parameters */


/*
* PURPOSE:
*    this function reads the settings parameters from the settings files
*/
void Getsettings()
{
     /* local variables */
     int i;
     char line_buffer[100];
     FILE *file;
     int axis;     /* dir=0; spd=1 */
    int sign;     /* neg=0; pos=1 */

     /* first load MS_PFA parameters */
     if((file = fopen("C:\\settings\\setup.txt","r")) != NULL)
     {
          fgets(line_buffer,90,file);  // read line 1 from the settings file;
          for (i = 0; i < 7; i++)
          {
               if (line_buffer[i] != '\t')
                    Subject_ID[i] = line_buffer[i];
               else
                    break;
          }

          fgets(line_buffer,90,file);  // read line 2 from the settings file;
          Dead_Zone_Shape_Status = atoi(line_buffer);

          fgets(line_buffer,90,file);  // read line 3  from settings file;
```

```
        Dead_Zone_Rad_X = atoi(line_buffer); //Dead zone radius Direction, 0
to 2048

        fgets(line_buffer,90,file);  //read line 4  from settings file;
        Dead_Zone_Rad_Y = atoi(line_buffer); //Dead zone radius Speed, 0 to
2048

        fgets(line_buffer,90,file);  // read line 5  from settings file;
        Template_Shape_Status = atoi(line_buffer);

        fgets(line_buffer,90,file);  // read line 6  from settings file;
        Template_Rad_X = atoi(line_buffer);

        fgets(line_buffer,90,file);  // read line 7  from settings file;
        Template_Rad_Y = atoi(line_buffer);

        fgets(line_buffer,90,file);  // read line 8 from settings file;
        Bias_Axis_Enabled_Status = atoi(line_buffer);

        fgets(line_buffer,90,file);  // read line 9 from settings file;
        Bias_Axis_Angle = atoi(line_buffer);

        fgets(line_buffer,90,file);  // read line 10 from settings file;
        Gain_Status = atoi(line_buffer);

        // min/default gain
        for (axis=0; axis<=1; axis++)
        {
            for (sign=0; sign<=1; sign++)
            {
               fgets(line_buffer,90,file);  // read lines 11-14
                 Gain_min[axis][sign] = atof(line_buffer);
            }
        }

        // max gain
        for (axis=0; axis<=1; axis++)
        {
            for (sign=0; sign<=1; sign++)
            {
               fgets(line_buffer,90,file);  // read lines 11-14
                 Gain_max[axis][sign] = atof(line_buffer);
            }
        }

        fgets(line_buffer,90,file);  // read line 19 from settings file;
        Gain_multiplier = atof(line_buffer);

        fgets(line_buffer,90,file);  // read line 20 from settings file;
        Alpha = atof(line_buffer);

        fgets(line_buffer,90,file);  // read line 21 from settings file;
        Beta = atof(line_buffer);

        fclose(file);
    }
    else
```

```
    {
         Dead_Zone_Shape_Status = 1;//if zero none, 1 = elliptical includes
                                    // circle, 2 = rectangular and square
         Dead_Zone_Rad_X = 100;  //Dead radius, 0 to 2048
         Dead_Zone_Rad_Y = 100;  //Dead radius, 0 to 2048

         Template_Shape_Status = 1;  //0 = none, 1 = elliptical, 2=asteroid,
3 = diamond
         Template_Rad_X = 2048;
         Template_Rad_Y = 2048;

         Bias_Axis_Enabled_Status = 0;
         Bias_Axis_Angle = 0;

         Gain_Status = 1;
         for (axis=0; axis<=1; axis++)
         {
              for (sign=0; sign<=1; sign++)
              {
                 Gain_min[axis][sign] = 1.0;
                   Gain_max[axis][sign] = 2.0;
              }
         }
         Gain_multiplier = 0.333f;

         Alpha = 0.01f;
         Beta = 0.01f;
    }

    /* now load WFLC parameters */
    if((file = fopen("C:\\settings\\wflc_setup.txt","r")) != NULL)
    {
         /* direction paramters are first */
         for (axis=0; axis<=1; axis++)
         {
              /* read high pass filter params */
              fgets(line_buffer,90,file);
              wflc[axis].iir_order = atoi(line_buffer);

              fgets(line_buffer,90,file);
              wflc[axis].iir_corner = atof(line_buffer);

              fgets(line_buffer,90,file);
              wflc[axis].iir_samp_rate = atof(line_buffer);

              fgets(line_buffer,90,file);
              wflc[axis].iir_gain = atof(line_buffer);

              /* read WFLC params */
              fgets(line_buffer,90,file);
              wflc[axis].mu = atof(line_buffer);

              fgets(line_buffer,90,file);
              wflc[axis].mu0 = atof(line_buffer);

              fgets(line_buffer,90,file);
              wflc[axis].mub = atof(line_buffer);
```

```
            fgets(line_buffer,90,file);
            wflc[axis].M = atoi(line_buffer);

            fgets(line_buffer,90,file);
            wflc[axis].w0 = atof(line_buffer);

            fgets(line_buffer,90,file);
            wflc[axis].w1 = atof(line_buffer);

            fgets(line_buffer,90,file);
            wflc[axis].wMp1 = atof(line_buffer);
        }

        fclose(file);
    }
    else
    {
        for (axis=0; axis<=1; axis++)
        {
            wflc[axis].iir_order = 2;
            wflc[axis].iir_corner = 2.0;
            wflc[axis].iir_samp_rate = 1.0/0.0170;
            wflc[axis].iir_gain = 1.0;
            wflc[axis].mu = 0.009;
            wflc[axis].mu0 = 1.2e-5;
            wflc[axis].mub = 0;
            wflc[axis].M = 4;
            wflc[axis].w0 = 3.75*2*M_PI;
            wflc[axis].w1 = 0;
            wflc[axis].wMp1 = 0;
        }
        if (DEBUG_FILTS) {
        }

    }
}


/*
* PURPOSE:
*    this function initializes the fatigue adaptation parameters
*/
int MS_PFA_initialize(void)
{
    int iStatusReturn = 0; //what would make this 0?
    DWORD TMPstart_time;   //this also serves as an indicator for how the sim
                           // last exited. if 0, last exit was safe
    char *line_buffer;

    //allocate memory for line_buffer
    line_buffer = (char *)malloc(90);

    /* initialize fatigue compensation parameters */
    /* if it exists, check the first param; if not, load default params */
    if ((fp_fa = fopen("C:\\Settings\\fa_params.txt", "r")) != NULL)
    {
```

```
        /* read line 1 */
        fgets(line_buffer,90,fp_fa);
        TMPstart_time = atoi(line_buffer);
        /* decide which params to use */
        if (TMPstart_time == 0)
            iStatusReturn = FA_load_defaults();
        else
        {
            FA.start_time = TMPstart_time;
            iStatusReturn = FA_load_last_saved();
        }
        fclose(fp_fa);
    }
    else
        FA_load_defaults();

    /* open fa_params.txt file for writing during program exection */
    fp_fa = fopen("C:\\Settings\\fa_params.txt", "w");
    /* get start position */
    if (fgetpos(fp_fa, &fa_start_pos) != 0)
        OutputDebugString("fgetpos error");

    /* initialize WFLC parameters */
    if (!(wflc[0].initialize()))
        OutputDebugString("===> WFLC not initialized\n");
    if (!(wflc[1].initialize()))
        OutputDebugString("===> WFLC not initialized\n");

    /* these are used in save_data_sample() */
    M_dir = wflc[0].M;
    M_spd = wflc[1].M;

    /* free line_buffer memory */
    free(line_buffer);

    return(iStatusReturn);
}


/*
* PURPOSE:
*   this function loads the FA params based on the settings.txt file
*/
int FA_load_defaults(void)
{
    int i, j;

    if (DEBUG_lo)
        OutputDebugString("\nloading default fa params...\n");

    FA.start_time = timeGetTime();
    FA.previous_mode = 1;
    FA.current_mode = 1;
    for (i=0; i<2; i++) {
        for (j=0; j<2; j++) {
            FA.Gain[i][j] = Gain_min[i][j];
            FA.k1[i][j] = Gain_min[i][j];
```

```
        }
    }

    return(0);
}

/*
* PURPOSE:
*    this function loads the FA params from the backup file
*/
int FA_load_last_saved()
{
    int i, j;
    char line_buffer[50];
    char fa_str_buffer[50];
    int iStatusReturn = 0; //what would make this non-zero?

    if (DEBUG_lo)
        OutputDebugString("\nloading last saved fa params...\n");

    fgets(line_buffer,90,fp_fa);  // read line 2
    FA.previous_mode = atoi(line_buffer);

    /* check to see that we're getting the right data */
    if (DEBUG_lo) {
        sprintf(fa_str_buffer, "previous mode [should be 1 or 0]: %d\n",
            FA.previous_mode);
        OutputDebugString(fa_str_buffer);
    }

    for (i=0; i<2; i++) {
        for (j=0; j<2; j++) {
            fgets(line_buffer,90,fp_fa);  // read lines 3-6
            FA.Gain[i][j] = atof(line_buffer);
        }
    }

    for (i=0; i<2; i++) {
        for (j=0; j<2; j++) {
            fgets(line_buffer,90,fp_fa);  // read lines 7-10
            FA.k1[i][j] = atof(line_buffer);
        }
    }

    for (i=0; i<2; i++) {
        for (j=0; j<2; j++) {
            fgets(line_buffer,90,fp_fa);  // read lines 11-14
            FA.k2[i][j] = atof(line_buffer);
        }
    }

    return(iStatusReturn);
}


/*
* PURPOSE:
```

```
*    this is a wrapper function to make things a little more pretty in the
*    main virtual driving loop
*
* INPUTS:
*    current_ICD_algorithm  like the name says
*    x                      direction reading
*    y                      speed reading
*
* OUTPUT:
*    iStatusReturn          for debugging
*/
int ICD_apply_algorithm(enum ICD_algorithms current_ICD_algorithm, float x,
     float y)
{
     int iStatusReturn;

     switch (current_ICD_algorithm)
     {
          case VGA:
               iStatusReturn = ICD_VGA(1, x, y);
               break;
          case MS_PFA:
               iStatusReturn = ICD_MS_PFA(0, x, y);
               break;
          case MS_PFA_FA:
               iStatusReturn = ICD_MS_PFA(1, x, y);
               break;
          case MJ_STF:
               iStatusReturn = ICD_VGA(0, x, y);
               break;
     }

     return(iStatusReturn);
}

/*
* PURPOSE:
*    this function applies the VGA or the MJ's STF algorithm to the signal. it
*    differs from the original VGA in that units are in terms of force rather
*    than the digital conversion of the signals off the strain gages.
*         DIG_zero is the digital output representing neutral input
*         DZ_boundary is 1.11 N
*         TMPT_boundary is set such that the corresponding digital out
*              produces the desired max speed of the wheelchair (~2.25 m/s or
*              ~7 ft/sec). That is, 2.25 m/s = ICD_dig2vel(TMPT_boundary).
*         Gain is set such that 6.63 N produces a template violation.
*              Gain = TMPT_boundary/(6.63 - 1.11 N)
*
* INPUT:
*    do_vga        tells the function which set of dead zone, gain, and
*                  template settings should be loaded
*    dir           pre-algorithm direction
*    spd           pre-algorithm speed
* OUTPUT:
*    data          data[0] is dir; data[1] is spd; output via extern
*    iStatusReturn success of function: 0=good, 1=deadzone violation,
*                       2=template violation
```

```
*/
int ICD_VGA(bool do_vga, float dir, float spd)
{
     /***** DECLARATIONS and INITIALIZATIONS *****/
     /* Local Variables */
     double dir_f, spd_f;   /* force input (N) */
     float input_magnitude_sq;  /* magnitude of user input squared */
     float theta;           /* angle of user input */
     int DzoneFlag = 0;     /* 1=inside deadzone, 0=outside deadzone */
     float DZ_x, DZ_y;      /* location on dead zone corresponding to input */
     int TempFlag = 0;      /* 1=outside template, 0=inside template */
     float TMPT_x, TMPT_y;  /* location on template corresponding to input */

     /* Debugging */
     int iStatusReturn = 0; /* 0=good, 1=deadzone violation, 2=template
                               violation */
     /* algorithm parameters [hard coded] */
     if (do_vga)
     {
          DZ_boundary = 1.1069;
          Gain = 371.0;
          TMPT_boundary = 6.6278;
     }
     else //MJ STF
     {
          DZ_boundary = 1.018;
          Gain = 691.0;
          TMPT_boundary = 3.982;
     }

     /***** APPLY THE ALGORITHM *****/
     /* Offset the axes so that (0,0) marks the origin */
     dir = dir - DIG_zero;
     spd = spd - DIG_zero;

     /* translate digital signal to force */
     dir_f = ICD_dig2force(0, dir);
     spd_f = ICD_dig2force(1, spd);

     /* find magnitude of user input squared */
     input_magnitude_sq = dir_f*dir_f + spd_f*spd_f;

     /* determine the location of the dead zone boundary */
     theta = (float)atan2(spd_f, dir_f);
     DZ_x = DZ_boundary*(float)cos(theta);
     DZ_y = DZ_boundary*(float)sin(theta);

     /* If within the deadzone, set input to zero and return */
     if (input_magnitude_sq <= (DZ_boundary*DZ_boundary))
     {
          dir = 0;
          spd = 0;
          DZ_x = 0;
          DZ_y = 0;
          TMPT_x = 0;
          TMPT_y = 0;
          DzoneFlag = 1;
```

```
        iStatusReturn = 1;
    }

    if (!DzoneFlag)
    {
        /* Make transition smooth outside of deadzone, keeping direction */
        dir_f = dir_f - DZ_x;
        spd_f = spd_f - DZ_y;
        /* apply same translation to TMPT_boundary (it's a circular tmpt) */
        TMPT_boundary = TMPT_boundary - DZ_boundary;

        /* Determine if template has been exceeded */
        /* recalculate magnitude of user input */
        input_magnitude_sq = dir_f*dir_f + spd_f*spd_f;

        /* determine the location of the template boundary */
        TMPT_x = TMPT_boundary*(float)cos(theta);
        TMPT_y = TMPT_boundary*(float)sin(theta);

        /* if outside the template boundary, set input to the corresponding
        location on the template */
        if ((input_magnitude_sq) >= (TMPT_boundary*TMPT_boundary)) {
            dir_f = TMPT_x;
            spd_f = TMPT_y;
            TempFlag = 1;
            iStatusReturn = 2;
        }

        /* Apply gain and convert back to digital units */
        dir = Gain*dir_f;
        spd = Gain*spd_f;
    }

    /* Reset the origin to the original position */
    data[0] = dir + DIG_zero;
    data[1] = spd + DIG_zero;
    data[2] = DZ_x + DIG_zero;
    data[3] = DZ_y + DIG_zero;
    data[4] = TMPT_x + DIG_zero;
    data[5] = TMPT_y + DIG_zero;
    data[6] = DzoneFlag;
    data[7] = TempFlag;

    if (DEBUG_lo) printf("ICD_VGA x: %lf\ty: %lf\n", data[0], data[1]);
    return(iStatusReturn);
}


/*
* PURPOSE:
*   this function transforms the digital signal to the representative force
*
* INPUTS:
*   axis     which axis is being used [0=dir; 1=spd]
*   dig          digital signal to be converted
* OUTPUT:
*   force        corresponding force (N)
```

```
*/
double ICD_dig2force(int axis, float dig)
{
     double force;
//   double a[5];
     switch (axis)
     {
          case 0:       //dir
               if (dig >= 0) //right
                    force = 0.01064*dig;
               else          //left
               {
                    //double a[5] = {
                    //   6.3121e-13, 2.3122e-9, -2.5613e-7, 9.3217e-3, 6.9416e-2
                    //};
                    //force = ICD_polyeval(dig, a, 4);
                    force = 0.0113*dig;
               }
               break;
          case 1:       //spd
               if (dig >= 0) //forward
                    force = 0.009994*dig;
               else          //reverse
               {
                    //double a[5] = {
                    // 3.2298e-13, 1.4706e-9, -4.1389e-7, 1.1160e-3, -5.1782e-2
                    //};
                    //force = ICD_polyeval(dig, a, 4);
                    force = 0.0129*dig;
               }
               break;
     }
     return(force);
}


/*
* PURPOSE:
*    this function evaluates an nth order polynomial
*
* INPUTS:
*    x        number to be evaluated
*    coef     array holding the coefficients, must be of size n+1
*    n        order of polynomial
*
* OUTPUT:
*    y        result
*/
double ICD_polyeval(double x, double coef[], int n)
{
     int i;
     double y = 0;
     for (i=0; i<=n; i++)
          y += coef[i]*pow(x,n-i);
     return(y);
}
```

```
/*
* PURPOSE:
*   this function stands in place of the calibration file to initialize
*   constants
*/
int ICD_calibration_constants(void)
{
    int i, j;

    Bias_Axis_Enabled_Status = 0;   /* 0=not used, 1=used */
    Bias_Axis_Angle = 5;            /* degrees */

    Dead_Zone_Shape_Status = 0;     /* 0=none, 1=eliptical, 2=rectangular */
    Dead_Zone_Rad_X = 300;
    Dead_Zone_Rad_Y = 300;

    Template_Shape_Status = 1;      /* 1=eliptical, 2=rectangular */
    Template_Rad_X = 2000;          /* 3=astroid, 4=diamond */
    Template_Rad_Y = 2000;          /* post gain values */

    for (i=0; i<2; i++) {
        for (j=0; j<2; j++) {
            Gain_min[i][j] = 1.0f; /* mininmum (baseline) gain */
            Gain_max[i][j] = 2.0f; /* maximum gain */
        }
    }

    Alpha = 0.000001f;     /* rate of increase while active */
    Beta = 0.000002f;      /* recovery rate while inactive */

    return(0);
}

/*
* PURPOSE:
*   this function provides an interpolated joystick input based on
surrounding
*   inputs using Neville's Iterated Interpolation method.
*
* NOTES:
*   this code is adapted from http://www.cs.uaf.edu/~bueler/nevM.htm 08/01/06
*
* INPUTS:
*   xx       time stamp of missing input
*   n        order of interpolation (n+1 = # of points)
*   x        time stamps of known inputs
*   Q        value of known inputs
* OUTPUT:
*   Q[0] p(xx)
*/
float neville(float xx, int n, DWORD x[], float Q[])
{
    int i, j;

    for (i=n; i>=1; i--)
    {
        for (j=1; j<=i; j++)
```

```
        {
            Q[j-1] = (xx-x[j-1])*Q[j] - (xx-x[j+n-i])*Q[j-1];
            Q[j-1] = Q[j-1]/(x[j+n-i]-x[j-1]);
        }
    }

    return(Q[0]);
}


/*
* PURPOSE:
*   this function is the driver for the MS_PFA and MS_PFA_FA
*
* INPUT:
*   do_fatigue   boolean for if gain should be adjusted for fatigue (1=yes)
* INPUT/OUTPUT:
*   x_reading    post-algorithm direction
*   y_reading    post-algorithm speed
*/
int ICD_MS_PFA(char do_fatigue, float dir, float spd)
{
    /***** DECLARATIONS *****/

    /* Local Parameters */
    int i, j;

    /* Algorithm Parameters */
    float angle, L1, L2;
    float one, two;

    /* for the filters */
    double dir_hpf, spd_hpf;

    /* for the bias axes */
    double BiasAngle;
    float dir_tmp;//prevents input dir from being overwritten during rotation
    /* if (DEBUG_lo) printf("BiasAngle = %.4d\n", BiasAngle); */

    /* for the dead zone */
    int DzoneFlag = 0;     /* 1=inside deadzone, 0=outside deadzone */
    float DZangle;         /* for rectangular deadzone, angle to vertex */
    float DZ_x, DZ_y;      /* local x- and y-boundary values */

    /* for the gain */
    DWORD current_time;
    DWORD delta_t;         /* change in time since phase/mode switch */

    /* for the template */
    int TP3XRad = 2000;
    int TP3YRad = 2000;
    int TempFlag = 0;      /* 1=outside template, 0=inside template */
    int Td[4];
    float TMPT_x = 0;
    float TMPT_y = 0;      /* local x- and y-boundary values */

    /* Debugging */
```

```
int iStatus = 0;
int iStatusReturn = 0; /* 0=good, 1=deadzone violation, 2+=template
                                violation */

/***** APPLY THE ALGORITHM *****/
/* offset the axes so that (0,0) marks the origin */
dir = dir - DIG_zero;
spd = spd - DIG_zero;

/* Apply Adaptive Filter to each axis */
if ((DO_WFLC) && (ActiveScreen == RACETRACKSCREEN))
{
     if (DEBUG_FILTS) {
     }
     /* Apply high-pass filter to each axis */
     dir_hpf = dir;
     spd_hpf = spd;
     if (DO_HPF)
     {
          for (i=0; i<wflc[0].filt.order; i++)
               dir_hpf = wflc[0].highpass_filter(dir_hpf, i);
          for (i=0; i<wflc[1].filt.order; i++)
               spd_hpf = wflc[1].highpass_filter(spd_hpf, i);
     }
     dir = dir - (float)wflc[0].wflc_filter(dir_hpf);
     spd = spd - (float)wflc[1].wflc_filter(spd_hpf);
     if (DEBUG_FILTS) {
     }
}

/* Bias Axis Adjustment */
/* moves both the speed and direction axes together maintains internal
   90 degree alignment */
if(Bias_Axis_Enabled_Status) {
     BiasAngle = Bias_Axis_Angle * M_PI/180.0;
     dir_tmp = dir;
     dir = (float)((cos(BiasAngle))*dir_tmp - (sin(BiasAngle))*spd);
     spd = (float)((sin(BiasAngle))*dir_tmp + (cos(BiasAngle))*spd);
}

/* Deadzone Determination */
/* find input magnitude squared */
L1 = dir*dir + spd*spd;

/* ellipse or rectangle */
switch (Dead_Zone_Shape_Status) {
     case 0:
          DZ_x = 0;
          DZ_y = 0;
          break;
     case 1:       /* elliptical deadzone */
          if (dir != 0) {       /* avoid dividing by 0 */
               /* find corresponding point on deadzone */
               L2 = spd/dir;
               angle = (float)atan2(Dead_Zone_Rad_X*L2, Dead_Zone_Rad_Y);
               one = (float)(Dead_Zone_Rad_X*cos(angle));
               two = (float)(Dead_Zone_Rad_Y*sin(angle));
```

```
            L2 = (one*one) + (two*two);

            /* thought about quadrant needs to go here */

            /* if within deadzone, set the flag; otherwise save
               intersection points */
            if (L1 < L2) DzoneFlag = 1;
            else {
                 angle = (float)atan2(spd,dir);
                 DZ_x = (float)(sqrt(L2)*cos(angle));
                 DZ_y = (float)(sqrt(L2)*sin(angle));
            }
       }
       else {                  /* dir == 0 */
            /* if within deadzone, set the flag; otherwise find
                intersection points */
            if (fabs(spd) < Dead_Zone_Rad_Y) DzoneFlag = 1;
            else {
                 DZ_x = 0.0f;
                 DZ_y =(float)(spd>0? Dead_Zone_Rad_Y:-Dead_Zone_Rad_Y);
            }
       }
       break;

  case 2:       /* rectangular deadzone */
       /* if within deadzone, set the flag; otherwise find intersection
          points */
       if ((fabs(dir)<=Dead_Zone_Rad_X) & (fabs(spd)<=Dead_Zone_Rad_Y))
            DzoneFlag = 1;
       else {
            DZangle = atan2((float)Dead_Zone_Rad_Y,
                 (float)Dead_Zone_Rad_X);
            if ((spd != 0) & (dir != 0)) {  /* avoid dividing by 0 */
                 L2 = (float)atan2(spd, dir);
                 if (((L2>=0)&(L2<=DZangle)) ||
                      ((L2<=0)&(L2>=-DZangle))) {
                      DZ_x = (float)Dead_Zone_Rad_X;
                      DZ_y = (float)(tan(L2)*Dead_Zone_Rad_X);
                 }
                 else if ((L2>DZangle) & (L2<=M_PI-DZangle)) {
                      DZ_x = (float)(Dead_Zone_Rad_Y/tan(L2));
                      DZ_y = (float)Dead_Zone_Rad_Y;
                 }
                 else if ((L2<-DZangle) & (L2>=DZangle-M_PI)){
                      DZ_x = (float)(-Dead_Zone_Rad_Y/tan(L2));
                      DZ_y = (float)-Dead_Zone_Rad_Y;
                 }
                 else {
                      DZ_x = (float)-Dead_Zone_Rad_X;
                      DZ_y = (float)(-tan(L2)*Dead_Zone_Rad_X);
                 }
            }
            else {                      /* spd or dir == 0 */
                 if ((spd==0) & (dir>=0)) {
                      DZ_x = (float)Dead_Zone_Rad_X;
                      DZ_y = 0.0f;
                 }
```

```
                    else if ((spd==0) & (dir<0)) {
                         DZ_x = (float)-Dead_Zone_Rad_X;
                         DZ_y = 0.0f;
                    }
                    else if ((dir==0) & (spd>=0)) {
                         DZ_x = 0.0f;
                         DZ_y = (float)Dead_Zone_Rad_Y;
                    }
                    else {
                         DZ_x = 0.0f;
                         DZ_y = (float)-Dead_Zone_Rad_Y;
                    }
               }
          }
          break;
}

/* if inside the deadzone, set inputs to 0 */
if (DzoneFlag) {
     dir = 0;
     spd = 0;
     DZ_x = 0;
     DZ_y = 0;
     TMPT_x = 0;
     TMPT_y = 0;
     iStatusReturn = 1;
}

/* Ensure smooth transition out of deadzone */
if (!DzoneFlag) {
     dir = dir - DZ_x;
     spd = spd - DZ_y;
}

/* Apply Gain */
if (!do_fatigue & !DzoneFlag) { /* MS_PFA and outside the deadzone */
     if (dir < 0)
          dir = Gain_min[0][0]*dir;
     else
          dir = Gain_min[0][1]*dir;
     if (spd < 0)
          spd = Gain_min[1][0]*spd;
     else
          spd = Gain_min[1][1]*spd;
}
else {                  /* MS_PFA_FA */
     /* Get time */
     current_time = timeGetTime();

     /* save the current mode */
     FA.current_mode = DzoneFlag;

     /* if mode has changed, reset start time, update previous_mode, and
     save last gains */
     if (FA.current_mode != FA.previous_mode) {
          FA.start_time = current_time;
          FA.previous_mode = FA.current_mode;
```

```
     for (i=0; i<2; i++) {
          for (j=0; j<2; j++) {
            FA.k1[i][j] = FA.Gain[i][j];
               FA.k2[i][j] = FA.Gain[i][j];
          }
     }
}

/* Determine the new gain */
delta_t = current_time - FA.start_time;
if (FA.current_mode) { /* if in the dead zone */
     for (i=0; i<2; i++) {
          for (j=0; j<2; j++) {
               FA.Gain[i][j] = Gain_min[i][j] + (FA.k1[i][j] -
                    Gain_min[i][j])*expf((-Beta*(float)delta_t));
          }
     }
}
else {                      /* not in the dead zone */
     for (i=0; i<2; i++) {
          for (j=0; j<2; j++) {
               FA.Gain[i][j] = Gain_max[i][j] - (Gain_max[i][j] -
                    FA.k2[i][j])*expf((-Alpha*(float)delta_t));
          }
     }
}

/* Save parameters to file */
if (OUTPUT_FA_PARAMS)
{
     /* set position to first position */
     if (fsetpos(fp_fa, &fa_start_pos) != 0)
          OutputDebugString("fsetpos error");

     /* write data points */
     fprintf(fp_fa, "%d\t[exit ok?] start_time\n", FA.start_time);
     fprintf(fp_fa, "%d\tprevious_mode\n", FA.previous_mode);
     fprintf(fp_fa, "%f\tgain +x\n", FA.Gain[0][0]);
     fprintf(fp_fa, "%f\tgain -x\n", FA.Gain[0][1]);
     fprintf(fp_fa, "%f\tgain +y\n", FA.Gain[1][0]);
     fprintf(fp_fa, "%f\tgain -y\n", FA.Gain[1][1]);
     fprintf(fp_fa, "%f\tk1 +x\n", FA.k1[0][0]);
     fprintf(fp_fa, "%f\tk1 -x\n", FA.k1[0][1]);
     fprintf(fp_fa, "%f\tk1 +y\n", FA.k1[1][0]);
     fprintf(fp_fa, "%f\tk1 -y\n", FA.k1[1][1]);
     fprintf(fp_fa, "%f\tk2 +x\n", FA.k2[0][0]);
     fprintf(fp_fa, "%f\tk2 -x\n", FA.k2[0][1]);
     fprintf(fp_fa, "%f\tk2 +y\n", FA.k2[1][0]);
     fprintf(fp_fa, "%f\tk2 -y\n", FA.k2[1][1]);
}

/* Apply the gain if outside the deadzone */
if (!DzoneFlag) {
     if (dir < 0)
          dir = FA.Gain[0][0]*dir;
     else
          dir = FA.Gain[0][1]*dir;
```

```
          if (spd < 0)
               spd = FA.Gain[1][0]*spd;
          else
               spd = FA.Gain[1][1]*spd;
     }
}

/* Apply Template */
if (!DzoneFlag) { /* apply only if not in deadzone */
     /* recompute input magnitude */
     L1 = dir*dir + spd*spd;

     switch (Template_Shape_Status) {
          case 0:
               break;

          case 1:       /* elliptical template */
               if(dir != 0) {         /* avoid dividing by 0 */
                    /* find corresponding point on tmpt in QI and QIV */
                    L2 = spd/dir;
                    angle = (float)atan2(Template_Rad_X*L2,Template_Rad_Y);
                    one = (float)(Template_Rad_X*cos(angle));
                    two = (float)(Template_Rad_Y*sin(angle));
                    L2 = (one*one) + (two*two);

                    /* save location; correct quandrant */
                    angle = (float)(atan2(spd, dir));
                    TMPT_x = (float)(sqrt(L2)*cos(angle));
                    TMPT_y = (float)(sqrt(L2)*sin(angle));

                    /* if outside the tmpt, set input to template value */
                    if (L1 > L2) {
                         TempFlag = 1;
                         dir = TMPT_x;
                         spd = TMPT_y;
                         iStatusReturn = 2;
                    }

               }
               else {                 /* dir == 0 */
                    /* determine where signal would intersect template */
                    TMPT_x = 0.0f;
                    TMPT_y =(float)(spd>0? Template_Rad_Y:-Template_Rad_Y);
                    /* if outside the tmpt, set input to template value */
                    if (fabs(spd) > Template_Rad_Y) {
                         TempFlag = 1;
                         dir = 0;
                         spd = TMPT_y;
                         iStatusReturn = 3;
                    }
               }
               break;
          case 2:       /* astroid template */
               /* points to center x and y */
               if (dir != 0) {         /* avoid dividing by 0 */
                    L2 = spd/dir;
                    angle = (float)atan2((powf((float)Template_Rad_X*
```

```
                fabs(L2)/(float)(Template_Rad_Y), 0.3333)), 1.0f);

one=(float)(Template_Rad_X*cos(angle)*cos(angle)*cos(angle));

two=(float)(Template_Rad_Y*sin(angle)*sin(angle)*sin(angle));

          L2 = one*one + two*two;
          angle = (float)atan2(spd, dir);
          TMPT_x = (float)(sqrt(L2)*cos(angle));
          TMPT_y = (float)(sqrt(L2)*sin(angle));

          /* if outside the tmpt, set input to template value */
          if (L1 > L2) {
               TempFlag = 1;
               dir = TMPT_x;
               spd = TMPT_y;
               iStatusReturn = 2;
          }
     }
     else {                  /* dir == 0 */
          TMPT_x = 0.0f;
          TMPT_y =(float)(spd>0? Template_Rad_Y:-Template_Rad_Y);
          if (fabs(spd) > Template_Rad_Y) {
               TempFlag = 1;
               dir = TMPT_x;
               spd = TMPT_y;
               iStatusReturn = 3;
          }
     }
     break;

 case 3:       /* diamond template */
     /* x- and y-radius */
     TP3XRad = Template_Rad_X;
     TP3YRad = Template_Rad_Y;

     L2 = (float)TP3YRad/TP3XRad;
     Td[0] = spd - (L2*dir - TP3YRad);
     Td[1] = (-L2*dir +TP3YRad) - spd;
     Td[2] = (L2*dir + TP3YRad) - spd;
     Td[3] = spd - (-L2*dir -TP3YRad);

     if (dir != 0) {
     L1 = fabs(spd/dir);
          if ((spd >= 0) && (dir >= 0)) {
               TMPT_x = (TP3YRad/(L1+L2));
               TMPT_y = (L1*TP3YRad/(L1+L2));
          }
          else if ((dir <= 0) && (spd >= 0)) {
               TMPT_x = (TP3YRad/(-L1-L2));
               TMPT_y = (L1*TP3YRad/(L1+L2));
          }
          else if ((dir<=0) && (spd<=0)) {
               TMPT_x = (-TP3YRad/(L1+L2));
               TMPT_y = (-TP3YRad*L1/(L1+L2));
          }
          else {
```

```
                        TMPT_x = (TP3YRad/(L1+L2));
                        TMPT_y = (-TP3YRad*L1/(L1+L2));
                    }
                }
                else {
                    TMPT_x = 0.0f;
                    TMPT_y = (float)(spd>=0? TP3YRad:-TP3YRad);
                }
                if (!(Td[0]>=0 && Td[1]>=0 && Td[2]>=0 && Td[3]>=0)) {
                    TempFlag = 1;
                    dir = TMPT_x;
                    spd = TMPT_y;
                    iStatusReturn = 2;
                }
                break;
        }
    }

    /* Remove axes offset and save data, if you want 2048,2048 to be zero,
     * add DIG_zero to each value */
    /* but the output should be centered at (0,0) */
    data[0] = dir + DIG_zero;
    data[1] = spd + DIG_zero;
    data[2] = DZ_x + DIG_zero;
    data[3] = DZ_y + DIG_zero;
    data[4] = TMPT_x + DIG_zero;
    data[5] = TMPT_y + DIG_zero;
    data[6] = DzoneFlag;
    data[7] = TempFlag;

    return(iStatusReturn);
}
```

F.3 CWFLC.H

```
/*****************************************************************************
    modified: 8/1/06
    by: kwb

    cWFLC.h

    NOTE: the maximum allowable order for the IIR high-pass filter is 5

    HISTORY
    kwb [7/28/06]: added high-pass filter params to project
*****************************************************************************/

#include <stdio.h>

struct iir_Nthord {
    int order;        /* filter order */
    double fc;        /* corner frequency (Hz) */
```

```
    double fs;        /* sampling rate (Hz) */
    double alpha;     /* analog frequency parameter */
    double acoef;     /* a coefficient */
    double bcoef;     /* b coefficient */
    double gain;      /* gain */
    double xn_1[5];   /* xn-1 */
    double yn_1[5];   /* yn-1 */
    double y[5];      /* local filter output */
    int type;         /* HPF or LPF */
};

class cWFLC
{
    private:
        int order_count;  /* counting variable for high-pass filter */
        double one, two, three;    /* intermediate vars in filter */
        double x[60];
        double wbias;
        double sumw0, sumcross;
        double input;
        bool wflc_initialized;

        FILE *fp_filts;

    public:

        /* highpass filter params */
        iir_Nthord filt;
        int iir_order;
        double iir_corner;
        double iir_samp_rate;
        double iir_gain;

        double mu, mu0, mub;   /* adapt rates for amp, freq, and bias
                                  weight*/
        unsigned int M;        /* filter order */
        double w0;             /* tremor frequency */
        double w1, wMp1;       /* initial guesses at tremor amplitudes */
        double w[60];
        double offset;
        double e, output;      /* filter error and output, respectively */

        bool initialize(void);
        double highpass_filter(float, int);
        double wflc_filter(double);
        void close(void);
        void Destroy();   //needed?

        cWFLC(void);
        ~cWFLC(void);
};
```

F.4 CWFLC.CPP

```
/*****************************************************************************
    modified: 8/1/06
    by: kwb

    cWFLC.cpp

    PURPOSE
    The purpose of this file is to provide the Weighted Fourier Linear
    Combiner to the ICD algorithm toolkit. The method is based on the
    algorithm given at http://www.cs.cmu.edu/~camr/wflc0.c. The following is
    the description from that site:
/***
    WFLC0.C         Weighted-frequency Fourier Linear Combiner 0.0
    Cameron N. Riviere, Carnegie Mellon University
    last mod.    11/5/98
    This is WF1H, modified to include harmonics.
    Adaptive weighted-frequency algorithm for noise canceling in 1-D data.
    Here, frequency of reference, as well as ampl., is an adaptive weight.
    Mu is the standard adapt. rate param.  mu0 is a 2nd param. on the
    frequency weight.  A bias (highpass) weight is used.
    Ampl. weights for harmonics will be initialized to 0.
    To run this program, type at the DOS prompt:
    WFLC0 filename ext mu mu0 mub M w0 w1 wM+1 offset
    Note:  This code is written in Turbo C for the PC.
/***
    A digital first-order IIR filter was harvested from
    http://www.ddj.com/dept/cpp/184401931. The order was doubled for improved
    cornering.

    HISTORY
*****************************************************************************/

#include <math.h>
#include <stdio.h>
#include <string.h>
#include <windows.h>
#include <direct.h>
// #include <mmsystem.h>

#include "cWFLC.h"

/* constants */
#define PI        3.141592653589
#define HPF       1
#define LPF       -1

/* global/externs */
extern int DEBUG_FILTS;

// e.g....
//extern float data[8];  // used to put limits on joystick input
//extern int dynamics_method;
//extern int acceleration_method;
```

```
/* initialize class variables */
cWFLC::cWFLC(void)
{
     offset = 0;
     mu = 0;
     mu0 = 0;
     mub = 0;
     M = 1;
     w0 = 0;
     w1 = 0;
     wMp1 = 0;
     sumw0 = 0;
     wflc_initialized = false;
}

/* nothing happens here */
cWFLC::~cWFLC(void)
{
}

bool cWFLC::initialize(void)
{
     unsigned int i;

     /* set high-pass params */
     order_count = 0;
     filt.order = iir_order;
     filt.fc = iir_corner;
     filt.fs = iir_samp_rate;
     filt.gain = iir_gain;
     filt.type = HPF;
     filt.alpha = tan(PI*iir_corner/iir_samp_rate);
     filt.acoef = (1 - filt.alpha)/(1 + filt.alpha);
     filt.bcoef = (1 + (filt.type*filt.acoef))/2.0;
     for (i=0; i<5; i++)
     {
          filt.xn_1[i] = 0;
          filt.yn_1[i] = 0;
          filt.y[i] = 0;
     }

     /* set WFLC params */
     sumw0 = 0;
     wbias = 0;
     for (i=1; i<=2*M; i++)
          w[i] = 0;
     w[1] = w1;
     w[M+1] = wMp1;

     wflc_initialized = true;

     if (DEBUG_FILTS)
     {
          fp_filts = fopen("c:\\SubjectFiles\\filter_debug.txt","w");
          fprintf(fp_filts, "order_count\tdatum\ty\n");
     }
```

```
    return(wflc_initialized);
}


/* IIR highpass filter */
double cWFLC::highpass_filter(float datum, int order_count)
{
    one = filt.bcoef*datum;
    two = filt.type*(-filt.bcoef)*filt.xn_1[order_count];
    three = filt.acoef*filt.yn_1[order_count];

    filt.yn_1[order_count] = one + two + three;
    filt.xn_1[order_count] = datum;
    filt.y[order_count] = filt.yn_1[order_count]*filt.gain;

    return(filt.y[order_count]);

    if (DEBUG_FILTS)
    {
        fprintf(fp_filts, "%i\t%f\t%f\n", order_count, datum,
            filt.y[order_count]);
    }
}


/* main WFLC algorithm */
double cWFLC::wflc_filter(double datum)
{
    /* local variables */
    unsigned int i;

    /* ensure variables have been initialized [this should happen once] */
    if (!wflc_initialized)
        initialize();

    /* remove offset */
    input = datum - offset;

    /* locate next sine & cosine samples */
    sumw0 += w0;
    for(i=1; i<=M; i++)
    {
        x[i] = sin(i*sumw0);
        x[M+i] = cos(i*sumw0);
    }

    /* output = truncated Fourier series */
    output = 0;
    for(i=1; i<=2*M; i++)
        output += w[i] * x[i];

    /* calculate error */
    e = input - output - wbias;

    /* update bias weight */
    wbias += 2*mub*e;
```

```
    /* update frequency weight, 'blind' to harmonics */
    sumcross = 0;
    for(i=1; i<M; i++)
        sumcross += i*(w[M+i]*x[i]-w[i]*x[M+i]);
    w0 -= 2*mu0*e*sumcross;

    /* update amplitude weights */
    for(i=1; i<=2*M; i++)
        w[i] += 2*mu*e*x[i];

    /* record data in data file */
    //done so every loop through SaveDataSample() call

    /*output*/
    return (e+offset);
}

void cWFLC::close(void)
{
    if (DEBUG_FILTS)
        fclose(fp_filts);
}
```

APPENDIX G

MILL POSITIONS FOR SELECTED HANDLES

Table 17: Mill table displacements for VCJ compliant mode characterization

angle	Handle 3		Handle 4	
(deg)	dx (in.)	dz (in.)	dx (in.)	dz (in.)
0	0	0	0.0000	0.0000
1	0.1519	-0.0826	0.1569	-0.0866
2	0.3053	-0.1626	0.3152	-0.1704
3	0.46	-0.2399	0.4750	-0.2515
4	0.6161	-0.3144	0.6361	-0.3298
5	0.7734	-0.3863	0.7987	-0.4052
6	0.9319	-0.4553	0.9625	-0.4778
7	1.0917	-0.5216	1.1275	-0.5476
8	1.2525	-0.5851	1.2937	-0.6144
9	1.4145	-0.6458	1.4611	-0.6783
10	1.5775	-0.7036	1.6296	-0.7393
11	1.7414	-0.7586	1.7991	-0.7974
12	1.9064	-0.8107	1.9696	-0.8524
13	2.0721	-0.86	2.1410	-0.9045
14	2.2388	-0.9063	2.3133	-0.9536
15	2.4062	-0.9497	2.4865	-0.9997
16	2.5743	-0.9902	2.6604	-1.0428
17	2.7431	-1.0278	2.8351	-1.0828
18	2.9126	-1.0623	3.0104	-1.1197
19	3.0826	-1.094	3.1863	-1.1536
20	3.2532	-1.1226	3.3628	-1.1844

APPENDIX H

MATLAB MODEL OF STATIC RESPONSE OF VCJ

Both MATLAB routines presented in this Appendix compute reaction forces for a MJ. `VCJv5_compliant_model()` computes forces for the VCJ with handles 3 and 4 and the spring configured to emulate a conventional MJ. `VCJv5_compliant_parametric_model()` computes forces for the VCJ but where the lever arm, spring rate, and spring pretension may be defined based on user input.

H.1 VCJV5_COMPLIANT_MODEL.M

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   modified: 12/07/06
%   by: kwb
%

function [theta_deg P P_b R_x R_y R_b F_f] = ...
    VCJv5_compliant_model(include_friction, plot_results)

%
%   >> PURPOSE <<
%
%   This program solves for the input force P given an initial deflection
%   of the joystick post.
%
%   >> INPUT <<
%
%   plot_results    boolean for if force to deflection curve should be
%                   plotted
%
%   >> OUTPUT <<
```

```
%
%   P               input force (lbs)
%   P_b             force between stick and boot (lbs)
%   R_x             reaction force at pivot, x-direction (lbs)
%   R_y             reaction force at pivot, y-direction (lbs)
%   R_b             reaction force at boot, y-direction (lbs)
%   F_f             force of friction (lbs)
%   theta_deg       angles for which forces are computed (degress)
%
%   >> NOMENCLATURE <<
%
%   see Figure in section 3.1.2.2
%
%   >> HISTORY <<
%   Version 1.0         karl brown          November 22 2006
%   original version. see notes from 11/22/06 in lab notebook.
%   Version 2.0         kwb                 November 27 2006
%   new model. see notes from 11/27 in lab notebook.
%   Version 2.1         kwb                 December 06 2006
%   remove coefficient of friction, update TURNS2INCHES
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

pounds2newtons = 4.44822162;
TURNS2INCHES = 0.05;        %based on 1/4-20 threads

if (nargin < 1)
    include_friction = input('Should friction be included [1=yes;0=no]? ');
end

%%% Define physical parameters (index corresponds to ID number)
% distance between bottom of handle and force application plane
l_handle = [0.85  0.91  1.14  1.54  1.70];
% correction cuz the handles screw on a little more than 1/2" deep
L_jp_factor = 0.035;
% length of stick w/o the handle (bottom of handle threads to middle of
% swivel bearing)
L_jp_alone = 2.266;
L_jp = l_handle + L_jp_alone - L_jp_factor;

L_mg = [0 0 2.43 2.99 0];
mg = [0 0 0.10 0.039+.1234 0];
% mg = [0 0 0.1 0.15 0];

C_hb = 0.85;    % percent-height above base where P_b is centered
h_b = C_hb*0.67;
r_b = 0.9/2;
h_mg = 0.1872;
mg_b = 0.039;
d_p = 0.215;
% d_p = 0.00;

% K = 28.05;        % spring rate (lbs/in) by manufacturer
K = 26.7;           % average rate of three springs from same package
N_turns2spring = 6.2;   % number of turns of lock nut from top of stick
                        % threads to top of spring [confirmed 12/7]
```

```
N_turns = [8.5 8.5 8.5 9.1 9.1];    % number of turns of lock nut from top
                                    % of stick threads to appropriate
                                    % pretension
del_0 = TURNS2INCHES*(N_turns - N_turns2spring);

theta_deg = 0.1:0.1:18;
theta = theta_deg*pi/180;

k = 1;  % counter

%%% Find forces for handles 3 and 4
for i = [3 4]
    for j = theta
        % angle-dependent parameters
        del = r_b*tan(j) + (d_p/cos(j)-d_p);
        Fs(k) = K*(del + del_0(i));
        L_b = del + h_b + d_p;
        w_mg = r_b*cos(j) - h_mg*sin(j);

        % define matrix
        if (include_friction)
%   P       P_b     R_x     R_y     R_b     F_f     Soln
A = [cos(j) -cos(j) -1      0       0       0       -Fs(k)*sin(j)
    -sin(j) sin(j)  0       -1      0       0       Fs(k)*cos(j)+mg(i)
    -L_jp(i)    L_b 0       0       0       0       mg(i)*L_mg(i)*sin(j)
    0       cos(j)  0       0       0       -1      Fs(k)*sin(j)
    0       -sin(j) 0       0       1       0       Fs(k)*cos(j)+mg_b
    0       -h_b    0       0       0       0       -Fs(k)*r_b-mg_b*w_mg];
        else
        %   P       P_b     R_x     R_y     R_b     Soln
        A = [cos(j) -cos(j) -1      0       0       -Fs(k)*sin(j)
            -sin(j) sin(j)  0       -1      0       Fs(k)*cos(j)+mg(i)
            -L_jp(i)    L_b 0       0       0       mg(i)*L_mg(i)*sin(j)
            0       -sin(j) 0       0       1       Fs(k)*cos(j)+mg_b
            0       -h_b    0       0       0       -Fs(k)*r_b-mg_b*w_mg];
        end

        % find solution
        TMP = rref(A);

        % check that solution exists; if not, exit
        R_TMP = rank(TMP);
        FULL_RANK = 5 + include_friction;
        if (R_TMP < FULL_RANK)
            beep();
            fprintf('rank of solution for A is not full\nexiting...\n');
            return;
        end

        % save answers
        [m n] = size(TMP);
        P(k,i) = TMP(1,n);
        P_b(k,i) = TMP(2,n);
        R_x(k,i) = TMP(3,n);
        R_y(k,i) = TMP(4,n);
```

```
        R_b(k,i) = TMP(5,n);
        if (include_friction)
            F_f(k,i) = TMP(6,n);
        else
            F_f(k,i) = 0;
        end

        k = k + 1;
    end
    k = 1;
end

if (nargin < 2)
    plot_results = input('Should input force be plotted [0=no; 1=yes]? ');
end
if (plot_results)
    figure

    % handle 3 force to deflection
    plot(P(:,3)*pounds2newtons, theta_deg, 'k');
    hold on;
    % define experimental force to deflection for handle 3 and plot
    ang = [8 12 16 18 17 14 10 6 4 2 1];
    force = [2.28 3.05 3.93 12.89 3.98 3.47 2.6 1.89 1.56 1.32 1.21];
    plot(force,ang,'ko')

    % handle 4 force to deflection
    plot(P(:,4)*pounds2newtons, theta_deg, 'k', 'LineWidth', 2.0);
    % define experimental force to deflection for handle 4 and plot
    ang = [8 12 16 18 17 14 10 6 4 2 1];
    force = [2.26 2.82 3.65 8.59 3.68 3.11 2.53 2.02 1.74 1.55 1.35];
    plot(force,ang,'k^')

    % set plot properties
    axis([0 7 0 20])
    title(['Experimental vs. theoretrical force to deflection for', ...
        ' Handles 3 and 4']);
    xlabel('force (N)'); ylabel('deflection (deg)');
    legend('handle 3 theoretical', 'handle 3 experimental', ...
        'handle 4 theoretical', 'handle 4 experimental', 'Location', ...
        'Best');
end
```

H.2 VCJV5_COMPLIANT_PARAMETRIC_MODEL.M

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   modified: 12/07/06
%   by: kwb
%

function [theta_deg P P_b R_x R_y R_b F_f] = ...
    VCJv5_compliant_parametric_model(L_jp, K, del_0, include_friction, ...
    plot_results)

%
%   >> PURPOSE <<
%
%   This program solves for the input force P given an initial deflection
%   of the joystick post. It further allows for easy manipulation of the
%   joystick length, spring rate, and spring pretension. It assumes that
%   the center of mass of the joystick handle is 85% high along the lever
%   arm.
%
%   Test points consisted of the following
%       L_jp = [3.0 3.5 4.0];
%       K = [20 30 40];
%       del_0 = [0.03 .06 .09];
%
%   >> INPUT <<
%
%   L_jp            length of joystick shaft (in)
%   K               spring rate (lbs/in)
%   del_0           pretension of spring (in)
%   include_friction    boolean for if the model should include friction
%                       force on sliding surface
%   plot_results    boolean for if force to deflection curve should be
%                   plotted
%
%   >> OUTPUT <<
%
%   P               input force (N)
%   P_b             force between stick and boot (N)
%   R_x             reaction force at pivot, x-direction (N)
%   R_y             reaction force at pivot, y-direction (N)
%   R_b             reaction force at boot, y-direction (N)
%   F_f             force of friction (N)
%   theta_deg       angles for which forces are computed (degress)
%
%   >> NOMENCLATURE <<
%
%   see Figure in section 3.1.2.2
%
%   >> HISTORY <<
%   Version 1.0         karl brown          November 30 2006
%   original version. based on version 2.0 of VCJv5_compliant_model()
%   Version 1.1         kwb                 December 06 2006
%   remove coefficient of friction.
```

```
%   Version 1.2         kwb                 December 06 2006
%   new model to include height of sliding surface plane above pivot center
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

pounds2newtons = 4.44822162;

if (nargin < 4)
    include_friction = input('Should friction be included [1=yes;0=no]? ');
end

%%% Define physical parameters
L_mg = 0.75*L_jp;
mg = 0.125;

C_hb = 0.85;    % percent-height above base where P_b is centered
h_b = C_hb*0.67;
r_b = 0.9/2;
h_mg = 0.1872;
mg_b = 0.039;
d_p = 0.215;
% d_p = 0.00;

theta_deg = 0.1:0.1:18;
theta = theta_deg*pi/180;

k = 1;  % counter

%%% Find forces
for j = theta
    % angle-dependent parameters
    del = r_b*tan(j) + (d_p/cos(j)-d_p);
    Fs(k) = K*(del + del_0);
    L_b = del + h_b + d_p;
    w_mg = r_b*cos(j) - h_mg*sin(j);

    % define matrix
    if (include_friction)
%   P       P_b     R_x     R_y     R_b     F_f     Soln
A = [cos(j) -cos(j) -1      0       0       0       -Fs(k)*sin(j)
    -sin(j) sin(j)  0       -1      0       0       Fs(k)*cos(j)+mg
    -L_jp   L_b     0       0       0       0       mg*L_mg*sin(j)
    0       cos(j)  0       0       0       -1      Fs(k)*sin(j)
    0       -sin(j) 0       0       1       0       Fs(k)*cos(j)+mg_b
    0       -h_b    0       0       0       0       -Fs(k)*r_b-mg_b*w_mg];
    else
        %   P       P_b     R_x     R_y     R_b     Soln
        A = [cos(j) -cos(j) -1      0       0       -Fs(k)*sin(j)
            -sin(j) sin(j)  0       -1      0       Fs(k)*cos(j)+mg
            -L_jp   L_b     0       0       0       mg*L_mg*sin(j)
            0       -sin(j) 0       0       1       Fs(k)*cos(j)+mg_b
            0       -h_b    0       0       0       -Fs(k)*r_b-mg_b*w_mg];
    end

    % find solution
```

```
    TMP = rref(A);

    % check that solution exists; if not, exit
    R_TMP = rank(TMP);
    FULL_RANK = 5 + include_friction;
    if (R_TMP < FULL_RANK)
        beep();
        fprintf('rank of solution for A is not full\nexiting...\n');
        return;
    end

    % save answers
    [m n] = size(TMP);
    P(k) = TMP(1,n)*pounds2newtons;
    P_b(k) = TMP(2,n)*pounds2newtons;
    R_x(k) = TMP(3,n)*pounds2newtons;
    R_y(k) = TMP(4,n)*pounds2newtons;
    R_b(k) = TMP(5,n)*pounds2newtons;
    if (include_friction)
        F_f(k) = TMP(6,n);
    else
        F_f(k) = 0;
    end

    k = k + 1;
end

if (nargin < 5)
    plot_results = input('Should input force be plotted [0=no; 1=yes]? ');
end
if (plot_results)
    figure
    plot(P, theta_deg, 'k');

    % set plot properties
    axis([0 7 0 20])
    title(['Theoretrical force to deflection of joystick']);
    xlabel('force (N)'); ylabel('deflection (deg)');
end
```

APPENDIX I

SOURCE CODE IMPLEMENTATION FOR *MSS TUNING*

I.1 TUNING_MS_STUDY.CPP

```
//---------------------------------------------------------------------------

#include <vcl.h>
#include <stdio.h>
#include "ICD.h"

#pragma hdrstop
USERES("Tuning_MS_Study.res");
USEFORM("Unit1.cpp", Form1);
USEFORM("Unit2.cpp", Form2);
USEFORM("Unit3.cpp", Form3);
USE("ICD.h", File);
USELIB("C:\Program Files\National Instruments\NI-DAQ\Lib\nidex32b.lib");
USELIB("C:\Program Files\National Instruments\NI-DAQ\Lib\nidaq32b.lib");
USELIB("C:\Program Files\Borland\CBuilder5\Lib\PSDK\winmm.lib");
USEUNIT("ICD_functions.cpp");
USEFORM("Unit4.cpp", Form4);
USEFORM("Unit5.cpp", Form5);
//---------------------------------------------------------------------------

//global variables
js_types read_type = DAQcard;   //serial or DAQcard. used in all units
     //note to self: i may want to consider making the 'Reset 0' buttons
    //enabled only if read_type == DAQcard....
int trial_length = 0;  //used for debugging: 0=short; 1=long trials
                       // remember to update unit 5, too
WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)
{
     int iStatus;

     if (read_type == DAQcard) {
          /***** CONFIGURE INPUT SETTINGS for NI-DAQcard *****/
          iStatus = ICD_initialize();
          if (DEBUG_lo) printf("iStatus, ICD_initialize = %d\n", iStatus);
```

```
        /* find offset voltages */
        iStatus = ICD_find_offset();
        if (DEBUG_lo) printf("iStatus, ICD_find_offset = %d\n", iStatus);
    }

    try
    {
        Application->Initialize();
        Application->CreateForm(__classid(TForm1), &Form1);
        Application->CreateForm(__classid(TForm2), &Form2);
        Application->CreateForm(__classid(TForm3), &Form3);
        Application->CreateForm(__classid(TForm4), &Form4);
        Application->CreateForm(__classid(TForm5), &Form5);
        Application->Run();
    }
    catch (Exception &exception)
    {
        Application->ShowException(&exception);
    }
    return 0;
}
//---------------------------------------------------------------------------
```

I.2 UNIT1.H

```
//---------------------------------------------------------------------------

#ifndef Unit1H
#define Unit1H
//---------------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <ComCtrls.hpp>
#include <ExtCtrls.hpp>
#include <Menus.hpp>
//---------------------------------------------------------------------------
class TForm1 : public TForm
{
    __published:  // IDE-managed Components
        TPanel *Panel1;
        TLabel *Label6;
        TLabel *Label7;
        TLabel *Label8;
        TLabel *Label9;
        TLabel *Label1;
        TLabel *Label2;
        TLabel *Label3;
        TButton *Button1;
        TEdit *Edit1;
```

```
        TEdit *Edit2;
        TEdit *Edit3;
        TEdit *Edit4;
        TButton *Button2;
        TRadioButton *RadioButton1;
        TRadioButton *RadioButton2;
        TRadioButton *RadioButton3;
        TTrackBar *TrackBar1;
        TTrackBar *TrackBar2;
        TEdit *Edit5;
        TEdit *Edit6;
        TLabel *Label11;
        TLabel *Label12;
        TTimer *Timer1;
        TMainMenu *MainMenu1;
        TButton *Button3;
        TEdit *Edit10;
        TLabel *Label13;
        void __fastcall Button1Click(TObject *Sender);
        void __fastcall Button2Click(TObject *Sender);
        void __fastcall Timer1Timer(TObject *Sender);
        void __fastcall RadioButton1Click(TObject *Sender);
        void __fastcall RadioButton2Click(TObject *Sender);
        void __fastcall RadioButton3Click(TObject *Sender);
        void __fastcall TrackBar1Change(TObject *Sender);
        void __fastcall TrackBar2Change(TObject *Sender);
        void __fastcall FormPaint(TObject *Sender);
        void __fastcall FormMouseDown(TObject *Sender, TMouseButton Button,
        TShiftState Shift, int X, int Y);
        void __fastcall FormMouseMove(TObject *Sender, TShiftState Shift,
        int X, int Y);
        void __fastcall FormMouseUp(TObject *Sender, TMouseButton Button,
        TShiftState Shift, int X, int Y);
//       void __fastcall TemplateBias1Click(TObject *Sender);
        void __fastcall Button3Click(TObject *Sender);
        void __fastcall Edit10Change(TObject *Sender);

private:      // User declarations
public:       // User declarations
        __fastcall TForm1(TComponent* Owner);
};
//---------------------------------------------------------------------------
extern PACKAGE TForm1 *Form1;
//---------------------------------------------------------------------------
#endif
```

I.3 UNIT1.CPP

```
/*****************************************************************************

    Version: 1.0
    Modified: 11/06/06
    By: kwb

    Unit1.cpp

    >> PURPOSE <<
    This code collects dead zone information for the subject and allows the
    clinician to set its shape and size. the clinician also enters the
    subject ID at the bottom of the screen.

    >> NOMENCLATURE <<
    do i want to include nomenclature???

    >> HISTORY <<
    Version 1.0       kwb      19 june 2006
    original version. based on tuning software for TBI study, last modified
    5/8/06.
    added capability to read from NI-DAQcard.
    remove fastcalls for adjusting center_x and center_y since my driver s/w
    does this automatically.
*****************************************************************************/

//---------------------------------------------------------------------------

#include <vcl.h>
#pragma hdrstop

#include <dos.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <mmsystem.h>
#include <windows.h>
#include <string.h>

#include "Unit1.h"
#include "Unit2.h"
#include "Unit3.h"
#include "ICD.h"

#define PI 3.1415926
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;

//global variables
extern int trial_length;   //used for debugging: 0=short; 1=long trials
                           // remember to update unit 5, too
extern js_types read_type;//serial or DAQcard. defined in Tuning_MS_Study.cpp
```

```
int MaxDZForce, MaxDZDirection, shape;
AnsiString subject_id;

struct time bt, et;
int ab1, ab2, direction, speed, count;
int data[30000];

bool mdown;
int xaxis=15, yaxis=10, mousex=300, mousey=300;
float angle=0;
int scale=4, cradius;
TPoint sr;

//function prototypes?
int boundary(void);
void Draw_Circle(void);
void canvas_draw(void);
void Draw_Rectangle(void);
void boundary_ellipse(void);
void Draw_Ellipse(void);

//possibly old serial varaiables
HANDLE hComm=NULL;
COMMTIMEOUTS ctmoNew={0},ctmoOld;
DWORD dwEvent,dwError;
COMSTAT cs;

//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
     : TForm(Owner)
{
     timeBeginPeriod(1);
     //printf("foo\n");
     //Form1->Button3->Enabled=false;
}

// this finds the deadzone boundary for the circle shape
int boundary(void)
{
     int i, xplot, yplot, radius=0, temp;
     for (i=0; i<count*2; i+=2) {
          xplot=(250*(data[i]-2048)/2048);
          yplot=(250*(data[i+1]-2048)/2048);
          temp=xplot*xplot+yplot*yplot;
          if (temp>radius) radius=temp;
     }

     return (int) sqrt(radius)+1;
}

void Draw_Circle(void)
{
     Form1->Refresh();

     Form1->Canvas->Brush->Color = clWhite;
     Form1->Canvas->Pen->Color = clRed;
```

```
    Form1->Canvas->Ellipse(mousex-scale*cradius,mousey-scale*cradius,
mousex+scale*cradius, mousey+scale*cradius);

}

// this causes the program to collect deadzone data from the joystick
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    // Local Variables
    int iStatus;

    // Device Readings
    float tmp_ICD_x_reading = 0.0;
    float tmp_ICD_y_reading = 0.0;
    unsigned char InBuff[10];
    DWORD dwBytesRead;

   // Initialization
    Form1->Refresh();
    Form1->Timer1->Enabled=true;
    Form1->Button1->Enabled=false;
    Form1->RadioButton1->Checked=false;
    Form1->RadioButton1->Enabled=true;
    Form1->RadioButton2->Checked=false;
    Form1->RadioButton2->Enabled=true;
    count=0;

    /* Serial Communication */
    if (read_type == serial) {
        DCB dcbCommPort;

hComm=CreateFile("COM1",GENERIC_READ|GENERIC_WRITE,0,NULL,OPEN_EXISTING,0,NUL
L);
        GetCommState(hComm,&dcbCommPort);
        BuildCommDCB("38400,N,8,1", &dcbCommPort);
        SetCommState(hComm,&dcbCommPort);
        SetCommMask(hComm,EV_RXCHAR);
        GetCommTimeouts(hComm,&ctmoNew);
        ctmoNew.ReadIntervalTimeout=2.7;
        ctmoNew.ReadTotalTimeoutConstant=0;
        ctmoNew.ReadTotalTimeoutMultiplier=0;
        SetCommTimeouts(hComm,&ctmoNew);
    }

    //Get start time
    gettime(&bt);

    while(1)
    {
        if (read_type == serial)
            ClearCommError(hComm,&dwError,&cs);

        Application->ProcessMessages();

        // based on type of data port, get data from joystick
        switch (read_type) {
            case DAQcard:
```

```
            iStatus = ICD_get_data(&tmp_ICD_x_reading,&tmp_ICD_y_reading);
               if (DEBUG_lo)
                    printf("iStatus, ICD_get_data = %d\n", iStatus);
               direction = data[count*2] = tmp_ICD_x_reading + 2048;
               speed = data[count*2+1] = tmp_ICD_y_reading + 2048;
               ++count;
               break;
          case serial:
               while(1) //keep trying to get data until you do
               {
                    if (WaitCommEvent(hComm,&dwEvent,NULL))
                    {
                         if (dwEvent & EV_RXCHAR)
                         {
                              ClearCommError(hComm,&dwError,&cs);
                              ReadFile(hComm,InBuff,4,&dwBytesRead,NULL);
                              ClearCommError(hComm,&dwError,&cs);

                              ab1=(unsigned int)(unsigned char)(InBuff[0]);
                              if ((ab1&240)!=16) break;

                              ab2=(unsigned int)(unsigned char)(InBuff[2]);
                              if ((ab2&240)!=32) break;

                              direction=data[count*2]=(ab1&15)*256 +
                                   (unsigned int)(unsigned char)(InBuff[1]);
                              speed=data[count*2+1]=(ab2&15)*256 +
                                   (unsigned int)(unsigned char)(InBuff[3]);

                              ++count;
                              break;
                         }
                    }
               break;
          }   //end switch
     }

     gettime(&et);
     if (trial_length == 1)
     {
          if (abs(et.ti_sec-bt.ti_sec)==30) break;
     }
     else
     {
          if (abs(et.ti_sec-bt.ti_sec)==10 ||
               abs(et.ti_sec-bt.ti_sec)==50) break;
     }
}

Form1->Timer1->Enabled=false;
Form1->Button1->Enabled=true;

if (read_type == serial) {
     SetCommMask(hComm,0);
     PurgeComm(hComm,PURGE_RXABORT);
     SetCommTimeouts(hComm,&ctmoOld);
     CloseHandle(hComm);
```

```
    }

}

// this draws the data points on the screen
void  canvas_draw(void)
{
    int xplot, yplot, i;
    for (i=0; i<count*2; i+=2) {
        xplot=300+(250*scale*(data[i]-2048)/2048);
        yplot=300-(250*scale*(data[i+1]-2048)/2048);

        Form1->Canvas->Brush->Color = clGray;
        Form1->Canvas->Pen->Color = clGray;
        Form1->Canvas->Ellipse(xplot-1,yplot-1,xplot+1, yplot+1);
    }
}

// this finds the deadzone boundary for the square shape
TPoint boundary_square(void)
{
    int i, xplot, yplot, radiusx=0, radiusy=0;
    TPoint sp;
    for (i=0; i<count*2; i+=2) {
        xplot=abs(250*(data[i]-2048)/2048);
        if (xplot>radiusx){
            radiusx=xplot;
            sp.x=xplot;
        }
        yplot=abs(250*(data[i+1]-2048)/2048);
        if (yplot>radiusy) {
            radiusy=yplot;
            sp.y=yplot;
        }
    }
    return sp;
}

void Draw_Rectangle(void)
{
    Form1->Refresh();
    Form1->Canvas->Brush->Color = clWhite;
    Form1->Canvas->Pen->Color = clRed;
    Form1->Canvas->Rectangle(mousex-scale*(sr.x+1),mousey-scale*(sr.y+1),
mousex+scale*(sr.x+1), mousey+scale*(sr.y+1));
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button2Click(TObject *Sender)
{
    canvas_draw();
}
//---------------------------------------------------------------------------


void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
```

```
    char strBuffer[9];

    sprintf(strBuffer, "%2d:%02d:%02d.%02d", bt.ti_hour, bt.ti_min,
        bt.ti_sec, bt.ti_hund);
    Form1->Edit3->Text=strBuffer;
    sprintf(strBuffer, "%2d:%02d:%02d.%02d", et.ti_hour, et.ti_min,
        et.ti_sec, et.ti_hund);
    Form1->Edit4->Text=strBuffer;

    Form1->Edit1->Text=IntToStr(direction);
    Form1->Edit2->Text=IntToStr(speed);
}
//---------------------------------------------------------------------------

void __fastcall TForm1::RadioButton1Click(TObject *Sender)
{
    mousex=300;mousey=300;
    Form1->Refresh();
    cradius=boundary();
    Draw_Circle();
    canvas_draw();
    shape=1;
    Form1->TrackBar1->Enabled=true;
    Form1->TrackBar2->Enabled=false;
    Form1->TrackBar1->Position=cradius;
    Form1->TrackBar2->Position=cradius;
    Form1->Edit5->Text="    "+IntToStr(TrackBar1->Position*2048/250);
    // Edit5->Text="    "+IntToStr(TrackBar1->Position);
    Form1->Edit6->Text="    "+IntToStr(TrackBar2->Position*2048/250);
    Form1->Button3->Enabled=true;
}
//---------------------------------------------------------------------------

void __fastcall TForm1::RadioButton2Click(TObject *Sender)
{
    mousex=300;mousey=300;
    Form1->Refresh();
    sr=boundary_square();
    Draw_Rectangle();
    canvas_draw();
    shape=2;
    Form1->TrackBar1->Enabled=true;
    Form1->TrackBar2->Enabled=true;

    Form1->TrackBar1->Position=sr.x;
    Form1->TrackBar2->Position=sr.y;
    Form1->Edit5->Text="    "+IntToStr(TrackBar1->Position*2048/250);
    Form1->Edit6->Text="    "+IntToStr(TrackBar2->Position*2048/250);
    Form1->Button3->Enabled=true;
}
//---------------------------------------------------------------------------

void boundary_ellipse(void)
{
    int i, xplot, yplot, temp, tempxplot=0, tempyplot=0, a, b;
    double anglepoint,rx,ry,rprime,r;
    for (i=0; i<count*2; i+=2) {
```

```
          xplot=(250*(data[i]-2048)/2048);
          yplot=(250*(data[i+1]-2048)/2048);

          //erika's code
          if (abs(xplot)>tempxplot)
               tempxplot=abs(xplot);
          if (abs(yplot)>tempyplot)
               tempyplot=abs(yplot);
     }
     for (i=0; i<count*2; i+=2) {
          xplot=(250*(data[i]-2048)/2048);
          yplot=(250*(data[i+1]-2048)/2048);

          if (xplot!=0)
               anglepoint=atan2(yplot,xplot);
          else
               if (yplot>=0)
                    anglepoint=0;
               else
                    anglepoint=3*PI/2;
          rprime=xplot*xplot+yplot*yplot;

          r=(tempxplot*cos(anglepoint))*(tempxplot*cos(anglepoint))+(tempyplot
          *sin(anglepoint))*(tempyplot*sin(anglepoint));

          if (rprime>r)
          {
               tempxplot=tempxplot+(sqrt(rprime)-sqrt(r))+1;
               tempyplot=tempyplot+(sqrt(rprime)-sqrt(r))+1;
          }
     }
     xaxis=tempxplot;
     yaxis=tempyplot;
}

//---------------------------------------------------------------------------
void Draw_Ellipse(void)
{
     int xc, yc, orignx, origny;
     float theta, xn, yn;

     Form1->Refresh();

     Form1->Canvas->Brush->Color = clWhite;
     Form1->Canvas->Pen->Color = clRed;

     for (theta=0;theta<=2*PI;theta+=0.01) {
          xn=scale*xaxis*cos(theta);
          yn=scale*yaxis*sin(theta);
          xc=xn*cos(angle)-yn*sin(angle)+mousex;
          yc=xn*sin(angle)+yn*cos(angle)+mousey;
          if (theta==0.0) {
               Form1->Canvas->MoveTo(xc,yc);
               orignx=xc;
               origny=yc;
          }
          else Form1->Canvas->LineTo(xc, yc);
```

```
     }

     Form1->Canvas->LineTo(orignx, origny);
}

void __fastcall TForm1::RadioButton3Click(TObject *Sender)
{
     mousex=300;mousey=300;
     Form1->Refresh();
     Form1->Canvas->Brush->Color = clWhite;
     Form1->Canvas->Pen->Color = clRed;
     boundary_ellipse();
     Form1->Canvas->Ellipse(300-scale*xaxis,300-scale*yaxis, 300+scale*xaxis,
300+scale*yaxis);

     canvas_draw();
     shape=3;
     Form1->TrackBar1->Enabled=true;
     Form1->TrackBar2->Enabled=true;

     Form1->TrackBar1->Position=xaxis;
     Form1->TrackBar2->Position=yaxis;

     Form1->Edit5->Text="    "+IntToStr(TrackBar1->Position*2048/250);
     Form1->Edit6->Text="    "+IntToStr(TrackBar2->Position*2048/250);
     Form1->Button3->Enabled=true;
}

void __fastcall TForm1::TrackBar1Change(TObject *Sender)
{
     xaxis=TrackBar1->Position;
     cradius=TrackBar1->Position;
     sr.x=TrackBar1->Position;
     Form1->Edit5->Text="    "+IntToStr(TrackBar1->Position*2048/250);
     // Edit5->Text="    "+IntToStr(TrackBar1->Position);
     switch(shape) {
          case 1:  {
               Draw_Circle();
               Form1->Edit6->Text="    "+IntToStr(TrackBar1->
Position*2048/250);
               Form1->TrackBar2->Position=TrackBar1->Position;
               break;
          }
          case 2:  {
               Draw_Rectangle();
               break;
          }
          case 3:  {
               Draw_Ellipse();
               break;
          }
     }
     canvas_draw();
}
//---------------------------------------------------------------------------

void __fastcall TForm1::TrackBar2Change(TObject *Sender)
```

```
{
     yaxis=TrackBar2->Position;
     sr.y=TrackBar2->Position;
     Form1->Edit6->Text="      "+IntToStr(TrackBar2->Position*2048/250);
     switch(shape) {

          case 2:  {
               Draw_Rectangle();
               break;
          }
          case 3:  {
               Draw_Ellipse();
               break;
          }

     }
     canvas_draw();
}
//---------------------------------------------------------------------------

void __fastcall TForm1::FormPaint(TObject *Sender)
{
     Canvas->Brush->Color = clWhite;
     Canvas->Pen->Color = clSilver;
     Canvas->Pen->Width = 3;
     Canvas->Rectangle(50, 50, 550, 550);

     Canvas->Brush->Color = clBlack;
     Canvas->Pen->Color = clBlack;
     Canvas->Ellipse(300-5,300-5,300+5, 300+5);
}
//---------------------------------------------------------------------------

void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button,
      TShiftState Shift, int X, int Y)
{
     mdown=true;
}
//---------------------------------------------------------------------------


// This function may be obsolete for the MS Study...
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift,
      int X, int Y)
{
     if(mdown) {
          Form1->Refresh();

          switch(shape) {
               case 1:  {
                    Draw_Circle();
                    break;
               }
               case 2:  {
                    Draw_Rectangle();
                    break;
               }
```

```
            case 3:  {
                 Draw_Ellipse();
                 break;
            }
        }

        canvas_draw();
        mousex=X;
        mousey=Y;
    }

}
//---------------------------------------------------------------------------

void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button,
      TShiftState Shift, int X, int Y)
{
    mdown=false;
}
//---------------------------------------------------------------------------

/*
// This function may be obsolete for the MS Study...
void __fastcall TForm1::TemplateBias1Click(TObject *Sender)
{
        Form1->Visible=false;
        Form2->Visible=true;
//        Form3->Visible=false;
}
*/

//---------------------------------------------------------------------------

void __fastcall TForm1::Button3Click(TObject *Sender)
{
    MaxDZDirection=StrToInt(Edit5->Text);
    MaxDZForce=StrToInt(Edit6->Text);
    Form1->Visible=false;
//   Form2->Visible=true;
    Form3->Visible=true;
}
//---------------------------------------------------------------------------


// this function handles the text box for the subject ID
void __fastcall TForm1::Edit10Change(TObject *Sender)
{
    subject_id = Form1->Edit10->Text;
}
//---------------------------------------------------------------------------
```

I.4 UNIT2.H

```
//---------------------------------------------------------------------------
#ifndef Unit2H
#define Unit2H
//---------------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <ComCtrls.hpp>
#include <ExtCtrls.hpp>
#include "CGAUGES.h"
//---------------------------------------------------------------------------
class TForm2 : public TForm
{
    __published:  // IDE-managed Components
        TPanel *Panel2;
        TLabel *Label6;
        TLabel *Label7;
        TEdit *Edit2;
        TEdit *Edit3;
        TPanel *Panel3;
        TLabel *Label3;
        TLabel *Label4;
        TLabel *Label5;
        TLabel *Label10;
        TButton *Button4;
        TComboBox *ComboBox1;
        TEdit *Edit1;
        TEdit *Edit6;
        TTrackBar *TrackBar2;
        TTrackBar *TrackBar1;
        TTrackBar *TrackBar3;
        TEdit *Edit7;
        TButton *Button3;
        TButton *Button5;
        TButton *Button6;
        TButton *Button7;
        TTimer *Timer1;
        TEdit *Edit8;
        TLabel *Label1;
        TEdit *Edit9;
        TLabel *Label11;
        TLabel *Label12;
        TEdit *Edit11;
        TEdit *Edit10;
        TLabel *Label14;
        TEdit *Edit12;
        TEdit *Edit13;
        TLabel *Label16;
        TEdit *Edit14;
        TEdit *Edit15;
        TLabel *Label18;
        TEdit *Edit16;
```

```
        TEdit *Edit17;
        TEdit *Edit18;
        TLabel *Label2;
        TLabel *Label19;
        TLabel *Label20;
        TEdit *Edit19;
        TEdit *Edit20;
        TLabel *Label21;
        TCheckBox *CheckBox1;
        TCheckBox *CheckBox2;
        TCheckBox *CheckBox3;
        TCheckBox *CheckBox4;
        TButton *Button1;
        TEdit *Edit21;
        TLabel *Label24;
        TLabel *Label25;
        TEdit *Edit22;
        TEdit *Edit23;
        TLabel *Label13;
        TLabel *Label15;
        TButton *Button8;
        TLabel *Label26;
        TLabel *Label8;
        TLabel *Label9;
        TLabel *Label17;
        TTimer *Timer2;
        TButton *Button2;
        TButton *Button9;
        TEdit *Edit4;
        TEdit *Edit5;
        TLabel *Label22;
        TLabel *Label23;
        void __fastcall Button3Click(TObject *Sender);
        void __fastcall Button5Click(TObject *Sender);
        void __fastcall Timer1Timer(TObject *Sender);
        void __fastcall Button6Click(TObject *Sender);
        void __fastcall Button7Click(TObject *Sender);
        void __fastcall FormPaint(TObject *Sender);
        void __fastcall FormActivate(TObject *Sender);
        void __fastcall ComboBox1Change(TObject *Sender);
        void __fastcall TrackBar1Change(TObject *Sender);
        void __fastcall TrackBar3Change(TObject *Sender);
        void __fastcall Button4Click(TObject *Sender);
        void __fastcall Button1Click(TObject *Sender);
        void __fastcall Button8Click(TObject *Sender);
        void __fastcall Timer2Timer(TObject *Sender);
        void __fastcall Button2Click(TObject *Sender);
        void __fastcall Button9Click(TObject *Sender);
    private: // User declarations

    public:      // User declarations
        __fastcall TForm2(TComponent* Owner);
};
//---------------------------------------------------------------------------
extern PACKAGE TForm2 *Form2;
//---------------------------------------------------------------------------
#endif
```

I.5 UNIT2.CPP

```
/*************************************************************************

    Version: 1.0
    Modified: 10/10/06
    By: kwb

    Unit2.cpp

    >> PURPOSE <<
    This code determines bias axes, default gain, template, and strength
    characaterization data for the subject.

    >> NOMENCLATURE <<
    do i want to include nomenclature???

    >> HISTORY <<
    Version 1.0       kwb      22 june 2006
    original version. based on tuning software for TBI study, last modified
    5/8/06.
    add capability to read from NI-DAQcard.
    though the layout has been adjusted to fit in one screen, much of the
    functionality remains the same.
    add two minute strength data collection
    gain is such that comfort force produces 90% max force needed [not 80%]
    make (+) and (-) axis gains independent

*************************************************************************
**/

//---------------------------------------------------------------------------

#include <vcl.h>
#pragma hdrstop

#include <dos.h>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <windows.h>
#include <mmsystem.h>

#include "Unit1.h"
#include "Unit2.h"
#include "Unit3.h"
#include "Unit4.h"
#include "ICD.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma link "CGAUGES"
#pragma resource "*.dfm"

#define PI 3.1415926
```

```
TForm2 *Form2;

//global variables
extern int trial_length; //0=short, 1=long; set in Tuning_MS_Study.cpp
extern js_types read_type;//serial or DAQcard. defined in Tuning_MS_Study.cpp
int trial_direction;   //0=forward,1=backward,2=left,3=right
//parameters to output to unit 3 for writing
extern int MaxDZForce, MaxDZDirection, shape;
extern AnsiString subject_id;
int index;  //template shape
float Biasangle;
float XGain[2], YGain[2];

//unit variables
bool strength_trial;
DWORD trial_time[2] = {6*1000, 30*1000}; // sec -> msec
DWORD Strial_time[2] = {10*1000, 120*1000}; // sec -> msec
DWORD sample_period = 10; //msec
int Fdata[50000], Bdata[50000], Ldata[50000], Rdata[50000], Sdata[100000];
DWORD Fdata_time[20000], Bdata_time[20000], Ldata_time[20000],
     Rdata_time[20000], Sdata_time[50000];
int Fsamplenum, Bsamplenum, Lsamplenum, Rsamplenum, Ssamplenum;
int Fmaxmeanforce=0,Bmaxmeanforce=0,Lmaxmeanforce=0,Rmaxmeanforce=0;
int Fcomfortforce=0, Bcomfortforce=0, Lcomfortforce=0,Rcomfortforce=0;
float LBiasangle;
int maxforcewindow = 50;
TPoint square[10];
TPoint interpoint;
int xvalue=250, yvalue=250;
//struct time bt, et;
int direction, speed;
float angle;
DWORD start_time, end_time;
float gauge_progress;

typedef void (*Selected0Function)();

//function prototypes
bool outcircle(int, int);
void Select_template(void);
void Draw_Diamond(TPoint *sp, TPoint *ep);
void Draw_Astroid(int x, int y);
void Circle(void);
void Ellipse(void);
void Astroid(void);
void Diamond(void);
void canvas_draw(void);

//serial variables
HANDLE hComm=NULL;
COMMTIMEOUTS ctmoNew={0},ctmoOld;
DWORD dwBytesRead;
DWORD dwEvent,dwError;
COMSTAT cs;

//---------------------------------------------------------------------------
__fastcall TForm2::TForm2(TComponent* Owner)
```

```
    : TForm(Owner)
{
//    Form2->Panel1->Enabled=false;
    Form2->Button8->Enabled=false;
    Form2->Button4->Enabled=false;
}

void Select_template(void)
{
    //   Form2->Refresh();
}

void Draw_Diamond(int x, int y)
{
    TPoint dia[5];

    dia[0].x=300+xvalue;
    dia[0].y=300;

    dia[1].x=300;
    dia[1].y=300-yvalue;

    dia[2].x=300-xvalue;
    dia[2].y=300;

    dia[3].x=300;
    dia[3].y=300+yvalue;

    Form2->Canvas->Brush->Color = clWhite;
    Form2->Canvas->Pen->Color = clSilver;
    Form2->Canvas->Polygon(dia,3);
}

void Diamond(void)
{
    Form2->Refresh();

    Draw_Diamond(250,250);
    Form2->Canvas->Brush->Color = clBlack;
    Form2->Canvas->Pen->Color = clBlack;
    Form2->Canvas->Ellipse(300-5,300-5,300+5, 300+5);

    Form2->TrackBar1->Position=xvalue=250;
    Form2->TrackBar2->Position=yvalue=250;
}
void Circle(void)
{
    // Form2->Refresh();

    Form2->Canvas->Brush->Color = clWhite;
    Form2->Canvas->Pen->Color = clSilver;
    Form2->Canvas->Ellipse(50,50,550,550);

    Form2->Canvas->Brush->Color = clBlack;
    Form2->Canvas->Pen->Color = clBlack;
    Form2->Canvas->Ellipse(300-5,300-5,300+5, 300+5);
```

```
     Form2->TrackBar1->Position=xvalue=250;
     Form2->TrackBar2->Position=yvalue=250;
     canvas_draw();
}

void Ellipse(void)
{
     Form2->Refresh();

     Form2->Canvas->Brush->Color = clWhite;
     Form2->Canvas->Pen->Color = clSilver;
     Form2->Canvas->Ellipse(50,175,550,425);

     Form2->Canvas->Brush->Color = clBlack;
     Form2->Canvas->Pen->Color = clBlack;
     Form2->Canvas->Ellipse(300-5,300-5,300+5, 300+5);

     Form2->TrackBar1->Position=xvalue=250;
     Form2->TrackBar2->Position=yvalue=250;
}

void Draw_Astroid(int x, int y)
{
     float i, a, b;

     Form2->Canvas->Brush->Color = clWhite;
     Form2->Canvas->Pen->Color = clSilver;
     Form2->Canvas->MoveTo(x+300, 300);

     for (i=PI/180; i<2*PI; i+=PI/180) {
          a=300+x*cos(i)*cos(i)*cos(i);
          b=300-y*sin(i)*sin(i)*sin(i);
          Form2->Canvas->LineTo(a,b);
     }
}

void Astroid(void)
{
     Form2->Refresh();

     Draw_Astroid(250, 250);

     Form2->Canvas->Brush->Color = clBlack;
     Form2->Canvas->Pen->Color = clBlack;
     Form2->Canvas->Ellipse(300-5,300-5,300+5, 300+5);

     Form2->TrackBar1->Position=xvalue=250;
     Form2->TrackBar2->Position=yvalue=250;
}

Selected0Function selectedresource[] =
{
     Select_template,
     Circle,
     Astroid,
     Diamond,
     Ellipse
```

```
};

// forward button click
void __fastcall TForm2::Button3Click(TObject *Sender)
{
     int maxforce=0, aforce[20000];
     int maxforceindex=0, j=0, i,indexh, indexl;
     int ab1, ab2;
     DWORD current_time;

     // Device Readings
     int iStatus;
     float tmp_ICD_x_reading = 0.0;
     float tmp_ICD_y_reading = 0.0;
     unsigned char InBuff[10];
     DWORD dwBytesRead;

     // initializations
     gauge_progress = 0;
     Timer1->Enabled=true;
     Timer2->Enabled=true;
     Button3->Enabled=False;
     Fsamplenum=1;
     trial_direction = 0;
     Form4->Visible=true;
     strength_trial = false;

     if (read_type == serial) {
          // Serial Port Communication
          DCB dcbCommPort;

     hComm=CreateFile("COM1",GENERIC_READ|GENERIC_WRITE,0,NULL,OPEN_EXISTING,0
,NULL);
          GetCommState(hComm,&dcbCommPort);
          BuildCommDCB("38400,N,8,1", &dcbCommPort);
          SetCommState(hComm,&dcbCommPort);
          SetCommMask(hComm,EV_RXCHAR);
          GetCommTimeouts(hComm,&ctmoNew);
          ctmoNew.ReadIntervalTimeout=2.7;
          ctmoNew.ReadTotalTimeoutConstant=0;
          ctmoNew.ReadTotalTimeoutMultiplier=0;
          SetCommTimeouts(hComm,&ctmoNew);
     }

     // Get start time and initial readings
     start_time = timeGetTime();
     switch (read_type) {
          case DAQcard:
               iStatus = ICD_get_data(&tmp_ICD_x_reading, &tmp_ICD_y_reading);
               if (DEBUG_lo) printf("iStatus, ICD_get_data = %d\n", iStatus);
               direction = Fdata[0] = tmp_ICD_x_reading + 2048;
               speed = Fdata[1] = tmp_ICD_y_reading + 2048;
               Fdata_time[0] = timeGetTime() - start_time;
               break;
          case serial:
               ClearCommError(hComm,&dwError,&cs);
               while(1)
```

```
          {
               if (WaitCommEvent(hComm,&dwEvent,NULL))
                    if (dwEvent & EV_RXCHAR)
                    {
                         ClearCommError(hComm,&dwError,&cs);
                         ReadFile(hComm,InBuff,4,&dwBytesRead,NULL);
                         ClearCommError(hComm,&dwError,&cs);

                         ab1=(unsigned int)(unsigned char)(InBuff[0]);
                         if ((ab1&240)!=16) break;
                         ab2=(unsigned int)(unsigned char)(InBuff[2]);
                         if ((ab2&240)!=32) break;

                         direction=Fdata[0]=(ab1&15)*256 +
                              (unsigned int)(unsigned char)(InBuff[1]);
                         speed=Fdata[1]=(ab2&15)*256 +
                              (unsigned int)(unsigned char)(InBuff[3]);
                         Fdata_time[0] = timeGetTime() - start_time;
                         break;
                    }
          }
          break;
} // end switch

while(1)
{
     if (read_type == serial)
          ClearCommError(hComm,&dwError,&cs);

     Application->ProcessMessages();

     switch (read_type) {
          case DAQcard:
               current_time = timeGetTime() - start_time;
               if ((current_time - Fdata_time[Fsamplenum-1]) >=
                    sample_period) {
                    iStatus = ICD_get_data(&tmp_ICD_x_reading,
                         &tmp_ICD_y_reading);
                    if (DEBUG_lo)
                         printf("iStatus, ICD_get_data = %d\n", iStatus);
                    direction = Fdata[Fsamplenum*2]=tmp_ICD_x_reading+2048;
                    speed = Fdata[Fsamplenum*2+1] = tmp_ICD_y_reading+2048;
                    Fdata_time[Fsamplenum] = timeGetTime() - start_time;
                    ++Fsamplenum;
               }
               break;
          case serial:
               while(1)
               {
                    if (WaitCommEvent(hComm,&dwEvent,NULL))
                         if (dwEvent & EV_RXCHAR)
                         {
                              ClearCommError(hComm,&dwError,&cs);
                              ReadFile(hComm,InBuff,4,&dwBytesRead,NULL);
                              ClearCommError(hComm,&dwError,&cs);

                              ab1=(unsigned int)(unsigned char)(InBuff[0]);
```

```
                              if ((ab1&240)!=16) break;
                              ab2=(unsigned int)(unsigned char)(InBuff[2]);
                              if ((ab2&240)!=32) break;

                              direction=Fdata[Fsamplenum*2]=(ab1&15)*256 +
                                   (unsigned int)(unsigned char)(InBuff[1]);
                              speed=Fdata[Fsamplenum*2+1]=(ab2&15)*256 +
                                   (unsigned int)(unsigned char)(InBuff[3]);

                            Fdata_time[Fsamplenum]=timeGetTime()-start_time;

                              ++Fsamplenum;
                              break;
                         }
                    }
                    break;
     } // end switch

     end_time = timeGetTime();
     if ((end_time - start_time) >= trial_time[trial_length]) break;
}

//close everything out
Timer1->Enabled=false;
Timer2->Enabled=false;
Form4->Close();

if (read_type == serial) {
     SetCommMask(hComm,0);
     PurgeComm(hComm,PURGE_RXABORT);
     SetCommTimeouts(hComm,&ctmoOld);
     CloseHandle(hComm);
}

Button3->Enabled=True;

for (i=0; i<Fsamplenum*2; i+=2) {
     aforce[j]=(int) sqrt((Fdata[i]-2048)*(Fdata[i]-2048)+(Fdata[i+1]-
          2048)*(Fdata[i+1]-2048));
     j++;
}

for (i=0; i<Fsamplenum; i++)
     if (maxforce<aforce[i])  {
          maxforce=aforce[i];
          maxforceindex=i;
     }


//max force is defined as the average force of the max force +/-50
//samples
if (maxforceindex-maxforcewindow<0) indexl=0;
else indexl=maxforceindex-maxforcewindow;
if (maxforceindex+maxforcewindow>Fsamplenum)indexh=Fsamplenum-1;
else indexh=maxforceindex+maxforcewindow;

for (j=indexl;j<=indexh;j++)
```

```
        Fmaxmeanforce+=aforce[j];
    Fmaxmeanforce= Fmaxmeanforce /(indexh-indexl+1);

    //comfort force is defined as the average force for the second half
    //of the test period
    for (i=Fsamplenum/2;i<Fsamplenum;i++)
        Fcomfortforce+=aforce[i];
    Fcomfortforce = Fcomfortforce / (Fsamplenum/2);

    Edit9->Text=IntToStr(Fmaxmeanforce);
    Edit8->Text=IntToStr(Fcomfortforce);

    int xplot, yplot, xsum=0, ysum=0, nsample=Fsamplenum;

    for (i=0; i<Fsamplenum*2; i+=2) {
        xplot=300+(250*(Fdata[i]-2048)/2048);
        yplot=300-(250*(Fdata[i+1]-2048)/2048);
        if (abs(xplot-300)<=20 & abs(yplot-300)<=20)
            nsample-=1;
        else {
            xsum+=xplot;
            ysum+=yplot;
        }
        Canvas->Brush->Color = clGray;
        Canvas->Pen->Color = clGray;
        Canvas->Ellipse(xplot-4,yplot-4,xplot+4, yplot+4);
    }

    Biasangle=PI-(PI/2-atan2(ysum/nsample-300,xsum/nsample-300));
    Form2->Canvas->Brush->Color = clRed;
    Form2->Canvas->Pen->Color = clRed;
    Form2->Canvas->MoveTo(300, 300);
    Form2->Canvas->LineTo((int)(xsum/nsample), (int) (ysum/nsample));

    if (Biasangle<0) Biasangle=2*PI+Biasangle;
    Form2->Edit16->Text=FloatToStr(Biasangle*180/PI);

}
//---------------------------------------------------------------------------

// backward button click
void __fastcall TForm2::Button5Click(TObject *Sender)
{
    int maxforce=0, aforce[20000];
    int maxforceindex=0, j=0, i, indexh, indexl;
    int ab1, ab2;
    DWORD current_time;

    // Device Readings
    int iStatus;
    float tmp_ICD_x_reading = 0.0;
    float tmp_ICD_y_reading = 0.0;
    unsigned char InBuff[10];
    DWORD dwBytesRead;

    // initializations
    gauge_progress = 100;
```

```
Timer1->Enabled=true;
Timer2->Enabled=true;
Button5->Enabled=False;
Bsamplenum=1;
trial_direction = 1;
Form4->Visible=true;
strength_trial = false;

if (read_type == serial) {
     // Serial Port Communication
     DCB dcbCommPort;

     hComm=CreateFile("COM1",GENERIC_READ|GENERIC_WRITE,0,NULL,OPEN_EXIST
        ING,0,NULL);
     GetCommState(hComm,&dcbCommPort);
     BuildCommDCB("38400,N,8,1", &dcbCommPort);
     SetCommState(hComm,&dcbCommPort);
     SetCommMask(hComm,EV_RXCHAR);
     GetCommTimeouts(hComm,&ctmoNew);
     ctmoNew.ReadIntervalTimeout=2.7;
     ctmoNew.ReadTotalTimeoutConstant=0;
     ctmoNew.ReadTotalTimeoutMultiplier=0;
     SetCommTimeouts(hComm,&ctmoNew);
}

// Get start time
start_time = timeGetTime();
switch (read_type) {
     case DAQcard:
          iStatus = ICD_get_data(&tmp_ICD_x_reading, &tmp_ICD_y_reading);
          if (DEBUG_lo) printf("iStatus, ICD_get_data = %d\n", iStatus);
          direction = Bdata[0] = tmp_ICD_x_reading + 2048;
          speed = Bdata[1] = tmp_ICD_y_reading + 2048;
          Bdata_time[0] = timeGetTime() - start_time;
          break;
     case serial:
          ClearCommError(hComm,&dwError,&cs);
          while(1)
          {
               if (WaitCommEvent(hComm,&dwEvent,NULL))
                    if (dwEvent & EV_RXCHAR)
                    {
                         ClearCommError(hComm,&dwError,&cs);
                         ReadFile(hComm,InBuff,4,&dwBytesRead,NULL);
                         ClearCommError(hComm,&dwError,&cs);

                         ab1=(unsigned int)(unsigned char)(InBuff[0]);
                         if ((ab1&240)!=16) break;
                         ab2=(unsigned int)(unsigned char)(InBuff[2]);
                         if ((ab2&240)!=32) break;

                         direction=Bdata[0]=(ab1&15)*256 +
                              (unsigned int)(unsigned char)(InBuff[1]);
                         speed=Bdata[1]=(ab2&15)*256 +
                              (unsigned int)(unsigned char)(InBuff[3]);
                         Bdata_time[0] = timeGetTime() - start_time;
                         break;
```

```
                    }
               }
               break;
     } // end switch

     while(1)
     {
          if (read_type == serial)
               ClearCommError(hComm,&dwError,&cs);

          Application->ProcessMessages();

          switch (read_type) {
               case DAQcard:
                    current_time = timeGetTime() - start_time;
                    if ((current_time - Bdata_time[Bsamplenum-1]) >=
sample_period) {
                         iStatus = ICD_get_data(&tmp_ICD_x_reading,
&tmp_ICD_y_reading);
                         if (DEBUG_lo)
                              printf("iStatus, ICD_get_data = %d\n", iStatus);
                         direction=Bdata[Bsamplenum*2]=tmp_ICD_x_reading + 2048;
                         speed = Bdata[Bsamplenum*2+1]=tmp_ICD_y_reading + 2048;
                         Bdata_time[Bsamplenum] = timeGetTime() - start_time;
                         ++Bsamplenum;
                    }
                    break;
               case serial:
                    while(1)
                    {
                         if (WaitCommEvent(hComm,&dwEvent,NULL))
                              if (dwEvent & EV_RXCHAR)
                              {
                                   ClearCommError(hComm,&dwError,&cs);
                                   ReadFile(hComm,InBuff,4,&dwBytesRead,NULL);
                                   ClearCommError(hComm,&dwError,&cs);

                                   ab1=(unsigned int)(unsigned char)(InBuff[0]);
                                   if ((ab1&240)!=16) break;
                                   ab2=(unsigned int)(unsigned char)(InBuff[2]);
                                   if ((ab2&240)!=32) break;

                                   direction=Bdata[Bsamplenum*2]=(ab1&15)*256 +
                                        (unsigned int)(unsigned char)(InBuff[1]);
                                   speed=Bdata[Bsamplenum*2+1]=(ab2&15)*256 +
                                        (unsigned int)(unsigned char)(InBuff[3]);
                                 Bdata_time[Bsamplenum]=timeGetTime()-start_time;

                                   ++Bsamplenum;
                                   break;
                              }
                    }
                    break;
          } // end switch

          end_time = timeGetTime();
          if ((end_time - start_time) >= trial_time[trial_length]) break;
```

```
     }

     Timer1->Enabled=false;
     Timer2->Enabled=false;
     Form4->Close();
     Button5->Enabled=True;
     if (read_type == serial) {
          SetCommMask(hComm,0);
          PurgeComm(hComm,PURGE_RXABORT);
          SetCommTimeouts(hComm,&ctmoOld);
          CloseHandle(hComm);
     }

     for (i=0; i<Bsamplenum*2; i+=2) {
          aforce[j]=(int) sqrt((Bdata[i]-2048)*(Bdata[i]-2048)+(Bdata[i+1]-
2048)*(Bdata[i+1]-2048));
          j++;
     }

     for (i=0; i<Bsamplenum; i++)
          if (maxforce<aforce[i])  {
               maxforce=aforce[i];
               maxforceindex=i;
          }

     if (maxforceindex-maxforcewindow<0) indexl=0;
     else indexl=maxforceindex-maxforcewindow;
     if (maxforceindex+maxforcewindow>Bsamplenum)indexh=Bsamplenum-1;
     else indexh=maxforceindex+maxforcewindow;

     for (j=indexl;j<=indexh;j++)
          Bmaxmeanforce+=aforce[j];
     Bmaxmeanforce= Bmaxmeanforce /(indexh-indexl+1);

     for (i=Bsamplenum/2;i<Bsamplenum;i++)
          Bcomfortforce+=aforce[i];
     Bcomfortforce = Bcomfortforce / (Bsamplenum/2);

     Edit11->Text=IntToStr(Bmaxmeanforce);
     Edit10->Text=IntToStr(Bcomfortforce);

     int xplot, yplot;

     for (i=0; i<Bsamplenum*2; i+=2) {
          xplot=300+(250*(Bdata[i]-2048)/2048);
          yplot=300-(250*(Bdata[i+1]-2048)/2048);

          Canvas->Brush->Color = clGray;
          Canvas->Pen->Color = clGray;
          Canvas->Ellipse(xplot-4,yplot-4,xplot+4, yplot+4);
     }

}
//---------------------------------------------------------------------------

// left button click
void __fastcall TForm2::Button6Click(TObject *Sender)
```

```
{
     int maxforce=0, aforce[20000];
     int maxforceindex=0, j=0, i, indexh, indexl;
     int ab1, ab2;
     DWORD current_time;

     // Device Readings
     int iStatus;
     float tmp_ICD_x_reading = 0.0;
     float tmp_ICD_y_reading = 0.0;
     unsigned char InBuff[10];
     DWORD dwBytesRead;

     // initializations
     gauge_progress = 100;
     Timer1->Enabled=true;
     Timer2->Enabled=true;
     Button6->Enabled=False;
     Lsamplenum=1;
     trial_direction = 2;
     Form4->Visible=true;
     strength_trial = false;

     if (read_type == serial) {
          // Serial Port Communication
          DCB dcbCommPort;

          hComm=CreateFile("COM1",GENERIC_READ|GENERIC_WRITE,0,NULL,OPEN_EXIST
               ING,0,NULL);
          GetCommState(hComm,&dcbCommPort);
          BuildCommDCB("38400,N,8,1", &dcbCommPort);
          SetCommState(hComm,&dcbCommPort);
          SetCommMask(hComm,EV_RXCHAR);
          GetCommTimeouts(hComm,&ctmoNew);
          ctmoNew.ReadIntervalTimeout=2.7;
          ctmoNew.ReadTotalTimeoutConstant=0;
          ctmoNew.ReadTotalTimeoutMultiplier=0;
          SetCommTimeouts(hComm,&ctmoNew);
     }

     // Get start time
     start_time = timeGetTime();
     switch (read_type) {
          case DAQcard:
               iStatus = ICD_get_data(&tmp_ICD_x_reading, &tmp_ICD_y_reading);
               if (DEBUG_lo) printf("iStatus, ICD_get_data = %d\n", iStatus);
               direction = Ldata[0] = tmp_ICD_x_reading + 2048;
               speed = Ldata[1] = tmp_ICD_y_reading + 2048;
               Ldata_time[0] = timeGetTime() - start_time;
               break;
          case serial:
               ClearCommError(hComm,&dwError,&cs);
               while(1)
               {
                    if (WaitCommEvent(hComm,&dwEvent,NULL))
                         if (dwEvent & EV_RXCHAR)
                         {
```

```
                              ClearCommError(hComm,&dwError,&cs);
                              ReadFile(hComm,InBuff,4,&dwBytesRead,NULL);
                              ClearCommError(hComm,&dwError,&cs);

                              ab1=(unsigned int)(unsigned char)(InBuff[0]);
                              if ((ab1&240)!=16) break;
                              ab2=(unsigned int)(unsigned char)(InBuff[2]);
                              if ((ab2&240)!=32) break;

                              direction=Ldata[0]=(ab1&15)*256 +
                                   (unsigned int)(unsigned char)(InBuff[1]);
                              speed=Ldata[1]=(ab2&15)*256 +
                                   (unsigned int)(unsigned char)(InBuff[3]);
                              Ldata_time[0] = timeGetTime() - start_time;
                              break;
                         }
                    }
                    break;
          } // end switch

          while(1)
          {
               if (read_type == serial)
                    ClearCommError(hComm,&dwError,&cs);

               Application->ProcessMessages();

               switch (read_type) {
                    case DAQcard:
                         current_time = timeGetTime() - start_time;
                         if ((current_time - Ldata_time[Lsamplenum-1])
                              >= sample_period) {
                              iStatus = ICD_get_data(&tmp_ICD_x_reading,
&tmp_ICD_y_reading);
                              if (DEBUG_lo)
                                   printf("iStatus, ICD_get_data = %d\n", iStatus);
                              direction = Ldata[Lsamplenum*2]=tmp_ICD_x_reading+2048;
                              speed = Ldata[Lsamplenum*2+1] = tmp_ICD_y_reading+2048;
                              Ldata_time[Lsamplenum] = timeGetTime() - start_time;
                              ++Lsamplenum;
                         }
                         break;
                    case serial:
                         while(1)
                         {
                              if (WaitCommEvent(hComm,&dwEvent,NULL))
                                   if (dwEvent & EV_RXCHAR)
                                   {
                                        ClearCommError(hComm,&dwError,&cs);
                                        ReadFile(hComm,InBuff,4,&dwBytesRead,NULL);
                                        ClearCommError(hComm,&dwError,&cs);

                                        ab1=(unsigned int)(unsigned char)(InBuff[0]);
                                        if ((ab1&240)!=16) break;
                                        ab2=(unsigned int)(unsigned char)(InBuff[2]);
                                        if ((ab2&240)!=32) break;
```

```
                                direction=Ldata[Lsamplenum*2]=(ab1&15)*256 +
                                     (unsigned int)(unsigned char)(InBuff[1]);
                                speed=Ldata[Lsamplenum*2+1]=(ab2&15)*256 +
                                     (unsigned int)(unsigned char)(InBuff[3]);
                              Ldata_time[Lsamplenum]=timeGetTime()-start_time;

                                ++Lsamplenum;
                                break;
                           }
                     }
                     break;
          } // end switch

          end_time = timeGetTime();
          if ((end_time - start_time) >= trial_time[trial_length]) break;
     }

     Timer1->Enabled=false;
     Timer2->Enabled=false;
     Form4->Close();
     Button6->Enabled=True;
     if (read_type == serial) {
          SetCommMask(hComm,0);
          PurgeComm(hComm,PURGE_RXABORT);
          SetCommTimeouts(hComm,&ctmoOld);
          CloseHandle(hComm);
     }

     for (i=0; i<Lsamplenum*2; i+=2) {
          aforce[j]=(int) sqrt((Ldata[i]-2048)*(Ldata[i]-2048)+(Ldata[i+1]-
2048)*(Ldata[i+1]-2048));
          j++;
     }

     for (i=0; i<Lsamplenum; i++)
          if (maxforce<aforce[i])  {
               maxforce=aforce[i];
               maxforceindex=i;
          }

     if (maxforceindex-maxforcewindow<0) indexl=0;
     else indexl=maxforceindex-maxforcewindow;
     if (maxforceindex+maxforcewindow>Lsamplenum)indexh=Lsamplenum-1;
     else indexh=maxforceindex+maxforcewindow;

     for (j=indexl;j<=indexh;j++)
          Lmaxmeanforce+=aforce[j];
     Lmaxmeanforce= Lmaxmeanforce /(indexh-indexl+1);

     for (i=Lsamplenum/2;i<Lsamplenum;i++)
          Lcomfortforce+=aforce[i];
     Lcomfortforce = Lcomfortforce / (Lsamplenum/2);

     Edit13->Text=IntToStr(Lmaxmeanforce);
     Edit12->Text=IntToStr(Lcomfortforce);

     int xplot, yplot, xsum=0, ysum=0, nsample=Lsamplenum;
```

```
     for (i=0; i<Lsamplenum*2; i+=2) {
          xplot=300+(250*(Ldata[i]-2048)/2048);
          yplot=300-(250*(Ldata[i+1]-2048)/2048);
          if (abs(xplot-300)<=20 & abs(yplot-300)<=20)
               nsample-=1;
          else {
               xsum+=xplot;
               ysum+=yplot;
          }
          Canvas->Brush->Color = clGray;
          Canvas->Pen->Color = clGray;
          Canvas->Ellipse(xplot-4,yplot-4,xplot+4, yplot+4);
     }

     LBiasangle=PI-(PI/2-atan2(ysum/nsample-300,xsum/nsample-300));
     Form2->Canvas->Brush->Color = clRed;
     Form2->Canvas->Pen->Color = clRed;
     Form2->Canvas->MoveTo(300, 300);
     Form2->Canvas->LineTo((int)(xsum/nsample), (int) (ysum/nsample));

     if (LBiasangle<0) LBiasangle=2*PI+LBiasangle;
     Form2->Edit20->Text=FloatToStr(LBiasangle*180/PI);

     Form2->Edit19->Text=FloatToStr(cos(fabs(LBiasangle-Biasangle)));
}
//---------------------------------------------------------------------------

//right button click
void __fastcall TForm2::Button7Click(TObject *Sender)
{
     int maxforce=0, aforce[20000];
     int maxforceindex=0, j=0, i, indexh, indexl;
     int ab1, ab2;
     DWORD current_time;

     // Device Readings
     int iStatus;
     float tmp_ICD_x_reading = 0.0;
     float tmp_ICD_y_reading = 0.0;
     unsigned char InBuff[10];
     DWORD dwBytesRead;

     // initializations
     gauge_progress = 0;
     Timer1->Enabled=true;
     Timer2->Enabled=true;
     Button7->Enabled=False;
     Rsamplenum=1;
     trial_direction = 3;
     Form4->Visible=true;
     strength_trial = false;

     if (read_type == serial) {
          // Serial Port Communication
          DCB dcbCommPort;
```

```
    hComm=CreateFile("COM1",GENERIC_READ|GENERIC_WRITE,0,NULL,OPEN_EXIST
        ING,0,NULL);
    GetCommState(hComm,&dcbCommPort);
    BuildCommDCB("38400,N,8,1", &dcbCommPort);
    SetCommState(hComm,&dcbCommPort);
    SetCommMask(hComm,EV_RXCHAR);
    GetCommTimeouts(hComm,&ctmoNew);
    ctmoNew.ReadIntervalTimeout=2.7;
    ctmoNew.ReadTotalTimeoutConstant=0;
    ctmoNew.ReadTotalTimeoutMultiplier=0;
    SetCommTimeouts(hComm,&ctmoNew);
}

// Get start time
start_time = timeGetTime();
switch (read_type) {
    case DAQcard:
        iStatus = ICD_get_data(&tmp_ICD_x_reading, &tmp_ICD_y_reading);
        if (DEBUG_lo) printf("iStatus, ICD_get_data = %d\n", iStatus);
        direction = Rdata[0] = tmp_ICD_x_reading + 2048;
        speed = Rdata[1] = tmp_ICD_y_reading + 2048;
        Rdata_time[0] = timeGetTime() - start_time;
        break;
    case serial:
        ClearCommError(hComm,&dwError,&cs);
        while(1)
        {
            if (WaitCommEvent(hComm,&dwEvent,NULL))
                if (dwEvent & EV_RXCHAR)
                {
                    ClearCommError(hComm,&dwError,&cs);
                    ReadFile(hComm,InBuff,4,&dwBytesRead,NULL);
                    ClearCommError(hComm,&dwError,&cs);

                    ab1=(unsigned int)(unsigned char)(InBuff[0]);
                    if ((ab1&240)!=16) break;
                    ab2=(unsigned int)(unsigned char)(InBuff[2]);
                    if ((ab2&240)!=32) break;

                    direction=Rdata[0]=(ab1&15)*256 +
                        (unsigned int)(unsigned char)(InBuff[1]);
                    speed=Rdata[1]=(ab2&15)*256 +
                        (unsigned int)(unsigned char)(InBuff[3]);
                    Rdata_time[0] = timeGetTime() - start_time;
                    break;
                }
        }
        break;
} // end switch

 while(1)
 {
    if (read_type == serial)
        ClearCommError(hComm,&dwError,&cs);

    Application->ProcessMessages();
```

```
          switch (read_type) {
               case DAQcard:
                    current_time = timeGetTime() - start_time;
                    if ((current_time - Rdata_time[Rsamplenum-1]) >=
                         sample_period) {
                         iStatus = ICD_get_data(&tmp_ICD_x_reading,
&tmp_ICD_y_reading);
                         if (DEBUG_lo)
                              printf("iStatus, ICD_get_data = %d\n", iStatus);
                         direction = Rdata[Rsamplenum*2]=tmp_ICD_x_reading+2048;
                         speed = Rdata[Rsamplenum*2+1] = tmp_ICD_y_reading+2048;
                         Rdata_time[Rsamplenum] = timeGetTime() - start_time;
                         ++Rsamplenum;
                    }
                    break;
               case serial:
                    while(1)
                    {
                         if (WaitCommEvent(hComm,&dwEvent,NULL))
                              if (dwEvent & EV_RXCHAR)
                              {
                                   ClearCommError(hComm,&dwError,&cs);
                                   ReadFile(hComm,InBuff,4,&dwBytesRead,NULL);
                                   ClearCommError(hComm,&dwError,&cs);

                                   ab1=(unsigned int)(unsigned char)(InBuff[0]);
                                   if ((ab1&240)!=16) break;
                                   ab2=(unsigned int)(unsigned char)(InBuff[2]);
                                   if ((ab2&240)!=32) break;

                                   direction=Rdata[Rsamplenum*2]=(ab1&15)*256 +
                                        (unsigned int)(unsigned char)(InBuff[1]);
                                   speed=Rdata[Rsamplenum*2+1]=(ab2&15)*256 +
                                        (unsigned int)(unsigned char)(InBuff[3]);
                                 Rdata_time[Rsamplenum]=timeGetTime()-start_time;

                                   ++Rsamplenum;
                                   break;
                              }
                    }
                    break;
          } // end switch

          end_time = timeGetTime();
          if ((end_time - start_time) >= trial_time[trial_length]) break;
     }

     Timer1->Enabled=false;
     Timer2->Enabled=false;
     Form4->Close();
     Button7->Enabled=True;
     if (read_type == serial) {
          SetCommMask(hComm,0);
          PurgeComm(hComm,PURGE_RXABORT);
          SetCommTimeouts(hComm,&ctmoOld);
          CloseHandle(hComm);
     }
```

```
    //find digital force input for all the data
    for (i=0; i<Rsamplenum*2; i+=2) {
        aforce[j]=(int) sqrt((Rdata[i]-2048)*(Rdata[i]-2048)+(Rdata[i+1]-
2048)*(Rdata[i+1]-2048));
        j++;
    }

    //find max digital force for the first half of the trial
    for (i=0; i<Rsamplenum; i++)
        if (maxforce<aforce[i])  {
            maxforce=aforce[i];
            maxforceindex=i;
        }

    //find index values for max force +/- 100 samples, but ensure that it's
    // within bounds for the first half of the trial
    if (maxforceindex-maxforcewindow<0) indexl=0;
    else indexl=maxforceindex-maxforcewindow;
    if (maxforceindex+maxforcewindow>Rsamplenum)indexh=Rsamplenum-1;
    else indexh=maxforceindex+maxforcewindow;

    //the maximum force is the averaged value of the max force +/- ~100
    // data points
    for (j=indexl;j<=indexh;j++)
        Rmaxmeanforce+=aforce[j];
    Rmaxmeanforce= Rmaxmeanforce /(indexh-indexl+1);

    //the comfort force is the average force for the 2nd half of the trial
    for (i=Rsamplenum/2;i<Rsamplenum;i++)
        Rcomfortforce+=aforce[i];
    Rcomfortforce = Rcomfortforce / (Rsamplenum/2);

    Edit15->Text=IntToStr(Rmaxmeanforce);
    Edit14->Text=IntToStr(Rcomfortforce);

    int xplot, yplot;

    for (i=0; i<Rsamplenum*2; i+=2) {
        xplot=300+(250*(Rdata[i]-2048)/2048);
        yplot=300-(250*(Rdata[i+1]-2048)/2048);

        Canvas->Brush->Color = clGray;
        Canvas->Pen->Color = clGray;
        Canvas->Ellipse(xplot-4,yplot-4,xplot+4, yplot+4);
    }

}
//---------------------------------------------------------------------------

//two minute button click
void __fastcall TForm2::Button2Click(TObject *Sender)
{
    int j=0, i;
    int ab1, ab2;
    DWORD current_time;
```

```
// Device Readings
int iStatus;
float tmp_ICD_x_reading = 0.0;
float tmp_ICD_y_reading = 0.0;
unsigned char InBuff[10];
DWORD dwBytesRead;

// initializations
gauge_progress = 0;
Timer1->Enabled=true;
Timer2->Enabled=true;
Button7->Enabled=False;
Ssamplenum = 1;
trial_direction = 0;
Form4->Visible=true;
strength_trial = true;

if (read_type == serial) {
     // Serial Port Communication
     DCB dcbCommPort;
     hComm=CreateFile("COM1",GENERIC_READ|GENERIC_WRITE,0,NULL,OPEN_EXIST
          ING,0,NULL);
     GetCommState(hComm,&dcbCommPort);
     BuildCommDCB("38400,N,8,1", &dcbCommPort);
     SetCommState(hComm,&dcbCommPort);
     SetCommMask(hComm,EV_RXCHAR);
     GetCommTimeouts(hComm,&ctmoNew);
     ctmoNew.ReadIntervalTimeout=2.7;
     ctmoNew.ReadTotalTimeoutConstant=0;
     ctmoNew.ReadTotalTimeoutMultiplier=0;
     SetCommTimeouts(hComm,&ctmoNew);
}

// Get start time
start_time = timeGetTime();
switch (read_type) {
     case DAQcard:
          iStatus = ICD_get_data(&tmp_ICD_x_reading, &tmp_ICD_y_reading);
          if (DEBUG_lo) printf("iStatus, ICD_get_data = %d\n", iStatus);
          direction = Sdata[0] = tmp_ICD_x_reading + 2048;
          speed = Sdata[1] = tmp_ICD_y_reading + 2048;
          Sdata_time[0] = timeGetTime() - start_time;
          break;
     case serial:
          ClearCommError(hComm,&dwError,&cs);
          while(1)
          {
               if (WaitCommEvent(hComm,&dwEvent,NULL))
                    if (dwEvent & EV_RXCHAR)
                    {
                         ClearCommError(hComm,&dwError,&cs);
                         ReadFile(hComm,InBuff,4,&dwBytesRead,NULL);
                         ClearCommError(hComm,&dwError,&cs);

                         ab1=(unsigned int)(unsigned char)(InBuff[0]);
                         if ((ab1&240)!=16) break;
                         ab2=(unsigned int)(unsigned char)(InBuff[2]);
```

```
                        if ((ab2&240)!=32) break;

                        direction=Sdata[0]=(ab1&15)*256 +
                             (unsigned int)(unsigned char)(InBuff[1]);
                        speed=Sdata[1]=(ab2&15)*256 +
                             (unsigned int)(unsigned char)(InBuff[3]);
                        Sdata_time[0] = timeGetTime() - start_time;
                        break;
                    }
               }
               break;
          } // end switch

          while(1)
          {
               if (read_type == serial)
                    ClearCommError(hComm,&dwError,&cs);

               Application->ProcessMessages();

               switch (read_type) {
                    case DAQcard:
                         current_time = timeGetTime() - start_time;
                         if ((current_time - Sdata_time[Ssamplenum-1]) >=
                              sample_period)
                         {
                              iStatus = ICD_get_data(&tmp_ICD_x_reading,
&tmp_ICD_y_reading);
                              if (DEBUG_lo)
                                   printf("iStatus, ICD_get_data = %d\n", iStatus);
                              direction = Sdata[Ssamplenum*2]=tmp_ICD_x_reading+2048;
                              speed = Sdata[Ssamplenum*2+1] = tmp_ICD_y_reading+2048;
                              Sdata_time[Ssamplenum] = timeGetTime() - start_time;
                              ++Ssamplenum;
                         }
                         break;
                    case serial:
                         while(1)
                         {
                              if (WaitCommEvent(hComm,&dwEvent,NULL))
                                   if (dwEvent & EV_RXCHAR)
                                   {
                                        ClearCommError(hComm,&dwError,&cs);
                                        ReadFile(hComm,InBuff,4,&dwBytesRead,NULL);
                                        ClearCommError(hComm,&dwError,&cs);

                                        ab1=(unsigned int)(unsigned char)(InBuff[0]);
                                        if ((ab1&240)!=16) break;
                                        ab2=(unsigned int)(unsigned char)(InBuff[2]);
                                        if ((ab2&240)!=32) break;

                                        direction=Sdata[Ssamplenum*2]=(ab1&15)*256 +
                                             (unsigned int)(unsigned char)(InBuff[1]);
                                        speed=Sdata[Ssamplenum*2+1]=(ab2&15)*256 +
                                             (unsigned int)(unsigned char)(InBuff[3]);
                                      Sdata_time[Ssamplenum]=timeGetTime()-start_time;
```

```
                                ++Ssamplenum;
                                break;
                            }
                    }
                    break;
            } // end switch

            end_time = timeGetTime();
            if ((end_time - start_time) >= Strial_time[trial_length]) break;
        }

        Timer1->Enabled=false;
        Timer2->Enabled=false;
        Form4->Close();
        Button7->Enabled=True;
        if (read_type == serial) {
            SetCommMask(hComm,0);
            PurgeComm(hComm,PURGE_RXABORT);
            SetCommTimeouts(hComm,&ctmoOld);
            CloseHandle(hComm);
        }
}
//---------------------------------------------------------------------------

//this timer controls the text display of the joystick forces
void __fastcall TForm2::Timer1Timer(TObject *Sender)
{
    char strBuffer[9];
    float time_remaining;

    if (strength_trial == true)
 time_remaining = (Strial_time[trial_length]-(end_time-start_time))/1000.0;
    else
 time_remaining = (trial_time[trial_length]-(end_time-start_time))/1000.0;

    sprintf(strBuffer, "  %d", (int)(time_remaining+.5));
    Edit21->Text=strBuffer;

    //update speed and direction in text box
    Edit2->Text=IntToStr(direction);
    Edit3->Text=IntToStr(speed);
}
//---------------------------------------------------------------------------

// this timer controls the feedback gauges
void __fastcall TForm2::Timer2Timer(TObject *Sender)
{
    int dir, spd;
    int force_limit = 2000;
    int force_magnitude, DZ_magnitude;
    int within_bounds;
    float gauge_velocity = .3;
//  DWORD gauge_time, gauge_dt;

    //determine if force is within bounds
    dir = direction - 2048;
    spd = speed - 2048;
```

```
     force_magnitude = sqrt(dir*dir + spd*spd);
     DZ_magnitude = sqrt(MaxDZForce*MaxDZForce+MaxDZDirection*MaxDZDirection);

     if ((force_magnitude > 0.9*force_limit) ||
          (force_magnitude < 1.5*DZ_magnitude))
     {
          within_bounds = 0;
     }
     else within_bounds = 1;

     //update height of gauge
//      gauge_time = timeGetTime();
//      gauge_dt = abs(gauge_time - end_time);
     if (within_bounds)
     {
          if ((trial_direction == 0) ||(trial_direction == 3))
               gauge_progress += gauge_velocity;
          else
               gauge_progress -= gauge_velocity;

          if (gauge_progress > 100)
               gauge_progress = 0;
          if (gauge_progress < 0)
               gauge_progress = 100;
     }

     if ((trial_direction == 0) ||(trial_direction == 1))
          Form4->CGauge2->Progress = (long)gauge_progress;
     else
          Form4->CGauge1->Progress = (long)gauge_progress;
}
//---------------------------------------------------------------------------

void __fastcall TForm2::FormPaint(TObject *Sender)
{
     square[0].x=50;
     square[0].y=50;

     square[1].x=550;
     square[1].y=50;

     square[2].x=550;
     square[2].y=550;

     square[3].x=50;
     square[3].y=550;

     Canvas->Brush->Color = clWhite;
     Canvas->Pen->Color = clSilver;
     Canvas->Pen->Width = 3;
     Canvas->Polygon(square,3);

     Canvas->Brush->Color = clBlack;
     Canvas->Pen->Color = clBlack;
     Canvas->Ellipse(300-5,300-5,300+5, 300+5);

}
```

```
//---------------------------------------------------------------------------



void __fastcall TForm2::FormActivate(TObject *Sender)
{
     Form2->Edit17->Text=IntToStr(MaxDZForce);
     Form2->Edit18->Text=IntToStr(MaxDZDirection);

}
//---------------------------------------------------------------------------

void canvas_draw(void)
{
     int xplot, yplot, i;

     for (i=0; i<Rsamplenum*2; i+=2) {
          xplot=300+(250*(Rdata[i]-2048)/2048);
          yplot=300-(250*(Rdata[i+1]-2048)/2048);

          Form2->Canvas->Brush->Color = clGray;
          Form2->Canvas->Pen->Color = clGray;

          Form2->Canvas->Ellipse(xplot-4,yplot-4,xplot+4, yplot+4);
     }

     for (i=0; i<Lsamplenum*2; i+=2) {
          xplot=300+(250*(Ldata[i]-2048)/2048);
          yplot=300-(250*(Ldata[i+1]-2048)/2048);

          Form2->Canvas->Brush->Color = clGray;
          Form2->Canvas->Pen->Color = clGray;

          Form2->Canvas->Ellipse(xplot-4,yplot-4,xplot+4, yplot+4);
     }

     for (i=0; i<Bsamplenum*2; i+=2) {
          xplot=300+(250*(Bdata[i]-2048)/2048);
          yplot=300-(250*(Bdata[i+1]-2048)/2048);

          Form2->Canvas->Brush->Color = clGray;
          Form2->Canvas->Pen->Color = clGray;

          Form2->Canvas->Ellipse(xplot-4,yplot-4,xplot+4, yplot+4);
     }

     for (i=0; i<Fsamplenum*2; i+=2) {
          xplot=300+(250*(Fdata[i]-2048)/2048);
          yplot=300-(250*(Fdata[i+1]-2048)/2048);

          Form2->Canvas->Brush->Color = clGray;
          Form2->Canvas->Pen->Color = clGray;

          Form2->Canvas->Ellipse(xplot-4,yplot-4,xplot+4, yplot+4);
     }

     switch(shape) {
```

```
         case 1:  {
              Form2->Canvas->Brush->Color = clWhite;
              Form2->Canvas->Pen->Color = clRed;
              Form2->Canvas->Ellipse(300-MaxDZDirection*250/2048,300-
MaxDZForce*250/2048, 300+MaxDZDirection*250/2048, 300+MaxDZForce*250/2048);
              break;
         }
         case 2:  {
              Form2->Canvas->Brush->Color = clWhite;
              Form2->Canvas->Pen->Color = clRed;
              Form2->Canvas->Rectangle(300-(MaxDZDirection*250/2048+1),300-
(MaxDZForce*250/2048+1), 300+(MaxDZDirection*250/2048+1),
300+(MaxDZForce*250/2048+1));
              break;
         }
         case 3:  {
              int xc, yc, orignx, origny;
              float theta, xn, yn;
              Form2->Canvas->Brush->Color = clWhite;
              Form2->Canvas->Pen->Color = clRed;

              for (theta=0;theta<=2*PI;theta+=0.01) {
                   xn=MaxDZDirection*250/2048*cos(theta);
                   yn=MaxDZForce*250/2048*sin(theta);
                   xc=xn+300;
                   yc=yn+300;
                   if (theta==0.0) {
                        Form2->Canvas->MoveTo(xc,yc);
                        orignx=xc;
                        origny=yc;
                   }
                   else Form2->Canvas->LineTo(xc, yc);
              }

              Form2->Canvas->LineTo(orignx, origny);
              break;
         }
     }
}

void __fastcall TForm2::ComboBox1Change(TObject *Sender)
{
     index=ComboBox1->ItemIndex;
     selectedresource[index]();
     Form2->Button8->Enabled=true;
}
//---------------------------------------------------------------------------

void repaint()
{

     Form2->Refresh();

     Form2->Canvas->Brush->Color = clWhite;
     Form2->Canvas->Pen->Color = clSilver;

     int xc, yc, orignx, origny;
```

```
     float theta, xn, yn;

     switch(index) {
          case 1:   {
               for (theta=0;theta<=2*PI;theta+=0.01) {
                    xn=xvalue*cos(theta);
                    yn=yvalue*sin(theta);
                    xc=xn*cos(angle)-yn*sin(angle)+300;
                    yc=xn*sin(angle)+yn*cos(angle)+300;
                    if (theta==0.0) {
                         Form2->Canvas->MoveTo(xc,yc);
                         orignx=xc;
                         origny=yc;
                    }
                    else Form2->Canvas->LineTo(xc, yc);
               }
               Form2->Canvas->LineTo(orignx, origny);
               break;
          }
          case 2:  {
               Draw_Astroid(xvalue,yvalue);
               break;
          }
          case 3:  {
               Draw_Diamond(xvalue,yvalue);
               break;
          }
         case 4: {
               //Form1->Canvas->Ellipse(300-xvalue,300-
yvalue,300+xvalue,300+yvalue);
               for (theta=0;theta<=2*PI;theta+=0.01) {
                    xn=xvalue*cos(theta);
                    yn=yvalue*sin(theta);
                    xc=xn*cos(angle)-yn*sin(angle)+300;
                    yc=xn*sin(angle)+yn*cos(angle)+300;
                    if (theta==0.0) {
                         Form2->Canvas->MoveTo(xc,yc);
                         orignx=xc;
                         origny=yc;
                    }
                    else Form2->Canvas->LineTo(xc, yc);
               }
               Form2->Canvas->LineTo(orignx, origny);
               break;
          }
     }

     Form2->Canvas->Brush->Color = clBlack;
     Form2->Canvas->Pen->Color = clBlack;
     Form2->Canvas->Ellipse(300-5,300-5,300+5, 300+5);
}

void __fastcall TForm2::TrackBar1Change(TObject *Sender)
{
     Form2->Edit1->Text = "     " + IntToStr(TrackBar1->Position*2048/250);
     xvalue = TrackBar1->Position;
```

```
    if (index==1) {
       Form2->Edit6->Text = "     " + IntToStr(TrackBar1->Position*2048/250);
        TrackBar2->Position=TrackBar1->Position;
    }
    else {
       Form2->Edit6->Text = "     " + IntToStr(TrackBar2->Position*2048/250);
    }
    yvalue = TrackBar2->Position;

    repaint();
}
//---------------------------------------------------------------------------

void __fastcall TForm2::TrackBar3Change(TObject *Sender)
{
    angle=TrackBar3->Position*PI/180;
    Form2->Edit7->Text = "     " + IntToStr(TrackBar3->Position);
    repaint();
}
//---------------------------------------------------------------------------

// when click to next step is pressed save bias axes force data and switch
forms
void __fastcall TForm2::Button4Click(TObject *Sender)
{
    FILE *ffdata, *fbdata, *fldata, *frdata, *fsdata;
    int i;
    int j;

    // write force data files from bias axes calculation
    ffdata=fopen("c:\\Settings\\forward_data.txt","w");
    fbdata=fopen("c:\\Settings\\backward_data.txt","w");
    fldata=fopen("c:\\Settings\\left_data.txt","w");
    frdata=fopen("c:\\Settings\\right_data.txt","w");
    fsdata=fopen("c:\\Settings\\strength_data.txt","w");
    fprintf(ffdata, "time (msec)\tdir\tspd\n");
    fprintf(fbdata, "time (msec)\tdir\tspd\n");
    fprintf(fldata, "time (msec)\tdir\tspd\n");
    fprintf(frdata, "time (msec)\tdir\tspd\n");
    fprintf(fsdata, "time (msec)\tdir\tspd\n");
    for (i=0; i<Fsamplenum; i++)
        fprintf(ffdata,"%d\t%d\t%d\n",Fdata_time[i],Fdata[2*i],Fdata[2*i+1]);
    for (i=0; i<Bsamplenum; i++)
        fprintf(fbdata,"%d\t%d\t%d\n",Bdata_time[i],Bdata[2*i],Bdata[2*i+1]);
    for (i=0; i<Lsamplenum; i++)
        fprintf(fldata,"%d\t%d\t%d\n",Ldata_time[i],Ldata[2*i],Ldata[2*i+1]);
    for (i=0; i<Rsamplenum; i++)
        fprintf(frdata,"%d\t%d\t%d\n",Rdata_time[i],Rdata[2*i],Rdata[2*i+1]);
    for (i=0; i<Ssamplenum; i++)
        fprintf(fsdata,"%d\t%d\t%d\n",Sdata_time[i],Sdata[2*i],Sdata[2*i+1]);
    fclose(ffdata);
    fclose(fbdata);
    fclose(fldata);
    fclose(frdata);
    fclose(fsdata);

    // switch forms
```

```
     Form2->Visible=false;
     Form3->Visible=true;
}
//---------------------------------------------------------------------------

void __fastcall TForm2::Button1Click(TObject *Sender)
{
     int maxForce=0, dzforce=0, meanComfort=0, Stickforce=4096;
     //max force is the max of the four directions
     if (maxForce<Fmaxmeanforce) maxForce=Fmaxmeanforce;
     if (maxForce<Bmaxmeanforce) maxForce=Bmaxmeanforce;
     if (maxForce<Lmaxmeanforce) maxForce=Lmaxmeanforce;
     if (maxForce<Rmaxmeanforce) maxForce=Rmaxmeanforce;

     //meanComfort is the mean of the comfort forces
     meanComfort=(Fcomfortforce+Bcomfortforce+Lcomfortforce+Rcomfortforce)/4;

     dzforce=(MaxDZForce>=MaxDZDirection ? MaxDZForce: MaxDZDirection);

     if (dzforce<0.4*maxForce) CheckBox2->Checked=true;
     else CheckBox2->Checked=false;

     if (cos(fabs(LBiasangle-Biasangle))<=0.7) CheckBox3->Checked=true;
     else CheckBox3->Checked=false;

     if (meanComfort>1.5*dzforce)  CheckBox1->Checked=true;
     else CheckBox1->Checked=false;

     if (maxForce < 0.5*Stickforce)  CheckBox4->Checked=true;
     else CheckBox4->Checked=false;
}
//---------------------------------------------------------------------------

void __fastcall TForm2::Button8Click(TObject *Sender)
{
     int xplot, yplot, i;
     int dir_comfortforce, spd_comfortforce;
     int dir_rot, spd_rot;

     for (i=0; i<Rsamplenum*2; i+=2) {
          xplot=300+(250*(Rdata[i]-2048)/2048);
          yplot=300-(250*(Rdata[i+1]-2048)/2048);

          Form2->Canvas->Brush->Color = clGray;
          Form2->Canvas->Pen->Color = clGray;

          Form2->Canvas->Ellipse(xplot-4,yplot-4,xplot+4, yplot+4);
     }

     for (i=0; i<Lsamplenum*2; i+=2) {
          xplot=300+(250*(Ldata[i]-2048)/2048);
          yplot=300-(250*(Ldata[i+1]-2048)/2048);

          Form2->Canvas->Brush->Color = clGray;
          Form2->Canvas->Pen->Color = clGray;

          Form2->Canvas->Ellipse(xplot-4,yplot-4,xplot+4, yplot+4);
```

```
    }

    for (i=0; i<Bsamplenum*2; i+=2) {
         xplot=300+(250*(Bdata[i]-2048)/2048);
         yplot=300-(250*(Bdata[i+1]-2048)/2048);

         Form2->Canvas->Brush->Color = clGray;
         Form2->Canvas->Pen->Color = clGray;

         Form2->Canvas->Ellipse(xplot-4,yplot-4,xplot+4, yplot+4);
    }

    for (i=0; i<Fsamplenum*2; i+=2) {
         xplot=300+(250*(Fdata[i]-2048)/2048);
         yplot=300-(250*(Fdata[i+1]-2048)/2048);

         Form2->Canvas->Brush->Color = clGray;
         Form2->Canvas->Pen->Color = clGray;

         Form2->Canvas->Ellipse(xplot-4,yplot-4,xplot+4, yplot+4);
    }

    switch(shape) {
         case 1:  {
              Form2->Canvas->Brush->Color = clWhite;
              Form2->Canvas->Pen->Color = clRed;
              Form2->Canvas->Ellipse(300-MaxDZDirection*250/2048,300-
MaxDZForce*250/2048, 300+MaxDZDirection*250/2048, 300+MaxDZForce*250/2048);
              break;
         }
         case 2:  {
              Form2->Canvas->Brush->Color = clWhite;
              Form2->Canvas->Pen->Color = clRed;
              Form2->Canvas->Rectangle(300-(MaxDZDirection*250/2048+1),300-
(MaxDZForce*250/2048+1), 300+(MaxDZDirection*250/2048+1),
300+(MaxDZForce*250/2048+1));
              break;
         }
         case 3:  {
              int xc, yc, orignx, origny;
              float theta, xn, yn;

              Form2->Canvas->Brush->Color = clWhite;
              Form2->Canvas->Pen->Color = clRed;

              for (theta=0;theta<=2*PI;theta+=0.01) {
                   xn=MaxDZDirection*250/2048*cos(theta);
                   yn=MaxDZForce*250/2048*sin(theta);
                   xc=xn+300;
                   yc=yn+300;
                   if (theta==0.0) {
                        Form2->Canvas->MoveTo(xc,yc);
                        orignx=xc;
                        origny=yc;
                   }
                   else Form2->Canvas->LineTo(xc, yc);
              }
```

```
            Form2->Canvas->LineTo(orignx, origny);
            break;
        }

    }

    //find gain for each direction
    XGain[0] = 0.90*(TrackBar1->Position*2048/250)/(float)Lcomfortforce;
    XGain[1] = 0.90*(TrackBar1->Position*2048/250)/(float)Rcomfortforce;
    YGain[0] = 0.90*(TrackBar2->Position*2048/250)/(float)Bcomfortforce;
    YGain[1] = 0.90*(TrackBar2->Position*2048/250)/(float)Fcomfortforce;
    Edit22->Text=FloatToStr(XGain[0]);
    Edit4->Text=FloatToStr(XGain[1]);
    Edit23->Text=FloatToStr(YGain[0]);
    Edit5->Text=FloatToStr(YGain[1]);

    Form2->Button4->Enabled=true;
}

void __fastcall TForm2::Button9Click(TObject *Sender)
{
    int iStatus;
    iStatus = ICD_find_offset();
}
//---------------------------------------------------------------------------
```

I.6 UNIT3.H

```
//---------------------------------------------------------------------------

#ifndef Unit3H
#define Unit3H
//---------------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <ExtCtrls.hpp>
#include <jpeg.hpp>
//---------------------------------------------------------------------------
class TForm3 : public TForm
{
__published:  // IDE-managed Components
    TPanel *Panel1;
    TLabel *Label1;
    TLabel *Label2;
    TLabel *Label3;
    TButton *Button1;
    TTimer *Timer1;
    TLabel *Label4;
    TLabel *Label5;
```

```
    TEdit *Edit1;
    TEdit *Edit2;
    TLabel *Label6;
    TEdit *Edit3;
    TButton *Button2;
    TButton *Button3;
    TLabel *Label7;
    TLabel *Label8;
    TEdit *Edit4;
    TLabel *Label11;
    TEdit *Edit5;
    TLabel *Label12;
    TEdit *Edit6;
    TLabel *Label13;
    TEdit *Edit7;
    TButton *Button4;
    TButton *Button5;
    TRadioButton *RadioButton1;
    TRadioButton *RadioButton2;
    TEdit *Edit8;
    TRadioButton *RadioButton3;
    TEdit *Edit9;
    TEdit *Edit10;
    TEdit *Edit11;
    TButton *Button6;
    void __fastcall Button1Click(TObject *Sender);
    void __fastcall Timer1Timer(TObject *Sender);

    void __fastcall FormPaint(TObject *Sender);
    void __fastcall Button2Click(TObject *Sender);
    void __fastcall Button3Click(TObject *Sender);
    void __fastcall RadioButton1Click(TObject *Sender);
    void __fastcall RadioButton2Click(TObject *Sender);
    void __fastcall Button5Click(TObject *Sender);
    void __fastcall Button4Click(TObject *Sender);
    void __fastcall RadioButton3Click(TObject *Sender);
    void __fastcall Button6Click(TObject *Sender);
    void __fastcall Edit6Change(TObject *Sender);
    void __fastcall Edit7Change(TObject *Sender);
private: // User declarations

public:      // User declarations
    __fastcall TForm3(TComponent* Owner);
};
//---------------------------------------------------------------------------
extern PACKAGE TForm3 *Form3;
//---------------------------------------------------------------------------
#endif
```

I.7 UNIT3.CPP

```
/*****************************************************************************

    Version: 1.0
    Modified: 11/28/06
    By: kwb

    Unit3.cpp

    >> PURPOSE <<
    This code determines the fatigue adjustment parameters. The form is run
    at the very beginning and then after the bias axes and gains are
    determined to determine the subject's fatigue index (FI). Times and FIs
    are then used to determine the alpha and beta parameters.

    >> NOMENCLATURE <<
    do i want to include nomenclature???

    >> HISTORY <<
    Version 1.0       kwb      20 june 2006
    original version
    11-28-06:   remove direction of force applicaton from panel

*****************************************************************************/
//---------------------------------------------------------------------------

#include <vcl.h>
#pragma hdrstop

#include <dos.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <windows.h>
#include <mmsystem.h>

#include "Unit1.h"
#include "Unit2.h"
#include "Unit3.h"
#include "Unit5.h"
#include "ICD.h"

#pragma package(smart_init)
#pragma resource "*.dfm"

#define PI 3.1415926
#define FI_MIN  30.0    //this is an arbitrary value and should be updated

TForm3 *Form3;

//global variables
extern int trial_length; //1=long, 0=short; set in Tuning_MS_Study.cpp
extern js_types read_type;//serial or DAQcard. defined in Tuning_MS_Study.cpp
```

```
//parameters to write to data files
extern AnsiString subject_id;
extern int MaxDZForce, MaxDZDirection, shape;
extern int index;   // template shape
extern float Biasangle;
extern float XGain[2], YGain[2];
extern float XGain_max[2], YGain_max[2], gain_multiplier;

//unit variables
DWORD fi_start_time[3], fi_end_time;
DWORD trial_time[2] = {8*1000, 32*1000}; // sec -> msec
int ab1, ab2, direction, speed, count;
int data[40000];
DWORD data_time[20000][3];  //time of data point collection
float input_mag[20000][3];  //input magnitude
float max_input;
float alpha = 0;
float beta = 0;

static unsigned int fi_count = 0; //index counter for FI
float FI[3];

//unit function prototypes
void update_fi_radio(void);
void find_alpha(void);
void find_beta(void);

//---------------------------------------------------------------------------
__fastcall TForm3::TForm3(TComponent* Owner)
     : TForm(Owner)
{
     //initialization of buttons
     Form3->Button2->Enabled = false;
     Form3->Button3->Enabled = false;
     Form3->Button4->Enabled = false;
     Form3->Button5->Enabled = true;
     update_fi_radio();

     //disable time feedback if long version (assumes subject testing is in
     // progress)
     if (trial_length == 1)
          Form3->Edit3->Enabled=false;
     else
          Form3->Edit3->Enabled=true;
}
//---------------------------------------------------------------------------

// this function collects force data after the Record Data button is clicked
void __fastcall TForm3::Button1Click(TObject *Sender)
{
     // Local Variables
     int iStatus;
     //serial varaiables
     HANDLE hComm=NULL;
     COMMTIMEOUTS ctmoNew={0},ctmoOld;
     DWORD dwBytesRead;
     DWORD dwEvent,dwError;
```

```
    COMSTAT cs;
    unsigned char InBuff[10];

    // Device Readings
    float tmp_ICD_x_reading = 0.0;
    float tmp_ICD_y_reading = 0.0;

    // Initialization
    Form3->Refresh();
    Form3->Timer1->Enabled=true;
    Form3->Button1->Enabled=false;
    Form3->Button4->Enabled=false;
    count=0;

    if (read_type == serial) {
         /* Serial Communication */
         DCB dcbCommPort;
         hComm=CreateFile("COM1",GENERIC_READ|GENERIC_WRITE,0,NULL,OPEN_EXIST
              ING,0,NULL);
         GetCommState(hComm,&dcbCommPort);
         BuildCommDCB("38400,N,8,1", &dcbCommPort);
         SetCommState(hComm,&dcbCommPort);
         SetCommMask(hComm,EV_RXCHAR);
         GetCommTimeouts(hComm,&ctmoNew);
         ctmoNew.ReadIntervalTimeout=2.7;
         ctmoNew.ReadTotalTimeoutConstant=0;
         ctmoNew.ReadTotalTimeoutMultiplier=0;
         SetCommTimeouts(hComm,&ctmoNew);
    }

    // Get start time
    fi_start_time[fi_count] = timeGetTime();

    while(1)
    {
         if (read_type == serial)
              ClearCommError(hComm,&dwError,&cs);

         Application->ProcessMessages();

         switch (read_type) {
              case DAQcard:
                   iStatus = ICD_get_data(&tmp_ICD_x_reading,
&tmp_ICD_y_reading);
                   if (DEBUG_lo)
                        printf("iStatus, ICD_get_data = %d\n", iStatus);
                   direction = data[count*2] = tmp_ICD_x_reading + 2048;
                   speed = data[count*2+1] = tmp_ICD_y_reading + 2048;
                   data_time[count][fi_count] =
                        timeGetTime()-fi_start_time[fi_count];
                   ++count;
                   break;
              case serial:
                   while(1)
                   {
                        if (WaitCommEvent(hComm,&dwEvent,NULL))
                             if (dwEvent & EV_RXCHAR)
```

```
                              {
                                   ClearCommError(hComm,&dwError,&cs);
                                   ReadFile(hComm,InBuff,4,&dwBytesRead,NULL);
                                   ClearCommError(hComm,&dwError,&cs);

                                   ab1=(unsigned int)(unsigned char)(InBuff[0]);
                                   if ((ab1&240)!=16) break;
                                   ab2=(unsigned int)(unsigned char)(InBuff[2]);
                                   if ((ab2&240)!=32) break;

                                   direction=data[count*2]=(ab1&15)*256 +
                                        (unsigned int)(unsigned char)(InBuff[1]);
                                   speed=data[count*2+1]=(ab2&15)*256 +
                                        (unsigned int)(unsigned char)(InBuff[3]);

                                   data_time[count][fi_count]=timeGetTime()-
fi_start_time[fi_count];

                                   ++count;
                                   break;
                              }
                         }
                         break;
               } // end switch

               fi_end_time = timeGetTime();
               if ((fi_end_time - fi_start_time[fi_count]) >=
trial_time[trial_length]) break;
          }

          Form3->Timer1->Enabled=False;
          Form3->Button1->Enabled=True;
          Form3->Button2->Enabled=True;
          Form3->Button3->Enabled=True;

          if (read_type == serial) {
               SetCommMask(hComm,0);
               PurgeComm(hComm,PURGE_RXABORT);
               SetCommTimeouts(hComm,&ctmoOld);
               CloseHandle(hComm);
          }

}
//---------------------------------------------------------------------------
void __fastcall TForm3::Timer1Timer(TObject *Sender)
{
     char strBuffer[9];
     float time_remaining;

     if (trial_length == 0)
     {
         time_remaining =
          (trial_time[trial_length]-(fi_end_time-fi_start_time[fi_count]))/
             1000.0;
          sprintf(strBuffer, " %d", (int)(time_remaining+.5));
          Form3->Edit3->Text=strBuffer;
//      Edit3->Text=IntToStr(time_remaining);
```

```
    }
    else
    {
        sprintf(strBuffer, " %d", (int)(trial_time[trial_length]/1000.0+1));
        Edit3->Text=strBuffer;
    }
    Form3->Edit1->Text=IntToStr(direction);
    Form3->Edit2->Text=IntToStr(speed);
}
//---------------------------------------------------------------------------

void __fastcall TForm3::FormPaint(TObject *Sender)
{
    Canvas->Brush->Color = clWhite;
    Canvas->Pen->Color = clSilver;
    Canvas->Pen->Width = 3;
    Canvas->Rectangle(50, 50, 550, 550);
}
//---------------------------------------------------------------------------

// plot the force magnitude when the user clicks Show Data
void __fastcall TForm3::Button2Click(TObject *Sender)
{
    // local variables
    int xplot, yplot, i, j=0;
    int xdata[20000], ydata[20000];

    // initialize
    max_input = 0;

    // calculate magnitude of input and find max value
    for (i=0; i<count*2; i+=2) {
        xdata[j] = data[i] - 2048;
        ydata[j] = data[i+1] - 2048;
        input_mag[j][fi_count]= sqrt(xdata[j]*xdata[j] + ydata[j]*ydata[j]);
        if (input_mag[j][fi_count] > max_input)
            max_input = input_mag[j][fi_count];
        j+=1;
    }

    // dimensions are scaled to fit the 500x500 plot window
    for (i=0; i<count; i++) {
        xplot=50+500*(float)data_time[i][fi_count]/trial_time[trial_length];
        yplot = 550 - 500*(input_mag[i][fi_count]/max_input);

        Form3->Canvas->Brush->Color = clGray;
        Form3->Canvas->Pen->Color = clGray;
        Form3->Canvas->Ellipse(xplot-1,yplot-1,xplot+1, yplot+1);
    }

}
//---------------------------------------------------------------------------

// calculate the fatigue index
//  FI = 100 - trapz(f_max_05-f, 5, 30)/(f_max_05*25)*100%
void __fastcall TForm3::Button3Click(TObject *Sender)
{
```

```
// local variables
int i;
int index_05;   // index where we reach 5 seconds
int max_input_05 = 0;
int index_30;   // index where we reach 30 seconds
float delta[2], area=0.0;
char strBuffer[9];

// find the index value for when 5 seconds occurs
for (i=250; i<count; i++) {
     if (data_time[i][fi_count] <= 5000)
          index_05 = i;
     else
          break;
}
// find the index value for when 30 seconds [or max time] occurs
for (i=index_05; i<count; i++) {
     if (data_time[i][fi_count] <= 30000) index_30 = i;
}

// find max value for the first 5 seconds
for (i=0; i<=index_05; i++) {
     if (input_mag[i][fi_count] > max_input_05)
          max_input_05 = input_mag[i][fi_count];
}

// find the area under the curve for the last part of the trial
for (i=index_05+1; i<index_30-1; i++) {
     delta[0] = max_input_05 - input_mag[i][fi_count];
     delta[1] = max_input_05 - input_mag[i+1][fi_count];
     area += ((delta[0]+delta[1])/2.0) *
          ((data_time[i+1][fi_count]-data_time[i][fi_count])/1000.0);
}

// calculate FI
FI[fi_count] = 1 - area/(max_input_05*(trial_time[trial_length]/1000-5));
FI[fi_count] = 100*FI[fi_count];

// update screen
sprintf(strBuffer, " %f", FI[fi_count]);
switch (fi_count) {
     case 0:
          Form3->Edit4->Text = strBuffer;
          Form3->Edit9->Text = IntToStr(fi_start_time[fi_count]);
          break;
     case 1:
          Form3->Edit5->Text = strBuffer;
          Form3->Edit10->Text = IntToStr(fi_start_time[fi_count]);
          break;
     case 2:
          Form3->Edit8->Text = strBuffer;
          Form3->Edit11->Text = IntToStr(fi_start_time[fi_count]);
          break;
}

++fi_count;
update_fi_radio();
```

```
     Form3->Button2->Enabled=false;
     Form3->Button3->Enabled=false;
     switch (fi_count) {
          case 1:
               Form3->Button4->Enabled=true;
               break;
          case 2:
               find_alpha();
               Form3->Button4->Enabled=true;
               Form3->Button5->Enabled=true;
               break;
          case 3:
               find_beta();
               Form3->Button5->Enabled=true;
               break;
     }
}
//---------------------------------------------------------------------------

// this function controls helps the user with which FI to calculate
void update_fi_radio(void)
{
     switch (fi_count) {
          case 0:
               Form3->RadioButton1->Checked=true;
               Form3->RadioButton2->Checked=false;
               Form3->RadioButton3->Checked=false;
               break;
          case 1:
               Form3->RadioButton1->Checked=false;
               Form3->RadioButton2->Checked=true;
               Form3->RadioButton3->Checked=false;
               break;
          case 2:
               Form3->RadioButton1->Checked=false;
               Form3->RadioButton2->Checked=false;
               Form3->RadioButton3->Checked=true;
               break;
          case 3:
               Form3->RadioButton1->Checked=false;
               Form3->RadioButton2->Checked=false;
               Form3->RadioButton3->Checked=false;
               Form3->Button1->Enabled = false;
               Form3->Button2->Enabled = false;
               Form3->Button3->Enabled = false;
               break;
     }
}

void __fastcall TForm3::RadioButton1Click(TObject *Sender)
{
     fi_count = 0;
     update_fi_radio();
     Form3->Button1->Enabled=true;
}
//---------------------------------------------------------------------------
```

```
void __fastcall TForm3::RadioButton2Click(TObject *Sender)
{
     fi_count = 1;
     update_fi_radio();
     Form3->Button1->Enabled=true;
     Form3->Button5->Enabled=false;
}
//---------------------------------------------------------------------------

void __fastcall TForm3::RadioButton3Click(TObject *Sender)
{
     fi_count = 2;
     update_fi_radio();
     Form3->Button1->Enabled=true;
     Form3->Button5->Enabled=false;
}
//---------------------------------------------------------------------------

// click to next step goes to bias axes collection based on where we are
void __fastcall TForm3::Button4Click(TObject *Sender)
{
     switch (fi_count) {
          case 1:
               Form3->Visible=false;
               Form2->Visible=true;
               break;
          case 2:
               Form3->Visible=false;
               //Form2->Visible=true;
               Form5->Visible=true;
               break;
     }
}
//---------------------------------------------------------------------------

//this function finds alpha
void find_alpha()
{
     float p;
     char strBuffer[9];
     p = (FI[1]-FI_MIN)/(FI[0]-FI_MIN);
     //if user is within less than 1% of FI_MIN, set p to 1%
     // we don't want the gain to increase too quickly
     if (p < 0.01) p = 0.01;
     alpha = -1.0/(fi_start_time[1]-fi_start_time[0])*log(p);
     sprintf(strBuffer, "%.5e", alpha);
     Form3->Edit6->Text=strBuffer;
}

//this function finds beta
void find_beta()
{
     float p;
     char strBuffer[9];
     p = (FI[2]-FI_MIN)/(FI[0]-FI_MIN);
     //if user is within more than 99% of FI[0], set p to 99%
```

```
    // we don't want the recovery rate to be too quick
    if (p > 0.99) p = 0.99;
    beta = -1.0/(fi_start_time[2]-fi_start_time[1])*log(1.0-p);
    sprintf(strBuffer, "%.5e", beta);
    Form3->Edit7->Text=strBuffer;
}

//upon finishing the last fatigue index test, write data to files and close
// forms
void __fastcall TForm3::Button5Click(TObject *Sender)
{
    // local variables
    char FileName[50];
    const char * dir = "c:\\Settings\\";
    FILE *fp, *ffdata, *fbdata, *fldata, *frdata, *fp_fi;
    int biaaxis, i, j;
    int gain_status = 1;   //not sure what the condition should be

    // write setup file for Virtual Driving
    fp=fopen("c:\\Settings\\Setup.txt","w");
    if (shape==3) shape=1;
    if (index==4) index=1;
    fprintf(fp, "%s\tSubject identifier\n", subject_id);
    fprintf(fp, "%d\tDead zone shape: 0 = no dead zone; 1 = ellipse; 2 =
        rectangle\n", shape);
    fprintf(fp, "%d\tDead zone x-axis: a number from 0 to 500 (ignored if DZ
        shape = zero)\n", MaxDZDirection);
    fprintf(fp, "%d\tDead zone y-axis: a number from 0 to 500 (ignored if DZ
        shape = zero)\n", MaxDZForce);
    fprintf(fp, "%d\tTemplate shape: 0 = no template; 1 = ellipse; 2 =
        asteroid; 3 = diamond\n", index);
    fprintf(fp, "%d\tTemplate x-axis: 100 to 2048 (ignored if template shape
        = zero)\n", Form2->TrackBar1->Position*2048/250);
    fprintf(fp, "%d\tTemplate y-axis: 100 to 2048 (ignored if template shape
        = zero)\n", Form2->TrackBar2->Position*2048/250);
    if ( (Biasangle*180/PI)<=10 || (Biasangle*180/PI)>=350)
        biaaxis=0;
    else
        biaaxis=1;
    fprintf(fp, "%d\tBias axis status: 0 = none; 1 = active\n", biaaxis);
    fprintf(fp, "%f\tBias axis angle: 0 to 360 degrees (ignored if biax axis
        status = 0)\n", Biasangle*180/PI);
    fprintf(fp, "%d\tGain status: 0 = off; 1 = on\n", gain_status);
    fprintf(fp, "%f\tGain -X: 1 to 20 (if no or overlimit, 1X will be
        used)\n", XGain[0]);
    fprintf(fp, "%f\tGain +X: 1 to 20 (if no or overlimit, 1X will be
        used)\n", XGain[1]);
    fprintf(fp, "%f\tGain -Y: 1 to 20 (if no or overlimit, 1X will be
        used)\n", YGain[0]);
    fprintf(fp, "%f\tGain +Y: 1 to 20 (if no or overlimit, 1X will be
        used)\n", YGain[1]);
    fprintf(fp, "%f\tMax Gain -X: 1 to 20 (if no or overlimit, 1X will be
        used)\n", XGain_max[0]);
    fprintf(fp, "%f\tMax Gain +X: 1 to 20 (if no or overlimit, 1X will be
        used)\n", XGain_max[1]);
    fprintf(fp, "%f\tMax Gain -Y: 1 to 20 (if no or overlimit, 1X will be
        used)\n", YGain_max[0]);
```

```
    fprintf(fp, "%f\tMax Gain +Y: 1 to 20 (if no or overlimit, 1X will be
        used)\n", YGain_max[1]);
    fprintf(fp, "%f\tGain Multiplier: 1, 10/9, 5/4, 10/7, 5/3, 2\n",
        gain_multiplier);
    fprintf(fp, "%e\talpha: 0 to ~1\n", alpha);
    fprintf(fp, "%e\tbeta: 0 to ~1\n", beta);
    fclose(fp);

    // write fatigue index data file
//  char strBuffer[9];
//  sprintf(strBuffer, "fi_test_data_%2d.txt", fi_count);
//  fp = fopen(strBuffer, "w");
    strcpy(FileName, dir);
    strcat(FileName, "fi_test_data");
//  strcat(FileName, subject_id);
    strcat(FileName, ".txt");
    fp_fi = fopen(FileName,"w");
    for (j=0; j<fi_count; j++) {
        fprintf(fp_fi, "t_start%1.1i\t%d\t", j, fi_start_time[j]);
    }
    fprintf(fp_fi,"\n");
    for (j=0; j<fi_count; j++) {
        fprintf(fp_fi, "FI_%1.1i\t%f\t", j, FI[j]);
    }
    fprintf(fp_fi,"\n");
    for (j=0; j<fi_count; j++) {
        fprintf(fp_fi, "time %1.1i\tinput_mag %1.1i\t", j, j);
    }
    fprintf(fp_fi,"\n");
    for (i=0; i<count; i++) {
        for (j=0; j<fi_count; j++) {
            fprintf(fp_fi, "%d\t%f\t", data_time[i][j], input_mag[i][j]);
        }
        fprintf(fp_fi, "\n");
    }
    fclose(fp_fi);

    // close forms
    Form1->Close();
    Form2->Close();
    Form3->Close();
    Form5->Close();

    // reset windows timer
    timeEndPeriod(1);

}
//---------------------------------------------------------------------------


void __fastcall TForm3::Button6Click(TObject *Sender)
{
    int iStatus;
    iStatus = ICD_find_offset();
}
//---------------------------------------------------------------------------
```

```
void __fastcall TForm3::Edit6Change(TObject *Sender)
{
    alpha = StrToFloat(Form3->Edit6->Text);
}
//---------------------------------------------------------------------------

void __fastcall TForm3::Edit7Change(TObject *Sender)
{
    beta = StrToFloat(Form3->Edit7->Text);
}
//---------------------------------------------------------------------------
```

I.8 UNIT4.H

```
//---------------------------------------------------------------------------

#ifndef Unit4H
#define Unit4H
//---------------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include "CGAUGES.h"
//---------------------------------------------------------------------------
class TForm4 : public TForm
{
    __published:  // IDE-managed Components
        TCGauge *CGauge1;
        TCGauge *CGauge2;
        void __fastcall Form4_SetGauges(TObject *Sender);
    private: // User declarations
    public:  // User declarations
        __fastcall TForm4(TComponent* Owner);
};
//---------------------------------------------------------------------------
extern PACKAGE TForm4 *Form4;
//---------------------------------------------------------------------------
#endif
```

I.9 UNIT4.CPP

```
/*****************************************************************************

     Version: 1.0
     Modified: 6/27/06
     By: kwb

     Unit4.cpp

     >> PURPOSE <<
     This code demonstrates that the user is applying a force to the joystick
     but hides the magnitude and direction.

     >> NOMENCLATURE <<
     do i want to include nomenclature???

     >> HISTORY <<
     Version 1.0       kwb      27 june 2006
     original version.

*****************************************************************************/
//---------------------------------------------------------------------------

#include <vcl.h>
#pragma hdrstop

#include "Unit4.h"

//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma link "CGAUGES"
#pragma resource "*.dfm"

TForm4 *Form4;

extern int trial_direction;

//---------------------------------------------------------------------------
__fastcall TForm4::TForm4(TComponent* Owner)
     : TForm(Owner)
{
}
//---------------------------------------------------------------------------
void __fastcall TForm4::Form4_SetGauges(TObject *Sender)
{
     switch (trial_direction) {
          case 0:   //forwards
               Form4->CGauge1->Visible=false;
               Form4->CGauge2->Visible=true;
               Form4->CGauge2->ForeColor=clGreen;
               Form4->CGauge2->BackColor=clWhite;
               break;
          case 1:   //backwards
               Form4->CGauge1->Visible=false;
```

```
            Form4->CGauge2->Visible=true;
            Form4->CGauge2->ForeColor=clWhite;
            Form4->CGauge2->BackColor=clGreen;
            break;
        case 2:   //left
            Form4->CGauge1->Visible=true;
            Form4->CGauge1->Enabled=true;
            Form4->CGauge1->ForeColor=clWhite;
            Form4->CGauge1->BackColor=clGreen;
            Form4->CGauge1->Progress=0;
            Form4->CGauge2->Visible=false;
            break;
        case 3:   //right
            Form4->CGauge1->Visible=true;
            Form4->CGauge1->Enabled=true;
            Form4->CGauge1->ForeColor=clGreen;
            Form4->CGauge1->BackColor=clWhite;
            Form4->CGauge1->Progress=0;
            Form4->CGauge2->Visible=false;
            break;
    }
}
//---------------------------------------------------------------------------
```

I.10 UNIT5.H

```
//---------------------------------------------------------------------------

#ifndef Unit5H
#define Unit5H
//---------------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include "CGAUGES.h"
#include <ExtCtrls.hpp>
//---------------------------------------------------------------------------
class TForm5 : public TForm
{
__published:  // IDE-managed Components
    TCGauge *CGauge1;
    TPanel *Panel2;
    TRadioButton *RadioButton3;
    TRadioButton *RadioButton4;
    TRadioButton *RadioButton5;
    TRadioButton *RadioButton6;
    TLabel *Label1;
    TLabel *Label4;
    TLabel *Label5;
    TLabel *Label6;
    TLabel *Label2;
```

```
    TEdit *Edit1;
    TLabel *Label3;
    TEdit *Edit2;
    TButton *Button1;
    TRadioButton *RadioButton7;
    TPanel *Panel1;
    TLabel *Label7;
    TRadioButton *RadioButton1;
    TRadioButton *RadioButton2;
    TEdit *Edit3;
    TEdit *Edit4;
    TButton *Button2;
    TEdit *Edit5;
    TShape *Shape1;
    TShape *Shape2;
    TTimer *Timer2;
    TButton *Button3;
    TEdit *Edit6;
    TEdit *Edit7;
    TEdit *Edit8;
    TEdit *Edit9;
    TRadioButton *RadioButton8;
    TRadioButton *RadioButton9;
    TLabel *Label8;
    TLabel *Label9;
    TRadioButton *RadioButton10;
    void __fastcall RadioButton1Click(TObject *Sender);
    void __fastcall RadioButton2Click(TObject *Sender);
    void __fastcall RadioButton3Click(TObject *Sender);
    void __fastcall RadioButton4Click(TObject *Sender);
    void __fastcall RadioButton5Click(TObject *Sender);
    void __fastcall RadioButton6Click(TObject *Sender);
    void __fastcall RadioButton7Click(TObject *Sender);
    void __fastcall RadioButton9Click(TObject *Sender);
    void __fastcall RadioButton8Click(TObject *Sender);
    void __fastcall Button1Click(TObject *Sender);
    void __fastcall Button2Click(TObject *Sender);
    void __fastcall FormActivate(TObject *Sender);
    void __fastcall Timer2Timer(TObject *Sender);
    void __fastcall Button3Click(TObject *Sender);
    void __fastcall RadioButton10Click(TObject *Sender);
private: // User declarations

public:      // User declarations
    __fastcall TForm5(TComponent* Owner);
};
//---------------------------------------------------------------------------
extern PACKAGE TForm5 *Form5;
//---------------------------------------------------------------------------
#endif
```

I.11 UNIT5.CPP

```
/*****************************************************************************

    Version: 1.0
    Modified: 11/28/06
    By: kwb

    Unit5.cpp

    >> PURPOSE <<
    This code lets the subject determine what the gain should be if fatigued.
    The subject is "taught" what the different settings feel like by pressing
    on the joystick such that the force gauge turns green. This is the
    equivalent amount of force required to cause the wheelchair to drive at
    full speed.

    >> NOMENCLATURE <<
    do i want to include nomenclature???

    >> HISTORY <<
    Version 1.0       kwb      29 june 2006
    original version
    11-28-06:   increase time segments from 30 seconds to 45 seconds

*****************************************************************************/
//---------------------------------------------------------------------------

#include <vcl.h>
#pragma hdrstop

#include <dos.h>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <windows.h>
#include <mmsystem.h>

//#include "Unit1.h"
#include "Unit2.h"
#include "Unit3.h"
#include "Unit5.h"
#include "ICD.h"

//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma link "CGAUGES"
#pragma resource "*.dfm"

TForm5 *Form5;

//function prototypes
void update_gain(void);

//global/extern variables
```

```
extern js_types read_type;//serial or DAQcard. defined in Tuning_MS_Study.cpp
extern float Biasangle;
extern float XGain[2], YGain[2];
float XGain_max[2], YGain_max[2];
float gain_multiplier;

//unit variables
int trial_length = 0; //1=long, 0=short; Tuning_MS_Study.cpp
enum possible_axes {left, back, right, forward} current_axis;
int direction, speed;
bool enable_data_collection = true;
DWORD trial_time[2] = {10*1000, 45*1000}; // sec -> msec
DWORD dt;

//serial variables
HANDLE hComm=NULL;
COMMTIMEOUTS ctmoNew={0},ctmoOld;
DWORD dwEvent,dwError;
COMSTAT cs;
DCB dcbCommPort;

//---------------------------------------------------------------------------
__fastcall TForm5::TForm5(TComponent* Owner)
    : TForm(Owner)
{
}
//---------------------------------------------------------------------------

//this should update the default gains whenever the form is made visible
void __fastcall TForm5::FormActivate(TObject *Sender)
{
    gain_multiplier = 1.0;
    current_axis = forward;
    direction = 2048;
    speed = 2048;

    char strBuffer[9];
    sprintf(strBuffer, "%.4f", XGain[0]);
    Form5->Edit1->Text=strBuffer;
    sprintf(strBuffer, "%.4f", YGain[0]);
    Form5->Edit2->Text=strBuffer;
    sprintf(strBuffer, "%.4f", XGain[1]);
    Form5->Edit6->Text=strBuffer;
    sprintf(strBuffer, "%.4f", YGain[1]);
    Form5->Edit7->Text=strBuffer;
    if (trial_length)
        Form5->Edit5->Visible=false;
    update_gain();
}
//---------------------------------------------------------------------------

void __fastcall TForm5::Button2Click(TObject *Sender)
{
    // Local Variables
    int iStatus;
    DWORD start_time, end_time;
    char strBuffer[9];
```

```
     int ab1, ab2;

     // Device Readings
     float tmp_ICD_x_reading = 0.0;
     float tmp_ICD_y_reading = 0.0;
     unsigned char InBuff[10];
     DWORD dwBytesRead;

     // Set buttons and timer
     Form5->Button1->Enabled=false;
     Form5->CGauge1->Enabled=true;
     Form5->Timer2->Enabled=true;

     start_time = timeGetTime();
     //   MessageBox(NULL, "Goodbye, cruel world!", "Note", MB_OK);

     if (read_type == serial) {
          /*  Serial Communication */
          hComm=CreateFile("COM1",GENERIC_READ|GENERIC_WRITE,0,NULL,OPEN_EXIST
               ING,0,NULL);
          GetCommState(hComm,&dcbCommPort);
          BuildCommDCB("38400,N,8,1", &dcbCommPort);
          SetCommState(hComm,&dcbCommPort);
          SetCommMask(hComm,EV_RXCHAR);
          GetCommTimeouts(hComm,&ctmoNew);
          ctmoNew.ReadIntervalTimeout=2.7;
          ctmoNew.ReadTotalTimeoutConstant=0;
          ctmoNew.ReadTotalTimeoutMultiplier=0;
          SetCommTimeouts(hComm,&ctmoNew);
     }

     while(1)
     {
          /* Serial Comm */
          if (read_type == serial)
               ClearCommError(hComm,&dwError,&cs);

          Application->ProcessMessages();

          switch (read_type) {
               case DAQcard:
                    iStatus = ICD_get_data(&tmp_ICD_x_reading,
&tmp_ICD_y_reading);
                    if (DEBUG_lo)
                         printf("iStatus, ICD_get_data = %d\n", iStatus);
                    direction = tmp_ICD_x_reading + 2048;
                    speed = tmp_ICD_y_reading + 2048;
                    break;
               case serial:
                    while(1)
                    {
                         if (WaitCommEvent(hComm,&dwEvent,NULL))
                         {
                              if (dwEvent & EV_RXCHAR)
                              {
                                   ClearCommError(hComm,&dwError,&cs);
                                   ReadFile(hComm,InBuff,4,&dwBytesRead,NULL);
```

```
                                ClearCommError(hComm,&dwError,&cs);

                                ab1=(unsigned int)(unsigned char)(InBuff[0]);
                                if ((ab1&240)!=16) break;

                                ab2=(unsigned int)(unsigned char)(InBuff[2]);
                                if ((ab2&240)!=32) break;

                                direction=(ab1&15)*256 +
                                     (unsigned int)(unsigned char)(InBuff[1]);
                                speed=(ab2&15)*256 +
                                     (unsigned int)(unsigned char)(InBuff[3]);

                                break;
                           }
                      }
                 }
                 break;
          }
          end_time = timeGetTime();
          dt = end_time - start_time;
          if (dt >= trial_time[trial_length]) break;
     }

     Form5->Timer2->Enabled=false;
     Form5->Button1->Enabled=true;

     if (read_type == serial) {
          //clean up serial
          SetCommMask(hComm,0);
          PurgeComm(hComm,PURGE_RXABORT);
          SetCommTimeouts(hComm,&ctmoOld);
          CloseHandle(hComm);
     }

}
//---------------------------------------------------------------------------


//this chunk of code updates the gain based on which radio button is checked
void __fastcall TForm5::RadioButton7Click(TObject *Sender)
{
     Form5->RadioButton7->Checked=true;
     gain_multiplier = 1;
     update_gain();
}
//---------------------------------------------------------------------------
void __fastcall TForm5::RadioButton3Click(TObject *Sender)
{
     Form5->RadioButton3->Checked=true;
     gain_multiplier = 10.0/9.0;
     update_gain();
}
//---------------------------------------------------------------------------
void __fastcall TForm5::RadioButton4Click(TObject *Sender)
{
     Form5->RadioButton4->Checked=true;
```

```
    gain_multiplier = 10.0/8.0;
    update_gain();
}
//---------------------------------------------------------------------------
void __fastcall TForm5::RadioButton5Click(TObject *Sender)
{
    Form5->RadioButton5->Checked=true;
    gain_multiplier = 10.0/7.0;
    update_gain();
}
//---------------------------------------------------------------------------
void __fastcall TForm5::RadioButton6Click(TObject *Sender)
{
    Form5->RadioButton6->Checked=true;
    gain_multiplier = 10.0/6.0;
    update_gain();
}
//---------------------------------------------------------------------------
void __fastcall TForm5::RadioButton10Click(TObject *Sender)
{
    Form5->RadioButton10->Checked=true;
    gain_multiplier = 10.0/5.0;
    update_gain();
}
//---------------------------------------------------------------------------

//this chunk of code updates the direction
void __fastcall TForm5::RadioButton1Click(TObject *Sender)
{
    Form5->RadioButton1->Checked=true;
    current_axis = left;
    update_gain();
}
//---------------------------------------------------------------------------
void __fastcall TForm5::RadioButton2Click(TObject *Sender)
{
    Form5->RadioButton2->Checked=true;
    current_axis = back;
    update_gain();
}
//---------------------------------------------------------------------------
void __fastcall TForm5::RadioButton9Click(TObject *Sender)
{
    Form5->RadioButton9->Checked=true;
    current_axis = forward;
    update_gain();
}
//---------------------------------------------------------------------------
void __fastcall TForm5::RadioButton8Click(TObject *Sender)
{
    Form5->RadioButton8->Checked=true;
    current_axis = right;
    update_gain();
}
//---------------------------------------------------------------------------

//this function updates the gain based on which radio buttons are selected
```

```
void update_gain(void)
{
     char strBuffer[9];
     int i;

     // calculate new gains
     for (i=0; i<=1; i++) {
          XGain_max[i] = gain_multiplier*XGain[i];
          YGain_max[i] = gain_multiplier*YGain[i];
     }

     // display the data
     sprintf(strBuffer, "%.4f", XGain_max[0]);
     Form5->Edit3->Text = strBuffer;
     sprintf(strBuffer, "%.4f", YGain_max[0]);
     Form5->Edit4->Text = strBuffer;
     sprintf(strBuffer, "%.4f", XGain_max[1]);
     Form5->Edit8->Text = strBuffer;
     sprintf(strBuffer, "%.4f", YGain_max[1]);
     Form5->Edit9->Text = strBuffer;
}


//This function takes joystick input, applies gain, determines which axis to
// be displayed, and updates the gauge accordingly
void __fastcall TForm5::Timer2Timer(TObject *Sender)
{
     //local vars
     float tmp_dir, dir, spd;
     float prog;

     //Offset axes and rotate
     tmp_dir = direction - 2048;
     spd = speed - 2048;
     dir = cos(Biasangle)*tmp_dir - sin(Biasangle)*spd;
     spd = sin(Biasangle)*tmp_dir + cos(Biasangle)*spd;
//   dir = tmp_dir;

     //based on clinician settings, update gauge
     switch (current_axis) {
          case left:
               prog = 75*(-XGain_max[0]*dir)/2048.0;
               break;
          case back:
               prog = 75*(-YGain_max[0]*spd)/2048.0;
               break;
          case right:
               prog = 75*(XGain_max[1]*dir)/2048.0;
               break;
          case forward:
               prog = 75*(YGain_max[1]*spd)/2048.0;
               break;
     }

     if ((prog < 65) || (prog > 85))
          Form5->CGauge1->ForeColor=clBlack;
     else
```

```
          Form5->CGauge1->ForeColor=clGreen;

     Form5->CGauge1->Progress = (long)prog;

     if (!trial_length)
     {
          char strBuffer[25];
          sprintf(strBuffer, "d:%.3f s:%.3f p:%.2f", dir, spd, prog);
          Form5->Edit5->Text = strBuffer;
     }

}
//---------------------------------------------------------------------------

void __fastcall TForm5::Button1Click(TObject *Sender)
{
     Form5->Visible=false;
     Form3->Visible=true;
}
//---------------------------------------------------------------------------


void __fastcall TForm5::Button3Click(TObject *Sender)
{
     int iStatus;
     iStatus = ICD_find_offset();
}
//---------------------------------------------------------------------------
```

I.12 ICD.H

```
/*************************************************************************
     version: 1.0
     modified: 7/2/06
     by: kwb

     >> description <<
     this is the header file for functions used to process user input signals

     >> history <<
     Version 1.0       kwb      21 may 2006
     Original version. based on version 1.0 of VCJ_ICD characterize. removed
     fatigue parameters. added js_types enum

*************************************************************************/

#ifndef ICD_H
#define ICD_H

/*** structures ***/
typedef struct {
     float dir_volts;
```

```
    float spd_volts;
    int dir_dig;
    int spd_dig;

} ICD_input;

/*** enums ***/
enum js_types {
    serial,
    DAQcard
};

/*** prototypes ***/
int ICD_initialize(void);  /* i don't think i'll need inputs */
int ICD_find_offset(void);
int ICD_get_data(float *, float *); /* pass by address direction and speed */
int ICD_VGA(float, float);
int ICD_calibration_constants(void);
int ICD_MS_PFA(char, float, float);
int ICD_close_device();

/*** globals ***/
/* address values for input and output channels */
#define ACH0      0    /* direction */
#define ACH1      1    /* speed */
#define DAC0OUT       0    /* direction */
#define DAC1OUT       1    /* speed */
#define DAQ_DEVICE    1

/* debugging characteristics */
#define DEBUG_ALL 0
#define DEBUG_hi  1    /* hi-level debugging */
#define DEBUG_lo  0    /* lo-level debugging */

#endif
```

I.13 ICD_FUNCTIONS.CPP

```
/*****************************************************************************

    Version: 1.0
    Modified: 11/28/06
    By: kwb

    ICD_functions.cpp

    >> PURPOSE <<
    This code acts as the input control device (ICD) for the variable
    compliance joystick. Its purposes are to
        * initialize the NI-DAQcard,
        * read analog inputs,
        * send outputs [digital signal to software or analog outputs in
```

```
          place of a movement sensing joystick].
     The flow of the program is modeled after the suggested flow charts in
     DAQ: Traditional NI-DAQ (Legacy) User Manual: Version 7.x, pgs 3-25++.

     >> NOMENCLATURE <<
     do i want to include nomenclature???

     >> HISTORY <<
     Version 1.0       kwb      19 june 2006
     original version. based on version 1.0 of VCJ_ICD calibrate. make
     compatible with borland c++. remove algorithm functions.
     11-28-06:   update scan rate and sample size for optimal readings

*************************************************************************/

/* Include files */
#include <stdio.h>
#include <sys/timeb.h>
#include <time.h>
#include <math.h>

#include "nidaqex.h"
#include "nidaq.h"
#include "ICD.h"

/* Global/Extern variables */
float X_Offset;            /* x-offset value */
float Y_Offset;            /* y-offset value */

/***** FOR REFERENCE ONLY *****/
// these are the parameters the calibration s/w should output
int Bias_Axis_Enabled_Status;   /* 0=not used, 1=used */
int Bias_Axis_Angle;            /* degrees */
int Dead_Zone_Shape_Status;     /* 1=eliptical, 2=rectangular */
int Dead_Zone_Rad_X;
int Dead_Zone_Rad_Y;
float Gain_X;
float Gain_Y;
int Template_Shape_Status; /* 1=eliptical, 2=rectangular */
int Template_Rad_X;        /* 3=astroid, 4=diamond */
int Template_Rad_Y;        /* post gain values */
float Gain_max[2];         /* maximum gain, [0]=dir, [1]=spd */
float Gain_min[2];         /* mininmum (baseline) gain, [0]=dir, [1]=spd */
float Alpha[2];            /* rate of increase while active,[0]=dir,[1]=spd*/
float Beta[2];             /* recovery rate while inactive,[0]=dir,[1]=spd */

/*
* PURPOSE:
*    this function initializes the NI-DAQcard.
*/
int ICD_initialize(void)
{
    /***** DECLARATIONS *****/

     /* Local Variables */

     /* Device Settings */
```

```
    i16 iInputMode = 0;    /* 1 for RSE, 0 for DIFF */
    i16 iInputRange = 0;   /* ignored */
    i16 iPolarity = 0;     /* 0=bipolar, 1=unipolar */
    i16 iDriveAIS = 1;     /* ignored */
    i32 lTimeout = 90;

    /* Debugging */
    i16 iStatus = 0;
   i16 iRetVal = 0;
   i16 iIgnoreWarning = 0;
    i16 iStatusReturn = 0;

    /***** CONFIGURE INPUT SETTINGS *****/
    /* this needs to be done only once */

    /* channel 0 */
    iStatus = AI_Configure(DAQ_DEVICE, ACH0, iInputMode, iInputRange,
        iPolarity, iDriveAIS);
    if (iStatus) iStatusReturn = iStatus;
    iRetVal = NIDAQErrorHandler(iStatus, "AI_Configure", iIgnoreWarning);
    if (DEBUG_lo) printf("iStatus, AI_Configure channel %d = %d\n", ACH0,
        iStatus);
    /* channel 1 */
    iStatus = AI_Configure(DAQ_DEVICE, ACH1, iInputMode, iInputRange,
        iPolarity, iDriveAIS);
    if (iStatus) iStatusReturn = iStatus;
    iRetVal = NIDAQErrorHandler(iStatus, "AI_Configure", iIgnoreWarning);
    if (DEBUG_lo) printf("iStatus, AI_Configure channel %d = %d\n", ACH1,
        iStatus);

    /* timeout */
   /* This sets a timeout limit (#Sec * 18ticks/Sec) so that if there
      is something wrong, the program won't hang on the SCAN_Op call. */
    iStatus = Timeout_Config(DAQ_DEVICE, lTimeout);
    if (iStatus) iStatusReturn = iStatus;
   iRetVal = NIDAQErrorHandler(iStatus, "Timeout_Config", iIgnoreWarning);
    if (DEBUG_lo) printf("iStatus, Timeout_Config = %d\n", iStatus);

    /* initialize offset (probably not needed, but just in case) */
    X_Offset = 0.0;
    Y_Offset = 0.0;

    if (DEBUG_hi)
        printf("\nThe time for opening the ICD is %d\n", timeGetTime());

    return(iStatusReturn);
}

/*
* PURPOSE:
*   this function finds the offset so that zero input results in zero output
*/
int ICD_find_offset(void) {

    /***** DECLARATIONS *****/
    /* Local Variables */
    int i;
```

```
/* Device Settings */
i16  numChan = 2;      /* number of channels to read */
i16 iReadings = 100;   /* number of readings per channel */
static i16 iGain[2] = {10, 10};     /* input gain */
static i16 iChan[2] = {ACH0, ACH1}; /* which channel(s) to read */
i16 iCount = numChan*iReadings;     /* number of readings in one go */
f64 iSampleRate = 25000;/* i think the max for the DAQcard is 200000 */
f64  iScanRate = 0.0;  /* this should get the maximum scanning rate */

/* Device Readings */
static i16 AI_buffer[200] = {0};
static i16 x_buffer[100] = {0};
static i16 y_buffer[100] = {0};
double loc_x_reading, loc_y_reading;/* local copies of the readings */

/* Debugging */
i16 iStatus = 0;
i16 iRetVal = 0;
i16 iIgnoreWarning = 0;
i16 iStatusReturn = 0;

/***** START SCAN *****/
/* high level function scans selected channel(s) */

/* read the channels specified by iChan */
iStatus = SCAN_Op(DAQ_DEVICE, numChan, iChan, iGain, AI_buffer,
     iCount, iSampleRate, iScanRate);
if (iStatus) iStatusReturn = iStatus;
iRetVal = NIDAQErrorHandler(iStatus, "SCAN_op", iIgnoreWarning);
if (DEBUG_lo) printf("iStatus, SCAN_op = %d\n", iStatus);

/***** CHECKING *****/
/* not needed if we do high level function scans */

/***** SIGNAL PROCESSING *****/
/* this isn't in the manual, but this is needed so that there is only one
   output from the function */

/* demuliplex the AI_buffer */
iStatus = SCAN_Demux(AI_buffer, iCount, numChan, 0);
if (iStatus) iStatusReturn = iStatus;
iRetVal = NIDAQErrorHandler(iStatus, "SCAN_Demux", iIgnoreWarning);
if (DEBUG_lo) printf("iStatus, SCAN_Demux = %d\n", iStatus);

for (i=0; i<iReadings; i++) {
     x_buffer[i] = AI_buffer[ACH0*iReadings + i];
     y_buffer[i] = AI_buffer[ACH1*iReadings + i];
}

/* take means to mitigate the noise, hopefully */
iRetVal = NIDAQMean(x_buffer, iReadings, WFM_DATA_I16, &loc_x_reading);
iRetVal = NIDAQMean(y_buffer, iReadings, WFM_DATA_I16, &loc_y_reading);

if (DEBUG_lo) {
     /* display results */
     printf("\nx-reading in ICD_get_data: %lf\n", loc_x_reading);
```

```
        printf("y-reading in ICD_get_data: %lf\n", loc_y_reading);
    }

    /* Output Result */
    X_Offset = (float)loc_x_reading;
    Y_Offset = (float)loc_y_reading;

    return(iStatusReturn);
}


/*
* PURPOSE:
*   this function will read the data from the NI-DAQcard
*
* OUTPUT:
*   x_reading         dithered direction
*   y_reading         dithered speed
*
*/
int ICD_get_data(float *x_reading, float *y_reading) {

    /***** DECLARATIONS *****/
    /* Local Variables */
    int i;                      /* for loop counter */

    /* Device Settings */
    i16  numChan =     2;       /* number of channels to read */
    i16  iReadings =   40;      /* number of readings per channel */
    static i16    iGain[2] =    {10, 10};     /* input gain */
    static i16    iChan[2] =    {ACH0, ACH1}; /* which channel(s) to read */
    i16  iCount = numChan*iReadings;     /* number of readings in one go */
    f64  iSampleRate = 75000; /* i think the max for the DAQcard is 200000 */
    f64  iScanRate =   0.0; /* this should get the maximum scanning rate */
    f64  dGainAdjust = 1.0;
    f64  dOffset =     0.0;

    /* Device Readings */
    static i16 AI_buffer[100] = {0};
    static i16 ICD_x_data[50] = {0};
    static i16 ICD_y_data[50] = {0};
    double loc_x_reading, loc_y_reading; /* local copies of the readings */

    /* Debugging */
    i16 iStatus = 0;
    i16 iRetVal = 0;
    i16 iIgnoreWarning = 0;
    i16 iStatusReturn = 0;

    /***** START SCAN *****/
    /* high level function scans selected channel(s) */

    /* read the channels specified by iChan iCount number of times */
    iStatus = SCAN_Op(DAQ_DEVICE, numChan, iChan, iGain, AI_buffer,
        iCount, iSampleRate, iScanRate);
    if (iStatus) iStatusReturn = iStatus;
    iRetVal = NIDAQErrorHandler(iStatus, "SCAN_op", iIgnoreWarning);
```

```
    if (DEBUG_lo) printf("iStatus, SCAN_op = %d\n", iStatus);

    /***** CHECKING *****/
    /* not needed if we do high level function scans */


    /***** SIGNAL PROCESSING *****/
    /* this isn't in the manual, but this is needed so that there is only one
       output from the function */

    /* demuliplex the AI_buffer */
    iStatus = SCAN_Demux(AI_buffer, iCount, numChan, 0);
    if (iStatus) iStatusReturn = iStatus;
    iRetVal = NIDAQErrorHandler(iStatus, "SCAN_Demux", iIgnoreWarning);
    if (DEBUG_lo) printf("iStatus, SCAN_Demux = %d\n", iStatus);

    for (i=0; i<iReadings; i++) {
        ICD_x_data[i] = AI_buffer[ACH0*iReadings + i];
        ICD_y_data[i] = AI_buffer[ACH1*iReadings + i];
    }

    /* take means to mitigate the noise, hopefully */
    iRetVal = NIDAQMean(ICD_x_data, iReadings, WFM_DATA_I16,
        &loc_x_reading);
    iRetVal = NIDAQMean(ICD_y_data, iReadings, WFM_DATA_I16,
        &loc_y_reading);

    /* adjust for offset */
    loc_x_reading = loc_x_reading - X_Offset;
    loc_y_reading = loc_y_reading - Y_Offset;

    if (DEBUG_lo) {
        /* display results */
        printf("\nx-reading in ICD_get_data: %lf\n", loc_x_reading);
        printf("y-reading in ICD_get_data: %lf\n", loc_y_reading);
    }

    /* return results */
    *x_reading = (float)loc_x_reading;
    *y_reading = (float)loc_y_reading;
    return(iStatusReturn);
}


/* this function closes the NI-DAQcard */
int ICD_close_device(void) {

    /***** DECLARATIONS *****/
    /* Local Variables */

    /* Debugging */
    i16 iStatus = 0;
    i16 iStatusReturn = 0;

    /***** CLEAN UP *****/
    /* this goes at the very end, i think */
```

```
    /* turn off any DAQ operation */
    /* oddly, DAQ_clear is undefined. so comment it out for the moment
    iStatus = DAQ_clear(DAQ_DEVICE);
    iRetVal = NIDAQErrorHandler(iStatus, "DAQ_clear", iIgnoreWarning);
    if (DEBUG_lo) printf("iStatus, DAQ_clear = %d\n", iStatus);
    */

   /* disable timeouts */
   iStatus = Timeout_Config(DAQ_DEVICE, -1);
    if (iStatus) iStatusReturn = iStatus;
    if (DEBUG_lo) printf("iStatus, Timeout_Config = %d\n", iStatus);

    /* Get and display current time */
    if (DEBUG_hi)
         printf("\nThe time of closing the ICD is %d\n", timeGetTime());

    return(iStatusReturn);
}
```

APPENDIX J

SOURCE CODE IMPLEMENTATION FOR *MSS INPUT ANALYSIS*

J.1 MSS_INPUT_ANALYSIS.M

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   modified: 11/14/06
%   by: kwb
%
%   >> PURPOSE <<
%
%   The goal of this software is to allow the rehab engineer quickly and
%   efficiently analyze joystick data from the first thirty minutes of
%   testing during the MS Study with the Variable Compliance Joystick.
%   Power Spectral Density and average force information will be presented.
%   Following analysis, the program should write parameters for the WFLC
%   and high-pass filter to a data file. This file is read by the Virtual
%   Driving Demo software.
%
%   >> NOMENCLATURE <<
%
%   speed_thresh    input spd above which indicates trial start
%
%   >> HISTORY <<
%   Version 1.0a        karl brown          28 August 2006
%   original version.  This version looks at trials on an individual basis.
%   Consequently, filter perameters are reset at the beginning of each
%   trial.
%   Version 1.0b        kwb                 13 November 2006
%   String trials together for filter analysis, update for new output file
%   Version 1.1         kwb                 To be released...
%   Begin filter analysis and write to setup file only after user says ok
%   to filter analysis, remove contingency of user input variables on
%   existing to determine if user should be prompted for input
%*************************************************************************/

% function iStatus = MSS_input_analysis(plot_force, plot_psd)

%%% Remove old variables if needed
```

```
clear files tmp_data data psd all_data file_time avg_force;

%%% Initialize variables
% constants
DRIVE_NAME_LENGTH = 21;
HEADER_LINES = 26;
MAKE_MOVIE = 1;
MAKE_3D_PSD = 1;
PLOT_HPF = 0;
% local variables
j = 1;
data_folder = 'C:\SubjectFiles\';
movie_n = 3;
movie_time = 5;    %seconds
out_filename = 'C:\settings\wflc_setup.txt';

%%% Get data from files
% get directory listing
all_files = dir(data_folder);

% get filenames for only the driving data (not WFLC data)
for i = 3:length(all_files)
    if (length(all_files(i).name) == DRIVE_NAME_LENGTH)
        files(j) = all_files(i);
        j = j + 1;
    end
end
for i = 1:length(files)
    files(i).full_name = strcat(data_folder, files(i).name);
end

% let user decide which trials to analyze
[file_id n] = MSS_get_trial_indexes(files);
files = files(file_id);

% determine order in which trials occured
for i = 1:n
    file_time(i,:) = [datenum(files(i).date, 0) i];
end
file_time = sortrows(file_time);
order = file_time(:,2)';

% store data into structures
j = 1;
for i = order
    fp1 = fopen(char(files(i).full_name));
    foo = textscan(fp1, '%s%s%s%s%s%s%s%s%s%s%s%s%s%s', HEADER_LINES);
    tmp_data = textscan(fp1, '%d%d%d%f%f%f%f%d%d%d%d%f%f%f%f%d');
    data(j).t = tmp_data{1,2};
    data(j).dt = tmp_data{1,3};
    data(j).raw_dir = tmp_data{1,4};
    data(j).raw_spd = tmp_data{1,5};
    data(j).raw_mag = sqrt(data(j).raw_dir.^2 + data(j).raw_spd.^2);
    fclose(fp1);
    j = j + 1;
end     %data is now in the order in which it occurred
```

```
        %files is in alphabetical order
        %file_id contains index values of files to original listing

%%% Average force for each episode
% (still need dig2force transformation)
j = 1;
for i = 1:n
    TMP_t = data(i).t;
    TMP_mag = data(i).raw_mag;
    avg_force(i) = trapz(double(TMP_t), TMP_mag) / ...
        (data(i).t(end) - data(i).t(1));
end

%%% PSD of each episode
for i = 1:n
    fs(i) = 1000/(mean(data(i).dt));
    psd(i).pow_dir = DB(periodogram(data(i).raw_dir - ...
        mean(data(i).raw_dir)));
    psd(i).pow_spd = DB(periodogram(data(i).raw_spd - ...
        mean(data(i).raw_spd)));
    psd(i).pow_mag = DB(periodogram(data(i).raw_mag - ...
        mean(data(i).raw_mag)));
    psd(i).freq = fs(i)/2*linspace(0, 1, length(psd(i).pow_dir));
end

%%% Display results
% Average force results for all episodes
plot_force = input('Should average forces be plotted [1=yes;0=no]? ');
if (plot_force)
    figure;
        bar(file_id(order), avg_force);
        title('Average Force for Each Episode');
        xlabel('episode number'); ylabel('average force (dig)');
    set(gcf, 'Name', 'Avg force per ep');
end

% PSD results per episode
plot_psd = input('Should individual episodes be plotted [1=yes;0=no]? ');
if (plot_psd)
    if (MAKE_MOVIE)
        figure;
        for i = 1:n
            plot(psd(i).freq,psd(i).pow_mag);
                title(['PSD Movie']);
                xlabel('freq (Hz)'); ylabel('power');
                axis([0 12 0 150]);
                text(5, 140, [files(order(i)).name ' n=' num2str(i)]);
                text(5, 132, [files(order(i)).date ' i=' ...
                    num2str(file_id(order(i)))]);
            F(i) = getframe;
        end
        plot(0, 0);     % done frame
            title(['PSD Movie']);
            xlabel('freq (Hz)'); ylabel('power');
            axis([0 12 0 150]);
            text(5.5, 90, 'done');
```

```
        F(i+1) = getframe;
        foo = input(['\nPress Enter to play movie. Then press Alt-Tab ' ...
            'to watch...']);
        movie_fps = n/movie_time;
        movie(F, movie_n, movie_fps);
        save psd_movie.mat F;
        fprintf(' movie data saved to psd_movie.mat.\n');
        fprintf([' after loaded, type ''movie(F, %i, %3.2f)'' at the ' ...
            'command prompt\n\n'], movie_n, movie_fps);
    else
%         for i = 1:n
%             figure;
%             plot(psd(i).freq,psd(i).pow_mag);
%                 title(['PSD for ' files(order(i)).name ' at ' ...
%                  files(order(i)).date ' i=' num2str(file_id(order(i)))]);
%                 xlabel('freq (Hz)'); ylabel('power');
%                 axis([0 12 0 150]);
%         end
    end
    if (MAKE_3D_PSD)
%         map = colormap(hot);
        map = colormap(copper);
        [cm cn] = size(map);
%         close(gcf);
        if (n > cm)
            printf(['\n Cannot make 3d plot of frequency spectrum. ' ...
                'Colormap not detailed enough.\n']);
        else
            figure;
            hold on;
            for i = 1:n
                z = ones(1,length(psd(i).freq));
                map_index = (floor(i*cm/n)-1);
                plot3(psd(i).freq,psd(i).pow_mag,i*z, ...
                    'Color',map(map_index,:));
            end
            title(['PSD for all episodes; oldest at bottom']);
                xlabel('freq (Hz)'); ylabel('power');
                axis([0 12 0 150]);
            rotate3d on;
            colormap(map);
            colorbar;
            set(gcf, 'Name', 'PSD all eps');
        end
    end
end

%%% look at 2-minute strength data
stg_analysis = input('Should strength data be analyzed [1=yes;0=no]? ');
if (stg_analysis)

    disp('Entering 2-minute data representation...');
    disp('  data is stored in sdata and spsd data structures');

    % get data
    fp_sd = fopen('C:\Settings\strength_data.txt');
```

```
    foo = textscan(fp_sd, '%s%s%s%s', 1);
    tmp_data = textscan(fp_sd, '%f%f%f');
    sdata.t = tmp_data{1,1};
    sdata.dir = tmp_data{1,2};
    sdata.spd = tmp_data{1,3};
    fclose(fp_sd);

    % a few transformations...
    sdata.dir = sdata.dir - 2048;
    sdata.spd = sdata.spd - 2048;
    sdata.mag = sqrt(sdata.dir.^2 + sdata.spd.^2);
    sdata.dt(1) = sdata.t(1);
    for j = 2:length(sdata.t)
        sdata.dt(j) = sdata.t(j) - sdata.t(j-1);
    end

    % PSD
    sfs = 1000/mean(sdata.dt);
    spsd.pow_mag = DB(periodogram(sdata.mag));
    spsd.freq = sfs/2*linspace(0,1,length(spsd.pow_mag));

    % plotting
    figure;
    subplot(2,1,1)
        plot(sdata.t,sdata.dir,':', sdata.t,sdata.spd,'--', ...
            sdata.t,sdata.mag);
        title('Raw Two-Minute Force Data');
        xlabel('time (ms)'); ylabel('force (dig)');
        legend('dir', 'spd', 'mag', 'Location', 'EastOutside');
    subplot(2,1,2)
        plot(spsd.freq,spsd.pow_mag);
        title('PSD');
        xlabel('frequency (Hz)'); ylabel('power');
        axis([0 15 0 150]);
    set(gcf, 'Name', '2-minute Data');

end

%%% Sample modification via high pass filter and WFLC
filt_analysis = input('Should the filter be analyzed [1=yes;0=no]? ');
if (filt_analysis)

    disp('Entering simulated high pass filter and WFLC...');

    % string data together
    all_data.raw_dir=[]; all_data.raw_spd=[]; all_data.t=[0];
    for i = 1:n
        all_data.raw_dir = [all_data.raw_dir; data(i).raw_dir];
        all_data.raw_spd = [all_data.raw_spd; data(i).raw_spd];
        all_data.t = [all_data.t; data(i).t+all_data.t(end)];
    end
    all_data.t = all_data.t(2:end);

    %%% Set default filter parameters
    % high pass filter
```

```
for i = 1:2
    hpf(i).order = 2;
    hpf(i).fc = 2;          %corner frquency
    hpf(i).fs = mean(fs);   %sampling frequency; ASSUME not a lot of
                            %variance among trials
    hpf(i).gain = 1;        %gain
    hpf(i).alpha = tan(pi*hpf(i).fc/hpf(i).fs);
    hpf(i).acoef = (1-hpf(i).alpha)/(1+hpf(i).alpha);
    hpf(i).type = 1;        %+1 for high-pass, -1 for low-pass
    hpf(i).bcoef = (1+hpf(i).type*hpf(i).acoef)/2;

    % WFLC
    wflc(i).mu = 0.009;
    wflc(i).mu0 = 1.2e-5;
    wflc(i).mub = 0;
    wflc(i).M = 4;
    wflc(i).w0 = 3.75*2*pi;
    wflc(i).w1 = 0;
    wflc(i).wMp1 = 0;
    wflc(i).offset = 0;
end

% determine PSD of input
all_data.pow_dir = DB(periodogram(all_data.raw_dir));
all_data.pow_spd = DB(periodogram(all_data.raw_spd));
all_data.freq = mean(fs)/2*linspace(0,1,length(all_data.pow_dir));

new_params = 1;
while (new_params)

    %apply high pass filter
    all_data.dir_hpf = MSS_highpass_filter(all_data.raw_dir, hpf(1));
    all_data.spd_hpf = MSS_highpass_filter(all_data.raw_spd, hpf(2));

    % determine PSD of hpf output
    all_data.pow_dir_hpf = DB(periodogram(all_data.dir_hpf));
    all_data.pow_spd_hpf = DB(periodogram(all_data.spd_hpf));

    %apply WFLC
    wflc_out_dir = MSS_wflc(all_data.dir_hpf, wflc(1), 0);
    all_data.dir_wflc = all_data.raw_dir - wflc_out_dir.e';
    wflc_out_spd = MSS_wflc(all_data.spd_hpf, wflc(2), 0);
    all_data.spd_wflc = all_data.raw_spd - wflc_out_spd.e';

    % determine PSD of final output (includes wflc and hpf)
    all_data.pow_dir_wflc = DB(periodogram(all_data.dir_wflc));
    all_data.pow_spd_wflc = DB(periodogram(all_data.spd_wflc));

    % Display results
    figure;
    subplot(2,1,1), plot(all_data.t,all_data.raw_dir,'b', ...
        all_data.t,all_data.dir_wflc,'r');
        title(['DIR: WFLC results for selected episodes']);
        legend('input', 'output');
        xlabel('time (msec)'); ylabel('force');
```

```
    axis([0 10000 -750 750]);
    pan on;
subplot(2,1,2), plot(all_data.freq,all_data.pow_dir,'b', ...
    all_data.freq,all_data.pow_dir_wflc,'r');
    title(['mu=' num2str(wflc(1).mu) ' mu0=' ...
        num2str(wflc(1).mu0) ' M=' num2str(wflc(1).M) ' w0=' ...
        num2str(wflc(1).w0) ' w1=' num2str(wflc(1).w1) ' wMp1=' ...
        num2str(wflc(1).wMp1)]);
    legend('input', 'output');
    xlabel('freq (Hz)'); ylabel('power');
    axis([0 12 0 150]);
set(gcf, 'Name', 'DIR time series');

figure;
subplot(2,1,1), plot(all_data.t,all_data.raw_spd,'b', ...
    all_data.t,all_data.spd_wflc,'r');
    title(['SPD: WFLC results for selected episodes']);
    legend('input', 'wflc');
    xlabel('time (msec)'); ylabel('force');
    axis([0 10000 -250 1250]);
    pan on;
subplot(2,1,2), plot(all_data.freq,all_data.pow_spd,'b', ...
    all_data.freq,all_data.pow_spd_wflc,'r');
    title(['mu=' num2str(wflc(2).mu) ' mu0=' ...
        num2str(wflc(2).mu0) ' M=' num2str(wflc(2).M) ' w0=' ...
        num2str(wflc(2).w0) ' w1=' num2str(wflc(2).w1) ' wMp1=' ...
        num2str(wflc(2).wMp1)]);
    legend('input', 'wflc');
    xlabel('freq (Hz)'); ylabel('power');
    axis([0 12 0 150]);
set(gcf, 'Name', 'SPD time series');

if (PLOT_HPF)
    figure;
    subplot(2,1,1), plot(all_data.freq,all_data.pow_dir, ...
            all_data.freq,all_data.pow_dir_hpf)
        title(['DIR: HPF infuence, fc=' num2str(hpf(1).fc)]);
        legend('input', 'output');
        xlabel('freq (Hz)'); ylabel('power');
        axis([0 12 0 150]);
    subplot(2,1,2), plot(all_data.freq,all_data.pow_spd, ...
            all_data.freq,all_data.pow_spd_hpf)
        title(['SPD: HPF infuence, fc=' num2str(hpf(2).fc)]);
        legend('input', 'output');
        xlabel('freq (Hz)'); ylabel('power');
        axis([0 12 0 150]);
end

figure;
plot(all_data.t,wflc_out_dir.w0/2/pi,all_data.t,wflc_out_spd.w0/2/pi);
    title('WFLC Frequency over Time');
    legend('dir', 'spd');
    xlabel('time (ms)'); ylabel('w0 (Hz)');
set(gcf, 'Name', 'w0 vs time');

% determine if user wants to adjust filter parameters
```

```
        new_params = input(['\nWould you like to adjust the parameters ' ...
            '[1=yes;0=no]? ']);
        if (new_params)
            for i = 1:2
                fprintf('\n');
                if (i == 1)
                    disp('Direction parameters:');
                else
                    disp('Speed parameters:');
                end
                [hpf(i) wflc(i)] = MSS_get_new_filt_params(hpf(i),wflc(i));
            end
        end
        new_params = input('\nShould the results be plotted [1=yes;0=no]? ');
    end
end

close_figure = input('\nClose all figures [0=no,1=yes]? ');
if (close_figure)
    close all
    fprintf('figures closed.\n\n');
end

if (filt_analysis)
    save_data = input('Should data be saved to setup file [0=no,1=yes]? ');
    if save_data
        %%% Write output file
        fp_out = fopen(out_filename, 'w');
        for i = 1:2
            fprintf(fp_out, '%2.0i\n', hpf(i).order);
            fprintf(fp_out, '%6.5f\n', hpf(i).fc);
            fprintf(fp_out, '%6.5f\n', hpf(i).fs);
            fprintf(fp_out, '%4.3f\n', hpf(i).gain);

            fprintf(fp_out, '%10.9f\n', wflc(i).mu);
            fprintf(fp_out, '%10.9f\n', wflc(i).mu0);
            fprintf(fp_out, '%10.9f\n', wflc(i).mub);
            fprintf(fp_out, '%2.0i\n', wflc(i).M);
            fprintf(fp_out, '%6.5f\n', wflc(i).w0);
            fprintf(fp_out, '%4.3f\n', wflc(i).w1);
            fprintf(fp_out, '%4.3f\n', wflc(i).wMp1);
        end
        iStatus = fclose(fp_out);
        if (iStatus == 0)
            fprintf('data saved to %s\n\n', out_filename);
        end
    else
        iStatus = 0;
    end
else
    iStatus = 0;
end
```

J.2 MSS_GET_TRIAL_INDEXES.M

```
function [trial_id n] = MSS_get_trial_indexes(files)

done = 0;
n = length(files);

fprintf('Which trial[s] should be analyzed [enter as array]?\n');
for i = 1:n
    fprintf('%i = %s\n', i, files(i).name);
end
while (~done)
    trial_id = input(': ');
    if (find((trial_id > n) | (trial_id < 1)))
        disp('out of bounds, try again');
    else
        done = 1;
    end
end
n = length(trial_id);
```

J.3 DRIVING_TIMES.M

```
% this code will truncate the data to times only when the subject was
% driving. see MSS_input_analysis.m for details about tmp_data, foo, and
% data.

data(i).dir = tmp_data{1,6};
data(i).spd = tmp_data{1,7};
data(i).x = tmp_data{1,8};
data(i).y = tmp_data{1,9};
TMPspeed_thresh(i) = foo{1}(4);

% find the threshold speed for each trial
j = 1;
for i = file_id
    tmp_speed_thresh(j,:) = (TMPspeed_thresh{i}(:));
    j = j + 1;
end
speed_thresh = str2num(tmp_speed_thresh);

% trim down data to only while subject was driving
j = 1;
for i = file_id
%     TMPspd = data(i).spd;
    % trial starts as soon as threshold is exceeded
    start_index = find(data(i).spd > speed_thresh(j));
    j = j + 1;
    % since chair does not move once it has finished, find the last
```

```
    % position, and then use the first index of all instances that the
    % chair was in that position as the end time index
    end_pos = [data(i).x(end) data(i).y(end)];
    end_index = find((data(i).x==end_pos(1)) & (data(i).y==end_pos(2)));
    % truncate the data
    data(i).t = data(i).t(start_index(1):end_index(1));
    data(i).dt = data(i).dt(start_index(1):end_index(1));
    data(i).raw_dir = data(i).raw_dir(start_index(1):end_index(1));
    data(i).raw_spd = data(i).raw_spd(start_index(1):end_index(1));
    data(i).raw_mag = sqrt(data(i).raw_dir.^2 + data(i).raw_spd.^2);
end
```

J.4 MSS_GET_NEW_FILT_PARAMS.M

```
function [hpf wflc] = MSS_get_new_filt_params(hpf, wflc)

ok = 0;
while (~ok)
    % HPF
    disp('Please enter HPF params [current]:');
    fprintf(' order [%i]', hpf.order);
    hpf.order = input(': ');
    fprintf(' corner freq [%3.2f]', hpf.fc);
    hpf.fc = input(': ');
    hpf.alpha = tan(pi*hpf.fc/hpf.fs);
    hpf.acoef = (1-hpf.alpha)/(1+hpf.alpha);
    hpf.type = 1;       %+1 for high-pass, -1 for low-pass
    hpf.bcoef = (1+hpf.type*hpf.acoef)/2;

    % WFLC
    disp('Please enter WFLC params [current]');
    fprintf(' mu [%3.2e]', wflc.mu);
    wflc.mu = input(': ');
    fprintf(' mu0 [%3.2e]', wflc.mu0);
    wflc.mu0 = input(': ');
    fprintf(' mub [%3.2e]', wflc.mub);
    wflc.mub = input(': ');
    fprintf(' M [%i]', wflc.M);
    wflc.M = input(': ');
    fprintf(' w0 [%3.3f*2*pi]', wflc.w0/2/pi);
    wflc.w0 = input(': ');
    wflc.w0 = wflc.w0*2*pi;
%     wflc.w1 = 0;
%     wflc.wMp1 = 0;
%     wflc.offset = 0;

    ok = input('Are these correct [1=yes;0=no]? ');
end
```

J.5 MSS_WFLC.M

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   modified: 8/24/06
%   by: kwb
%
%   >> PURPOSE <<
%   This function implements an Mth order WFLC filter. It is based
%   on the algorithm given at http://www.cs.cmu.edu/~camr/wflc0.c. The
%   following is the description from that site:
%
%%%
%
%   WFLC0.C         Weighted-frequency Fourier Linear Combiner 0.0
%   Cameron N. Riviere, Carnegie Mellon University
%   last mod.    11/5/98
%   This is WF1H, modified to include harmonics.
%   Adaptive weighted-frequency algorithm for noise canceling in 1-D data.
%   Here, frequency of reference, as well as ampl., is an adaptive weight.
%   Mu is the standard adapt. rate param.  mu0 is a 2nd param. on the
%       frequency weight.  A bias (highpass) weight is used.
%   Ampl. weights for harmonics will be initialized to 0.
%   To run this program, type at the DOS prompt:
%   WFLC0 filename ext mu mu0 mub M w0 w1 wM+1 offset
%   Note:  This code is written in Turbo C for the PC.
%
%%%
%
%   >> NOMENCLATURE <<
%
%   >> DEFAULT VALUES <<
%
%     wflc.mu = 0.009;
%     wflc.mu0 = 1.2e-5;
%     wflc.mub = 0;
%     wflc.M = 4;
%     wflc.w0 = 3.75*2*pi;
%     wflc.w1 = 0;
%     wflc.wMp1 = 0;
%     wflc.offset = 0;
%
%   >> HISTORY <<
%   Version 1.0         karl brown          13 November 2006
%   original version. based on test_wflc_tmp.m
%*************************************************************************/

function out = MSS_wflc(data, wflc, plot_results)

% initializations
w0_1 = wflc.w0;
wbias=0;
for i = 1+1:2*wflc.M+1
    w(i) = 0;
end
```

```
w(1+1) = wflc.w1;
w(wflc.M+1+1) = wflc.wMp1;
sumw0 = 0;

for j = 1:length(data)

    % remove offset
    input = data(j) - wflc.offset;

    % locate next sine & cosine samples */
    sumw0 = sumw0 + wflc.w0;
    for i = 2:wflc.M+1
        x(i) = sin(i*sumw0);
        x(wflc.M+i) = cos(i*sumw0);
    end

    % output = truncated Fourier series */
    output = 0;
    for i=1+1:2*wflc.M+1
        output = output + w(i)*x(i);
    end

    % calculate error */
    e = input - output - wbias;

    % update bias weight */
    wbias = wbias + 2*wflc.mub*e;

    % update frequency weight, 'blind' to harmonics */
    sumcross = 0;
    for i=1+1:wflc.M-1+1
        sumcross = sumcross + i*(w(wflc.M+i)*x(i)-w(i)*x(wflc.M+i));
    end
    wflc.w0 = wflc.w0 - 2*wflc.mu0*e*sumcross;

    % update amplitude weights */
    for i = 1+1:2*wflc.M+1
        w(i) = w(i) + 2*wflc.mu*e*x(i);
    end

    % output*/
    out.e(j) = e + wflc.offset;
    out.output(j) = output;
    out.w0(j) = wflc.w0;
    out.w1(j) = w(1+1);
    out.wMp1(j) = w(wflc.M+1+1);
    out.wbias(j) = wbias + wflc.offset;
    k = 1;
end

% plot results
if (plot_results)
    t = data(:,1);
    figure; plot(t,data(:,2), t,out.e, t,out.output+wflc.offset);
    legend('data', 'e', ['output+' num2str(wflc.offset)]);
```

```
    title(['mu=' num2str(wflc.mu) ' mu0=' num2str(wflc.mu0) ' M=' ...
        num2str(wflc.M) ' w0=' num2str(w0_1) ' w1=' num2str(wflc.w1) ...
        ' wMp1=' num2str(wflc.wMp1)]);
    xlabel('time (sec)'); ylabel('output');
    axis([20 30 -7 7]);
    pan on;

    % determine PSD of input
    pow = DB(periodogram(data(:,2) - mean(data(:,2))));
    freq = 62.5/2*linspace(0,1,length(pow));
    figure;
    subplot(2,1,1); plot(freq,pow);
    title('PSD of input');
    axis([0 6 -100 100]);
    xlabel('freq (Hz)'); ylabel('power');

    % determin PSD of output
    pow = DB(periodogram(out.output));
    freq = 62.5/2*linspace(0,1,length(pow));
    subplot(2,1,2); plot(freq,pow);
    title('PSD of output');
    axis([0 6 -100 100]);
    xlabel('freq (Hz)'); ylabel('power');
end
```

J.6 MSS_HIGHPASS_FILTER.M

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   modified: 8/24/06
%   by: kwb
%
%   >> PURPOSE <<
%
%   This function implements an Nth order IIR high pass filter. It is
%   based on code from http://www.ddj.com/dept/cpp/184401931.
%
%   >> INPUTS <<
%   datum       vector of time series data
%   filt        structure containing filter parameters
%
%   >> NOMENCLATURE <<
%
%   >> HISTORY <<
%   Version 1.0         karl brown          13 November 2006
%   original version.
%*************************************************************************/

function y = MSS_highpass_filter(data, filt)

global filts;
```

```
filts = filt;

%initialize filter
filts.xn1(1:filts.order) = 0; filts.yn1(1:filts.order) = 0;

data_hpf = data;
for i = 1:length(data)
    for j = 1:filts.order
        data_hpf(i) = iir_highpass(data_hpf(i), j);
    end
end
y = data_hpf;

end

%%%%%%%%
% this is the actual filter
function yy = iir_highpass(datum, order_count)

global filts;

filts.yn1(order_count) = filts.bcoef*datum + ...
    filts.type*(-filts.bcoef)*filts.xn1(order_count) + ...
    filts.acoef*filts.yn1(order_count);
filts.xn1(order_count) = datum;
filts.y(order_count) = filts.yn1(order_count)*filts.gain;
yy = filts.y(order_count);

end
```

APPENDIX K

DEMOGRAPHIC QUESTIONNAIRE

Subject ID: _____

PART A - Personal Data

First Name: _____ Last Name: _____ Middle Initial: _____

Street: _____

City: _____

State: _____ Zip code: _____

Telephone: _____(Home) _____ (Work, if applicable)

Gender: _____ Male (0) _____ Female (1)

Date of Birth: __/__/__ Weight: _____

Ethnic Origin:
_____ African-American (1)
_____ American Indian (2)
_____ Asian-American (3)
_____ Caucasian (4)
_____ Hispanic (5)
_____ Other (6): _____

Veteran Status: _____ U.S. Veteran (1) _____ I am **not** a U.S. Veteran (0)

Type of MS:
_____ Relapsing-remitting (1)
_____ Primary progressive (2)
_____ Secondary progressive (3)
_____ Progressive relapsing (4)
_____ Other (5): _____

Date of diagnosis: __/__/__

What is your experience with manual and electric powered wheelchairs?

- ❑ Someone else pushes me in a manual wheelchair
- ❑ I push myself in a manual wheelchair
- ❑ I can use an electric power wheelchair in certain environments
- ❑ I can independently use an electric power wheelchair
- ❑ Other: (specify) _____

What make and model of wheelchair(s) do you own?_____

☺ **Thank you for taking the time to complete this questionnaire** ☺
We appreciate your participation!

Figure 97: Demographic questionnaire.

APPENDIX L

SOLVING THE VDS LOOP RATE PROBLEM

For digital filters to work properly, the input samples must have equal weights in time [60]. The easiest way to achieve this is to sample the signal at a constant rate. Since the HERL IJ processes its data onboard and does not incorporate a digital filter, the loop rate of the VDS did not need to be constant. However, the VCJ software is more integrated into the VDS by processing its input each time VDS software makes a request for a sample, and it includes high-pass and adaptive filters. Therefore, to ensure that the VCJ digital filters function properly, the loop rate of the VDS needed to be more consistent. Specifically, the period should not deviate by more than 2 ms off its average. After the source of the timing aberrances was located, restructuring the loop, controlling the time at which data is acquired, and handling events when data acquisition is delayed produced a consistent loop rate for the VDS software.

L.1 IDENTIFICATION OF THE TIMING PROBLEM

Two main events appeared to cause the inconsistent loop rate: lengthy DAQ reads and Windows messages. The loop period for a sample trial, shown in **Figure 98**, is mostly consistent except for every 5.007 s, when it doubles, and about every 20 s, when there is a delay. Other trials showed slightly more variability in loop rate over 30-s episodes. To isolate the potential sources for the delay, time stamps at selected points in the loop were written to a data file. These stamps were organized and plotted to determine which specific processes might take longer than normal. Table 18 shows descriptive statistics for the individual processes. The red circles in **Figure 99**

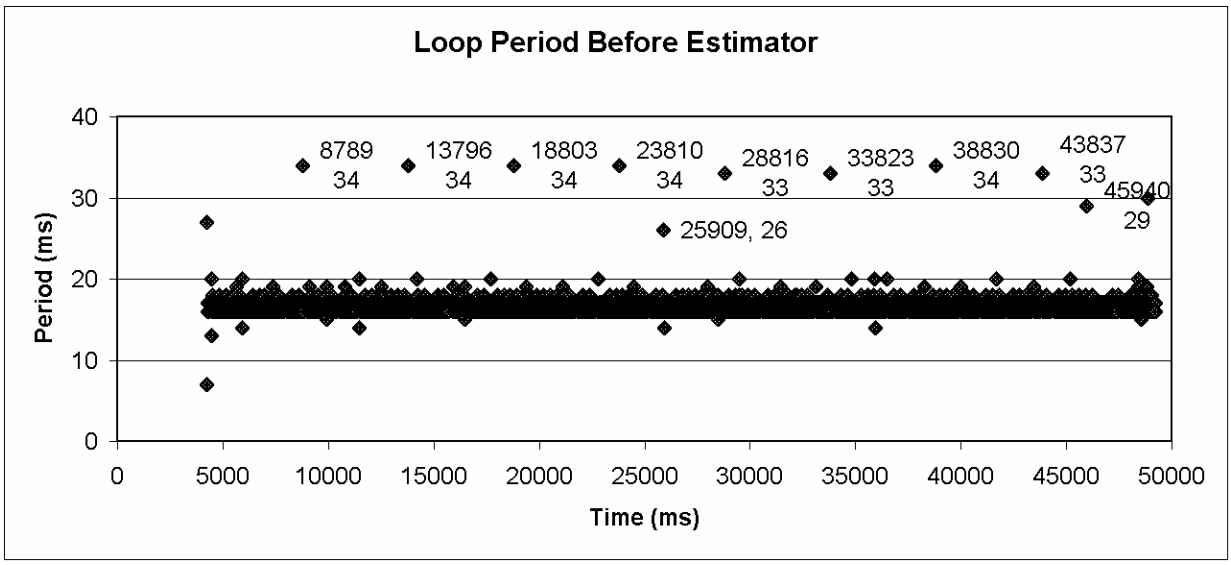


Figure 98: Period of loop without any intervention.

Table 18: Descriptive statistics for time required to complete various processes in the VDS loop. Values are in ms.

Description	Line Number	Average	Standard Deviation	Min	Max
Beginning of loop	233	0.5138	0.9513	0	13
Process keyboard inputs	431	0.3460	0.4951	0	5
Acquire joystick data	446	5.5709	0.8056	4	18
Apply joystick algorithm	467	0.6728	0.5100	0	5
Update chair position	522	0.4043	0.4948	0	2
Check crash, save data, and draw image	569	9.4059	0.9241	1	18

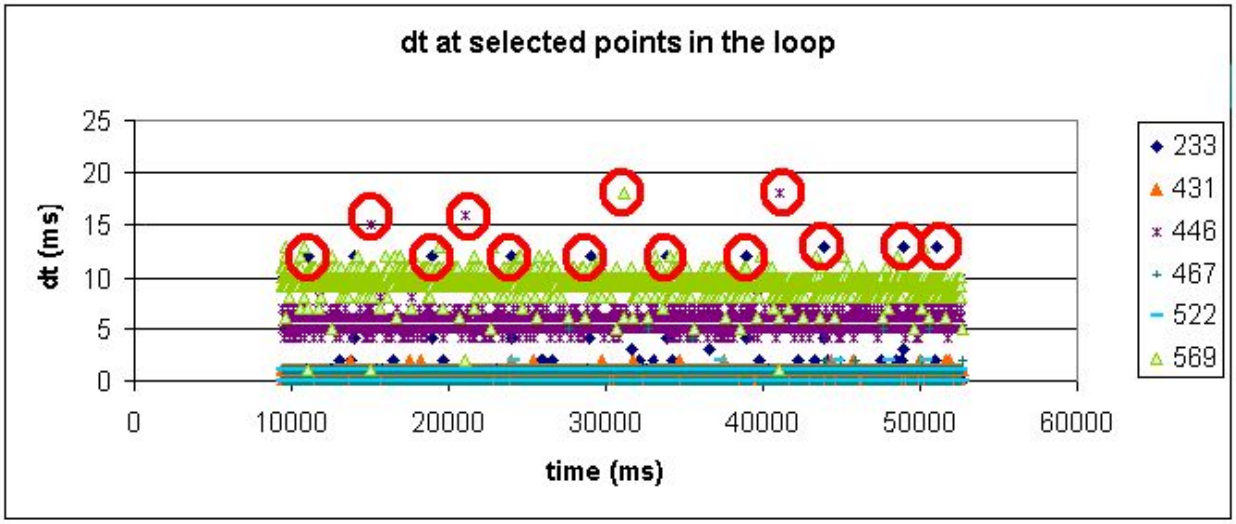


Figure 99: Time stamps at selected points in the loop (233=beginning, 431=process keyboard inputs, 446=acquire joystick data, 467=apply joystick algorithm, 522=update chair position, 569=check chair crash and update screen).

highlight processes that took longer than normal. Based on this information, while the processes do not always appear consistent, background processes between the end and beginning of the loop are the most likely source for the delays every 5 s. The delay every 20 s is likely because of the reading from the DAQCard or aberrances in rendering the screen.

L.2 ORIGINAL SOLUTIONS

Since events happening outside the loop are not under control of the VDS software but the operating system (OS), the first attempt to solve the problem was simply to identify when a delay occurred and interpolate the missing point so that the filters could have equally-timed samples. Neville's Iterated Interpolation method was chosen as the interpolating algorithm for its computational simplicity and apparent accuracy [87]. Unfortunately, while this estimation method caught most of the delays that occurred every 5 s, the 20 s delays were not handled consistently (**Figure 100**). Further analysis revealed these delays to be associated with longer readings from the DAQCard. Thus, alternate methods for acquiring data from the DAQCard, including continuous reads and registering events, were investigated.

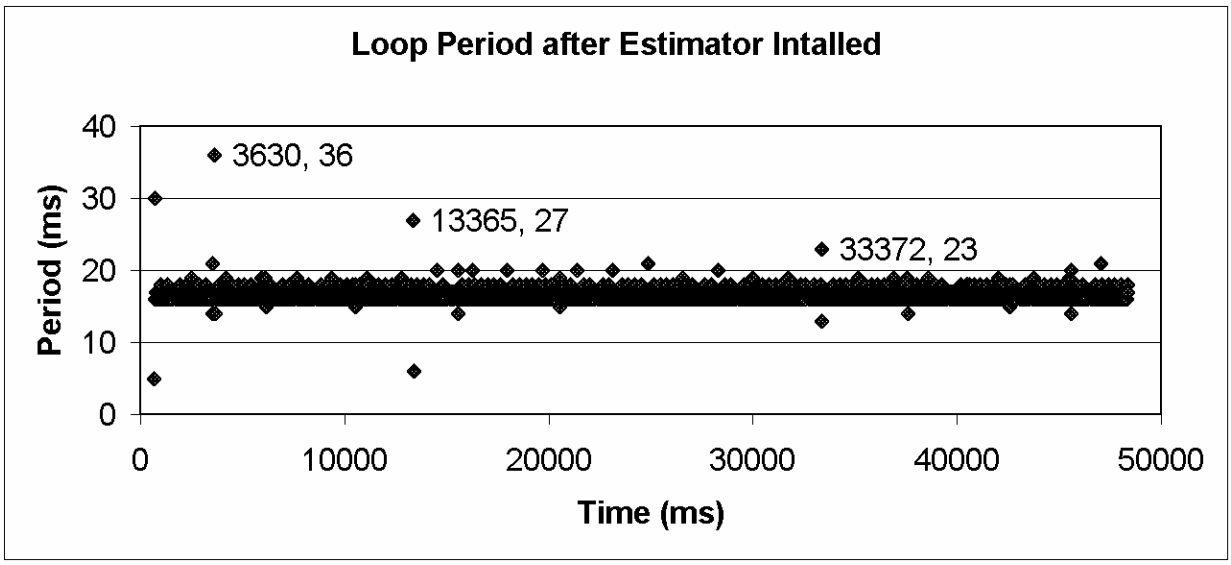


Figure 100: Loop period after first estimator was installed.

At the time, the method for acquiring data from the DAQCard involved starting a scan, collecting 50 points at 65 kHz, and averaging the data. This process takes approximately 5.6 ms to complete. An alternative method for collecting samples is to make the DAQCard acquire data into a PC buffer continuously, while the VDS software is carrying out other processes. Ideally, all the VDS software would need to do to get a sample would be to read the latest points out of the buffer, which should take less than a millisecond and minimize any potential for delay in the data acquisition process. Unfortunately, this method proved to be too much for the OS to handle.

The first problem was that the index to the PC buffer for the latest sample would not update until 1,024 samples had been acquired. Or, as later discovered, the default settings in NI's PCMIA DAQCards cause the data to be transferred from its internal buffer to the PC buffer in blocks of 1,024 samples, even in continuous acquisition mode. These settings can be changed to cause data to be transferred every time the half buffer is full or at every sample. In the former case, the buffer size is still too large at 512 samples. To have the data be at most 2 ms old, 512 samples would need to be acquired at about 250 kHz and transferred across the bus, which is beyond the 200-kHz limit of the DAQCard. In the latter case, transferring the data to the PC buffer at every sample caused the entire software to behave erratically. In essence, transferring data across the PCMCIA bus at every sample interrupts all other processes happening in the OS while the transfer is taking place.

The second alternative, registering events, theoretically should provide more control over how much and how often data is transferred across the PCMCIA bus. Registering an event causes a user-defined function to execute any time an event occurs, such as after `n` number of samples are acquired or `t` seconds have elapsed or Windows receives a message, independent of other processes happening in the OS. In our case, we would want somewhere between 10 and 50 samples to be transferred across the bus after 2 ms at most to keep noise low and to ensure consistent timing. While the events functioned as expected during the VDS's initialization procedures, Windows would stop handling the events or the VDS software would not release control to the callback function after about 0.5 s into the main driving loop. Therefore, after all methods had been exhausted to reduce loop rate dependency on DAQCard functions and none worked, using a synchronous scan as originally implemented appeared to be only option for acquiring data.

L.3 PROPOSED SOLUTION

Restructuring the loop, controlling the time at which data is acquired, and handling events when data acquisition is delayed ultimately produced a consistent loop rate for the VDS software. In process of implementing registered events, the source of the delay every 5.007 ms revealed itself. The Windows OS monitors and handles its processes through messages, something the VDS software handles with the `PeekMessage()` function at the very beginning of the loop. If Windows provides a message to the VDS software, VDS software dispatches it appropriately; *otherwise* it processes the virtual driving simulation. That is, the virtual driving simulation part of the VDS software will not execute until all Windows messages have been processed. And, every 5 s Windows would provide the `MOUSE_MOVE` message to the software thus causing the hiccup. Unfortunately, there did not appear to be a method for removing the mouse messages without uninstalling the mouse. Thus, the problem was solved by removing the `else` from the message handling portion of the VDS software, allowing the driving simulation to execute even if Windows provides a message. To prevent the loop from executing when the user desires to exit the program, closing routines were added to the Windows message processing function

`WndProc()` in the VDS software. This solution, however, does not address the instances when the read function takes longer than normal.

Control of when reads begin and logic for handling extended reads compensate for long read times. Controlling when reads begin is easily solved by calling the read function only after the average loop period has elapsed since the last read began. If the read takes longer than normal, action is taken based on the amount of time that the read takes beyond the normal read time, or excess time. There are five regions beyond the expected read time in which the excess time can occur, illustrated in **Figure 101**. If the excess time is in region 1, or within 2 ms of the expected read time, the reading is used and the loop continues as normal. If the excess time is in region 2, between 2 ms and 6 ms after the expected read time, the missing sample is interpolated with Neville's Iterative Interpolation method, the interpolated reading is fed through the filters, and the program goes to the beginning of the loop to acquire a new reading. If the excess time is in region 3, between 6 and 15 ms after the expected read time, the missing sample is interpolated and fed through the filters, the subsequent sample is interpolated, and the program waits until the subsequent sample would have occurred before continuing with the simulation. This is the only case in which an interpolated point is used to command the chair. If the excess time is in region 4, between 15 and 19 ms following the expected read time (or ±2 ms of the following read time), the missing point is interpolated and fed through the filters, and the acquired point is used for the rest of the loop. Lastly, if the excess time is in region 5, the missing point is interpolated and fed through the filters and the excess time will be updated. The logic described above is repeated until the excess time lies in any of regions 1 through 4.

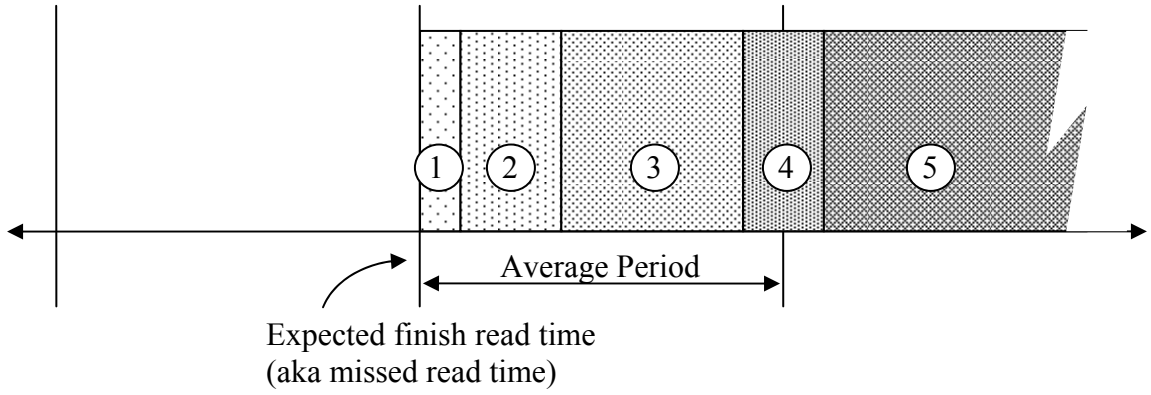


Figure 101: Possible regions of loop execution in which excess time may lie.

L.4 RESULTS

The resulting intervention appears to provide adequate control of the loop period because the new loop rate is within the specifications and the interpolative measures do not appear to be out of bounds. A 95-s trial with the intervention (**Figure 102**) resulted in an average period of 17.01 ms (standard deviation of 0.235 ms) and min and max periods of 14 and 19 ms, respectively. The minimum occurred only once; otherwise the minimum period was 15 ms. The frequency of the interpolated points is provided in Table 19. Interpolated points accounted for only 1.03% of all samples. Interpolated points in context of the raw input for several cases when an error code was generated are provided in **Figure 103** through **Figure 105**, demonstrating that the interpolative algorithm does not appear to generate unreasonable values. However, data in **Figure 105** show that it is possible for a string of type 2 (corresponding to region 2 in **Figure 101**) errors to be generated, inhibiting the chair position from being updated for brief periods of time.

L.5 DISCUSSION

The proposed solution contains a few limitations such as glitches, decreased performance, limited multi-tasking, and occasional software crashes. The glitches are likely a result of the string of type 2 errors and cause the chair to appear to jump a short distance on the screen. The frequency of these glitches is about 1 in 5 or 6 trials. If, in the context of subject testing, a glitch influences the subject's driving, the given trial can be re-run.

While the average loop period before the proposed solution was implemented was about 16.4 ms to 16.9 ms, the fastest average period achieved here is 17.0 ms. The source is likely a result of the timing variables being in integer (DWORD) format. Defining the average period to be 16 ms or 16.5 ms causes the program to crash. An alternative way to improve the loop rate would be to switch from using the `timeGetTime()` function to sampling the processor clock. This would improve the timing resolution to microseconds, but the processor clock is reported to skip a few seconds randomly. The cost of switching timing does not seem justified at this time for the potential improvement in performance.

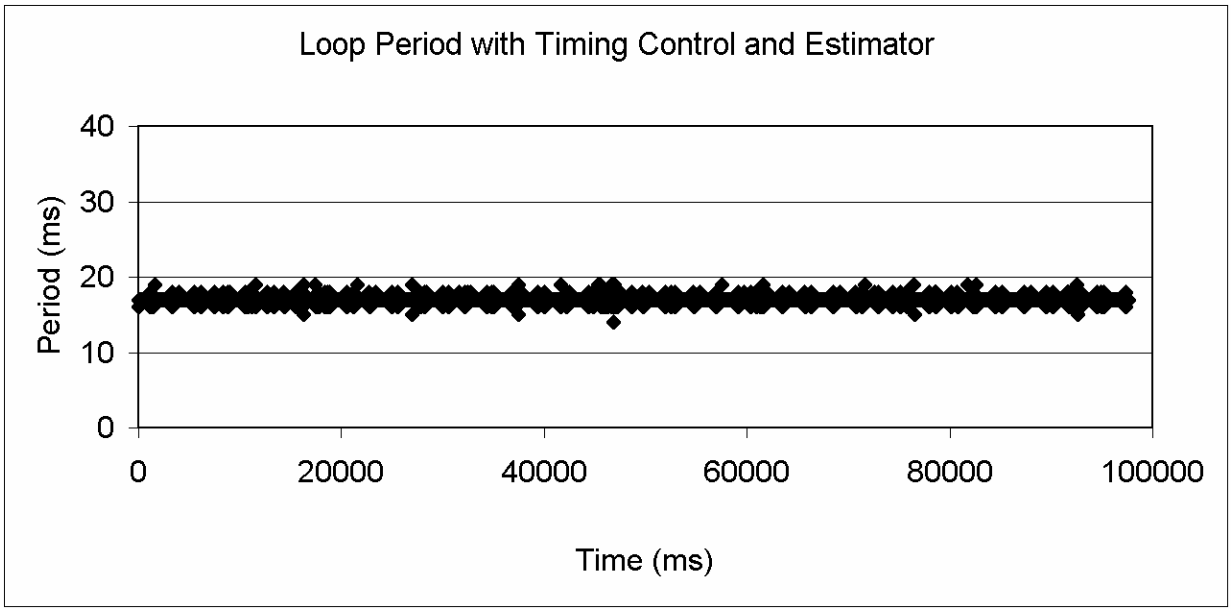


Figure 102: Loop period with timing control and estimator.

Table 19: Number of instances of each excess time scenario during a 95-s trial.

Error Code	Corresponding region, Definition	Frequency
0	1, raw input used	5711
1	5	0
2	2, go back and get a new reading	41
3	3, previous sample interpolated	9
4	3, following sample interpolated	9
5	4	0

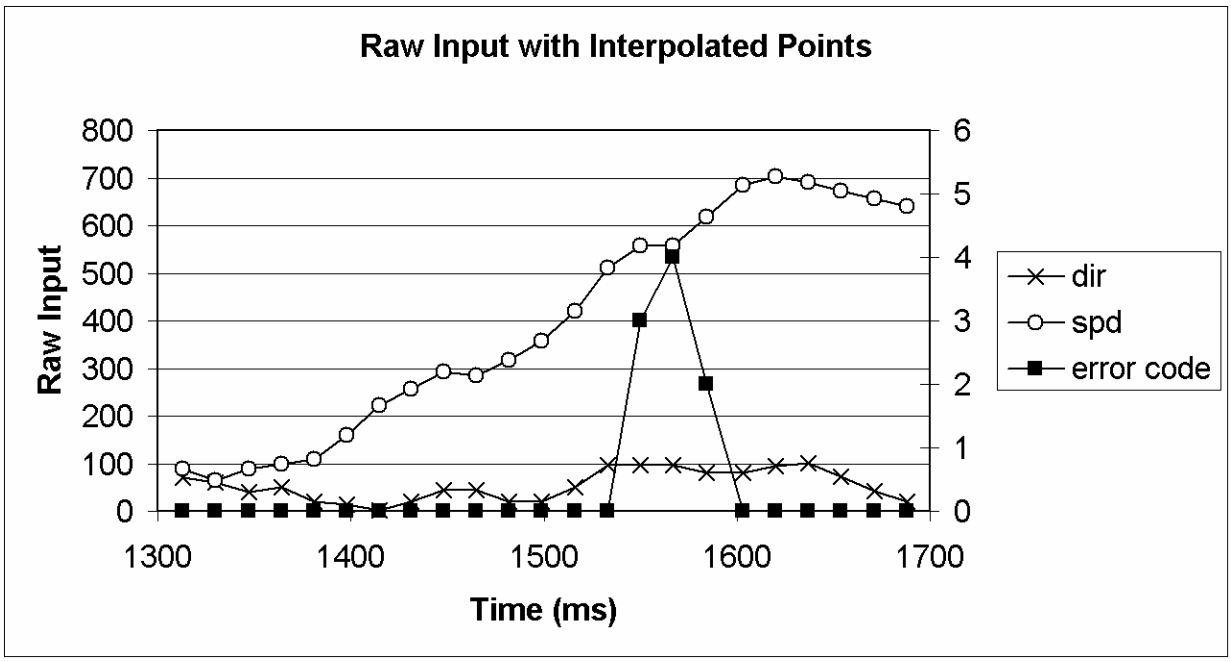


Figure 103: Raw input with interpolated points. The direction and speed axes are on the left, and the error code axis is on the right.

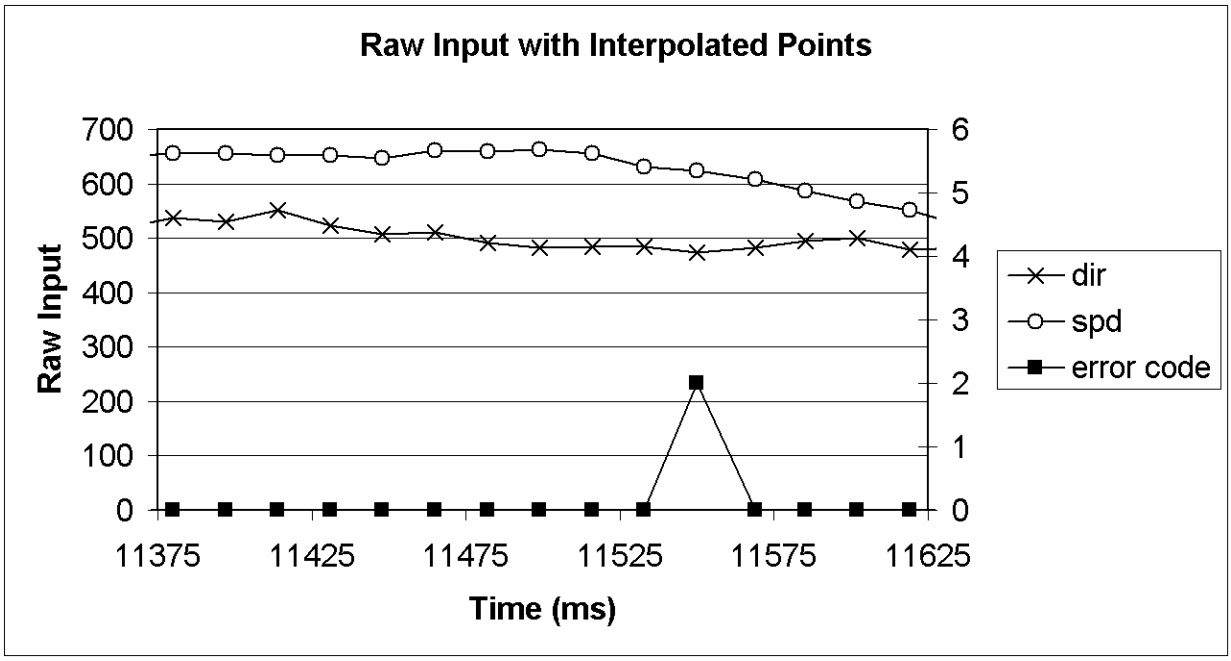


Figure 104: Raw input with an interpolated point. The direction and speed axes are on the left, and the error code axis is on the right.

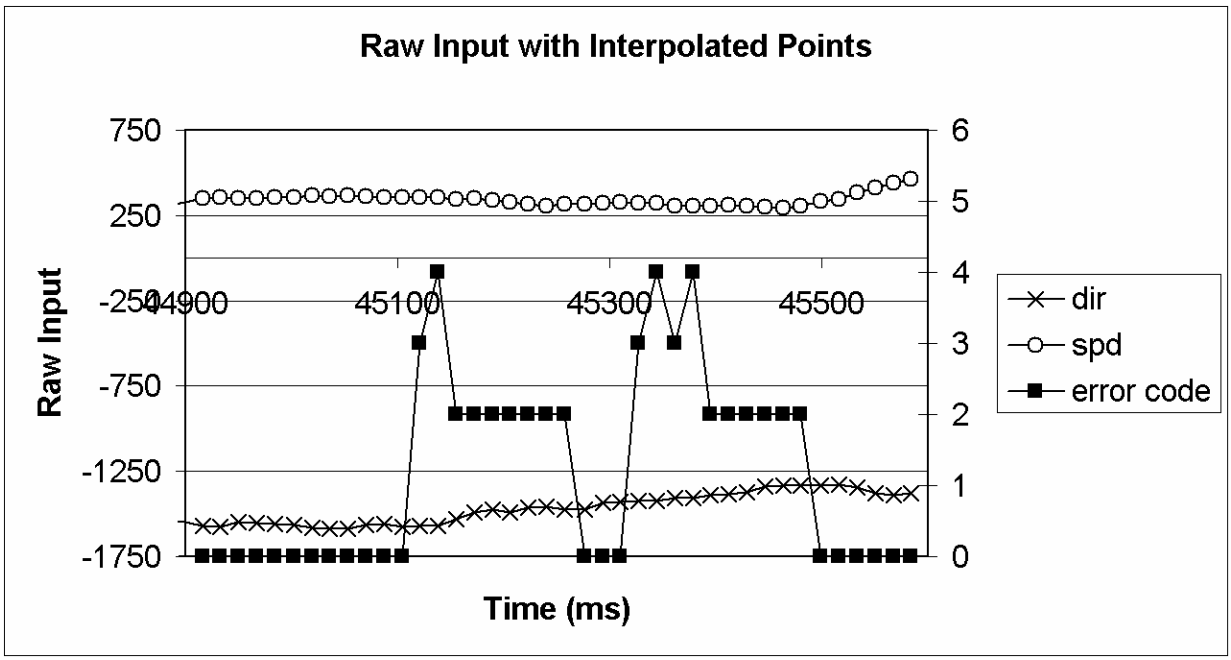


Figure 105: Raw input with string of interpolated points. The direction and speed axes are on the left, and the error code axis is on the right.

Because the timing intervention is sensitive to delays in the loop, other tasks such as showing current parameters or adjusting joystick settings cannot be performed while the timing intervention is active. Displaying the information on the screen takes too much processing power. Therefore, if the clinician would like to update parameters, the timing intervention will not be implemented while information is displayed on the screen. Also, the VDS software will not process any keyboard inputs while the subject is driving on any of the tracks except the rectangular track. While displaying information on the screen is a nice feature, it is not part of the study protocol and therefore not needed.

Occasionally, the software does stop responding, or crash. The exact cause of the crash has not been identified, but its effect is limited only to the current task. Data collected from other driving tasks is saved whenever the user presses the escape key to go back to the main menu of the VDS software at the completion of a track. Crashes seem to occur between 5% and 10% of the time the program is run.

L.6 CONCLUSIONS

The goal of improving the consistency of the loop period has been achieved with error handling and interpolative procedures. Unfortunately, the VDS environment could not support the more elegant method of continuous or buffered sampling. The procedures introduce a few limitations such as glitches, reduced performance, and limited multi-tasking; but they do not seem catastrophic.

BIBLIOGRAPHY

[1] Fehr L, Langbein E, Skaar SB. Adequacy of power wheelchair control interfaces for persons with severe disabilities: A clinical survey. *Journal of Rehabilitation Research and Development*. 37: 353-360, 2000.

[2] Beringhause S, Rosen M, Huang S. Evaluation of a damped joystick for people disabled by intention tremor. *Proceedings of the 12th Annual Conference on Rehabilitation Technology*. New Orleans, LA, pgs 41-2, 1989.

[3] Brienza DM, Angelo J, A force feedback joystick and control algorithm for wheelchair obstacle avoidance. *Disability & Rehabilitation*. 18: 123-9, 1996.

[4] Cooper RA, Widman LM, Jones DK, Robertson RN. Force sensing control for electric powered wheelchairs. *IEEE Transactions on Control Systems Technology.* 8: 112-117, 2000.

[5] Guo G, Grindle G, Cooper RA. Development of a Compact Chin-Operated Force Sensing Joystick. *Saudi Journal of Disability and Rehabilitation.* 9: 212-217, 2005.

[6] Powell F, Inigo RM. Pressure sensitive joystick and controller for front wheel steering wheelchairs. *Proceedings of the RESNA 15th Annual International Conference.* Toronto, Canada, pgs 304-306, 1992.

[7] Fay BT, Boninger ML. The science behind mobility devices for individuals with multiple sclerosis. *Medical Engineering & Physics*. 24: 375-83, 2002.

[8] Noonan CW, Kathman SJ, White MC. Prevalence estimates for MS in the United States and evidence of an increasing trend for women. *Neurology.* 58: 136-8, 2002.

[9] Noseworthy JH, Lucchinetti C, Rodriguez M. Multiple Sclerosis. *New England Journal of Medicine*. 343: 939-952, 2000.

[10] Pittock SJ, Mayr WT, McClelland RL, Jorgensen NW, Weigand SD, Noseworthy JH, Weinshenker BG, Rodriguez M. Change in MS-related disability in a population-based cohort: a 10-year follow-up study. *Neurology*. 62: 51-9, 2004.

[11] Zifko UA, Rupp M, Schwarz S, Zipko HT, Maida EM. Modafinil in treatment of fatigue in multiple sclerosis. Results of an open-label study. *Journal of Neurology*. 249: 983-7, 2002.

[12] Schwid SR, Covington M, Segal BM, Goodman AD. Fatigue in multiple sclerosis: current understanding and future directions. *Journal of Rehabilitation Research & Development*. 39: 211-24, 2002.

[13] Flachenecker P, Kumpfel T, Kallmann B, Gottschalk M, Grauer O, Rieckmann P, Trenkwalder C, Toyka KV. Fatigue in multiple sclerosis: a comparison of different rating scales and correlation to clinical parameters. *Multiple Sclerosis*. 8: 523-6, 2002.

[14] Krupp LB. Fatigue in Multiple Sclerosis: Definition, Pathophysiology and Treatment. *CNS Drugs.* 17: 225-234, 2003.

[15] Krupp LB, Soefer MH, Pollina DA, Smiroldo J, Coyle PK. Fatigue measures for clinical trials in multiple sclerosis. *Neurology.* 50: A126, 1998.

[16] Schwid SR, Thornton CA, Pandya S, Manzur KL, Sanjak M, Petrie MD, McDermott MP, Goodman AD. Quantitative assessment of motor fatigue and strength in MS. *Neurology.* 53: 743-50, 1999.

[17] Fisk JD, Pontefract A, Ritvo PG, Archibald CJ, Murray TJ. The impact of fatigue on patients with multiple sclerosis. *Canadian Journal of Neurological Sciences*. 21: 9-14, 1994.

[18] Krupp LB, Elkins LE. Fatigue and declines in cognitive functioning in multiple sclerosis. *Neurology.* 55: 934–39, 2000.

[19] Freal JE, Kraft GH, Coryell JK. Symptomatic fatigue in multiple sclerosis. *Archives of Physical Medicine & Rehabilitation*. 65: 135-8, 1984.

[20] Djaldetti R, Ziv I, Achiron A, Melamed E. Fatigue in multiple sclerosis compared with chronic fatigue syndrome: A quantitative assessment. *Neurology.* 46: 632-635, 1996.

[21] Alusi SH, Glickman S, Aziz TZ, Bain PG. Tremor in multiple sclerosis. *J. Neurol. Neurosurg. Psychiatry.* 66: 131-134, 1999.

[22] Alusi SH, Worthington J, Glickman S, Bain PG. A study of tremor in multiple sclerosis. *Brain.* 124 (Pt 4): 720-30, 2001.

[23] Liu X, Miall RC, Aziz TZ, Palace JA, Stein JF. Distal versus proximal arm tremor in multiple sclerosis assessed by visually guided tracking tasks. *Journal of Neurology, Neurosurgery & Psychiatry*. 66: 43-7, 1999.

[24] Liu X, Miall C, Aziz TZ, Palace JA, Haggard PN, Stein JF. Analysis of action tremor and impaired control of movement velocity in multiple sclerosis during visually guided wrist-tracking tasks. *Movement Disorders.* 12: 992-9, 1997.

[25] Gillen G, Improving mobility and community access in an adult with ataxia. American Journal of Occupational Therapy. 56: 462-6, 2002.

[26] Cooper RA. *Wheelchair Selection and Configuration*. Demos, New York: 1998.

[27] Routhier F, Vincent C, Desrosiers J, Nadeau S. Mobility of wheelchair users: a proposed performance assessment framework. *Disability and Rehabilitation*. 25: 19-34, 2003.

[28] Cooper RA, Fitzgerald SG, Boninger ML, Prins K, Rentschler AJ, Arva J, O'connor TJ. Evaluation of a pushrim-activated, power-assisted wheelchair. *Archives of Physical Medicine & Rehabilitation.* 82: 702-8, 2001.

[29] Algood SD, Cooper RA, Fitzgerald SG, Cooper R, Boninger ML. Impact of a pushrim-activated power-assisted wheelchair on the metabolic demands, stroke frequency, and range of motion among subjects with tetraplegia. *Archives of Physical Medicine & Rehabilitation.* 85: 1865–71, 2004.

[30] Corfman TA. Cooper RA. Boninger ML. Koontz AM. Fitzgerald SG. Range of motion and stroke frequency differences between manual wheelchair propulsion and pushrim-activated power-assisted wheelchair propulsion. *Journal of Spinal Cord Medicine.* 26: 135-40, 2003.

[31] Davies A, De Souza LH, Frank AO. Changes in the quality of life in severely disabled people following provision of powered indoor/outdoor chairs. *Disability & Rehabilitation*. 25: 286-90, 2003 .

[32] Blake DJ and Bodine C. An overview of assistive technology for persons with multiple sclerosis. *Journal of Rehabilitation Research and Development*. 39: 299-312, 2002.

[33] Cook AM, Hussey SM. *Assistive Technologies: Principles and Practices*. Mosby, St. Louis, MO: 2002.

[34] Cooper RA, Jones DK, Fitzgerald S, Boninger ML, Albright SJ. Analysis of position and isometric joysticks for powered wheelchair driving. *IEEE Transactions on Biomedical Engineering*. 47: 902-10, 2000.

[35] Cooper RA, Spaeth DM, Jones DK, Boninger ML, Fitzgerald SG, Guo S. Comparison of virtual and real electric powered wheelchair driving using a position sensing joystick and an isometric joystick. *Medical Engineering & Physics*. 24: 703-8, 2002.

[36] Pellegrini N, Guillon B, Prigent H, Pellegrini M, Orlikovski D, Raphael J, Lofaso F. Optimization of power wheelchair control for patients with severe Duchenne muscular dystrophy. *Neuromuscular Disorders*. 14: 297-300, 2004.

[37] Repperger DW, Phillips CA, Chelette TL. Performance study involving a force-reflecting joystick for spastic individuals performing two types of tracking tasks. *Perceptual & Motor Skills.* 81: 561-2, 1995.

[38] Repperger DW. Management of spasticity via electrically operated force reflecting sticks. *Proceedings of the RESNA 14th Annual Conference*. Kansas City, MO, pg 27, 1991.

[39] Rosen M, Sloan M, Biber C. A Damped Joystick: Adaptive Control for the Tremor-Disabled. *Proc 2nd Interagency Conf on Rehab Eng*. Atlanta, GA, pp. 74-79, 1979.

[40] Hendriks J, Rosen M, Berube N, Aisen M. A second-generation joystick for people disabled by tremor. *Proceedings of the 14th Annual RESNA Conference*. Kansas City, MO, pgs 248-51, 1991.

[41] Pledgie S, Barner KE, Agrawal SK, Rahman T. Tremor suppression through impedance control. *IEEE Transactions on Rehabilitation Engineering*. 8: 53-9, 2000.

[42] Riley PO, Rosen MJ. Evaluating Manual Control Devices for Those with Tremor Disability. *J. of Rehab. Research and Development.* 24: 99-110, 1987.

[43] Gonzalez JG, Heredia EA, Rahman T, Barner KE, Arce GR. Optimal digital filtering for tremor suppression. *IEEE Transactions on Biomedical Engineering.* 47: 664-73, 2000.

[44] Riviere CN, Thakor NV. Adaptive Human Machine Interface for Persons with Tremor. *Eng. Medicine Biology Conference*. 1995.

[45] Riviere CN, Reich SG, Thakor NV. Adaptive Fourier modeling [sic] for quantification of tremor. *Journal of Neuroscience Methods*. 74: 77-87, 1997.

[46] Riviere CN, Rader RS, Thakor NV. Adaptive cancelling of physiological tremor for improved precision in microsurgery. *IEEE Transactions on Biomedical Engineering.* 45: 839-46, 1998.

[47] Nho W. Development and evaluation of an enhanced weighted frequency Fourier linear combiner algorithm using bandwidth information. Doctoral Dissertation. University of Pittsburgh, 2006.

[48] Phillips CA, Repperger DW, Chelette TL. The acceleration-velocity relationship: identification of normal and spastic upper extremity movement. *Computers in Biology & Medicine*. 27: 309-28, 1997.

[49] Aisen ML, Arnold A, Baiges I, Maxwell S, Rosen M. The Effect of Mechanical Damping Loads on Disabling Action Tremor. *Neurology.* 43: 1346-50, 1993.

[50] Kotovsky J, Rosen MJ. A wearable tremor-suppression orthosis. *Journal of Rehabilitation Research & Development*. 35: 373-87, 1998.

[51] Rosen MJ, Arnold AS, Baiges IJ, Aisen ML, Eglowstein SR. Design of a controlled-energy-dissipation orthosis (CEDO) for functional suppression of intention tremors. *Journal of Rehabilitation Research & Development*. 32: 1-16, 1995.

[52] Cooper R. *Rehabilitation Engineering Applied to Mobility and Manipulation*. Institute of Physics Publishing, London, England: 1995.

[53] Martín P, Mazo M, Fernández I, Lázaro JL, Rodríguez FJ, Gardel A. Multifunctional and autonomous, high performance architecture: application to a wheelchair for disabled people that integrates different control and guidance strategies. *Microprocessors and Microsystems*. 23: 1-6, 1999.

[54] Mazo M, et al. An Integral System for Assisted Mobility. *IEEE Robotics and Automation Magazine.* 7: 46-56, (2001).

[55] LoPresti E, Simpson R, Hayashi S, Nourbakhsh I, Miller D. Development of a Smart Wheelchair Component System. *Proc 26th Ann Conf RESNA.* Atlanta, GA, 2003.

[56] Hayashi ST, LoPresti EF, Simpson RC, Nourbaksh I, Miller D. An inexpensive, alternative, drop-off detection solution. *Proc 26th Ann Conf RESNA*. Atlanta, GA, 2003.

[57] Trispel S, Rau G, Bruderman U. User interactive adaptation of the electrical wheelchair control interface. *Proc. 2nd Int. Conf. Rehabilitation Eng*. Ottawa, Canada, 1984.

[58] Feys P, Romberg A, Ruutiainen J, Davies-Smith A, Jones R, Avizzano CA, Bergamasco M, Ketelaer P. Assistive technology to improve PC interaction for people with intention tremor. *Journal of Rehabilitation Research & Development*. 38: 235-43, 2001.

[59] Aylor J, Ramey R, Schaddegg J, Reger S. Versatile wheelchair control system. *Medical & Biological Engineering & Computing*. 17: 110-4, 1979.

[60] Oppenheim AV, Willsky AS. *Signals and Systems*. 2nd ed. Prentice Hall, New Jersey: 1997.

[61] Horowitz P, Hill W. *The Art of Electronics*. 2nd ed. Cambridge University Press, New York: 1994.

[62] JC 200: Multi-axis inductive joystick. Penny and Giles product specifications sheet.

[63] Karki J. Active low-pass filter design. Texas Instruments. September 2002.

[64] Hsu D, Huang W, Thakor N. On-line adaptive canceling [sic] of pathological tremor for computer pen handwriting. *Proc IEEE*. pgs 113-114, 1995.

[65] Rao R, Seliktar R, Rahman T. Evaluation of an isometric and position joystick in a target acquisition task for individuals with cerebral palsy. *IEEE Trans. Rehab. Eng.* 8: 118-25, 2000.

[66] Stewart HA, Noble G, Seeger BR. Isometric Joystick: A study of control by adolescents and young adults with cerebral palsy. *The Australian Occupational Therapy Journal.* 39: 33-39, 1991.

[67] Spaeth DM. Evaluation of an isometric joystick with control enhancing algorithms for improved driving of electric powered wheelchairs. Doctoral Dissertation, Department of Rehabilitation Science and Technology, University of Pittsburgh, 2002.

[68] Dicianno BE, Spaeth DM, Cooper RA, Fitzgerald SG, Boninger ML. Advancements in power wheelchair joystick technology: Effects of isometric joysticks and signal conditioning on driving performance. *American Journal of Physical Medicine & Rehabilitation*. 85: 631-9, 2006.

[69] Dicianno B, Spaeth DM, Cooper RA, Fitzgerald SG, Boninger ML, Brown KW. Force Control Strategies while Driving Electric Powered Wheelchairs with Isometric and Movement-Sensing Joysticks. *IEEE Transactions on Neural Systems and Rehab Engineering.* Accepted.

[70] Cooper RA. Personal Powered Mobility for Persons with TBI. Dept of Ed Grant Number H133A020502.

[71] Dicianno B. Joystick Use for Virtual Electric Power Wheelchair Driving in Individuals with Spastic Cerebral Palsy. NIH Grant Number K12 HD001097-09.

[72] Brown K, Spaeth D, Cooper R. Designing an Improved Variable Compliance Joystick for Driving Electric Powered Wheelchairs. *Proc 27th Ann Conf. RESNA*. Orlando, FL, 2004.

[73] Spaeth DM, Cooper RA, Puhlman J. Developing a wheelchair joystick with adjustable compliance. *Proc 28th Ann Conf. RESNA*. Atlanta, GA, 2005.

[74] Widman LM, Design and construction of a force-sensing joystick controller for a power wheelchair. Master's Thesis. California State University, Sacramento. 1992.

[75] Chaffin DB, Andersson GBJ, Martin BJ. *Occupational Biomechanics*. 3rd ed. John Wiley & Sons, New York: 1999.

[76] Popov I. *Engineering Mechanics of Solids.* Prentice Hall, New Jersey: 1999.

[77] Ding D, Cooper RA, Spaeth D. Isometric joystick tuning interface and assessment. *Proc 27th Ann Conf RESNA*. Orlando, FL, 2004.

[78] Brown K, Spaeth D, Cooper R. Characterization of a position sensing joystick. *Proc. 29th Ann Conf. RESNA*. Atlanta, GA, 2006.

[79] Arihara M, Sakamoto K. Contribution of motor unit activity enhanced by acute fatigue to physiological tremor of finger. *Electromyogr Clin Neurophysiol* 39: 235–247, 1999.

[80] Morrison S, Kavanagh J, Obst SJ, Irwin J, Haseler LJ. The effects of unilateral muscle fatigue on bilateral physiological tremor. *Experimental Brain Research*. 167: 609-21, 2005.

[81] Spaeth DM, Cooper RA. Isometric Controls with Personalized Algorithms for Driving Electric Powered Wheelchairs. Dept of Vet Affairs. Grant No. B3287R.

[82] Hasdai A, Jessel AS, Weiss PL. Use of a computer simulator for training children with disabilities in the operation of a powered wheelchair. *Am J Occup Ther.* 52: 215–20, 1997.

[83] Mahajan HP, Spaeth DM, Bevly AJ, III, Ding D, Cooper RA. A wheelchair driving simulator for people with low visual attention span. *Proc. 29th Ann Conf. RESNA.* Atlanta, GA, 2006.

[84] Jonkers I, Nuyens G, Seghers J, Nuttin M, Spaepen A. Muscular effort in multiple sclerosis patients during powered wheelchair manoeuvres. *Clinical Biomechanics*. 19: 929-38, 2004.

[85] Accot J, Zhai S. Performance evaluation of input devices in trajectory-based tasks: An application of The Steering Law. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. Pittsburgh, PA, pgs. 466-472, 1999.

[86] Blanchard P, Devaney RL, Hall GR. *Differential Equations.* Brooks/Cole Publishing Co., Boston, MA: 1996.

[87] Burden RL, Faires JD. *Numerical Analysis*. 6th ed. Brooks/Cole Pub. Co., Cincinnati: 1997.