A Force-controlled Indenter to Elicit a Tactile Afferent Response that Accommodates for the Skin's Viscoelastic Relaxation and Differing Thickness

_____

A Thesis
Presented to
the faculty of the School of Engineering and Applied Science
University of Virginia

_____

In partial fulfillment
of the requirements for the degree

Master of Science

by

Sean Lawrence Gallahan

May
2015

Approval Sheet

The thesis
is submitted in partial fulfillment of the requirements
for the degree of

Master of Science

_____

Author

The thesis has been read and approved by the examining committee:

Gregory Gerling, Ph.D.
_____
Advisor

Amy Laviers, Ph.D.
_____

Reid Bailey, Ph.D.
_____

Ellen Lumpkin, Ph.D.
_____

Accepted for the School of Engineering and Applied Sciences:

Dean, School of Engineering and Applied Science

May
2015

TABLE OF CONTENTS

Abstract……4

Introduction……..5

Methodology……...…….6

Results…..……..……20

Discussion……..…….……...25

References……..28

Appendix A……31

Appendix B……32

Appendix C……42

Appendix D……..109

1. Abstract

Tactile afferents elicit neural biopotentials upon deformation of the skin's surface. To study the underlying input-output relationship, typically the stimulus input is displacement, though force may be more naturalistic. Force control is advantageous because equal pressure can be delivered to specimens that vary in skin thickness and it can counterbalance the skin's viscoelastic relaxation. However, current force-controlled indenters approach an afferent's receptive field at an angle so as to create an unequal stress distribution and base their feedback control indirectly on motor performance as opposed to directly measuring probe force. The proposed research seeks to address these issues by designing, building, and evaluating a single-axis indenter. The indenter consists of hardware (motion controller coupled with a mechanical sled and custom designed electronics) and software (control algorithm and user interface). The control algorithm utilizes a sequential, pre-planned trajectory and then a real-time feedback technique to counterbalance the three phases of skin relaxation that accompany the early hold (< .19 sec), intermediate hold (>~0.19 sec and <~1.5 sec) and late hold (>~1.5 sec) of the stimulus. The graphical user interface (Python software) traverses a user intuitively through a series of experimental tasks. With collaborators at Columbia University, we conducted two experiments. The results of the experiments indicate the indenter 1) can clamp an achieved force in skins of differing thickness and 2) evoke neurophysiological responses of slowly adapting afferents whereby the firing rate drop between the early and late phases of the sustained hold is less during force control, leaving mostly neural adaptation.

2. Introduction

Our sense of touch is informed by the responses of tactile afferents embedded in the skin. The patterns of spiking response differ according to afferent type (Johnson, 2001). For example, slowly adapting afferents will respond to both moving ramp and sustained hold phases of a stimulus. In experiments, what is typically held sustained is the displacement, or position. This is displacement control. Two phenomena are observed when displacement control is employed; the skin rapidly decays at a double exponential rate (at least) during the static phase (Williams et. al, 2010 and Pubols and Benkich, 1986) and differences in skin thickness can lead to unequally delivered sets of stimuli (Wang et al, 2013). Given the first phenomena, it is hard to decouple if the observed decay or adaptation in the spike firing of the neural response is due to skin relaxation or neuronal mechanisms. Given the second, changes in skin thickness of 10s to 100s of microns between specimens means that the same set of displacements will elicit a different sensitivity function, which does not allow for accurate cross comparison (Wang et al, 2013). Furthermore, it is unclear whether humans (and other mammals) actively attempt to control force over displacement, but force control seems more likely (Srinivasan and Lamonte, 1995 and Tiest and Kappers, 2009).

At present, there are few compressive, force controlled indenters on the market. One off-the-shelf model, the 300 series (Aurora Scientific Inc, Canada) was designed for the application of tension to muscles, but has also been configured for compression to the tactile end organs (Jang, 2009 and Levy and Strassman, 2002). While this indenter has a ramp up time of less than 200 msec at the lowest velocity setting and can switch between force and displacement control, its rotary lever mechanism can create an unequal state of stress at the skin surface. Secondly, there is little documentation on the readout or accuracy of its force clamp, and its feedback

control is based indirectly on motor performance as opposed to direct measurement of force at the probe tip. Another alternative force control indenter has likewise not reported the real-time measurement of the force clamp, and has typically been used at larger forces around and above 4 Newtons (Johansson, 2004).

To address the aforementioned issues of skin phenomena and other equipment, this work focuses upon the design, construction, and evaluation of a compressive force-controlled indenter.

3. Methodology

The Virginia force-controlled indenter was built to deliver ramp and hold stimuli to afferent end organs of the skin. The design metrics include delivering forces over the range of 1 – 2500 mN with a 1.5 mm tip, at ramp durations of at or less than 600 msec to 100 mN force, and achieving force clamps with error of less than 5% of that achieved. Off-the-shelf and custom-built components were used, and major sub-systems include (as *described in Section 3.2.*): *Actuators, Mounts and Sensors, Control Algorithms, User Controls and Displays*, and *Integration and Electronics Interfacing*.

As indicated in Fig. 1, a series of transformations are inherent in the indenter's operation. Displacements and/or forces are commanded at the graphical user interface. Control algorithms work with the motion controller to move the vertical sled. The position of the probe tip at a cantilever attached the vertical sled is monitored by laser displacement sensor and load cell and recorded by an analog-to-digital converter. In force-control mode, probe measurements are relayed back to the control algorithms and motion controller for position updates of the vertical sled, to accommodate for skin relaxation and maintain constant force. The force control algorithm splits into two parts: a pre-programmed trajectory to reach the desired force and clamp

the skin's early decay and feedback control to counteract the skin's later decay. The extremely rapid and large magnitude of the skin's relaxation (described in detail in Section 3.1) calls for the utilization of the pre-programmed trajectory.

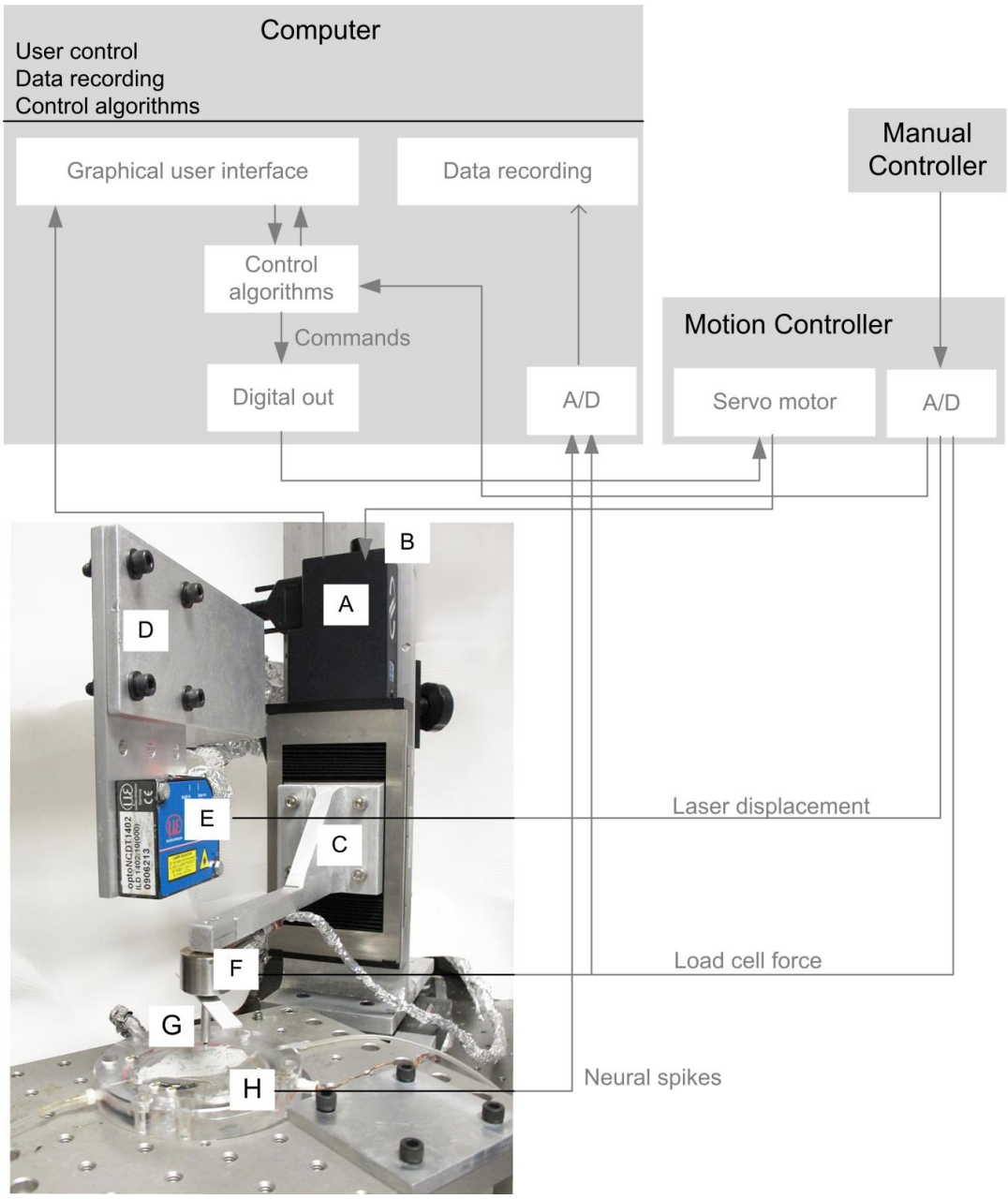

Figure 1: Components of and data flow between the indenter's sub-systems. In particular, the actuators, sensors and mount consist of a motion sled (A), t-beam (B), cantilever (C), laser cantilever and height adjust (D), laser (E), load cell (F), magnetic base, tip (G), and prep stage and voltage reading equipment (H).

To test the indenter's performance, two sets of experiments were run. Experiment 1 compared displacement- and force-controlled indentations into mouse skin of differing thickness, from two animals with three body sites, at a ramp velocity of 0.5 mm/secec. Experiment 2 examined an ex vivo neural recording from a slowly adapting fiber in the skin of one mouse.

3.1 Exploration into force decay as a result of skin relaxation

An understanding of both the magnitude and timing of the skin's relaxation is a vital step to precede design. As such, force versus time data for twenty-one indentations into ex vivo mouse skin, across five specimens, were conducted as described in Wellnitz (2010). The skin was pinned on top of a silicone elastomer (Sylgard, Dow Corning) and nylon (L'eggs winter-weight pantyhose). Indentations spanned across a range of peak forces, from 66 to 660 mN with a 1.5 mm tip. These displacement-controlled experiments used ramp up acceleration of 6.35 mm/sec^2 and velocity of 0.5 mm/sec. The results indicate that the skin's relaxation best fits a triple exponential function where the medians of the three time constants, across the twenty-one indentations, are 189, 1050, 136,119 msec, respectively. One example indentation and fitting is shown in Fig. 2. The results also demonstrate that as peak force increases, the time constants (τ) also increase. Based on these findings, we determined that the indenter, in feedback control mode only, with a timing cycle of 20-30 msec per movement as well as the need to stop between each movement as a result of the controller's sigmoidal displacements, could not keep up with the magnitude of the skin's rapid decay. Therefore, we developed a pre-programmed trajectory method to clamp this rapid early decay.

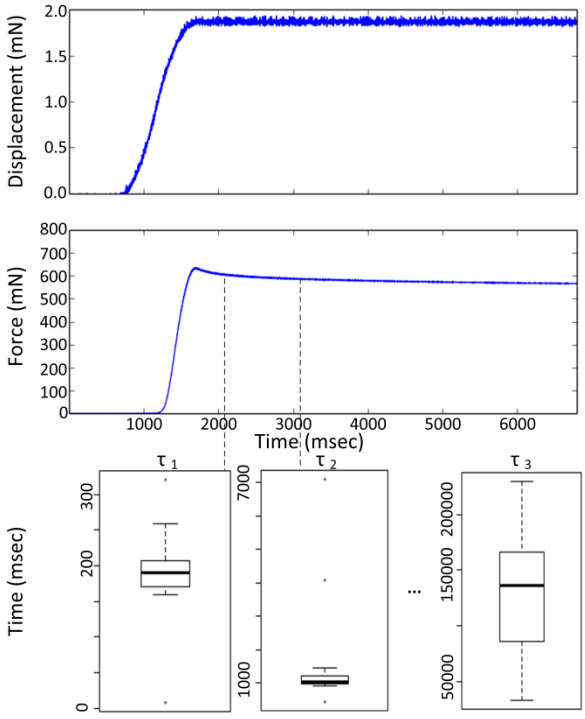

Figure 2: An example of the skin's relaxation under displacement control. The upper trace shows that displacement is clamped to 1.9 mm resulting in a peak force of about 660 mN in the middle trace, which decays in the steady-state to about 14% of the peak force. Triple exponential functions were fitted to force traces, starting at their peak force, over magnitudes from 66 mN to 660 mN. For example, when fit to a triple exponential function, the force trace above yields time constants of 255, 1470, and 229,156 msec. The time constant τ_3 is not annotated in the figure because it extends well past the 5 second hold period.

3.2 Apparatus

Actuators, Mount, and Sensors

Actuators, sensors and mount components respond to commands from the motion controller, monitor the sensors (e.g., laser and load cell), and fasten the actuators and sensors to the Faraday cage on a vibration free table. A single 50 mm length DC motor driven sled (50 cc UTS Mid-Travel Steel Linear Stages, Newport Corporation, USA) drives the indenter tip in the z-direction into the skin prep (Fig. 1, mark A). The sled is constrained to a maximum velocity of 40 mm/sec, a maximum acceleration of 160 mm/sec^2 and a minimum movement of 0.3 micrometers. Two x- and y- direction actuated stages (100 mm travel, Siskiyou Inc., Oregon),

oriented perpendicular to each other, position the probe tip lateral to the surface of the prep between experiments.

For stability, the sled is held upright by an aluminum t-beam (length = 92.0 mm, width = 63.5 mm, height = 381.0 mm) welded to an aluminum base (length = 177.8 mm, width = 101.6 mm, height = 12.7 mm) (Fig. 1, mark B; and Appendix B). The base of the t-beam connects directly to the x-y stages and sit atop a low profile magnetic base with a 300 N vertical maximum force threshold (Thorlabs, New Jersey). Attached perpendicular to the sled's moving platform is a lightweight, aluminum cantilever arm (length = 190.5 mm, width = 12.7 mm, height = 12.7 mm to 76.2 mm, base = 75.0 by 75.0 mm) (Fig. 1, mark C).

A load cell (Model 31, Honeywell, USA), aluminum sheath and ceramic tip (Fig. 1, mark F and mark G) are positioned at the end of the cantilever arm. The load cell operates within a maximum load of 250 grams (~2.5 Newtons) with a resolution of +- 0.5 mN and outputs analog voltage linearly correlated to force. The ceramic tip (10.0 mm length, 1.6 mm diameter), with a filleted end, connects to the load cell via an aluminum sheath (22.5 mm length, 3.2 mm diameter). The aluminum sheath and ceramic tip were carefully designed to limit vibrational noise through reduction of mass and volume and collectively weigh 4.1 grams. Located on the back of the base is a large steel shaft that connects to a holster and angled beam to hold the displacement laser (OptoNCDT1402, Micro-Epsilon, Germany) with a 10.0 mm range and 1.0 um resolution directly over top of a small rectangular piece of aluminum and measures displacement of the ceramic tip (Fig. 1, mark E).

3.2.2 Control Algorithms

The control algorithms command those displacements, forces, velocities and accelerations specified at the user interface, send commands to the motion controller, and receive data from sensors. The mechanical indenter employs separate algorithms for displacement and force control.

3.2.2.2 Displacement Control

Under displacement control (Fig. 2, top), the probe tip is commanded to a position into the skin relative to their point of contact. The three input variables are displacement, velocity, and hold duration.

3.2.2.3 Force Control

Under force control, the sled is commanded to a force and then achieved force is measured and fed back to the algorithm by the load cell. To accomplish the force clamp, the algorithm utilizes two temporally successive tables concatenated into a single trajectory, and a PID feedback loop (Fig. 3). A trajectory is a pre-planned sequence of positions by which to move the tip over time and at a certain rates. The trajectory consists of two separate components: 1) over the ramp up (~600 msec duration), a look up table to move to a displacement paired to a commanded force and 2) over the first and second time constant of the triple exponential decay (from start of force clamp to ~1.5 s), a series of discrete movements every 1 ms to clamp the force. The trajectories are populated via a series of calibrations (described below) that correspond to levels of force.

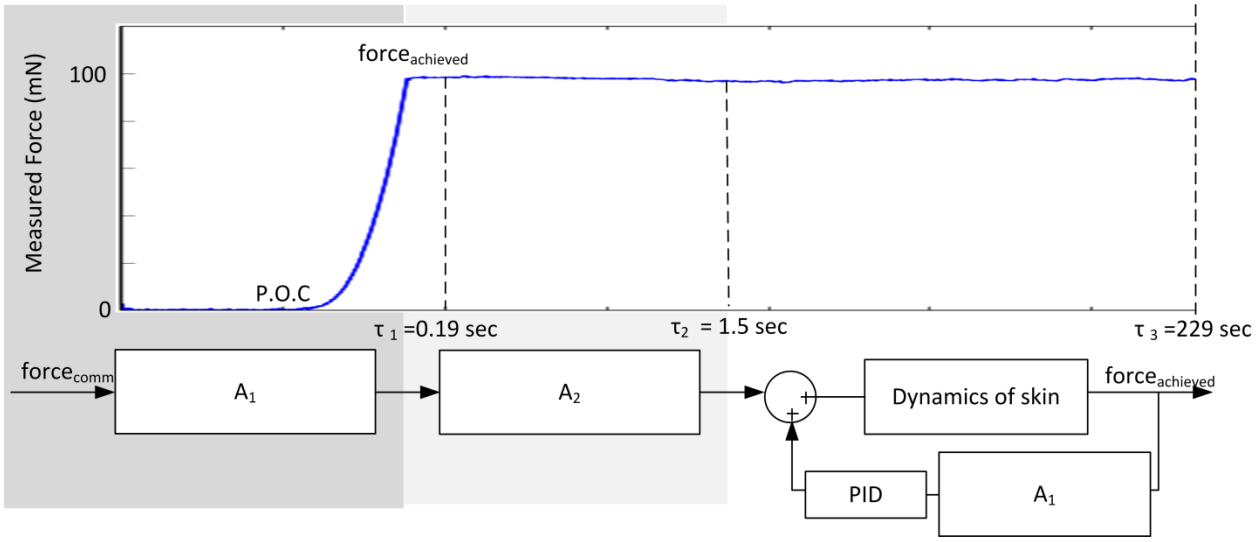

Figure 3: The force control strategy involves three main steps. The first two steps consist of a pre-planned trajectory based on prior calibration experiments. In particular, the A_1 table is a force-to-displacement look-up that accounts for the displacements neeeded over the course of the stimulus ramp to achieve a desired peak force. The A_1 table is built up as indicated in Fig. 4. In addition to its use in guiding motion over the course of the ramp, it is also used to help build the A_2 look-up table. Then, the A_2 table is a second force-to-displacment look-up table that accounts for the displacements needed from the end of the A_1 table's ramp phase until about τ_2=1.5 seconds, a second set of displacements are configured, annotated. The A_2 table is built up as indicated in Fig. 5. The third step is a PID feedback loop that converts the error in force back into a displacement to counteract the slower decay late in the hold. The feedback loop runs until the commanded hold time ends, which may be longer or shorter then τ_3.

The PID feedback loop, following the completion of the trajectory, controls the force clamp to the end of the hold, adjusting for less skin relaxation during the third exponential. In particular, the algorithm adjusts to the current load cell measurement, as the current force may be above or below commanded force. PID multiplication values, represented in Eqns. 1, 2 and 3 with 'mult' subscripts, were found during experimentation into mouse skin on top of sylgard (description in Wang, 2013). Note that dt varies depending on the loop time of each feedback iteration. Accurate P_{mult}, I_{mult}, and D_{mult} vary greatly based on skin thickness, and the interface allows the user to adjust PID values according to perceived skin thickness, via a continuous slider with roughly three levels of 'thin' (P_{mult} = .0005, D_{mult} = .000005, I_{mult} = .0005), 'normal' (P_{mult} = .005, D_{mult} = .00005, I_{mult} = .0005) and 'thick' (P_{mult} = .05, D_{mult} = .0005, I_{mult} = .0005). These functions then combine to be sled movement, Eqn. 4, to adjust for the decay.

$P_{error} = P_{mult}*error_{current}$ (1)

$I_{error} = I_{mult} * Integrator$ (Integrator = $Integrator_{prev} + Integrator*dt$) (2)

$D_{error} = D_{mult} * (error_{current} - error_{prev})/dt$ (3)

Movement (mm) = $P_{error} + I_{error} + D_{error}$ (4)

3.2.2.3.1 Calibration

A two-step calibration procedure is used to create a trajectory. The first step corresponds to the A_1 force-to-displacement table that accounts for movements over the stimulus ramp, and the second step corresponds to the A_2 force-to-displacement table that accounts for movements to counteract skin relaxation over the stimulus hold. In particular, for the first step of calibration, the A_1 force-to-displacement table, displacement from the motion controller and force from the load cell (e.g., d_{max}=0.18 mm to f_{max}=50 mN) at a rate of 10 kHz are coupled to form a series of relationships over the duration of the stimulus ramp. This series of force and displacement relationships are fitted by an exponential function (Fig. 4, A_1 : eq. 5), which will in later steps convert fitted force decay back to displacement.

Force (mN) = $ae^{b*displacement(mm)}+c$ (5)

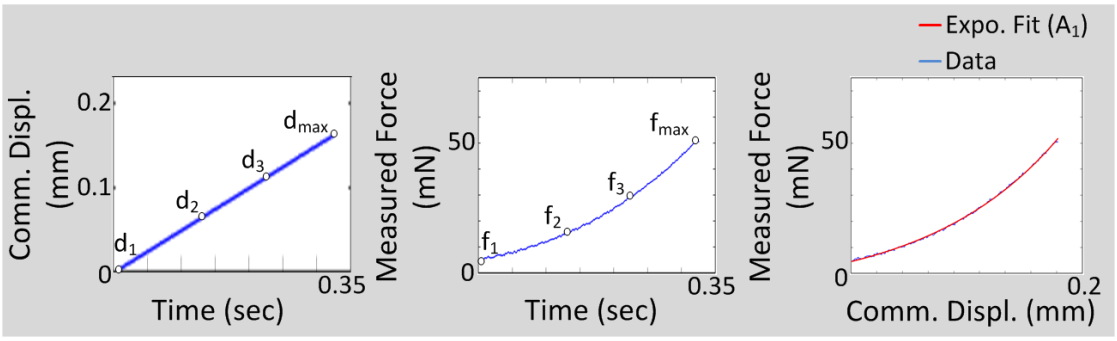

Figure 4: To build up the A_1 force-to-displacement table that accounts for movements over the stimulus ramp, a calibration stimulus is indented into the skin at a slow commanded velocity from the point of contact to a commanded force (f_{max}). When at the commanded force, the corresponding displacement is then saved as d_{max}. During stimulation up to f_{max} (shown above as 50 mN), displacements d_1, d_2, d_3 are recorded with their corresponding force values f_1, f_2, f_3. After mapping the forces to displacements, an exponential function is fitted between the two, which is denoted A_1.

For the second step of calibration, the A_2 force-to-displacement table, a calibration stimulus is indented into the skin to the desired peak force given the parameters from A_1. Then the stimulus is held for 1.5 s to determine the magnitude and timing of the force relaxation at that force magnitude (Fig. 5). Force traces of the skin's relaxation starting at the peak force are fit with a triple decay exponential function (Eqn. 6, Fig. 4).

$$\text{Skin Relaxation (mN)} = Ae^{-B(t-t_0)} + Ce^{-D(t-t_0)} + Fe^{-G(t-t_0)} \quad (6)$$

$$\text{Displacement (mm)} = -\ln\left((Ae^{-B(t-t_0)} + Ce^{-D(t-t_0)} + Fe^{-G(t-t_0)})/a\right)/b \quad (7)$$

Force fit over the hold of the stimulus by Eqn. 6 can be translated back into displacement using A_1 force-to-displacement table, as had been identified in Eqn. 5. These displacements over time are used to create the A_2 portion of the trajectory (Eqn. 7), calculated in intervals of 1 msec. In particular, we take displacement at time t and subtract the displacement at the previous millisecond for a total of 1500 movements over the 1.5 s time period. The velocity of each

displacement is the derivative of Eqn. 3 at each time t. This velocity determines the smoothness of movement between each point as a result of the acceleration to achieve each velocity during each time interval. Note that between each successive ramp-and-hold indentation is a sleep period of 60 s to allow the skin to relax completely to its original state.

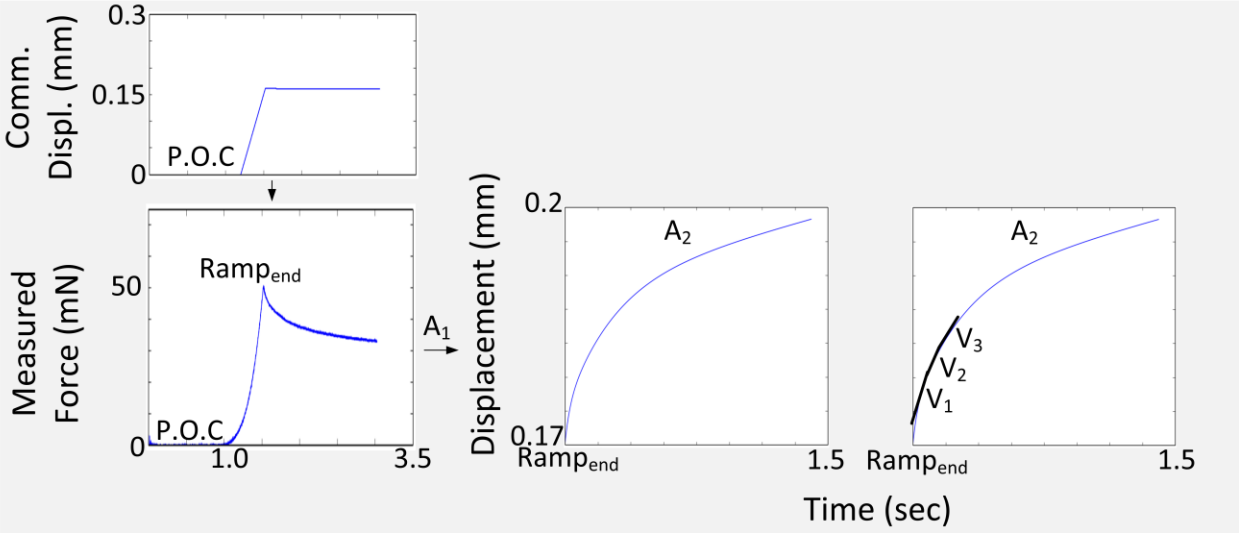

Figure 5: To build up the A_2 force-to-displacement table that accounts for movements to counteract skin relaxation over the stimulus hold, a calibration stimulus is indented into the skin to the desired peak force ($Ramp_{end}$) given the parameters from A_1. Then the stimulus is held for 1-5 seconds to determine the magnitude and timing of the force relaxation at that force magnitude. In the center figure, given the parameters from A_1, the A_2 force-to-displacement table is built. In the rightmost figure, from this trace of displacements, velocity is calculated at each millisecond.

3.2.3 User Controls and Displays

The system consists of physical controls, software controls and data displays that give the user the ability to manually adjust the z axis for probe-to-skin contact, manipulate system variables, position the actuator in the x and y axis, command control algorithms and collect data. A custom built, three position joystick (Fig. A-3) gives the user the ability to manually actuate the z-stage up, down or idle it at a commanded velocity once activated in the user interface. A potentiometer, protruding from the electronics box, manipulates the operating range of the force

output from the load cell to increase or decrease force clarity. A separate joystick positions the entire mount system via x-y axis stages (Siskiyou, Inc.). The user sends commands to control algorithms (manual, force, and displacement control) through a custom built graphical user interface (Python).

3.2.3.1 Graphical User Interface

The graphical user interface walks the user through the indentation protocol for displacement and force control indentations. The interface splits into two panels: the left panel leads the user through the 5 or 6 steps while the right panel provides a log history, prior indentation status, and continuous feedback from the load cell (Fig. 6).

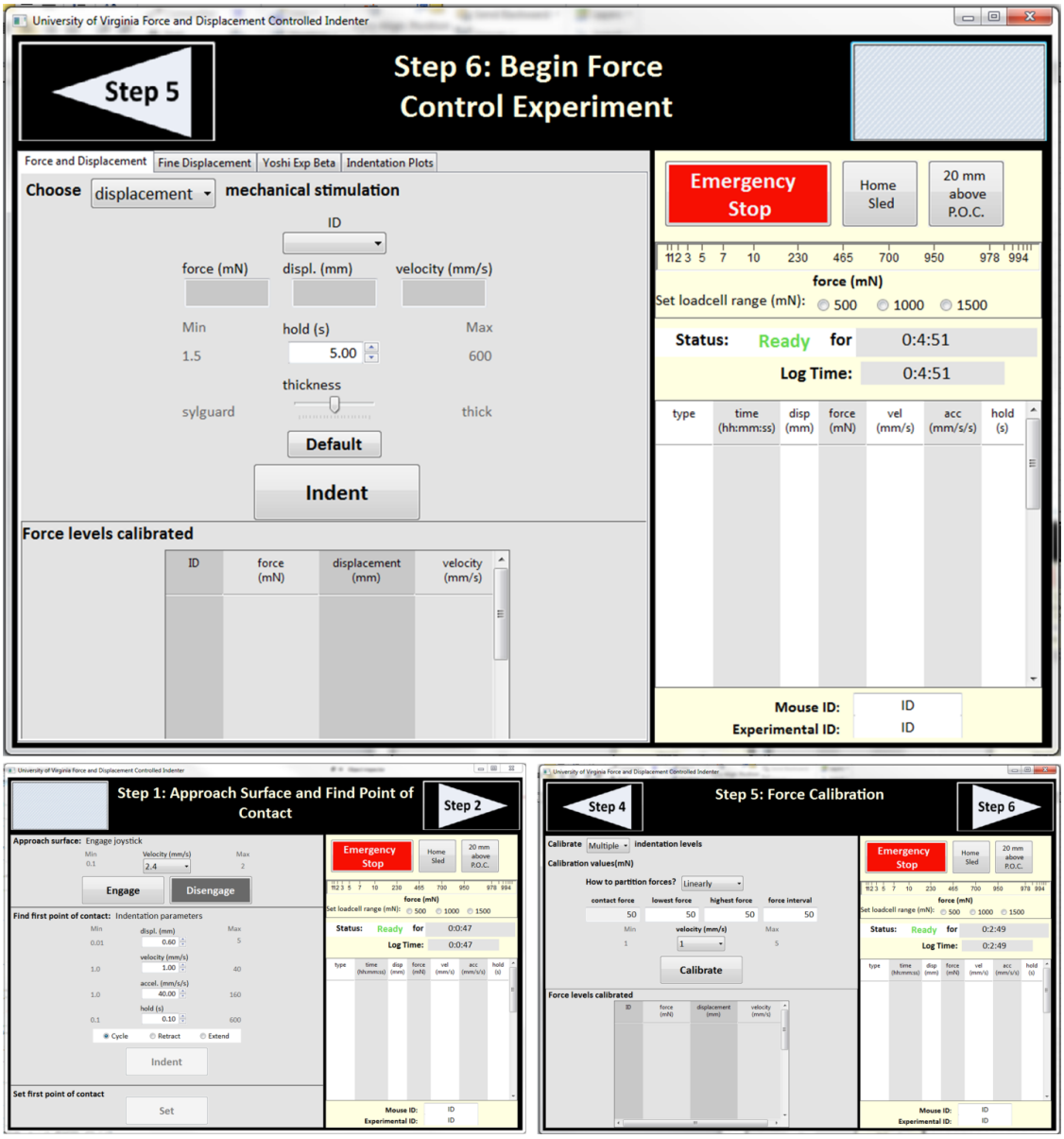

Figure 6: To afford experiments in modes of both force and displacement control, the graphical user interface is based on a series of sequential steps. Each step has an overall goal. For example, step 1 allows the user to position the tip at the surface of the skin and define the first point of contact (bottom left). Before conducting a force controlled stimulation (top), the user chooses a range of forces over which to calibrate (bottom right). Each stimulation is logged in the panel to the right of each screenshot.

Steps 1 (Fig. 6, lower left) through 3 (not shown) allow the user to define the point of contact with the skin and set the lower threshold and saturation forces for an afferent. Step 4 allows the user to choose between experiments in force or displacement control and depending on that choice, step 5 runs displacement control and steps 5 and 6 calibrate and run force control

(Fig. 6, center and lower right). To validate the design of the interface, the interface went through heuristic evaluations and numerous end-user reviews.

3.2.3.2 Data Recording

Recording software (LabCharts, AD Instruments), located on the same computer as the user interface, connects to a standalone A/D converter (PowerLab Systems 16/35, AD Instruments) through a data translation board (DT3016) and records force (mV), laser displacement (mV) and neural potential (mV) data. The software samples data at 20,000 Hz, displays data to the user, and saves data as a Matlab file for later post-experimental analysis.

3.2.4 Integration and Electronics Interfacing

Integrators and electronic interface communicate commands to the sled via the motion controller, power all sensors and the joystick, and digitize feedback and sensor information that is then sent to the control algorithms. Integration and electronics interfacing splits into three components: the motion controller, a custom built electronics box (Fig. A-1,2), and an A/D converter. The motion controller, the XPS model Q8 (Newport Corporation, USA) drives the vertically oriented sled. This motion controller, however, only controls displacement, its rate and acceleration. Therefore, algorithms have been built to modify its function (as described in the section, *Control Algorithms*). Sensor input for the control algorithms (load cell, laser displacement sensor and operator's manual joystick feedback) pass through a custom built electronics box. The custom built circuitry box powers each sensor and joystick using a global switching power supply (GPM55BG, Condor) and transmits the joystick and displacement laser voltage while amplifying the load cell at a rate dictated by the potentiometer resistance, splitting into the motion controller

and A/D converter (Powerlab Systems 16/35, AD Instruments), and filtering at a rate of 500 and 5,000 Hz.

3.3 Experimental Methods

To test the indenter's performance in both displacement and force control, two experiments were employed: 1) multiple indentations of different stimulus magnitudes into skin tissue of varying thickness and 2) indentation in an ex vivo neural recording from a slowly adapting fiber in the skin of one mouse.

Experiment 1 tested the mechanical indenter's ability to hold different forces in different skin thickness. The indentations consisted of five force levels (10, 20, 50, 100 and 200 mN) at velocity of 0.5 mm/sec with an acceleration of 80 mm/sec^2 by three iterations each and their corresponding displacements were applied in a random order. This added up to a total of 60 indentations. The indentations were delivered to the ex vivo skin of two, 10 week old animals, in three different locations each, leg, belly and back. These skin locations exhibited visually different thickness. The skin was draped on top of a petri dish of silicone-elastomer (Sylgard, Dow Corning) of 0.25 inches thick, neo-hookean hyper-elasticity, with a piece of wet nylon (L'eggs winter-weight pantyhose) between the skin and silicone-elastomer.

In addition to qualitatively evaluating the force clamps individually and their consistency between repetitions, two metrics employed were across the entirety of the hold, the achieved force was subtracted point by point from the commanded force (Eqn. 8) and the percent difference between commanded and achieved force at the first point of the force clamp (Eqn. 9).

Ave. dev. of $F_{ach.}$ from the F_{peak} over hold (mN) = $\frac{1}{N} * \sum abs(F_{ach.} - F_{peak})$ (8)

Percent difference between $force_{achieved}$ from $force_{comm.}$ = $\frac{|(F_{ac\square ieved} - F_{comm.})|}{F_{com}}$ (9)

Experiment 2 evaluated the AM afferent's response to force control. One mouse was used of 8 to 10 weeks old. One force level, 300 mN, aimed to cover a wide range in the afferent response, and at a velocity of 1 mm/sec.

All data was analyzed using the Matlab extension of Python. Data for experiment 1 output as a .txt file and contains load cell output from the motion controller's A/D and displacement read from the sled. Data for Experiment 2 were output as .mat files from a separate recording software Lab Charts and contains load cell, displacement laser, and the action potential voltage over time as recorded from the neuron.

4. Results

Fig. 7 shows a series of displacement and force controlled indentations from Experiment 1. Visually, the force control in the central Fig. approximately hold the achieved force with slight dips after the trajectory (>1.5 seconds), although the achieved force undershoots that commanded. In Fig. 8, upper, achieved forces of 50, 100, and 200 mN holds all stayed below 5% of the commanded force and the majority of 20 mN was below 5%, over the duration of the hold. Holding 10 mN below +-5% is at the range of the load cells clarity. In Fig. 8, lower, as force increased the deviation of the achieved from the commanded force improved (Fig. 8, bottom).

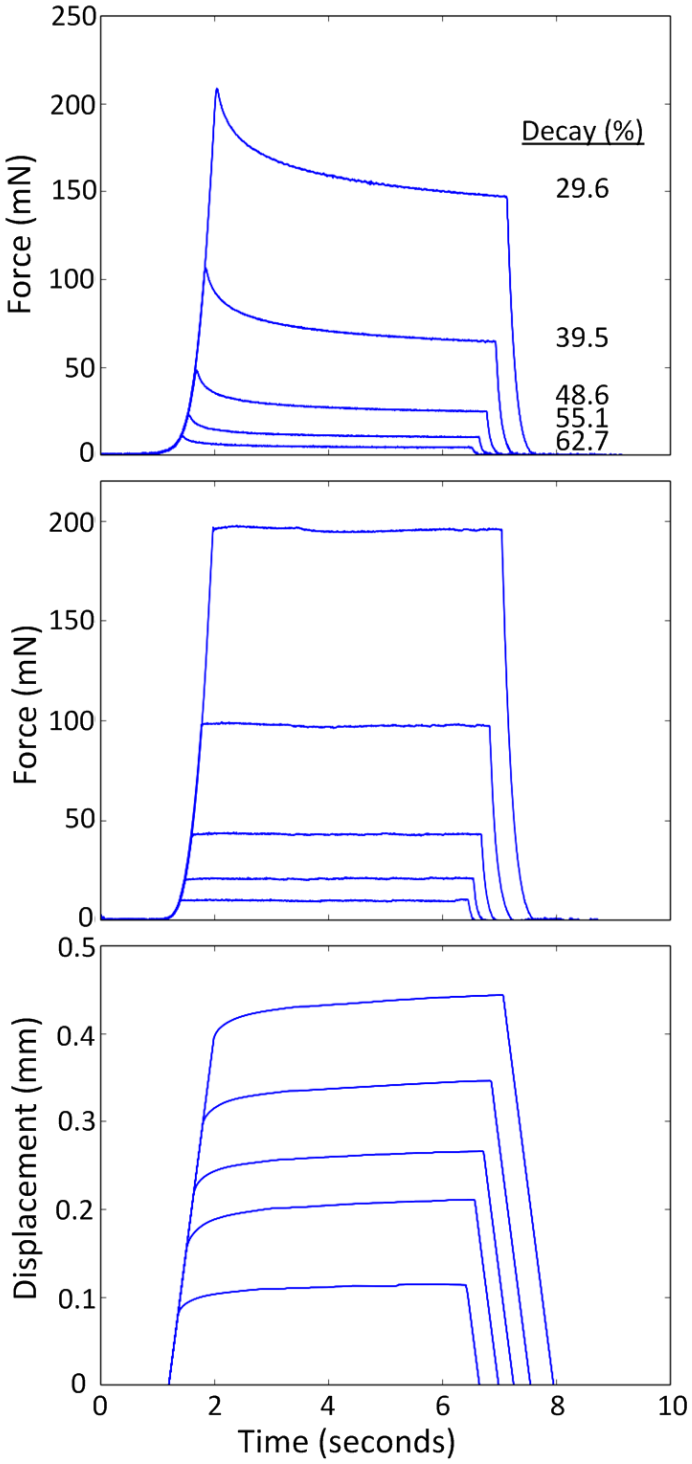

Figure 7: In the upper figure are a set of displacement controlled indentations, where the displacements commanded coincided with peak forces of about 10, 20, 50, 100 and 200 mN. Force decays from peak to steady state of between 63% when peak force is 10 mN. At a peak force of 200 mN the force decreases by 30% . In the lower two figures are a set of force controlled indentations, with the force clamp shown in the middle figure and the increasing displacement of the tip needed to maintain that force clamp in the lower figure.

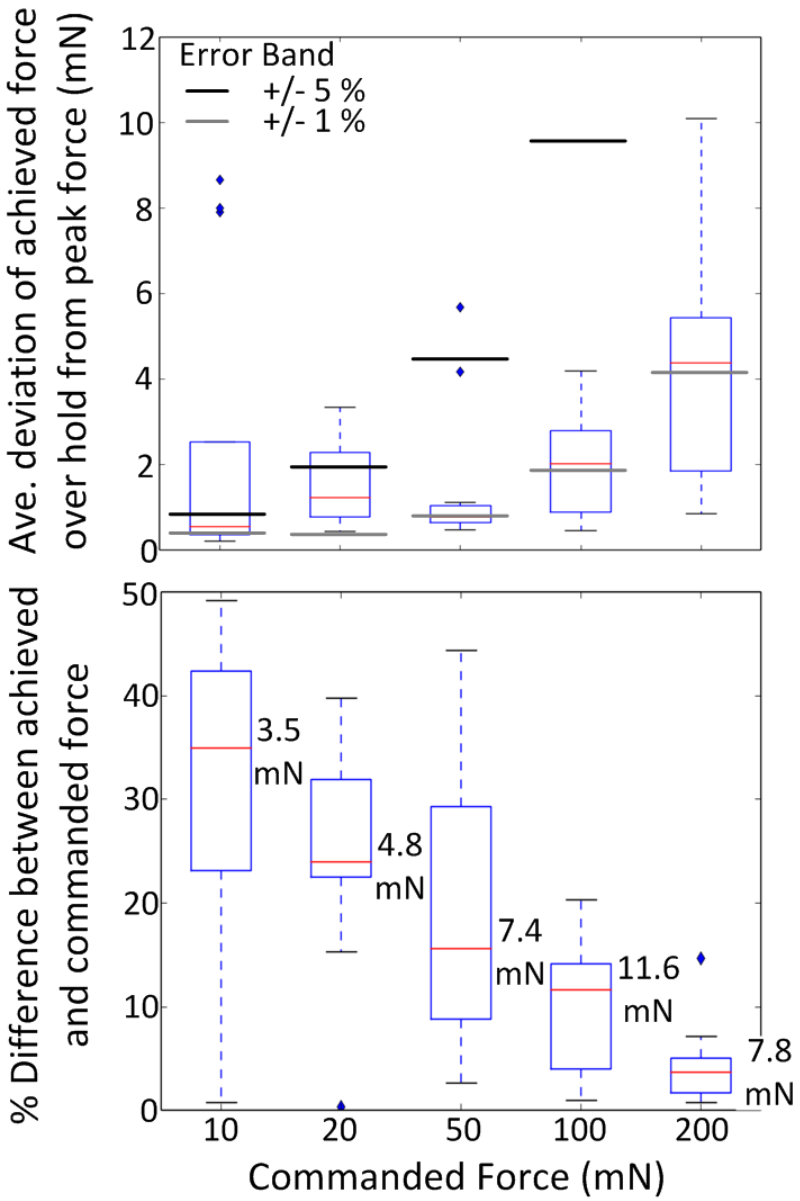

Figure 8: In the top figure, the achieved force was subtracted point by point from the commanded force across the entirety of the hold. It has a positive trend with an increase in force, as well as increase in spread. However, achieved force levels at 10, 20, and 50 mN all have medians of less than +/- 5% error bar from that commanded. In the lower figure, in force control mode the percent difference between commanded and achieved force has a negative trend with increasing force. Both the maximum and minimum, including outliers, are labeled in mN. In both figures there are 12 iterations per force level across 2 different mice and 3 different body sites per mouse.

Fig. 9 shows three successive indentations at each of five force levels into the same piece and location on a skin prep. Despite a repeated displacement trace (top), the achieved force for the first indentation is closer to commanded force, where indentation repetitions 2 and 3 fall 3 – 15 mN short of the first indentation. Possible reasoning behind the phenomenon to be further explored in the discussion.

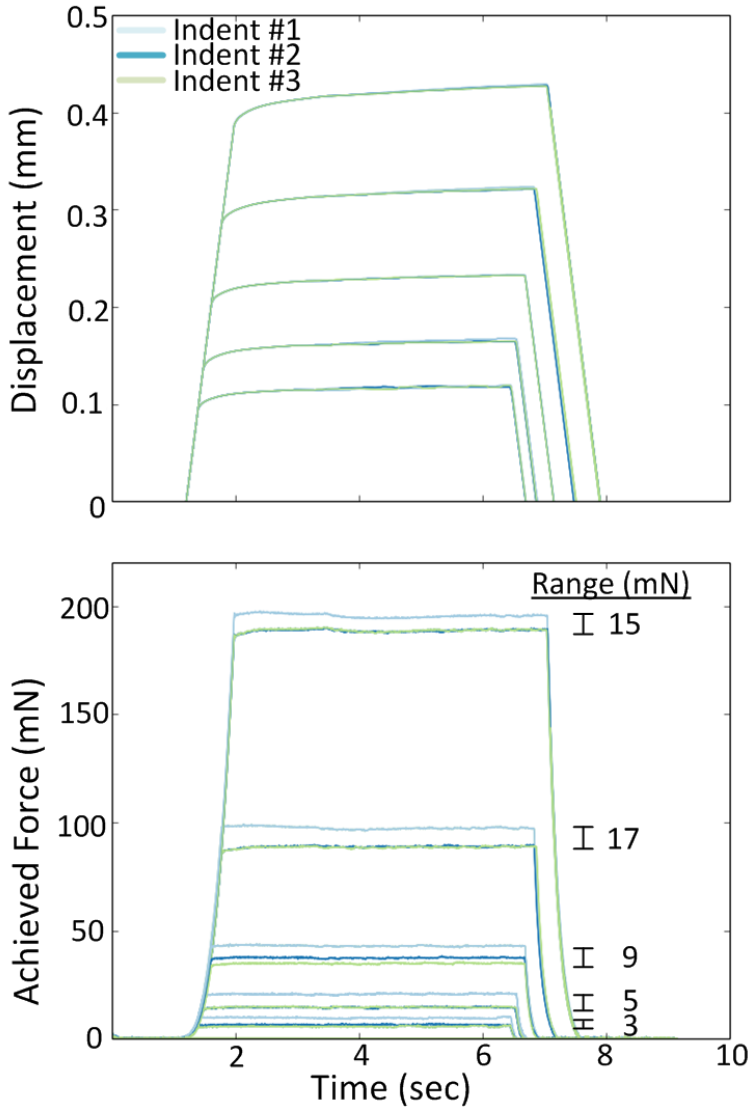

Figure 9: Force control indentations at five levels of commanded force (10, 20, 50, 100, and 200 mN) with three iterations per level in random order for a total of 15 indentations. While the first set of indentations clamped forces near that commanded, the second and third set of force clamps were noticeably lower than those commanded. The range of achieved force per level of force is listed on the lower image.

Fig. 10 shows indenter's ability to deliver approximately the same force regardless of skin thickness (Fig. 10).

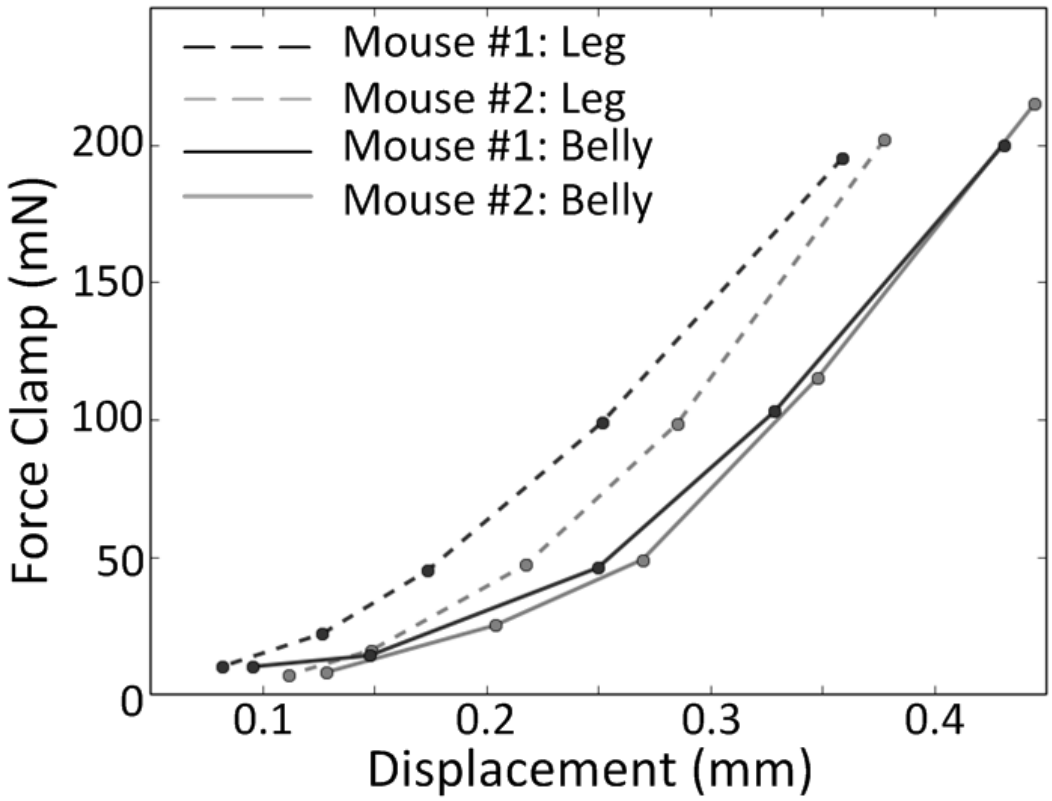

Figure 10: The observed force-displacement relationship differs between animals and body sites. This is likely due to skin thickness, and is in alignment with observations in Wang, et. al (2013). This experiment consisted of 5 different force levels (10, 20, 50, 100, and 200 mN) across two mice and 2 different body sites for a total of 20 indentations.

Fig. 11 demonstrates the response of a slowly adapting afferent in both force and displacement control. Displacement control at 300 mN elicited a higher differential spike firing, over the early hold of the stimulus as compared to the sustained hold of the stimulus, than force control. The force control indentations maintained firing rate across the entire hold in a much more constant fashion.

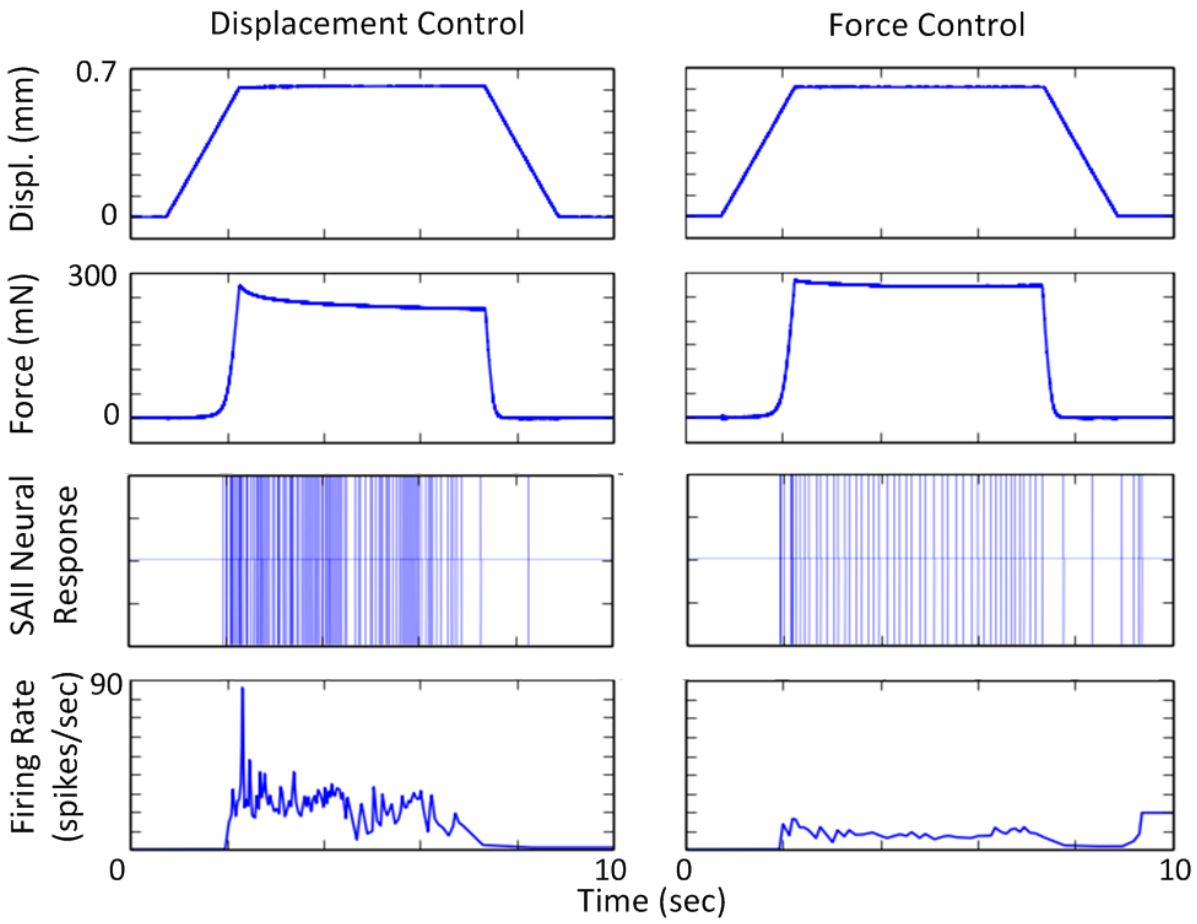

Figure 11: Ex vivo neurophysiological recordings from a mouse AM afferent in both displacement and force control. In panels A and B, commanded force was about 300 mN. The average drop in firing rate from early (<.5 sec) to late hold (3-5 sec) is 30.8 to 17.3 spikes/sec (43%) in displacement control compared to 10.86 and 8.3 spikes/sec (23%) in force control.

5. Discussion

The work herein describes and demonstrates the design and capabilities of a novel force-controlled indenter to operate both force and displacement control, deliver forces over a range of 1-2500 mN with a 1.5 mm tip, ramp into the skin at about 600 msec to 100 mN, and clamping forces to less than 5% error across the hold. Indenting and clamping force on different parts of the mouse and the afferent readings from experiments 1 and 2 demonstrate the mechanical indenter's ability to hold a clamp within 1-5% of the achieved force, ramp to and achieve commanded forces for large force values, and indent into, force clamp, and distinguish skins

with different thicknesses. Studying the neuron revealed the slowly adapting afferent fires as a result of the clamped force and the small displacement adjustments the indenter makes to clamp force don't elicit artificial neural spikes.

Force control indentations are an integral part in studying and understanding neuronal response to stimuli. Force control provides a consistent stimuli to skins of different thicknesses, decouples afferent response from stimuli magnitude and skin relaxation, and it has become more apparent through studies that our body acts and reacts to hold force and not displacement .

The original force control algorithm was entirely PID feedback across the clamp but there were several problems with this approach. Firstly, there was a 50 ms delay between the ramp period and the start of PID. This delay, during the greatest period of decay, drops the force clamp a large percentage off of the commanded force. Secondly, PID control would require a large number of parameters to account for varying velocity, commanded force, skin thickness, and the difference in decay between the early and late hold. Finally, the controller cycle is 20-30 ms between each movement, again leaving time for decay and large, correctional movements in the early hold.

The force control algorithm herein was built around indentation into a mouse skin on top of sylgard substrate with the use of a 1.5 mm ceramic tip; however the mechanical indenter can be manipulated to indent into any substrate, skin and with any tip size. To do so, a study must be run to fit, what is likely, an exponential decay function during displacement control, as described in section 3.1. The results of the study will provide an estimation of the optimal exponential decay fit and the trajectory time constants.

There are a few shortcomings with this force control methodology and stimulator. Firstly, the design of the algorithm and timing between movements required heavily on the design

elements of the motion controller. For example, at max the motion controller can only move 160 $\frac{mm}{s^2}$, which can limit the controller's ability to fit a calculated trajectory. Other stimulators have a similar problem as there is no ideal motion controller with infinite capabilities. If a trajectory can't be completed as a result of passing the maximum acceleration, the interface gives the user opportunity to re-calibrate at a different velocity. Another artifact and shortcoming of the controller is the discrete nature of trajectory movements. Discrete movements can elicit false afferent firings. Large discrete movements occur in other controllers, notably the Aurora 300 series, as a result of large time intervals between movements (Jang et. al, 2009 and Ge and Khalsa, 2002). The time between movements have been set to 1 msec to limit the size of discrete movements and limit neuronal impact. Finally, experiment one and the metric, % early hold error, demonstrated the controller's inability to repeat force control indentations at the same force (Fig. 9). A possible reason is that the skin hasn't been pre-conditioned enough and once pre-conditioned, calibrations would be able to repeatedly achieve commanded force control. Other work has included an inertial component to the force read at the tip (Cheng et al., 2013). To rectify the problem, if an initial force is above or below the commanded force, instead of moving towards that force after the trajectory, the system clamps the achieved force. Such a solution, while not ideal, will prevent excessive movement once the trajectory ends, in order to reach the commanded force. Other possible sources of variance could be the phosphate and SIF solution in chamber the mouse skin rests on, skin slipping from attaching to the tip as well as floating on top of the phosphate solution.

7. References

Aurora Scientific, Canada. (2014). Retrieved from: http://www.aurorascientific.com/product/neuroscience-products/300c-i-mechanical-stimulator/

Cheng, X et al. (2013). A mechatronic platform for studies of tactile sensory system on prosthesis hand. Intelligent Robotics and Applications (vol. 81, issue 2). p.500-508. Retrieved from: http://link.springer.com/chapter/10.1007%2F978-3-642-40852-6_51

Ge, W and Khalsa, P. S. (2002) Encoding of compressive stress during indentation by slowly adapting type I mechanoreceptors in rat hairy skin. J Neurophysiology (vol. 87) p. 1686-1693. Retrieved from: http://jn.physiology.org/content/jn/87/4/1686.full.pdf

Jang, J H. et. al (2009) Nociceptive sensitization by complement C5a and C3a in mouse. Retrieved from: http://www.ncbi.nlm.nih.gov/pubmed/20031321

Johnson, K O. (2001) The roles and functions of cutaneous mechanoreceptors. Current Opinion in Neurobiology (vol. 11, issue 4). p. 455-461. Retrieved from: http://www.sciencedirect.com/science/article/pii/S0959438800002348

Johansson, R. S. (2004) First spikes in ensembles of human tactile afferents code complex spatial fingertip events. Nature Neuroscience (vol. 7) p. 170-177. Retrieved from: http://www.nature.com/neuro/journal/v7/n2/full/nn1177.html

Levy, D and Strassman A. M. (2002) Mechanical response properties of A and C primary afferent neurons innervating the rat intracranial dura. Journal of Neurophysiology (vol. 88, issue 6) p. 3021-3031. Retrieved from: http://jn.physiology.org/content/88/6/3021.long

Pubols, B. H. Jr. and Benkich M.E. (1986). Relations between stimulus force, skin displacement, and discharge characteristics of slowly adapting type one cutaneous mechanoreceptors in glaborous skin of squirrel monkey hand. Somatosensory Research (vol 4, issue 2). p.111-125. Retrieved from: http://www.ncbi.nlm.nih.gov/pubmed/3809832

Srinivasan, M.A. and Lamotte, R.H. (1995). Tactile discrimination of softness. Journal of Neurophysiology (vol. 73, issue 1). p. 88-101. Retrieved from: http://www.ncbi.nlm.nih.gov/pubmed/7714593

Tiest, W.M. and Kappers, A.M. (2009). Cues for Haptic Perception of Compliance. Haptic, IEEE Transactions (vol. 2, issue 4). p. 189-199. Retrieved from: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4906991&tag=1

Wang, Y. et al. (2013) Hyperelastic material properties of mouse skin under compression. Plos one (vol. 8, issue 6). Retrieved from: http://www.plosone.org/article/info%3Adoi%2F10.1371%2Fjournal.pone.0067439

Wang, Y et. al. (2013) Natural variation in skin thickness argues for mechanical stimulus control by force instead of displacement. Joint Eurohaptics Conf Symp Haptic Interface Virtual Environ Teleoper Syst. p. 645-650. Retrieved from: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=06548484&tag=1

Wellnitz, S.A., Lesniak, D.L., Gerling, G. J., and Lumpkin, E.A. (2010). The regularity of sustained firing reveals two populations of slowly adapting touch receptors in mouse hairy skin. Journal of Neurophysiology. 103 (6). 3378-3388 .

Williams, A et al. (2010) Skin relaxation predicts neural firing rate adaptation in SAI touch receptors. Conf proc IEEE Eng Biol Soc. p. 6678-6681. Retried from: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5626264

Appendix A: Circuit diagrams for power supply, electronics box, and manual joystick

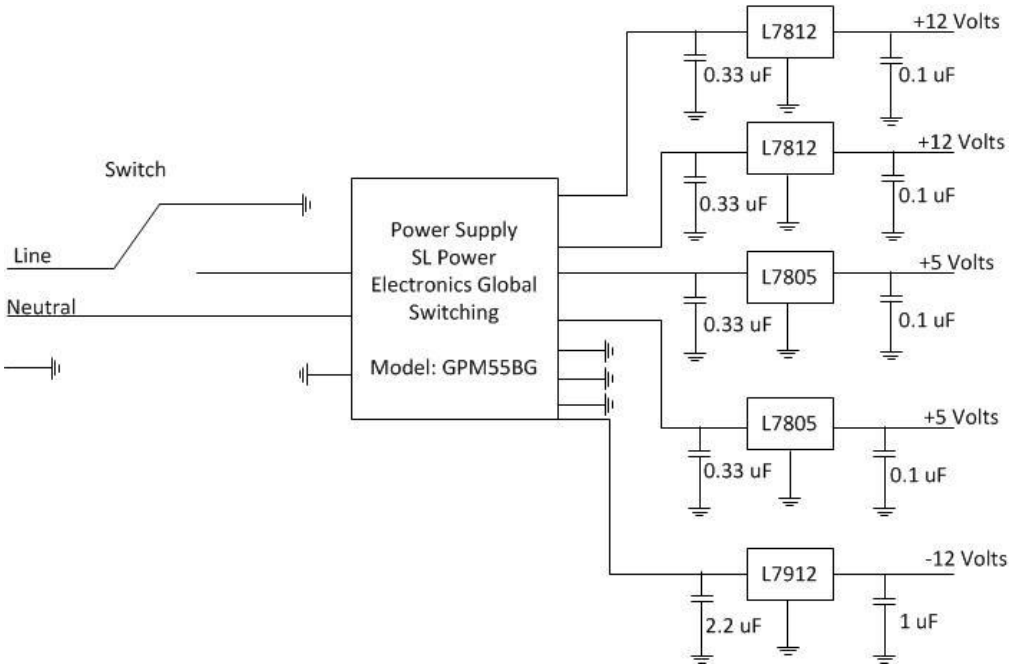

Figure A-1: Power supply and voltage clamps that power the sensors and manual joystick.

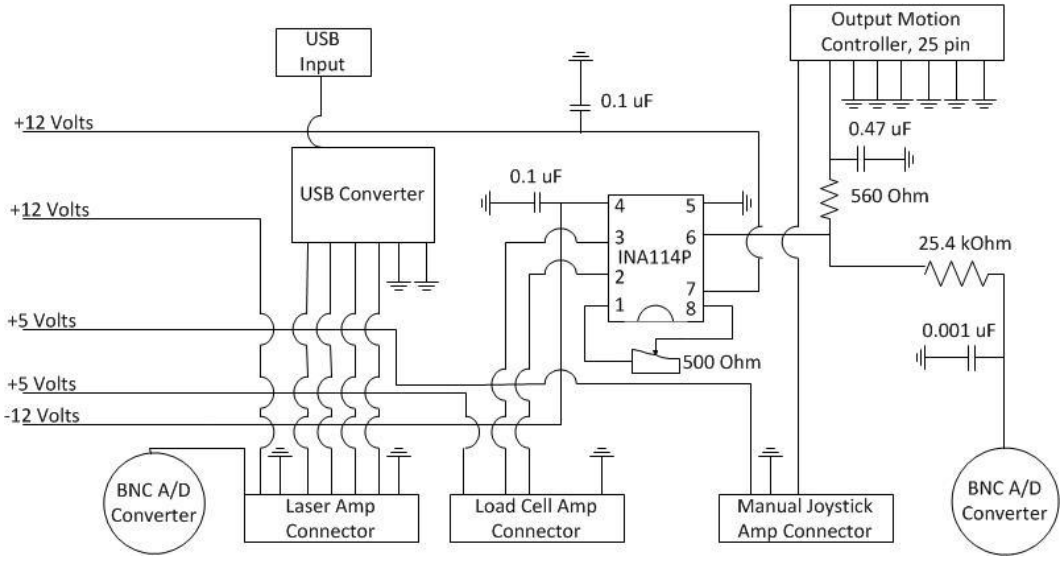

Figure A-2: Connections to the sensors and manual joystick. This part of the power passes signals from the sensors to both of the motion controller and A/D converter. After splitting the load cell signal, the signals pass through low pass filters that filter the signal at 500 and 5000 Hz and then travel into the motion controller and A/D converter, respectively.

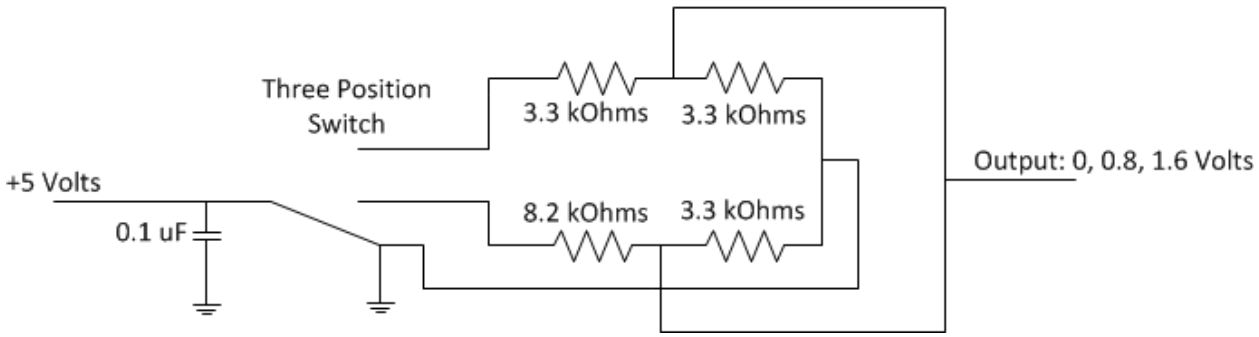

Figure A-3: Manual joystick that responds and sends voltage signals to the motion controller based off of the position of a three position switch.

Appendix B: Specifications on equipment mounts, including the probe tip, t-beam, etc.

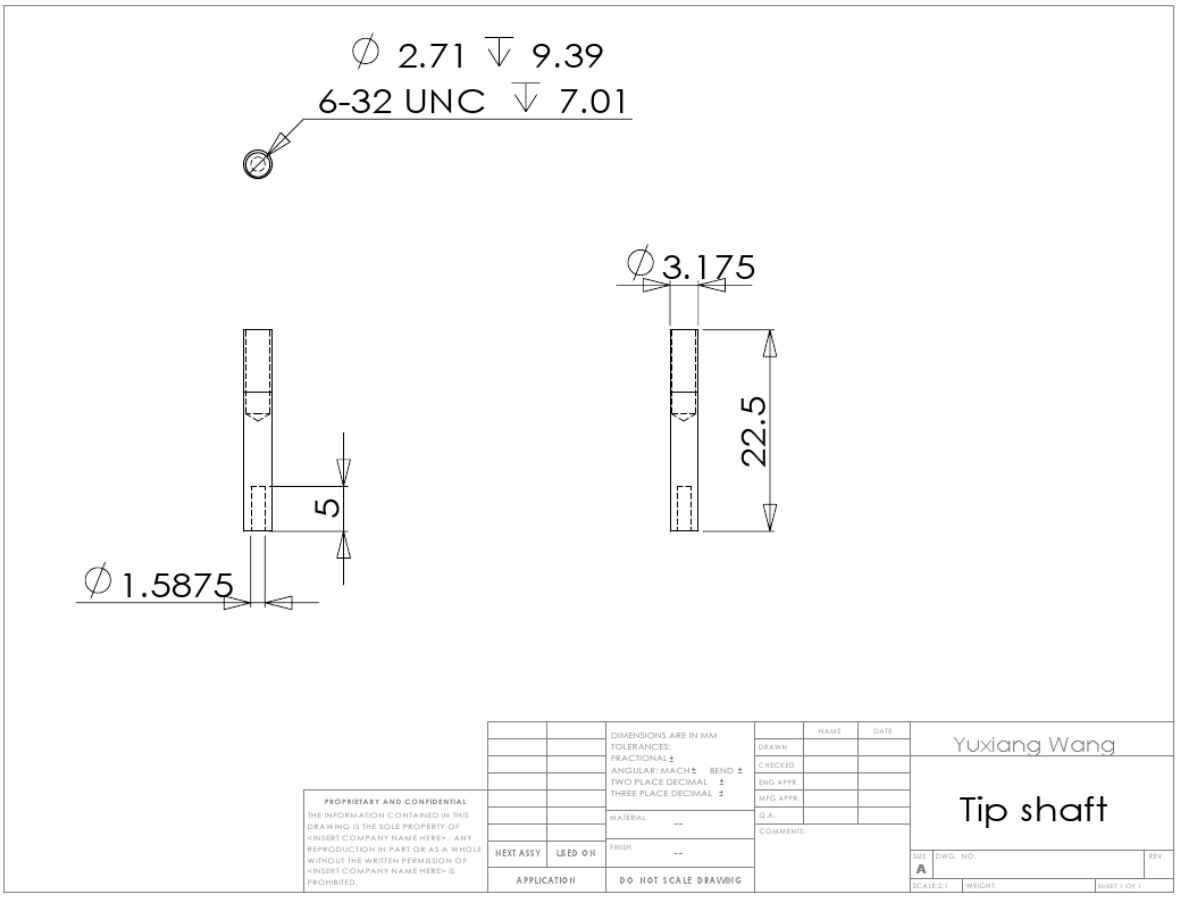

Figure B-1: Tip Shaft

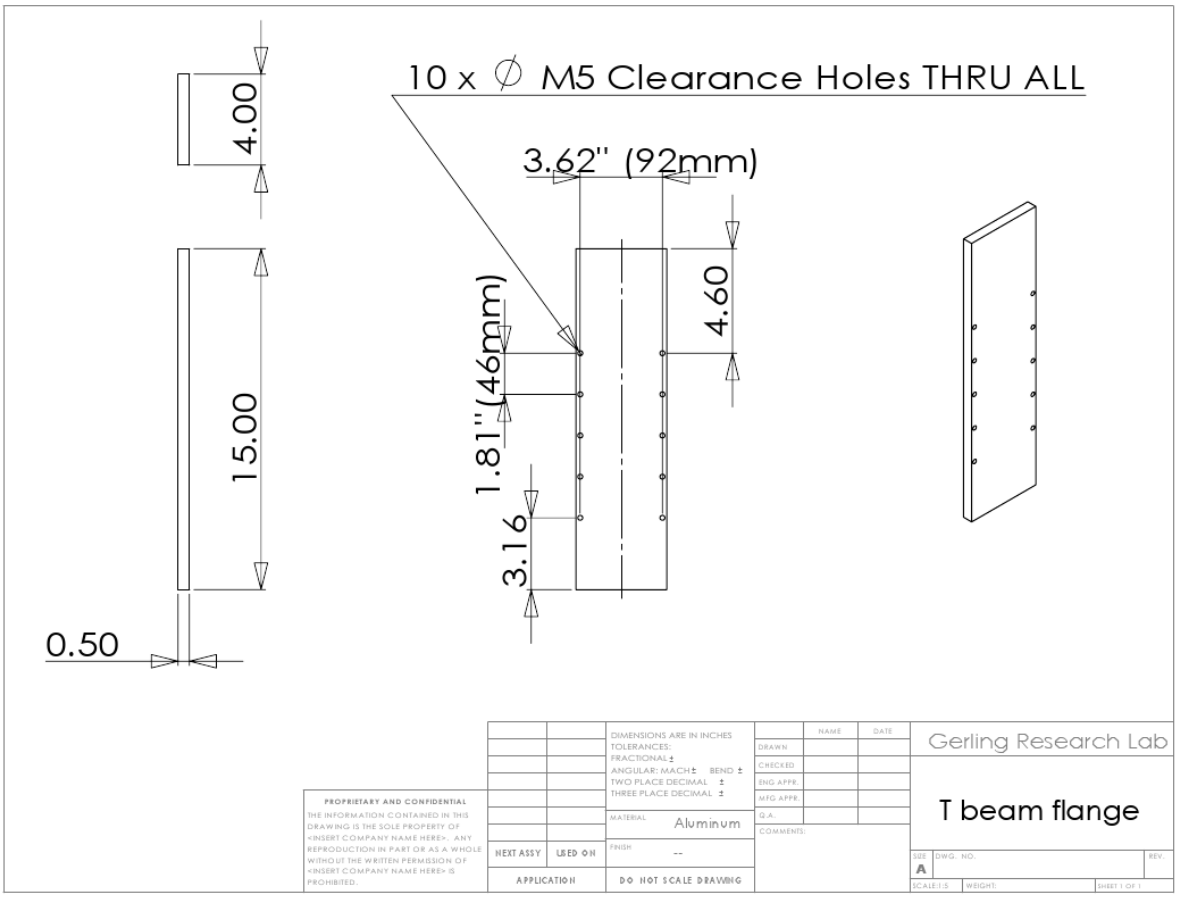

Figure B-2: T-Beam flange

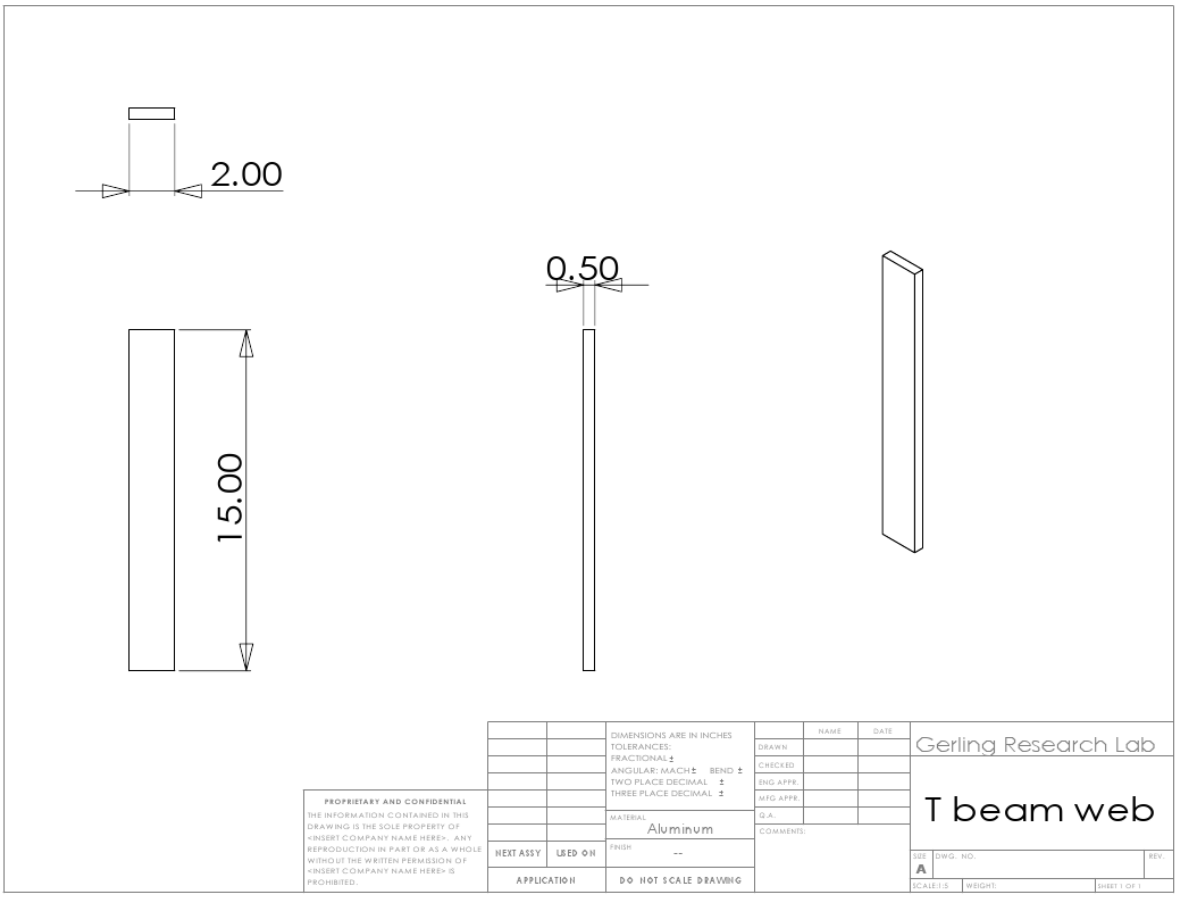

Figure B-3: T-beam web

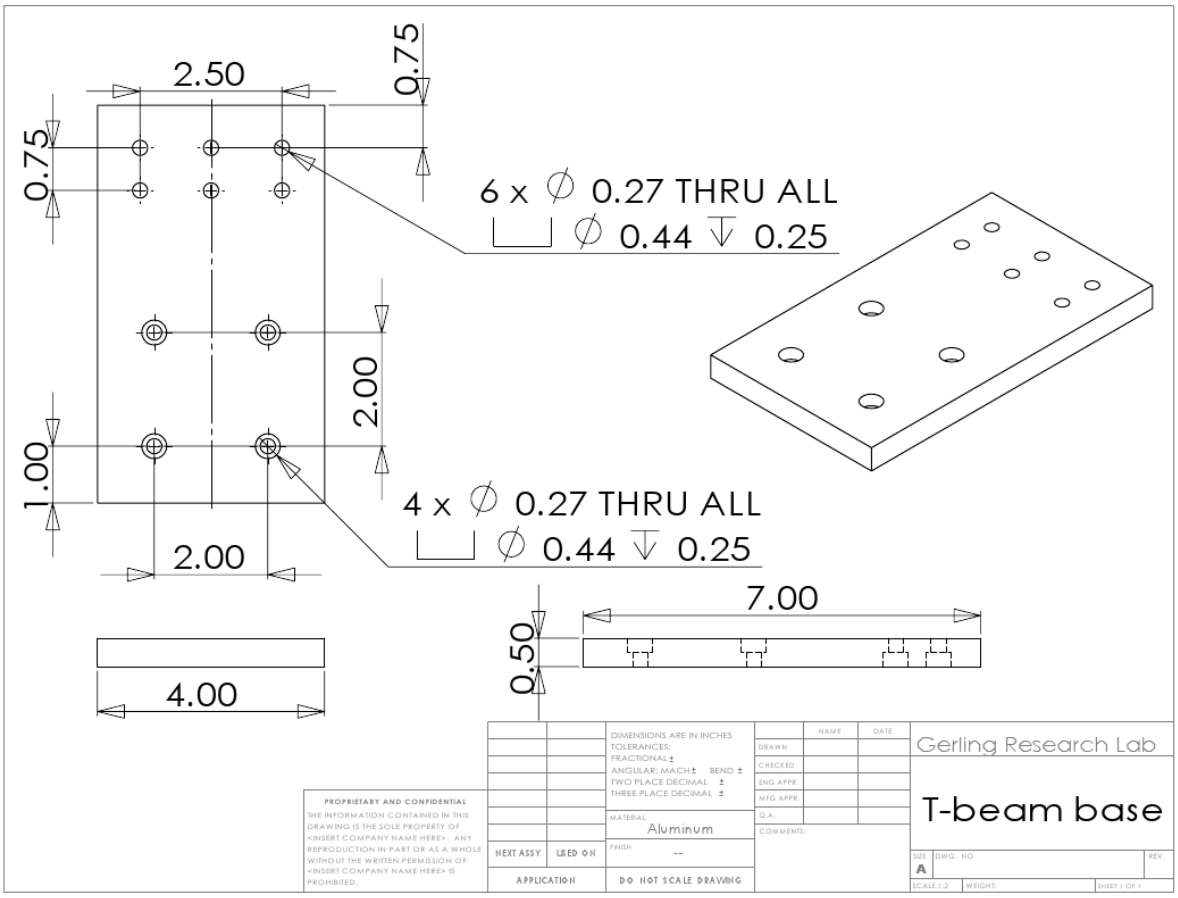

Figure B-4: T-beam base

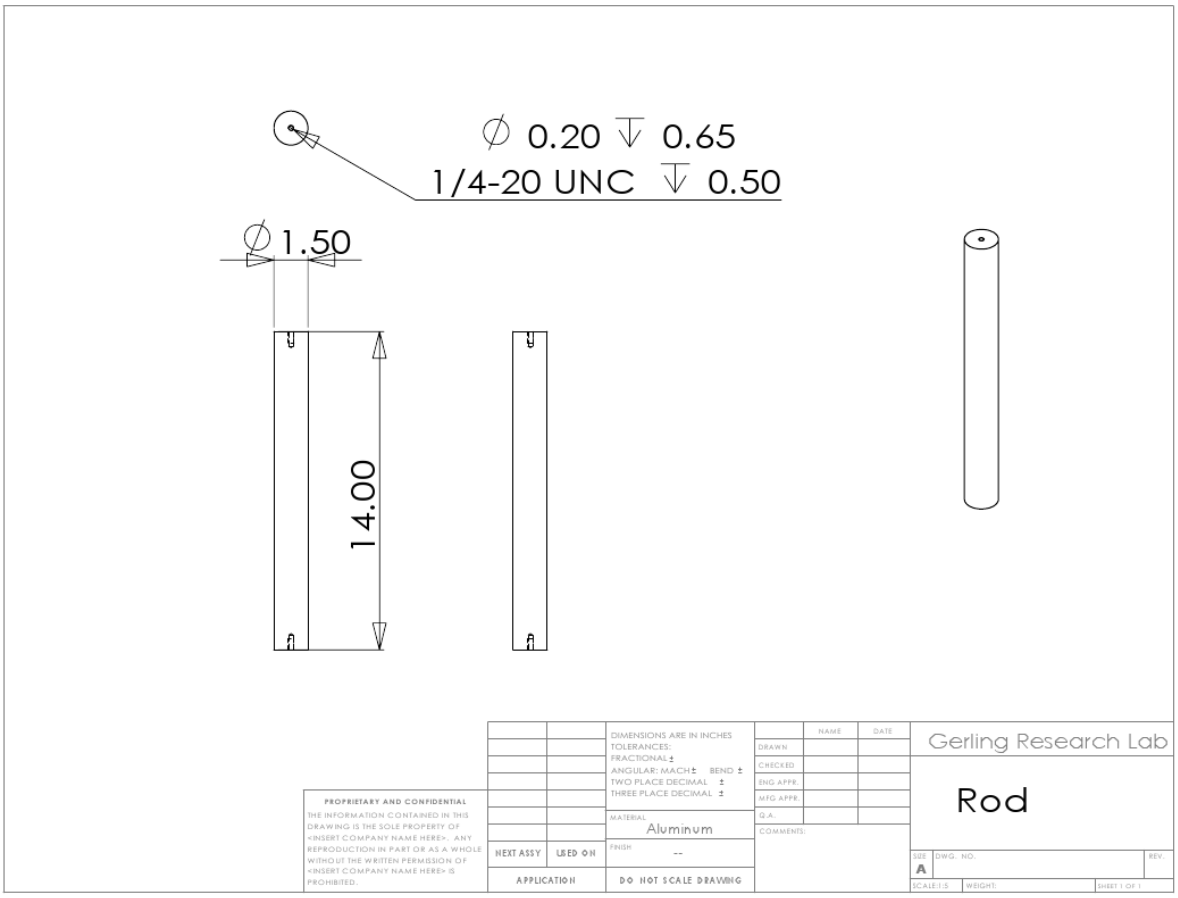

Figure B-5: Rod

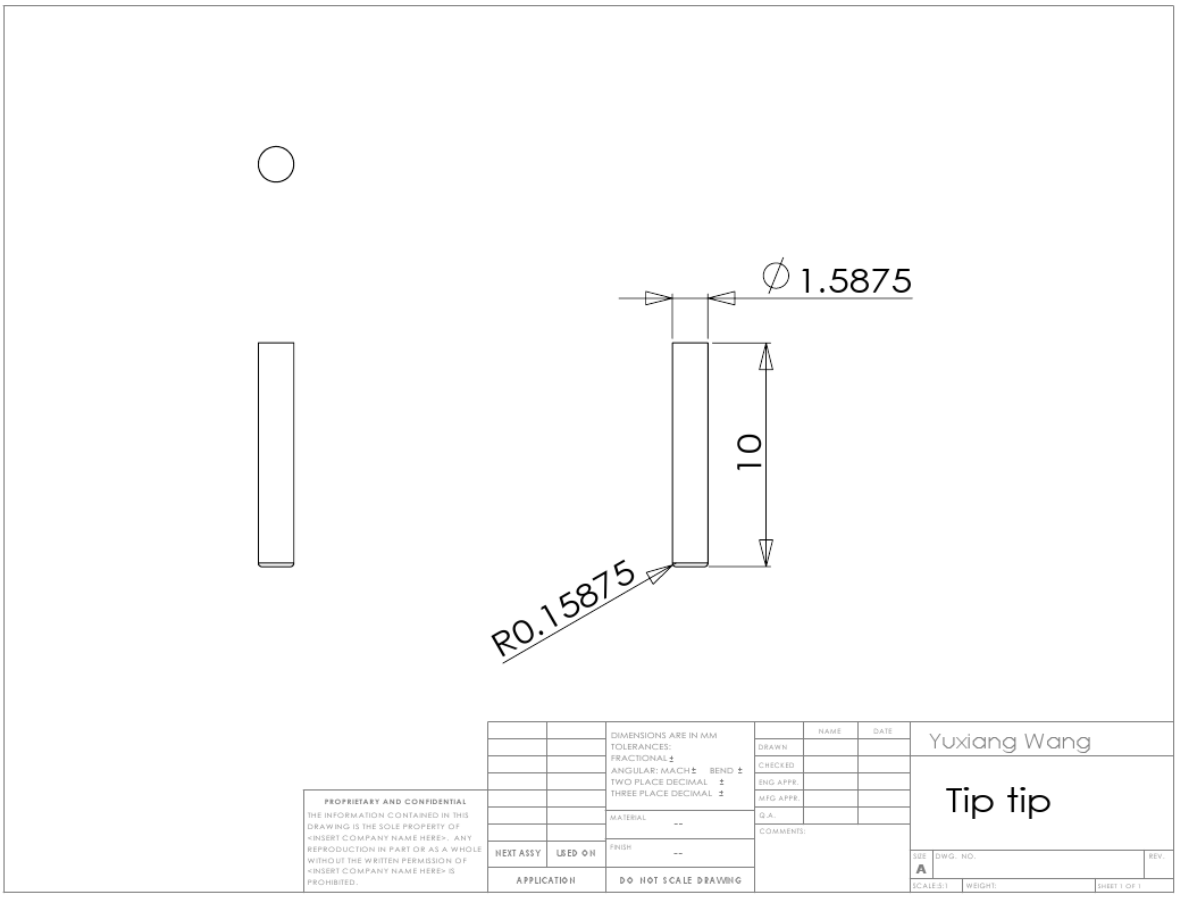

Figure B-6: Tip

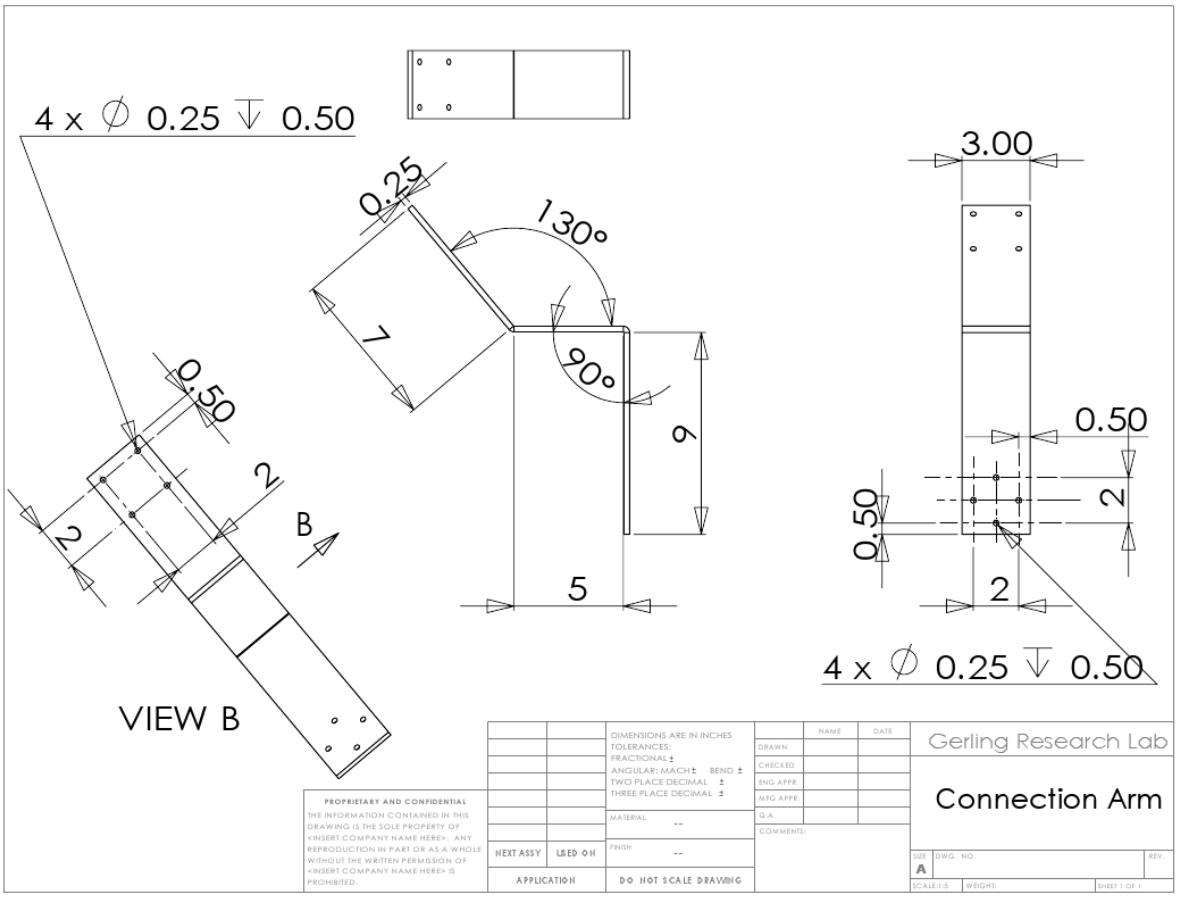

Figure B-7: Connection Arm

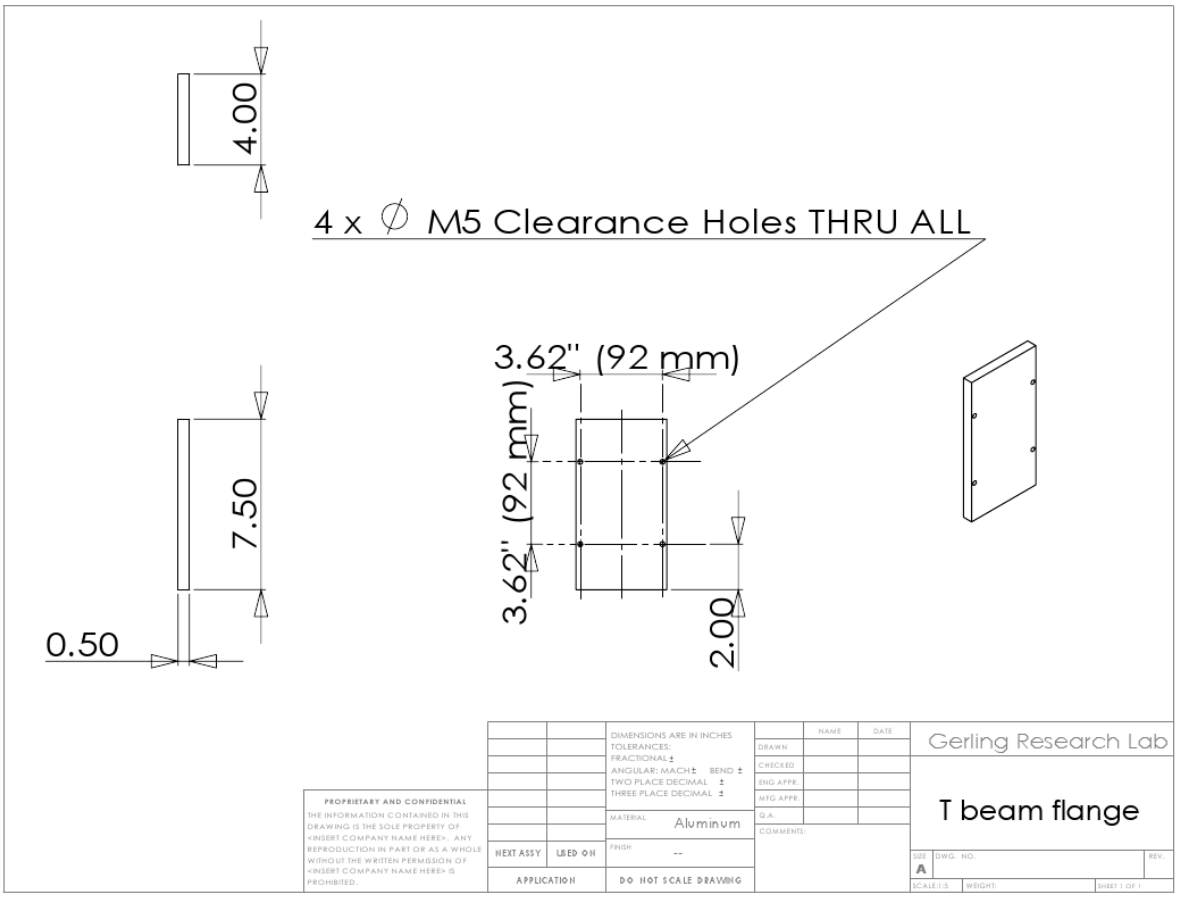

Figure B-8: T-beam flange

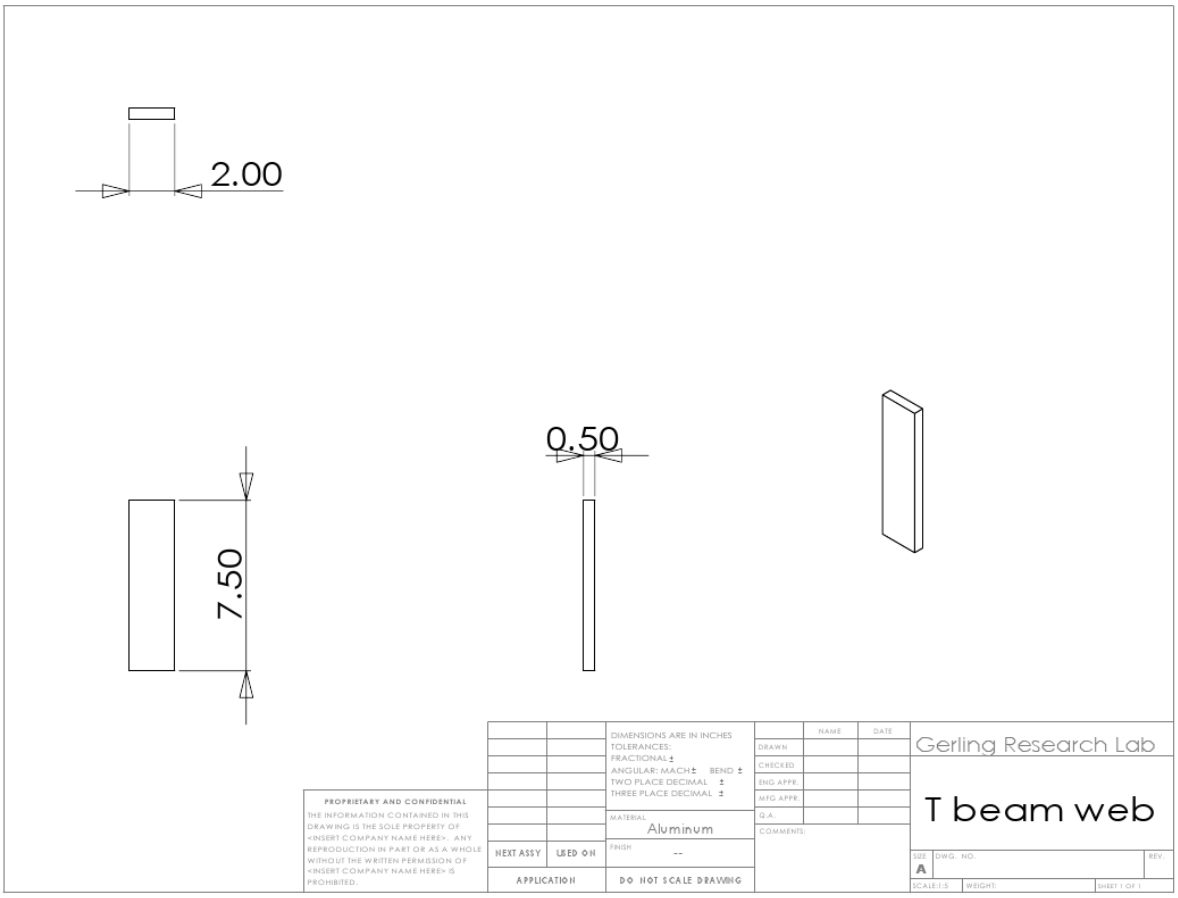

Figure B-9: T-beam web

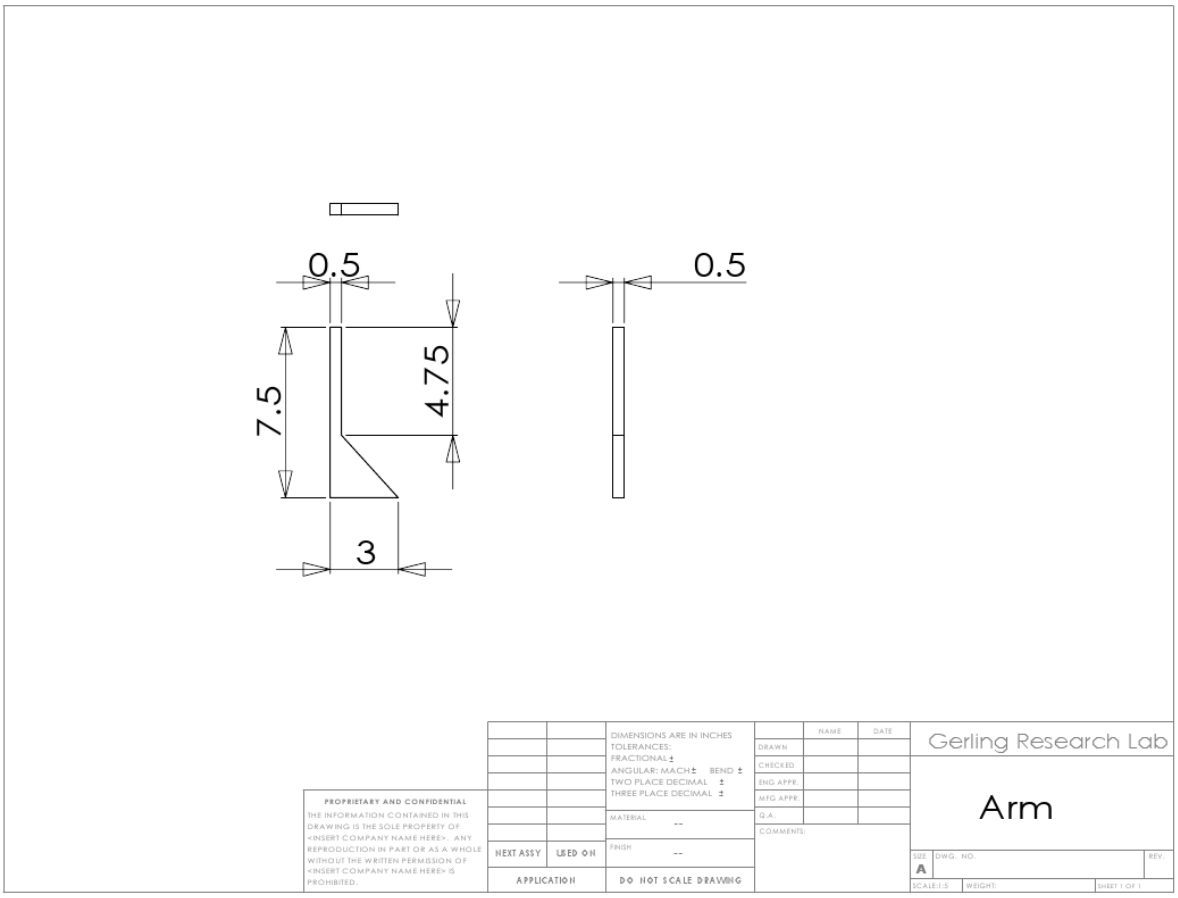

Figure B-9: Arm

Appendix C- Python code for the user interface

```
import sys
import ftplib
import os

sys.path.append('C:\\WinPython-64bit-2.7.6.4\\python-2.7.6.amd64\\site-packages\\wx-3.0-msw')

import wx
import wx.xrc
import wx.lib.agw.floatspin as FS
import numpy as np

from scipy.optimize import curve_fit
import scipy as sy
from threading import Timer
from threading import Thread
import threading
from array import *
import datetime
import time
import wx
import wx.lib.agw.floatspin as FS
from threading import *
import serial
import winsound
import math
import wx.lib.newevent
import random
import XPS_C8_drivers
import mechanical_Indenter_PID_5_PID_UVA
from wx.lib.pubsub import pub
from wx.lib.pubsub import setupkwargs
from tempfile import mkstemp
from shutil import move
from os import remove, close
# Instantiate the XPS class
myxps = XPS_C8_drivers.XPS()
myIndenter = mechanical_Indenter_PID_5_PID_UVA.mecIndent()
# Connect to the XPS
socketId = myxps.TCP_ConnectToServer('192.168.0.254', 5001, 20)

# Check connection passed
if (socketId == -1):
print 'Connection to XPS failed, check IP & Port'
sys.exit ()
global group
group = 'M'
positioner = group + '.P1'


############################################### GLOBAL UNIVERSITY VARIABLE
###############################################
#### Either 'UVA' or COL'
global univ_Var
univ_Var= 'UVA'

#########################################################################################################################
#######

global trajectRoute
trajectRoute = ''
global fileLoc
fileLoc = ''
global servoCycle
servoCycle = 0

if(univ_Var == 'UVA'):
trajectRoute = 'public/Trajectories'
```

```
fileLoc = 'C:\WinPython-64bit-2.7.6.4\python-2.7.6.amd64\ColumbiaTest'
servoCycle = 10

else:
trajectRoute = 'Public/Trajectories'
fileLoc = 'C:\Documents and Settings\Lumpkin Lab\Desktop\VirginiaForceIndent'
servoCycle = 8

global yInt
yInt = 0
global pointOfContact_1
pointOfContact_1 = 0
############################################ To Change Calibration of Load Cell change here
#####################################################################
def loadCellCalibrateForce(force):
global yInt
multiplier = 0
if(univ_Var == 'UVA'):
multiplier = 7.5079
else:
multiplier = 6.022
return multiplier*force+yInt

def loadCellCalibrateVoltage(voltage):
global yInt
multiplier = 0
if(univ_Var == 'UVA'):
multiplier = 7.5079
else:
multiplier = 6.022
#calibrationFunc.setPosition(str(multiplier)+"*x+"+str(yInt))
return (voltage+(-yInt))/multiplier


############ Global Variables ########################
global manMode
manMode = False
global dispMode
dispMode = False
global save_log_dialogFileName
save_log_dialogFileName = ''
global checkBox_ID
checkBox_ID = 3000
global checkBox_ID_f
checkBox_ID_f = 4000
global checkBox_ID_force
checkBox_ID_force = 5000
checkBox_ID_disp = 5001
global boxID_fileName
boxID_fileName = []
global forceID_fileName
forceID_fileName = []
global fineboxID_fileName
fineboxID_fileName = []
global disp_CheckBoxCounter
disp_CheckBoxCounter = []
global cb_list
cb_list = []
global cb_list_f
cb_list_f = []
global cb_list_force
global cb_list_disp
cb_list_force = []
cb_list_disp = []
global compForceCounter
compForceCounter = 0
global CompleteDisplacement
CompleteDisplacement = []
global feedForward
feedForward = []
```

```
global CompleteForce
CompleteForce=[]
global peakRange
peakRange = 1000
global cw
cw = 0
global Alone
Alone = True
global potent_Mode
global presentImages
presentImages = []
global checkBox_ID_val
checkBox_ID_val = 5000
global cb_list_Values
cb_list_Values = []
global checkBox_ID_values
checkBox_ID_values = []
global tauVal
tauVal = []
global minMaxForce
minMaxForce = 0
##########################################################################################################################
####################################
##########################################################Thread Manual
Class#############################################################################################


##########################################################################################################################
####################################
##########################################################Thread Manual
Class#############################################################################################
class ManualThread (threading.Thread):
#pause = threading.Event()
stop = False
def __init__(self):
Thread.__init__(self)
self.start()
self.stoprequest = threading.Event()
######################Set Sleep Time of Thread#####################################
def sleep(self,sec):
time.sleep(sec)

############################## Thread#################################################
def run(self):
global yInt
global univ_Var
global manual_speed
global upSpeed
global downSpeed
if(univ_Var == 'UVA'):
down_Volt = 3.5
up_Volt = 4.5
else:
up_Volt = 1.05
down_Volt = .1


###Clear force entries on other pages##############
#ForceEntries.Clear()
#ForceEntries1.Clear()
#ForceEntries2.Clear()
#for m in range(len(CompleteForce)):
# CompleteForce.pop([m])
# CompleteDisplacement.pop([m])
up_Volt = 0
down_Volt = 0
stop_now = False
countY =0
voltY = 0
```

```
startTime=datetime.datetime.now()
stopTime=startTime+datetime.timedelta(seconds=.2)
while(not(stop_now)):
timeCurrent = datetime.datetime.now()
if(timeCurrent>stopTime):
stop_now = True
voltY = voltY+ myxps.GPIOAnalogGet(socketId,["GPIO2.ADC1"])[1]
countY = countY+1

yInt = float(voltY)/countY

#######Create and add log file entry#########
#time_string = datetime.datetime.now()
#vel_string = str(speed.getfloats2var())
#addManualModeFileEntry(time_string,vel_string)
#addManualModeLogEntry(time_string,vel_string)

########################################Initiate Load sled##############################################
myIndenter.greeting(socketId,group)


####################### Retrieve speed from text box, set both sled upward movement and sled downward movement based off of this
variable.
upSpeed = -manual_speed
downSpeed = manual_speed
##########Count the amount of times the manual mode has been engaged. Initiate the status of the sled to stopped and
areStopped = True
engaged = False
###### Enable group jog mode setting within motion controller########
[errorCode,returnString]=myxps.GroupJogModeEnable(socketId,group)
if(errorCode !=0):
myIndenter.displayErrorAndClose (socketId, errorCode, 'GroupJogModeEnable')
sys.exit()

#thresholdCount = 0
#firstContact1=0
count = 0
########## While in manual mode (manual mode is set to true) ######################
while(manMode):
if(count%50 == 0):
cw_val = loadCellCalibrateVoltage(myxps.GPIOAnalogGet(socketId,["GPIO2.ADC1"])[1])*1000
wx.CallAfter(pub.sendMessage,"cw",cw_update = cw_val)
#Read Voltage from box
if(GPIOLocation.__eq__("GPIO2.ADC2")):
voltage2 = myxps.GPIOAnalogGet(socketId,[GPIOLocation])[1]

if(voltage2 > 3.5 and voltage2<4.5):
#if(voltage2 > .1 and voltage2<1.05):
areStopped = False
[errorCode, returnString] = myxps.GroupJogParametersSet(socketId, group,[upSpeed],[10])
if (errorCode != 0):
myIndenter.displayErrorAndClose (socketId, errorCode, 'GroupJogParametersSet Up')
sys.exit()

elif(voltage2<3.5):
#elif(voltage2>1.1):
areStopped = False
[errorCode, returnString] = myxps.GroupJogParametersSet(socketId, group,[downSpeed],[10])
if (errorCode != 0):
myIndenter.displayErrorAndClose (socketId, errorCode, 'GroupJogParametersSet Down')
sys.exit()
elif(voltage2>4.5):
#elif(voltage2<.1):
areStopped = True
[errorCode,returnString]=myxps.GroupMoveAbort(socketId,group)
if(errorCode !=0):
myIndenter.displayErrorAndClose (socketId, errorCode, 'GroupMoveAbort')
sys.exit()
[errorCode,returnString]=myxps.GroupJogModeEnable(socketId,group)
if(errorCode !=0):
```

```
myIndenter.displayErrorAndClose (socketId, errorCode, 'GroupJogModeEnable')
sys.exit()
count = count+1

myxps.GroupMoveAbort(socketId,group)
[errorCode, returnString]=myxps.PositionerSGammaParametersSet(socketId,positioner,10,160,.005,.05)

#myxps.GroupMoveRelative(socketId,group,[-.6])
##########################################################################################################################
#####################################
############################################################Thread Max and Min Disp
Class#############################################################################################
class DispThread (threading.Thread):
#pause = threading.Event()
stop = False
def __init__(self):
Thread.__init__(self)
self.start()
self.stoprequest = threading.Event()
######################Set Sleep Time of Thread#####################################
def sleep(self,sec):
time.sleep(sec)

############################Run Thread#########################################################
def run(self):
global setMin
global setMax
#global displayMinForce_box_2
#global displayMinDisp_box_2
#global dispMaxDisp_box_2
#global dispMaxForce_box_2
#global pointOfContact_1
########## While in manual mode (manual mode is set to true) ######################
while(dispMode):
dispVal_array = []
#Limit number of displayed values
if (not(setMin) and not(setMax) ):
currentForce = loadCellCalibrateVoltage(myxps.GPIOAnalogGet(socketId,["GPIO2.ADC1"])[1])*1000
displacement_Value = myxps.GroupPositionCurrentGet(socketId,positioner,1)[1]-pointOfContact_1

elif(setMin and not(setMax) ):
currentForce = loadCellCalibrateVoltage(myxps.GPIOAnalogGet(socketId,["GPIO2.ADC1"])[1])*1000
displacement_Value = myxps.GroupPositionCurrentGet(socketId,positioner,1)[1]-pointOfContact_1
dispVal_array.append(currentForce)
dispVal_array.append(displacement_Value)
wx.CallAfter(pub.sendMessage,"Disp_Val",disp_update = dispVal_array)
time.sleep(.3)
#wid.Refresh()


class PicturePopUp(wx.Frame):
def __init__(self):
wx.Frame.__init__(self,None,id = wx.ID_ANY,title = "Experiment Plots", pos = wx.DefaultPosition, size = (500,400),style =
wx.DEFAULT_FRAME_STYLE|wx.TAB_TRAVERSAL )
popUpSizer = wx.FlexGridSizer( 0, 1, 0, 0 )
popUpSizer.SetFlexibleDirection( wx.BOTH )
popUpSizer.SetNonFlexibleGrowMode( wx.FLEX_GROWMODE_SPECIFIED )
popUpSizer.SetMinSize( wx.Size(500,400 ) )

global plotScroll
global plotSizer
plotScroll = wx.ScrolledWindow( self, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.HSCROLL|wx.VSCROLL )
plotScroll.SetScrollRate( 5, 5 )
plotScroll.SetMinSize( wx.Size(500,400 ) )

plotSizer = wx.FlexGridSizer( 0, 1, 0, 0 )
plotSizer.SetFlexibleDirection( wx.VERTICAL )
plotSizer.SetNonFlexibleGrowMode( wx.FLEX_GROWMODE_SPECIFIED )
plotSizer.SetMinSize( wx.Size(500,400 ) )
```

```
plotScroll.SetSizer( plotSizer )
plotScroll.Layout()
plotSizer.Fit( plotScroll )
popUpSizer.Add( plotScroll, 1, wx.EXPAND|wx.ALL|wx.ALIGN_CENTER_VERTICAL|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.SetSizer( popUpSizer )
self.Layout()
self.Centre( wx.BOTH )
def __del__( self ):
pass

#########################################################################################################################################
#####################################
############################################################Thread Max and Min Disp
Class#############################################################################################
class loadCell (threading.Thread):
#pause = threading.Event()
stop = False
def __init__(self):
Thread.__init__(self)
self.start()
self.stoprequest = threading.Event()
######################Set Sleep Time of Thread#####################################
def sleep(self,sec):
time.sleep(sec)

############################Run Thread#################################################
def run(self):
#global cw
#global wid
global Alone
Alone = True
while Alone:
########## While in manual mode (manual mode is set to true) ######################
cw_val = loadCellCalibrateVoltage(myxps.GPIOAnalogGet(socketId,["GPIO2.ADC1"])[1])*1000
wx.CallAfter(pub.sendMessage,"cw",cw_update = cw_val)
time.sleep(.3)
#wid.Refresh()

class Widget(wx.Panel):
def __init__(self, parent, id):
wx.Panel.__init__(self, parent, id, size=(-1, 30), style=wx.SUNKEN_BORDER)
self.parent = parent
self.font = wx.Font(9, wx.FONTFAMILY_DEFAULT, wx.FONTSTYLE_NORMAL,
wx.FONTWEIGHT_NORMAL, False, 'Courier 10 Pitch')

self.Bind(wx.EVT_PAINT, self.OnPaint)
self.Bind(wx.EVT_SIZE, self.OnSize)


def OnPaint(self, event):
global peakRange
global cw
num = np.logspace(0,1,num = 7)
numHigh = []
numMid = []
low = 1
#high = math.log(1500,10)
#difference = (float(high)-low)/float(7)
difference = float(75)/7
#print difference
for i in range(0,8):
numHigh.append(math.log(low,10))
low = low+difference
mid = float(345)/1500*peakRange
differenceHighLow = float(peakRange-peakRange*.05-10)/4
for i in range(0,4):
numMid.append(mid)
mid = mid+differenceHighLow
ratio = float(peakRange*.05)/numHigh[len(numHigh)-1]
dc = wx.PaintDC(self)
```

```
dc.SetFont(self.font)
w, h = self.GetSize()
#print w

#step = int(round(log(w)))
ratioLow =float(w)/40
if(peakRange == 1500):
ratioMed =float(w)/3000
ratioHigh =float(w)/300

elif(peakRange ==1000):
ratioMed =float(w)/2000
ratioHigh =float(w)/200

else:
ratioMed =float(w)/1000
ratioHigh =float(w)/95



for x in range(0,len(numHigh)):
numHigh[x]=numHigh[x]*ratio+peakRange*.95
j = 0

#till = (w / 1500) * cw
#full = (w / 1500) * 1500*.9

if cw<10:
till = float(cw)*ratioLow
dc.SetPen(wx.Pen('#FFFFB8'))
dc.SetBrush(wx.Brush('#FFFFB8'))
dc.DrawRectangle(0, 0, till, 30)
elif (cw>10 and cw<(.95*peakRange)):
till = float(cw)*ratioMed+num[len(num)-1]*ratioLow
dc.SetPen(wx.Pen('#FFFFB8'))
dc.SetBrush(wx.Brush('#FFFFB8'))
dc.DrawRectangle(0, 0, till, 30)
else:
full = numMid[len(numMid)-1]*ratioMed+num[len(num)-1]*ratioLow
till = cw*ratioHigh-(numHigh[0]*ratioHigh-numMid[len(numMid)-1]*ratioMed-num[len(num)-1]*ratioLow)
dc.SetPen(wx.Pen('#FFFFB8'))
dc.SetBrush(wx.Brush('#FFFFB8'))
dc.DrawRectangle(0, 0, full, 30)
dc.SetPen(wx.Pen('#ffafaf'))
dc.SetBrush(wx.Brush('#ffafaf'))
dc.DrawRectangle(full, 0, till-full, 30)
dc.SetPen(wx.Pen('#5C5142'))
for i in range(0,len(num)):
dc.DrawLine(num[i]*ratioLow, 0, num[i]*ratioLow, 6)
width, height = dc.GetTextExtent(str('{0:.0f}'.format(num[i])))
dc.DrawText(str('{0:.0f}'.format(num[i])), num[i]*ratioLow-width/2, 6)

for i in range(0,len(numMid)):
dc.DrawLine(num[len(num)-1]*ratioLow+numMid[i]*ratioMed, 0, num[len(num)-1]*ratioLow+numMid[i]*ratioMed, 6)
if(i<3):
width, height = dc.GetTextExtent(str('{0:.0f}'.format(numMid[i])))
dc.DrawText(str('{0:.0f}'.format(numMid[i])), numMid[i]*ratioMed+num[len(num)-1]*ratioLow-width/2, 6)
#print numHigh[]*ratioHigh-numMid[len(numMid)-1]*ratioMed-num[len(num)-1]*ratioLow
#print numHigh[2]*ratioHigh-(numHigh[2]*ratioHigh-numMid[len(numMid)-1]*ratioMed-num[len(num)-1]*ratioLow)

for i in range(0,len(numHigh)):
#dc.DrawLine(num[len(num)-1]*ratioLow+numMid[len(numMid)-1]*ratioMed+numHigh[i]*ratioHigh, 0, num[len(num)-
1]*ratioLow+numMid[len(numMid)-1]*ratioMed+numHigh[i]*ratioHigh, 6)
dc.DrawLine(round(numHigh[i]*ratioHigh-(numHigh[0]*ratioHigh-numMid[len(numMid)-1]*ratioMed-num[len(num)-1]*ratioLow)), 0,
round(numHigh[i]*ratioHigh-(numHigh[0]*ratioHigh-numMid[len(numMid)-1]*ratioMed-num[len(num)-1]*ratioLow)), 6)
if(i<2 or i==4):
width, height = dc.GetTextExtent(str('{0:.0f}'.format(numHigh[i])))
dc.DrawText(str('{0:.0f}'.format(numHigh[i])), round(numHigh[i]*ratioHigh-(numHigh[0]*ratioHigh-numMid[len(numMid)-1]*ratioMed-
num[len(num)-1]*ratioLow))-width/2, 6)
```

```
def OnSize(self, event):
self.Refresh()


###########################################################################
## Class UVAColumbiaInterface
###########################################################################

class UVAColumbiaInterface ( wx.Frame ):
def __init__( self ):
global displayMinForce_box_2
global displayMinDisp_box_2
global dispMaxDisp_box_2
global dispMaxForce_box_2
global save_log_dialogFileName
global wid
global potent_Mode
wx.Frame.__init__ ( self, None, id = wx.ID_ANY, title = u"University of Virginia Force and Displacement Controlled Indenter", pos =
wx.DefaultPosition, size = wx.Size( 1100,780 ), style = wx.DEFAULT_FRAME_STYLE|wx.TAB_TRAVERSAL )
self.SetSizeHintsSz( wx.DefaultSize, wx.DefaultSize )
UISizer = wx.FlexGridSizer( 1, 2, 0, 0 )
UISizer.SetFlexibleDirection( wx.BOTH )
UISizer.SetNonFlexibleGrowMode( wx.FLEX_GROWMODE_SPECIFIED)
self.UI_Panel = wx.Panel( self, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.TAB_TRAVERSAL )
self.UI_Panel.SetBackgroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_BTNTEXT ) )
topandBottomSizer = wx.FlexGridSizer( 2, 1, 0, 0 )
topandBottomSizer.SetFlexibleDirection( wx.VERTICAL )
topandBottomSizer.SetNonFlexibleGrowMode( wx.FLEX_GROWMODE_SPECIFIED )
self.headerPanel = wx.Panel( self.UI_Panel, wx.ID_ANY, wx.DefaultPosition, wx.Size( 1100,105 ), wx.TAB_TRAVERSAL )
StepTitleSizer = wx.GridBagSizer( 0, 0 )
StepTitleSizer.SetFlexibleDirection( wx.BOTH )
StepTitleSizer.SetNonFlexibleGrowMode( wx.FLEX_GROWMODE_SPECIFIED )
# StepTitleSizer.SetMinSize( wx.Size( 1100,135 ) )
self.leftButtonPanel = wx.Panel( self.headerPanel, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.TAB_TRAVERSAL )
leftButton_sizer = wx.GridBagSizer( 0, 0 )
leftButton_sizer.SetFlexibleDirection( wx.BOTH )
leftButton_sizer.SetNonFlexibleGrowMode( wx.FLEX_GROWMODE_SPECIFIED )
self.leftButton_panel = wx.Panel( self.leftButtonPanel, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.TAB_TRAVERSAL )
leftButton_interiorSizer = wx.BoxSizer( wx.VERTICAL )
scaled_NoButton = self.scale_bitmap(wx.Bitmap( "GUI_Images/NoButton.bmp", wx.BITMAP_TYPE_ANY ),200,100)
self.NoButton_Left = wx.BitmapButton( self.leftButton_panel, wx.ID_ANY, scaled_NoButton, wx.DefaultPosition, wx.Size( -1,-1 ),
wx.BU_AUTODRAW )
leftButton_interiorSizer.Add( self.NoButton_Left, 0, wx.EXPAND, 5 )
scaled_Step1_Left = self.scale_bitmap(wx.Bitmap( "GUI_Images/Step1_Left.bmp", wx.BITMAP_TYPE_ANY ),200,100)
self.Step1_Left = wx.BitmapButton( self.leftButton_panel, 1, scaled_Step1_Left, wx.DefaultPosition, wx.DefaultSize, wx.BU_AUTODRAW )
self.Step1_Left.Enable( False )
self.Step1_Left.Hide()
self.Bind(wx.EVT_BUTTON,self.changePagesLeft,id=1)

leftButton_interiorSizer.Add( self.Step1_Left, 0, 0, 5 )
scaled_Step2_Left = self.scale_bitmap(wx.Bitmap( "GUI_Images/Step2_Left.bmp", wx.BITMAP_TYPE_ANY ),200,100)

self.Step2_Left = wx.BitmapButton( self.leftButton_panel, 2,scaled_Step2_Left, wx.DefaultPosition, wx.DefaultSize, wx.BU_AUTODRAW )
self.Step2_Left.Enable( False )
self.Step2_Left.Hide()
self.Bind(wx.EVT_BUTTON,self.changePagesLeft,id=2)

leftButton_interiorSizer.Add( self.Step2_Left, 0, 0, 5 )
scaled_Step3_Left = self.scale_bitmap(wx.Bitmap( "GUI_Images/Step3_Left.bmp", wx.BITMAP_TYPE_ANY ),200,100)

self.Step3_Left = wx.BitmapButton( self.leftButton_panel, 3,scaled_Step3_Left, wx.DefaultPosition, wx.DefaultSize, wx.BU_AUTODRAW )
self.Step3_Left.Enable( False )
self.Step3_Left.Hide()
self.Bind(wx.EVT_BUTTON,self.changePagesLeft,id=3)
leftButton_interiorSizer.Add( self.Step3_Left, 0, 0, 5 )
scaled_Step4_Left = self.scale_bitmap(wx.Bitmap( "GUI_Images/Step4_Left.bmp", wx.BITMAP_TYPE_ANY ),200,100)

self.Step4_Left = wx.BitmapButton( self.leftButton_panel, 22, scaled_Step4_Left, wx.DefaultPosition, wx.DefaultSize, wx.BU_AUTODRAW )
self.Step4_Left.Enable( False )
```

```
self.Step4_Left.Hide()
self.Bind(wx.EVT_BUTTON,self.changePagesLeft,id=22)

leftButton_interiorSizer.Add( self.Step4_Left, 0, 0, 5 )

scaled_Step5_Left = self.scale_bitmap(wx.Bitmap( "GUI_Images/Step5_Left.bmp", wx.BITMAP_TYPE_ANY ),200,100)

self.Step5_Left = wx.BitmapButton( self.leftButton_panel, 4, scaled_Step5_Left, wx.DefaultPosition, wx.DefaultSize, wx.BU_AUTODRAW )
self.Step5_Left.Enable( False )
self.Step5_Left.Hide()
self.Bind(wx.EVT_BUTTON,self.changePagesLeft,id=4)

leftButton_interiorSizer.Add( self.Step5_Left, 0, 0, 5 )
self.leftButton_panel.SetSizer( leftButton_interiorSizer )
self.leftButton_panel.Layout()
leftButton_interiorSizer.Fit( self.leftButton_panel )
leftButton_sizer.Add( self.leftButton_panel, wx.GBPosition( 0, 0 ), wx.GBSpan( 1, 1 ), wx.EXPAND, 5 )
self.leftButtonPanel.SetSizer( leftButton_sizer )
self.leftButtonPanel.Layout()
leftButton_sizer.Fit( self.leftButtonPanel )
StepTitleSizer.Add( self.leftButtonPanel, wx.GBPosition( 0, 0 ), wx.GBSpan( 1, 1 ), wx.EXPAND, 5 )
self.step_panel = wx.Panel( self.headerPanel, wx.ID_ANY, wx.DefaultPosition, wx.Size( 650,100 ), wx.TAB_TRAVERSAL )
step_sizer = wx.BoxSizer( wx.VERTICAL )
self.upperStep_text = wx.StaticText( self.step_panel, wx.ID_ANY, "Step 1: Approach Surface and Find Point of Contact", wx.DefaultPosition,
wx.DefaultSize, wx.ALIGN_CENTRE )
self.upperStep_text.Wrap( -1 )
self.upperStep_text.SetFont( wx.Font( 26, 74, 90, 92, False, "Calibri" ) )
self.upperStep_text.SetForegroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_INFOBK ) )
step_sizer.Add( self.upperStep_text, 1, wx.ALIGN_CENTER_HORIZONTAL|wx.RIGHT|wx.LEFT, 5 )
self.step_panel.SetSizer( step_sizer )
self.step_panel.Layout()
StepTitleSizer.Add( self.step_panel, wx.GBPosition( 0, 1 ),wx.GBSpan(1,3), wx.EXPAND|wx.ALL, 5 )
self.rightButtonPanel = wx.Panel( self.headerPanel, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.TAB_TRAVERSAL )
rightButtonSizer = wx.GridBagSizer( 0, 0 )
rightButtonSizer.SetFlexibleDirection( wx.BOTH )
rightButtonSizer.SetNonFlexibleGrowMode( wx.FLEX_GROWMODE_SPECIFIED )
self.rightButtonInteriorPanel = wx.Panel( self.rightButtonPanel, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.TAB_TRAVERSAL )
RightButtonInteriorSizer = wx.BoxSizer( wx.HORIZONTAL )
scaled_Step2_Right = self.scale_bitmap(wx.Bitmap( "GUI_Images/Step2_Right.bmp", wx.BITMAP_TYPE_ANY ),200,100)
self.Step2_Right = wx.BitmapButton( self.rightButtonInteriorPanel, 5, scaled_Step2_Right, wx.DefaultPosition, wx.DefaultSize,
wx.BU_AUTODRAW )
RightButtonInteriorSizer.Add( self.Step2_Right, 0, 0, 5 )
self.Bind(wx.EVT_BUTTON,self.changePagesRight,id=5)

scaled_Step3_Right = self.scale_bitmap(wx.Bitmap( "GUI_Images/Step3_Right.bmp", wx.BITMAP_TYPE_ANY ),200,100)
self.Step3_Right = wx.BitmapButton( self.rightButtonInteriorPanel, 6,scaled_Step3_Right, wx.DefaultPosition, wx.DefaultSize,
wx.BU_AUTODRAW )
self.Step3_Right.Enable( False )
self.Step3_Right.Hide()
self.Bind(wx.EVT_BUTTON,self.changePagesRight,id=6)

RightButtonInteriorSizer.Add( self.Step3_Right, 0, 0, 5 )
scaled_Step4_Right = self.scale_bitmap(wx.Bitmap( "GUI_Images/Step4_Right.bmp", wx.BITMAP_TYPE_ANY ),200,100)
self.Step4_Right = wx.BitmapButton( self.rightButtonInteriorPanel, 7, scaled_Step4_Right, wx.DefaultPosition, wx.DefaultSize,
wx.BU_AUTODRAW )
self.Step4_Right.Enable( False )
self.Step4_Right.Hide()
self.Bind(wx.EVT_BUTTON,self.changePagesRight,id=7)

RightButtonInteriorSizer.Add( self.Step4_Right, 0, 0, 5 )
scaled_NoButton_Right = self.scale_bitmap(wx.Bitmap( "GUI_Images/NoButton.bmp", wx.BITMAP_TYPE_ANY ),200,100)

self.NoButton = wx.BitmapButton( self.rightButtonInteriorPanel, 8,scaled_NoButton_Right, wx.DefaultPosition, wx.DefaultSize,
wx.BU_AUTODRAW )
self.NoButton.Enable( False )
self.NoButton.Hide()
self.Bind(wx.EVT_BUTTON,self.changePagesRight,id=8)

RightButtonInteriorSizer.Add( self.NoButton, 0, 0, 5 )
scaled_Step6_Right = self.scale_bitmap(wx.Bitmap( "GUI_Images/Step6_Right.bmp", wx.BITMAP_TYPE_ANY ),200,100)
```

```
self.Step6_Right = wx.BitmapButton( self.rightButtonInteriorPanel, 9, scaled_Step6_Right, wx.DefaultPosition, wx.DefaultSize,
wx.BU_AUTODRAW )
self.Step6_Right.Enable( False )
self.Step6_Right.Hide()
self.Bind(wx.EVT_BUTTON,self.changePagesRight,id=9)

RightButtonInteriorSizer.Add( self.Step6_Right, 0, 0, 5 )
self.rightButtonInteriorPanel.SetSizer( RightButtonInteriorSizer )
self.rightButtonInteriorPanel.Layout()
RightButtonInteriorSizer.Fit( self.rightButtonInteriorPanel )
rightButtonSizer.Add( self.rightButtonInteriorPanel, wx.GBPosition( 0, 0 ), wx.GBSpan( 1, 1 ), wx.EXPAND, 5 )
self.rightButtonPanel.SetSizer( rightButtonSizer )
self.rightButtonPanel.Layout()
rightButtonSizer.Fit( self.rightButtonPanel )
StepTitleSizer.Add( self.rightButtonPanel, wx.GBPosition( 0, 4 ), wx.GBSpan( 1, 1 ), 0, 5 )
self.headerPanel.SetSizer( StepTitleSizer )
self.headerPanel.Layout()
topandBottomSizer.Add( self.headerPanel, 1, wx.EXPAND |wx.ALL, 5 )
self.decisionAndLogPanel = wx.Panel( self.UI_Panel, wx.ID_ANY, wx.DefaultPosition, wx.Size( 1100,630 ), wx.TAB_TRAVERSAL )
decision_LogSizer = wx.FlexGridSizer( 1, 2, 0, 0 )
decision_LogSizer.SetFlexibleDirection( wx.HORIZONTAL )
decision_LogSizer.SetNonFlexibleGrowMode( wx.FLEX_GROWMODE_SPECIFIED )
self.decisionPanel = wx.Panel( self.decisionAndLogPanel, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.TAB_TRAVERSAL )
self.decisionPanel.SetBackgroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_3DLIGHT ) )
self.decisionPanel.SetMinSize( wx.Size( 660,-1 ) )
self.decisionPanel.SetMaxSize( wx.Size( 660,-1 ) )
decisionSizer = wx.BoxSizer( wx.VERTICAL )
decisionSizer.SetMinSize( wx.Size( 660,-1 ) )
self.step1_panel = wx.Panel( self.decisionPanel, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.TAB_TRAVERSAL )
step1_sizer = wx.FlexGridSizer( 3, 1, 0, 0 )
step1_sizer.SetFlexibleDirection( wx.VERTICAL )
step1_sizer.SetNonFlexibleGrowMode( wx.FLEX_GROWMODE_SPECIFIED )
step1_sizer.SetMinSize( wx.Size( 660,600 ) )
self.forceRange_text_1 = wx.Panel( self.step1_panel, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.SIMPLE_BORDER )
forceRange_sizer_1 = wx.FlexGridSizer( 3, 1, 0, 0 )
forceRange_sizer_1.SetFlexibleDirection( wx.BOTH )
forceRange_sizer_1.SetNonFlexibleGrowMode( wx.FLEX_GROWMODE_SPECIFIED )
forceRangeTextSizer_1 = wx.FlexGridSizer( 0, 3, 0, 0 )
forceRangeTextSizer_1.SetFlexibleDirection( wx.BOTH )
forceRangeTextSizer_1.SetNonFlexibleGrowMode( wx.FLEX_GROWMODE_SPECIFIED )
forceRangeTextSizer_1.SetMinSize( wx.Size( 660,-1 ) )
self.approachSurface_text_1 = wx.StaticText( self.forceRange_text_1, wx.ID_ANY, u"Approach surface:", wx.DefaultPosition, wx.Size( -1,30 ),
0 )
self.approachSurface_text_1.Wrap( -1 )
self.approachSurface_text_1.SetFont( wx.Font( 14, 74, 90, 92, False, "Calibri" ) )
forceRangeTextSizer_1.Add( self.approachSurface_text_1, 0, wx.LEFT|wx.RIGHT, 5 )
self.engageJoystick_text_1 = wx.StaticText( self.forceRange_text_1, wx.ID_ANY, u"Engage joystick", wx.DefaultPosition, wx.Size( 475,30 ), 0
)
self.engageJoystick_text_1.Wrap( -1 )
self.engageJoystick_text_1.SetFont( wx.Font( 14, 74, 90, 90, False, "Calibri" ) )
forceRangeTextSizer_1.Add( self.engageJoystick_text_1, 0, wx.LEFT|wx.RIGHT, 5 )
forceRange_sizer_1.Add( forceRangeTextSizer_1, 1, wx.EXPAND, 5 )
velocity_sizer_1 = wx.GridBagSizer( 0, 0 )
velocity_sizer_1.SetFlexibleDirection( wx.VERTICAL )
velocity_sizer_1.SetNonFlexibleGrowMode( wx.FLEX_GROWMODE_SPECIFIED )
velocity_sizer_1.SetMinSize( wx.Size( 300,-1 ) )
self.min_text_app_1 = wx.StaticText( self.forceRange_text_1, wx.ID_ANY, u"Min", wx.DefaultPosition, wx.DefaultSize, 0 )
self.min_text_app_1.Wrap( -1 )
self.min_text_app_1.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.min_text_app_1.SetForegroundColour((96,96,96))
velocity_sizer_1.Add( self.min_text_app_1, wx.GBPosition( 0, 0 ), wx.GBSpan( 1, 1 ),
wx.LEFT|wx.RIGHT|wx.ALIGN_CENTER_VERTICAL|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.vel_text_app_1 = wx.StaticText( self.forceRange_text_1, wx.ID_ANY, u"Velocity (mm/s)", wx.DefaultPosition, wx.DefaultSize,
wx.ALIGN_CENTRE )
self.vel_text_app_1.Wrap( -1 )
self.vel_text_app_1.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.vel_text_app_1.SetMinSize( wx.Size( 150,-1 ) )
velocity_sizer_1.Add( self.vel_text_app_1, wx.GBPosition( 0, 1 ), wx.GBSpan( 1, 1 ),
wx.RIGHT|wx.LEFT|wx.ALIGN_BOTTOM|wx.ALIGN_CENTER_HORIZONTAL, 5 )
```

```
self.max_text_app_1 = wx.StaticText( self.forceRange_text_1, wx.ID_ANY, u"Max", wx.DefaultPosition, wx.DefaultSize,
wx.ALIGN_CENTRE )
self.max_text_app_1.Wrap( -1 )
self.max_text_app_1.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.max_text_app_1.SetForegroundColour((96,96,96) )
velocity_sizer_1.Add( self.max_text_app_1, wx.GBPosition( 0, 2 ), wx.GBSpan( 1, 1 ),
wx.LEFT|wx.RIGHT|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.minVal_text_app_1 = wx.StaticText( self.forceRange_text_1, wx.ID_ANY, u"0.1", wx.DefaultPosition, wx.Size( -1,-1 ), 0 )
self.minVal_text_app_1.Wrap( -1 )
self.minVal_text_app_1.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.minVal_text_app_1.SetForegroundColour( (96,96,96) )
velocity_sizer_1.Add( self.minVal_text_app_1, wx.GBPosition( 1, 0 ), wx.GBSpan( 1, 1 ),
wx.LEFT|wx.RIGHT|wx.ALIGN_CENTER_HORIZONTAL, 5 )

speeds = ['.05','.1','.2','.3','.4','.5','.6','1.2','2.4']
#self.vel_appoach_1=wx.ComboBox(self.forceRange_text_1,value = '.3', choices = speeds, style = wx.CB_READONLY,wx.DefaultPosition)
self.vel_appoach_1=wx.ComboBox(self.forceRange_text_1,value = '2.4',choices = speeds, style = wx.CB_READONLY)#, wx.DefaultPosition)
self.vel_appoach_1.SetFont( wx.Font( 14, 74, 90, 90, False, "Calibri" ) )
self.vel_appoach_1.SetForegroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_BTNTEXT ) )
self.vel_appoach_1.SetBackgroundColour( wx.Colour( 185, 185, 185 ) )
self.vel_appoach_1.SetMinSize( wx.Size( 100,-1 ) )
self.vel_appoach_1.Bind(wx.EVT_TEXT,self.UpdateSpeed)

velocity_sizer_1.Add( self.vel_appoach_1, wx.GBPosition( 1, 1 ), wx.GBSpan( 1, 1 ),
wx.ALIGN_CENTER_HORIZONTAL|wx.RIGHT|wx.LEFT, 5 )
self.maxVal_text_app_1 = wx.StaticText( self.forceRange_text_1, wx.ID_ANY, u"2", wx.DefaultPosition, wx.DefaultSize, 0 )
self.maxVal_text_app_1.Wrap( -1 )
self.maxVal_text_app_1.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.maxVal_text_app_1.SetForegroundColour( (96,96,96) )
velocity_sizer_1.Add( self.maxVal_text_app_1, wx.GBPosition( 1, 2 ), wx.GBSpan( 1, 1 ),
wx.ALL|wx.ALIGN_CENTER_VERTICAL|wx.ALIGN_CENTER_HORIZONTAL, 5 )
forceRange_sizer_1.Add( velocity_sizer_1, 1, wx.ALIGN_CENTER_HORIZONTAL, 5 )
button_approach_sizer = wx.GridSizer( 0, 2, 0, 0 )
self.Engage_Button = wx.Button( self.forceRange_text_1, 10, u"Engage", wx.DefaultPosition, wx.Size( 175,60 ), 0 )
self.Engage_Button.SetFont( wx.Font( 18, 74, 90, 92, False, "Calibri" ) )
self.Bind(wx.EVT_BUTTON,self.Engage,id=10)

button_approach_sizer.Add( self.Engage_Button, 0, wx.ALL|wx.ALIGN_RIGHT, 5 )
self.Disengage_Button = wx.Button( self.forceRange_text_1, 11, u"Disengage", wx.DefaultPosition, wx.Size( 175,60 ), 0 )
self.Disengage_Button.SetFont( wx.Font( 18, 74, 90, 92, False, "Calibri" ) )
self.Disengage_Button.SetForegroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_BTNHIGHLIGHT ) )
self.Disengage_Button.SetBackgroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_3DDKSHADOW ) )
self.Bind(wx.EVT_BUTTON,self.Disengage,id=11)

button_approach_sizer.Add( self.Disengage_Button, 0, wx.TOP|wx.BOTTOM|wx.LEFT, 5 )
forceRange_sizer_1.Add( button_approach_sizer, 0, wx.EXPAND|wx.ALIGN_CENTER_HORIZONTAL|wx.ALIGN_CENTER_VERTICAL, 5
)
self.forceRange_text_1.SetSizer( forceRange_sizer_1 )
self.forceRange_text_1.Layout()
forceRange_sizer_1.Fit( self.forceRange_text_1 )
step1_sizer.Add( self.forceRange_text_1, 1, wx.EXPAND, 5 )
self.findPOC_panel_1 = wx.Panel( self.step1_panel, wx.ID_ANY, wx.DefaultPosition, wx.Size( 660,375 ), wx.SIMPLE_BORDER )
findPOC_sizer_1 = wx.FlexGridSizer( 3, 1, 0, 0 )
findPOC_sizer_1.SetFlexibleDirection( wx.BOTH )
findPOC_sizer_1.SetNonFlexibleGrowMode( wx.FLEX_GROWMODE_SPECIFIED )
textPOC_sizer_1 = wx.FlexGridSizer( 0, 2, 0, 0 )
textPOC_sizer_1.SetFlexibleDirection( wx.HORIZONTAL )
textPOC_sizer_1.SetNonFlexibleGrowMode( wx.FLEX_GROWMODE_SPECIFIED )
textPOC_sizer_1.SetMinSize( wx.Size( 660,-1 ) )
self.findPOC_text_1 = wx.StaticText( self.findPOC_panel_1, wx.ID_ANY, u"Find first point of contact:", wx.DefaultPosition, wx.DefaultSize, 0
)
self.findPOC_text_1.Wrap( -1 )
self.findPOC_text_1.SetFont( wx.Font( 14, 74, 90, 92, False, "Calibri" ) )
textPOC_sizer_1.Add( self.findPOC_text_1, 0, wx.ALL, 5 )
self.indentParm_text_1 = wx.StaticText( self.findPOC_panel_1, wx.ID_ANY, u"Indentation parameters", wx.DefaultPosition, wx.DefaultSize, 0
)
self.indentParm_text_1.Wrap( -1 )
self.indentParm_text_1.SetFont( wx.Font( 14, 74, 90, 90, False, "Calibri" ) )
self.indentParm_text_1.SetMinSize( wx.Size( 450,-1 ) )
textPOC_sizer_1.Add( self.indentParm_text_1, 0, wx.TOP|wx.RIGHT|wx.LEFT, 5 )
```

```
findPOC_sizer_1.Add( textPOC_sizer_1, 1, wx.EXPAND, 5 )
POC_sizer_21 = wx.GridBagSizer( 0, 0 )
POC_sizer_21.SetFlexibleDirection( wx.BOTH )
POC_sizer_21.SetNonFlexibleGrowMode( wx.FLEX_GROWMODE_SPECIFIED )
self.max_text_21 = wx.StaticText( self.findPOC_panel_1, wx.ID_ANY, u"Max", wx.DefaultPosition, wx.DefaultSize, 0 )
self.max_text_21.Wrap( -1 )
self.max_text_21.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.max_text_21.SetForegroundColour( (96,96,96))
POC_sizer_21.Add( self.max_text_21, wx.GBPosition( 0, 2 ), wx.GBSpan( 1, 1 ),
wx.ALIGN_RIGHT|wx.ALIGN_CENTER_VERTICAL|wx.BOTTOM|wx.RIGHT|wx.LEFT, 5 )
self.displacement_text_21 = wx.StaticText( self.findPOC_panel_1, wx.ID_ANY, u"displ. (mm)", wx.DefaultPosition, wx.DefaultSize,
wx.ALIGN_LEFT )
self.displacement_text_21.Wrap( -1 )
self.displacement_text_21.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
POC_sizer_21.Add( self.displacement_text_21, wx.GBPosition( 0, 1 ), wx.GBSpan( 1, 1 ),
wx.LEFT|wx.RIGHT|wx.TOP|wx.ALIGN_BOTTOM|wx.ALIGN_LEFT, 5 )
self.min_text_21 = wx.StaticText( self.findPOC_panel_1, wx.ID_ANY, u"Min", wx.DefaultPosition, wx.DefaultSize, wx.ALIGN_CENTRE )
self.min_text_21.Wrap( -1 )
self.min_text_21.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.min_text_21.SetForegroundColour( (96,96,96))
POC_sizer_21.Add( self.min_text_21, wx.GBPosition( 0, 0 ), wx.GBSpan( 1, 1 ),
wx.ALIGN_LEFT|wx.ALIGN_CENTER_VERTICAL|wx.BOTTOM|wx.RIGHT, 5 )
self.minDisp_text_21 = wx.StaticText( self.findPOC_panel_1, wx.ID_ANY, u"0.01", wx.DefaultPosition, wx.Size( -1,-1 ), wx.ALIGN_LEFT )
self.minDisp_text_21.Wrap( -1 )
self.minDisp_text_21.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.minDisp_text_21.SetForegroundColour((96,96,96) )
self.minDisp_text_21.SetMinSize( wx.Size( 100,-1 ) )

POC_sizer_21.Add( self.minDisp_text_21, wx.GBPosition( 1, 0 ), wx.GBSpan( 1, 1 ),
wx.ALIGN_LEFT|wx.TOP|wx.BOTTOM|wx.ALIGN_CENTER_VERTICAL, 5 )
self.disp_spin_21 = FS.FloatSpin( self.findPOC_panel_1, -1, value = .6, min_val=.01, max_val=1,increment = .01,agwStyle=FS.FS_RIGHT )
self.disp_spin_21.SetFormat("%f")
self.disp_spin_21.SetDigits(2)
self.disp_spin_21.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.disp_spin_21.SetForegroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_BTNTEXT ) )
self.disp_spin_21.SetBackgroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_BTNSHADOW ) )
POC_sizer_21.Add( self.disp_spin_21, wx.GBPosition( 1, 1 ), wx.GBSpan( 1, 1 ),
wx.ALIGN_CENTER_VERTICAL|wx.ALIGN_CENTER_HORIZONTAL|wx.RIGHT|wx.LEFT|wx.BOTTOM, 5 )
self.minVel_text_21 = wx.StaticText( self.findPOC_panel_1, wx.ID_ANY, u"1.0", wx.DefaultPosition, wx.Size( -1,-1 ), wx.ALIGN_CENTRE )
self.minVel_text_21.Wrap( -1 )
self.minVel_text_21.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.minVel_text_21.SetForegroundColour((96,96,96) )
POC_sizer_21.Add( self.minVel_text_21, wx.GBPosition( 3, 0 ), wx.GBSpan( 1, 1 ),
wx.TOP|wx.BOTTOM|wx.ALIGN_CENTER_VERTICAL|wx.ALIGN_LEFT, 5 )
self.max_dispText_21 = wx.StaticText( self.findPOC_panel_1, wx.ID_ANY, u"5", wx.DefaultPosition,wx.DefaultSize,wx.ALIGN_RIGHT )
self.max_dispText_21.Wrap( -1 )
self.max_dispText_21.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.max_dispText_21.SetForegroundColour((96,96,96) )
self.max_dispText_21.SetMinSize( wx.Size( 100,-1 ) )


POC_sizer_21.Add( self.max_dispText_21, wx.GBPosition( 1, 2 ), wx.GBSpan( 1, 1 ),wx.LEFT|wx.RIGHT|wx.ALIGN_RIGHT, 5 )
self.vel_text_21 = wx.StaticText( self.findPOC_panel_1, wx.ID_ANY, u"velocity (mm/s)", wx.DefaultPosition, wx.DefaultSize,
wx.ALIGN_LEFT )
self.vel_text_21.Wrap( -1 )
self.vel_text_21.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
POC_sizer_21.Add( self.vel_text_21, wx.GBPosition( 2, 1 ), wx.GBSpan( 1, 1 ),
wx.ALIGN_BOTTOM|wx.LEFT|wx.RIGHT|wx.ALIGN_LEFT|wx.TOP, 5 )
self.vel_spin_21 = FS.FloatSpin( self.findPOC_panel_1, -1,value = 1, min_val=1, max_val=2,increment = .2, agwStyle=FS.FS_RIGHT)
self.vel_spin_21.SetFormat("%f")
self.vel_spin_21.SetDigits(2)
self.vel_spin_21.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.vel_spin_21.SetForegroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_BTNTEXT ) )
self.vel_spin_21.SetBackgroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_BTNSHADOW ) )
POC_sizer_21.Add( self.vel_spin_21, wx.GBPosition( 3, 1 ), wx.GBSpan( 1, 1 ),
wx.ALIGN_CENTER_HORIZONTAL|wx.RIGHT|wx.LEFT|wx.BOTTOM, 5 )
self.maxVel_text_21 = wx.StaticText( self.findPOC_panel_1, wx.ID_ANY, u"40", wx.DefaultPosition, wx.DefaultSize, 0 )
self.maxVel_text_21.Wrap( -1 )
self.maxVel_text_21.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.maxVel_text_21.SetForegroundColour((96,96,96) )
```

```
POC_sizer_21.Add( self.maxVel_text_21, wx.GBPosition( 3, 2 ), wx.GBSpan( 1, 1 ),
wx.LEFT|wx.RIGHT|wx.ALIGN_RIGHT|wx.ALIGN_CENTER_VERTICAL, 5 )
self.acc_text_21 = wx.StaticText( self.findPOC_panel_1, wx.ID_ANY, u"accel. (mm/s/s)", wx.DefaultPosition, wx.DefaultSize,
wx.ALIGN_LEFT )
self.acc_text_21.Wrap( -1 )
self.acc_text_21.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
POC_sizer_21.Add( self.acc_text_21, wx.GBPosition( 4, 1 ), wx.GBSpan( 1, 1 ),wx.RIGHT|wx.LEFT|wx.ALIGN_RIGHT|wx.TOP, 5 )
self.min_Acc_text_21 = wx.StaticText( self.findPOC_panel_1, wx.ID_ANY, u"1.0", wx.DefaultPosition, wx.DefaultSize, 0 )
self.min_Acc_text_21.Wrap( -1 )
self.min_Acc_text_21.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.min_Acc_text_21.SetForegroundColour( (96,96,96))
POC_sizer_21.Add( self.min_Acc_text_21, wx.GBPosition( 5, 0 ), wx.GBSpan( 1, 1 ), wx.ALIGN_CENTER_VERTICAL|wx.ALIGN_LEFT, 5
)
self.acc_spin_21 =FS.FloatSpin( self.findPOC_panel_1, -1, value = 40, min_val=1, max_val=160.0,increment = .01,agwStyle=FS.FS_RIGHT )
self.acc_spin_21.SetFormat("%f")
self.acc_spin_21.SetDigits(2)
self.acc_spin_21.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.acc_spin_21.SetForegroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_BTNTEXT ) )
self.acc_spin_21.SetBackgroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_BTNSHADOW ) )
POC_sizer_21.Add( self.acc_spin_21, wx.GBPosition( 5, 1 ), wx.GBSpan( 1, 1 ),
wx.RIGHT|wx.LEFT|wx.ALIGN_CENTER_HORIZONTAL|wx.BOTTOM, 5 )
self.max_Acc_text_21 = wx.StaticText( self.findPOC_panel_1, wx.ID_ANY, u"160", wx.DefaultPosition, wx.DefaultSize, 0 )
self.max_Acc_text_21.Wrap( -1 )
self.max_Acc_text_21.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.max_Acc_text_21.SetForegroundColour( (96,96,96))
POC_sizer_21.Add( self.max_Acc_text_21, wx.GBPosition( 5, 2 ), wx.GBSpan( 1, 1 ),
wx.LEFT|wx.RIGHT|wx.ALIGN_CENTER_VERTICAL|wx.ALIGN_RIGHT, 5 )
self.hold_text_21 = wx.StaticText( self.findPOC_panel_1, wx.ID_ANY, u"hold (s) ", wx.DefaultPosition, wx.DefaultSize, wx.ALIGN_LEFT )
self.hold_text_21.Wrap( -1 )
self.hold_text_21.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
POC_sizer_21.Add( self.hold_text_21, wx.GBPosition( 6, 1 ), wx.GBSpan( 1, 1 ),
wx.RIGHT|wx.LEFT|wx.TOP|wx.ALIGN_CENTER_VERTICAL|wx.ALIGN_LEFT, 5 )
self.min_holdText_21 = wx.StaticText( self.findPOC_panel_1, wx.ID_ANY, u"0.1", wx.DefaultPosition, wx.DefaultSize, wx.ALIGN_CENTRE
)
self.min_holdText_21.Wrap( -1 )
self.min_holdText_21.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.min_holdText_21.SetForegroundColour( (96,96,96))
POC_sizer_21.Add( self.min_holdText_21, wx.GBPosition( 7, 0 ), wx.GBSpan( 1, 1 ),wx.ALIGN_LEFT|wx.ALIGN_CENTER_VERTICAL, 5
)
self.hold_spin_21 = FS.FloatSpin( self.findPOC_panel_1,-1,min_val=.1, max_val=600.0,increment = .1, agwStyle=FS.FS_RIGHT )
self.hold_spin_21.SetFormat("%f")
self.hold_spin_21.SetDigits(2)
self.hold_spin_21.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.hold_spin_21.SetForegroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_BTNTEXT ) )
self.hold_spin_21.SetBackgroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_BTNSHADOW ) )
POC_sizer_21.Add( self.hold_spin_21, wx.GBPosition( 7, 1 ), wx.GBSpan( 1, 1 ),
wx.ALIGN_CENTER_HORIZONTAL|wx.RIGHT|wx.LEFT|wx.BOTTOM, 5 )
self.maxHold_text_21 = wx.StaticText( self.findPOC_panel_1, wx.ID_ANY, u"600", wx.DefaultPosition, wx.DefaultSize, 0 )
self.maxHold_text_21.Wrap( -1 )
self.maxHold_text_21.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.maxHold_text_21.SetForegroundColour((96,96,96) )
POC_sizer_21.Add( self.maxHold_text_21, wx.GBPosition( 7, 2 ), wx.GBSpan( 1, 1 ),
wx.LEFT|wx.RIGHT|wx.ALIGN_RIGHT|wx.ALIGN_CENTER_VERTICAL, 5 )
self.radioButton_panel_21 = wx.Panel( self.findPOC_panel_1, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.TAB_TRAVERSAL )
self.radioButton_panel_21.SetBackgroundColour(wx.SystemSettings.GetColour(wx.SYS_COLOUR_HIGHLIGHTTEXT ))

radioButton_sizer_21 = wx.GridSizer( 0, 3, 0, 0 )
self.cycle_21 = wx.RadioButton( self.radioButton_panel_21, wx.ID_ANY, u"Cycle", wx.DefaultPosition, wx.DefaultSize, 0 )
self.cycle_21.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.cycle_21.SetValue(True)

radioButton_sizer_21.Add( self.cycle_21, 0, wx.ALL|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.retract_21 = wx.RadioButton( self.radioButton_panel_21, wx.ID_ANY, u"Retract", wx.DefaultPosition, wx.DefaultSize, 0 )
self.retract_21.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
radioButton_sizer_21.Add( self.retract_21, 0, wx.ALL|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.extend_21 = wx.RadioButton( self.radioButton_panel_21, wx.ID_ANY, u"Extend", wx.DefaultPosition, wx.DefaultSize, 0 )
self.extend_21.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
radioButton_sizer_21.Add( self.extend_21, 0, wx.ALL|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.radioButton_panel_21.SetSizer( radioButton_sizer_21 )
self.radioButton_panel_21.Layout()
```

```
radioButton_sizer_21.Fit( self.radioButton_panel_21 )
POC_sizer_21.Add( self.radioButton_panel_21, wx.GBPosition( 8, 0 ), wx.GBSpan( 1, 3 ), wx.EXPAND |wx.ALL, 5 )
findPOC_sizer_1.Add( POC_sizer_21, 0, wx.ALIGN_CENTER_VERTICAL|wx.ALIGN_CENTER_HORIZONTAL, 5 )
button_poc_sizer = wx.BoxSizer( wx.VERTICAL )

self.indent_poc_1 = wx.Button( self.findPOC_panel_1, 12, u"Indent", wx.DefaultPosition, wx.Size( 175,60 ), 0 )
self.indent_poc_1.SetFont( wx.Font( 18, 74, 90, 92, False, "Calibri" ) )
self.Bind(wx.EVT_BUTTON,self.displacementIndent,id=12)
self.indent_poc_1.Disable()
self.indent_poc_1.SetMinSize( wx.Size( 175,60 ) )

button_poc_sizer.Add( self.indent_poc_1, 0, wx.ALIGN_CENTER_HORIZONTAL|wx.TOP|wx.BOTTOM, 5 )
findPOC_sizer_1.Add( button_poc_sizer, 1, wx.EXPAND, 5 )
self.findPOC_panel_1.SetSizer( findPOC_sizer_1 )
self.findPOC_panel_1.Layout()
step1_sizer.Add( self.findPOC_panel_1, 1, wx.EXPAND, 5 )
self.setPOC_panel_1 = wx.Panel( self.step1_panel, wx.ID_ANY, wx.DefaultPosition, wx.Size( 660,140 ), wx.SIMPLE_BORDER )
setPOC_sizer_1 = wx.BoxSizer( wx.VERTICAL )
setPOC_sizer_1.SetMinSize( wx.Size( -1,250 ) )
setPOC_text_sizer_1 = wx.FlexGridSizer( 3, 1, 0, 0 )
setPOC_text_sizer_1.SetFlexibleDirection( wx.BOTH )
setPOC_text_sizer_1.SetNonFlexibleGrowMode( wx.FLEX_GROWMODE_SPECIFIED )
self.setPOC_text_1 = wx.StaticText( self.setPOC_panel_1, wx.ID_ANY, u"Set first point of contact", wx.DefaultPosition, wx.DefaultSize,
wx.ALIGN_LEFT )
self.setPOC_text_1.Wrap( -1 )
self.setPOC_text_1.SetFont( wx.Font( 14, 74, 90, 92, False, "Calibri" ) )
self.setPOC_text_1.SetMinSize( wx.Size( 200,-1 ) )
setPOC_text_sizer_1.Add( self.setPOC_text_1, 1, wx.TOP|wx.BOTTOM|wx.LEFT, 5 )
setPOC_sizer_1.Add( setPOC_text_sizer_1, 1, 0, 5 )
setPOC_button_sizer_1 = wx.FlexGridSizer( 3, 0, 0, 0 )
setPOC_button_sizer_1.SetFlexibleDirection( wx.BOTH )
setPOC_button_sizer_1.SetNonFlexibleGrowMode( wx.FLEX_GROWMODE_SPECIFIED )

self.setPOC_button_1 = wx.Button( self.setPOC_panel_1, 13, u"Set", wx.DefaultPosition, wx.Size( 175,60 ), 0 )
self.setPOC_button_1.SetFont( wx.Font( 18, 74, 90, 92, False, "Calibri" ) )
self.Bind(wx.EVT_BUTTON,self.setPointOfContact1,id=13)
self.setPOC_button_1.Disable()
setPOC_button_sizer_1.Add( self.setPOC_button_1, 0, wx.ALIGN_CENTER_HORIZONTAL|wx.TOP|wx.BOTTOM, 5 )
setPOC_sizer_1.Add( setPOC_button_sizer_1, 0, wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.setPOC_panel_1.SetSizer( setPOC_sizer_1 )
self.setPOC_panel_1.Layout()
step1_sizer.Add( self.setPOC_panel_1, 1, wx.EXPAND, 5 )
self.step1_panel.SetSizer( step1_sizer )
self.step1_panel.Layout()
step1_sizer.Fit( self.step1_panel )
decisionSizer.Add( self.step1_panel, 1, wx.EXPAND, 5 )
self.step2_panel = wx.Panel( self.decisionPanel, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.TAB_TRAVERSAL )
self.step2_panel.Enable( False )
self.step2_panel.Hide()
step2_sizer = wx.FlexGridSizer( 3, 1, 0, 0 )
step2_sizer.SetFlexibleDirection( wx.VERTICAL )
step2_sizer.SetNonFlexibleGrowMode( wx.FLEX_GROWMODE_SPECIFIED )
step2_sizer.SetMinSize( wx.Size( 660,600 ) )
self.findPOC_panel_2 = wx.Panel( self.step2_panel, wx.ID_ANY, wx.DefaultPosition, wx.Size( 660,400 ), wx.SIMPLE_BORDER )
findPOC_sizer_2 = wx.FlexGridSizer( 3, 1, 0, 0 )
findPOC_sizer_2.SetFlexibleDirection( wx.BOTH )
findPOC_sizer_2.SetNonFlexibleGrowMode( wx.FLEX_GROWMODE_SPECIFIED )
panel_header_sizer_2 = wx.FlexGridSizer( 0, 2, 0, 0 )
panel_header_sizer_2.SetFlexibleDirection( wx.HORIZONTAL )
panel_header_sizer_2.SetNonFlexibleGrowMode( wx.FLEX_GROWMODE_SPECIFIED )
panel_header_sizer_2.SetMinSize( wx.Size( 660,-1 ) )
self.findPOC_text2 = wx.StaticText( self.findPOC_panel_2, wx.ID_ANY, u"Indentation parameters", wx.DefaultPosition, wx.DefaultSize, 0 )
self.findPOC_text2.Wrap( -1 )
self.findPOC_text2.SetFont( wx.Font( 14, 74, 90, 92, False, "Calibri" ) )
panel_header_sizer_2.Add( self.findPOC_text2, 0, wx.ALL, 5 )
findPOC_sizer_2.Add( panel_header_sizer_2, 1, wx.EXPAND, 5 )
POC_sizer_2 = wx.GridBagSizer( 0, 0 )
POC_sizer_2.SetFlexibleDirection( wx.BOTH )
POC_sizer_2.SetNonFlexibleGrowMode( wx.FLEX_GROWMODE_SPECIFIED )
self.max_text_2 = wx.StaticText( self.findPOC_panel_2, wx.ID_ANY, u"Max", wx.DefaultPosition, wx.DefaultSize, 0 )
```

```
self.max_text_2.Wrap( -1 )
self.max_text_2.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.max_text_2.SetForegroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_SCROLLBAR ) )
POC_sizer_2.Add( self.max_text_2, wx.GBPosition( 0, 2 ), wx.GBSpan( 1, 1 ),
wx.ALIGN_RIGHT|wx.ALIGN_CENTER_VERTICAL|wx.RIGHT|wx.TOP|wx.BOTTOM, 5 )
self.displacement_text_2 = wx.StaticText( self.findPOC_panel_2, wx.ID_ANY, u"displ. (mm)", wx.DefaultPosition, wx.DefaultSize,
wx.ALIGN_LEFT )
self.displacement_text_2.Wrap( -1 )
self.displacement_text_2.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )

POC_sizer_2.Add( self.displacement_text_2, wx.GBPosition( 0, 1 ), wx.GBSpan( 1, 1 ), wx.LEFT|wx.RIGHT|wx.ALIGN_BOTTOM, 5 )
self.min_text_2 = wx.StaticText( self.findPOC_panel_2, wx.ID_ANY, u"Min", wx.DefaultPosition, wx.DefaultSize, wx.ALIGN_LEFT )
self.min_text_2.Wrap( -1 )
self.min_text_2.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.min_text_2.SetForegroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_SCROLLBAR ) )
POC_sizer_2.Add( self.min_text_2, wx.GBPosition( 0, 0 ), wx.GBSpan( 1, 1 ),
wx.RIGHT|wx.TOP|wx.BOTTOM|wx.ALIGN_LEFT|wx.ALIGN_CENTER_VERTICAL, 5 )
self.minDisp_text_2 = wx.StaticText( self.findPOC_panel_2, wx.ID_ANY, u"0.01", wx.DefaultPosition, wx.Size( -1,-1 ), wx.ALIGN_LEFT )
self.minDisp_text_2.Wrap( -1 )
self.minDisp_text_2.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.minDisp_text_2.SetForegroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_SCROLLBAR ) )
self.minDisp_text_2.SetMinSize( wx.Size( 100,-1 ) )

POC_sizer_2.Add( self.minDisp_text_2, wx.GBPosition( 1, 0 ), wx.GBSpan( 1, 1 ),
wx.ALIGN_LEFT|wx.TOP|wx.BOTTOM|wx.ALIGN_CENTER_VERTICAL, 5 )
self.disp_spin_2 = FS.FloatSpin( self.findPOC_panel_2, -1,value = .6, min_val=.01, max_val=5,increment = .1, agwStyle=FS.FS_RIGHT )
self.disp_spin_2.SetFormat("%f")
self.disp_spin_2.SetDigits(2)
self.disp_spin_2.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.disp_spin_2.SetForegroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_BTNTEXT ) )
self.disp_spin_2.SetBackgroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_BTNSHADOW ) )
POC_sizer_2.Add( self.disp_spin_2, wx.GBPosition( 1, 1 ), wx.GBSpan( 1, 1 ),
wx.BOTTOM|wx.RIGHT|wx.LEFT|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.minVel_text_2 = wx.StaticText( self.findPOC_panel_2, wx.ID_ANY, u"1.0", wx.DefaultPosition, wx.Size( -1,-1 ), wx.ALIGN_LEFT )
self.minVel_text_2.Wrap( -1 )
self.minVel_text_2.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.minVel_text_2.SetForegroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_SCROLLBAR ) )
POC_sizer_2.Add( self.minVel_text_2, wx.GBPosition( 3, 0 ), wx.GBSpan( 1, 1 ),
wx.ALIGN_LEFT|wx.TOP|wx.BOTTOM|wx.RIGHT|wx.ALIGN_CENTER_VERTICAL, 5 )
self.max_dispText_2 = wx.StaticText( self.findPOC_panel_2, wx.ID_ANY, u"5", wx.DefaultPosition, wx.DefaultSize, wx.ALIGN_RIGHT )
self.max_dispText_2.Wrap( -1 )
self.max_dispText_2.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.max_dispText_2.SetForegroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_SCROLLBAR ) )
self.max_dispText_2.SetMinSize( wx.Size( 100,-1 ) )

POC_sizer_2.Add( self.max_dispText_2, wx.GBPosition( 1, 2 ), wx.GBSpan( 1, 1 ),
wx.ALL|wx.ALIGN_CENTER_HORIZONTAL|wx.ALIGN_CENTER_VERTICAL, 5 )
self.vel_text_2 = wx.StaticText( self.findPOC_panel_2, wx.ID_ANY, u"velocity (mm/s)", wx.DefaultPosition, wx.DefaultSize,
wx.ALIGN_LEFT )
self.vel_text_2.Wrap( -1 )
self.vel_text_2.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
POC_sizer_2.Add( self.vel_text_2, wx.GBPosition( 2, 1 ), wx.GBSpan( 1, 1 ), wx.ALIGN_BOTTOM|wx.LEFT|wx.RIGHT|wx.TOP, 5 )
self.vel_spin_2 = FS.FloatSpin( self.findPOC_panel_2, -1,value = 1, min_val=1, max_val=40.0,increment = .01, agwStyle=FS.FS_RIGHT )
self.vel_spin_2.SetFormat("%f")
self.vel_spin_2.SetDigits(2)
self.vel_spin_2.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.vel_spin_2.SetForegroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_BTNTEXT ) )
self.vel_spin_2.SetBackgroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_BTNSHADOW ) )
POC_sizer_2.Add( self.vel_spin_2, wx.GBPosition( 3, 1 ), wx.GBSpan( 1, 1 ),
wx.BOTTOM|wx.RIGHT|wx.LEFT|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.maxVel_text_2 = wx.StaticText( self.findPOC_panel_2, wx.ID_ANY, u"40", wx.DefaultPosition, wx.DefaultSize, 0 )
self.maxVel_text_2.Wrap( -1 )
self.maxVel_text_2.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.maxVel_text_2.SetForegroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_SCROLLBAR ) )
POC_sizer_2.Add( self.maxVel_text_2, wx.GBPosition( 3, 2 ), wx.GBSpan( 1, 1 ),
wx.ALL|wx.ALIGN_RIGHT|wx.ALIGN_CENTER_VERTICAL, 5 )
self.acc_text_2 = wx.StaticText( self.findPOC_panel_2, wx.ID_ANY, u"accel. (mm/s/s)", wx.DefaultPosition, wx.DefaultSize,
wx.ALIGN_CENTRE )
self.acc_text_2.Wrap( -1 )
self.acc_text_2.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
```

```
POC_sizer_2.Add( self.acc_text_2, wx.GBPosition( 4, 1 ), wx.GBSpan( 1, 1 ), wx.TOP|wx.RIGHT|wx.LEFT, 5 )
self.min_Acc_text_2 = wx.StaticText( self.findPOC_panel_2, wx.ID_ANY, u"1.0", wx.DefaultPosition, wx.DefaultSize, 0 )
self.min_Acc_text_2.Wrap( -1 )
self.min_Acc_text_2.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.min_Acc_text_2.SetForegroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_SCROLLBAR ) )
POC_sizer_2.Add( self.min_Acc_text_2, wx.GBPosition( 5, 0 ), wx.GBSpan( 1, 1 ),
wx.ALIGN_LEFT|wx.TOP|wx.BOTTOM|wx.RIGHT|wx.ALIGN_CENTER_VERTICAL, 5 )
self.acc_spin_2 = FS.FloatSpin( self.findPOC_panel_2, -1,value = 40, min_val=.1, max_val=40.0,increment = .5, agwStyle=FS.FS_RIGHT )
self.acc_spin_2.SetFormat("%f")
self.acc_spin_2.SetDigits(2)
self.acc_spin_2.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.acc_spin_2.SetForegroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_BTNTEXT ) )
self.acc_spin_2.SetBackgroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_BTNSHADOW ) )
POC_sizer_2.Add( self.acc_spin_2, wx.GBPosition( 5, 1 ), wx.GBSpan( 1, 1 ),
wx.BOTTOM|wx.RIGHT|wx.LEFT|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.max_Acc_text_2 = wx.StaticText( self.findPOC_panel_2, wx.ID_ANY, u"160", wx.DefaultPosition, wx.DefaultSize, 0 )
self.max_Acc_text_2.Wrap( -1 )
self.max_Acc_text_2.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.max_Acc_text_2.SetForegroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_SCROLLBAR ) )
POC_sizer_2.Add( self.max_Acc_text_2, wx.GBPosition( 5, 2 ), wx.GBSpan( 1, 1 ),
wx.ALL|wx.ALIGN_RIGHT|wx.ALIGN_CENTER_VERTICAL, 5 )
self.hold_text_2 = wx.StaticText( self.findPOC_panel_2, wx.ID_ANY, u"hold (s) ", wx.DefaultPosition, wx.DefaultSize, wx.ALIGN_CENTRE )
self.hold_text_2.Wrap( -1 )
self.hold_text_2.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
POC_sizer_2.Add( self.hold_text_2, wx.GBPosition( 6, 1 ), wx.GBSpan( 1, 1 ),
wx.TOP|wx.RIGHT|wx.LEFT|wx.ALIGN_CENTER_VERTICAL|wx.ALIGN_LEFT, 5 )
self.min_holdText_2 = wx.StaticText( self.findPOC_panel_2, wx.ID_ANY, u"0.8", wx.DefaultPosition, wx.DefaultSize, wx.ALIGN_CENTRE )
self.min_holdText_2.Wrap( -1 )
self.min_holdText_2.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.min_holdText_2.SetForegroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_SCROLLBAR ) )
POC_sizer_2.Add( self.min_holdText_2, wx.GBPosition( 7, 0 ), wx.GBSpan( 1, 1 ),
wx.ALIGN_LEFT|wx.TOP|wx.BOTTOM|wx.RIGHT|wx.ALIGN_CENTER_VERTICAL, 5 )
self.hold_spin_2 = FS.FloatSpin( self.findPOC_panel_2, -1,min_val=.1, max_val=600,increment = .5, agwStyle=FS.FS_RIGHT )
self.hold_spin_2.SetFormat("%f")
self.hold_spin_2.SetDigits(2)
self.hold_spin_2.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.hold_spin_2.SetForegroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_BTNTEXT ) )
self.hold_spin_2.SetBackgroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_BTNSHADOW ) )
POC_sizer_2.Add( self.hold_spin_2, wx.GBPosition( 7, 1 ), wx.GBSpan( 1, 1 ),
wx.BOTTOM|wx.RIGHT|wx.LEFT|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.maxHold_text_2 = wx.StaticText( self.findPOC_panel_2, wx.ID_ANY, u"600", wx.DefaultPosition, wx.DefaultSize, 0 )
self.maxHold_text_2.Wrap( -1 )
self.maxHold_text_2.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.maxHold_text_2.SetForegroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_SCROLLBAR ) )
POC_sizer_2.Add( self.maxHold_text_2, wx.GBPosition( 7, 2 ), wx.GBSpan( 1, 1 ),
wx.ALL|wx.ALIGN_RIGHT|wx.ALIGN_CENTER_VERTICAL, 5 )
self.radioButton_panel_2 = wx.Panel( self.findPOC_panel_2, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.TAB_TRAVERSAL )
self.radioButton_panel_2.SetBackgroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_HIGHLIGHTTEXT ) )
radioButton_sizer_2 = wx.GridSizer( 0, 3, 0, 0 )
self.cycle_2 = wx.RadioButton( self.radioButton_panel_2, wx.ID_ANY, u"Cycle", wx.DefaultPosition, wx.DefaultSize, 0 )
self.cycle_2.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.cycle_2.SetValue(True)

radioButton_sizer_2.Add( self.cycle_2, 0, wx.ALL|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.retract_2 = wx.RadioButton( self.radioButton_panel_2, wx.ID_ANY, u"Retract", wx.DefaultPosition, wx.DefaultSize, 0 )
self.retract_2.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )

radioButton_sizer_2.Add( self.retract_2, 0, wx.ALL|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.extend_2 = wx.RadioButton( self.radioButton_panel_2, wx.ID_ANY, u"Extend", wx.DefaultPosition, wx.DefaultSize, 0 )
self.extend_2.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
radioButton_sizer_2.Add( self.extend_2, 0, wx.ALL|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.radioButton_panel_2.SetSizer( radioButton_sizer_2 )
self.radioButton_panel_2.Layout()
radioButton_sizer_2.Fit( self.radioButton_panel_2 )
POC_sizer_2.Add( self.radioButton_panel_2, wx.GBPosition( 8, 0 ), wx.GBSpan( 1, 3 ), wx.EXPAND |wx.ALL, 5 )
findPOC_sizer_2.Add( POC_sizer_2, 1, wx.ALIGN_CENTER_HORIZONTAL|wx.ALIGN_CENTER_VERTICAL, 5 )
indentButton_sizer_2 = wx.BoxSizer( wx.VERTICAL )
self.indent_Button_2 = wx.Button( self.findPOC_panel_2, 14, u"Indent", wx.DefaultPosition, wx.Size( 175,60 ), 0 )
self.indent_Button_2.SetFont( wx.Font( 18, 74, 90, 92, False, "Calibri" ) )
self.Bind(wx.EVT_BUTTON,self.displacementIndent2,id=14)
```

```
indentButton_sizer_2.Add( self.indent_Button_2, 0, wx.ALIGN_CENTER_HORIZONTAL|wx.ALL, 5 )
findPOC_sizer_2.Add( indentButton_sizer_2, 1, wx.EXPAND, 5 )
self.findPOC_panel_2.SetSizer( findPOC_sizer_2 )
self.findPOC_panel_2.Layout()
step2_sizer.Add( self.findPOC_panel_2, 1, wx.EXPAND, 5 )
self.DispMin_DispMax_panel = wx.Panel( self.step2_panel, wx.ID_ANY, wx.DefaultPosition, wx.Size( 660,-1 ), wx.SIMPLE_BORDER )
dispDisplay_sizer_2 = wx.BoxSizer( wx.HORIZONTAL )
dispDisplay_sizer_2.SetMinSize( wx.Size( -1,250 ) )
dispMin_sizer_2 = wx.FlexGridSizer( 3, 1, 0, 0 )
dispMin_sizer_2.SetFlexibleDirection( wx.BOTH )
dispMin_sizer_2.SetNonFlexibleGrowMode( wx.FLEX_GROWMODE_SPECIFIED )
self.Disp_min_text_2 = wx.StaticText( self.DispMin_DispMax_panel, wx.ID_ANY, u"Displacement Minimum", wx.DefaultPosition,
wx.DefaultSize, wx.ALIGN_CENTRE )
self.Disp_min_text_2.Wrap( -1 )
self.Disp_min_text_2.SetFont( wx.Font( 14, 74, 90, 92, False, "Calibri" ) )
self.Disp_min_text_2.SetMinSize( wx.Size( 330,-1 ) )
dispMin_sizer_2.Add( self.Disp_min_text_2, 1, wx.TOP|wx.BOTTOM, 5 )
self.displaySizer_2 = wx.FlexGridSizer( 0, 2, 0, 0 )
self.displaySizer_2.SetFlexibleDirection( wx.BOTH )
self.displaySizer_2.SetNonFlexibleGrowMode( wx.FLEX_GROWMODE_SPECIFIED )
self.dispMin_text_2 = wx.StaticText( self.DispMin_DispMax_panel, wx.ID_ANY, u"Disp (mm.)", wx.DefaultPosition, wx.DefaultSize, 0 )
self.dispMin_text_2.Wrap( -1 )
self.dispMin_text_2.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.displaySizer_2.Add( self.dispMin_text_2, 0, wx.ALL|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.forceMin_text_2 = wx.StaticText( self.DispMin_DispMax_panel, wx.ID_ANY, u"Force (mN)", wx.DefaultPosition, wx.DefaultSize, 0 )
self.forceMin_text_2.Wrap( -1 )
self.forceMin_text_2.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.displaySizer_2.Add( self.forceMin_text_2, 0, wx.ALL|wx.ALIGN_CENTER_HORIZONTAL, 5 )
displayMinDisp_box_2 = wx.TextCtrl( self.DispMin_DispMax_panel, wx.ID_ANY, wx.EmptyString, wx.DefaultPosition, wx.DefaultSize, 0 )
displayMinDisp_box_2.SetFont( wx.Font( 14, 74, 90, 90, False, "Calibri" ) )
self.displaySizer_2.Add( displayMinDisp_box_2, 0, wx.BOTTOM|wx.RIGHT|wx.LEFT, 5 )
displayMinForce_box_2 = wx.TextCtrl( self.DispMin_DispMax_panel, wx.ID_ANY, wx.EmptyString, wx.DefaultPosition, wx.DefaultSize, 0 )
displayMinForce_box_2.SetFont( wx.Font( 14, 74, 90, 90, False, "Calibri" ) )
self.displaySizer_2.Add( displayMinForce_box_2, 0, wx.BOTTOM|wx.RIGHT|wx.LEFT, 5 )
dispMin_sizer_2.Add( self.displaySizer_2, 1, wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.setMin_button_2 = wx.Button( self.DispMin_DispMax_panel, 15, u"Set Min", wx.DefaultPosition, wx.Size( 175,60 ), 0 )
self.setMin_button_2.SetFont( wx.Font( 18, 74, 90, 92, False, "Calibri" ) )
self.Bind(wx.EVT_BUTTON,self.setDispMin,id=15)

dispMin_sizer_2.Add( self.setMin_button_2, 0, wx.ALL|wx.ALIGN_CENTER_HORIZONTAL, 5 )
dispDisplay_sizer_2.Add( dispMin_sizer_2, 1, 0, 5 )
dispMax_sizer_2 = wx.FlexGridSizer( 3, 0, 0, 0 )
dispMax_sizer_2.SetFlexibleDirection( wx.BOTH )
dispMax_sizer_2.SetNonFlexibleGrowMode( wx.FLEX_GROWMODE_SPECIFIED )
self.Disp_max_text_2 = wx.StaticText( self.DispMin_DispMax_panel, wx.ID_ANY, u"Displacement Maximum", wx.DefaultPosition,
wx.DefaultSize, wx.ALIGN_CENTRE )
self.Disp_max_text_2.Wrap( -1 )
self.Disp_max_text_2.SetFont( wx.Font( 14, 74, 90, 92, False, "Calibri" ) )
self.Disp_max_text_2.SetMinSize( wx.Size( 330,-1 ) )
dispMax_sizer_2.Add( self.Disp_max_text_2, 0, wx.TOP|wx.BOTTOM, 5 )
dispMax_sizer_2_1 = wx.FlexGridSizer( 2, 2, 0, 0 )
dispMax_sizer_2_1.SetFlexibleDirection( wx.BOTH )
dispMax_sizer_2_1.SetNonFlexibleGrowMode( wx.FLEX_GROWMODE_SPECIFIED )
self.dispMax_text_2 = wx.StaticText( self.DispMin_DispMax_panel, wx.ID_ANY, u"Displ. (mm)", wx.DefaultPosition, wx.DefaultSize, 0 )
self.dispMax_text_2.Wrap( -1 )
self.dispMax_text_2.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
dispMax_sizer_2_1.Add( self.dispMax_text_2, 0, wx.ALL|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.forceMax_text_2 = wx.StaticText( self.DispMin_DispMax_panel, wx.ID_ANY, u"Force (mN)", wx.DefaultPosition, wx.DefaultSize, 0 )
self.forceMax_text_2.Wrap( -1 )
self.forceMax_text_2.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
dispMax_sizer_2_1.Add( self.forceMax_text_2, 0, wx.ALL|wx.ALIGN_CENTER_HORIZONTAL, 5 )
dispMaxDisp_box_2 = wx.TextCtrl( self.DispMin_DispMax_panel, wx.ID_ANY, wx.EmptyString, wx.DefaultPosition, wx.DefaultSize, 0 )
dispMaxDisp_box_2.SetFont( wx.Font( 14, 74, 90, 90, False, "Calibri" ) )
dispMax_sizer_2_1.Add( dispMaxDisp_box_2, 0, wx.BOTTOM|wx.RIGHT|wx.LEFT, 5 )
dispMaxForce_box_2 = wx.TextCtrl( self.DispMin_DispMax_panel, wx.ID_ANY, wx.EmptyString, wx.DefaultPosition, wx.DefaultSize, 0 )
dispMaxForce_box_2.SetFont( wx.Font( 14, 74, 90, 90, False, "Calibri" ) )
dispMax_sizer_2_1.Add( dispMaxForce_box_2, 0, wx.BOTTOM|wx.RIGHT|wx.LEFT, 5 )
dispMax_sizer_2.Add( dispMax_sizer_2_1, 0, wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.setMax_button_2 = wx.Button( self.DispMin_DispMax_panel, 16, u"Set Max", wx.DefaultPosition, wx.Size( 175,60 ), 0 )
```

```
self.setMax_button_2.SetFont( wx.Font( 18, 74, 90, 92, False, "Calibri" ) )
self.Bind(wx.EVT_BUTTON,self.setDispMax,id=16)

dispMax_sizer_2.Add( self.setMax_button_2, 0, wx.ALL|wx.ALIGN_CENTER_HORIZONTAL, 5 )
dispDisplay_sizer_2.Add( dispMax_sizer_2, 1, wx.EXPAND, 5 )
self.DispMin_DispMax_panel.SetSizer( dispDisplay_sizer_2 )
self.DispMin_DispMax_panel.Layout()
step2_sizer.Add( self.DispMin_DispMax_panel, 1, wx.EXPAND, 5 )
self.step2_panel.SetSizer( step2_sizer )
self.step2_panel.Layout()
step2_sizer.Fit( self.step2_panel )
decisionSizer.Add( self.step2_panel, 1, wx.EXPAND, 5 )
self.step3_panel = wx.Panel( self.decisionPanel, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.TAB_TRAVERSAL )
self.step3_panel.Enable( False )
self.step3_panel.Hide()
step3_sizer = wx.FlexGridSizer( 3, 1, 0, 0 )
step3_sizer.SetFlexibleDirection( wx.VERTICAL )
step3_sizer.SetNonFlexibleGrowMode( wx.FLEX_GROWMODE_SPECIFIED )
step3_sizer.SetMinSize( wx.Size( 660,600 ) )
self.pre_indent_panel_3 = wx.Panel( self.step3_panel, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.SIMPLE_BORDER )
pre_indent_sizer_3 = wx.FlexGridSizer( 4, 1, 0, 0 )
pre_indent_sizer_3.SetFlexibleDirection( wx.BOTH )
pre_indent_sizer_3.SetNonFlexibleGrowMode( wx.FLEX_GROWMODE_SPECIFIED )
pre_indent_text_sizer = wx.FlexGridSizer( 0, 3, 0, 0 )
pre_indent_text_sizer.SetFlexibleDirection( wx.BOTH )
pre_indent_text_sizer.SetNonFlexibleGrowMode( wx.FLEX_GROWMODE_SPECIFIED )
pre_indent_text_sizer.SetMinSize( wx.Size( 660,-1 ) )
self.pre_indent_text_3 = wx.StaticText( self.pre_indent_panel_3, wx.ID_ANY, u"Pre-indentation:", wx.DefaultPosition, wx.Size( -1,25 ), 0 )
self.pre_indent_text_3.Wrap( -1 )
self.pre_indent_text_3.SetFont( wx.Font( 14, 74, 90, 92, False, "Calibri" ) )
pre_indent_text_sizer.Add( self.pre_indent_text_3, 0, wx.ALL, 5 )
self.preindent_descr_3 = wx.StaticText( self.pre_indent_panel_3, wx.ID_ANY, u"This button initiates 3 max displacement stimuli in sequence",
wx.DefaultPosition, wx.Size( 475,25 ), 0 )
self.preindent_descr_3.Wrap( -1 )
self.preindent_descr_3.SetFont( wx.Font( 14, 74, 90, 90, False, "Calibri" ) )
pre_indent_text_sizer.Add( self.preindent_descr_3, 0, wx.ALL, 5 )
pre_indent_sizer_3.Add( pre_indent_text_sizer, 1, wx.EXPAND, 5 )
self.preindent_disp_3 = wx.StaticText( self.pre_indent_panel_3, wx.ID_ANY, u"Pre-Indent Displacement", wx.DefaultPosition,wx.DefaultSize,
0 )
self.preindent_disp_3.Wrap( -1 )
self.preindent_disp_3.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
pre_indent_sizer_3.Add( self.preindent_disp_3, 0, wx.ALL|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.highestDisp_3 = wx.TextCtrl( self.pre_indent_panel_3, wx.ID_ANY,u"0", wx.DefaultPosition, wx.DefaultSize, 0 )
self.highestDisp_3.SetFont( wx.Font( 14, 74, 90, 90, False, "Calibri" ) )
pre_indent_sizer_3.Add( self.highestDisp_3,0,wx.ALL|wx.ALIGN_CENTER_HORIZONTAL)#, wx.GBPosition( 1, 1 ), wx.GBSpan( 1, 1 ),
wx.ALIGN_CENTER_HORIZONTAL|wx.RIGHT|wx.LEFT, 5 )
self.start_preindent_button_3 = wx.Button( self.pre_indent_panel_3, 17, u"Start", wx.DefaultPosition, wx.DefaultSize, 0 )
self.start_preindent_button_3.SetFont( wx.Font( 18, 74, 90, 92, False, "Calibri" ) )
self.start_preindent_button_3.SetMinSize( wx.Size( 175,60 ) )
self.Bind(wx.EVT_BUTTON,self.preIndent,id=17)

pre_indent_sizer_3.Add( self.start_preindent_button_3, 0, wx.ALL|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.pre_indent_panel_3.SetSizer( pre_indent_sizer_3 )
self.pre_indent_panel_3.Layout()
pre_indent_sizer_3.Fit( self.pre_indent_panel_3 )
step3_sizer.Add( self.pre_indent_panel_3, 1, wx.EXPAND, 5 )
self.findPOC_panel_3 = wx.Panel( self.step3_panel, wx.ID_ANY, wx.DefaultPosition, wx.Size( 660,350), wx.SIMPLE_BORDER )
findPOC_sizer_3 = wx.FlexGridSizer( 3, 1, 0, 0 )
findPOC_sizer_3.SetFlexibleDirection( wx.BOTH )
findPOC_sizer_3.SetNonFlexibleGrowMode( wx.FLEX_GROWMODE_SPECIFIED )
panel_header_sizer_3 = wx.FlexGridSizer( 0, 2, 0, 0 )
panel_header_sizer_3.SetFlexibleDirection( wx.HORIZONTAL )
panel_header_sizer_3.SetNonFlexibleGrowMode( wx.FLEX_GROWMODE_SPECIFIED )
panel_header_sizer_3.SetMinSize( wx.Size( 660,-1 ) )
self.findPOC_text_3 = wx.StaticText( self.findPOC_panel_3, wx.ID_ANY, u"Find point of contact:", wx.DefaultPosition, wx.DefaultSize, 0 )
self.findPOC_text_3.Wrap( -1 )
self.findPOC_text_3.SetFont( wx.Font( 14, 74, 90, 92, False, "Calibri" ) )
panel_header_sizer_3.Add( self.findPOC_text_3, 0, 5 )
self.findPOC_text_descr_3 = wx.StaticText( self.findPOC_panel_3, wx.ID_ANY, u"Indentation parameters", wx.DefaultPosition,
wx.DefaultSize, 0 )
```

```
self.findPOC_text_descr_3.Wrap( -1 )
self.findPOC_text_descr_3.SetFont( wx.Font( 14, 74, 90, 90, False, "Calibri" ) )
self.findPOC_text_descr_3.SetMinSize( wx.Size( 450,-1 ) )
panel_header_sizer_3.Add( self.findPOC_text_descr_3, 0, 5 )
findPOC_sizer_3.Add( panel_header_sizer_3, 1, wx.EXPAND, 5 )
POC_sizer_3 = wx.GridBagSizer( 0, 0 )
POC_sizer_3.SetFlexibleDirection( wx.BOTH )
POC_sizer_3.SetNonFlexibleGrowMode( wx.FLEX_GROWMODE_SPECIFIED )
self.max_text_3 = wx.StaticText( self.findPOC_panel_3, wx.ID_ANY, u"Max", wx.DefaultPosition, wx.DefaultSize, 0 )
self.max_text_3.Wrap( -1 )
self.max_text_3.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.max_text_3.SetForegroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_SCROLLBAR ) )
POC_sizer_3.Add( self.max_text_3, wx.GBPosition( 0, 2 ), wx.GBSpan( 1, 1 ),
wx.ALIGN_RIGHT|wx.ALIGN_CENTER_VERTICAL|wx.LEFT|wx.TOP|wx.BOTTOM, 5 )
self.displacement_text_3 = wx.StaticText( self.findPOC_panel_3, wx.ID_ANY, u"displ. (mm)", wx.DefaultPosition, wx.DefaultSize,
wx.ALIGN_CENTRE )
self.displacement_text_3.Wrap( -1 )
self.displacement_text_3.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
POC_sizer_3.Add( self.displacement_text_3, wx.GBPosition( 0, 1 ), wx.GBSpan( 1, 1 ), wx.LEFT|wx.RIGHT|wx.ALIGN_BOTTOM, 5 )
self.min_text_3 = wx.StaticText( self.findPOC_panel_3, wx.ID_ANY, u"Min", wx.DefaultPosition, wx.Size( 40,-1 ), wx.ALIGN_LEFT )
self.min_text_3.Wrap( -1 )
self.min_text_3.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.min_text_3.SetForegroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_SCROLLBAR ) )
POC_sizer_3.Add( self.min_text_3, wx.GBPosition( 0, 0 ), wx.GBSpan( 1, 1 ),
wx.RIGHT|wx.ALIGN_LEFT|wx.ALIGN_CENTER_VERTICAL, 5 )
self.minDisp_text_3 = wx.StaticText( self.findPOC_panel_3, wx.ID_ANY, u"0.01", wx.DefaultPosition, wx.Size( -1,-1 ), wx.ALIGN_LEFT )
self.minDisp_text_3.Wrap( -1 )
self.minDisp_text_3.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.minDisp_text_3.SetForegroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_SCROLLBAR ) )
self.minDisp_text_3.SetMinSize( wx.Size( 100,-1 ) )

POC_sizer_3.Add( self.minDisp_text_3, wx.GBPosition( 1, 0 ), wx.GBSpan( 1, 1 ),
wx.ALIGN_LEFT|wx.ALIGN_CENTER_VERTICAL|wx.RIGHT, 5 )
self.disp_spin_3 = FS.FloatSpin( self.findPOC_panel_3,-1,value = .6, min_val=.01, max_val=5,increment = .01, agwStyle=FS.FS_RIGHT )
self.disp_spin_3.SetFormat("%f")
self.disp_spin_3.SetDigits(2)
self.disp_spin_3.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.disp_spin_3.SetForegroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_BTNTEXT ) )
self.disp_spin_3.SetBackgroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_BTNSHADOW ) )
POC_sizer_3.Add( self.disp_spin_3, wx.GBPosition( 1, 1 ), wx.GBSpan( 1, 1 ),
wx.ALIGN_CENTER_HORIZONTAL|wx.LEFT|wx.RIGHT|wx.BOTTOM, 5 )
self.minVel_text_3 = wx.StaticText( self.findPOC_panel_3, wx.ID_ANY, u"1.0", wx.DefaultPosition, wx.Size( -1,-1 ), wx.ALIGN_LEFT )
self.minVel_text_3.Wrap( -1 )
self.minVel_text_3.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.minVel_text_3.SetForegroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_SCROLLBAR ) )
POC_sizer_3.Add( self.minVel_text_3, wx.GBPosition( 3, 0 ), wx.GBSpan( 1, 1 ),
wx.ALIGN_LEFT|wx.RIGHT|wx.ALIGN_CENTER_VERTICAL, 5 )
self.max_dispText_3 = wx.StaticText( self.findPOC_panel_3, wx.ID_ANY, u"5", wx.DefaultPosition, wx.DefaultSize, wx.ALIGN_RIGHT )
self.max_dispText_3.Wrap( -1 )
self.max_dispText_3.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.max_dispText_3.SetForegroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_SCROLLBAR ) )
self.max_dispText_3.SetMinSize( wx.Size( 100,-1 ) )

POC_sizer_3.Add( self.max_dispText_3, wx.GBPosition( 1, 2 ), wx.GBSpan( 1, 1 ),
wx.LEFT|wx.ALIGN_RIGHT|wx.ALIGN_CENTER_VERTICAL, 5 )
self.vel_text_3 = wx.StaticText( self.findPOC_panel_3, wx.ID_ANY, u"velocity (mm/s)", wx.DefaultPosition, wx.DefaultSize,
wx.ALIGN_LEFT )
self.vel_text_3.Wrap( -1 )
self.vel_text_3.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
POC_sizer_3.Add( self.vel_text_3, wx.GBPosition( 2, 1 ), wx.GBSpan( 1, 1 ), wx.ALIGN_LEFT|wx.LEFT|wx.RIGHT, 5 )
self.vel_spin_3 = FS.FloatSpin( self.findPOC_panel_3, -1,value = 1, min_val=1, max_val=40.0,increment = .5, agwStyle=FS.FS_RIGHT )
self.vel_spin_3.SetFormat("%f")
self.vel_spin_3.SetDigits(2)
self.vel_spin_3.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.vel_spin_3.SetForegroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_BTNTEXT ) )
self.vel_spin_3.SetBackgroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_BTNSHADOW ) )
POC_sizer_3.Add( self.vel_spin_3, wx.GBPosition( 3, 1 ), wx.GBSpan( 1, 1 ),
wx.ALIGN_CENTER_HORIZONTAL|wx.LEFT|wx.RIGHT|wx.BOTTOM, 5 )
self.maxVel_text_3 = wx.StaticText( self.findPOC_panel_3, wx.ID_ANY, u"40", wx.DefaultPosition, wx.DefaultSize, 0 )
self.maxVel_text_3.Wrap( -1 )
```

```
self.maxVel_text_3.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.maxVel_text_3.SetForegroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_SCROLLBAR ) )
POC_sizer_3.Add( self.maxVel_text_3, wx.GBPosition( 3, 2 ), wx.GBSpan( 1, 1 ),
wx.LEFT|wx.ALIGN_RIGHT|wx.ALIGN_CENTER_VERTICAL, 5 )
self.acc_text_3 = wx.StaticText( self.findPOC_panel_3, wx.ID_ANY, u"accel. (mm/s/s)", wx.DefaultPosition, wx.DefaultSize,
wx.ALIGN_CENTRE )
self.acc_text_3.Wrap( -1 )
self.acc_text_3.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
POC_sizer_3.Add( self.acc_text_3, wx.GBPosition( 4, 1 ), wx.GBSpan( 1, 1 ),wx.RIGHT|wx.LEFT, 5 )
self.min_Acc_text_3 = wx.StaticText( self.findPOC_panel_3, wx.ID_ANY, u"1.0", wx.DefaultPosition, wx.DefaultSize, 0 )
self.min_Acc_text_3.Wrap( -1 )
self.min_Acc_text_3.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.min_Acc_text_3.SetForegroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_SCROLLBAR ) )
POC_sizer_3.Add( self.min_Acc_text_3, wx.GBPosition( 5, 0 ), wx.GBSpan( 1, 1 ),
wx.ALIGN_CENTER_VERTICAL|wx.RIGHT|wx.ALIGN_LEFT, 5 )
self.acc_spin_3 = FS.FloatSpin( self.findPOC_panel_3, -1,value = 40, min_val=1, max_val=160,increment = 1, agwStyle=FS.FS_RIGHT )
self.acc_spin_3.SetFormat("%f")
self.acc_spin_3.SetDigits(2)
self.acc_spin_3.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.acc_spin_3.SetForegroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_BTNTEXT ) )
self.acc_spin_3.SetBackgroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_BTNSHADOW ) )
POC_sizer_3.Add( self.acc_spin_3, wx.GBPosition( 5, 1 ), wx.GBSpan( 1, 1 ),
wx.ALIGN_CENTER_HORIZONTAL|wx.LEFT|wx.RIGHT|wx.BOTTOM, 5 )
self.max_Acc_text_3 = wx.StaticText( self.findPOC_panel_3, wx.ID_ANY, u"160", wx.DefaultPosition, wx.DefaultSize, 0 )
self.max_Acc_text_3.Wrap( -1 )
self.max_Acc_text_3.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.max_Acc_text_3.SetForegroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_SCROLLBAR ) )
POC_sizer_3.Add( self.max_Acc_text_3, wx.GBPosition( 5, 2 ), wx.GBSpan( 1, 1 ),
wx.LEFT|wx.ALIGN_CENTER_VERTICAL|wx.ALIGN_RIGHT, 5 )
self.hold_text_3 = wx.StaticText( self.findPOC_panel_3, wx.ID_ANY, u"hold (s) ", wx.DefaultPosition, wx.DefaultSize, wx.ALIGN_CENTRE )
self.hold_text_3.Wrap( -1 )
self.hold_text_3.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
POC_sizer_3.Add( self.hold_text_3, wx.GBPosition( 6, 1 ), wx.GBSpan( 1, 1
),wx.RIGHT|wx.LEFT|wx.ALIGN_CENTER_VERTICAL|wx.ALIGN_LEFT, 5 )
self.min_holdText_3 = wx.StaticText( self.findPOC_panel_3, wx.ID_ANY, u".1", wx.DefaultPosition, wx.DefaultSize, wx.ALIGN_LEFT )
self.min_holdText_3.Wrap( -1 )
self.min_holdText_3.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.min_holdText_3.SetForegroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_SCROLLBAR ) )
POC_sizer_3.Add( self.min_holdText_3, wx.GBPosition( 7, 0 ), wx.GBSpan( 1, 1 ),
wx.ALIGN_LEFT|wx.RIGHT|wx.ALIGN_CENTER_VERTICAL, 5 )
self.hold_spin_3 = FS.FloatSpin( self.findPOC_panel_3, -1,value = 1, min_val=.1, max_val=600,increment = .1, agwStyle=FS.FS_RIGHT )
self.hold_spin_3.SetFormat("%f")
self.hold_spin_3.SetDigits(2)
self.hold_spin_3.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.hold_spin_3.SetForegroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_BTNTEXT ) )
self.hold_spin_3.SetBackgroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_BTNSHADOW ) )
POC_sizer_3.Add( self.hold_spin_3, wx.GBPosition( 7, 1 ), wx.GBSpan( 1, 1 ),
wx.BOTTOM|wx.ALIGN_CENTER_HORIZONTAL|wx.LEFT|wx.RIGHT, 5 )
self.maxHold_3 = wx.StaticText( self.findPOC_panel_3, wx.ID_ANY, u"600", wx.DefaultPosition, wx.DefaultSize, 0 )
self.maxHold_3.Wrap( -1 )
self.maxHold_3.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.maxHold_3.SetForegroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_SCROLLBAR ) )
POC_sizer_3.Add( self.maxHold_3, wx.GBPosition( 7, 2 ), wx.GBSpan( 1, 1 ),
wx.LEFT|wx.TOP|wx.BOTTOM|wx.ALIGN_RIGHT|wx.ALIGN_CENTER_VERTICAL, 5 )
self.radioButton_panel_3 = wx.Panel( self.findPOC_panel_3, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.TAB_TRAVERSAL )
self.radioButton_panel_3.SetBackgroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_HIGHLIGHTTEXT ) )

radioButton_sizer_3 = wx.GridSizer( 0, 3, 0, 0 )
self.cycle_3 = wx.RadioButton( self.radioButton_panel_3, wx.ID_ANY, u"Cycle", wx.DefaultPosition, wx.DefaultSize, 0 )
radioButton_sizer_3.Add( self.cycle_3, 0, wx.ALL|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.cycle_3.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.retract_3 = wx.RadioButton( self.radioButton_panel_3, wx.ID_ANY, u"Retract", wx.DefaultPosition, wx.DefaultSize, 0 )
radioButton_sizer_3.Add( self.retract_3, 0, wx.ALL|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.retract_3.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.extend_3 = wx.RadioButton( self.radioButton_panel_3, wx.ID_ANY, u"Extend", wx.DefaultPosition, wx.DefaultSize, 0 )
radioButton_sizer_3.Add( self.extend_3, 0, wx.ALL|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.extend_3.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.extend_3.SetValue(True)
self.radioButton_panel_3.SetSizer( radioButton_sizer_3 )
self.radioButton_panel_3.Layout()
```

```
radioButton_sizer_3.Fit( self.radioButton_panel_3 )
POC_sizer_3.Add( self.radioButton_panel_3, wx.GBPosition( 8, 0 ), wx.GBSpan( 1, 3 ), wx.EXPAND |wx.ALL, 5 )
findPOC_sizer_3.Add( POC_sizer_3, 1, wx.ALIGN_CENTER_HORIZONTAL|wx.ALIGN_CENTER_VERTICAL, 5 )
indentButton_sizer_3 = wx.BoxSizer( wx.VERTICAL )
self.indent_Button_3 = wx.Button( self.findPOC_panel_3, 18, u"Indent", wx.DefaultPosition, wx.Size( 175,60 ), 0 )
self.indent_Button_3.SetFont( wx.Font( 18, 74, 90, 92, False, "Calibri" ) )
self.Bind(wx.EVT_BUTTON,self.displacementIndent3,id=18)
indentButton_sizer_3.Add( self.indent_Button_3, 0, wx.ALIGN_CENTER_HORIZONTAL|wx.ALL, 5 )

findPOC_sizer_3.Add( indentButton_sizer_3, 1, wx.EXPAND, 5 )
self.findPOC_panel_3.SetSizer( findPOC_sizer_3 )
self.findPOC_panel_3.Layout()
step3_sizer.Add( self.findPOC_panel_3, 1, wx.EXPAND, 5 )

self.setPOC_panel_3 = wx.Panel( self.step3_panel, wx.ID_ANY, wx.DefaultPosition, wx.Size( 660,300 ), wx.SIMPLE_BORDER )
setPOC_sizer_3 = wx.BoxSizer( wx.VERTICAL )
setPOC_sizer_3.SetMinSize( wx.Size( -1,-1 ) )
setPOC_text_sizer_3 = wx.FlexGridSizer( 3, 1, 0, 0 )
setPOC_text_sizer_3.SetFlexibleDirection( wx.BOTH )
setPOC_text_sizer_3.SetNonFlexibleGrowMode( wx.FLEX_GROWMODE_SPECIFIED )
self.setPOC_text_3 = wx.StaticText( self.setPOC_panel_3, wx.ID_ANY, u"Set point of contact and retract 0.6 mm", wx.DefaultPosition,
wx.DefaultSize, wx.ALIGN_LEFT )
self.setPOC_text_3.Wrap( -1 )
self.setPOC_text_3.SetFont( wx.Font( 14, 74, 90, 92, False, "Calibri" ) )
self.setPOC_text_3.SetMinSize( wx.Size( -1,-1 ) )
setPOC_text_sizer_3.Add( self.setPOC_text_3, 1, wx.TOP|wx.BOTTOM|wx.LEFT, 5 )
setPOC_sizer_3.Add( setPOC_text_sizer_3, 1, 0, 5 )
setPOC_button_sizer_3 = wx.FlexGridSizer( 3, 0, 0, 0 )
setPOC_button_sizer_3.SetFlexibleDirection( wx.BOTH )
setPOC_button_sizer_3.SetNonFlexibleGrowMode( wx.FLEX_GROWMODE_SPECIFIED )

self.setPOC_button_3 = wx.Button( self.setPOC_panel_3, 19, u"Set", wx.DefaultPosition, wx.Size( 175,60 ), 0 )
self.setPOC_button_3.SetFont( wx.Font( 18, 74, 90, 92, False, "Calibri" ) )
self.Bind(wx.EVT_BUTTON,self.setPointOfContactFinal,id=19)
setPOC_button_sizer_3.Add( self.setPOC_button_3, 0, wx.ALIGN_CENTER_HORIZONTAL|wx.TOP|wx.BOTTOM|
wx.ALIGN_CENTER_VERTICAL, 5 )
setPOC_sizer_3.Add( setPOC_button_sizer_3, 0, wx.ALIGN_CENTER_HORIZONTAL|wx.ALIGN_CENTER_VERTICAL, 5 )
self.setPOC_panel_3.SetSizer( setPOC_sizer_3 )
self.setPOC_panel_3.Layout()
step3_sizer.Add( self.setPOC_panel_3, 1, wx.EXPAND, 5 )

self.step3_panel.SetSizer( step3_sizer )
self.step3_panel.Layout()
step3_sizer.Fit( self.step3_panel )
decisionSizer.Add( self.step3_panel, 1, wx.EXPAND, 5 )
self.step4_panel = wx.Panel( self.decisionPanel, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.TAB_TRAVERSAL )
self.step4_panel.Enable( False )
self.step4_panel.Hide()
decision_sizer = wx.GridSizer( 0, 2, 0, 0 )
decision_sizer.SetMinSize( wx.Size( 660,630 ) )

self.forceControl_button_4 = wx.Button( self.step4_panel, 20, u"Force Control", wx.DefaultPosition, wx.Size( 175,60 ), 0 )
self.forceControl_button_4.SetFont( wx.Font( 18, 74, 90, 92, False, "Calibri" ) )
self.Bind(wx.EVT_BUTTON,self.change_forcePage,id=20)

decision_sizer.Add( self.forceControl_button_4, 0, wx.ALL|wx.ALIGN_CENTER_HORIZONTAL|wx.ALIGN_CENTER_VERTICAL, 5 )
self.dispControl_button_4 = wx.Button( self.step4_panel, 21, u"Disp. Control", wx.DefaultPosition, wx.Size( 175,60 ), 0 )
self.dispControl_button_4.SetFont( wx.Font( 18, 74, 90, 92, False, "Calibri" ) )
self.Bind(wx.EVT_BUTTON,self.change_dispPage,id=21)

decision_sizer.Add( self.dispControl_button_4, 0, wx.ALL|wx.ALIGN_CENTER_VERTICAL|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.step4_panel.SetSizer( decision_sizer )
self.step4_panel.Layout()
decision_sizer.Fit( self.step4_panel )
decisionSizer.Add( self.step4_panel, 1, wx.EXPAND|wx.ALL, 5 )
self.step5_force_panel = wx.Panel( self.decisionPanel, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.TAB_TRAVERSAL )
self.step5_force_panel.Enable( False )
self.step5_force_panel.Hide()
sizer_5 = wx.FlexGridSizer( 5, 1, 0, 0 )
sizer_5.SetFlexibleDirection( wx.BOTH )
```

```
sizer_5.SetNonFlexibleGrowMode( wx.FLEX_GROWMODE_SPECIFIED )
sizer_5.SetMinSize( wx.Size( 660,600 ) )
self.text_panel_5 = wx.Panel( self.step5_force_panel, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.TAB_TRAVERSAL )
self.text_panel_5.SetMinSize( wx.Size( 660,-1 ) )
text_sizer_5 = wx.FlexGridSizer( 0, 2, 0, 0 )
text_sizer_5.SetFlexibleDirection( wx.BOTH )
text_sizer_5.SetNonFlexibleGrowMode( wx.FLEX_GROWMODE_SPECIFIED )
text_sizer_5_sub = wx.FlexGridSizer( 0, 3, 0, 0 )
text_sizer_5_sub.SetFlexibleDirection( wx.BOTH )
text_sizer_5_sub.SetNonFlexibleGrowMode( wx.FLEX_GROWMODE_SPECIFIED )
text_sizer_5_sub.SetMinSize( wx.Size( 660,-1 ) )
self.calibrate_text_5 = wx.StaticText( self.text_panel_5, wx.ID_ANY, u"Calibrate", wx.DefaultPosition, wx.Size( -1,30 ), 0 )
self.calibrate_text_5.Wrap( -1 )
self.calibrate_text_5.SetFont( wx.Font( 14, 74, 90, 92, False, "Calibri" ) )
text_sizer_5_sub.Add( self.calibrate_text_5, 0, wx.ALL, 5 )
Calibrate_level_choiceChoices = [ u"Multiple", u"Single" ]
self.Calibrate_level_choice = wx.Choice( self.text_panel_5, 1001, wx.DefaultPosition, wx.Size( -1,-1 ), Calibrate_level_choiceChoices,
wx.CB_SORT )
self.Calibrate_level_choice.SetSelection( 0 )
self.Calibrate_level_choice.SetFont( wx.Font( 14, 74, 90, 90, False, "Calibri" ) )
self.Bind(wx.EVT_CHOICE,self.changeMult_Single,id=1001)

text_sizer_5_sub.Add( self.Calibrate_level_choice, 0, wx.ALL|wx.ALIGN_CENTER_VERTICAL|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.indentation_text_5 = wx.StaticText( self.text_panel_5, wx.ID_ANY, u"indentation levels", wx.DefaultPosition, wx.Size( 150,30 ), 0 )
self.indentation_text_5.Wrap( -1 )
self.indentation_text_5.SetFont( wx.Font( 14, 74, 90, 92, False, "Calibri" ) )
text_sizer_5_sub.Add( self.indentation_text_5, 0, wx.ALL, 5 )
text_sizer_5.Add( text_sizer_5_sub, 1, wx.EXPAND, 5 )
self.text_panel_5.SetSizer( text_sizer_5 )
self.text_panel_5.Layout()
text_sizer_5.Fit( self.text_panel_5 )
sizer_5.Add( self.text_panel_5, 0, 5 )
self.calib_level_text_5 = wx.StaticText( self.step5_force_panel, wx.ID_ANY, u"Calibration values(mN)", wx.DefaultPosition, wx.Size( -1,-1 ), 0
)
self.calib_level_text_5.Wrap( -1 )
self.calib_level_text_5.SetFont( wx.Font( 14, 74, 90, 92, False, "Calibri" ) )
sizer_5.Add( self.calib_level_text_5, 1, wx.ALIGN_CENTER_VERTICAL|wx.TOP|wx.BOTTOM|wx.RIGHT|wx.LEFT|wx.ALIGN_LEFT, 5 )

self.calibration_param_5_panel = wx.Panel( self.step5_force_panel, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.TAB_TRAVERSAL
)
self.calibration_param_5_panel.Enable( False )
self.calibration_param_5_panel.Hide()
calibration_param_5_sizer = wx.FlexGridSizer( 4, 1, 0, 0 )
calibration_param_5_sizer.SetFlexibleDirection( wx.BOTH )
calibration_param_5_sizer.SetNonFlexibleGrowMode( wx.FLEX_GROWMODE_SPECIFIED )
params_5_sizer = wx.GridBagSizer( 0, 0 )
params_5_sizer.SetFlexibleDirection( wx.BOTH )
params_5_sizer.SetNonFlexibleGrowMode( wx.FLEX_GROWMODE_SPECIFIED )
self.highest_Force_5 = wx.StaticText( self.calibration_param_5_panel, wx.ID_ANY, u"highest force", wx.DefaultPosition, wx.DefaultSize, 0 )
self.highest_Force_5.Wrap( -1 )
self.highest_Force_5.SetFont( wx.Font( 12, 74, 90, 92, False, "Calibri" ) )
params_5_sizer.Add( self.highest_Force_5, wx.GBPosition( 0, 1 ), wx.GBSpan( 1, 1 ),
wx.TOP|wx.RIGHT|wx.LEFT|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.highestForce_textCtrl_5 = wx.TextCtrl( self.calibration_param_5_panel, wx.ID_ANY, u"50", wx.DefaultPosition, wx.DefaultSize, 0 )
self.highestForce_textCtrl_5.SetFont( wx.Font( 14, 74, 90, 90, False, "Calibri" ) )
params_5_sizer.Add( self.highestForce_textCtrl_5, wx.GBPosition( 1, 1 ), wx.GBSpan( 1, 1 ),
wx.ALIGN_CENTER_HORIZONTAL|wx.RIGHT|wx.LEFT, 5 )
self.min_text_5 = wx.StaticText( self.calibration_param_5_panel, wx.ID_ANY, u"Min", wx.DefaultPosition, wx.DefaultSize, wx.ALIGN_LEFT
)
self.min_text_5.Wrap( -1 )
self.min_text_5.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.min_text_5.SetForegroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_SCROLLBAR ) )
self.min_text_5.SetMinSize( wx.Size( 100,-1 ) )

params_5_sizer.Add( self.min_text_5, wx.GBPosition( 2, 0 ), wx.GBSpan( 1, 1 ), wx.ALL|wx.ALIGN_LEFT, 5 )
self.Velocity_text_5 = wx.StaticText( self.calibration_param_5_panel, wx.ID_ANY, u"velocity (mm/s)", wx.DefaultPosition, wx.DefaultSize, 0 )
self.Velocity_text_5.Wrap( -1 )
self.Velocity_text_5.SetFont( wx.Font( 12, 74, 90, 92, False, "Calibri" ) )
params_5_sizer.Add( self.Velocity_text_5, wx.GBPosition( 2, 1 ), wx.GBSpan( 1, 1 ),
wx.ALIGN_CENTER_HORIZONTAL|wx.TOP|wx.RIGHT|wx.LEFT, 5 )
```

```
self.max_text_5 = wx.StaticText( self.calibration_param_5_panel, wx.ID_ANY, u"Max", wx.DefaultPosition, wx.DefaultSize,
wx.ALIGN_RIGHT )
self.max_text_5.Wrap( -1 )
self.max_text_5.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.max_text_5.SetForegroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_SCROLLBAR ) )
self.max_text_5.SetMinSize( wx.Size( 100,-1 ) )

params_5_sizer.Add( self.max_text_5, wx.GBPosition( 2, 2 ), wx.GBSpan( 1, 1 ), wx.ALL|wx.ALIGN_RIGHT, 5 )
self.min_val_1_5 = wx.StaticText( self.calibration_param_5_panel, wx.ID_ANY, u"1", wx.DefaultPosition, wx.DefaultSize, 0 )
self.min_val_1_5.Wrap( -1 )
self.min_val_1_5.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.min_val_1_5.SetForegroundColour((64,64,64))
params_5_sizer.Add( self.min_val_1_5, wx.GBPosition( 3, 0 ), wx.GBSpan( 1, 1 ), wx.ALL|wx.ALIGN_LEFT, 5 )
self.vel_choice_5Choices = [ u"0.5",u"0.75",u"1", u"2", u"3", u"4", u"5", wx.EmptyString ]
self.vel_choice_5 = wx.Choice( self.calibration_param_5_panel, wx.ID_ANY, wx.DefaultPosition, wx.Size( 100,-1 ), self.vel_choice_5Choices,
0 )
self.vel_choice_5.SetSelection( 0 )
self.vel_choice_5.SetFont( wx.Font( 14, 74, 90, 90, False, "Calibri" ) )
params_5_sizer.Add( self.vel_choice_5, wx.GBPosition( 3, 1 ), wx.GBSpan( 1, 1 ),
wx.ALIGN_CENTER_HORIZONTAL|wx.BOTTOM|wx.RIGHT|wx.LEFT, 5 )
self.max_val_1_5 = wx.StaticText( self.calibration_param_5_panel, wx.ID_ANY, u"5", wx.DefaultPosition, wx.DefaultSize, 0 )
self.max_val_1_5.Wrap( -1 )
self.max_val_1_5.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.max_val_1_5.SetForegroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_SCROLLBAR ) )
params_5_sizer.Add( self.max_val_1_5, wx.GBPosition( 3, 2 ), wx.GBSpan( 1, 1 ), wx.ALL|wx.ALIGN_RIGHT, 5 )
calibration_param_5_sizer.Add( params_5_sizer, 0, wx.ALIGN_CENTER_VERTICAL|wx.ALIGN_CENTER_HORIZONTAL, 5 )
calibrate_button_sizer_1 = wx.BoxSizer( wx.VERTICAL )
self.calibrate_button_5 = wx.Button( self.calibration_param_5_panel, 23, u"Calibrate", wx.DefaultPosition, wx.Size( 175,60 ), 0 )
self.calibrate_button_5.SetFont( wx.Font( 18, 74, 90, 92, False, "Calibri" ) )
self.Bind(wx.EVT_BUTTON,self.Calibrate_Single,id=23)

calibrate_button_sizer_1.Add( self.calibrate_button_5, 0, wx.ALL|wx.ALIGN_CENTER_HORIZONTAL, 5 )
calibration_param_5_sizer.Add( calibrate_button_sizer_1, 0, wx.ALIGN_CENTER_HORIZONTAL, 5 )

self.calibration_param_5_panel.SetSizer( calibration_param_5_sizer )
self.calibration_param_5_panel.Layout()
calibration_param_5_sizer.Fit( self.calibration_param_5_panel )
sizer_5.Add( self.calibration_param_5_panel, 1, wx.ALL|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.calib_param_mutliple_5 = wx.Panel( self.step5_force_panel, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.TAB_TRAVERSAL )
calib_sizer_multiple_5 = wx.FlexGridSizer( 5, 1, 0, 0 )
calib_sizer_multiple_5.SetFlexibleDirection( wx.BOTH )
calib_sizer_multiple_5.SetNonFlexibleGrowMode( wx.FLEX_GROWMODE_SPECIFIED )
partition_5_mult_sizer = wx.FlexGridSizer( 0, 2, 0, 0 )
partition_5_mult_sizer.SetFlexibleDirection( wx.BOTH )
partition_5_mult_sizer.SetNonFlexibleGrowMode( wx.FLEX_GROWMODE_SPECIFIED )
self.how_to_part_text_5_mult = wx.StaticText( self.calib_param_mutliple_5, wx.ID_ANY, u"How to partition forces?", wx.DefaultPosition,
wx.DefaultSize, 0 )
self.how_to_part_text_5_mult.Wrap( -1 )
self.how_to_part_text_5_mult.SetFont( wx.Font( 14, 74, 90, 92, False, "Calibri" ) )
partition_5_mult_sizer.Add( self.how_to_part_text_5_mult, 0, wx.ALL, 5 )
line_expo_choiceChoices = [ u"Linearly", u"Exponentially" ]
self.line_expo_choice = wx.Choice( self.calib_param_mutliple_5, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, line_expo_choiceChoices,
0 )
self.line_expo_choice.SetSelection( 0 )
self.line_expo_choice.SetFont( wx.Font( 14, 74, 90, 90, False, "Calibri" ) )
partition_5_mult_sizer.Add( self.line_expo_choice, 0, wx.ALIGN_RIGHT|wx.ALIGN_CENTER_VERTICAL|wx.ALL, 5 )
calib_sizer_multiple_5.Add( partition_5_mult_sizer, 1, wx.EXPAND, 5 )
params_5_sizer_mult = wx.GridBagSizer( 0, 0 )
params_5_sizer_mult.SetFlexibleDirection( wx.BOTH )
params_5_sizer_mult.SetNonFlexibleGrowMode( wx.FLEX_GROWMODE_SPECIFIED )
self.contact_Force_5_mult = wx.StaticText( self.calib_param_mutliple_5, wx.ID_ANY, u"contact force", wx.DefaultPosition, wx.DefaultSize, 0
)
self.contact_Force_5_mult.Wrap( -1 )
self.contact_Force_5_mult.SetFont( wx.Font( 12, 74, 90, 92, False, "Calibri" ) )

params_5_sizer_mult.Add( self.contact_Force_5_mult, wx.GBPosition( 0, 0 ), wx.GBSpan( 1, 1 ),
wx.TOP|wx.RIGHT|wx.LEFT|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.highest_Force_5_mult = wx.StaticText( self.calib_param_mutliple_5, wx.ID_ANY, u"highest force", wx.DefaultPosition, wx.DefaultSize, 0
)
self.highest_Force_5_mult.Wrap( -1 )
```

```
self.highest_Force_5_mult.SetFont( wx.Font( 12, 74, 90, 92, False, "Calibri" ) )
params_5_sizer_mult.Add( self.highest_Force_5_mult, wx.GBPosition( 0, 2 ), wx.GBSpan( 1, 1 ),
wx.TOP|wx.RIGHT|wx.LEFT|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.low_force_text_mult = wx.StaticText( self.calib_param_mutliple_5, wx.ID_ANY, u"lowest force", wx.DefaultPosition, wx.DefaultSize, 0 )
self.low_force_text_mult.Wrap( -1 )
self.low_force_text_mult.SetFont( wx.Font( 12, 74, 90, 92, False, "Calibri" ) )
params_5_sizer_mult.Add( self.low_force_text_mult, wx.GBPosition( 0, 1 ), wx.GBSpan( 1, 1 ),
wx.ALL|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.force_interval_text_mult = wx.StaticText( self.calib_param_mutliple_5, wx.ID_ANY, u"force interval", wx.DefaultPosition, wx.DefaultSize,
0 )
self.force_interval_text_mult.Wrap( -1 )
self.force_interval_text_mult.SetFont( wx.Font( 12, 74, 90, 92, False, "Calibri" ) )
params_5_sizer_mult.Add( self.force_interval_text_mult, wx.GBPosition( 0, 3 ), wx.GBSpan( 1, 1 ),
wx.ALL|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.POCForce_textCtrl_5_mult = wx.TextCtrl( self.calib_param_mutliple_5, wx.ID_ANY, u"50", wx.DefaultPosition, wx.DefaultSize,
wx.ALIGN_RIGHT|wx.TE_READONLY )
self.POCForce_textCtrl_5_mult.SetFont( wx.Font( 14, 74, 90, 90, False, "Calibri" ) )
params_5_sizer_mult.Add( self.POCForce_textCtrl_5_mult, wx.GBPosition( 1, 0 ), wx.GBSpan( 1, 1 ), wx.RIGHT|wx.LEFT, 5 )
self.lowestForce_textCtrl_5_mult = wx.TextCtrl( self.calib_param_mutliple_5, wx.ID_ANY, u"50", wx.DefaultPosition, wx.DefaultSize,
wx.ALIGN_RIGHT )
self.lowestForce_textCtrl_5_mult.SetFont( wx.Font( 14, 74, 90, 90, False, "Calibri" ) )
params_5_sizer_mult.Add( self.lowestForce_textCtrl_5_mult, wx.GBPosition( 1, 1 ), wx.GBSpan( 1, 1 ), wx.RIGHT|wx.LEFT, 5 )
self.highestForce_textCtrl_5_mult = wx.TextCtrl( self.calib_param_mutliple_5, wx.ID_ANY, u"50", wx.DefaultPosition, wx.DefaultSize,
wx.ALIGN_RIGHT)
self.highestForce_textCtrl_5_mult.SetFont( wx.Font( 14, 74, 90, 90, False, "Calibri" ) )
params_5_sizer_mult.Add( self.highestForce_textCtrl_5_mult, wx.GBPosition( 1, 2 ), wx.GBSpan( 1, 1 ),
wx.ALIGN_CENTER_HORIZONTAL|wx.RIGHT|wx.LEFT, 5 )
self.interval_textCtrl_5_mult = wx.TextCtrl( self.calib_param_mutliple_5, wx.ID_ANY, u"50", wx.DefaultPosition, wx.DefaultSize,
wx.ALIGN_RIGHT )
self.interval_textCtrl_5_mult.SetFont( wx.Font( 14, 74, 90, 90, False, "Calibri" ) )
params_5_sizer_mult.Add( self.interval_textCtrl_5_mult, wx.GBPosition( 1, 3 ), wx.GBSpan( 1, 1 ), wx.RIGHT|wx.LEFT, 5 )
calib_sizer_multiple_5.Add( params_5_sizer_mult, 1, wx.EXPAND, 5 )
params_5_sizer_mult_1 = wx.GridBagSizer( 0, 0 )
params_5_sizer_mult_1.SetFlexibleDirection( wx.BOTH )
params_5_sizer_mult_1.SetNonFlexibleGrowMode( wx.FLEX_GROWMODE_SPECIFIED )
self.min_text_5_mult = wx.StaticText( self.calib_param_mutliple_5, wx.ID_ANY, u"Min", wx.DefaultPosition, wx.DefaultSize, 0 )
self.min_text_5_mult.Wrap( -1 )
self.min_text_5_mult.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.min_text_5_mult.SetForegroundColour( (96,96,96) )
self.min_text_5_mult.SetMinSize( wx.Size(100,-1 ) )

params_5_sizer_mult_1.Add( self.min_text_5_mult, wx.GBPosition( 0, 0 ), wx.GBSpan( 1, 1 ), wx.ALL|wx.ALIGN_LEFT, 5 )
self.Velocity_text_5_mult = wx.StaticText( self.calib_param_mutliple_5, wx.ID_ANY, u"velocity (mm/s)", wx.DefaultPosition, wx.DefaultSize,
0 )
self.Velocity_text_5_mult.Wrap( -1 )
self.Velocity_text_5_mult.SetFont( wx.Font( 12, 74, 90, 92, False, "Calibri" ) )
params_5_sizer_mult_1.Add( self.Velocity_text_5_mult, wx.GBPosition( 0, 1 ), wx.GBSpan( 1, 1 ),
wx.ALIGN_CENTER_HORIZONTAL|wx.TOP|wx.RIGHT|wx.LEFT, 5 )
self.max_text_5_mult = wx.StaticText( self.calib_param_mutliple_5, wx.ID_ANY, u"Max", wx.DefaultPosition, wx.DefaultSize,
wx.ALIGN_RIGHT )
self.max_text_5_mult.Wrap( -1 )
self.max_text_5_mult.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.max_text_5_mult.SetForegroundColour( (96,96,96) )
self.max_text_5_mult.SetMinSize( wx.Size(100,-1 ) )

params_5_sizer_mult_1.Add( self.max_text_5_mult, wx.GBPosition( 0, 2 ), wx.GBSpan( 1, 1 ), wx.ALL|wx.ALIGN_RIGHT, 5 )
self.min_val_1_5_mult = wx.StaticText( self.calib_param_mutliple_5, wx.ID_ANY, u"1", wx.DefaultPosition, wx.DefaultSize, 0 )
self.min_val_1_5_mult.Wrap( -1 )
self.min_val_1_5_mult.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.min_val_1_5_mult.SetForegroundColour((96,96,96))
params_5_sizer_mult_1.Add( self.min_val_1_5_mult, wx.GBPosition( 1, 0 ), wx.GBSpan( 1, 1 ), wx.ALL|wx.ALIGN_LEFT, 5 )
self.vel_choice_5_multChoices = [ u"0.5",u"0.75",u"1", u"2", u"3", u"4", wx.EmptyString ]
self.vel_choice_5_mult = wx.Choice( self.calib_param_mutliple_5, wx.ID_ANY, wx.DefaultPosition, wx.Size( 100,-1 ),
self.vel_choice_5_multChoices, 0 )
self.vel_choice_5_mult.SetSelection( 0 )
self.vel_choice_5_mult.SetFont( wx.Font( 14, 74, 90, 90, False, "Calibri" ) )
params_5_sizer_mult_1.Add( self.vel_choice_5_mult, wx.GBPosition( 1, 1 ), wx.GBSpan( 1, 1 ),
wx.ALIGN_CENTER_HORIZONTAL|wx.BOTTOM|wx.RIGHT|wx.LEFT, 5 )
self.max_val_1_5_mult = wx.StaticText( self.calib_param_mutliple_5, wx.ID_ANY, u"5", wx.DefaultPosition, wx.DefaultSize, 0 )
self.max_val_1_5_mult.Wrap( -1 )
```

```
self.max_val_1_5_mult.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.max_val_1_5_mult.SetForegroundColour((96,96,96) )
params_5_sizer_mult_1.Add( self.max_val_1_5_mult, wx.GBPosition( 1, 2 ), wx.GBSpan( 1, 1 ), wx.ALL|wx.ALIGN_RIGHT, 5 )
calib_sizer_multiple_5.Add( params_5_sizer_mult_1, 1, wx.ALIGN_CENTER_HORIZONTAL, 5 )

calibrate_button_sizer_mult = wx.BoxSizer( wx.VERTICAL )
self.calibrate_button_5_mult = wx.Button( self.calib_param_mutliple_5, 24, u"Calibrate", wx.DefaultPosition, wx.Size( 175,60 ), 0 )
self.calibrate_button_5_mult.SetFont( wx.Font( 18, 74, 90, 92, False, "Calibri" ) )
self.Bind(wx.EVT_BUTTON,self.Calibrate_Mult,id=24)

calibrate_button_sizer_mult.Add( self.calibrate_button_5_mult, 0, wx.ALL|wx.ALIGN_CENTER_HORIZONTAL, 5 )
calib_sizer_multiple_5.Add( calibrate_button_sizer_mult, 1, wx.EXPAND, 5 )
self.calib_param_mutliple_5.SetSizer( calib_sizer_multiple_5 )
self.calib_param_mutliple_5.Layout()
calib_sizer_multiple_5.Fit( self.calib_param_mutliple_5 )
sizer_5.Add( self.calib_param_mutliple_5, 0, wx.ALL|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.m_panel79 = wx.Panel( self.step5_force_panel, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize,
wx.TAB_TRAVERSAL|wx.SIMPLE_BORDER )
bSizer55 = wx.BoxSizer( wx.VERTICAL )
self.levels_calibrate_text_6 = wx.StaticText( self.m_panel79, wx.ID_ANY, u"Force levels calibrated", wx.DefaultPosition, wx.DefaultSize,
wx.ALIGN_LEFT )
self.levels_calibrate_text_6.Wrap( -1 )
self.levels_calibrate_text_6.SetFont( wx.Font( 14, 74, 90, 92, False, "Calibri" ) )
bSizer55.Add( self.levels_calibrate_text_6, 0, wx.RIGHT|wx.LEFT, 5 )
self.calib_table_panel_5 = wx.ScrolledWindow( self.m_panel79, wx.ID_ANY, wx.DefaultPosition, (370,300),
wx.HSCROLL|wx.SIMPLE_BORDER|wx.VSCROLL )
self.calib_table_panel_5.SetScrollRate( 5, 5 )
calib_table_sizer_5 = wx.FlexGridSizer( 0, 7, 0, 0 )
calib_table_sizer_5.SetFlexibleDirection( wx.BOTH )
calib_table_sizer_5.SetNonFlexibleGrowMode( wx.FLEX_GROWMODE_SPECIFIED )
self.calib_table_panel_5.SetMinSize( wx.Size( 370,300 ) )

self.calib_ID_panel_5 = wx.Panel( self.calib_table_panel_5, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.TAB_TRAVERSAL )
self.calib_ID_panel_5.SetBackgroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_SCROLLBAR ) )
self.calib_ID_panel_5.SetMinSize( wx.Size( -1,600 ) )
self.calib_ID_sizer_5 = wx.BoxSizer( wx.VERTICAL )
self.calib_ID_sizer_5.SetMinSize( wx.Size( -1,600 ) )
self.calib_ID_header_5 = wx.StaticText( self.calib_ID_panel_5, wx.ID_ANY, u"ID", wx.DefaultPosition, wx.Size( 50,30 ),
wx.ALIGN_CENTRE )
self.calib_ID_header_5.Wrap( -1 )
self.calib_ID_header_5.SetBackgroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_SCROLLBAR ) )
self.calib_ID_sizer_5.Add( self.calib_ID_header_5, 0, wx.ALL, 5 )
self.calib_ID_staticline_5 = wx.StaticLine( self.calib_ID_panel_5, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.LI_HORIZONTAL )
self.calib_ID_sizer_5.Add( self.calib_ID_staticline_5, 0, wx.EXPAND|wx.TOP|wx.BOTTOM, 5 )
self.calib_ID_panel_5.SetSizer( self.calib_ID_sizer_5 )
self.calib_ID_panel_5.Layout()
self.calib_ID_sizer_5.Fit( self.calib_ID_panel_5 )
calib_table_sizer_5.Add( self.calib_ID_panel_5, 1, wx.EXPAND|wx.BOTTOM, 5 )
self.calib_force_panel_5 = wx.Panel( self.calib_table_panel_5, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.TAB_TRAVERSAL )
self.calib_force_panel_5.SetBackgroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_3DLIGHT ) )
self.calib_force_panel_5.SetMinSize( wx.Size( 100,600 ) )
self.calib_force_sizer_5 = wx.BoxSizer( wx.VERTICAL )
self.calib_force_sizer_5.SetMinSize( wx.Size( 100,600 ) )
self.calib_Force_header_5 = wx.StaticText( self.calib_force_panel_5, wx.ID_ANY, u"force\n(mN)", wx.DefaultPosition, wx.Size( -1,30 ),
wx.ALIGN_CENTRE )
self.calib_Force_header_5.Wrap( -1 )
self.calib_force_sizer_5.Add( self.calib_Force_header_5, 0, wx.ALL|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.calib_force_staticline = wx.StaticLine( self.calib_force_panel_5, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.LI_HORIZONTAL )
self.calib_force_sizer_5.Add( self.calib_force_staticline, 0, wx.EXPAND|wx.TOP|wx.BOTTOM, 5 )
self.calib_force_panel_5.SetSizer( self.calib_force_sizer_5 )
self.calib_force_panel_5.Layout()
self.calib_force_sizer_5.Fit( self.calib_force_panel_5 )
calib_table_sizer_5.Add( self.calib_force_panel_5, 1, wx.EXPAND|wx.BOTTOM, 5 )
self.calib_disp_panel_5 = wx.Panel( self.calib_table_panel_5, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.TAB_TRAVERSAL )
self.calib_disp_panel_5.SetBackgroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_SCROLLBAR ) )
self.calib_disp_panel_5.SetMinSize( wx.Size( -1,600 ) )
self.calib_disp_sizer_5 = wx.BoxSizer( wx.VERTICAL )
self.calib_disp_sizer_5.SetMinSize( wx.Size( 100,600 ) )
self.calib_disp_header_5 = wx.StaticText( self.calib_disp_panel_5, wx.ID_ANY, u"displacement\n(mm)", wx.DefaultPosition, wx.Size( -1,30 ),
wx.ALIGN_CENTRE )
```

```
self.calib_disp_header_5.Wrap( -1 )
self.calib_disp_sizer_5.Add( self.calib_disp_header_5, 0, wx.ALL|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.calib_disp_staticline_5 = wx.StaticLine( self.calib_disp_panel_5, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.LI_HORIZONTAL
)
self.calib_disp_sizer_5.Add( self.calib_disp_staticline_5, 0, wx.EXPAND|wx.TOP|wx.BOTTOM, 5 )
self.calib_disp_panel_5.SetSizer( self.calib_disp_sizer_5 )
self.calib_disp_panel_5.Layout()
self.calib_disp_sizer_5.Fit( self.calib_disp_panel_5 )
calib_table_sizer_5.Add( self.calib_disp_panel_5, 1, wx.EXPAND|wx.BOTTOM, 5 )
self.calib_vel_panel_5 = wx.Panel( self.calib_table_panel_5, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.TAB_TRAVERSAL )
self.calib_vel_panel_5.SetMinSize( wx.Size( -1,600 ) )
self.calib_vel_sizer_5 = wx.BoxSizer( wx.VERTICAL )
self.calib_vel_sizer_5.SetMinSize( wx.Size( 100,600 ) )
self.calib_vel_header_5 = wx.StaticText( self.calib_vel_panel_5, wx.ID_ANY, u"velocity\n(mm/s)", wx.DefaultPosition, wx.Size( 50,30 ),
wx.ALIGN_CENTRE )
self.calib_vel_header_5.Wrap( -1 )
self.calib_vel_sizer_5.Add( self.calib_vel_header_5, 0, wx.ALL|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.calib_vel_staticline_5 = wx.StaticLine( self.calib_vel_panel_5, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.LI_HORIZONTAL )
self.calib_vel_sizer_5.Add( self.calib_vel_staticline_5, 0, wx.EXPAND|wx.TOP|wx.BOTTOM, 5 )
self.calib_vel_panel_5.SetSizer( self.calib_vel_sizer_5 )
self.calib_vel_panel_5.Layout()
self.calib_vel_sizer_5.Fit( self.calib_vel_panel_5 )
calib_table_sizer_5.Add( self.calib_vel_panel_5, 1, wx.EXPAND|wx.BOTTOM, 5 )
self.calib_table_panel_5.SetSizer( calib_table_sizer_5 )
self.calib_table_panel_5.Layout()
calib_table_sizer_5.Fit( self.calib_table_panel_5 )
bSizer55.Add( self.calib_table_panel_5, 0, wx.ALIGN_CENTER_VERTICAL|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.m_panel79.SetSizer( bSizer55 )
self.m_panel79.Layout()
bSizer55.Fit( self.m_panel79 )
sizer_5.Add( self.m_panel79, 1, wx.EXPAND, 5 )
self.step5_force_panel.SetSizer( sizer_5 )
self.step5_force_panel.Layout()
sizer_5.Fit( self.step5_force_panel )
decisionSizer.Add( self.step5_force_panel, 1, wx.EXPAND, 5 )
self.step6_force_panel = wx.Notebook( self.decisionPanel, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.NB_TOP )
self.step6_force_panel.Enable( False )
self.step6_force_panel.Hide()
self.Force_Displacement = wx.Panel( self.step6_force_panel, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.TAB_TRAVERSAL )
Force_Disp_tab_sizer_6 = wx.GridBagSizer( 0, 0 )
Force_Disp_tab_sizer_6.SetFlexibleDirection( wx.BOTH )
Force_Disp_tab_sizer_6.SetNonFlexibleGrowMode( wx.FLEX_GROWMODE_SPECIFIED )
text_sizer_5_sub1 = wx.FlexGridSizer( 0, 3, 0, 0 )
text_sizer_5_sub1.SetFlexibleDirection( wx.BOTH )
text_sizer_5_sub1.SetNonFlexibleGrowMode( wx.FLEX_GROWMODE_SPECIFIED )
text_sizer_5_sub1.SetMinSize( wx.Size( 660,-1 ) )
self.calibrate_text_51 = wx.StaticText( self.Force_Displacement, wx.ID_ANY, u"Choose", wx.DefaultPosition, wx.Size( -1,30 ), 0 )
self.calibrate_text_51.Wrap( -1 )
self.calibrate_text_51.SetFont( wx.Font( 14, 74, 90, 92, False, "Calibri" ) )
text_sizer_5_sub1.Add( self.calibrate_text_51, 0, wx.ALL, 5 )
type_mec_IndentChoices = [ u"force", u"displacement" ]
self.type_mec_Indent = wx.Choice( self.Force_Displacement, wx.ID_ANY, wx.DefaultPosition, wx.Size( -1,-1 ), type_mec_IndentChoices,
wx.CB_SORT )
self.type_mec_Indent.SetSelection( 0 )
self.type_mec_Indent.SetFont( wx.Font( 14, 74, 90, 90, False, "Calibri" ) )
text_sizer_5_sub1.Add( self.type_mec_Indent, 0, wx.ALL|wx.ALIGN_CENTER_VERTICAL|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.indentation_text_51 = wx.StaticText( self.Force_Displacement, wx.ID_ANY, u"mechanical stimulation", wx.DefaultPosition, wx.Size(
200,30 ), 0 )
self.indentation_text_51.Wrap( -1 )
self.indentation_text_51.SetFont( wx.Font( 14, 74, 90, 92, False, "Calibri" ) )
text_sizer_5_sub1.Add( self.indentation_text_51, 0, wx.ALL, 5 )
Force_Disp_tab_sizer_6.Add( text_sizer_5_sub1, wx.GBPosition( 0, 0 ), wx.GBSpan( 1, 1 ), wx.EXPAND, 5 )

force_disp_indentation_parm_6 = wx.GridBagSizer( 0, 0 )
force_disp_indentation_parm_6.SetFlexibleDirection( wx.BOTH )
force_disp_indentation_parm_6.SetNonFlexibleGrowMode( wx.FLEX_GROWMODE_SPECIFIED )
self.max_text12 = wx.StaticText( self.Force_Displacement, wx.ID_ANY, u"Max", wx.DefaultPosition, wx.DefaultSize, wx.ALIGN_RIGHT)
self.max_text12.Wrap( -1 )
self.max_text12.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.max_text12.SetForegroundColour( (96,96,96) )
```

```
force_disp_indentation_parm_6.Add( self.max_text12, wx.GBPosition( 4, 2 ), wx.GBSpan( 1, 1 ),
wx.ALL|wx.ALIGN_CENTER_VERTICAL|wx.ALIGN_RIGHT, 5 )
self.displacement_text12 = wx.StaticText( self.Force_Displacement, wx.ID_ANY, u"force (mN)", wx.DefaultPosition, wx.DefaultSize,
wx.ALIGN_LEFT )
self.displacement_text12.Wrap( -1 )
self.displacement_text12.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
force_disp_indentation_parm_6.Add( self.displacement_text12, wx.GBPosition( 2, 0 ), wx.GBSpan( 1, 1 ),
wx.LEFT|wx.RIGHT|wx.ALIGN_LEFT|wx.ALIGN_BOTTOM, 5 )
self.min_text12 = wx.StaticText( self.Force_Displacement, wx.ID_ANY, u"Min", wx.DefaultPosition, wx.DefaultSize, wx.ALIGN_LEFT )
self.min_text12.Wrap( -1 )
self.min_text12.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.min_text12.SetForegroundColour((96,96,96) )
force_disp_indentation_parm_6.Add( self.min_text12, wx.GBPosition( 4, 0 ), wx.GBSpan( 1, 1 ),
wx.ALL|wx.ALIGN_LEFT|wx.ALIGN_CENTER_VERTICAL, 5 )
#self.minDisp_text12 = wx.StaticText( self.Force_Displacement, wx.ID_ANY, u"0.01", wx.DefaultPosition, wx.Size( -1,-1 ), wx.ALIGN_LEFT
)
#self.minDisp_text12.Wrap( -1 )
#self.minDisp_text12.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
#self.minDisp_text12.SetForegroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_SCROLLBAR ) )
#self.minDisp_text12.SetMinSize( wx.Size( 100,-1 ) )

#force_disp_indentation_parm_6.Add( self.minDisp_text12, wx.GBPosition( 3, 0 ), wx.GBSpan( 1, 1 ),
wx.ALIGN_CENTER_HORIZONTAL|wx.TOP|wx.BOTTOM|wx.ALIGN_CENTER_VERTICAL, 5 )
#self.minVel_text12 = wx.StaticText( self.Force_Displacement, wx.ID_ANY, u"1.0", wx.DefaultPosition, wx.Size( -1,-1 ),
wx.ALIGN_CENTRE )
#self.minVel_text12.Wrap( -1 )
#self.minVel_text12.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
#self.minVel_text12.SetForegroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_SCROLLBAR ) )
#force_disp_indentation_parm_6.Add( self.minVel_text12, wx.GBPosition( 5, 0 ), wx.GBSpan( 1, 1 ),
wx.ALIGN_CENTER_HORIZONTAL|wx.TOP|wx.BOTTOM|wx.LEFT|wx.ALIGN_CENTER_VERTICAL, 5 )
self.F_D_force_textCtrl_6 = wx.TextCtrl( self.Force_Displacement, wx.ID_ANY, wx.EmptyString, wx.DefaultPosition, wx.Size( 90,-1 ),
wx.TE_RIGHT|wx.TE_READONLY)
self.F_D_force_textCtrl_6.SetFont( wx.Font( 14, 74, 90, 90, False, "Calibri" ) )
self.F_D_force_textCtrl_6.SetBackgroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_SCROLLBAR ) )
force_disp_indentation_parm_6.Add( self.F_D_force_textCtrl_6, wx.GBPosition( 3, 0 ), wx.GBSpan( 1, 1 ),
wx.BOTTOM|wx.LEFT|wx.RIGHT|wx.ALIGN_CENTER_HORIZONTAL, 5 )

self.F_D_disp_textCtrl_6 = wx.TextCtrl( self.Force_Displacement, wx.ID_ANY, wx.EmptyString, wx.DefaultPosition, wx.Size( 90,-1 ),
wx.TE_RIGHT|wx.TE_READONLY )
self.F_D_disp_textCtrl_6.SetFont( wx.Font( 14, 74, 90, 90, False, "Calibri" ) )
self.F_D_disp_textCtrl_6.SetBackgroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_SCROLLBAR ) )
force_disp_indentation_parm_6.Add( self.F_D_disp_textCtrl_6, wx.GBPosition( 3, 1 ), wx.GBSpan( 1, 1 ),
wx.BOTTOM|wx.LEFT|wx.RIGHT|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.F_D_vel_textCtrl_6 = wx.TextCtrl( self.Force_Displacement, wx.ID_ANY, wx.EmptyString, wx.DefaultPosition, wx.Size( 90,-1 ),
wx.TE_RIGHT|wx.TE_READONLY )
self.F_D_vel_textCtrl_6.SetFont( wx.Font( 14, 74, 90, 90, False, "Calibri" ) )
self.F_D_vel_textCtrl_6.SetBackgroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_SCROLLBAR ) )
force_disp_indentation_parm_6.Add( self.F_D_vel_textCtrl_6, wx.GBPosition( 3, 2 ), wx.GBSpan( 1, 1 ),
wx.BOTTOM|wx.LEFT|wx.RIGHT|wx.ALIGN_CENTER_HORIZONTAL, 5 )
#self.max_dispText12 = wx.StaticText( self.Force_Displacement, wx.ID_ANY, u"5", wx.DefaultPosition, wx.DefaultSize, wx.ALIGN_RIGHT )
#self.max_dispText12.Wrap( -1 )
#self.max_dispText12.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
#self.max_dispText12.SetForegroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_SCROLLBAR ) )
#self.max_dispText12.SetMinSize( wx.Size( 100,-1 ) )

#force_disp_indentation_parm_6.Add( self.max_dispText12, wx.GBPosition( 3, 2 ), wx.GBSpan( 1, 1 ),
wx.ALL|wx.ALIGN_CENTER_HORIZONTAL|wx.ALIGN_CENTER_VERTICAL, 5 )
self.vel_text12 = wx.StaticText( self.Force_Displacement, wx.ID_ANY, u"velocity (mm/s)", wx.DefaultPosition, wx.DefaultSize,
wx.ALIGN_LEFT )
self.vel_text12.Wrap( -1 )
self.vel_text12.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
force_disp_indentation_parm_6.Add( self.vel_text12, wx.GBPosition( 2, 2 ), wx.GBSpan( 1, 1 ),
wx.ALIGN_BOTTOM|wx.LEFT|wx.RIGHT|wx.TOP|wx.ALIGN_LEFT, 5 )
self.acc_text12 = wx.StaticText( self.Force_Displacement, wx.ID_ANY, u"displ. (mm)", wx.DefaultPosition, wx.DefaultSize, wx.ALIGN_LEFT
)
self.acc_text12.Wrap( -1 )
self.acc_text12.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
force_disp_indentation_parm_6.Add( self.acc_text12, wx.GBPosition( 2, 1 ), wx.GBSpan( 1, 1 ),
wx.TOP|wx.RIGHT|wx.LEFT|wx.ALIGN_LEFT|wx.ALIGN_BOTTOM, 5 )
#self.maxVel_text12 = wx.StaticText( self.Force_Displacement, wx.ID_ANY, u"40", wx.DefaultPosition, wx.DefaultSize, 0 )
```

```
#self.maxVel_text12.Wrap( -1 )
#self.maxVel_text12.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
#self.maxVel_text12.SetForegroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_SCROLLBAR ) )
#force_disp_indentation_parm_6.Add( self.maxVel_text12, wx.GBPosition( 5, 2 ), wx.GBSpan( 1, 1 ),
wx.ALL|wx.ALIGN_CENTER_HORIZONTAL|wx.ALIGN_CENTER_VERTICAL, 5 )
#self.min_Acc_text12 = wx.StaticText( self.Force_Displacement, wx.ID_ANY, u"1.0", wx.DefaultPosition, wx.DefaultSize, 0 )
#self.min_Acc_text12.Wrap( -1 )
#self.min_Acc_text12.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
#self.min_Acc_text12.SetForegroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_SCROLLBAR ) )
#force_disp_indentation_parm_6.Add( self.min_Acc_text12, wx.GBPosition( 7, 0 ), wx.GBSpan( 1, 1 ),
wx.ALIGN_CENTER_HORIZONTAL|wx.TOP|wx.BOTTOM|wx.LEFT|wx.ALIGN_CENTER_VERTICAL, 5 )
#self.max_Acc_text12 = wx.StaticText( self.Force_Displacement, wx.ID_ANY, u"5", wx.DefaultPosition, wx.DefaultSize, 0 )
#self.max_Acc_text12.Wrap( -1 )
#self.max_Acc_text12.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
#self.max_Acc_text12.SetForegroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_SCROLLBAR ) )
#force_disp_indentation_parm_6.Add( self.max_Acc_text12, wx.GBPosition( 7, 2 ), wx.GBSpan( 1, 1 ),
wx.ALL|wx.ALIGN_CENTER_HORIZONTAL|wx.ALIGN_CENTER_VERTICAL, 5 )
self.hold_text12 = wx.StaticText( self.Force_Displacement, wx.ID_ANY, u"hold (s) ", wx.DefaultPosition, wx.DefaultSize, wx.ALIGN_LEFT )
self.hold_text12.Wrap( -1 )
self.hold_text12.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
force_disp_indentation_parm_6.Add( self.hold_text12, wx.GBPosition( 4, 1 ), wx.GBSpan( 1, 1 ),
wx.TOP|wx.RIGHT|wx.LEFT|wx.ALIGN_CENTER_VERTICAL|wx.ALIGN_LEFT, 5 )
self.min_holdText12 = wx.StaticText( self.Force_Displacement, wx.ID_ANY, u"1.5", wx.DefaultPosition, wx.DefaultSize, wx.ALIGN_LEFT )
self.min_holdText12.Wrap( -1 )
self.min_holdText12.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.min_holdText12.SetForegroundColour( (96,96,96))
self.min_holdText12.SetMinSize( wx.Size( 100,-1 ) )

force_disp_indentation_parm_6.Add( self.min_holdText12, wx.GBPosition( 5, 0 ), wx.GBSpan( 1, 1 ),
wx.ALIGN_CENTER_HORIZONTAL|wx.TOP|wx.BOTTOM|wx.LEFT|wx.ALIGN_CENTER_VERTICAL, 5 )
self.F_D_holdTime_6 = FS.FloatSpin( self.Force_Displacement, -1, value = 5, min_val=1.5, max_val=600,increment =
.5,agwStyle=FS.FS_RIGHT)
self.F_D_holdTime_6.SetFormat("%f")
self.F_D_holdTime_6.SetDigits(2)
self.F_D_holdTime_6.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.F_D_holdTime_6.SetForegroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_BTNTEXT ) )
self.F_D_holdTime_6.SetBackgroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_BTNSHADOW ) )
force_disp_indentation_parm_6.Add( self.F_D_holdTime_6, wx.GBPosition( 5, 1 ), wx.GBSpan( 1, 1 ),
wx.ALIGN_CENTER_HORIZONTAL|wx.BOTTOM|wx.RIGHT|wx.LEFT, 5 )
self.maxHold_text12 = wx.StaticText( self.Force_Displacement, wx.ID_ANY, u"600", wx.DefaultPosition, wx.DefaultSize, wx.ALIGN_RIGHT
)
self.maxHold_text12.Wrap( -1 )
self.maxHold_text12.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.maxHold_text12.SetForegroundColour((96,96,96))
self.maxHold_text12.SetMinSize( wx.Size( 100,-1 ) )

force_disp_indentation_parm_6.Add( self.maxHold_text12, wx.GBPosition( 5, 2 ), wx.GBSpan( 1, 1 ),
wx.ALL|wx.ALIGN_CENTER_HORIZONTAL|wx.ALIGN_CENTER_VERTICAL, 5 )
self.thickness_text12 = wx.StaticText( self.Force_Displacement, wx.ID_ANY, u"thickness", wx.DefaultPosition, wx.DefaultSize,
wx.ALIGN_LEFT )
self.thickness_text12.Wrap( -1 )
self.thickness_text12.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
force_disp_indentation_parm_6.Add( self.thickness_text12, wx.GBPosition( 6, 1 ), wx.GBSpan( 1, 1 ),
wx.TOP|wx.RIGHT|wx.LEFT|wx.ALIGN_CENTER_VERTICAL|wx.ALIGN_LEFT, 5 )
self.min_thicknessText12 = wx.StaticText( self.Force_Displacement, wx.ID_ANY, u"sylguard", wx.DefaultPosition, wx.DefaultSize,
wx.ALIGN_LEFT )
self.min_thicknessText12.Wrap( -1 )
self.min_thicknessText12.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.min_thicknessText12.SetForegroundColour((96,96,96) )
self.min_thicknessText12.SetMinSize( wx.Size( 100,-1 ) )

force_disp_indentation_parm_6.Add( self.min_thicknessText12, wx.GBPosition( 7, 0 ), wx.GBSpan( 1, 1 ),
wx.ALIGN_CENTER_HORIZONTAL|wx.TOP|wx.BOTTOM|wx.LEFT|wx.ALIGN_CENTER_VERTICAL, 5 )
self.F_D_skinThickness = wx.Slider( self.Force_Displacement, -1,0, -10,10,style=wx.SL_HORIZONTAL | wx.SL_AUTOTICKS)
force_disp_indentation_parm_6.Add( self.F_D_skinThickness, wx.GBPosition( 7, 1 ), wx.GBSpan( 1, 1 ),
wx.ALIGN_CENTER_HORIZONTAL|wx.BOTTOM|wx.RIGHT|wx.LEFT, 5 )

self.F_D_default_button = wx.Button( self.Force_Displacement, 100, u"Default", wx.DefaultPosition, wx.Size( 100,30 ), 0 )
self.F_D_default_button.SetFont( wx.Font( 14, 74, 90, 92, False, "Calibri" ) )
self.Bind(wx.EVT_BUTTON,self.restoreDefaults,id=100)
```

```
force_disp_indentation_parm_6.Add( self.F_D_default_button, wx.GBPosition( 8, 1 ), wx.GBSpan( 1, 1 ),
wx.ALIGN_CENTER_HORIZONTAL|wx.BOTTOM|wx.RIGHT|wx.LEFT, 5 )



self.maxthick_text12 = wx.StaticText( self.Force_Displacement, wx.ID_ANY, u"thick", wx.DefaultPosition, wx.DefaultSize,
wx.ALIGN_RIGHT )
self.maxthick_text12.Wrap( -1 )
self.maxthick_text12.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.maxthick_text12.SetForegroundColour((96,96,96) )
self.maxthick_text12.SetMinSize( wx.Size( 100,-1 ) )

force_disp_indentation_parm_6.Add( self.maxthick_text12, wx.GBPosition( 7, 2 ), wx.GBSpan( 1, 1 ),
wx.ALL|wx.ALIGN_CENTER_HORIZONTAL|wx.ALIGN_CENTER_VERTICAL, 5 )

F_D_ID_6Choices = []
self.F_D_ID_6 = wx.Choice( self.Force_Displacement, 24000, wx.DefaultPosition, wx.DefaultSize, F_D_ID_6Choices, 0 )
self.F_D_ID_6.SetSelection( 0 )
force_disp_indentation_parm_6.Add( self.F_D_ID_6, wx.GBPosition( 1, 1 ), wx.GBSpan( 1, 1 ),
wx.ALIGN_CENTER_HORIZONTAL|wx.RIGHT|wx.LEFT, 5 )
self.Bind(wx.EVT_CHOICE,self.chosenID,id=24000)
self.m_staticText1271 = wx.StaticText( self.Force_Displacement, wx.ID_ANY, u"ID", wx.DefaultPosition, wx.DefaultSize, 0 )
self.m_staticText1271.Wrap( -1 )
self.m_staticText1271.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
force_disp_indentation_parm_6.Add( self.m_staticText1271, wx.GBPosition( 0, 1 ), wx.GBSpan( 1, 1 ),
wx.ALIGN_CENTER_HORIZONTAL|wx.ALIGN_BOTTOM|wx.RIGHT|wx.LEFT, 5 )
Force_Disp_tab_sizer_6.Add( force_disp_indentation_parm_6, wx.GBPosition( 1, 0 ), wx.GBSpan( 1, 1 ),
wx.ALIGN_CENTER_HORIZONTAL|wx.ALIGN_CENTER_VERTICAL, 5 )
F_D_indentButton_sizer_6 = wx.BoxSizer( wx.VERTICAL )
self.F_D_Indent_button = wx.Button( self.Force_Displacement, 25, u"Indent", wx.DefaultPosition, wx.Size( 175,60 ), 0 )
self.F_D_Indent_button.SetFont( wx.Font( 18, 74, 90, 92, False, "Calibri" ) )
self.Bind(wx.EVT_BUTTON,self.runIndent_FD_6,id=25)

F_D_indentButton_sizer_6.Add( self.F_D_Indent_button, 0, wx.ALIGN_CENTER_HORIZONTAL|wx.RIGHT|wx.LEFT, 5 )
Force_Disp_tab_sizer_6.Add( F_D_indentButton_sizer_6, wx.GBPosition( 2, 0 ), wx.GBSpan( 1, 1 ), wx.EXPAND, 5 )
self.m_panel80 = wx.Panel( self.Force_Displacement, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize,
wx.TAB_TRAVERSAL|wx.SIMPLE_BORDER )
bSizer57 = wx.BoxSizer( wx.VERTICAL )
self.levels_calibrate_text_6 = wx.StaticText( self.m_panel80, wx.ID_ANY, u"Force levels calibrated", wx.DefaultPosition, wx.DefaultSize, 0 )
self.levels_calibrate_text_6.Wrap( -1 )
self.levels_calibrate_text_6.SetFont( wx.Font( 14, 74, 90, 92, False, "Calibri" ) )
bSizer57.Add( self.levels_calibrate_text_6, 0, 0, 5 )
self.F_D_table_panel_6 = wx.ScrolledWindow( self.m_panel80, wx.ID_ANY, wx.DefaultPosition, (360,300),
wx.HSCROLL|wx.SIMPLE_BORDER|wx.VSCROLL )
self.F_D_table_panel_6.SetScrollRate( 5, 5 )
F_D_table_sizer_6 = wx.FlexGridSizer( 0, 7, 0, 0 )
F_D_table_sizer_6.SetFlexibleDirection( wx.BOTH )
F_D_table_sizer_6.SetNonFlexibleGrowMode( wx.FLEX_GROWMODE_SPECIFIED )
self.F_D_table_panel_6.SetMinSize( wx.Size( 360,300 ) )
self.F_D_ID_panel_6 = wx.Panel( self.F_D_table_panel_6, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.TAB_TRAVERSAL )
self.F_D_ID_panel_6.SetBackgroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_SCROLLBAR ) )
self.F_D_ID_panel_6.SetMinSize( wx.Size( -1,700 ) )
self.F_D_ID_sizer_6 = wx.BoxSizer( wx.VERTICAL )
self.F_D_ID_sizer_6.SetMinSize( wx.Size( -1,700 ) )
self.F_D_ID_header_6 = wx.StaticText( self.F_D_ID_panel_6, wx.ID_ANY, u"ID", wx.DefaultPosition, wx.Size( 50,30 ),
wx.ALIGN_CENTRE )
self.F_D_ID_header_6.Wrap( -1 )
self.F_D_ID_header_6.SetBackgroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_SCROLLBAR ) )
self.F_D_ID_sizer_6.Add( self.F_D_ID_header_6, 0, wx.ALL, 5 )
self.type_staticline1 = wx.StaticLine( self.F_D_ID_panel_6, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.LI_HORIZONTAL )
self.F_D_ID_sizer_6.Add( self.type_staticline1, 0, wx.EXPAND|wx.TOP|wx.BOTTOM, 5 )
self.F_D_ID_panel_6.SetSizer( self.F_D_ID_sizer_6 )
self.F_D_ID_panel_6.Layout()
self.F_D_ID_sizer_6.Fit( self.F_D_ID_panel_6 )
F_D_table_sizer_6.Add( self.F_D_ID_panel_6, 1, wx.EXPAND|wx.BOTTOM, 5 )
self.F_D_force_panel_6 = wx.Panel( self.F_D_table_panel_6, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.TAB_TRAVERSAL )
self.F_D_force_panel_6.SetBackgroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_3DLIGHT ) )
self.F_D_force_panel_6.SetMinSize( wx.Size( 100,700 ) )
self.F_D_force_sizer_6 = wx.BoxSizer( wx.VERTICAL )
self.F_D_force_sizer_6.SetMinSize( wx.Size( 100,700 ) )
```

```
self.F_D_Force_header_6 = wx.StaticText( self.F_D_force_panel_6, wx.ID_ANY, u"force\n(mN)", wx.DefaultPosition, wx.Size( -1,30 ),
wx.ALIGN_CENTRE )
self.F_D_Force_header_6.Wrap( -1 )
self.F_D_force_sizer_6.Add( self.F_D_Force_header_6, 0, wx.ALL|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.time_staticline1 = wx.StaticLine( self.F_D_force_panel_6, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.LI_HORIZONTAL )
self.F_D_force_sizer_6.Add( self.time_staticline1, 0, wx.EXPAND|wx.TOP|wx.BOTTOM, 5 )

self.F_D_force_panel_6.SetSizer( self.F_D_force_sizer_6 )
self.F_D_force_panel_6.Layout()
self.F_D_force_sizer_6.Fit( self.F_D_force_panel_6 )
F_D_table_sizer_6.Add( self.F_D_force_panel_6, 1, wx.EXPAND|wx.BOTTOM, 5 )
self.F_D_disp_panel_6 = wx.Panel( self.F_D_table_panel_6, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.TAB_TRAVERSAL )
self.F_D_disp_panel_6.SetBackgroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_SCROLLBAR ) )
self.F_D_disp_panel_6.SetMinSize( wx.Size( -1,700 ) )
self.F_D_disp_sizer_6 = wx.BoxSizer( wx.VERTICAL )
self.F_D_disp_sizer_6.SetMinSize( wx.Size( 100,700 ) )
self.F_D_disp_header_6 = wx.StaticText( self.F_D_disp_panel_6, wx.ID_ANY, u"displacement\n(mm)", wx.DefaultPosition, wx.Size( -1,30 ),
wx.ALIGN_CENTRE )
self.F_D_disp_header_6.Wrap( -1 )
self.F_D_disp_sizer_6.Add( self.F_D_disp_header_6, 0, wx.ALL|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.disp_staticline1 = wx.StaticLine( self.F_D_disp_panel_6, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.LI_HORIZONTAL )
self.F_D_disp_sizer_6.Add( self.disp_staticline1, 0, wx.EXPAND|wx.TOP|wx.BOTTOM, 5 )
self.F_D_disp_panel_6.SetSizer( self.F_D_disp_sizer_6 )
self.F_D_disp_panel_6.Layout()
self.F_D_disp_sizer_6.Fit( self.F_D_disp_panel_6 )
F_D_table_sizer_6.Add( self.F_D_disp_panel_6, 1, wx.EXPAND|wx.BOTTOM, 5 )
self.F_D_vel_panel_6 = wx.Panel( self.F_D_table_panel_6, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.TAB_TRAVERSAL )
self.F_D_vel_panel_6.SetMinSize( wx.Size( -1,700 ) )
self.F_D_vel_sizer_6 = wx.BoxSizer( wx.VERTICAL )
self.F_D_vel_sizer_6.SetMinSize( wx.Size( 100,700 ) )
self.F_D_vel_header_6 = wx.StaticText( self.F_D_vel_panel_6, wx.ID_ANY, u"velocity\n(mm/s)", wx.DefaultPosition, wx.Size( 50,30 ),
wx.ALIGN_CENTRE )
self.F_D_vel_header_6.Wrap( -1 )
self.F_D_vel_sizer_6.Add( self.F_D_vel_header_6, 0, wx.ALL|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.vel_staticline1 = wx.StaticLine( self.F_D_vel_panel_6, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.LI_HORIZONTAL )
self.F_D_vel_sizer_6.Add( self.vel_staticline1, 0, wx.EXPAND|wx.TOP|wx.BOTTOM, 5 )
self.F_D_vel_panel_6.SetSizer( self.F_D_vel_sizer_6 )
self.F_D_vel_panel_6.Layout()
self.F_D_vel_sizer_6.Fit( self.F_D_vel_panel_6 )
F_D_table_sizer_6.Add( self.F_D_vel_panel_6, 1, wx.EXPAND|wx.BOTTOM, 5 )
self.F_D_table_panel_6.SetSizer( F_D_table_sizer_6 )
self.F_D_table_panel_6.Layout()
F_D_table_sizer_6.Fit( self.F_D_table_panel_6 )
bSizer57.Add( self.F_D_table_panel_6, 1, wx.ALL|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.m_panel80.SetSizer( bSizer57 )
self.m_panel80.Layout()
bSizer57.Fit( self.m_panel80 )
Force_Disp_tab_sizer_6.Add( self.m_panel80, wx.GBPosition( 3, 0 ), wx.GBSpan( 1, 1 ), wx.EXPAND, 5 )
self.Force_Displacement.SetSizer( Force_Disp_tab_sizer_6 )
self.Force_Displacement.Layout()
Force_Disp_tab_sizer_6.Fit( self.Force_Displacement )
self.step6_force_panel.AddPage( self.Force_Displacement, u"Force and Displacement", True )
self.fineDisp_step6 = wx.Panel( self.step6_force_panel, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.TAB_TRAVERSAL )
fineDisp_sizer_step6 = wx.FlexGridSizer( 0, 2, 0, 0 )
fineDisp_sizer_step6.SetFlexibleDirection( wx.BOTH )
fineDisp_sizer_step6.SetNonFlexibleGrowMode( wx.FLEX_GROWMODE_SPECIFIED )
self.fineDisp_panel_6 = wx.Panel( self.fineDisp_step6, wx.ID_ANY, wx.DefaultPosition, wx.Size( 660,350 ), wx.SIMPLE_BORDER )
self.fineDisp_panel_6.SetMinSize( wx.Size( -1,600 ) )
fineDisp_sizer_6 = wx.FlexGridSizer( 5, 1, 0, 0 )
fineDisp_sizer_6.SetFlexibleDirection( wx.BOTH )
fineDisp_sizer_6.SetNonFlexibleGrowMode( wx.FLEX_GROWMODE_SPECIFIED )
fine_indentparm_text_sizer_6 = wx.FlexGridSizer( 0, 2, 0, 0 )
fine_indentparm_text_sizer_6.SetFlexibleDirection( wx.HORIZONTAL )
fine_indentparm_text_sizer_6.SetNonFlexibleGrowMode( wx.FLEX_GROWMODE_SPECIFIED )
fine_indentparm_text_sizer_6.SetMinSize( wx.Size( 660,-1 ) )
self.fine_indent_text_6 = wx.StaticText( self.fineDisp_panel_6, wx.ID_ANY, u"Indentation parameters", wx.DefaultPosition, wx.DefaultSize, 0
)
self.fine_indent_text_6.Wrap( -1 )
self.fine_indent_text_6.SetFont( wx.Font( 14, 74, 90, 92, False, "Calibri" ) )
fine_indentparm_text_sizer_6.Add( self.fine_indent_text_6, 0, wx.ALL, 5 )
```

```
fineDisp_sizer_6.Add( fine_indentparm_text_sizer_6, 1, wx.EXPAND, 5 )
fine_parm_sizer_6 = wx.GridBagSizer( 0, 0 )
fine_parm_sizer_6.SetFlexibleDirection( wx.BOTH )
fine_parm_sizer_6.SetNonFlexibleGrowMode( wx.FLEX_GROWMODE_SPECIFIED )
self.max_text = wx.StaticText( self.fineDisp_panel_6, wx.ID_ANY, u"Max", wx.DefaultPosition, wx.DefaultSize, 0 )
self.max_text.Wrap( -1 )
self.max_text.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.max_text.SetForegroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_SCROLLBAR ) )
fine_parm_sizer_6.Add( self.max_text, wx.GBPosition( 0, 2 ), wx.GBSpan( 1, 1 ),
wx.ALIGN_RIGHT|wx.ALIGN_CENTER_VERTICAL|wx.LEFT|wx.TOP|wx.BOTTOM, 5 )
self.displacement_text = wx.StaticText( self.fineDisp_panel_6, wx.ID_ANY, u"displ. (mm)", wx.DefaultPosition, wx.DefaultSize,
wx.ALIGN_CENTRE )
self.displacement_text.Wrap( -1 )
self.displacement_text.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
fine_parm_sizer_6.Add( self.displacement_text, wx.GBPosition( 0, 1 ), wx.GBSpan( 1, 1 ), wx.LEFT|wx.RIGHT|wx.ALIGN_BOTTOM, 5 )
self.min_text = wx.StaticText( self.fineDisp_panel_6, wx.ID_ANY, u"Min", wx.DefaultPosition, wx.DefaultSize, wx.ALIGN_LEFT )
self.min_text.Wrap( -1 )
self.min_text.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.min_text.SetForegroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_SCROLLBAR ) )
fine_parm_sizer_6.Add( self.min_text, wx.GBPosition( 0, 0 ), wx.GBSpan( 1, 1 ),
wx.TOP|wx.BOTTOM|wx.RIGHT|wx.ALIGN_LEFT|wx.ALIGN_CENTER_VERTICAL, 5 )
self.minDisp_text = wx.StaticText( self.fineDisp_panel_6, wx.ID_ANY, u"0.01", wx.DefaultPosition, wx.Size( -1,-1 ), wx.ALIGN_LEFT )
self.minDisp_text.Wrap( -1 )
self.minDisp_text.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.minDisp_text.SetForegroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_SCROLLBAR ) )
self.minDisp_text.SetMinSize( wx.Size( 100,-1 ) )

fine_parm_sizer_6.Add( self.minDisp_text, wx.GBPosition( 1, 0 ), wx.GBSpan( 1, 1 ),
wx.ALIGN_LEFT|wx.RIGHT|wx.TOP|wx.BOTTOM|wx.ALIGN_CENTER_VERTICAL, 5 )
self.fine_disp_6 = FS.FloatSpin( self.fineDisp_panel_6, -1,min_val=.01, max_val=5,increment = .01, agwStyle=FS.FS_RIGHT )
self.fine_disp_6.SetFormat("%f")
self.fine_disp_6.SetDigits(2)
self.fine_disp_6.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.fine_disp_6.SetForegroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_BTNTEXT ) )
self.fine_disp_6.SetBackgroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_BTNSHADOW ) )
fine_parm_sizer_6.Add( self.fine_disp_6, wx.GBPosition( 1, 1 ), wx.GBSpan( 1, 1 ),
wx.BOTTOM|wx.RIGHT|wx.LEFT|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.minVel_text = wx.StaticText( self.fineDisp_panel_6, wx.ID_ANY, u"1.0", wx.DefaultPosition, wx.Size( -1,-1 ), wx.ALIGN_CENTRE )
self.minVel_text.Wrap( -1 )
self.minVel_text.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.minVel_text.SetForegroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_SCROLLBAR ) )
fine_parm_sizer_6.Add( self.minVel_text, wx.GBPosition( 3, 0 ), wx.GBSpan( 1, 1 ),
wx.ALIGN_LEFT|wx.TOP|wx.BOTTOM|wx.RIGHT|wx.ALIGN_CENTER_VERTICAL, 5 )
self.max_dispText = wx.StaticText( self.fineDisp_panel_6, wx.ID_ANY, u"5", wx.DefaultPosition, wx.DefaultSize, wx.ALIGN_RIGHT )
self.max_dispText.Wrap( -1 )
self.max_dispText.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.max_dispText.SetForegroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_SCROLLBAR ) )
self.max_dispText.SetMinSize( wx.Size( 100,-1 ) )

fine_parm_sizer_6.Add( self.max_dispText, wx.GBPosition( 1, 2 ), wx.GBSpan( 1, 1 ),
wx.TOP|wx.BOTTOM|wx.LEFT|wx.ALIGN_RIGHT|wx.ALIGN_CENTER_VERTICAL, 5 )
self.vel_text = wx.StaticText( self.fineDisp_panel_6, wx.ID_ANY, u"velocity (mm/s)", wx.DefaultPosition, wx.DefaultSize,
wx.ALIGN_CENTRE )
self.vel_text.Wrap( -1 )
self.vel_text.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
fine_parm_sizer_6.Add( self.vel_text, wx.GBPosition( 2, 1 ), wx.GBSpan( 1, 1 ), wx.ALIGN_BOTTOM|wx.LEFT|wx.RIGHT|wx.TOP, 5 )
self.fine_vel_6 = FS.FloatSpin( self.fineDisp_panel_6, -1,min_val=1, max_val=40.0,increment = .1, agwStyle=FS.FS_RIGHT )
self.fine_vel_6.SetFormat("%f")
self.fine_vel_6.SetDigits(2)
self.fine_vel_6.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.fine_vel_6.SetForegroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_BTNTEXT ) )
self.fine_vel_6.SetBackgroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_BTNSHADOW ) )
fine_parm_sizer_6.Add( self.fine_vel_6, wx.GBPosition( 3, 1 ), wx.GBSpan( 1, 1 ),
wx.BOTTOM|wx.RIGHT|wx.LEFT|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.maxVel_text = wx.StaticText( self.fineDisp_panel_6, wx.ID_ANY, u"40", wx.DefaultPosition, wx.DefaultSize, 0 )
self.maxVel_text.Wrap( -1 )
self.maxVel_text.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.maxVel_text.SetForegroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_SCROLLBAR ) )
fine_parm_sizer_6.Add( self.maxVel_text, wx.GBPosition( 3, 2 ), wx.GBSpan( 1, 1 ),
wx.LEFT|wx.TOP|wx.BOTTOM|wx.ALIGN_RIGHT|wx.ALIGN_CENTER_VERTICAL, 5 )
```

```
self.acc_text = wx.StaticText( self.fineDisp_panel_6, wx.ID_ANY, u"accel (mm/s/s)", wx.DefaultPosition, wx.DefaultSize,
wx.ALIGN_CENTRE )
self.acc_text.Wrap( -1 )
self.acc_text.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
fine_parm_sizer_6.Add( self.acc_text, wx.GBPosition( 4, 1 ), wx.GBSpan( 1, 1 ), wx.TOP|wx.RIGHT|wx.LEFT, 5 )
self.min_Acc_text = wx.StaticText( self.fineDisp_panel_6, wx.ID_ANY, u"1.0", wx.DefaultPosition, wx.DefaultSize, 0 )
self.min_Acc_text.Wrap( -1 )
self.min_Acc_text.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.min_Acc_text.SetForegroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_SCROLLBAR ) )
fine_parm_sizer_6.Add( self.min_Acc_text, wx.GBPosition( 5, 0 ), wx.GBSpan( 1, 1 ),
wx.ALIGN_LEFT|wx.TOP|wx.BOTTOM|wx.RIGHT|wx.ALIGN_CENTER_VERTICAL, 5 )
self.fine_acc_6 = FS.FloatSpin( self.fineDisp_panel_6, -1,min_val=1, max_val=160,increment = 1, agwStyle=FS.FS_RIGHT)
self.fine_acc_6.SetFormat("%f")
self.fine_acc_6.SetDigits(2)
self.fine_acc_6.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.fine_acc_6.SetForegroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_BTNTEXT ) )
self.fine_acc_6.SetBackgroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_BTNSHADOW ) )
fine_parm_sizer_6.Add( self.fine_acc_6, wx.GBPosition( 5, 1 ), wx.GBSpan( 1, 1 ),
wx.BOTTOM|wx.RIGHT|wx.LEFT|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.max_Acc_text = wx.StaticText( self.fineDisp_panel_6, wx.ID_ANY, u"160", wx.DefaultPosition, wx.DefaultSize, 0 )
self.max_Acc_text.Wrap( -1 )
self.max_Acc_text.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.max_Acc_text.SetForegroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_SCROLLBAR ) )
fine_parm_sizer_6.Add( self.max_Acc_text, wx.GBPosition( 5, 2 ), wx.GBSpan( 1, 1 ),
wx.TOP|wx.LEFT|wx.BOTTOM|wx.ALIGN_RIGHT|wx.ALIGN_CENTER_VERTICAL, 5 )
self.hold_text = wx.StaticText( self.fineDisp_panel_6, wx.ID_ANY, u"hold (s) ", wx.DefaultPosition, wx.DefaultSize, wx.ALIGN_CENTRE )
self.hold_text.Wrap( -1 )
self.hold_text.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
fine_parm_sizer_6.Add( self.hold_text, wx.GBPosition( 6, 1 ), wx.GBSpan( 1, 1 ),
wx.TOP|wx.RIGHT|wx.LEFT|wx.ALIGN_CENTER_VERTICAL|wx.ALIGN_LEFT, 5 )
self.min_holdText = wx.StaticText( self.fineDisp_panel_6, wx.ID_ANY, u"0.1", wx.DefaultPosition, wx.DefaultSize, wx.ALIGN_CENTRE )
self.min_holdText.Wrap( -1 )
self.min_holdText.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.min_holdText.SetForegroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_SCROLLBAR ) )
fine_parm_sizer_6.Add( self.min_holdText, wx.GBPosition( 7, 0 ), wx.GBSpan( 1, 1 ),
wx.ALIGN_LEFT|wx.TOP|wx.BOTTOM|wx.LEFT|wx.ALIGN_CENTER_VERTICAL, 5 )
self.fine_hold_6 = FS.FloatSpin( self.fineDisp_panel_6, -1,min_val=.1, max_val=600,increment = .1, agwStyle=FS.FS_RIGHT)
self.fine_hold_6.SetFormat("%f")
self.fine_hold_6.SetDigits(2)
self.fine_hold_6.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.fine_hold_6.SetForegroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_BTNTEXT ) )
self.fine_hold_6.SetBackgroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_BTNSHADOW ) )
fine_parm_sizer_6.Add( self.fine_hold_6, wx.GBPosition( 7, 1 ), wx.GBSpan( 1, 1 ),
wx.BOTTOM|wx.RIGHT|wx.LEFT|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.maxHold_text = wx.StaticText( self.fineDisp_panel_6, wx.ID_ANY, u"600", wx.DefaultPosition, wx.DefaultSize, 0 )
self.maxHold_text.Wrap( -1 )
self.maxHold_text.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.maxHold_text.SetForegroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_SCROLLBAR ) )
fine_parm_sizer_6.Add( self.maxHold_text, wx.GBPosition( 7, 2 ), wx.GBSpan( 1, 1 ),
wx.TOP|wx.BOTTOM|wx.LEFT|wx.ALIGN_RIGHT|wx.ALIGN_CENTER_VERTICAL, 5 )
self.fine_radio_panel_6 = wx.Panel( self.fineDisp_panel_6, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.TAB_TRAVERSAL )
self.fine_radio_panel_6.SetBackgroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_HIGHLIGHTTEXT ) )
fine_radio_sizer_6 = wx.GridSizer( 0, 3, 0, 0 )
self.cycle_6 = wx.RadioButton( self.fine_radio_panel_6, wx.ID_ANY, u"Cycle", wx.DefaultPosition, wx.DefaultSize, 0 )
fine_radio_sizer_6.Add( self.cycle_6, 0, wx.ALL|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.cycle_6.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.cycle_6.SetValue(True)
self.retract_6 = wx.RadioButton( self.fine_radio_panel_6, wx.ID_ANY, u"Retract", wx.DefaultPosition, wx.DefaultSize, 0 )
fine_radio_sizer_6.Add( self.retract_6, 0, wx.ALL|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.retract_6.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.extend_6 = wx.RadioButton( self.fine_radio_panel_6, wx.ID_ANY, u"Extend", wx.DefaultPosition, wx.DefaultSize, 0 )
fine_radio_sizer_6.Add( self.extend_6, 0, wx.ALL|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.extend_6.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )

self.fine_radio_panel_6.SetSizer( fine_radio_sizer_6 )
self.fine_radio_panel_6.Layout()
fine_radio_sizer_6.Fit( self.fine_radio_panel_6 )
fine_parm_sizer_6.Add( self.fine_radio_panel_6, wx.GBPosition( 8, 0 ), wx.GBSpan( 1, 3 ), wx.EXPAND |wx.ALL, 5 )
fineDisp_sizer_6.Add( fine_parm_sizer_6, 1, wx.ALIGN_CENTER_HORIZONTAL|wx.ALIGN_CENTER_VERTICAL, 5 )
fine_button_sizer_6 = wx.BoxSizer( wx.VERTICAL )
```

```
self.fine_indent_button_6 = wx.Button( self.fineDisp_panel_6, 27, u"Indent", wx.DefaultPosition, wx.Size( 175,60 ), 0 )
self.fine_indent_button_6.SetFont( wx.Font( 18, 74, 90, 92, False, "Calibri" ) )
self.Bind(wx.EVT_BUTTON,self.fine_Disp_6_Command,id=27)

fine_button_sizer_6.Add( self.fine_indent_button_6, 0, wx.ALIGN_CENTER_HORIZONTAL|wx.ALL, 5 )
fineDisp_sizer_6.Add( fine_button_sizer_6, 1, wx.EXPAND, 5 )
self.m_panel78 = wx.Panel( self.fineDisp_panel_6, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.SIMPLE_BORDER )
gbSizer16 = wx.GridBagSizer( 0, 0 )
gbSizer16.SetFlexibleDirection( wx.BOTH )
gbSizer16.SetNonFlexibleGrowMode( wx.FLEX_GROWMODE_SPECIFIED )
fine_priorText_6 = wx.BoxSizer( wx.VERTICAL )
self.prior_Disp_text_6 = wx.StaticText( self.m_panel78, wx.ID_ANY, u"Prior displacements", wx.DefaultPosition, wx.DefaultSize, 0 )
self.prior_Disp_text_6.Wrap( -1 )
self.prior_Disp_text_6.SetFont( wx.Font( 14, 74, 90, 92, False, "Calibri" ) )
fine_priorText_6.Add( self.prior_Disp_text_6, 0, wx.RIGHT|wx.LEFT, 5 )
gbSizer16.Add( fine_priorText_6, wx.GBPosition( 0, 0), wx.GBSpan( 1, 1 ), wx.EXPAND, 5 )

self.prior_disp_table_panel6 = wx.ScrolledWindow( self.m_panel78, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize ,
wx.HSCROLL|wx.SIMPLE_BORDER|wx.VSCROLL )
self.prior_disp_table_panel6.SetScrollRate( 5, 5 )
self.prior_disp_table_panel6.SetMinSize( wx.Size( 320,350 ) )
prior_disp = wx.FlexGridSizer( 0, 3, 0, 0 )
prior_disp.SetFlexibleDirection( wx.BOTH )
prior_disp.SetNonFlexibleGrowMode( wx.FLEX_GROWMODE_SPECIFIED )
self.Disp_panel_6 = wx.Panel( self.prior_disp_table_panel6, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.TAB_TRAVERSAL )
self.Disp_panel_6.SetBackgroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_SCROLLBAR ) )
self.Disp_panel_6.SetMinSize( wx.Size( -1,350 ) )
self.disp_sizer11 = wx.BoxSizer( wx.VERTICAL )
self.disp_sizer11.SetMinSize( wx.Size( -1,350 ) )
self.ID_header11 = wx.StaticText( self.Disp_panel_6, wx.ID_ANY, u"displacement (mm)", wx.DefaultPosition, wx.Size( 100,30 ),
wx.ALIGN_CENTRE )
self.ID_header11.Wrap( -1 )
self.ID_header11.SetBackgroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_SCROLLBAR ) )
self.disp_sizer11.Add( self.ID_header11, 0, wx.ALL, 5 )
self.type_staticline11 = wx.StaticLine( self.Disp_panel_6, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.LI_HORIZONTAL )
self.disp_sizer11.Add( self.type_staticline11, 0, wx.EXPAND|wx.TOP|wx.BOTTOM, 5 )
self.Disp_panel_6.SetSizer( self.disp_sizer11 )
self.Disp_panel_6.Layout()
self.disp_sizer11.Fit( self.Disp_panel_6 )
prior_disp.Add( self.Disp_panel_6, 1, wx.EXPAND|wx.BOTTOM, 5 )
self.velcity_panel_6 = wx.Panel( self.prior_disp_table_panel6, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.TAB_TRAVERSAL )
self.velcity_panel_6.SetBackgroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_3DLIGHT ) )
self.velcity_panel_6.SetMinSize( wx.Size( 100,350 ) )
self.velocity_sizer11 = wx.BoxSizer( wx.VERTICAL )
self.velocity_sizer11.SetMinSize( wx.Size( 100,350 ) )
self.force_header11 = wx.StaticText( self.velcity_panel_6, wx.ID_ANY, u"velocity\n(mm/s)", wx.DefaultPosition, wx.Size( -1,30 ),
wx.ALIGN_CENTRE )
self.force_header11.Wrap( -1 )
self.velocity_sizer11.Add( self.force_header11, 0, wx.ALL|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.time_staticline11 = wx.StaticLine( self.velcity_panel_6, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.LI_HORIZONTAL )
self.velocity_sizer11.Add( self.time_staticline11, 0, wx.EXPAND|wx.TOP|wx.BOTTOM, 5 )
self.velcity_panel_6.SetSizer( self.velocity_sizer11 )
self.velcity_panel_6.Layout()
self.velocity_sizer11.Fit( self.velcity_panel_6 )
prior_disp.Add( self.velcity_panel_6, 1, wx.EXPAND|wx.BOTTOM, 5 )
self.acceleration_panel_6 = wx.Panel( self.prior_disp_table_panel6, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.TAB_TRAVERSAL
)
self.acceleration_panel_6.SetBackgroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_SCROLLBAR ) )
self.acceleration_panel_6.SetMinSize( wx.Size( -1,350 ) )
self.acc_sizer11 = wx.BoxSizer( wx.VERTICAL )
self.acc_sizer11.SetMinSize( wx.Size( 100,350 ) )
self.disp_header11 = wx.StaticText( self.acceleration_panel_6, wx.ID_ANY, u"acceleration\n(mm/s/s)", wx.DefaultPosition, wx.Size( -1,30 ),
wx.ALIGN_CENTRE )
self.disp_header11.Wrap( -1 )
self.acc_sizer11.Add( self.disp_header11, 0, wx.ALL|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.disp_staticline11 = wx.StaticLine( self.acceleration_panel_6, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.LI_HORIZONTAL )
self.acc_sizer11.Add( self.disp_staticline11, 0, wx.EXPAND|wx.TOP|wx.BOTTOM, 5 )
self.acceleration_panel_6.SetSizer( self.acc_sizer11 )
self.acceleration_panel_6.Layout()
self.acc_sizer11.Fit( self.acceleration_panel_6 )
```

```
prior_disp.Add( self.acceleration_panel_6, 1, wx.EXPAND|wx.BOTTOM, 5 )
self.prior_disp_table_panel6.SetSizer( prior_disp )
self.prior_disp_table_panel6.Layout()
prior_disp.Fit( self.prior_disp_table_panel6 )
gbSizer16.Add( self.prior_disp_table_panel6, wx.GBPosition( 1, 1 ), wx.GBSpan( 1, 1 ), wx.EXPAND, 5 )
self.m_panel78.SetSizer( gbSizer16 )
self.m_panel78.Layout()
gbSizer16.Fit( self.m_panel78 )
fineDisp_sizer_6.Add( self.m_panel78, 1, wx.EXPAND, 5 )
self.fineDisp_panel_6.SetSizer( fineDisp_sizer_6 )
self.fineDisp_panel_6.Layout()
fineDisp_sizer_step6.Add( self.fineDisp_panel_6, 1, wx.EXPAND, 5 )
self.fineDisp_step6.SetSizer( fineDisp_sizer_step6 )
self.fineDisp_step6.Layout()
fineDisp_sizer_step6.Fit( self.fineDisp_step6 )
self.step6_force_panel.AddPage( self.fineDisp_step6, u"Fine Displacement", False )
self.yoshiExp = wx.Panel(self.step6_force_panel,wx.ID_ANY,wx.DefaultPosition,wx.DefaultSize,wx.TAB_TRAVERSAL)
yoshi_sizer = wx.FlexGridSizer(0,1,0,0)
yoshi_sizer.SetFlexibleDirection(wx.BOTH)
yoshi_sizer.SetNonFlexibleGrowMode(wx.FLEX_GROWMODE_SPECIFIED)
self.yoshiExp.SetSizer(yoshi_sizer)
self.yoshiExp.Layout()
yoshi_sizer.Fit( self.yoshiExp )
self.step6_force_panel.AddPage( self.yoshiExp, u"Yoshi Exp Beta", False )
self.indentation_plot_6 = wx.Panel( self.step6_force_panel, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.TAB_TRAVERSAL )
indentation_plot_sizer_6 = wx.FlexGridSizer( 0, 1, 0, 0 )
indentation_plot_sizer_6.SetFlexibleDirection( wx.BOTH )
indentation_plot_sizer_6.SetNonFlexibleGrowMode( wx.FLEX_GROWMODE_SPECIFIED )
self.ip_select_text = wx.StaticText( self.indentation_plot_6, wx.ID_ANY, u"Select plots to view", wx.DefaultPosition, wx.Size( 660,-1 ), 0 )
self.ip_select_text.Wrap( -1 )
self.ip_select_text.SetFont( wx.Font( 14, 74, 90, 92, False, "Calibri" ) )
indentation_plot_sizer_6.Add( self.ip_select_text, 0, wx.TOP|wx.BOTTOM|wx.RIGHT, 5 )
self.ip_force_disp_plot = wx.StaticText( self.indentation_plot_6, wx.ID_ANY, u"Force and displacement plots", wx.DefaultPosition,
wx.DefaultSize, 0 )
self.ip_force_disp_plot.Wrap( -1 )
self.ip_force_disp_plot.SetFont( wx.Font( 14, 74, 90, 92, False, "Calibri" ) )
indentation_plot_sizer_6.Add( self.ip_force_disp_plot, 0, wx.RIGHT, 5 )
self.window_6_f_d = wx.ScrolledWindow( self.indentation_plot_6, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize,
wx.HSCROLL|wx.SIMPLE_BORDER|wx.VSCROLL )
self.window_6_f_d.SetScrollRate( 5, 5 )
self.window_6_f_d.SetMinSize( wx.Size( 430,200 ) )

self.sizer_6_f_d = wx.FlexGridSizer( 0, 4, 0, 0 )
self.sizer_6_f_d.SetFlexibleDirection( wx.BOTH )
self.sizer_6_f_d.SetNonFlexibleGrowMode( wx.FLEX_GROWMODE_SPECIFIED )
self.type_panel_f_d_6 = wx.Panel( self.window_6_f_d, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.TAB_TRAVERSAL )
self.type_panel_f_d_6.SetBackgroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_SCROLLBAR ) )
self.type_panel_f_d_6.SetMinSize( wx.Size( -1,200 ) )
self.type_sizer_f_d_6 = wx.BoxSizer( wx.VERTICAL )
self.type_header_f_d_6 = wx.StaticText( self.type_panel_f_d_6, wx.ID_ANY, u"View \n Force | Disp.", wx.DefaultPosition, wx.Size( 100,30 ),
wx.ALIGN_CENTRE )
self.type_header_f_d_6.Wrap( -1 )
self.type_header_f_d_6.SetBackgroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_SCROLLBAR ) )
self.type_sizer_f_d_6.Add( self.type_header_f_d_6, 0, wx.TOP|wx.RIGHT|wx.LEFT, 5 )
self.type_line_f_d_6 = wx.StaticLine( self.type_panel_f_d_6, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.LI_HORIZONTAL )
self.type_sizer_f_d_6.Add( self.type_line_f_d_6, 0, wx.EXPAND|wx.TOP|wx.BOTTOM, 5 )
self.type_gridSizer_f_d_6 = wx.GridSizer( 0, 2, 0, 0 )
noText = wx.StaticText( self.type_panel_f_d_6, wx.ID_ANY, u" ", wx.DefaultPosition, wx.DefaultSize, wx.ALIGN_LEFT )
self.type_gridSizer_f_d_6.Add(noText,0,wx.ALL|wx.ALIGN_CENTER_HORIZONTAL, 5)
self.type_sizer_f_d_6.Add( self.type_gridSizer_f_d_6, 1, wx.EXPAND, 5 )
self.type_panel_f_d_6.SetSizer( self.type_sizer_f_d_6 )
self.type_panel_f_d_6.Layout()
self.type_sizer_f_d_6.Fit( self.type_panel_f_d_6 )
self.sizer_6_f_d.Add( self.type_panel_f_d_6, 1, wx.ALIGN_CENTER_HORIZONTAL|wx.EXPAND, 5 )
self.force_panel_f_d_6 = wx.Panel( self.window_6_f_d, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.TAB_TRAVERSAL )
self.force_panel_f_d_6.SetBackgroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_3DLIGHT ) )
self.force_panel_f_d_6.SetMinSize( wx.Size( 100,200 ) )
self.force_sizer_f_d_6 = wx.BoxSizer( wx.VERTICAL )
self.force_sizer_f_d_6.SetMinSize( wx.Size( 100,-1 ) )
```

```
self.force_header_f_d_6 = wx.StaticText( self.force_panel_f_d_6, wx.ID_ANY, u"Force\n(mN)", wx.DefaultPosition, wx.Size( -1,30 ),
wx.ALIGN_CENTRE )
self.force_header_f_d_6.Wrap( -1 )
self.force_sizer_f_d_6.Add( self.force_header_f_d_6, 0, wx.ALIGN_CENTER_HORIZONTAL|wx.TOP|wx.RIGHT|wx.LEFT, 5 )
self.time_line_f_d_6 = wx.StaticLine( self.force_panel_f_d_6, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.LI_HORIZONTAL )
self.force_sizer_f_d_6.Add( self.time_line_f_d_6, 0, wx.EXPAND|wx.TOP|wx.BOTTOM, 5 )
self.force_panel_f_d_6.SetSizer( self.force_sizer_f_d_6 )
self.force_panel_f_d_6.Layout()
self.force_sizer_f_d_6.Fit( self.force_panel_f_d_6 )
self.sizer_6_f_d.Add( self.force_panel_f_d_6, 1, wx.EXPAND, 5 )
self.disp_panel_f_d_6 = wx.Panel( self.window_6_f_d, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.TAB_TRAVERSAL )
self.disp_panel_f_d_6.SetBackgroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_SCROLLBAR ) )
self.disp_panel_f_d_6.SetMinSize( wx.Size( -1,200 ) )
self.disp_sizer_f_d_6 = wx.BoxSizer( wx.VERTICAL )
self.disp_sizer_f_d_6.SetMinSize( wx.Size( 100,-1 ) )
self.disp_header_f_d_6 = wx.StaticText( self.disp_panel_f_d_6, wx.ID_ANY, u"Displacement\n(mm)", wx.DefaultPosition, wx.Size( -1,30 ),
wx.ALIGN_CENTRE )
self.disp_header_f_d_6.Wrap( -1 )
self.disp_sizer_f_d_6.Add( self.disp_header_f_d_6, 0, wx.ALIGN_CENTER_HORIZONTAL|wx.TOP|wx.RIGHT|wx.LEFT, 5 )
self.disp_line_f_d_6 = wx.StaticLine( self.disp_panel_f_d_6, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.LI_HORIZONTAL )
self.disp_sizer_f_d_6.Add( self.disp_line_f_d_6, 0, wx.EXPAND|wx.TOP|wx.BOTTOM, 5 )
self.disp_panel_f_d_6.SetSizer( self.disp_sizer_f_d_6 )
self.disp_panel_f_d_6.Layout()
self.disp_sizer_f_d_6.Fit( self.disp_panel_f_d_6 )
self.sizer_6_f_d.Add( self.disp_panel_f_d_6, 1, wx.EXPAND, 5 )
self.velocity_panel_f_d_6 = wx.Panel( self.window_6_f_d, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.TAB_TRAVERSAL )
self.velocity_panel_f_d_6.SetBackgroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_3DLIGHT ) )
self.velocity_panel_f_d_6.SetMinSize( wx.Size( -1,200 ) )
self.velocity_sizer_f_d_6 = wx.BoxSizer( wx.VERTICAL )
self.velocity_sizer_f_d_6.SetMinSize( wx.Size( -1,200 ) )
self.velocity_header_f_d_6 = wx.StaticText( self.velocity_panel_f_d_6, wx.ID_ANY, u"Velocity \n (mm/s)", wx.DefaultPosition, wx.Size(
100,30 ), wx.ALIGN_CENTRE )
self.velocity_header_f_d_6.Wrap( -1 )
self.velocity_header_f_d_6.SetBackgroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_3DLIGHT ) )
self.velocity_sizer_f_d_6.Add( self.velocity_header_f_d_6, 0, wx.TOP|wx.RIGHT|wx.LEFT, 5 )
self.velocity_line_f_d_6 = wx.StaticLine( self.velocity_panel_f_d_6, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.LI_HORIZONTAL )
self.velocity_sizer_f_d_6.Add( self.velocity_line_f_d_6, 0, wx.EXPAND|wx.TOP|wx.BOTTOM, 5 )
self.velocity_panel_f_d_6.SetSizer( self.velocity_sizer_f_d_6 )
self.velocity_panel_f_d_6.Layout()
self.velocity_sizer_f_d_6.Fit( self.velocity_panel_f_d_6 )
self.sizer_6_f_d.Add( self.velocity_panel_f_d_6, 1, wx.EXPAND, 5 )
self.window_6_f_d.SetSizer( self.sizer_6_f_d )
self.window_6_f_d.Layout()
self.sizer_6_f_d.Fit( self.window_6_f_d )
indentation_plot_sizer_6.Add( self.window_6_f_d, 0,
wx.ALIGN_CENTER_HORIZONTAL|wx.ALIGN_CENTER_VERTICAL|wx.BOTTOM|wx.RIGHT|wx.LEFT, 5 )
self.fine_disp_text_6 = wx.StaticText( self.indentation_plot_6, wx.ID_ANY, u"Fine displacement plots", wx.DefaultPosition, wx.DefaultSize, 0 )
self.fine_disp_text_6.Wrap( -1 )
self.fine_disp_text_6.SetFont( wx.Font( 14, 74, 90, 92, False, "Calibri" ) )
indentation_plot_sizer_6.Add( self.fine_disp_text_6, 0, wx.RIGHT, 5 )
self.window_6_fd = wx.ScrolledWindow( self.indentation_plot_6, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize,
wx.HSCROLL|wx.SIMPLE_BORDER|wx.VSCROLL )
self.window_6_fd.SetScrollRate( 5, 5 )
self.window_6_fd.SetMinSize( wx.Size( 430,200 ) )

sizer_6_fd = wx.FlexGridSizer( 0, 7, 0, 0 )
sizer_6_fd.SetFlexibleDirection( wx.BOTH )
sizer_6_fd.SetNonFlexibleGrowMode( wx.FLEX_GROWMODE_SPECIFIED )
self.type_panel_fd_6 = wx.Panel( self.window_6_fd, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.TAB_TRAVERSAL )
self.type_panel_fd_6.SetBackgroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_SCROLLBAR ) )
self.type_panel_fd_6.SetMinSize( wx.Size( -1,200 ) )
self.type_sizer_fd_6 = wx.BoxSizer( wx.VERTICAL )
self.type_sizer_fd_6.SetMinSize( wx.Size( -1,200 ) )
self.type_header_fd_6 = wx.StaticText( self.type_panel_fd_6, wx.ID_ANY, u"View ", wx.DefaultPosition, wx.Size( 100,30 ),
wx.ALIGN_CENTRE )
self.type_header_fd_6.Wrap( -1 )
self.type_header_fd_6.SetBackgroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_SCROLLBAR ) )
self.type_sizer_fd_6.Add( self.type_header_fd_6, 0, wx.TOP|wx.RIGHT|wx.LEFT, 5 )
self.type_line_fd_6 = wx.StaticLine( self.type_panel_fd_6, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.LI_HORIZONTAL )
self.type_sizer_fd_6.Add( self.type_line_fd_6, 0, wx.EXPAND|wx.TOP|wx.BOTTOM, 5 )
```

```
self.type_panel_fd_6.SetSizer( self.type_sizer_fd_6 )
self.type_panel_fd_6.Layout()
self.type_sizer_fd_6.Fit( self.type_panel_fd_6 )
sizer_6_fd.Add( self.type_panel_fd_6, 1, wx.ALIGN_CENTER_HORIZONTAL|wx.EXPAND, 5 )
self.disp_panel_fd_6 = wx.Panel( self.window_6_fd, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.TAB_TRAVERSAL )
self.disp_panel_fd_6.SetBackgroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_3DLIGHT ) )
self.disp_panel_fd_6.SetMinSize( wx.Size( 100,200 ) )
self.disp_sizer_fd_6 = wx.BoxSizer( wx.VERTICAL )
self.disp_sizer_fd_6.SetMinSize( wx.Size( 100,-1 ) )
self.disp_header_fd_6 = wx.StaticText( self.disp_panel_fd_6, wx.ID_ANY, u"Displacement \n (mm)", wx.DefaultPosition, wx.Size( -1,30 ),
wx.ALIGN_CENTRE )
self.disp_header_fd_6.Wrap( -1 )
self.disp_sizer_fd_6.Add( self.disp_header_fd_6, 0, wx.ALIGN_CENTER_HORIZONTAL|wx.TOP|wx.RIGHT|wx.LEFT, 5 )
self.disp_line_fd_6 = wx.StaticLine( self.disp_panel_fd_6, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.LI_HORIZONTAL )
self.disp_sizer_fd_6.Add( self.disp_line_fd_6, 0, wx.EXPAND|wx.TOP|wx.BOTTOM, 5 )
self.disp_panel_fd_6.SetSizer( self.disp_sizer_fd_6 )
self.disp_panel_fd_6.Layout()
self.disp_sizer_fd_6.Fit( self.disp_panel_fd_6 )
sizer_6_fd.Add( self.disp_panel_fd_6, 1, wx.EXPAND, 5 )
self.vel_panel_fd_6 = wx.Panel( self.window_6_fd, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.TAB_TRAVERSAL )
self.vel_panel_fd_6.SetBackgroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_SCROLLBAR ) )
self.vel_panel_fd_6.SetMinSize( wx.Size( -1,350 ) )
self.vel_sizer_fd_6 = wx.BoxSizer( wx.VERTICAL )
self.vel_sizer_fd_6.SetMinSize( wx.Size( 100,-1 ) )
self.vel_header_fd_6 = wx.StaticText( self.vel_panel_fd_6, wx.ID_ANY, u"Velocity\n(mm/s)", wx.DefaultPosition, wx.Size( -1,30 ),
wx.ALIGN_CENTRE )
self.vel_header_fd_6.Wrap( -1 )
self.vel_sizer_fd_6.Add( self.vel_header_fd_6, 0, wx.ALIGN_CENTER_HORIZONTAL|wx.TOP|wx.RIGHT|wx.LEFT, 5 )
self.vel_line_fd_6 = wx.StaticLine( self.vel_panel_fd_6, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.LI_HORIZONTAL )
self.vel_sizer_fd_6.Add( self.vel_line_fd_6, 0, wx.EXPAND|wx.TOP|wx.BOTTOM, 5 )
self.vel_panel_fd_6.SetSizer( self.vel_sizer_fd_6 )
self.vel_panel_fd_6.Layout()
self.vel_sizer_fd_6.Fit( self.vel_panel_fd_6 )
sizer_6_fd.Add( self.vel_panel_fd_6, 1, wx.EXPAND, 5 )
self.acc_panel_fd_6 = wx.Panel( self.window_6_fd, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.TAB_TRAVERSAL )
self.acc_panel_fd_6.SetBackgroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_3DLIGHT ) )
self.acc_panel_fd_6.SetMinSize( wx.Size( -1,350 ) )
self.acc_sizer_fd_6 = wx.BoxSizer( wx.VERTICAL )
self.acc_header_fd_6 = wx.StaticText( self.acc_panel_fd_6, wx.ID_ANY, u"Acceleration \n (mm/s/s)", wx.DefaultPosition, wx.Size( 100,30 ),
wx.ALIGN_CENTRE )
self.acc_header_fd_6.Wrap( -1 )
self.acc_header_fd_6.SetBackgroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_3DLIGHT ) )
self.acc_sizer_fd_6.Add( self.acc_header_fd_6, 0, wx.TOP|wx.RIGHT|wx.LEFT, 5 )
self.acc_line_fd_6 = wx.StaticLine( self.acc_panel_fd_6, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.LI_HORIZONTAL )
self.acc_sizer_fd_6.Add( self.acc_line_fd_6, 0, wx.EXPAND|wx.TOP|wx.BOTTOM, 5 )
self.acc_panel_fd_6.SetSizer( self.acc_sizer_fd_6 )
self.acc_panel_fd_6.Layout()
self.acc_sizer_fd_6.Fit( self.acc_panel_fd_6 )
sizer_6_fd.Add( self.acc_panel_fd_6, 1, wx.EXPAND, 5 )
self.window_6_fd.SetSizer( sizer_6_fd )
self.window_6_fd.Layout()
sizer_6_fd.Fit( self.window_6_fd )
indentation_plot_sizer_6.Add( self.window_6_fd, 0, wx.BOTTOM|wx.RIGHT|wx.LEFT|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.viewButton_ip_6 = wx.Button( self.indentation_plot_6, 28, u"View", wx.DefaultPosition, wx.Size( 175,60 ), 0 )
self.viewButton_ip_6.SetFont( wx.Font( 18, 74, 90, 92, False, "Calibri" ) )
self.Bind(wx.EVT_BUTTON,self.view_ip,id=28)

indentation_plot_sizer_6.Add( self.viewButton_ip_6, 0, wx.ALL|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.indentation_plot_6.SetSizer( indentation_plot_sizer_6 )
self.indentation_plot_6.Layout()
indentation_plot_sizer_6.Fit( self.indentation_plot_6 )
self.step6_force_panel.AddPage( self.indentation_plot_6, u"Indentation Plots", False )
decisionSizer.Add( self.step6_force_panel, 1, wx.EXPAND, 5 )


self.step5_disp_panel = wx.Notebook( self.decisionPanel, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.NB_TOP)
self.step5_disp_panel.Enable( False )
self.step5_disp_panel.Hide()
self.fine_disp_panel_5 = wx.Panel( self.step5_disp_panel, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.TAB_TRAVERSAL )
fine_disp_sizer_5 = wx.FlexGridSizer( 0, 1, 0, 0 )
```

```
fine_disp_sizer_5.SetFlexibleDirection( wx.BOTH )
fine_disp_sizer_5.SetNonFlexibleGrowMode( wx.FLEX_GROWMODE_SPECIFIED )

self.indentParm_text_5 = wx.StaticText( self.fine_disp_panel_5, wx.ID_ANY, u"Indentation parameters", wx.DefaultPosition, wx.Size( 660,-1 ),
0 )
self.indentParm_text_5.Wrap( -1 )
self.indentParm_text_5.SetFont( wx.Font( 14, 74, 90, 92, False, "Calibri" ) )
fine_disp_sizer_5.Add( self.indentParm_text_5, 0, wx.TOP|wx.BOTTOM|wx.LEFT, 5 )
param_dp_5 = wx.GridBagSizer( 0, 0 )
param_dp_5.SetFlexibleDirection( wx.BOTH )
param_dp_5.SetNonFlexibleGrowMode( wx.FLEX_GROWMODE_SPECIFIED )
self.max_text_dp_5 = wx.StaticText( self.fine_disp_panel_5, wx.ID_ANY, u"Max", wx.DefaultPosition, wx.DefaultSize, 0 )
self.max_text_dp_5.Wrap( -1 )
self.max_text_dp_5.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.max_text_dp_5.SetForegroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_SCROLLBAR ) )
param_dp_5.Add( self.max_text_dp_5, wx.GBPosition( 0, 2 ), wx.GBSpan( 1, 1 ),
wx.ALIGN_RIGHT|wx.ALIGN_CENTER_VERTICAL|wx.LEFT|wx.BOTTOM|wx.TOP, 5 )
self.displacement_text_dp_5 = wx.StaticText( self.fine_disp_panel_5, wx.ID_ANY, u"displ. (mm)", wx.DefaultPosition, wx.DefaultSize,
wx.ALIGN_CENTRE )
self.displacement_text_dp_5.Wrap( -1 )
self.displacement_text_dp_5.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
param_dp_5.Add( self.displacement_text_dp_5, wx.GBPosition( 0, 1 ), wx.GBSpan( 1, 1 ),
wx.LEFT|wx.RIGHT|wx.ALIGN_LEFT|wx.ALIGN_BOTTOM, 5 )
self.min_text_dp_5 = wx.StaticText( self.fine_disp_panel_5, wx.ID_ANY, u"Min", wx.DefaultPosition, wx.DefaultSize, wx.ALIGN_CENTRE )
self.min_text_dp_5.Wrap( -1 )
self.min_text_dp_5.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.min_text_dp_5.SetForegroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_SCROLLBAR ) )
param_dp_5.Add( self.min_text_dp_5, wx.GBPosition( 0, 0 ), wx.GBSpan( 1, 1 ),
wx.RIGHT|wx.TOP|wx.BOTTOM|wx.ALIGN_LEFT|wx.ALIGN_CENTER_VERTICAL, 5 )
self.minDisp_text_dp_5 = wx.StaticText( self.fine_disp_panel_5, wx.ID_ANY, u"0.01", wx.DefaultPosition, wx.Size( -1,-1 ), wx.ALIGN_LEFT
)
self.minDisp_text_dp_5.Wrap( -1 )
self.minDisp_text_dp_5.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.minDisp_text_dp_5.SetForegroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_SCROLLBAR ) )
self.minDisp_text_dp_5.SetMinSize( wx.Size( 100,-1 ) )

param_dp_5.Add( self.minDisp_text_dp_5, wx.GBPosition( 1, 0 ), wx.GBSpan( 1, 1 ),
wx.ALIGN_LEFT|wx.TOP|wx.BOTTOM|wx.ALIGN_CENTER_VERTICAL, 5 )
self.disp_spin_dp_5 = FS.FloatSpin( self.fine_disp_panel_5,-1, value = 1, min_val=.01, max_val=5,increment = .01,agwStyle=FS.FS_RIGHT )
self.disp_spin_dp_5.SetFormat("%f")
self.disp_spin_dp_5.SetDigits(2)
self.disp_spin_dp_5.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.disp_spin_dp_5.SetForegroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_BTNTEXT ) )
self.disp_spin_dp_5.SetBackgroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_BTNSHADOW ) )
param_dp_5.Add( self.disp_spin_dp_5, wx.GBPosition( 1, 1 ), wx.GBSpan( 1, 1 ),
wx.BOTTOM|wx.RIGHT|wx.LEFT|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.minVel_text_dp_5 = wx.StaticText( self.fine_disp_panel_5, wx.ID_ANY, u"1.0", wx.DefaultPosition, wx.Size( -1,-1 ),
wx.ALIGN_CENTRE )
self.minVel_text_dp_5.Wrap( -1 )
self.minVel_text_dp_5.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.minVel_text_dp_5.SetForegroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_SCROLLBAR ) )
param_dp_5.Add( self.minVel_text_dp_5, wx.GBPosition( 3, 0 ), wx.GBSpan( 1, 1 ),
wx.ALIGN_LEFT|wx.TOP|wx.BOTTOM|wx.RIGHT|wx.ALIGN_CENTER_VERTICAL, 5 )
self.max_dispText_dp_5 = wx.StaticText( self.fine_disp_panel_5, wx.ID_ANY, u"5", wx.DefaultPosition, wx.DefaultSize, wx.ALIGN_RIGHT )
self.max_dispText_dp_5.Wrap( -1 )
self.max_dispText_dp_5.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.max_dispText_dp_5.SetForegroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_SCROLLBAR ) )
self.max_dispText_dp_5.SetMinSize( wx.Size( 100,-1 ) )

param_dp_5.Add( self.max_dispText_dp_5, wx.GBPosition( 1, 2 ), wx.GBSpan( 1, 1 ),
wx.TOP|wx.BOTTOM|wx.ALIGN_RIGHT|wx.LEFT|wx.ALIGN_CENTER_VERTICAL, 5 )
self.vel_text_dp_5 = wx.StaticText( self.fine_disp_panel_5, wx.ID_ANY, u"velocity (mm/s)", wx.DefaultPosition, wx.DefaultSize,
wx.ALIGN_CENTRE )
self.vel_text_dp_5.Wrap( -1 )
self.vel_text_dp_5.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
param_dp_5.Add( self.vel_text_dp_5, wx.GBPosition( 2, 1 ), wx.GBSpan( 1, 1 ), wx.ALIGN_LEFT|wx.LEFT|wx.RIGHT|wx.TOP, 5 )
self.vel_spin_dp_5 = FS.FloatSpin( self.fine_disp_panel_5,-1, min_val=1, max_val=40.0,increment = .01,agwStyle=FS.FS_RIGHT )
self.vel_spin_dp_5.SetFormat("%f")
self.vel_spin_dp_5.SetDigits(2)
self.vel_spin_dp_5.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
```

```
self.vel_spin_dp_5.SetForegroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_BTNTEXT ) )
self.vel_spin_dp_5.SetBackgroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_BTNSHADOW ) )
param_dp_5.Add( self.vel_spin_dp_5, wx.GBPosition( 3, 1 ), wx.GBSpan( 1, 1 ),
wx.BOTTOM|wx.RIGHT|wx.LEFT|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.maxVel_text_dp_5 = wx.StaticText( self.fine_disp_panel_5, wx.ID_ANY, u"40", wx.DefaultPosition, wx.DefaultSize, 0 )
self.maxVel_text_dp_5.Wrap( -1 )
self.maxVel_text_dp_5.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.maxVel_text_dp_5.SetForegroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_SCROLLBAR ) )
param_dp_5.Add( self.maxVel_text_dp_5, wx.GBPosition( 3, 2 ), wx.GBSpan( 1, 1 ),
wx.TOP|wx.BOTTOM|wx.LEFT|wx.ALIGN_RIGHT|wx.ALIGN_CENTER_VERTICAL, 5 )
self.acc_text_dp_5 = wx.StaticText( self.fine_disp_panel_5, wx.ID_ANY, u"accel. (mm/s/s)", wx.DefaultPosition, wx.DefaultSize,
wx.ALIGN_CENTRE )
self.acc_text_dp_5.Wrap( -1 )
self.acc_text_dp_5.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
param_dp_5.Add( self.acc_text_dp_5, wx.GBPosition( 4, 1 ), wx.GBSpan( 1, 1 ), wx.TOP|wx.RIGHT|wx.LEFT|wx.ALIGN_LEFT, 5 )
self.min_Acc_text_dp_5 = wx.StaticText( self.fine_disp_panel_5, wx.ID_ANY, u"1.0", wx.DefaultPosition, wx.DefaultSize, 0 )
self.min_Acc_text_dp_5.Wrap( -1 )
self.min_Acc_text_dp_5.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.min_Acc_text_dp_5.SetForegroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_SCROLLBAR ) )
param_dp_5.Add( self.min_Acc_text_dp_5, wx.GBPosition( 5, 0 ), wx.GBSpan( 1, 1 ),
wx.ALIGN_LEFT|wx.TOP|wx.BOTTOM|wx.RIGHT|wx.ALIGN_CENTER_VERTICAL, 5 )
self.acc_spin_dp_5 = FS.FloatSpin( self.fine_disp_panel_5, -1,min_val=1, max_val=160.0,increment = 1, agwStyle=FS.FS_RIGHT )
self.acc_spin_dp_5.SetFormat("%f")
self.acc_spin_dp_5.SetDigits(2)
self.acc_spin_dp_5.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.acc_spin_dp_5.SetForegroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_BTNTEXT ) )
self.acc_spin_dp_5.SetBackgroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_BTNSHADOW ) )
param_dp_5.Add( self.acc_spin_dp_5, wx.GBPosition( 5, 1 ), wx.GBSpan( 1, 1 ),
wx.BOTTOM|wx.RIGHT|wx.LEFT|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.max_Acc_text_dp_5 = wx.StaticText( self.fine_disp_panel_5, wx.ID_ANY, u"160", wx.DefaultPosition, wx.DefaultSize, 0 )
self.max_Acc_text_dp_5.Wrap( -1 )
self.max_Acc_text_dp_5.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.max_Acc_text_dp_5.SetForegroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_SCROLLBAR ) )
param_dp_5.Add( self.max_Acc_text_dp_5, wx.GBPosition( 5, 2 ), wx.GBSpan( 1, 1 ),
wx.LEFT|wx.TOP|wx.BOTTOM|wx.ALIGN_RIGHT|wx.ALIGN_CENTER_VERTICAL, 5 )
self.hold_text_dp_5 = wx.StaticText( self.fine_disp_panel_5, wx.ID_ANY, u"hold (s) ", wx.DefaultPosition, wx.DefaultSize,
wx.ALIGN_CENTRE )
self.hold_text_dp_5.Wrap( -1 )
self.hold_text_dp_5.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
param_dp_5.Add( self.hold_text_dp_5, wx.GBPosition( 6, 1 ), wx.GBSpan( 1, 1 ),
wx.TOP|wx.RIGHT|wx.LEFT|wx.ALIGN_LEFT|wx.ALIGN_CENTER_VERTICAL, 5 )
self.min_holdText_dp_5 = wx.StaticText( self.fine_disp_panel_5, wx.ID_ANY, u"1", wx.DefaultPosition, wx.DefaultSize,
wx.ALIGN_CENTRE )
self.min_holdText_dp_5.Wrap( -1 )
self.min_holdText_dp_5.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.min_holdText_dp_5.SetForegroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_SCROLLBAR ) )
param_dp_5.Add( self.min_holdText_dp_5, wx.GBPosition( 7, 0 ), wx.GBSpan( 1, 1
),wx.ALIGN_LEFT|wx.TOP|wx.BOTTOM|wx.RIGHT|wx.ALIGN_CENTER_VERTICAL, 5 )
self.hold_spin_dp_5 = FS.FloatSpin( self.fine_disp_panel_5, -1,min_val=1, max_val=600,increment = .5, agwStyle=FS.FS_RIGHT )
self.hold_spin_dp_5.SetFormat("%f")
self.hold_spin_dp_5.SetDigits(2)
self.hold_spin_dp_5.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.hold_spin_dp_5.SetForegroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_BTNTEXT ) )
self.hold_spin_dp_5.SetBackgroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_BTNSHADOW ) )
param_dp_5.Add( self.hold_spin_dp_5, wx.GBPosition( 7, 1 ), wx.GBSpan( 1, 1 ),
wx.BOTTOM|wx.RIGHT|wx.LEFT|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.maxHold_text_dp_5 = wx.StaticText( self.fine_disp_panel_5, wx.ID_ANY, u"600", wx.DefaultPosition, wx.DefaultSize, 0 )
self.maxHold_text_dp_5.Wrap( -1 )
self.maxHold_text_dp_5.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.maxHold_text_dp_5.SetForegroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_SCROLLBAR ) )
param_dp_5.Add( self.maxHold_text_dp_5, wx.GBPosition( 7, 2 ), wx.GBSpan( 1, 1 ),
wx.LEFT|wx.TOP|wx.BOTTOM|wx.ALIGN_RIGHT|wx.ALIGN_CENTER_VERTICAL, 5 )
self.CRE_panel_5 = wx.Panel( self.fine_disp_panel_5, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.TAB_TRAVERSAL )
self.CRE_panel_5.SetBackgroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_HIGHLIGHTTEXT ) )
CRE_sizer_5 = wx.GridSizer( 0, 3, 0, 0 )
self.cycle_dp_5 = wx.RadioButton( self.CRE_panel_5, wx.ID_ANY, u"Cycle", wx.DefaultPosition, wx.DefaultSize, 0 )
CRE_sizer_5.Add( self.cycle_dp_5, 0, wx.ALL|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.cycle_dp_5.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.cycle_dp_5.SetValue(True)
self.retract_dp_5 = wx.RadioButton( self.CRE_panel_5, wx.ID_ANY, u"Retract", wx.DefaultPosition, wx.DefaultSize, 0 )
```

```
CRE_sizer_5.Add( self.retract_dp_5, 0, wx.ALL|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.retract_dp_5.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )

self.extend_dp_5 = wx.RadioButton( self.CRE_panel_5, wx.ID_ANY, u"Extend", wx.DefaultPosition, wx.DefaultSize, 0 )
CRE_sizer_5.Add( self.extend_dp_5, 0, wx.ALL|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.extend_dp_5.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )

self.CRE_panel_5.SetSizer( CRE_sizer_5 )
self.CRE_panel_5.Layout()
CRE_sizer_5.Fit( self.CRE_panel_5 )
param_dp_5.Add( self.CRE_panel_5, wx.GBPosition( 8, 0 ), wx.GBSpan( 1, 3 ), wx.EXPAND |wx.ALL, 5 )
fine_disp_sizer_5.Add( param_dp_5, 1, wx.ALIGN_CENTER_HORIZONTAL, 5 )
button_sizer_dp_5 = wx.BoxSizer( wx.VERTICAL )
self.indent_Button_dp_5 = wx.Button( self.fine_disp_panel_5, 29, u"Indent", wx.DefaultPosition, wx.Size( 175,60 ), 0 )
self.indent_Button_dp_5.SetFont( wx.Font( 18, 74, 90, 92, False, "Calibri" ) )
self.Bind(wx.EVT_BUTTON,self.displacementIndentExp,id=29)

button_sizer_dp_5.Add( self.indent_Button_dp_5, 0, wx.ALIGN_CENTER_HORIZONTAL|wx.ALL, 5 )
fine_disp_sizer_5.Add( button_sizer_dp_5, 1, wx.EXPAND, 5 )
self.m_panel77 = wx.Panel( self.fine_disp_panel_5, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize,wx.SIMPLE_BORDER )
gbSizer15 = wx.GridBagSizer( 0, 0 )
gbSizer15.SetFlexibleDirection( wx.BOTH )
gbSizer15.SetNonFlexibleGrowMode( wx.FLEX_GROWMODE_SPECIFIED )


self.prior_disp_dp_5 = wx.StaticText( self.m_panel77, wx.ID_ANY, u"Prior displacements", wx.DefaultPosition, wx.DefaultSize, 0 )
self.prior_disp_dp_5.Wrap( -1 )
self.prior_disp_dp_5.SetFont( wx.Font( 14, 74, 90, 92, False, "Calibri" ) )
gbSizer15.Add( self.prior_disp_dp_5, wx.GBPosition( 0, 0 ), wx.GBSpan( 1, 1 ), wx.ALL, 5 )
self.window_panel_dp_5 = wx.ScrolledWindow( self.m_panel77, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize,
wx.HSCROLL|wx.SIMPLE_BORDER|wx.VSCROLL )
self.window_panel_dp_5.SetScrollRate( 5, 5 )
self.window_panel_dp_5.SetMinSize( wx.Size( 320,350 ) )
window_sizer_dp_5 = wx.FlexGridSizer( 0, 7, 0, 0 )
window_sizer_dp_5.SetFlexibleDirection( wx.BOTH )
window_sizer_dp_5.SetNonFlexibleGrowMode( wx.FLEX_GROWMODE_SPECIFIED )

self.Disp_panel_dp_5 = wx.Panel( self.window_panel_dp_5, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.TAB_TRAVERSAL )
self.Disp_panel_dp_5.SetBackgroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_SCROLLBAR ) )
self.disp_sizer_dp_5 = wx.BoxSizer( wx.VERTICAL )
self.disp_sizer_dp_5.SetMinSize( wx.Size( -1,350 ) )
self.disp_text_dp_5 = wx.StaticText( self.Disp_panel_dp_5, wx.ID_ANY, u"Displacement (mm)", wx.DefaultPosition, wx.Size( 100,30 ),
wx.ALIGN_CENTRE )
self.disp_text_dp_5.Wrap( -1 )
self.disp_text_dp_5.SetBackgroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_SCROLLBAR ) )
self.disp_sizer_dp_5.Add( self.disp_text_dp_5, 0, wx.TOP|wx.RIGHT|wx.LEFT, 5 )
self.disp_line_dp_5 = wx.StaticLine( self.Disp_panel_dp_5, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.LI_HORIZONTAL )
self.disp_sizer_dp_5.Add( self.disp_line_dp_5, 0, wx.EXPAND|wx.TOP|wx.BOTTOM, 5 )
self.Disp_panel_dp_5.SetSizer( self.disp_sizer_dp_5 )
self.Disp_panel_dp_5.Layout()
self.disp_sizer_dp_5.Fit( self.Disp_panel_dp_5 )
window_sizer_dp_5.Add( self.Disp_panel_dp_5, 1, wx.EXPAND|wx.BOTTOM, 5 )
self.velcity_panel_dp_5 = wx.Panel( self.window_panel_dp_5, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.TAB_TRAVERSAL )
self.velcity_panel_dp_5.SetBackgroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_3DLIGHT ) )
self.velcity_panel_dp_5.SetMinSize( wx.Size( 100,350 ) )
self.velocity_sizer_dp_5 = wx.BoxSizer( wx.VERTICAL )
self.velocity_sizer_dp_5.SetMinSize( wx.Size( 100,350 ) )
self.vel_text_dp_5 = wx.StaticText( self.velcity_panel_dp_5, wx.ID_ANY, u"Velocity\n(mm/s)", wx.DefaultPosition, wx.Size( -1,30 ),
wx.ALIGN_CENTRE )
self.vel_text_dp_5.Wrap( -1 )
self.velocity_sizer_dp_5.Add( self.vel_text_dp_5, 0, wx.ALIGN_CENTER_HORIZONTAL|wx.TOP|wx.RIGHT|wx.LEFT, 5 )
self.vel_line_dp_5 = wx.StaticLine( self.velcity_panel_dp_5, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.LI_HORIZONTAL )
self.velocity_sizer_dp_5.Add( self.vel_line_dp_5, 0, wx.EXPAND|wx.TOP|wx.BOTTOM, 5 )
self.velcity_panel_dp_5.SetSizer( self.velocity_sizer_dp_5 )
self.velcity_panel_dp_5.Layout()
self.velocity_sizer_dp_5.Fit( self.velcity_panel_dp_5 )
window_sizer_dp_5.Add( self.velcity_panel_dp_5, 1, wx.EXPAND|wx.BOTTOM, 5 )
self.acceleration_panel_dp_5 = wx.Panel( self.window_panel_dp_5, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.TAB_TRAVERSAL
)
self.acceleration_panel_dp_5.SetBackgroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_SCROLLBAR ) )
```

```
self.acceleration_panel_dp_5.SetMinSize( wx.Size( -1,350 ) )
self.acc_sizer_dp_5 = wx.BoxSizer( wx.VERTICAL )
self.acc_sizer_dp_5.SetMinSize( wx.Size( 100,350 ) )
self.acc_text_dp_5 = wx.StaticText( self.acceleration_panel_dp_5, wx.ID_ANY, u"Acceleration\n(mm/s/s)", wx.DefaultPosition, wx.Size( -1,30
), wx.ALIGN_CENTRE )
self.acc_text_dp_5.Wrap( -1 )
self.acc_sizer_dp_5.Add( self.acc_text_dp_5, 0, wx.ALIGN_CENTER_HORIZONTAL|wx.TOP|wx.RIGHT|wx.LEFT, 5 )
self.acc_line_dp_5 = wx.StaticLine( self.acceleration_panel_dp_5, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.LI_HORIZONTAL )
self.acc_sizer_dp_5.Add( self.acc_line_dp_5, 0, wx.EXPAND|wx.TOP|wx.BOTTOM, 5 )
self.acceleration_panel_dp_5.SetSizer( self.acc_sizer_dp_5 )
self.acceleration_panel_dp_5.Layout()
self.acc_sizer_dp_5.Fit( self.acceleration_panel_dp_5 )
window_sizer_dp_5.Add( self.acceleration_panel_dp_5, 1, wx.EXPAND, 5 )

self.window_panel_dp_5.SetSizer( window_sizer_dp_5 )
self.window_panel_dp_5.Layout()
window_sizer_dp_5.Fit( self.window_panel_dp_5 )
gbSizer15.Add( self.window_panel_dp_5, wx.GBPosition( 1, 1 ), wx.GBSpan( 1, 1 ), wx.EXPAND, 5 )

self.m_panel77.SetSizer( gbSizer15 )
self.m_panel77.Layout()
gbSizer15.Fit( self.m_panel77 )
fine_disp_sizer_5.Add( self.m_panel77, 1, wx.EXPAND, 5 )
self.fine_disp_panel_5.SetSizer( fine_disp_sizer_5 )
self.fine_disp_panel_5.Layout()
fine_disp_sizer_5.Fit( self.fine_disp_panel_5 )
self.step5_disp_panel.AddPage( self.fine_disp_panel_5, u"Fine Displacement", True )
self.indentPlot_panel_5 = wx.Panel( self.step5_disp_panel, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.TAB_TRAVERSAL )
indentplot_sizer_5 = wx.FlexGridSizer( 0, 1, 0, 0 )
indentplot_sizer_5.SetFlexibleDirection( wx.BOTH )
indentplot_sizer_5.SetNonFlexibleGrowMode( wx.FLEX_GROWMODE_SPECIFIED )

self.select_plots_text_5 = wx.StaticText( self.indentPlot_panel_5, wx.ID_ANY, u"Select plots to view", wx.DefaultPosition, wx.Size( 660,-1 ), 0
)
self.select_plots_text_5.Wrap( -1 )
self.select_plots_text_5.SetFont( wx.Font( 14, 74, 90, 92, False, "Calibri" ) )
indentplot_sizer_5.Add( self.select_plots_text_5, 0, wx.TOP|wx.BOTTOM|wx.RIGHT, 5 )
self.fine_disp_text_5 = wx.StaticText( self.indentPlot_panel_5, wx.ID_ANY, u"Fine displacement plots", wx.DefaultPosition, wx.DefaultSize, 0
)
self.fine_disp_text_5.Wrap( -1 )
self.fine_disp_text_5.SetFont( wx.Font( 14, 74, 90, 92, False, "Calibri" ) )
indentplot_sizer_5.Add( self.fine_disp_text_5, 0, wx.RIGHT, 5 )
self.window_panel_fd_5 = wx.ScrolledWindow( self.indentPlot_panel_5, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize,
wx.HSCROLL|wx.SIMPLE_BORDER|wx.VSCROLL )
self.window_panel_fd_5.SetScrollRate( 5, 5 )
window_sizer_fd_5 = wx.FlexGridSizer( 0, 7, 0, 0 )
window_sizer_fd_5.SetFlexibleDirection( wx.BOTH )
window_sizer_fd_5.SetNonFlexibleGrowMode( wx.FLEX_GROWMODE_SPECIFIED )
self.window_panel_fd_5.SetMinSize( wx.Size( 400,300 ) )

self.view_panel_fd_5 = wx.Panel( self.window_panel_fd_5, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.TAB_TRAVERSAL )
self.view_panel_fd_5.SetBackgroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_SCROLLBAR ) )
self.view_panel_fd_5.SetMinSize( wx.Size( 65,300 ) )
self.view_sizer_fd_5 = wx.BoxSizer( wx.VERTICAL )
self.view_sizer_fd_5.SetMinSize( wx.Size(65,-1 ) )
self.view_text_fd_5 = wx.StaticText( self.view_panel_fd_5, wx.ID_ANY, u"View ", wx.DefaultPosition, wx.Size( 65,30 ),
wx.ALIGN_CENTRE )
self.view_text_fd_5.Wrap( -1 )
self.view_text_fd_5.SetBackgroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_SCROLLBAR ) )
self.view_sizer_fd_5.Add( self.view_text_fd_5, 0, wx.TOP|wx.RIGHT|wx.LEFT, 5 )

self.view_line_fd_5 = wx.StaticLine( self.view_panel_fd_5, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.LI_HORIZONTAL )
self.view_sizer_fd_5.Add( self.view_line_fd_5, 0, wx.EXPAND|wx.TOP|wx.BOTTOM, 5 )
self.view_panel_fd_5.SetSizer( self.view_sizer_fd_5 )
self.view_panel_fd_5.Layout()
self.view_sizer_fd_5.Fit( self.view_panel_fd_5 )
window_sizer_fd_5.Add( self.view_panel_fd_5, 1, wx.ALIGN_CENTER_HORIZONTAL|wx.EXPAND, 5 )
self.disp_panel_fd_5 = wx.Panel( self.window_panel_fd_5, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.TAB_TRAVERSAL )
self.disp_panel_fd_5.SetBackgroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_3DLIGHT ) )
self.disp_panel_fd_5.SetMinSize( wx.Size( 100,300 ) )
```

```
self.disp_sizer_fd_5 = wx.BoxSizer( wx.VERTICAL )
self.disp_sizer_fd_5.SetMinSize( wx.Size( 100,300 ) )
self.disp_text_fd_5 = wx.StaticText( self.disp_panel_fd_5, wx.ID_ANY, u"Displacement \n (mm)", wx.DefaultPosition, wx.Size( -1,30 ),
wx.ALIGN_CENTRE )
self.disp_text_fd_5.Wrap( -1 )
self.disp_sizer_fd_5.Add( self.disp_text_fd_5, 0, wx.ALIGN_CENTER_HORIZONTAL|wx.TOP|wx.RIGHT|wx.LEFT, 5 )
self.disp_line_fd_5 = wx.StaticLine( self.disp_panel_fd_5, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.LI_HORIZONTAL )
self.disp_sizer_fd_5.Add( self.disp_line_fd_5, 0, wx.EXPAND|wx.TOP|wx.BOTTOM, 5 )
self.disp_panel_fd_5.SetSizer( self.disp_sizer_fd_5 )
self.disp_panel_fd_5.Layout()
self.disp_sizer_fd_5.Fit( self.disp_panel_fd_5 )
window_sizer_fd_5.Add( self.disp_panel_fd_5, 1, wx.EXPAND, 5 )
self.vel_panel_fd_5 = wx.Panel( self.window_panel_fd_5, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.TAB_TRAVERSAL )
self.vel_panel_fd_5.SetBackgroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_SCROLLBAR ) )
self.vel_panel_fd_5.SetMinSize( wx.Size( -1,300 ) )
self.vel_sizer_fd_5 = wx.BoxSizer( wx.VERTICAL )
self.vel_sizer_fd_5.SetMinSize( wx.Size( 100,300 ) )
self.vel_text_fd_5 = wx.StaticText( self.vel_panel_fd_5, wx.ID_ANY, u"Velocity\n(mm/s)", wx.DefaultPosition, wx.Size( -1,30 ),
wx.ALIGN_CENTRE )
self.vel_text_fd_5.Wrap( -1 )
self.vel_sizer_fd_5.Add( self.vel_text_fd_5, 0, wx.ALIGN_CENTER_HORIZONTAL|wx.TOP|wx.RIGHT|wx.LEFT, 5 )
self.vel_line_fd_5 = wx.StaticLine( self.vel_panel_fd_5, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.LI_HORIZONTAL )
self.vel_sizer_fd_5.Add( self.vel_line_fd_5, 0, wx.EXPAND|wx.TOP|wx.BOTTOM, 5 )
self.vel_panel_fd_5.SetSizer( self.vel_sizer_fd_5 )
self.vel_panel_fd_5.Layout()
self.vel_sizer_fd_5.Fit( self.vel_panel_fd_5 )
window_sizer_fd_5.Add( self.vel_panel_fd_5, 1, wx.EXPAND, 5 )
self.acc_panel_fd_5 = wx.Panel( self.window_panel_fd_5, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.TAB_TRAVERSAL )
self.acc_panel_fd_5.SetBackgroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_3DLIGHT ) )
self.acc_panel_fd_5.SetMinSize( wx.Size( -1,300 ) )
self.acc_sizer_fd_5 = wx.BoxSizer( wx.VERTICAL )
self.acc_text_fd_5 = wx.StaticText( self.acc_panel_fd_5, wx.ID_ANY, u"Acceleration \n (mm/s/s)", wx.DefaultPosition, wx.Size( 100,30 ),
wx.ALIGN_CENTRE )
self.acc_text_fd_5.Wrap( -1 )
self.acc_text_fd_5.SetBackgroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_3DLIGHT ) )
self.acc_sizer_fd_5.Add( self.acc_text_fd_5, 0, wx.TOP|wx.RIGHT|wx.LEFT, 5 )
self.acc_line_fd_5 = wx.StaticLine( self.acc_panel_fd_5, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.LI_HORIZONTAL )
self.acc_sizer_fd_5.Add( self.acc_line_fd_5, 0, wx.EXPAND|wx.TOP|wx.BOTTOM, 5 )
self.acc_panel_fd_5.SetSizer( self.acc_sizer_fd_5 )
self.acc_panel_fd_5.Layout()
self.acc_sizer_fd_5.Fit( self.acc_panel_fd_5 )
window_sizer_fd_5.Add( self.acc_panel_fd_5, 1, wx.EXPAND, 5 )
self.window_panel_fd_5.SetSizer( window_sizer_fd_5 )
self.window_panel_fd_5.Layout()
window_sizer_fd_5.Fit( self.window_panel_fd_5 )
indentplot_sizer_5.Add( self.window_panel_fd_5, 0, wx.BOTTOM|wx.RIGHT|wx.LEFT|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.view_button_fd_5 = wx.Button( self.indentPlot_panel_5, 30, u"View", wx.DefaultPosition, wx.Size( 175,60 ), 0 )
self.view_button_fd_5.SetFont( wx.Font( 18, 74, 90, 92, False, "Calibri" ) )
self.Bind(wx.EVT_BUTTON,self.viewDispPlots,id=30)

indentplot_sizer_5.Add( self.view_button_fd_5, 0, wx.ALL|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.indentPlot_panel_5.SetSizer( indentplot_sizer_5 )
self.indentPlot_panel_5.Layout()
indentplot_sizer_5.Fit( self.indentPlot_panel_5 )
self.step5_disp_panel.AddPage( self.indentPlot_panel_5, u"Indentation Plots", False )
decisionSizer.Add( self.step5_disp_panel, 1, wx.EXPAND, 5 )
self.decisionPanel.SetSizer( decisionSizer )
self.decisionPanel.Layout()
decisionSizer.Fit( self.decisionPanel )
decision_LogSizer.Add( self.decisionPanel, 1, wx.EXPAND|wx.TOP|wx.BOTTOM, 5 )
self.logPanel = wx.Panel( self.decisionAndLogPanel, wx.ID_ANY, wx.DefaultPosition, wx.Size( 550,600 ), wx.TAB_TRAVERSAL )
self.logPanel.SetBackgroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_INFOBK ) )
logSizer = wx.FlexGridSizer( 5, 1, 0, 0 )
logSizer.SetFlexibleDirection( wx.BOTH )
logSizer.SetNonFlexibleGrowMode( wx.FLEX_GROWMODE_SPECIFIED )
self.emergencyButtons = wx.Panel( self.logPanel, wx.ID_ANY, wx.DefaultPosition, wx.Size( 366,80 ), wx.TAB_TRAVERSAL )
embuttonsSizer = wx.BoxSizer( wx.HORIZONTAL )
embuttonsSizer.SetMinSize( wx.Size( 366,80 ) )
self.emergencyStop = wx.Button( self.emergencyButtons, 31, u"Emergency \n Stop", wx.DefaultPosition, wx.Size( 175,70 ),
0|wx.RAISED_BORDER )
```

```
self.emergencyStop.SetFont( wx.Font( 18, 74, 90, 92, False, "Calibri" ) )
self.emergencyStop.SetForegroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_INACTIVEBORDER ) )
self.emergencyStop.SetBackgroundColour( wx.Colour( 247, 15, 2 ) )
#self.Bind(wx.EVT_BUTTON,self.viewDispPlots,id=31)

embuttonsSizer.Add( self.emergencyStop, 0, wx.ALIGN_CENTER_VERTICAL|wx.ALL, 5 )
self.homeSled = wx.Button( self.emergencyButtons, 32, u"Home \n Sled", wx.DefaultPosition, wx.Size( 80,70 ), 0 )
self.homeSled.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.Bind(wx.EVT_BUTTON,self.home,id=32)

embuttonsSizer.Add( self.homeSled, 0, wx.ALL|wx.ALIGN_CENTER_VERTICAL, 5 )
self.above_0 = wx.Button( self.emergencyButtons, 33, u"20 mm \n above P.O.C.", wx.DefaultPosition, wx.Size( 80,70 ), 0 )
self.above_0.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.Bind(wx.EVT_BUTTON,self.highestPoint,id=33)

embuttonsSizer.Add( self.above_0, 0, wx.ALL|wx.ALIGN_CENTER_VERTICAL, 5 )
self.emergencyButtons.SetSizer( embuttonsSizer )
self.emergencyButtons.Layout()
logSizer.Add( self.emergencyButtons, 1, wx.EXPAND|wx.ALL, 5 )
self.force_Slider = wx.Panel( self.logPanel, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.TAB_TRAVERSAL|wx.SIMPLE_BORDER
)
self.force_Slider.SetMinSize( wx.Size( 350,80 ) )
force_Slider_sizer = wx.BoxSizer(wx.VERTICAL)
force_Slider_sizer.SetMinSize( wx.Size( 350,80) )

wid = Widget(self.force_Slider,-1)
force_Slider_sizer.Add(wid,1,wx.EXPAND)
self.text_forceRead_label = wx.StaticText( self.force_Slider, wx.ID_ANY, "force (mN)", wx.DefaultPosition, wx.Size( -1,-1 ),
wx.ALIGN_CENTRE )
self.text_forceRead_label.Wrap( -1 )
self.text_forceRead_label.SetFont( wx.Font( 12, 74, 90, 92, False, "Calibri" ) )
force_Slider_sizer.Add(self.text_forceRead_label,1,wx.ALIGN_CENTER_HORIZONTAL)
set_Potent = wx.FlexGridSizer( 0, 2, 0, 0 )
set_Potent.SetFlexibleDirection( wx.BOTH )
set_Potent.SetNonFlexibleGrowMode( wx.FLEX_GROWMODE_SPECIFIED )
self.text_setlcRange = wx.StaticText( self.force_Slider, wx.ID_ANY, "Set loadcell range (mN): ", wx.DefaultPosition, wx.Size( -1,-1 ),
wx.ALIGN_CENTRE )
self.text_setlcRange.Wrap( -1 )
self.text_setlcRange.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
set_Potent.Add(self.text_setlcRange,1,wx.ALIGN_CENTER_HORIZONTAL)

potent_Val = wx.GridSizer( 0, 3, 0, 0 )
self.value_500 = wx.RadioButton( self.force_Slider, wx.ID_ANY, u"500", wx.DefaultPosition, wx.DefaultSize, 0 )
potent_Val.Add( self.value_500, 0, wx.ALL|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.value_500.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )

self.value_1000 = wx.RadioButton( self.force_Slider, wx.ID_ANY, u"1000", wx.DefaultPosition, wx.DefaultSize, 0 )
potent_Val.Add( self.value_1000, 0, wx.ALL|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.value_1000.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )

self.value_1500 = wx.RadioButton( self.force_Slider, wx.ID_ANY, u"1500", wx.DefaultPosition, wx.DefaultSize, 0 )
potent_Val.Add( self.value_1500, 0, wx.ALL|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.value_1500.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.Bind(wx.EVT_RADIOBUTTON,self.setPotentVal1,self.value_1500)
self.Bind(wx.EVT_RADIOBUTTON,self.setPotentVal2,self.value_1000)
self.Bind(wx.EVT_RADIOBUTTON,self.setPotentVal3,self.value_500)



set_Potent.Add(potent_Val,1,wx.ALIGN_CENTER_HORIZONTAL)

force_Slider_sizer.Add(set_Potent,1,wx.EXPAND)

self.force_Slider.SetSizer( force_Slider_sizer )
self.force_Slider.Layout()
logSizer.Add( self.force_Slider, 1, wx.EXPAND|wx.TOP|wx.BOTTOM, 5 )
self.status_panel = wx.Panel( self.logPanel, wx.ID_ANY, wx.DefaultPosition, wx.Size( 416,80 ), 0 )
self.sizer_panel = wx.GridBagSizer( 0, 0 )
self.sizer_panel.SetFlexibleDirection( wx.BOTH )
self.sizer_panel.SetNonFlexibleGrowMode( wx.FLEX_GROWMODE_ALL )
```

```
#sizer_panel.SetMinSize( wx.Size( 416,70 ) )
self.text_status = wx.StaticText( self.status_panel, wx.ID_ANY, "Status:", wx.DefaultPosition, wx.Size( 100,30 ), wx.ALIGN_CENTRE )
self.text_status.Wrap( -1 )
self.text_status.SetFont( wx.Font( 14, 74, 90, 92, False, "Calibri" ) )
self.text_status.SetBackgroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_WINDOW ) )
self.sizer_panel.Add( self.text_status, wx.GBPosition( 0, 0 ), wx.GBSpan( 1, 1 ), wx.TOP, 5 )
self.Status = wx.StaticText( self.status_panel, wx.ID_ANY, "Ready", wx.DefaultPosition, wx.Size( 70,30 ), wx.ALIGN_CENTRE )
self.Status.Wrap( -1 )
self.Status.SetFont( wx.Font( 16, 74, 90, 92, False, "Calibri" ) )
self.Status.SetForegroundColour( wx.Colour( 101, 218, 80 ) )
self.Status.SetBackgroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_WINDOW ) )
self.sizer_panel.Add( self.Status, wx.GBPosition( 0, 1 ), wx.GBSpan( 1, 1 ), wx.TOP, 5 )
self.text_for = wx.StaticText( self.status_panel, wx.ID_ANY, u" for ", wx.DefaultPosition, wx.Size( 40,30), wx.ALIGN_RIGHT )
self.text_for.Wrap( -1 )
self.text_for.SetFont( wx.Font( 14, 74, 90, 92, False, "Calibri") )
self.text_for.SetBackgroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_WINDOW ) )
self.sizer_panel.Add( self.text_for, wx.GBPosition( 0, 2 ), wx.GBSpan( 1, 1 ), wx.TOP, 5 )
self.time_status = wx.StaticText( self.status_panel, wx.ID_ANY, u" hh:mm:ss", wx.DefaultPosition, wx.Size( 200,30 ), wx.ALIGN_LEFT )
self.time_status.Wrap( -1 )
self.time_status.SetFont( wx.Font( 14, 74, 90, 90, False, "Calibri" ) )
self.time_status.SetBackgroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_3DLIGHT ) )
self.timer = wx.Timer(self)
self.global_log_start_time =datetime.datetime.now()
self.local_log_start_time = datetime.datetime.now()
self.Bind(wx.EVT_TIMER,self.update,self.timer)
self.sizer_panel.Add( self.time_status, wx.GBPosition( 0, 3 ), wx.GBSpan( 1, 1 ), wx.TOP, 5 )
self.text_logtime = wx.StaticText( self.status_panel, wx.ID_ANY, u"Log Time: ", wx.DefaultPosition, wx.DefaultSize, 0 )
self.text_logtime.Wrap( -1 )
self.text_logtime.SetFont( wx.Font( 14, 74, 90, 92, False, "Calibri" ) )
self.sizer_panel.Add( self.text_logtime, wx.GBPosition( 1, 1 ), wx.GBSpan( 1, 2 ), wx.ALIGN_RIGHT|wx.TOP|wx.BOTTOM, 5 )
self.time_log = wx.StaticText( self.status_panel, wx.ID_ANY, u" hh:mm:ss", wx.DefaultPosition, wx.Size( 150,-1 ), 0 )
self.time_log.Wrap( -1 )
self.time_log.SetFont( wx.Font( 14, 74, 90, 90, False, "Calibri" ) )
self.time_log.SetBackgroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_3DLIGHT ) )
self.sizer_panel.Add( self.time_log, wx.GBPosition( 1, 3 ), wx.GBSpan( 1, 1 ), wx.ALL, 5 )
self.status_panel.SetSizer( self.sizer_panel )
self.status_panel.Layout()
logSizer.Add( self.status_panel, 1, wx.EXPAND|wx.RIGHT, 5 )
self.scrolled_log_panel = wx.ScrolledWindow( self.logPanel, wx.ID_ANY, wx.DefaultPosition, wx.Size( 416,300 ),
wx.HSCROLL|wx.SIMPLE_BORDER|wx.VSCROLL )
self.scrolled_log_panel.SetScrollRate( 5, 5 )
logData_sizer = wx.FlexGridSizer( 0, 7, 0, 0 )
logData_sizer.SetFlexibleDirection( wx.BOTH )
logData_sizer.SetNonFlexibleGrowMode( wx.FLEX_GROWMODE_SPECIFIED )
logData_sizer.SetMinSize( wx.Size( 416,800 ) )
self.type_panel = wx.Panel( self.scrolled_log_panel, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.TAB_TRAVERSAL )
# self.type_panel.SetMinSize( wx.Size( -1, ) )
self.type_panel.SetBackgroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_WINDOW ) )

self.type_sizer = wx.BoxSizer( wx.VERTICAL )
self.type_sizer.SetMinSize( wx.Size( -1,800 ) )
self.type_header = wx.StaticText( self.type_panel, wx.ID_ANY, u"type", wx.DefaultPosition, wx.Size( 50,30 ), wx.ALIGN_CENTRE )
self.type_header.Wrap( -1 )
self.type_sizer.Add( self.type_header, 0, wx.ALL|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.type_staticline = wx.StaticLine( self.type_panel, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.LI_HORIZONTAL )
self.type_sizer.Add( self.type_staticline, 0, wx.EXPAND|wx.TOP|wx.BOTTOM, 5 )
self.type_panel.SetSizer( self.type_sizer )
self.type_panel.Layout()
self.type_sizer.Fit( self.type_panel )
logData_sizer.Add( self.type_panel, 1, wx.EXPAND, 5 )
self.time_panel = wx.Panel( self.scrolled_log_panel, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.TAB_TRAVERSAL )
self.time_panel.SetBackgroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_3DLIGHT ) )
self.time_panel.SetMinSize( wx.Size( -1,800 ) )
self.time_sizer = wx.BoxSizer( wx.VERTICAL )
self.time_sizer.SetMinSize( wx.Size( -1,800 ) )
self.time_header = wx.StaticText( self.time_panel, wx.ID_ANY, u"time\n(hh:mm:ss)", wx.DefaultPosition, wx.Size( -1,30 ),
wx.ALIGN_CENTRE )
self.time_header.Wrap( -1 )
self.time_sizer.Add( self.time_header, 0, wx.ALL|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.time_staticline = wx.StaticLine( self.time_panel, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.LI_HORIZONTAL )
```

```
self.time_sizer.Add( self.time_staticline, 0, wx.EXPAND|wx.TOP|wx.BOTTOM, 5 )
self.time_panel.SetSizer( self.time_sizer )
self.time_panel.Layout()
self.time_sizer.Fit( self.time_panel )
logData_sizer.Add( self.time_panel, 1, wx.EXPAND, 5 )
self.disp_panel = wx.Panel( self.scrolled_log_panel, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.TAB_TRAVERSAL )
self.disp_panel.SetBackgroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_WINDOW ) )
self.disp_panel.SetMinSize( wx.Size( -1,800 ) )
self.disp_sizer = wx.BoxSizer( wx.VERTICAL )
self.disp_sizer.SetMinSize( wx.Size( -1,800 ) )
self.disp_header = wx.StaticText( self.disp_panel, wx.ID_ANY, u"disp\n(mm)", wx.DefaultPosition, wx.Size( -1,30 ), wx.ALIGN_CENTRE )
self.disp_header.Wrap( -1 )
self.disp_sizer.Add( self.disp_header, 0, wx.ALL, 5 )
self.disp_staticline = wx.StaticLine( self.disp_panel, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.LI_HORIZONTAL )
self.disp_sizer.Add( self.disp_staticline, 0, wx.EXPAND|wx.TOP|wx.BOTTOM, 5 )
self.disp_panel.SetSizer( self.disp_sizer )
self.disp_panel.Layout()
self.disp_sizer.Fit( self.disp_panel )
logData_sizer.Add( self.disp_panel, 1, wx.EXPAND, 5 )
self.force_panel = wx.Panel( self.scrolled_log_panel, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.TAB_TRAVERSAL )
self.force_panel.SetBackgroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_3DLIGHT ) )
self.force_panel.SetMinSize( wx.Size( -1,800 ) )
self.force_sizer = wx.BoxSizer( wx.VERTICAL )
self.force_sizer.SetMinSize( wx.Size( -1,800 ) )
self.force_header = wx.StaticText( self.force_panel, wx.ID_ANY, u"force\n(mN)", wx.DefaultPosition, wx.Size( 40,30 ), wx.ALIGN_CENTRE )
self.force_header.Wrap( -1 )
self.force_sizer.Add( self.force_header, 0, wx.ALL|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.force_staticline = wx.StaticLine( self.force_panel, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.LI_HORIZONTAL )
self.force_sizer.Add( self.force_staticline, 0, wx.EXPAND|wx.TOP|wx.BOTTOM, 5 )
self.force_panel.SetSizer( self.force_sizer )
self.force_panel.Layout()
self.force_sizer.Fit( self.force_panel )
logData_sizer.Add( self.force_panel, 1, wx.EXPAND, 5 )
self.vel_panel = wx.Panel( self.scrolled_log_panel, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.TAB_TRAVERSAL )
self.vel_panel.SetMinSize( wx.Size( -1,800 ) )
self.vel_panel.SetBackgroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_WINDOW ) )
self.vel_sizer = wx.BoxSizer( wx.VERTICAL )
self.vel_sizer.SetMinSize( wx.Size( -1,800 ) )
self.vel_header = wx.StaticText( self.vel_panel, wx.ID_ANY, u"vel\n(mm/s)", wx.DefaultPosition, wx.Size( 50,30 ), wx.ALIGN_CENTRE )
self.vel_header.Wrap( -1 )

self.vel_sizer.Add( self.vel_header, 0, wx.ALL, 5 )
self.vel_staticline = wx.StaticLine( self.vel_panel, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.LI_HORIZONTAL )
self.vel_sizer.Add( self.vel_staticline, 0, wx.EXPAND|wx.TOP|wx.BOTTOM, 5 )
self.vel_panel.SetSizer( self.vel_sizer )
self.vel_panel.Layout()
self.vel_sizer.Fit( self.vel_panel )
logData_sizer.Add( self.vel_panel, 1, wx.EXPAND, 5 )
self.acc_panel = wx.Panel( self.scrolled_log_panel, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.TAB_TRAVERSAL )
self.acc_panel.SetBackgroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_3DLIGHT ) )
self.acc_panel.SetMinSize( wx.Size( -1,800 ) )
self.acc_sizer = wx.BoxSizer( wx.VERTICAL )
self.acc_sizer.SetMinSize( wx.Size( -1,800 ) )
self.acc_header = wx.StaticText( self.acc_panel, wx.ID_ANY, u"acc\n(mm/s/s)", wx.DefaultPosition, wx.Size( 50,30 ), wx.ALIGN_CENTRE )
self.acc_header.Wrap( -1 )
self.acc_sizer.Add( self.acc_header, 0, wx.ALL|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.acc_staticline = wx.StaticLine( self.acc_panel, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.LI_HORIZONTAL )
self.acc_sizer.Add( self.acc_staticline, 0, wx.EXPAND|wx.TOP|wx.BOTTOM, 5 )
self.acc_panel.SetSizer( self.acc_sizer )
self.acc_panel.Layout()
self.acc_sizer.Fit( self.acc_panel )
logData_sizer.Add( self.acc_panel, 1, wx.EXPAND, 5 )
self.hold_panel = wx.Panel( self.scrolled_log_panel, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.TAB_TRAVERSAL )
self.hold_panel.SetMinSize( wx.Size( 45,800 ) )
self.hold_panel.SetBackgroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_WINDOW ) )

self.hold_sizer = wx.BoxSizer( wx.VERTICAL )
self.hold_sizer.SetMinSize( wx.Size( 45,800 ) )
self.hold_header = wx.StaticText( self.hold_panel, wx.ID_ANY, u"hold\n(s)", wx.DefaultPosition, wx.Size( 40,30 ), wx.ALIGN_CENTRE )
self.hold_header.Wrap( -1 )
```

```
self.hold_sizer.Add( self.hold_header, 0, wx.TOP|wx.BOTTOM|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.hold_staticline = wx.StaticLine( self.hold_panel, wx.ID_ANY, wx.DefaultPosition, wx.DefaultSize, wx.LI_HORIZONTAL )
self.hold_sizer.Add( self.hold_staticline, 0, wx.EXPAND|wx.TOP|wx.BOTTOM, 5 )
self.hold_panel.SetSizer( self.hold_sizer )
self.hold_panel.Layout()
self.hold_sizer.Fit( self.hold_panel )
logData_sizer.Add( self.hold_panel, 1, wx.EXPAND, 5 )
self.scrolled_log_panel.SetSizer( logData_sizer )
self.scrolled_log_panel.Layout()
logSizer.Add( self.scrolled_log_panel, 1, wx.EXPAND, 5 )
self.mouse_exp_ID = wx.Panel( self.logPanel, wx.ID_ANY, wx.DefaultPosition, wx.Size( 416,100 ), wx.TAB_TRAVERSAL )
mouse_exp_ID_sizer = wx.GridSizer( 2, 2, 0, 0 )
self.text_MouseId = wx.StaticText( self.mouse_exp_ID, wx.ID_ANY, u"Mouse ID: ", wx.DefaultPosition, wx.DefaultSize, 0 )
self.text_MouseId.Wrap( -1 )
self.text_MouseId.SetFont( wx.Font( 12, 74, 90, 92, False, "Calibri" ) )
mouse_exp_ID_sizer.Add( self.text_MouseId, 0, wx.ALL|wx.ALIGN_RIGHT, 5 )
self.mouse_ID = wx.TextCtrl( self.mouse_exp_ID, 10000, u"ID", wx.DefaultPosition, wx.DefaultSize,
wx.TE_CENTRE|wx.TE_PROCESS_ENTER )
self.mouse_ID.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.Bind(wx.EVT_TEXT_ENTER, self.UpdateLogFile_mouse,id=10000)

mouse_exp_ID_sizer.Add( self.mouse_ID, 0, wx.BOTTOM|wx.RIGHT|wx.LEFT, 5 )
self.text_ExperimentalID = wx.StaticText( self.mouse_exp_ID, wx.ID_ANY, u"Experimental ID: ", wx.DefaultPosition, wx.DefaultSize,
wx.TE_CENTRE )
self.text_ExperimentalID.SetFont( wx.Font( 12, 74, 90, 92, False, "Calibri" ) )

mouse_exp_ID_sizer.Add( self.text_ExperimentalID, 0, wx.ALL|wx.ALIGN_RIGHT, 5 )
self.experimental_ID = wx.TextCtrl( self.mouse_exp_ID, 9999, u"ID", wx.DefaultPosition, wx.DefaultSize,
wx.TE_CENTRE|wx.TE_PROCESS_ENTER )
self.experimental_ID.SetFont( wx.Font( 12, 74, 90, 90, False, "Calibri" ) )
self.Bind(wx.EVT_TEXT_ENTER, self.UpdateLogFile_Exp,id=9999)

mouse_exp_ID_sizer.Add( self.experimental_ID, 0, wx.BOTTOM|wx.RIGHT|wx.LEFT, 5 )
self.mouse_exp_ID.SetSizer( mouse_exp_ID_sizer )
self.mouse_exp_ID.Layout()
logSizer.Add( self.mouse_exp_ID, 1, wx.EXPAND |wx.ALL, 5 )
#
self.logPanel.SetSizer( logSizer )
self.logPanel.Layout()
decision_LogSizer.Add( self.logPanel, 1, wx.EXPAND|wx.ALL, 5 )
self.decisionAndLogPanel.SetSizer( decision_LogSizer )
self.decisionAndLogPanel.Layout()
topandBottomSizer.Add( self.decisionAndLogPanel, 1, wx.EXPAND|wx.RIGHT|wx.LEFT, 5 )
self.UI_Panel.SetSizer( topandBottomSizer )
self.UI_Panel.Layout()
topandBottomSizer.Fit( self.UI_Panel )
UISizer.Add( self.UI_Panel, 0, wx.EXPAND, 5 )
self.SetSizer( UISizer )
self.Layout()
self.Centre( wx.BOTH )

##################################################### Button Commands ##########################################
save_log_dialogFileName = "VirginiaIndenterLog_"+datetime.datetime.now().strftime("%Y-%m-%d")+ "_" +
datetime.datetime.now().strftime("%H_%M_%S")
self.timer.Start(1000)
potent_Mode = loadCell()
pub.subscribe(self.OnNewCW, "cw")
pub.subscribe(self.OnNewDisp, "Disp_Val")
log_started_string = "{Log started at " +datetime.datetime.now().strftime("%H:%M:%S") + " on "+datetime.datetime.now().strftime("%Y-%m-
%d")+ "}\n"
Mouseid_string = "Mouse ID:___" + "\n" + "Experimental ID:_____"
FILE = open(fileLoc+'\\'+ save_log_dialogFileName+'.txt',"a")
FILE.write(log_started_string+"\n"+Mouseid_string+"\n")

self.Show()
def replace(self,file_path, pattern, subst):
#Create temp file
fh, abs_path = mkstemp()
new_file = open(abs_path,'w')
old_file = open(file_path)
```

```
for line in old_file:
new_file.write(line.replace(pattern, subst))
#close temp file
new_file.close()
close(fh)
old_file.close()
#Remove original file
remove(file_path)
#Move new file
move(abs_path, file_path)
def UpdateLogFile_mouse(self,event):
global fileLoc
global save_log_dialogFileName
mouseId = self.mouse_ID.GetValue()
self.replace(fileLoc+'\\'+ save_log_dialogFileName+'.txt',"Mouse ID:___","Mouse ID: " +mouseId )
def UpdateLogFile_Exp(self,event):
global fileLoc
global save_log_dialogFileName
experimentalId = self.experimental_ID.GetValue()
self.replace(fileLoc+'\\'+ save_log_dialogFileName+'.txt',"Experimental ID:_____","Experimental ID: " +experimentalId )
def OnNewCW(self,cw_update):
global cw
cw = cw_update
wid.Refresh()
def OnNewDisp(self,disp_update):
global cw
global setMin
global setMax
global displayMinDisp_box_2
global displayMinForce_box_2
global dispMaxDisp_box_2
global dispMaxForce_box_2
cw = disp_update[0]
displacement = disp_update[1]
if (not(setMin) and not(setMax) ):
displayMinDisp_box_2.SetValue((str('{0:.2f}'.format(displacement))))
displayMinForce_box_2.SetValue(str('{0:.2f}'.format((cw))))
elif(setMin and not(setMax) ):
dispMaxDisp_box_2.SetValue(str('{0:.2f}'.format(displacement)))
dispMaxForce_box_2.SetValue(str('{0:.2f}'.format(cw)))
wid.Refresh()
def changePagesRight(self,event):
current_Step = self.upperStep_text.GetLabel()[5]
if(current_Step == '1'):
global dispMode
global Disp_Mode
global setMin
global setMax
global Alone
global Disp_Mode
Alone = False
self.upperStep_text.SetLabel("Step 2: Find Minimum and \n Maximum Displacement")
self.Step2_Right.Hide()
self.Step2_Right.Disable()
self.NoButton_Left.Hide()
self.NoButton_Left.Disable()
self.Step1_Left.Show()
self.Step1_Left.Enable()
self.Step3_Right.Show()
self.Step3_Right.Enable(True)
self.step1_panel.Hide()
self.step1_panel.Disable()
self.step2_panel.Show()
self.step2_panel.Enable(True)
dispMode = True
setMin = False
setMax= False
Disp_Mode = DispThread()
self.headerPanel.Layout()
self.Layout()
```

```
elif(current_Step == '2'):
global potent_Mode
dispMode = False
self.upperStep_text.SetLabel("Step 3: Pre-Indent and \n Finalize Point of Contact")
self.Step3_Right.Hide()
self.Step3_Right.Disable()
self.Step1_Left.Hide()
self.Step1_Left.Disable()
self.Step2_Left.Show()
self.Step2_Left.Enable()
self.Step4_Right.Show()
self.Step4_Right.Enable(True)
self.step2_panel.Hide()
self.step2_panel.Disable()
self.step3_panel.Show()
self.step3_panel.Enable(True)
self.headerPanel.Layout()
self.Layout()
potent_Mode = loadCell()

elif(current_Step == '3'):
self.upperStep_text.SetLabel("Step 4: Choose Force or \n Displacement Control Mode")
self.Step4_Right.Hide()
self.Step4_Right.Disable()
self.Step2_Left.Hide()
self.Step2_Left.Disable()
self.Step3_Left.Show()
self.Step3_Left.Enable()
self.NoButton.Show()
self.NoButton.Enable(True)
self.step3_panel.Hide()
self.step3_panel.Disable()
self.step4_panel.Show()
self.step4_panel.Enable(True)
self.headerPanel.Layout()
self.Layout()
elif(current_Step == '5'):
label_val = self.upperStep_text.GetLabel()[8]

if(label_val=='F'):
self.upperStep_text.SetLabel("Step 6: Begin Force \n Control Experiment")
self.Step6_Right.Hide()
self.Step6_Right.Disable()
self.Step4_Left.Hide()
self.Step4_Left.Disable()
self.Step5_Left.Show()
self.Step5_Left.Enable(True)
self.NoButton.Show()
self.NoButton.Enable(True)
self.step5_force_panel.Hide()
self.step5_force_panel.Disable()
self.step6_force_panel.Show()
self.step6_force_panel.Enable(True)
self.headerPanel.Layout()
self.Layout()
def sendToScreen(self,event):
self.upperStep_text.SetLabel("Step 5: Force Calibration")
self.NoButton.Hide()
self.NoButton.Disable()
self.Step5_Left.Hide()
self.Step5_Left.Disable()
self.Step6_Right.Show()
self.Step6_Right.Enable()
self.Step4_Left.Show()
self.Step4_Left.Enable()
self.step6_force_panel.Hide()
self.step6_force_panel.Disable()
self.step5_force_panel.Show()
self.step5_force_panel.Enable()
self.headerPanel.Layout()
```

```
self.Layout()
def changePagesLeft(self,event):
current_Step = self.upperStep_text.GetLabel()[5]
if(current_Step == '2'):
global dispMode
dispMode = False
global potent_Mode
self.upperStep_text.SetLabel("Step 1: Approach Surface \n and Find Point of Contact")
self.Step2_Left.Hide()
self.Step2_Left.Disable()
self.Step3_Right.Hide()
self.Step3_Right.Disable()
self.Step2_Right.Show()
self.Step2_Right.Enable()
self.NoButton_Left.Show()
self.NoButton_Left.Enable(True)
self.step2_panel.Hide()
self.step2_panel.Disable()
self.step1_panel.Show()
self.step1_panel.Enable(True)
self.headerPanel.Layout()
self.Layout()
potent_Mode = loadCell()

elif(current_Step == '3'):
self.upperStep_text.SetLabel("Step 2: Find Minimum and \n Maximum Displacement")
self.Step3_Left.Hide()
self.Step3_Left.Disable()
self.Step4_Right.Hide()
self.Step4_Right.Disable()
self.Step3_Right.Show()
self.Step3_Right.Enable()
self.Step2_Left.Show()
self.Step2_Left.Enable(True)
self.step3_panel.Hide()
self.step3_panel.Disable()
self.step2_panel.Show()
self.step2_panel.Enable(True)
self.headerPanel.Layout()
self.Layout()
elif(current_Step == '4'):
self.upperStep_text.SetLabel("Step 3: Pre-Indent and \n Finalize Point of Contact")
self.Step3_Left.Hide()
self.Step3_Left.Disable()
self.NoButton.Hide()
self.NoButton.Disable()
self.Step4_Right.Show()
self.Step4_Right.Enable()
self.Step2_Left.Show()
self.Step2_Left.Enable(True)
self.step4_panel.Hide()
self.step4_panel.Disable()
self.step3_panel.Show()
self.step3_panel.Enable(True)
self.headerPanel.Layout()
self.Layout()
elif(current_Step == '5'):
label_val = self.upperStep_text.GetLabel()[8]
self.upperStep_text.SetLabel("Step 4: Choose Force or \n Displacement Control Mode")
self.Step4_Left.Hide()
self.Step4_Left.Disable()
self.Step6_Right.Hide()
self.Step6_Right.Disable()
self.NoButton.Show()
self.NoButton.Enable()
self.Step3_Left.Show()
self.Step3_Left.Enable(True)
if(label_val=='F'):
self.step5_force_panel.Hide()
self.step5_force_panel.Disable()
```

```
else:
self.step5_disp_panel.Hide()
self.step5_disp_panel.Disable()
self.step4_panel.Show()
self.step4_panel.Enable(True)
self.headerPanel.Layout()
self.Layout()
elif(current_Step == '6'):
self.upperStep_text.SetLabel("Step 5: Force Calibration")
self.NoButton.Hide()
self.NoButton.Disable()
self.Step5_Left.Hide()
self.Step5_Left.Disable()
self.Step6_Right.Show()
self.Step6_Right.Enable()
self.Step4_Left.Show()
self.Step4_Left.Enable()
self.step6_force_panel.Hide()
self.step6_force_panel.Disable()
self.step5_force_panel.Show()
self.step5_force_panel.Enable()
self.headerPanel.Layout()
self.Layout()
def change_forcePage(self,event):
self.upperStep_text.SetLabel("Step 5: Force Calibration")
self.Step3_Left.Hide()
self.Step3_Left.Disable()
self.NoButton.Hide()
self.NoButton.Disable()
self.Step4_Left.Show()
self.Step4_Left.Enable()
self.Step6_Right.Show()
self.Step6_Right.Enable(True)
self.step5_force_panel.Show()
self.step5_force_panel.Enable(True)
self.step4_panel.Hide()
self.step4_panel.Disable()
self.headerPanel.Layout()
self.Layout()
def change_dispPage(self,event):
self.upperStep_text.SetLabel("Step 5: Begin Displacement \n Controlled Indentation")
self.Step3_Left.Hide()
self.Step3_Left.Disable()
self.NoButton.Hide()
self.NoButton.Disable()
self.Step4_Left.Show()
self.Step4_Left.Enable(True)
self.NoButton.Show()
self.NoButton.Enable(True)
self.step4_panel.Hide()
self.step4_panel.Disable()
self.step5_disp_panel.Show()
self.step5_disp_panel.Enable(True)
self.headerPanel.Layout()
self.Layout()
def changeMult_Single(self,event):
if(self.Calibrate_level_choice.GetString(self.Calibrate_level_choice.GetSelection())[0]=='M'):
self.calibration_param_5_panel.Hide()
self.calibration_param_5_panel.Disable()
self.calib_param_mutliple_5.Show()
self.calib_param_mutliple_5.Enable(True)
self.Layout()

else:
self.calib_param_mutliple_5.Hide()
self.calib_param_mutliple_5.Disable()
self.calibration_param_5_panel.Show()
self.calibration_param_5_panel.Enable(True)
self.Layout()
```

```
#########################################################################################################################
#
# Log Commands, Table Updates
#
#########################################################################
def destroyTables(self):
valueLength = len(self.F_D_ID_sizer_6.GetChildren())
if self.F_D_ID_sizer_6.GetChildren():
for x in range(0,valueLength-2):

self.F_D_ID_sizer_6.Hide(valueLength-x-1)
self.F_D_ID_sizer_6.Remove(valueLength-x-1)
self.F_D_ID_panel_6.Layout()
self.F_D_force_sizer_6.Hide(valueLength-x-1)
self.F_D_force_sizer_6.Remove(valueLength-x-1)
self.F_D_force_panel_6.Layout()
self.F_D_disp_sizer_6.Hide(valueLength-x-1)
self.F_D_disp_sizer_6.Remove(valueLength-x-1)
self.F_D_disp_panel_6.Layout()
self.F_D_vel_sizer_6.Hide(valueLength-x-1)
self.F_D_vel_sizer_6.Remove(valueLength-x-1)
self.F_D_vel_panel_6.Layout()
self.calib_ID_sizer_5.Hide(valueLength-x-1)
self.calib_ID_sizer_5.Remove(valueLength-x-1)
self.calib_ID_panel_5.Layout()
self.calib_force_sizer_5.Hide(valueLength-x-1)
self.calib_force_sizer_5.Remove(valueLength-x-1)
self.calib_force_panel_5.Layout()
self.calib_disp_sizer_5.Hide(valueLength-x-1)
self.calib_disp_sizer_5.Remove(valueLength-x-1)
self.calib_disp_panel_5.Layout()
self.calib_vel_sizer_5.Hide(valueLength-x-1)
self.calib_vel_sizer_5.Remove(valueLength-x-1)
self.calib_vel_panel_5.Layout()
valueLength = len(self.F_D_force_sizer_6.GetChildren())
if self.F_D_force_sizer_6.GetChildren():
for x in range(0, valueLength - 2):
self.F_D_force_sizer_6.Hide(valueLength-x-1)
self.F_D_force_sizer_6.Remove(valueLength-x-1)
self.F_D_force_panel_6.Layout()
def forceControlTableWrite (self,ID,disp,vel,force):
IDLog = wx.StaticText( self.calib_ID_panel_5, -1, str(ID))
self.calib_ID_sizer_5.Add( IDLog, 0, wx.ALL|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.calib_ID_panel_5.Layout()
self.calib_ID_sizer_5.Fit( self.calib_ID_panel_5 )
ForceLog = wx.StaticText( self.calib_force_panel_5, -1, str(force))
self.calib_force_sizer_5.Add( ForceLog, 0, wx.ALL|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.calib_force_panel_5.Layout()
self.calib_force_sizer_5.Fit( self.calib_force_panel_5 )
dispLog = wx.StaticText( self.calib_disp_panel_5, -1, str(disp))
self.calib_disp_sizer_5.Add( dispLog, 0, wx.ALL|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.calib_disp_panel_5.Layout()
self.calib_disp_sizer_5.Fit( self.calib_disp_panel_5 )
velLog = wx.StaticText( self.calib_vel_panel_5, -1, str(vel))
self.calib_vel_sizer_5.Add( velLog, 0, wx.ALL|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.calib_vel_panel_5.Layout()
self.calib_vel_sizer_5.Fit( self.calib_vel_panel_5 )
IDLog_1 = wx.StaticText( self.F_D_ID_panel_6, -1, str(ID))
self.F_D_ID_sizer_6.Add( IDLog_1, 0, wx.ALL|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.F_D_ID_panel_6.Layout()
self.F_D_ID_sizer_6.Fit( self.F_D_ID_panel_6 )
ForceLog_1 = wx.StaticText( self.F_D_force_panel_6, 20000+ID, str(force))
self.F_D_force_sizer_6.Add( ForceLog_1, 0, wx.ALL|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.F_D_force_panel_6.Layout()
self.F_D_force_sizer_6.Fit( self.F_D_force_panel_6 )
dispLog_1 = wx.StaticText( self.F_D_disp_panel_6, 21000+ID, str(disp))
self.F_D_disp_sizer_6.Add( dispLog_1, 0, wx.ALL|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.F_D_disp_panel_6.Layout()
self.F_D_disp_sizer_6.Fit( self.F_D_disp_panel_6 )
velLog_1 = wx.StaticText( self.F_D_vel_panel_6, 22000+ID, str(vel))
```

```
self.F_D_vel_sizer_6.Add( velLog_1, 0, wx.ALL|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.F_D_vel_panel_6.Layout()
self.F_D_vel_sizer_6.Fit( self.F_D_vel_panel_6 )
def dispControlTableWrite (self,disp,vel,acc,var):
if(var == 'd'):
dispLog = wx.StaticText( self.Disp_panel_dp_5, -1, str(disp))
self.disp_sizer_dp_5.Add( dispLog, 0, wx.ALL|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.Disp_panel_dp_5.Layout()
self.disp_sizer_dp_5.Fit( self.Disp_panel_dp_5 )
velLog = wx.StaticText( self.velcity_panel_dp_5, -1, str(vel))
self.velocity_sizer_dp_5.Add( velLog, 0, wx.ALL|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.velcity_panel_dp_5.Layout()
self.velocity_sizer_dp_5.Fit( self.velcity_panel_dp_5 )
accLog = wx.StaticText( self.acceleration_panel_dp_5, -1, str(acc))
self.acc_sizer_dp_5.Add( accLog, 0, wx.ALL|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.acceleration_panel_dp_5.Layout()
self.acc_sizer_dp_5.Fit( self.acceleration_panel_dp_5 )
else:
dispLog = wx.StaticText( self.Disp_panel_6, -1, str(disp))
self.disp_sizer11.Add( dispLog, 0, wx.ALL|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.Disp_panel_6.Layout()
self.disp_sizer11.Fit( self.Disp_panel_6 )
velLog = wx.StaticText( self.velcity_panel_6, -1, str(vel))
self.velocity_sizer11.Add( velLog, 0, wx.ALL|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.velcity_panel_6.Layout()
self.velocity_sizer11.Fit( self.velcity_panel_6 )
accLog = wx.StaticText( self.acceleration_panel_6, -1, str(acc))
self.acc_sizer11.Add( accLog, 0, wx.ALL|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.acceleration_panel_6.Layout()
self.acc_sizer11.Fit( self.acceleration_panel_6 )
def dispViewTableWrite (self,disp,vel,acc,val):
global checkBox_ID
global cb_list
global checkBox_ID_f
global cb_list_f
if(val == 'd'):
viewLog = wx.CheckBox(self.view_panel_fd_5,checkBox_ID)
cb_list.append([viewLog,disp,vel,acc])
#wx.EVT_CHECKBOX(self,checkBox_ID,self.dispBoxClick)
self.view_sizer_fd_5.Add(viewLog,0,wx.ALL|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.view_panel_fd_5.Layout()
self.view_sizer_fd_5.Fit( self.view_panel_fd_5 )
dispLog = wx.StaticText( self.disp_panel_fd_5, -1, str(disp))
self.disp_sizer_fd_5.Add( dispLog, 0, wx.ALL|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.disp_panel_fd_5.Layout()
self.disp_sizer_fd_5.Fit( self.disp_panel_fd_5 )
velLog = wx.StaticText( self.vel_panel_fd_5, -1, str(vel))
self.vel_sizer_fd_5.Add( velLog, 0, wx.ALL|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.vel_panel_fd_5.Layout()
self.vel_sizer_fd_5.Fit( self.vel_panel_fd_5 )
accLog = wx.StaticText( self.acc_panel_fd_5, -1, str(acc))
self.acc_sizer_fd_5.Add( accLog, 0, wx.ALL|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.acc_panel_fd_5.Layout()
self.acc_sizer_fd_5.Fit( self.acc_panel_fd_5 )
checkBox_ID = checkBox_ID+1
else:
viewLog = wx.CheckBox(self.type_panel_fd_6,checkBox_ID_f)
cb_list_f.append([viewLog,disp,vel,acc])
#wx.EVT_CHECKBOX(self,checkBox_ID,self.dispBoxClick)
self.type_sizer_fd_6.Add(viewLog,0,wx.ALL|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.type_panel_fd_6.Layout()
self.type_sizer_fd_6.Fit( self.type_panel_fd_6 )
dispLog = wx.StaticText( self.disp_panel_fd_6, -1, str(disp))
self.disp_sizer_fd_6.Add( dispLog, 0, wx.ALL|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.disp_panel_fd_6.Layout()
self.disp_sizer_fd_6.Fit( self.disp_panel_fd_6 )
velLog = wx.StaticText( self.vel_panel_fd_6, -1, str(vel))
self.vel_sizer_fd_6.Add( velLog, 0, wx.ALL|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.vel_panel_fd_6.Layout()
self.vel_sizer_fd_6.Fit( self.vel_panel_fd_6 )
```

```
accLog = wx.StaticText( self.acc_panel_fd_6, -1, str(acc))
self.acc_sizer_fd_6.Add( accLog, 0, wx.ALL|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.acc_panel_fd_6.Layout()
self.acc_sizer_fd_6.Fit( self.acc_panel_fd_6 )
checkBox_ID_f = checkBox_ID_f+1
def forceViewArrayWrite (self,Force,disp,vel,f0d,checkBox_ID_val):
###ID in 5,000, forces as value 5010 displacement 5011

global cb_list_Values
global presentImages
sizeMove = len(presentImages)
isAVal = False
ForceBool = False
if(f0d):
for x in range(0, sizeMove):
ForceBool = True
if(presentImages[x][0][0]==Force and presentImages[x][0][1]==vel and not(presentImages[x][0][3])):
isAVal = True

presentImages[x][0][3]=True
presentImages[x][0].append(checkBox_ID_val)
ForceLog = wx.CheckBox(self.type_panel_f_d_6,checkBox_ID_val)
cb_list_Values.append([ForceLog,Force,vel,disp,ForceBool])
elif(presentImages[x][0][0]==Force and not(presentImages[x][0][1]==vel)):
isAVal = True

presentImages.append([[Force,vel,disp,True,checkBox_ID_val],[Force,vel,disp,False]])
ForceLog = wx.CheckBox(self.type_panel_f_d_6,checkBox_ID_val)
cb_list_Values.append([ForceLog,Force,vel,disp,ForceBool])
elif(presentImages[x][0][0]==Force and presentImages[x][0][1]==vel and presentImages[x][0][3]):
isAVal = True

presentImages[x][0]= [Force,vel,disp,True,checkBox_ID_val]
ForceLog = wx.CheckBox(self.type_panel_f_d_6,checkBox_ID_val)
cb_list_Values.append([ForceLog,Force,vel,disp,ForceBool])

if(not(isAVal)):

presentImages.append([[Force,vel,disp,True,checkBox_ID_val],[Force,vel,disp,False]])
ForceLog = wx.CheckBox(self.type_panel_f_d_6,checkBox_ID_val)
cb_list_Values.append([ForceLog,Force,vel,disp,ForceBool])
return [cb_list_Values,presentImages]
else:
for x in range(0, sizeMove):
ForceBool = False
if(presentImages[x][1][0]==Force and presentImages[x][1][1]==vel and not(presentImages[x][1][3])):
isAVal = True

presentImages[x][1][3]=True
presentImages[x][1].append(checkBox_ID_val)
ForceLog = wx.CheckBox(self.type_panel_f_d_6,checkBox_ID_val)
cb_list_Values.append([ForceLog,Force,vel,disp,ForceBool])
elif(presentImages[x][1][0]==Force and not(presentImages[x][1][1]==vel)):
isAVal = True

presentImages.append([[Force,vel,disp,False],[Force,vel,disp,True,checkBox_ID_val]])
ForceLog = wx.CheckBox(self.type_panel_f_d_6,checkBox_ID_val)
cb_list_Values.append([ForceLog,Force,vel,disp,ForceBool])
elif(presentImages[x][1][0]==Force and presentImages[x][1][1]==vel and presentImages[x][1][3]):
isAVal = True

presentImages[x][1]= [Force,vel,disp,True,checkBox_ID_val]
ForceLog = wx.CheckBox(self.type_panel_f_d_6,checkBox_ID_val)
cb_list_Values.append([ForceLog,Force,vel,disp,ForceBool])
if(not(isAVal)):

presentImages.append([[Force,vel,disp,False],[Force,vel,disp,True,checkBox_ID_val]])
ForceLog = wx.CheckBox(self.type_panel_f_d_6,checkBox_ID_val)
cb_list_Values.append([ForceLog,Force,vel,disp,ForceBool])
return [cb_list_Values,presentImages]
```

```
def forceViewTableWrite (self,cb_list_Values,presentImages):
valueLength = len(self.force_sizer_f_d_6.GetChildren())
if self.type_sizer_f_d_6.GetChildren():
for x in range(0,valueLength-2):

self.force_sizer_f_d_6.Hide(valueLength-x-1)
self.force_sizer_f_d_6.Remove(valueLength-x-1)
self.force_panel_f_d_6.Layout()
self.disp_sizer_f_d_6.Hide(valueLength-x-1)
self.disp_sizer_f_d_6.Remove(valueLength-x-1)
self.disp_panel_f_d_6.Layout()
self.velocity_sizer_f_d_6.Hide(valueLength-x-1)
self.velocity_sizer_f_d_6.Remove(valueLength-x-1)
self.velocity_panel_f_d_6.Layout()


self.type_gridSizer_f_d_6.Destroy()
self.type_gridSizer_f_d_6 = wx.GridSizer( 0, 2, 0, 0 )
#if self.type_gridSizer_f_d_6.GetChildren():
# for x in range(0,len(self.type_gridSizer_f_d_6.GetChildren())):
#if(x%2 == 0):
# self.type_gridSizer_f_d_6.Hide(x)
# self.type_gridSizer_f_d_6.Remove(x)
# self.type_gridSizer_f_d_6.Layout()
# print len(self.type_gridSizer_f_d_6.GetChildren())
for x in range(0,len(presentImages)):
if(presentImages[x][0][3]):
boxId = presentImages[x][0][4]
for y in range(0,len(cb_list_Values)):
if(boxId == cb_list_Values[y][0].GetId()):
forceBox = cb_list_Values[y][0]
self.type_gridSizer_f_d_6.Add(forceBox,0,wx.ALL|wx.ALIGN_CENTER_HORIZONTAL, 5 )
else:
noText = wx.StaticText( self.type_panel_f_d_6, wx.ID_ANY, u"_", wx.DefaultPosition, wx.DefaultSize, wx.ALIGN_LEFT )
self.type_gridSizer_f_d_6.Add(noText,0,wx.ALL|wx.ALIGN_CENTER_HORIZONTAL, 5)
#wx.EVT_CHECKBOX(self,checkBox_ID,self.dispBoxClick)
if(presentImages[x][1][3]):
boxId = presentImages[x][1][4]
for y in range(0,len(cb_list_Values)):
if(boxId == cb_list_Values[y][0].GetId()):
dispBox = cb_list_Values[y][0]
self.type_gridSizer_f_d_6.Add(dispBox,0,wx.ALL|wx.ALIGN_CENTER_HORIZONTAL, 5 )
else:
noText = wx.StaticText( self.type_panel_f_d_6, wx.ID_ANY, u"_", wx.DefaultPosition, wx.DefaultSize, wx.ALIGN_LEFT )
self.type_gridSizer_f_d_6.Add(noText,wx.ID_ANY,wx.ALL|wx.ALIGN_CENTER_HORIZONTAL, 5)
self.type_gridSizer_f_d_6.Layout()
self.type_sizer_f_d_6.Add(self.type_gridSizer_f_d_6)
#self.type_sizer_f_d_6.Fit( self.type_panel_f_d_6 )

#self.type_panel_f_d_6.Layout()

for x in range(0,len(presentImages)):
forceLog = wx.StaticText( self.force_panel_f_d_6, -1, str(presentImages[x][0][0]))
self.force_sizer_f_d_6.Add( forceLog, 0, wx.ALL|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.force_panel_f_d_6.Layout()
self.force_sizer_f_d_6.Fit( self.force_panel_f_d_6 )
dispLog = wx.StaticText( self.disp_panel_f_d_6, -1, str(presentImages[x][0][2]))
self.disp_sizer_f_d_6.Add( dispLog, 0, wx.ALL|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.disp_panel_f_d_6.Layout()
self.disp_sizer_f_d_6.Fit( self.disp_panel_f_d_6 )
velLog = wx.StaticText( self.velocity_panel_f_d_6, -1, str(presentImages[x][0][1]))
self.velocity_sizer_f_d_6.Add( velLog, 0, wx.ALL|wx.ALIGN_CENTER_HORIZONTAL, 5 )
self.velocity_panel_f_d_6.Layout()
self.velocity_sizer_f_d_6.Fit( self.velocity_panel_f_d_6 )


def logWrite (self,type_indent,time,disp,force,vel,acc,hold):
global fileLoc
global save_log_dialogFileName
global yInt
```

```
if(type_indent == "Manual"):
FILE = open(fileLoc+'\\'+ save_log_dialogFileName+'.txt',"a")
FILE.write("time: " + str(time) + " , " + "vel: " +str(vel)+ " , " + "{Manual MODE engaged}\n")
elif(type_indent == "Force"):
FILE = open(fileLoc+'\\'+ save_log_dialogFileName+'.txt',"a")
FILE.write("type: " + str(type_indent) +"force: " + force + ", time: " + str(time) + " , " + "disp: " + str(disp) + " , " + "vel: " + str(vel) + " , " +
"acc: " + str(acc) +"\n")
else:
FILE = open(fileLoc+'\\'+ save_log_dialogFileName+'.txt',"a")
FILE.write("Calibration Parameters: Voltage = " + "6.022*force + " + str(yInt)+"\n")
FILE.write("type: " + str(type_indent) + ", " +"force: " + force + ", time: " + str(time) + " , " + "disp: " + str(disp) + " , " + "vel: " + str(vel) + " , "
+ "acc: " + str(acc) +"\n")
typeLog = wx.StaticText( self.type_panel, -1, str(type_indent))
self.type_sizer.Add( typeLog, 0, wx.ALL|wx.ALIGN_RIGHT, 5 )
self.type_panel.Layout()
self.type_sizer.Fit( self.type_panel )
timeLog = wx.StaticText( self.time_panel, -1, str(time))
self.time_sizer.Add( timeLog, 0, wx.ALL|wx.ALIGN_RIGHT, 5 )
self.time_panel.Layout()
self.time_sizer.Fit( self.time_panel )
dispLog = wx.StaticText( self.disp_panel, -1, str(disp))
self.disp_sizer.Add( dispLog, 0, wx.ALL|wx.ALIGN_RIGHT, 5 )
self.disp_panel.Layout()
self.disp_sizer.Fit( self.disp_panel )
forceLog = wx.StaticText( self.force_panel, -1, str(force))
self.force_sizer.Add( forceLog, 0, wx.ALL|wx.ALIGN_RIGHT, 5 )
self.force_panel.Layout()
self.force_sizer.Fit( self.force_panel )
velLog = wx.StaticText( self.vel_panel, -1, str(vel))
self.vel_sizer.Add( velLog, 0, wx.ALL|wx.ALIGN_RIGHT, 5 )
self.vel_panel.Layout()
self.vel_sizer.Fit( self.vel_panel )
accLog = wx.StaticText( self.acc_panel, -1, str(acc))
self.acc_sizer.Add( accLog, 0, wx.ALL|wx.ALIGN_RIGHT, 5 )
self.acc_panel.Layout()
self.acc_sizer.Fit( self.acc_panel )
holdLog = wx.StaticText( self.hold_panel, -1, str(hold))
self.hold_sizer.Add( holdLog, 0, wx.ALL|wx.ALIGN_RIGHT, 5 )
self.hold_panel.Layout()
self.hold_sizer.Fit( self.hold_panel )
#########################################################################################################################
#
# Page one - Manual Mode Indentations
#
#########################################################################################################################


def Engage(self, event):
global manual_speed
global manMode
global GPIOLocation
global Manual_thread
global Manual_buttonCol
global Alone
Alone = False
self.resetEventTimer()
current_Time = datetime.datetime.now()
logTime_String = current_Time.strftime("%H:%M:%S")
self.Status.SetLabel("Manual")
self.Status.SetForegroundColour((255,0,0))
self.sizer_panel.Layout()
#statusEntry.SetValue("Manual Mode")
#statusEntry.SetForegroundColour(wx.RED)
Manual_buttonCol = self.Engage_Button.GetBackgroundColour()
manual_speed = float(self.vel_appoach_1.GetValue())
self.logWrite ("Manual" ,logTime_String,",",manual_speed,",")

manMode = True
GPIOLocation = "GPIO2.ADC2"
self.Engage_Button.SetForegroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_BTNHIGHLIGHT ) )
```

```
self.Engage_Button.SetBackgroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_3DDKSHADOW ) )
self.Disengage_Button.SetForegroundColour( wx.SystemSettings.GetColour(wx.SYS_COLOUR_BTNTEXT) )
self.Disengage_Button.SetBackgroundColour( Manual_buttonCol )
self.indent_poc_1.Disable()
self.setPOC_button_1.Disable()
Manual_thread = ManualThread()
def Disengage(self,event):
global manMode
global Manual_buttonCol
global potent_Mode
manMode = False
time.sleep(1)

self.resetEventTimer()
self.Disengage_Button.SetForegroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_BTNHIGHLIGHT ) )
self.Disengage_Button.SetBackgroundColour( wx.SystemSettings.GetColour( wx.SYS_COLOUR_3DDKSHADOW ) )
self.Engage_Button.SetForegroundColour( wx.SystemSettings.GetColour(wx.SYS_COLOUR_BTNTEXT) )
self.Engage_Button.SetBackgroundColour( Manual_buttonCol )
self.indent_poc_1.Enable()
self.setPOC_button_1.Enable()
self.Status.SetLabel("Ready")
self.Status.SetForegroundColour((101,218,80))
self.sizer_panel.Layout()
potent_Mode = loadCell()

def UpdateSpeed(self,event):
global areStopped
global upSpeed
global downSpeed
areStopped = True
if(areStopped):
upSpeed = (-float(self.vel_appoach_1.GetValue()))
downSpeed = (float(self.vel_appoach_1.GetValue()))

##############################################################################################
#
# Displacement Control
#
##############################################################################################

def displacementControlFunct(self,disp_Type,yInt):
global pointOfContact_Disp
global fileLoc
global save_log_dialogFileName
global servoCycle
global checkBox_ID
global checkBox_ID_f
global Alone
global potent_Mode
self.Status.SetForegroundColour((255,0,0))
self.Status.SetLabel("Busy")
self.sizer_panel.Layout()
dispFileName = ''
forceFileName = ''
type_disp = ''
Cycle = False
Retract = False
Extend = False
Alone = False

if(disp_Type == '1'):
displacement = float(self.disp_spin_21.GetValue())
velocity = float(self.vel_spin_21.GetValue())
acceleration = float(self.acc_spin_21.GetValue())
hold = float(self.hold_spin_21.GetValue())
if(self.cycle_21.GetValue()):
Cycle = self.cycle_21.GetValue()
type_disp = 'C'
Retract = False
Extend = False
```

```
if(self.retract_21.GetValue()):
Retract = self.retract_21.GetValue()
type_disp = 'R'
Extend = False
Cycle = False
if(self.extend_21.GetValue()):
Extend = self.extend_21.GetValue()
type_disp = 'E'
Cycle = False
Retract=False
elif(disp_Type == '2'):
displacement = float(self.disp_spin_2.GetValue())
velocity = float(self.vel_spin_2.GetValue())
acceleration = float(self.acc_spin_2.GetValue())
hold = float(self.hold_spin_2.GetValue())
if(self.cycle_2.GetValue()):
Cycle = self.cycle_2.GetValue()
type_disp = 'C'
Retract = False
Extend = False
if(self.retract_2.GetValue()):
Retract = self.retract_2.GetValue()
type_disp = 'R'
Extend = False
Cycle = False
if(self.extend_2.GetValue()):
Extend = self.extend_2.GetValue()
type_disp = 'E'
Cycle = False
Retract=False
elif(disp_Type == '3'):
displacement = float(self.disp_spin_3.GetValue())
velocity = float(self.vel_spin_3.GetValue())
acceleration = float(self.acc_spin_3.GetValue())
hold = float(self.hold_spin_3.GetValue())
if(self.cycle_3.GetValue()):
Cycle = self.cycle_3.GetValue()
type_disp = 'C'
Retract = False
Extend = False
if(self.retract_3.GetValue()):
Retract = self.retract_3.GetValue()
type_disp = 'R'
Extend = False
Cycle = False
if(self.extend_3.GetValue()):
Extend = self.extend_3.GetValue()
type_disp = 'E'
Cycle = False
Retract=False

elif(disp_Type == 'D'):
displacement = float(self.disp_spin_dp_5.GetValue())
velocity = float(self.vel_spin_dp_5.GetValue())
acceleration = float(self.acc_spin_dp_5.GetValue())
hold = float(self.hold_spin_dp_5.GetValue())
if(self.cycle_dp_5.GetValue()):
Cycle = self.cycle_dp_5.GetValue()
type_disp = 'C'
Retract = False
Extend = False
myIndenter.Record_DispControl(socketId,positioner,hold,servoCycle)

if(self.retract_dp_5.GetValue()):
Retract = self.retract_dp_5.GetValue()
type_disp = 'R'
Extend = False
Cycle = False
if(self.extend_dp_5.GetValue()):
Extend = self.extend_dp_5.GetValue()
```

```
type_disp = 'E'
Cycle = False
Retract=False
forceFileName = save_log_dialogFileName+ "disp_"+str(displacement)+"vel_"+str(velocity)+'_forceTrace.png'
dispFileName = save_log_dialogFileName+"disp_"+str(displacement)+"vel_"+str(velocity)+'_displacementTrace.png'
boxID_fileName.append([checkBox_ID,dispFileName,forceFileName])
self.dispControlTableWrite (displacement,velocity,acceleration,'d')
self.dispViewTableWrite (displacement,velocity,acceleration,'d')
elif(disp_Type == 'F'):
displacement = float(self.fine_disp_6.GetValue())
velocity = float(self.fine_vel_6.GetValue())
acceleration = float(self.fine_acc_6.GetValue())
hold = float(self.fine_hold_6.GetValue())
if(self.cycle_6.GetValue()):
Cycle = self.cycle_6.GetValue()
type_disp = 'C'
Retract = False
Extend = False
myIndenter.Record_DispControl(socketId,positioner,hold,servoCycle)

if(self.retract_6.GetValue()):
Retract = self.retract_6.GetValue()
type_disp = 'R'
Extend = False
Cycle = False
if(self.extend_6.GetValue()):
Extend = self.extend_6.GetValue()
type_disp = 'E'
Cycle = False
Retract=False
forceFileName = save_log_dialogFileName+ "disp_"+str(displacement)+"vel_"+str(velocity)+'_forceTrace.png'
dispFileName = save_log_dialogFileName+"disp_"+str(displacement)+"vel_"+str(velocity)+'_displacementTrace.png'
fineboxID_fileName.append([checkBox_ID_f,dispFileName,forceFileName])
self.dispControlTableWrite (displacement,velocity,acceleration,'f')
self.dispViewTableWrite (displacement,velocity,acceleration,'f')


ED = False
voltage = myIndenter.fineDisplacementControl(myxps,socketId,positioner,group, displacement, velocity, acceleration, hold, Cycle, Retract,
Extend,yInt)
global minMaxForce
minMaxForce = loadCellCalibrateVoltage(voltage)
if(disp_Type == 'D' or disp_Type == 'F' ):
chosenForce = ''
typeIndent = 'd'
myIndenter.endRecord_DispControl(socketId,positioner,hold)


myIndenter.output(chosenForce,velocity,displacement,hold,pointOfContact_Disp,socketId,fileLoc,yInt,servoCycle,dispFileName,forceFileName,
typeIndent)
current_Time = datetime.datetime.now()
logTime_String = current_Time.strftime("%H:%M:%S")
self.logWrite ("Disp. "+type_disp ,logTime_String,displacement,",velocity,acceleration,hold)
if (type_disp == 'R' or type_disp == 'E'):
potent_Mode = loadCell()
ED = True


self.Status.SetLabel("Ready")
self.Status.SetForegroundColour((101,218,80))
self.sizer_panel.Layout()
ED = False
return ED


#############################################################################
#
# Step 1
#
#############################################################################
```

```
def displacementIndent(self,event):
global yInt
self.resetEventTimer()
global Alone
global potent_Mode
Alone= False
disp_Type = '1'
ED = self.displacementControlFunct(disp_Type,yInt)
time.sleep(.1)

if(not(ED)):
potent_Mode = loadCell()

def setPointOfContact1(self,event):
global potent_Mode
global Alone
Alone = False
global pointOfContact_1
pointOfContact_1 = myIndenter.setPointOfContact_1(myxps,socketId,positioner)
[errorCode, returnString]=myxps.PositionerSGammaParametersSet(socketId,positioner,10,160,.005,.05)
myxps.GroupMoveRelative(socketId,group,[-.6])
time.sleep(.5)
potent_Mode = loadCell()
################################################################################
#
# Step 2
#
################################################################################
def displacementIndent2(self,event):
global yInt
global dispMode
global setMax
dispMode = False
self.resetEventTimer()

disp_Type = '2'
self.displacementControlFunct(disp_Type,yInt)
time.sleep(.1)
if(setMax == False):
dispMode = True
global Disp_Mode
Disp_Mode = DispThread()


def setDispMin(self,event):
global setMin
setMin = True
global displayMinForce_box_2
global displayMinDisp_box_2
global dispThresh
global forceThresh
global minMaxForce
#dispThresh = myxps.GroupPositionCurrentGet(socketId,positioner,1)[1]
#forceThresh = loadCellCalibrateVoltage(myxps.GPIOAnalogGet(socketId,["GPIO2.ADC1"])[1])*1000
forceThresh = minMaxForce*1000
dispThresh = float(self.disp_spin_2.GetValue())+pointOfContact_1
displayMinDisp_box_2.SetBackgroundColour( "Grey")
displayMinDisp_box_2.SetValue(str('{0:.2f}'.format(float(dispThresh-pointOfContact_1-.6))))
self.displaySizer_2.Layout()
displayMinForce_box_2.SetBackgroundColour( "Grey" )
#displayMinForce_box_2.SetValue(str('{0:.0f}'.format(forceThresh)))
displayMinForce_box_2.SetValue(str('{0:.0f}'.format(forceThresh)))

self.displaySizer_2.Layout()
self.POCForce_textCtrl_5_mult.SetValue(str('{0:.2f}'.format(forceThresh)))


def setDispMax(self,event):
global setMax
global dispMode
```

```
global dispMaxDisp_box_2
global dispMaxForce_box_2
global dispSatur
global forceSatur
global potent_Mode
global minMaxForce
#dispSatur = myxps.GroupPositionCurrentGet(socketId,positioner,1)[1]
#forceSatur = loadCellCalibrateVoltage(myxps.GPIOAnalogGet(socketId,["GPIO2.ADC1"])[1])*1000
forceSatur = minMaxForce*1000
dispSatur = float(self.disp_spin_2.GetValue())+pointOfContact_1
dispMode = False
setMax = True
dispMaxDisp_box_2.SetBackgroundColour( "Grey" )
dispMaxDisp_box_2.SetValue(str('{0:.2f}'.format(float(dispSatur-pointOfContact_1-.6))))
dispMaxForce_box_2.SetBackgroundColour( "Grey" )
dispMaxForce_box_2.SetValue(str('{0:.0f}'.format(forceSatur)))

self.displaySizer_2.Layout()
self.highestDisp_3.SetValue(str('{0:.2f}'.format(dispSatur-pointOfContact_1-.6)))
self.highestForce_textCtrl_5_mult.SetValue(str('{0:.2f}'.format(forceSatur)))
forceSatur = forceSatur/1000
potent_Mode = loadCell()
#############################################################################
#
# Step 3
#
#############################################################################

def preIndent(self,event):
global pointOfContact_1
global Alone
Alone = False

global potent_Mode
self.resetEventTimer()
self.Status.SetForegroundColour((255,0,0))
self.Status.SetLabel("Busy")
self.sizer_panel.Layout()
dispSatur = self.highestDisp_3.GetValue()
dispSatur = float(dispSatur)
for x in range(0,3):
time.sleep(5)
myxps.GroupMoveAbsolute(socketId,group,[dispSatur+pointOfContact_1])
time.sleep(5)
myxps.GroupMoveAbsolute(socketId,group,[pointOfContact_1])
myxps.GroupMoveRelative(socketId,group,[-.6])
self.resetEventTimer()

self.Status.SetLabel("Ready")
self.Status.SetForegroundColour((101,218,80))
self.sizer_panel.Layout()

potent_Mode = loadCell()

def displacementIndent3(self,event):
global yInt
global potent_Mode
global Alone
Alone = False
self.resetEventTimer()
disp_Type = '3'
ED = self.displacementControlFunct(disp_Type,yInt)
time.sleep(.1)
if(not(ED)):
potent_Mode = loadCell()

self.resetEventTimer()

def setPointOfContactFinal(self,event):
global potent_Mode
```

```
global Alone
Alone = False
global pointOfContact_Disp

self.Status.SetForegroundColour((255,0,0))
self.Status.SetLabel("Busy")
self.sizer_panel.Layout()

pointOfContact_Disp = myIndenter.setPointOfContact_1(myxps,socketId,positioner)
[errorCode, returnString]=myxps.PositionerSGammaParametersSet(socketId,positioner,10,160,.005,.05)
myxps.GroupMoveRelative(socketId,group,[-.6])
time.sleep(.5)


self.Status.SetLabel("Ready")
self.Status.SetForegroundColour((101,218,80))
self.sizer_panel.Layout()

potent_Mode = loadCell()
##############################################################################
#
# Step 5: Displacement
#
##############################################################################

def displacementIndentExp(self,event):
global Alone
global yInt
global potent_Mode
Alone = False
self.resetEventTimer()
disp_Type = 'D'
ED = self.displacementControlFunct(disp_Type,yInt)
if(not(ED)):
potent_Mode = loadCell()

self.resetEventTimer()

def scale_bitmap(self,bitmap, width, height):
image = wx.ImageFromBitmap(bitmap)
image = image.Scale(width, height, wx.IMAGE_QUALITY_HIGH)
result = wx.BitmapFromImage(image)
return result
def viewDispPlots(self,event):
global boxID_fileName
global cb_list
cb_Id = []
for x in range(0,len(cb_list)):
if(cb_list[x][0].GetValue()):
cb_Id.append(cb_list[x][0].GetId())
forcePlot = []
dispPlot = []
for x in range(0,len(boxID_fileName)):
for y in range(0,len(cb_Id)):
if(boxID_fileName[x][0]==cb_Id[y]):
dispPlot.append(boxID_fileName[x][1])
forcePlot.append(boxID_fileName[x][2])

frame = PicturePopUp()
global plotScroll
global plotSizer
for i in range(0,len(dispPlot)):
dispPlotImage = wx.Image(fileLoc+'\\'+dispPlot[i], wx.BITMAP_TYPE_ANY).ConvertToBitmap()
forcePlotImage = wx.Image(fileLoc+'\\'+forcePlot[i], wx.BITMAP_TYPE_ANY).ConvertToBitmap()
disIndex = dispPlot[i].find('disp_')
velIndex = dispPlot[i].find('vel_')
dispVal = dispPlot[disIndex+5:disIndex+8]
velVal = dispPlot[velIndex+4:velIndex+7]
self.plotPanel = wx.Panel( plotScroll, i+4000, wx.DefaultPosition, wx.DefaultSize, wx.TAB_TRAVERSAL )
self.plotPanel.SetMinSize( wx.Size(500,-1 ) )
```

```
plotPanel_sizer_pop = wx.FlexGridSizer( 2, 1, 0, 0 )
plotPanel_sizer_pop.SetFlexibleDirection( wx.VERTICAL )
plotPanel_sizer_pop.SetNonFlexibleGrowMode( wx.FLEX_GROWMODE_SPECIFIED )
plotPanel_sizer_pop.SetMinSize( wx.Size(500,-1 ) )

self.textHeader_pop = wx.StaticText( self.plotPanel, wx.ID_ANY, "Displacement Control: Displ. - "+ str(cb_list[i][1])+ " Velocity - " +
str(cb_list[i][2]) + " Accel.- " + str(cb_list[i][3]), wx.DefaultPosition, wx.DefaultSize, 0 )
self.textHeader_pop.Wrap( -1 )
self.textHeader_pop.SetFont( wx.Font( 14, 74, 90, 92, False, "Calibri" ) )
self.textHeader_pop.SetMinSize( wx.Size(500,-1 ) )
plotPanel_sizer_pop.Add( self.textHeader_pop, 0, wx.ALL, 5 )
plots = wx.GridSizer( 0, 2, 0, 0 )
plots.SetMinSize( wx.Size(500,100 ) )
dispbitmap = self.scale_bitmap(dispPlotImage, 200, 100)
self.displaceImage = wx.StaticBitmap(self.plotPanel, -1, dispbitmap)
plots.Add( self.displaceImage, 0, wx.TOP|wx.BOTTOM|wx.ALIGN_CENTER, 5 )
forcebitmap = self.scale_bitmap(forcePlotImage, 200, 100)

self.forceImage = wx.StaticBitmap(self.plotPanel, -1, forcebitmap)
plots.Add( self.forceImage, 0, wx.TOP|wx.BOTTOM|wx.ALIGN_CENTER, 5 )

plotPanel_sizer_pop.Add( plots, 1, wx.EXPAND, 5 )
self.plotPanel.SetSizer( plotPanel_sizer_pop )
self.plotPanel.Layout()
plotPanel_sizer_pop.Fit( self.plotPanel )
plotSizer.Add(self.plotPanel,0,wx.ALL|wx.ALIGN_CENTER_HORIZONTAL, 5 )
plotScroll.Layout()
plotSizer.Fit( plotScroll )
#print dispPlot
frame.Show()
#############################################################################
#
# Step 5: Force Control
#
#############################################################################
def Calibrate_Single(self,event):
global pointOfContact_Disp
global feedForward
global compForceCounter
global CompleteDisplacement
global CompleteForce
global forceSatur
global yInt
global trajectRoute
global Alone
global tauVal
POC = pointOfContact_Disp
Alone = False

self.Status.SetForegroundColour((255,0,0))
self.Status.SetLabel("Busy")
self.sizer_panel.Layout()
self.destroyTables()
self.resetEventTimer()
failure = False
partition = False
interval = ''
velIndex = self.vel_choice_5.GetCurrentSelection()
vel = self.vel_choice_5Choices[velIndex]
highForce = self.highestForce_textCtrl_5.GetValue()
saturation = ''
forceThresh = ''
vel = float(vel)
highForce = float(highForce)/1000
CRCNSForce =
myIndenter.CalibrateCRCNSForce(interval,partition,vel,CompleteForce,CompleteDisplacement,forceThresh,pointOfContact_Disp,highForce,sat
uration,compForceCounter,failure,feedForward,yInt,socketId, positioner,group,trajectRoute,True,forceSatur)
CompleteForce = CRCNSForce[0]
compForceCounter = CRCNSForce[1]
tauVal.append(CRCNSForce[2])
```

```
self.F_D_ID_6.Clear()
for x in range(0,len(CompleteForce)):
ID = x+1
self.F_D_ID_6.Append(str(ID))
disp = CompleteForce[x][1]
force = CompleteForce[x][0]*1000
force = str('{0:.0f}'.format(force))
vel = CompleteForce[x][2]
self.forceControlTableWrite(ID,disp-POC,vel,force)
self.F_D_Calibrate_button_6 = wx.Button( self.F_D_force_panel_6,26, u"Calibrate New \n Force", wx.DefaultPosition, wx.DefaultSize, 0 )
self.Bind(wx.EVT_BUTTON,self.sendToScreen,id=26)
self.F_D_force_sizer_6.Add( self.F_D_Calibrate_button_6, 0, wx.ALL, 5 )
self.F_D_force_panel_6.Layout()
self.F_D_force_sizer_6.Fit( self.F_D_force_panel_6 )
current_Time = datetime.datetime.now()
logTime_String = current_Time.strftime("%H:%M:%S")
FILE = open(fileLoc+'\\'+ save_log_dialogFileName+'.txt',"a")
FILE.write("Calibration from " + str(forceThresh) + " to " + str(highForce) + " by interval " + str(interval) + " at time: " + str(logTime_String) + "
, " + "vel: " +str(vel)+"\n")
self.Status.SetLabel("Ready")
self.Status.SetForegroundColour((101,218,80))
self.sizer_panel.Layout()
self.resetEventTimer()
global potent_Mode
potent_Mode = loadCell()
#print CRCNSForce

def Calibrate_Mult(self,event):
global pointOfContact_Disp
global feedForward
global compForceCounter
global CompleteDisplacement
global CompleteForce
global forceSatur
global yInt
global trajectRoute
global Alone
global tauVal
POC = pointOfContact_Disp
Alone = False

self.Status.SetForegroundColour((255,0,0))
self.Status.SetLabel("Busy")
self.sizer_panel.Layout()

self.destroyTables()
self.resetEventTimer()

failure = False
partition = False
interval = self.interval_textCtrl_5_mult.GetValue()
partition = self.line_expo_choice.GetCurrentSelection()
if(partition == 1):
partition = True
interval = "Exponential"
velIndex = self.vel_choice_5_mult.GetCurrentSelection()
vel = self.vel_choice_5_multChoices[velIndex]
highForce = self.highestForce_textCtrl_5_mult.GetValue()
forceThresh = self.lowestForce_textCtrl_5_mult.GetValue()
saturation = forceSatur

interval = float(interval)/1000
vel = float(vel)
highForce = float(highForce)/1000
forceThresh = float(forceThresh)

CRCNSForce =
myIndenter.CalibrateCRCNSForce(interval,partition,vel,CompleteForce,CompleteDisplacement,forceThresh,pointOfContact_Disp,highForce,sat
uration,compForceCounter,failure,feedForward,yInt,socketId, positioner,group,trajectRoute,False,forceSatur)
```

```
CompleteForce = CRCNSForce[0]
compForceCounter = CRCNSForce[1]
tauVal = (CRCNSForce[2])
self.F_D_ID_6.Clear()

for x in range(0,len(CompleteForce)):
ID = x+1
self.F_D_ID_6.Append(str(ID))
disp = CompleteForce[x][1]
force = CompleteForce[x][0]*1000
force = str('{0:.0f}'.format(force))
vel = CompleteForce[x][2]
self.forceControlTableWrite(ID,disp-POC,vel,force)
self.F_D_Calibrate_button_6 = wx.Button( self.F_D_force_panel_6,26, u"Calibrate New \n Force", wx.DefaultPosition, wx.DefaultSize, 0 )
self.Bind(wx.EVT_BUTTON,self.sendToScreen,id=26)
self.F_D_force_sizer_6.Add( self.F_D_Calibrate_button_6, 0, wx.ALL, 5 )
self.F_D_force_panel_6.Layout()
self.F_D_force_sizer_6.Fit( self.F_D_force_panel_6 )
current_Time = datetime.datetime.now()
logTime_String = current_Time.strftime("%H:%M:%S")
FILE = open(fileLoc+'\\'+ save_log_dialogFileName+'.txt',"a")
FILE.write("Calibration from " + str(forceThresh) + " to " + str(highForce) + " by interval " + str(interval) + " at time: " + str(logTime_String) + "
, " + "vel: " +str(vel)+"\n")
self.Status.SetLabel("Ready")
self.Status.SetForegroundColour((101,218,80))
self.sizer_panel.Layout()
self.resetEventTimer()
global potent_Mode
potent_Mode = loadCell()
##########################################################################################################################
###############
#
# Step 6: Force Control
#
##########################################################################################################################
###############
def restoreDefaults(self,event):
self.F_D_skinThickness.SetValue(1)
def chosenID(self,event):
global CompleteForce
global CompleteDisplacement
global pointOfContact_Disp
POC = pointOfContact_Disp
ID = self.F_D_ID_6.GetCurrentSelection()
self.F_D_force_textCtrl_6.SetValue(str('{0:.2f}'.format(CompleteForce[ID][0]*1000)))
self.F_D_disp_textCtrl_6.SetValue(str('{0:.2f}'.format(CompleteForce[ID][1]-POC)) )
self.F_D_vel_textCtrl_6.SetValue(str('{0:.2f}'.format(CompleteForce[ID][2])))
def runIndent_FD_6(self,event):
global Alone
global pointOfContact_Disp
global fileLoc
global save_log_dialogFileName
global servoCycle
global yInt
global feedForward
global group
global socketId
global presentImages
global checkBox_ID_val
global checkBox_ID_values
global forceID_fileName
global tauVal
self.resetEventTimer()

skin_Thickness = self.F_D_skinThickness.GetValue()
skin_Thickness = 10**(float(skin_Thickness)/10)
checkBox_ID_val = checkBox_ID_val+1
Alone = False

self.Status.SetForegroundColour((255,0,0))
```

```
self.Status.SetLabel("Busy")
self.sizer_panel.Layout()

complete = False
typeIndent = ''
ActInt = self.F_D_ID_6.GetCurrentSelection()
Durstr = self.F_D_holdTime_6.GetValue()
velocity =float(self.F_D_vel_textCtrl_6.GetValue())
displacementExp = CompleteForce[ActInt][1]
currentTau = 0
totalTrajTime = CompleteForce[ActInt][5]
for x in range(len(tauVal)):
if(tauVal[x][0]==displacementExp and tauVal[x][1]==velocity):
currentTau = tauVal[x][2]
Dur = float(Durstr)-1.5
POC = pointOfContact_Disp
hold = str(float(Dur)+1.5)
holdVal = float(hold)
if(self.type_mec_Indent.GetCurrentSelection()==1):
forceActVal = int(ActInt)
AccMax = 160
expVol = float(loadCellCalibrateForce(CompleteForce[ActInt][0]))
### def RunIndentation(self,velocity, expDisp , Acceleration,
Duration,expVol,Force,idString,hairCycle,Diag,forceInd,POC,feedForward,positioner,socketId,group,fileLoc,yInt,servo,dispFileName,forceFile
Name):
myIndenter.RunIndentation(velocity,displacementExp,AccMax,Dur,expVol,CompleteForce[ActInt][0],forceActVal,POC,CompleteForce[ActInt]
[4],positioner,socketId,group,fileLoc,yInt,totalTrajTime,skin_Thickness)#CompleteForce[ActInt][5],trajectRoute)
current_Time = datetime.datetime.now()
logTime_String = current_Time.strftime("%H:%M:%S")

self.logWrite ("Force" ,logTime_String,displacementExp,str('{0:.0f}'.format(CompleteForce[ActInt][0]*1000)),velocity,",Durstr)
typeIndent = 'f'
complete = True
elif(self.type_mec_Indent.GetCurrentSelection()==0):
DispActInt = int(ActInt)
Dur = Dur+1

myIndenter.dispForce(velocity,holdVal,DispActInt,CompleteForce[DispActInt][4],socketId,positioner,group,POC,displacementExp)
complete = True
current_Time = datetime.datetime.now()
logTime_String = current_Time.strftime("%H:%M:%S")
typeIndent = 'd'
self.logWrite ("Disp. C" ,logTime_String,displacementExp,str('{0:.0f}'.format(CompleteForce[ActInt][0]*1000)),velocity,",Durstr)


else:
complete = False
if(complete):
dispFileName = ''
forceFileName = ''
Force = ''
Forceprint = CompleteForce[ActInt][0]*1000

if(len(presentImages)==0):
presentImages.append([[Forceprint,velocity,displacementExp-pointOfContact_Disp,False],[Forceprint,velocity,displacementExp-
pointOfContact_Disp,False]])
if(typeIndent == 'd'):
forceFileName = save_log_dialogFileName+ "disp_"+str(displacementExp-pointOfContact_Disp)+"vel_"+str(velocity)+'_forceTrace.png'
dispFileName = save_log_dialogFileName+"disp_"+str(displacementExp-pointOfContact_Disp)+"vel_"+str(velocity)+'_displacementTrace.png'
Force = 0
f0d=False

checkBox_ID_values,presentImages = self.forceViewArrayWrite(Forceprint,displacementExp-
pointOfContact_Disp,velocity,f0d,checkBox_ID_val)
presentImages = sorted(presentImages,key=lambda x:x[0])

forceID_fileName.append([checkBox_ID_val,dispFileName,forceFileName])

#self.forceViewTableWrite(checkBox_ID_values,presentImages)
```

```
else:
forceFileName = save_log_dialogFileName+
"Force_"+str('{0:.0f}'.format(CompleteForce[ActInt][0]*1000))+"vel_"+str(velocity)+'_forceTrace.png'
dispFileName =
save_log_dialogFileName+"Force_"+str('{0:.0f}'.format(CompleteForce[ActInt][0]*1000))+"vel_"+str(velocity)+'_displacementTrace.png'
Force = CompleteForce[ActInt][0]*1000
f0d=True
#if(5000+multInt<=checkBox_ID_force):
checkBox_ID_values,presentImages = self.forceViewArrayWrite(Force,displacementExp-pointOfContact_Disp,velocity,f0d,checkBox_ID_val)
presentImages = sorted(presentImages,key=lambda x:x[0])
#self.forceViewTableWrite(checkBox_ID_values,presentImages)
forceID_fileName.append([checkBox_ID_val,dispFileName,forceFileName])

myIndenter.output(Force,velocity,displacementExp-
pointOfContact_Disp,hold,pointOfContact_Disp,socketId,fileLoc,yInt,50,dispFileName,forceFileName,typeIndent)
self.Status.SetLabel("Ready")
self.Status.SetForegroundColour((101,218,80))
self.sizer_panel.Layout()
self.resetEventTimer()

global potent_Mode
potent_Mode = loadCell()
def fine_Disp_6_Command(self,event):
global Alone
Alone = False
global potent_Mode

global yInt
disp_Type = 'F'
ED = self.displacementControlFunct(disp_Type,yInt)
time.sleep(.1)
if(not(ED)):
potent_Mode = loadCell()
def view_ip(self,event):
global forceID_fileName
global checkBox_ID_values
global fineboxID_fileName
global cb_list_f
cb_Id_force = []
cb_Id_fine = []
for x in range(0,len(checkBox_ID_values)):
if(checkBox_ID_values[x][0].GetValue()):
cb_Id_force.append(checkBox_ID_values[x][0].GetId())
for x in range(0,len(cb_list_f)):
if(cb_list_f[x][0].GetValue()):
cb_Id_fine.append(cb_list_f[x][0].GetId())
forcePlot = []
dispPlot = []
forcePlot_fine = []
dispPlot_fine =[]
for x in range(0,len(forceID_fileName)):
for y in range(0,len(cb_Id_force)):
if(forceID_fileName[x][0]==cb_Id_force[y]):
dispPlot.append(forceID_fileName[x][1])
forcePlot.append(forceID_fileName[x][2])
for x in range(0,len(fineboxID_fileName)):
for y in range(0,len(cb_Id_fine)):
if(fineboxID_fileName[x][0]==cb_Id_fine[y]):
forcePlot_fine.append(fineboxID_fileName[x][1])
dispPlot_fine.append(fineboxID_fileName[x][2])
frame = PicturePopUp()
global plotScroll
global plotSizer
for i in range(0,len(dispPlot)):
dispPlotImage = wx.Image(fileLoc+'\\'+dispPlot[i], wx.BITMAP_TYPE_ANY).ConvertToBitmap()
forcePlotImage = wx.Image(fileLoc+'\\'+forcePlot[i], wx.BITMAP_TYPE_ANY).ConvertToBitmap()
self.plotPanel = wx.Panel( plotScroll, i+4000, wx.DefaultPosition, wx.DefaultSize, wx.TAB_TRAVERSAL )
self.plotPanel.SetMinSize( wx.Size(500,-1 ) )
plotPanel_sizer_pop = wx.FlexGridSizer( 2, 1, 0, 0 )
plotPanel_sizer_pop.SetFlexibleDirection( wx.VERTICAL )
```

```
plotPanel_sizer_pop.SetNonFlexibleGrowMode( wx.FLEX_GROWMODE_SPECIFIED )
plotPanel_sizer_pop.SetMinSize( wx.Size(500,-1 ) )

if(checkBox_ID_values[i][4]):

self.textHeader_pop = wx.StaticText( self.plotPanel, wx.ID_ANY, "Force Control: Force - "+ str(checkBox_ID_values[i][1])+ " Displ.- " +
str(checkBox_ID_values[i][3]) + " Vel.- " + str(checkBox_ID_values[i][2]), wx.DefaultPosition, wx.DefaultSize, 0 )
self.textHeader_pop.Wrap( -1 )
self.textHeader_pop.SetFont( wx.Font( 14, 74, 90, 92, False, "Calibri" ) )
self.textHeader_pop.SetMinSize( wx.Size(500,-1 ) )

else:
self.textHeader_pop = wx.StaticText( self.plotPanel, wx.ID_ANY, "Displacement Control: Force - "+ str(checkBox_ID_values[i][1])+ " Displ.- "
+ str(checkBox_ID_values[i][3]) + " Vel.- " + str(checkBox_ID_values[i][2]), wx.DefaultPosition, wx.DefaultSize, 0 )
self.textHeader_pop.Wrap( -1 )
self.textHeader_pop.SetFont( wx.Font( 14, 74, 90, 92, False, "Calibri" ) )
self.textHeader_pop.SetMinSize( wx.Size(500,-1 ) )
plotPanel_sizer_pop.Add( self.textHeader_pop, 0, wx.ALL, 5 )
plots = wx.GridSizer( 0, 2, 0, 0 )
plots.SetMinSize( wx.Size(500,100 ) )
dispbitmap = self.scale_bitmap(dispPlotImage, 200, 100)
self.displaceImage = wx.StaticBitmap(self.plotPanel, -1, dispbitmap)
plots.Add( self.displaceImage, 0, wx.TOP|wx.BOTTOM|wx.ALIGN_CENTER, 5 )
forcebitmap = self.scale_bitmap(forcePlotImage, 200, 100)

self.forceImage = wx.StaticBitmap(self.plotPanel, -1, forcebitmap)
plots.Add( self.forceImage, 0, wx.TOP|wx.BOTTOM|wx.ALIGN_CENTER, 5 )

plotPanel_sizer_pop.Add( plots, 1, wx.EXPAND, 5 )
self.plotPanel.SetSizer( plotPanel_sizer_pop )
self.plotPanel.Layout()
plotPanel_sizer_pop.Fit( self.plotPanel )
plotSizer.Add(self.plotPanel,0,wx.ALL|wx.ALIGN_CENTER_HORIZONTAL, 5 )
plotScroll.Layout()
plotSizer.Fit( plotScroll )
#print dispPlot
for i in range(0,len(dispPlot_fine)):
dispPlotImage = wx.Image(fileLoc+'\\'+dispPlot_fine[i], wx.BITMAP_TYPE_ANY).ConvertToBitmap()
forcePlotImage = wx.Image(fileLoc+'\\'+forcePlot_fine[i], wx.BITMAP_TYPE_ANY).ConvertToBitmap()
self.plotPanel = wx.Panel( plotScroll, i+4000, wx.DefaultPosition, wx.DefaultSize, wx.TAB_TRAVERSAL )
self.plotPanel.SetMinSize( wx.Size(500,-1 ) )
plotPanel_sizer_pop = wx.FlexGridSizer( 2, 1, 0, 0 )
plotPanel_sizer_pop.SetFlexibleDirection( wx.VERTICAL )
plotPanel_sizer_pop.SetNonFlexibleGrowMode( wx.FLEX_GROWMODE_SPECIFIED )
plotPanel_sizer_pop.SetMinSize( wx.Size(500,-1 ) )

self.textHeader_pop = wx.StaticText( self.plotPanel, wx.ID_ANY, "Displacement Control: Force - "+ str(cb_list_f[i][1])+ " Displ.- " +
str(cb_list_f[i][2]) + " Velocity- " + str(cb_list_f[i][3]), wx.DefaultPosition, wx.DefaultSize, 0 )
self.textHeader_pop.Wrap( -1 )
self.textHeader_pop.SetFont( wx.Font( 14, 74, 90, 92, False, "Calibri" ) )
self.textHeader_pop.SetMinSize( wx.Size(500,-1 ) )
plotPanel_sizer_pop.Add( self.textHeader_pop, 0, wx.ALL, 5 )
plots = wx.GridSizer( 0, 2, 0, 0 )
plots.SetMinSize( wx.Size(500,100 ) )
dispbitmap = self.scale_bitmap(dispPlotImage, 200, 100)
self.displaceImage = wx.StaticBitmap(self.plotPanel, -1, dispbitmap)
plots.Add( self.displaceImage, 0, wx.TOP|wx.BOTTOM|wx.ALIGN_CENTER, 5 )
forcebitmap = self.scale_bitmap(forcePlotImage, 200, 100)

self.forceImage = wx.StaticBitmap(self.plotPanel, -1, forcebitmap)
plots.Add( self.forceImage, 0, wx.TOP|wx.BOTTOM|wx.ALIGN_CENTER, 5 )

plotPanel_sizer_pop.Add( plots, 1, wx.EXPAND, 5 )
self.plotPanel.SetSizer( plotPanel_sizer_pop )
self.plotPanel.Layout()
plotPanel_sizer_pop.Fit( self.plotPanel )
plotSizer.Add(self.plotPanel,0,wx.ALL|wx.ALIGN_CENTER_HORIZONTAL, 5 )
plotScroll.Layout()
plotSizer.Fit( plotScroll )
frame.Show()
```

```
#########################################################################################################################################
########
#
# Log Commands
#
#########################################################################################################################################
#########
def Global_log_time_update_timerTick(self):

global_log_clock_time = datetime.datetime.now()-self.global_log_start_time

s = global_log_clock_time.seconds
hours, remainder = divmod(s,3600)
minutes, seconds = divmod(remainder,60)
self.time_log.SetLabel(' '+'%s:%s:%s'%(hours,minutes,seconds))
def local_log_time_update_timerTick(self):
local_log_clock_time = datetime.datetime.now()-self.local_log_start_time

s = local_log_clock_time.seconds
hours, remainder = divmod(s,3600)
minutes, seconds = divmod(remainder,60)
self.time_status.SetLabel(' '+'%s:%s:%s'%(hours,minutes,seconds))
def resetEventTimer(self):
self.local_log_start_time = datetime.datetime.now()
def update(self,event):
self.Global_log_time_update_timerTick()
self.local_log_time_update_timerTick()
def setPotentVal1(self,event):
global peakRange
global wid
peakRange = 1500
wid.Refresh()

def setPotentVal2(self,event):
global peakRange
global wid
peakRange = 1000
wid.Refresh()

def setPotentVal3(self,event):
global wid
global peakRange
peakRange =500
wid.Refresh()


def home(self,event):
global Alone
Alone = False
global potent_Mode
myIndenter.greeting(socketId,group)
potent_Mode = loadCell()

def highestPoint(self,event):
global Alone
Alone = False
global potent_Mode
myxps.GroupMoveAbsolute(socketId,group,[-20])
potent_Mode = loadCell()


def __del__( self ):
pass
if __name__=='__main__':
app = wx.App()
frame = UVAColumbiaInterface()
app.MainLoop()
```

Appendix D- Python code for the control algorithms

```
# -*- coding: utf-8 -*-
"""
Created on Mon Dec 29 15:04:02 2014

@author: Sean Gallahan
"""
# Instantiate the XPS class
import XPS_C8_drivers
import sys
import time
from scipy.optimize import minimize
from scipy.misc import derivative
from matplotlib import pyplot as plt
import numpy as np
import random
from scipy.optimize import curve_fit
import datetime
import os
import math
import ftplib


myxps = XPS_C8_drivers.XPS()

############################################### GLOBAL UNIVERSITY VARIABLE
###############################################
#### Either 'UVA' or COL'
global univ_Var
univ_Var= 'UVA'

global lopt_globe
lopt_globe = []

global trajectTimeAll
trajectTimeAll = []

global splits
splits=[]

global totalTrajTime
totalTrajTime = 0
#########################################################################################################################
#######



class mecIndent:
# Display error function : simplify error print out and closes socket

def upload(self,ftp,file):
ext = os.path.splitext(file)[1]
if ext in (".txt",".htm",".html",".trj"):
ftp.storlines("STOR " + file,open(file))
else:
ftp.storbinary("STOR " + file,open(file,"rb"),1024)




def myround(self,x, base=.005):
return float(base * round(float(x)/base))
def displayErrorAndClose (self,socketId, errorCode, APIName):
if (errorCode != -2) and (errorCode != -108):
[errorCode2, errorString] = myxps.ErrorStringGet(socketId, errorCode)
if (errorCode2 != 0):
print(APIName,' : ERROR ',str(errorCode))
else:
```

```
print(APIName,' : ',errorString)
else:
if (errorCode == -2):
print(APIName,' : TCP timeout')
if (errorCode == -108):
print(APIName,' : The TCP/IP connection was closed by an administrator')
myxps.TCP_CloseSocket(socketId)
return
########################################### To Change Calibration of Load Cell change here
#################################################################
def loadCellCalibrateForce(self,force,yInt):
multiplier = 0
if(univ_Var == 'UVA'):
multiplier = 7.5079
else:
multiplier = 6.022
return multiplier*force+yInt
def loadCellCalibrateVoltage(self,voltage,yInt):
multiplier = 0
if(univ_Var == 'UVA'):
multiplier = 7.5079
else:
multiplier = 6.022
#calibrationFunc.setPosition(str(multiplier)+"*x+"+str(yInt))
return (voltage+(-yInt))/multiplier

#turns of group, restarts group, and homes group. Must be done every time the progrma is activated
def greeting(self,socketId,group):
[errorCode, returnString] = myxps.GroupKill(socketId, group)
if (errorCode != 0):
self.displayErrorAndClose (socketId, errorCode, 'GroupKill')
sys.exit()
[errorCode, returnString] = myxps.GroupInitialize(socketId, group)
if (errorCode != 0):
self.displayErrorAndClose (socketId, errorCode, 'GroupInitialize')
sys.exit()
[errorCode, returnString] = myxps.GroupHomeSearch(socketId, group)
if (errorCode != 0):
self.displayErrorAndClose (socketId, errorCode, 'GroupHomeSearch')
sys.exit()
######### Assumes that the controller has already started from -.6 ###################################
def fineDisplacementControl(self,myxps,socketId,positioner,group, displacement, velocity, acceleration, hold_duration, cycle, retract,
extend,yInt):
[errorCode, returnString]=myxps.PositionerSGammaParametersSet(socketId,positioner,velocity,acceleration,.005,.05)
peakVolt = 0
voltage= myxps.GPIOAnalogGet(socketId,["GPIO2.ADC1"])[1]
if (cycle):

[errorCode, returnString] = myxps.GroupMoveRelative(socketId, group,[displacement])
peakVolt = myxps.GPIOAnalogGet(socketId,["GPIO2.ADC1"])[1]
#if(self.loadCellCalibrateVoltage(voltage,yInt)>2):
# myxps.GroupMoveRelative(socketId,group,[-displacement])
time.sleep(hold_duration)
[errorCode, returnString] = myxps.GroupMoveRelative(socketId, group, [-(displacement)])

elif (retract):
[errorCode, returnString] = myxps.GroupMoveRelative(socketId, group,[-displacement])
if(self.loadCellCalibrateVoltage(voltage,yInt)>2):
myxps.GroupMoveRelative(socketId,group,[-displacement])
else:
[errorCode, returnString] = myxps.GroupMoveRelative(socketId, group,[displacement])
voltage = myxps.GPIOAnalogGet(socketId,["GPIO2.ADC1"])[1]
if((self.loadCellCalibrateVoltage(voltage,yInt)>2)):
myxps.GroupMoveRelative(socketId,group,[-displacement])
return peakVolt

def setPointOfContact_1(self,myxps,socketId,positioner):
return myxps.GroupPositionCurrentGet(socketId,positioner,1)[1]
def Record_DispControl(self,socketId,positioner,holdTime,servoCycle):
if (univ_Var== 'COL'):
```

```
servo = .000125
else:
servo = .0001
numbPoints = (holdTime+2)/(servo*servoCycle)
numbPoints = int(numbPoints)
[errorCode,returnString]=myxps.GatheringReset(socketId)
if (errorCode != 0):
self.displayErrorAndClose (socketId, errorCode, 'GatheringReset')
sys.exit()
[errorCode, returnString]=myxps.GatheringConfigurationSet(socketId,[positioner+".CurrentPosition","GPIO2.ADC1"])
if (errorCode != 0):
self.displayErrorAndClose (socketId, errorCode, 'GatheringConfigurationSet')
sys.exit()

[errorCode, returnString]=myxps.EventExtendedConfigurationTriggerSet(socketId,[positioner+".SGamma.MotionStart"],["0"],["0"],["0"],["0"])
if (errorCode != 0):
self.displayErrorAndClose (socketId, errorCode, 'EventExtendedConfigurationTriggerSet')
sys.exit()


[errorCode,
returnString]=myxps.EventExtendedConfigurationActionSet(socketId,["GatheringRun"],[str(numbPoints)],[str(servoCycle)],["0"],["0"])
if (errorCode != 0):
self.displayErrorAndClose (socketId, errorCode, 'EventExtendedConfigurationActionSet')
sys.exit()
[errorCode, returnString]=myxps.EventExtendedStart(socketId)
if (errorCode != 0):
self.displayErrorAndClose (socketId, errorCode, 'Event Extended Start')
sys.exit()
def endRecord_DispControl(self,socketId,positioner,holdTime):
[errorCode, returnString] = myxps.GatheringStopAndSave(socketId)
if (errorCode != 0):
self.displayErrorAndClose (socketId, errorCode, 'GatheringStopandSave')
sys.exit()
#########################################################################################################################
#
# Graphical Output
#
#########################################################################################################################
##########################################################Graphical Output From the Motion
Controller##############################
def output(self,chosenForce,vel,disp,hold,POC,socketId,fileLoc,yInt,servo,dispFileName,forceFileName,typeIndent):
global univ_Var
CurrentNumber = myxps.GatheringCurrentNumberGet(socketId)[1]
iterations = CurrentNumber/500
LeftOver = CurrentNumber%500

Code = []
C=[]
starting= 0
count = 0
#Iterates over the 500 points that the program can read at a maximum time
endCount = 500
for y in range(iterations+1):
count = count + 1
if (y == (iterations) and (LeftOver !=0)):
C = [errorCode, returnString]=myxps.GatheringDataMultipleLinesGet(socketId,starting,LeftOver)
if (errorCode != 0):
self.displayErrorAndClose (socketId, errorCode, 'GatheringDataMultipleLinesGet')
sys.exit()
C.remove(0)
Code[0] = Code[0] + C[0]
elif(y<iterations):
if(y==0):
C = [errorCode, returnString]=myxps.GatheringDataMultipleLinesGet(socketId,starting,endCount)
if (errorCode != 0):
self.displayErrorAndClose (socketId, errorCode, 'GatheringDataMultipleLinesGet')
sys.exit()
C.remove(0)
Code.append(C[0])
```

```
starting = endCount + starting

else:
C = [errorCode, returnString]=myxps.GatheringDataMultipleLinesGet(socketId,starting,endCount)
if (errorCode != 0):
self.displayErrorAndClose (socketId, errorCode, 'GatheringDataMultipleLinesGet')
sys.exit()
C.remove(0)
Code[0] = Code[0] + C[0]
starting = endCount + starting
#Creates arrays that will moderate over gathered data
x = Code[0].count(';')
displacement = [None]*x
Force = [None]*x
Forcestr=[None]*x
displacementstr=[None]*x
dis= Code[0].index(';')
volt = -1
time = []
timestr=[]
#aboveRange =[]
#belowRange=[]

multiplier = 0
if(univ_Var == 'UVA'):
multiplier = .0001
else:
multiplier = .000125
for m in range(x):
z = m*multiplier*servo
time.append(z)
timestr.append(str(z))
FILE = open(fileLoc+ 'force ' + str(chosenForce)+ ' vel '+str('{0:.0f}'.format(vel))+'disp '+str('{0:.0f}'.format(disp))+'TimeEntry'+'.txt',"a")
FILE.write(timestr[m]+'\n')
#aboveRange.append(chosenForce*1.01)
#belowRange.append(chosenForce*.99)
if(m==x-1):

displacement[m] = float(Code[0][volt+1:dis])-(POC)
displacementstr[m] = str(float(Code[0][volt+1:dis])-(POC))
FILE = open(fileLoc+ 'force ' + str(chosenForce)+ ' vel '+str('{0:.0f}'.format(vel))+'disp '+str('{0:.0f}'.format(disp))+'DispEntry'+'.txt',"a")
FILE.write(displacementstr[m]+'\n')
volt=Code[0].index('\n',volt+1,len(Code[0]))

Force[m] = self.loadCellCalibrateVoltage((float(Code[0][dis+1:volt])),yInt)
Forcestr[m]=str(self.loadCellCalibrateVoltage((float(Code[0][dis+1:volt])),yInt))
FILE = open(fileLoc+ 'force ' + str(chosenForce)+ ' vel '+str('{0:.0f}'.format(vel))+'disp '+str('{0:.0f}'.format(disp))+'ForceEntry'+'.txt',"a")
FILE.write(Forcestr[m]+'\n')
else:
displacement[m] = float(Code[0][volt+1:dis])-(POC)
displacementstr[m] = str(float(Code[0][volt+1:dis])-(POC))
FILE = open(fileLoc+'force ' + str(chosenForce)+ ' vel '+str('{0:.0f}'.format(vel))+'disp '+str('{0:.0f}'.format(disp))+'DispEntry'+'.txt',"a")
FILE.write(displacementstr[m]+'\n')
volt=Code[0].index('\n',volt+1,len(Code[0]))
Force[m] = self.loadCellCalibrateVoltage((float(Code[0][dis+1:volt])),yInt)
Forcestr[m]=str(self.loadCellCalibrateVoltage((float(Code[0][dis+1:volt])),yInt))
FILE = open(fileLoc+ 'force ' + str(chosenForce)+ ' vel '+str('{0:.0f}'.format(vel))+'disp '+str('{0:.0f}'.format(disp))+'ForceEntry'+'.txt',"a")
FILE.write(Forcestr[m]+'\n')

dis= Code[0].index(';',dis+1)
#global trajectTimeAll
#trajectTime = []
#minDiff = 1
#closestVal = 0
#for x in range(0,len(trajectTimeAll)):
#print abs(chosenForce/1000 - trajectTimeAll[x][0])
# if(abs(chosenForce/1000 - trajectTimeAll[x][0])<minDiff):
# minDiff = abs(chosenForce/1000 - trajectTimeAll[x][0])
# closestVal = x
#trajectTime = trajectTimeAll[closestVal][1]
```

```
#for x in range(0,len(trajectTime)):
# trajectTime[x]=self.myround(trajectTime[x])
# trajectTime[x]=str('{0:.4f}'.format(trajectTime[x]))
# trajectTime[x]=float(trajectTime[x])
#trajectTimeVal = []
#trajectForceVal = []
#for x in range(0,len(trajectTime)):
# trajectTimeVal.append(float(trajectTime[x]))
# trajectForceVal.append(Force[int(float(trajectTimeVal[x])/.001)])
#xAxis = hold+2
plt.figure(figsize=(4.5,2.0))

plt.xlabel("Time (s)")
if(typeIndent == 'd'):
plt.title("Displacement: " + str(disp) + " Velocity: " + str(vel))
else:
plt.title("Displacement vs. Time")
plt.ylabel("Displacement (mm)")
plt.ylim(0,disp*1.2)
plt.plot(time,displacement,'b')
plt.savefig(fileLoc+'\\'+dispFileName)
plt.figure(figsize=(4.5,2.0))
if(typeIndent == 'd'):
plt.title("Force vs. Time")
else:
plt.title("Force: " + str(chosenForce) + " vel: " + str(vel))

plt.xlabel("Time (s)")
plt.ylabel("Force (mN)")
plt.plot(time,Force,'b')#trajectTimeVal,trajectForceVal,'g^')
plt.plot()
plt.savefig(fileLoc+'\\'+forceFileName)

##########################################################################################################################################
#####
#
# Force Control Calibration
#
##########################################################################################################################################
#####
def identifyLocation(self,maxForce, socketId,group,yInt,POC,positioner,vel):



[errorCode,returnString]=myxps.GatheringReset(socketId)
if(errorCode !=0):
self.displayErrorAndClose (socketId,errorCode,'GatheringReset')
sys.exit()

[errorCode, returnString]=myxps.GatheringConfigurationSet(socketId,[positioner+".CurrentPosition","GPIO2.ADC1"])
if (errorCode != 0):
self.displayErrorAndClose (socketId, errorCode, 'GatheringConfigurationSet')
sys.exit()

[errorCode, returnString]=myxps.EventExtendedConfigurationTriggerSet(socketId,[positioner+".Jog.MotionStart"],["0"],["0"],["0"],["0"])
if (errorCode != 0):
self.displayErrorAndClose (socketId, errorCode, 'EventExtendedConfigurationTriggerSet')
sys.exit()

########################################################## Change HERE
#############################################################################
[errorCode, returnString]=myxps.EventExtendedConfigurationActionSet(socketId,["GatheringRun"],["25000"],["5"],["0"],["0"])
if (errorCode != 0):
self.displayErrorAndClose (socketId, errorCode, 'EventExtendedConfigurationActionSet')
sys.exit()

[errorCode, returnString]=myxps.EventExtendedStart(socketId)
if (errorCode != 0):
self.displayErrorAndClose (socketId, errorCode, 'Event Extended Start')
sys.exit()
```

```
[errorCode,returnString]=myxps.GroupJogModeEnable(socketId,group)
if(errorCode !=0):
self.displayErrorAndClose (socketId, errorCode, 'GroupJogModeEnable')
sys.exit()

[errorCode, returnString] = myxps.GroupJogParametersSet(socketId, group,[vel],[80])
if (errorCode != 0):
self.displayErrorAndClose (socketId, errorCode, 'GroupJogParametersSet')
sys.exit()
voltage = myxps.GPIOAnalogGet(socketId,["GPIO2.ADC1"])
expectedVolt = self.loadCellCalibrateForce(maxForce,yInt)*1.01
while (voltage[1] <= expectedVolt):
if(self.loadCellCalibrateVoltage(voltage[1],yInt)>1.5):
myxps.GroupMoveAbsolute(socketId, group, [POC-.6])
voltage = myxps.GPIOAnalogGet(socketId,["GPIO2.ADC1"])
[errorCode,returnString]=myxps.GroupMoveAbort(socketId,group)
[errorCode,returnString]= myxps.GatheringStopAndSave(socketId)
if(errorCode!=0):
self.displayErrorAndClose(socketId,errorCode,'GatheringStopandSave')
sys.exit()
[errorCode, returnString] = myxps.GroupMoveAbsolute(socketId, group, [POC-.6])

linOut = self.linearOutput(POC,socketId,yInt)
return linOut
#Calibrate the Indenter for CRCNS Force (Multiple Displacements)
def linearOutput(self,POC,socketId,yInt):
CurrentNumber = myxps.GatheringCurrentNumberGet(socketId)[1]

iterations1 = CurrentNumber/500
LeftOver = CurrentNumber%500

Code = []
C=[]
starting1= 0
count1 = 0
#Iterates over the 500 points that the program can read at a maximum time
endCount = 500
for y in range(iterations1+1):
count1 = count1 + 1
if (y == (iterations1) and (LeftOver !=0)):
C = [errorCode, returnString]=myxps.GatheringDataMultipleLinesGet(socketId,starting1,LeftOver)
if (errorCode != 0):
self.displayErrorAndClose (socketId, errorCode, 'GatheringDataMultipleLinesGet')
sys.exit()
C.remove(0)
Code[0] = Code[0] + C[0]
elif(y<iterations1):
if(y==0):
C = [errorCode, returnString]=myxps.GatheringDataMultipleLinesGet(socketId,starting1,endCount)
if (errorCode != 0):
self.displayErrorAndClose (socketId, errorCode, 'GatheringDataMultipleLinesGet')
sys.exit()
C.remove(0)
Code.append(C[0])
starting1 = endCount + starting1

else:
C = [errorCode, returnString]=myxps.GatheringDataMultipleLinesGet(socketId,starting1,endCount)
if (errorCode != 0):
self.displayErrorAndClose (socketId, errorCode, 'GatheringDataMultipleLinesGet')
sys.exit()
C.remove(0)
Code[0] = Code[0] + C[0]
starting1 = endCount + starting1
#Creates arrays that will moderate over gathered data
x = Code[0].count(';')
Forcestr=[None]*x
```

```
displacementstr=[None]*x
identifyDisp = [None]*x
identifyForce = [None]*x
dis= Code[0].index(';')
volt = -1
time = []
timestr=[]
for m in range(x):
################## Need to modify for frequency of information taken#################
z = m*.0005
time.append(z)
timestr.append(str(z))
if(m==x-1):

identifyDisp[m] = float(Code[0][volt+1:dis])-(POC)
displacementstr[m] = str(float(Code[0][volt+1:dis])-(POC))
volt=Code[0].index('\n',volt+1,len(Code[0]))

identifyForce[m] = self.loadCellCalibrateVoltage((float(Code[0][dis+1:volt])),yInt)
Forcestr[m]=str(self.loadCellCalibrateVoltage((float(Code[0][dis+1:volt])),yInt))+","
else:
identifyDisp[m] = float(Code[0][volt+1:dis])-(POC)
displacementstr[m] = str(float(Code[0][volt+1:dis])-(POC))
volt=Code[0].index('\n',volt+1,len(Code[0]))
identifyForce[m] = self.loadCellCalibrateVoltage((float(Code[0][dis+1:volt])),yInt)
Forcestr[m]=str(self.loadCellCalibrateVoltage((float(Code[0][dis+1:volt])),yInt))+","
dis= Code[0].index(';',dis+1)


return[identifyDisp,identifyForce]
#plot(identifyForce)
#show()
def
CalibrateCRCNSForce(self,interval,partition,vel,CompleteForce,CompleteDisplacement,forceThresh,POC,highForce,saturation,compForceCount
er,failure,feedForward,yInt,socketId, positioner,group,trajectRoute,single,maxForce):
firstTime = 0
################################### Time Sleep ################################################
time.sleep(60)
###############################################################################################
linOut = self.identifyLocation(maxForce, socketId,group,yInt,POC,positioner,vel)
identifyForce = linOut[1]
identifyDisp = linOut[0]
firstRun = True
expectedDisplacement = []
forceCalibrate = []
val_Greater = False
if(single):
force = highForce
if(not(CompleteForce.__contains__(force))):
forceCalibrate.append((force))
#CompleteForce.append([force,vel])
numberOfCalibrate = len(forceCalibrate)
else:

firstContactForceCali = float(forceThresh)/1000
saturForceCali = float(saturation)
expo = [.010,.020,.040,.080,.160,.320,.640]
if(highForce<=saturForceCali):
if(partition):
for x in range(len(CompleteForce)):
if(not(CompleteForce[x][0] == (firstContactForceCali))):
#CompleteForce.append([(firstContactForceCali),vel])
forceCalibrate.append((firstContactForceCali))
elif((CompleteForce[x][0] == (firstContactForceCali) and not(CompleteForce[x][1]==vel))):
#CompleteForce.append([(firstContactForceCali),vel])
forceCalibrate.append((firstContactForceCali))
for z in range(0,len(expo)):
if((expo[z]>firstContactForceCali) or (expo[z] == firstContactForceCali)):
if((expo[z]<highForce)):
for m in range(len(CompleteForce)):
```

```
if(not(CompleteForce[m][0]==(expo[z]))):
forceCalibrate.append((expo[z]))
elif((CompleteForce[m][0] == (expo[z]) and not(CompleteForce[m][1]==vel))):
forceCalibrate.append((expo[z]))
for y in range(len(CompleteForce)):
if(not(CompleteForce[y][0] == (highForce))):
forceCalibrate.append(highForce)
elif((CompleteForce[y][0]==(highForce) and not(CompleteForce[y][1]==vel))):
forceCalibrate.append((expo[z]))
else:
if(not(interval==firstContactForceCali)):
if(not(CompleteForce.__contains__([firstContactForceCali,vel]))):
forceCalibrate.append((firstContactForceCali))
lowerLimit = interval
while(lowerLimit<highForce):
if(not(CompleteForce.__contains__([lowerLimit,vel]))):
forceCalibrate.append(lowerLimit)
lowerLimit = lowerLimit+interval
else:
lowerLimit = lowerLimit+interval
lowerLimit = lowerLimit - interval
if(not(CompleteForce.__contains__([highForce,vel]))):
forceCalibrate.append(highForce)
random.shuffle(forceCalibrate)
numberOfCalibrate = len(forceCalibrate)
else:
val_Greater = True
print "The Maximum Force is greater then the calibrated line, please lower maximum force"
if(not(val_Greater)):
for y in range(0,numberOfCalibrate):
############################SLEEEEEEEEP##################
if((forceCalibrate[y])>.001):
time.sleep(60)
############################################################ def
LocationFindCRCNSForce(self,Expectedvoltage,expectedDisplacement,count,vel,socketId,positioner,identifyForce,group,
POC,identifyDisp,feed_forward,firstTime,yInt,trajectRoute ):
######
LFCF=self.LocationFindCRCNSForce(self.loadCellCalibrateForce(forceCalibrate[y],yInt),expectedDisplacement,y,vel,socketId,positioner,identi
fyForce, group, POC, identifyDisp, feedForward, firstTime, yInt,trajectRoute,firstRun)
############################
expectedDisplacement = LFCF[3]#### return [failure,feedForward,firstTime] ####################################
failure = LFCF[0]
feedForward = LFCF[1]
totalTrajTime = LFCF[5]
CompleteForce.append([forceCalibrate[y],expectedDisplacement,vel,failure,feedForward,totalTrajTime])
firstRun = False
#expectedDisplacement = LFCF[3]
splits = LFCF[4]
#for m in range(0,len(expectedDisplacement)):
# CompleteDisplacement.append(expectedDisplacement[m])
locationArray = []
for x in range(0,len(CompleteForce)):
if(CompleteForce[x][3]):
locationArray.append(x)
countUp = 0
for x in range(0,len(locationArray)):
del(CompleteForce[locationArray[x]-countUp])
countUp = countUp+1
CompleteForce = sorted(CompleteForce,key=lambda x:x[0])
#CompleteDisplacement=sorted(CompleteDisplacement,key=lambda x:x[1])
return [CompleteForce,compForceCounter,splits]
def find_nearest(self,array,value):
idx = (np.abs(array-value)).argmin()
return idx
def LocationFindCRCNSForce(self,Expectedvoltage,expectedDisplacement,count,vel,socketId,positioner,identifyForceVal,group,
POC,identifyDispVal,feed_forward,firstTime,yInt,trajectRoute,firstRun):
counter_point = ""
#sleepTime = 0
#if(vel == 1):
sleepTime = 1.5
```

```
#else:
# sleepTime = 1
if(univ_Var == 'UVA'):
counter_point= "10"
else:
counter_point = "8"
expectedValue = 0
breaker=True

expectedValue = 0

for x in range(len(identifyForceVal)):
if(identifyForceVal[x]<= self.loadCellCalibrateVoltage(Expectedvoltage,yInt) and identifyDispVal[x]>0):
expectedValue = identifyDispVal[x]

identifyDisp=[]
identifyForce=[]
identifyDispVal = np.array(identifyDispVal)

indexVal = self.find_nearest(identifyDispVal,0)
identifyDisp.append(identifyDispVal[indexVal:len(identifyDispVal)])
identifyForce.append(identifyForceVal[indexVal:len(identifyForceVal)])
forceZero = identifyForce[0][0]

#print identifyForce
for i in range(len(identifyDisp[0])):
identifyForce[0][i]=identifyForce[0][i]-forceZero
identifyDisp = np.array(identifyDisp)
identifyForce = np.array(identifyForce)
expectedValue = expectedValue+POC
relDisp = expectedValue - POC +.6
expectedDisplacement = expectedValue


chooseFile = 'Calib_force '+str(self.loadCellCalibrateVoltage(Expectedvoltage,yInt))+ ' vel ' +str('{0:.0f}'.format(vel))+'.trj'

leg1_velfinal = vel
leg1_velstart = 0
leg1_time = float(leg1_velfinal-leg1_velstart)/80
leg1_disp = float(leg1_velfinal + leg1_velstart)/2*leg1_time


#if(vel ==1):
# leg3_velfinal = .0068*np.log(self.loadCellCalibrateVoltage(Expectedvoltage,yInt))+.056
#elif(vel==2):
# leg3_velfinal=.0053*np.log(self.loadCellCalibrateVoltage(Expectedvoltage,yInt))+.0484
leg3_velfinal = 0
leg3_velstart = vel
leg3_time = float(leg3_velfinal-leg3_velstart)/(-80)
leg3_disp = float(leg3_velfinal + leg3_velstart)/2*leg3_time


leg2_velfinal = vel
leg2_disp = float(relDisp)-leg1_disp-leg3_disp
leg2_time = leg2_disp/vel

FILECal = open(chooseFile,"a")
FILECal.write(str('{0:.6f}'.format(leg1_time))+',' + str('{0:.6f}'.format(leg1_disp))+ ','+str('{0:.6f}'.format(leg1_velfinal))+'\n')
FILECal.write(str('{0:.6f}'.format(leg2_time))+',' + str('{0:.6f}'.format(leg2_disp))+ ','+str('{0:.6f}'.format(leg2_velfinal))+'\n')
FILECal.write(str('{0:.6f}'.format(leg3_time))+',' + str('{0:.6f}'.format(leg3_disp))+ ','+str('{0:.6f}'.format(leg3_velfinal))+'\n')
#FILECal.write(str('{0:.6f}'.format(.020))+',' + str('{0:.6f}'.format(.0004))+ ','+str('{0:.6f}'.format(0))+'\n')
FILECal.close()
ftps = ftplib.FTP('192.168.0.254','Administrator','Administrator')
ftps.cwd(trajectRoute)
self.upload(ftps,chooseFile)
ftps.quit()

os.remove(chooseFile)
[errorCode, returnString]=myxps.PositionerSGammaParametersSet(socketId,positioner,vel,80,.005,.05)
```

```
[errorCode,returnString]=myxps.GatheringReset(socketId)
if(errorCode !=0):
self.displayErrorAndClose (socketId,errorCode,'GatheringReset')
sys.exit()

[errorCode, returnString]=myxps.GatheringConfigurationSet(socketId,[positioner+".CurrentPosition","GPIO2.ADC1"])
if (errorCode != 0):
self.displayErrorAndClose (socketId, errorCode, 'GatheringConfigurationSet')
sys.exit()

[errorCode, returnString]=myxps.EventExtendedConfigurationTriggerSet(socketId,[group+".PVT.TrajectoryStart"],["0"],["0"],["0"],["0"])
#[errorCode, returnString]=myxps.EventExtendedConfigurationTriggerSet(socketId,[positioner+".SGamma.MotionStart"],["0"],["0"],["0"],["0"])

if (errorCode != 0):
self.displayErrorAndClose (socketId, errorCode, 'EventExtendedConfigurationTriggerSet')
sys.exit()

if(self.loadCellCalibrateVoltage(Expectedvoltage,yInt)<=.015):
[errorCode, returnString]=myxps.EventExtendedConfigurationActionSet(socketId,["GatheringRun"],["10000"],[counter_point],["0"],["0"])
if (errorCode != 0):
self.displayErrorAndClose (socketId, errorCode, 'EventExtendedConfigurationActionSet')
sys.exit()
else:
[errorCode, returnString]=myxps.EventExtendedConfigurationActionSet(socketId,["GatheringRun"],["10000"],[counter_point],["0"],["0"])
if (errorCode != 0):
self.displayErrorAndClose (socketId, errorCode, 'EventExtendedConfigurationActionSet')
sys.exit()
[errorCode, returnString]=myxps.EventExtendedStart(socketId)
if (errorCode != 0):
self.displayErrorAndClose (socketId, errorCode, 'Event Extended Start')
sys.exit()

[errorCode,returnString]=myxps.MultipleAxesPVTExecution(socketId,group,chooseFile,1)

#[errorCode, returnString] = myxps.GroupMoveRelative(socketId, group, [relDisp])
time.sleep(sleepTime)

[errorCode,returnString]= myxps.GatheringStopAndSave(socketId)
if(errorCode!=0):
self.displayErrorAndClose(socketId,errorCode,'GatheringStopandSave')
sys.exit()
[errorCode, returnString] = myxps.GroupMoveAbsolute(socketId, group, [POC-.6])
################################################## outputCal = [zerVal,
dRamp,fRamp,holdDisp,holdForce,holdTime,timeRamp,dispRamp,ForceRamp] ##################################
outputCal = self.outputCalib(POC,breaker,expectedValue,self.loadCellCalibrateVoltage(Expectedvoltage,yInt),False,1,socketId,yInt,vel)
firstTime = 0
zerVal = outputCal[0]
failure = False
holdTime = outputCal[5]
holdForce = outputCal[4]
fRamp = outputCal[2]
dRamp = outputCal[1]
timeRamp = outputCal[6]
dispRamp =outputCal[7]
assumeVal = [.008,3,.013,4,.35,.02]

calcChangeVal = self.calculateChangeVal(zerVal, firstTime, failure,
breaker,expectedValue,Expectedvoltage,holdTime,holdForce,timeRamp,dispRamp,fRamp,dRamp,feed_forward,POC,yInt,vel,socketId,relDisp,ft
ps,trajectRoute,group,identifyForce,identifyDisp,assumeVal)
############################## calcChangeVal = [failure, feedForward,firstTime]
##############################################
failureVal = calcChangeVal[0]
feedForward = calcChangeVal[1]
firstTime = calcChangeVal[2]
splits = calcChangeVal[3]
totalTrajTime = calcChangeVal[4]
return [failureVal,feedForward,firstTime,expectedDisplacement,splits,totalTrajTime]
def outputCalib(self,POC, breakAp,peakDisp,expectedForce,disp, modValue,socketId,yInt,vel):
zerVal = False
dRamp =[]
```

```
fRamp = []
holdForce=[]
holdDisp=[]
holdTime=[]
allForce = []
allTime = []

CurrentNumber = myxps.GatheringCurrentNumberGet(socketId)[1]

iterations1 = CurrentNumber/500
LeftOver = CurrentNumber%500

Code = []
C=[]
starting1= 0
count1 = 0
#Iterates over the 500 points that the program can read at a maximum time
endCount = 500
for y in range(iterations1+1):
count1 = count1 + 1
if (y == (iterations1) and (LeftOver !=0)):
C = [errorCode, returnString]=myxps.GatheringDataMultipleLinesGet(socketId,starting1,LeftOver)
if (errorCode != 0):
self.displayErrorAndClose (socketId, errorCode, 'GatheringDataMultipleLinesGet')
sys.exit()
C.remove(0)
Code[0] = Code[0] + C[0]
elif(y<iterations1):
if(y==0):
C = [errorCode, returnString]=myxps.GatheringDataMultipleLinesGet(socketId,starting1,endCount)
if (errorCode != 0):
self.displayErrorAndClose (socketId, errorCode, 'GatheringDataMultipleLinesGet')
sys.exit()
C.remove(0)
Code.append(C[0])
starting1 = endCount + starting1

else:
C = [errorCode, returnString]=myxps.GatheringDataMultipleLinesGet(socketId,starting1,endCount)
if (errorCode != 0):
self.displayErrorAndClose (socketId, errorCode, 'GatheringDataMultipleLinesGet')
sys.exit()
C.remove(0)
Code[0] = Code[0] + C[0]
starting1 = endCount + starting1
#Creates arrays that will moderate over gathered data
x = Code[0].count(';')
displacement = [None]*x
Force = [None]*x
Forcestr=[None]*x
displacementstr=[None]*x
dis= Code[0].index(';')
volt = -1
time = []
timestr=[]

for m in range(x):
################## Need to modify for frequency of information taken#################
z = m*.001
time.append(z)
timestr.append(str(z))
if(m==x-1):

displacement[m] = float("%.3f"%(float(Code[0][volt+1:dis])-(POC)))
displacementstr[m] = str(float(Code[0][volt+1:dis])-(POC))
volt=Code[0].index('\n',volt+1,len(Code[0]))

Force[m] = self.loadCellCalibrateVoltage((float(Code[0][dis+1:volt])),yInt)
Forcestr[m]=str(self.loadCellCalibrateVoltage((float(Code[0][dis+1:volt])),yInt))+","
else:
```

```
displacement[m] = float("%.3f"%(float(Code[0][volt+1:dis])-(POC)))
displacementstr[m] = str(float(Code[0][volt+1:dis])-(POC))
volt=Code[0].index('\n',volt+1,len(Code[0]))
Force[m] = self.loadCellCalibrateVoltage((float(Code[0][dis+1:volt])),yInt)
Forcestr[m]=str(self.loadCellCalibrateVoltage((float(Code[0][dis+1:volt])),yInt))+","
dis= Code[0].index(';',dis+1)


forceHolder = []

for x in range(len(Force)):
if(displacement[x]>0):
forceHolder.append(Force[x])
else:
forceHolder.append(0)
peakerDisp = 1
location = 0
if(disp):
valDisp = float("%.2f"%(peakDisp-.6))
highDisp = displacement.index((valDisp))

dispRamp = displacement[0:highDisp]

ForceRamp = Force[0:highDisp]

timeRamp = time[0:highDisp]

for x in range(500):
allForce.append(Force[highDisp+x])
allTime.append(time[highDisp+x])
if(x%modValue==0):
holdDisp.append(displacement[highDisp+x])
holdForce.append((Force[highDisp+x]))
holdTime.append(time[highDisp+x])
counter = 0
holdDisp = np.array(holdDisp)
holdForce = np.array(holdForce)
holdTime = np.array(holdTime)
if(breakAp):
for i in range(len(dispRamp)):
if(dispRamp[i]>0 and ForceRamp[i]>float(expectedForce)/2 and counter < 1):
location = i
counter = counter+1

dRamp.append(dispRamp[location:len(dispRamp)])
fRamp.append(ForceRamp[location:len(dispRamp)])
dRamp = dRamp[0]
fRamp = fRamp[0]


else:
plt.figure(1)
plt.subplot(211)
plt.plot(time,displacement,'b')
plt.ylim(0,.3)
plt.subplot(212)
plt.plot(time,Force,'b')
plt.ylim(0,.075)
plt.plot()
#if(vel == 1):
totalCells = 1500
#else:
# totalCells = 1000
highDisp = len(time) - totalCells
if(not(peakerDisp==0)):

dispRamp = displacement[0:highDisp]
counter = 0
ForceRamp = Force[0:highDisp]
for i in range(len(dispRamp)):
```

```
if(dispRamp[i]>0 and counter<1 ): #and ForceRamp[i]>.002):
location = i
counter = counter+1

dispRamp = dispRamp[location:highDisp]

ForceRamp = Force[location:highDisp]

timeRamp = time[location:highDisp]

#dispRamp = np.array(dispRamp)
#ForceRamp = np.array(ForceRamp)
#timeRamp = np.array(timeRamp)
for x in range(len(displacement[highDisp:len(displacement)])):
allForce.append(Force[highDisp+x])
allTime.append(time[highDisp+x])
if(x%modValue==0):
holdDisp.append(displacement[highDisp+x])
holdForce.append((Force[highDisp+x]))
holdTime.append(time[highDisp+x])
counter = 0
holdDisp = np.array(holdDisp)
holdForce = np.array(holdForce)
holdTime = np.array(holdTime)
if(breakAp):
dRamp.append(dispRamp)
fRamp.append(ForceRamp)
dRamp = dRamp[0]
fRamp = fRamp[0]
dRamp = np.array(dRamp)
fRamp = np.array(fRamp)
else:
zerVal = True
return [zerVal, dRamp,fRamp,holdDisp,holdForce,holdTime,timeRamp,dispRamp,ForceRamp]

def triple_exp(self,params, x):
y = params[0] * np.exp(x * params[1])\
+ params[2] * np.exp(x * params[3])\
+ params[4] * np.exp(x * params[5])
return y


def rise_exp(self,params, x):
y = params[0] * np.exp(x * params[1]) + params[2]
return y


def get_r2(self,params, func, x, y, sign=1.):
ynew = func(params, x)
sstot = ((y - y.mean()) ** 2).sum()
ssres = ((y - ynew) ** 2).sum()
r2 = 1 - ssres / sstot
return r2 * sign #def func(parameters,x):
# return parameters[0]*np.exp(x*parameters[1])+parameters[3]
#def r2(x,y, parameters, sign=1.):
# newy = func(parameters, x)
# sst = y.var() * y.shape()
# sse =
# return sign*
def calculateChangeVal(self,zerVal, firstTime, failure,
breaker,expectedValue,Expectedvoltage,holdTime,holdForce,timeRamp,dispRamp,fRamp,dRamp,feedForward,POC,yInt,vel,socketId,relDisp,ftp
s,trajectRoute,group,identifyForce, identifyDisp,assumeVal):
global lopt_globe
global splits
timeMoved = 0
global failureVal
failureVal = False
feedForward = []
chooseFile = 'force '+str(self.loadCellCalibrateVoltage(Expectedvoltage,yInt))+' vel ' +str('{0:.0f}'.format(vel))+'.trj'
totalTrajTime = 0
```

```
chooseFileDisp = 'disp '+str(expectedValue)+' vel ' +str('{0:.0f}'.format(vel))+'.trj'
if(not(failure)):
try:
os.remove(chooseFile)
except OSError:
pass
try:
os.remove(chooseFileDisp)
except OSError:
pass
if(not(zerVal)):
#hyper_displ = identifyDisp[0]
#hyper_force = identifyForce[0]
#hyper_displ = np.array(hyper_displ)
#hyper_force = np.array(hyper_force)
hyper_displ = dRamp
hyper_force = fRamp
hyper_displ = np.array(hyper_displ)
hyper_force = np.array(hyper_force)
#x0 = np.array([hyper_force.mean(), 1., -hyper_force.mean()])
#bounds = ((0, None), (0, None), (None, None))
#res = minimize(
# self.get_r2, x0, args=(self.rise_exp, hyper_displ, hyper_force, -1),
# method='SLSQP', bounds=bounds)
#hyper_param = res.x
#lopt = hyper_param
#lopt_2,lcov = curve_fit(self.fitfuncPosExpo,identifyDisp[0],identifyForce[0],p0=(.007,13.9,-.003))
#lopt,lcov = curve_fit(self.fitfuncPosExpo,identifyDisp[0],identifyForce[0],p0=(hyper_force.mean(),13.9,-hyper_force.mean()))
lopt,lcov = curve_fit(self.fitfuncPosExpo,dRamp,fRamp,p0=(hyper_force.mean(),13.9,-hyper_force.mean()),maxfev=3000)
lopt_globe.append([expectedValue,vel,lopt[0],lopt[1],lopt[2]])
yD = []
#yD_2=[]
for y in range(0,len(dRamp)):
#yD_2.append(self.fitfuncPosExpo(identifyDisp[0][y],lopt_2[0],lopt_2[1],lopt_2[2]))
#yD.append(self.fitfuncPosExpo(dRamp,lopt[0],lopt[1],lopt[2]))
yD.append(lopt[0]*np.exp(dRamp[y]*lopt[1])+lopt[2])

plt.figure(2)
plt.plot(dRamp,fRamp,'b')
plt.plot(dRamp,yD,'r',linewidth = 2.0)
plt.ylim(0,.075)
plt.show()
plt.figure(3)
plt.subplot(211)
plt.plot(timeRamp,fRamp,'b')
plt.ylim(0,.075)
plt.subplot(212)
plt.plot(timeRamp,dRamp,'b')
plt.ylim(0,.7)
plt.show()
#plt.plot(identifyDisp[0],yD_2,'g')

#plt.show()
#except RuntimeError:
# lopt = [0,0,0]
# if(not(firstRun)):
# for x in range(len(lopt_globe)):
# if(lopt_globe[x][1]==vel):
#
# lopt[0] = lopt_globe[0][2]
# lopt[1] = lopt_globe[0][3]
# lopt[2] = lopt_globe[0][4]
# else:
# lopt,lcov = curve_fit(self.fitfuncPosExpo,identifyDisp[0],identifyForce[0],p0=(.007,13.9,-.003))

#print "runtime lopt:" + str(lopt)
# lopt_globe.append([expectedValue,vel,lopt[0],lopt[1],lopt[2]])
# yD=[]
# for y in range(0,len(dRamp)):
#yD.append(self.fitfuncPosExpo(identifyDisp[0][y],lopt[0],lopt[1],lopt[2]))
```

```
# yD.append(self.fitfuncPosExpo(dRamp,lopt[0],lopt[1],lopt[2]))
try:
offsetForce = max(holdForce)
diffForce = self.loadCellCalibrateVoltage(Expectedvoltage,yInt)-offsetForce
for x in range(len(holdForce)):
holdForce[x] = holdForce[x]+diffForce
holdTime = (holdTime-holdTime[0])
holdForce = holdForce
if(self.loadCellCalibrateVoltage(Expectedvoltage,yInt)<=.02):
startingPoint =
[self.loadCellCalibrateVoltage(Expectedvoltage,yInt)*.85,12,self.loadCellCalibrateVoltage(Expectedvoltage,yInt)*.1,1.5,self.loadCellCalibrateV
oltage(Expectedvoltage,yInt)*.05,.02]
else:
startingPoint =
[self.loadCellCalibrateVoltage(Expectedvoltage,yInt)*.85,16,self.loadCellCalibrateVoltage(Expectedvoltage,yInt)*.1,4,self.loadCellCalibrateVol
tage(Expectedvoltage,yInt)*.05,.012]
#x0 = np.array([1, -1, 1, -1, 1, -1]) * holdForce.max()
#x0=np.array([self.loadCellCalibrateVoltage(Expectedvoltage,yInt)*.85,16,self.loadCellCalibrateVoltage(Expectedvoltage,yInt)*.1,4,self.loadCe
llCalibrateVoltage(Expectedvoltage,yInt)*.05,.012])
#bounds = ((0, None), (None, 0),
# (0, None), (None, 0),
# (0, None), (None, 0))
#res = minimize(
# self.get_r2, x0, args=(self.triple_exp, holdTime, holdForce, -1),
# method='SLSQP', bounds=bounds)
#triple_exp_params = res.x
#print triple_exp_params
#tripopt = triple_exp_params
altTripFit = []
tripopt,tripcov = curve_fit(self.fitfuncTrip,holdTime,holdForce,p0 =
(startingPoint[0],startingPoint[1],startingPoint[2],startingPoint[3],startingPoint[4],startingPoint[5]), maxfev = 3000)
#print tripopt
#tripFit =[]
tripFit = []
negTripFit = []
for y in range(0,len(holdTime)):
tripFit.append(self.fitfuncTrip(holdTime[y],tripopt[0],tripopt[1],tripopt[2],tripopt[3],tripopt[4],tripopt[5]))
#altTripFit.append(triple_exp_params[0]*math.exp(holdTime[y]*-triple_exp_params[1])+triple_exp_params[2]*math.exp(holdTime[y]*-
triple_exp_params[3])+triple_exp_params[4]*math.exp(holdTime[y]*-triple_exp_params[5]))
negTripFit.append((self.fitfuncTrip(holdTime[y],-tripopt[0],tripopt[1],-tripopt[2],tripopt[3],-tripopt[4],tripopt[5])))
tripFit[0]
negTripFit[0]
difference = tripFit[0]-negTripFit[0]
for n in range(0,len(holdTime)):
negTripFit[n] = negTripFit[n]+difference
#plt.plot(holdTime,holdForce)
#plt.plot(holdTime,tripFit,'r')
#plt.show()
#plt.plot(holdTime,altTripFit,'g')
tripoptVals = []
tripoptVals.append(tripopt[1])
tripoptVals.append(tripopt[3])
tripoptVals.append(tripopt[5])
tripoptVals = sorted(tripoptVals)
splits.append([expectedValue,vel,1/tripoptVals[1]/.001])
firstMove = 1/tripoptVals[2]/.001
timeStoppers = 1/tripoptVals[1]/.001
timeStop = timeStoppers#/.001
negyaj =[]
for x in range(0,len(tripFit)):
negyaj.append((math.log((tripFit[x]-lopt[2])/(lopt[0]))/lopt[1]))
#x0 = np.array([assumeVal[0],tripopt[1],assumeVal[2],tripopt[3],assumeVal[4],tripopt[5]])
#bounds = ((0, None), (None, 0),
# (0, None), (None, 0),
# (0, None), (None, 0))
#res = minimize(
# self.get_r2, x0, args=(self.triple_exp, holdTime, holdForce, -1),
# method='SLSQP', bounds=bounds)
#triple_exp_params.append(res.x)
#dopt = triple_exp_params
```

```
#dopt,dcov = curve_fit(self.fitfuncTrip,holdTime,negyaj,p0 = (assumeVal[0],tripopt[1],assumeVal[2],tripopt[3],assumeVal[4],tripopt[5])
,maxfev=3000)
dispHold = []
dispHold1=[]
zerVal = 0
for x in range(0,len(holdTime)):
#dispHold1.append(math.log((tripopt[0]*math.exp(-tripopt[1]*holdTime[x])+tripopt[2]*math.exp(-
tripopt[3]*holdTime[x])+tripopt[4]*math.exp(-tripopt[5]*holdTime[x])-lopt[2])/lopt[0])/lopt[1]) #self.fitfuncTrip(holdTime[x],-dopt[0],dopt[1],-
dopt[2],dopt[3],-dopt[4],dopt[5]))#offsetVal))#-dopt[2],dopt[3],offsetVal))

dispHold1.append(math.log((tripopt[0]*math.exp(-tripopt[1]*holdTime[x])+tripopt[2]*math.exp(-tripopt[3]*holdTime[x])+tripopt[4]*math.exp(-
tripopt[5]*holdTime[x]))/lopt[0]+1)/lopt[1]) #self.fitfuncTrip(holdTime[x],-dopt[0],dopt[1],-dopt[2],dopt[3],-dopt[4],dopt[5]))#offsetVal))#-
dopt[2],dopt[3],offsetVal))
if(x==0):
zerVal = dispHold1[0]
dispHold.append(-dispHold1[x]+zerVal+zerVal)
plt.figure(4)
plt.plot(holdTime,dispHold)
#plt.plot(holdTime,dispHold1)
plt.show()
def f(x):
return -math.log((tripopt[0]*math.exp(-tripopt[1]*x)+tripopt[2]*math.exp(-tripopt[3]*x)+tripopt[4]*math.exp(-
tripopt[5]*x))/lopt[0]+1)/lopt[1]+zerVal+zerVal
#return -math.log((tripopt[0]*math.exp(-tripopt[1]*x)+tripopt[2]*math.exp(-tripopt[3]*x)+tripopt[4]*math.exp(-tripopt[5]*x)-
lopt[2])/lopt[0])/lopt[1]+zerVal+zerVal

#print str(derivative(f,holdTime[0],dx = 1e-3))
#print str(derivative(f,holdTime[2],dx = 1e-3))

#velocityVal = [-((-dopt[0]*(dopt[1]))*math.exp(((holdTime[0])*(-dopt[1])))+(-dopt[2]*(dopt[3]))*math.exp(holdTime[0]*-dopt[3])+(-
dopt[4]*(dopt[5]))*math.exp(holdTime[0]*-dopt[5]))]
#velocityVal =
[1/(lopt[1]*(tripopt[0][0]*math.exp(tripopt[0][1]*holdTime[0])+tripopt[0][2]*math.exp(tripopt[0][3]*holdTime[0])+tripopt[0][4]*math.exp(tripo
pt[0][5]*holdTime[0])+lopt[2]))]
velocityVal = [derivative(f,holdTime[0],dx = 1e-3)]
#holdTime = holdTime[0:timeStop]
totalMove = []
for x in xrange(1,len(holdTime)):
timeMoved = timeMoved + .001
#totalMove.append(-dopt[0]*math.exp(((timeMoved)*(dopt[1])))  -dopt[2]*math.exp((timeMoved*(dopt[3])))-
dopt[4]*math.exp((timeMoved*(dopt[5]))))
totalMove.append(dispHold[x])
#velocityVal.append(-((-dopt[0]*dopt[1])*math.exp((timeMoved)*(-dopt[1]))+(-dopt[2]*(dopt[3]))*math.exp(timeMoved*-dopt[3])+(-
dopt[4]*(dopt[5]))*math.exp(timeMoved*-dopt[5])))
#velocityVal.append(1/(lopt[1]*(tripopt[0][0]*math.exp(tripopt[0][1]*holdTime[x])+tripopt[0][2]*math.exp(tripopt[0][3]*holdTime[x])+tripopt[
0][4]*math.exp(tripopt[0][5]*holdTime[x])+lopt[2])))
velocityVal.append(derivative(f,holdTime[x],dx = 1e-3))
dispMove = []
timeMove = []
velMove = [0]*len(dispMove)
num = np.linspace(1,10,num = 10)
movementArray = []
velArray = []
for x in range(1,len(totalMove)):
movementArray.append(totalMove[x]-totalMove[x-1])
velArray.append(velocityVal[x])
for x in range(0,len(num)):
if(x == 0):
#
mover = totalMove[int(num[x])]-totalMove[0]
timeDiff = num[x]
else:
mover = totalMove[int(num[x])-1]-totalMove[int(num[x])-int(num[x-1])-1]
timeDiff = num[x]-num[x-1]
if(mover<0):
mover = -mover
dispMove.append(mover)
timeMove.append(timeDiff/1000)
```

```
velMove.append(velocityVal[int(num[x])-1])
leg1_velfinal = vel
leg1_velstart = 0
leg1_time = float(leg1_velfinal-leg1_velstart)/80
leg1_disp = float(leg1_velfinal + leg1_velstart)/2*leg1_time
leg3_velfinal = velocityVal[0]
leg3_velstart = vel
################ DISP MODIFIER ############################
leg3_time = float(leg3_velfinal-leg3_velstart)/-80
leg3_disp = float(leg3_velfinal + leg3_velstart)/2*leg3_time
leg2_velfinal = vel
mult_ratio = 1
leg2_disp = float(relDisp*mult_ratio)-leg1_disp-leg3_disp
#if(vel==2):
# leg2_disp = leg2_disp*1.004
#
#elif(vel==3):
# leg2_disp = leg2_disp*1.006
#elif(vel==4):
# leg2_disp = leg2_disp*1.008
#elif(vel==5):
# leg2_disp = leg2_disp*1.01

leg2_time = leg2_disp/vel
global trajectTimeAll
trajectTime = []
trajectTime.append(0)
trajectTime.append(leg1_time)
trajectTime.append(leg1_time+leg2_time)
for x in range(0,len(num)):
if(x==0):
trajectTime.append(leg1_time+leg2_time+leg3_time)
else:
trajectTime.append(leg1_time+leg2_time+leg3_time+num[x]/1000)
trajectTimeAll.append([self.loadCellCalibrateVoltage(Expectedvoltage,yInt),trajectTime])
############################## Ramp Time Graph Value ###################################
global rampTimeGraph
rampTimeGraph= leg1_time+leg2_time+leg3_time
#print rampTimeGraph
#############################################################################################
FILE = open(chooseFile,"a")
FILE.write(str('{0:.6f}'.format(leg1_time))+',' + str('{0:.6f}'.format(leg1_disp))+ ','+str('{0:.6f}'.format(leg1_velfinal))+'\n')
FILE.write(str('{0:.6f}'.format(leg2_time))+',' + str('{0:.6f}'.format(leg2_disp))+ ','+str('{0:.6f}'.format(leg2_velfinal))+'\n')
FILE.write(str('{0:.6f}'.format(leg3_time))+',' + str('{0:.6f}'.format(leg3_disp))+ ','+str('{0:.6f}'.format(leg3_velfinal))+'\n')
countUp = 0
for m in range (0,len(movementArray)-1):
FILE.write(str('{0:.6f}'.format(.001))+',' + str('{0:.6f}'.format(movementArray[m]))+ ','+str('{0:.6f}'.format(velArray[m+1]))+'\n')
countUp = countUp+.001
FILE.write(str('{0:.6f}'.format(.005))+',' + str('{0:.6f}'.format(movementArray[len(movementArray)-1]))+ ','+str('{0:.6f}'.format(0))+'\n')
totalTrajTime = countUp+.005+rampTimeGraph
FILEDisp = open(chooseFileDisp,"a")
FILEDisp.write(str('{0:.4f}'.format(leg1_time))+',' + str('{0:.4f}'.format(leg1_disp))+ ','+str('{0:.4f}'.format(leg1_velfinal))+'\n')
FILEDisp.write(str('{0:.4f}'.format(leg2_time))+',' + str('{0:.4f}'.format(leg2_disp))+ ','+str('{0:.4f}'.format(leg2_velfinal))+'\n')
FILEDisp.write(str('{0:.4f}'.format(leg3_time))+',' + str('{0:.4f}'.format(leg3_disp))+ ','+str('{0:.4f}'.format(0))+'\n')
feedForward = [chooseFile,chooseFileDisp]
FILE.close()
FILEDisp.close()
ftps = ftplib.FTP('192.168.0.254','Administrator','Administrator')
ftps.cwd(trajectRoute)
self.upload(ftps,chooseFile)
self.upload(ftps,chooseFileDisp)
ftps.quit()
os.remove(chooseFileDisp)
failureVal = False
[errorCode,returnString]=myxps.MultipleAxesPVTVerification(socketId,group,chooseFile)
if (errorCode != 0):
failureVal = True
if(leg3_velfinal>vel):
print "Failed Calibration for " + str(self.loadCellCalibrateVoltage(Expectedvoltage,yInt)) + "at " + str(vel) + " mm/s. Please calibrate at a
GREATER velocity to add to table."
```

```
else:
print "Failed Calibration for " + str(self.loadCellCalibrateVoltage(Expectedvoltage,yInt)) + "at " + str(vel) + " mm/s. Please calibrate at a
LESSER velocity to add to table."
except RuntimeError:
if(firstTime<1):
print "Failed Calibration Force Decay for " + str(self.loadCellCalibrateVoltage(Expectedvoltage,yInt))+", running again"
firstTime = firstTime+1
assumeVal = [assumeVal[0]*.9,10,assumeVal[2]*.9,4,assumeVal[4]*.9,.02]
self.calculateChangeVal(zerVal,firstTime,failure,
breaker,expectedValue,Expectedvoltage,holdTime,holdForce,timeRamp,dispRamp,fRamp,dRamp,feedForward,POC,yInt,vel,socketId,relDisp,ftp
s,trajectRoute,group,identifyForce,identifyDisp,assumeVal)
else:
print "Failed calibration for " + str(self.loadCellCalibrateVoltage(Expectedvoltage,yInt))
failure=True

self.calculateChangeVal(zerVal,firstTime,failure,
breaker,expectedValue,Expectedvoltage,holdTime,holdForce,timeRamp,dispRamp,fRamp,dRamp,feedForward,POC,yInt,vel,socketId,relDisp,ftp
s,trajectRoute,group,identifyForce,identifyDisp,assumeVal)
#print "do i get here"
#return [failure, feedForward,firstTime,splits]

else:
failureVal = True
return [failureVal, feedForward,firstTime,splits,totalTrajTime]



def fitfuncLog(self, x,a,b):
return a*np.log(x)+b
def fitfuncLine(self, x,a,b):
return a*(x)+b
def fitfuncExpo(self, x,a,b):
return a*np.exp(-b*x)
def fitfuncExpo2(self, x,a,b,c):
return a*np.exp(-b*x)+c
def fitfuncPosExpo(self, x,a,b,c):
return a*np.exp(b*x)+c
def fitfuncDub(self,x,a,b,c,d):
return a*np.exp(-b*x)+c*np.exp(-d*x)
def fitfuncTrip(self,x,a,b,c,d,e,f):
return a*np.exp(-b*x)+c*np.exp(-d*x)+e*np.exp(-f*x)
def fitfuncTripC(self,x,a,b,c,d,e,f,g):
return a*np.exp(-b*x)+c*np.exp(-d*x)+e*np.exp(-f*x)+g
def fitfuncExpoQuad(self,x,a,b,c):
return a*x**2+b*x+c

#########################################################################################################################
############
#
# Force Control Indentation
#
#
myIndenter.RunIndentation(velocity,displacementExp,AccMax,Dur,expVol,CompleteForce[ActInt][0],forceActStr,forceActVal,POC,feedForwar
d,positioner,socketId,group,fileLoc,yInt)
#########################################################################################################################
#

def RunIndentation(self,velocity, expDisp , Acceleration,
Duration,expVol,Force,forceInd,POC,feedForward,positioner,socketId,group,fileLoc,yInt,totalTrajTime,skin_Thickness):
global univ_Var
counter_point = ""


if(univ_Var == 'UVA'):
counter_point= "10"
else:
counter_point = "8"
expDisp = float(expDisp)
expAcc = float(Acceleration)
expDur = float(Duration)
```

```
myxps.GroupMoveAbsolute(socketId,group,[POC-.6])
[errorCode,returnString]=myxps.GatheringReset(socketId)
if (errorCode != 0):
self.displayErrorAndClose (socketId, errorCode, 'GatheringReset')
sys.exit()
[errorCode, returnString]=myxps.GatheringConfigurationSet(socketId,[positioner+".CurrentPosition","GPIO2.ADC1"])
if (errorCode != 0):
self.displayErrorAndClose (socketId, errorCode, 'GatheringConfigurationSet')
sys.exit()

[errorCode, returnString]=myxps.EventExtendedConfigurationTriggerSet(socketId,[group+".PVT.TrajectoryStart"],["0"],["0"],["0"],["0"])
if (errorCode != 0):
self.displayErrorAndClose (socketId, errorCode, 'EventExtendedConfigurationTriggerSet')
sys.exit()

########################################################################## Change
Here##########################################################################
[errorCode, returnString]=myxps.EventExtendedConfigurationActionSet(socketId,["GatheringRun"],["2000"],["50"],["0"],["0"])
if (errorCode != 0):
self.displayErrorAndClose (socketId, errorCode, 'EventExtendedConfigurationActionSet')
sys.exit()
[errorCode, returnString]=myxps.EventExtendedStart(socketId)
if (errorCode != 0):
self.displayErrorAndClose (socketId, errorCode, 'Event Extended Start')
sys.exit()
self.runForce_indent(expDisp,expVol,velocity,expDur,expAcc,Force,forceInd,POC,group,feedForward,positioner,socketId,yInt,totalTrajTime,ski
n_Thickness)

[errorCode, returnString] = myxps.GatheringStopAndSave(socketId)
if (errorCode != 0):
self.displayErrorAndClose (socketId, errorCode, 'GatheringStopandSave')
sys.exit()




############# Run Control Theory Algorithm (PID) to correct for the settling in the skin ###############
def
runForce_indent(self,expDisp,expectedVoltage,velocity,expDur,expAcc,Force,forceInd,POC,group,feedForward,positioner,socketId,yInt,totalTra
jTime,skin_Thickness):

lopt_globe_val = []
for x in range(len(lopt_globe)):
if(lopt_globe[x][0]==expDisp and lopt_globe[x][1]==velocity):
lopt_globe_val.append(lopt_globe[x][2])
lopt_globe_val.append(lopt_globe[x][3])
lopt_globe_val.append(lopt_globe[x][4])
fileName = feedForward[0]

Integrator_max = 500
Integrator_min = -500
################################## Will Probably need Modification Here ##############################################
if(velocity == 1 or velocity ==2 or velocity == .5):
Pmult = .005
Dmult = .00005
IMult = .0005
if(velocity == 3 or velocity == 4):
Pmult = .005
Dmult = .00005
IMult = .0005
if(velocity == 5):
Pmult = .002
Dmult = .0008
IMult = .00005
Pmult = Pmult*skin_Thickness
Dmult = Dmult*skin_Thickness
```

```
Integrator= 0
Integrator_max = 1*10**-6/IMult

LastError = 0

time_to_stop1= False
count = 0
[errorCode, returnString]=myxps.PositionerSGammaParametersSet(socketId,positioner,1,100,.005,.05)
#[errorCode, returnString]=myxps.PositionerSGammaParametersSet(socketId,positioner,.5,100,.005,.01)

# startTime1 = datetime.datetime.now()

[errorCode,returnString]=myxps.MultipleAxesPVTExecution(socketId,group,fileName,1)
expectedForce = self.loadCellCalibrateVoltage(myxps.GPIOAnalogGet(socketId,["GPIO2.ADC1"])[1],yInt)
#print expectedForce
# stopTime1 = datetime.datetime.now()
# difference = stopTime1-startTime1
# difference = divmod(difference.total_seconds(),60)
# differenceVal = totalTrajTime - difference[1]
startTime=datetime.datetime.now()
stopTime=startTime+datetime.timedelta(seconds=expDur)
prevTime = datetime.datetime.now()
dtprev = 0
while(not(time_to_stop1)):
now = datetime.datetime.now()
if(stopTime<now):
time_to_stop1 = True
dt = now - prevTime

dt = divmod(dt.total_seconds(),60)
dtprev = dt[1]+dtprev

error = (math.log((expectedForce-lopt_globe_val[2])/lopt_globe_val[0])/lopt_globe_val[1]) -
(math.log((self.loadCellCalibrateVoltage(myxps.GPIOAnalogGet(socketId,["GPIO2.ADC1"])[1],yInt)-
lopt_globe_val[2])/lopt_globe_val[0])/lopt_globe_val[1])
#print "error: "+ str(error)
P_value = Pmult * error

if(not(dt[1]==0)):
D_value = Dmult * ((error-LastError))/dt[1]
else:
D_value = 0
Integrator += error*dt[1]

if Integrator > Integrator_max:

Integrator = Integrator_max
# elif Integrator < Integrator_min:
# Integrator = Integrator_min

I_value = Integrator * IMult
PID = P_value + I_value + D_value
#print "P_value: " + str(P_value)
#print "I_value: " + str(I_value)
#print "D_value: " + str(D_value)

LastError = error
prevTime = datetime.datetime.now()
#print PID
[errorCode, returnString] = myxps.GroupMoveRelative(socketId, group,[(PID)])
#print "PID" + str(PID)
##
#if(count == 0):
# stopTime1 = datetime.datetime.now()
# endBeg = stopTime1-startTime1

#print "first move time" + str(divmod(endBeg.total_seconds(),60)[1])
# count = 1

#if((voltage>8)):
```

```
# myxps.GroupMoveAbsolute(socketId,group,[POC-.6])
#print "values" + str(dtprev)
[errorCode, returnString]=myxps.PositionerSGammaParametersSet(socketId,positioner,velocity,80,.005,.05)

[errorCode, returnString] = myxps.GroupMoveAbsolute(socketId, group,[POC-.6])
##########################################################################################################################
#########################
#
######################################## Displacement Control in Step 6
#####################################################################
#
##########################################################################################################################
#########################33


def dispForce(self,vel,holdDur,dispInd,feedForward,socketId,positioner,group,POC,disp):
global univ_Var
counter_point = ""
if(univ_Var == 'UVA'):
counter_point= "10"
else:
counter_point = "8"
fileName = feedForward[1]

[errorCode,returnString]=myxps.GatheringReset(socketId)
if (errorCode != 0):
self.displayErrorAndClose (socketId, errorCode, 'GatheringReset')
sys.exit()

[errorCode, returnString]=myxps.GatheringConfigurationSet(socketId,[positioner+".CurrentPosition","GPIO2.ADC1"])
if (errorCode != 0):
self.displayErrorAndClose (socketId, errorCode, 'GatheringConfigurationSet')
sys.exit()
[errorCode, returnString]=myxps.EventExtendedConfigurationTriggerSet(socketId,[positioner+".SGamma.MotionStart"],["0"],["0"],["0"],["0"])

if (errorCode != 0):
self.displayErrorAndClose (socketId, errorCode, 'EventExtendedConfigurationTriggerSet')
sys.exit()

############################################################################ Change Here
#################################################################
[errorCode, returnString]=myxps.EventExtendedConfigurationActionSet(socketId,["GatheringRun"],["2000"],["50"],["0"],["0"])
if (errorCode != 0):
self.displayErrorAndClose (socketId, errorCode, 'EventExtendedConfigurationActionSet')
sys.exit()
[errorCode, returnString]=myxps.EventExtendedStart(socketId)
if (errorCode != 0):
self.displayErrorAndClose (socketId, errorCode, 'Event Extended Start')
sys.exit()

[errorCode, returnString]=myxps.PositionerSGammaParametersSet(socketId,positioner,int(vel),80,.005,.05)

#[errorCode,returnString]=myxps.MultipleAxesPVTExecution(socketId,group,fileName,1)
[errorCode, returnString] = myxps.GroupMoveAbsolute(socketId, group, [disp])


time.sleep(holdDur)


[errorCode, returnString] = myxps.GroupMoveAbsolute(socketId, group, [POC-.6])

[errorCode, returnString] = myxps.GatheringStopAndSave(socketId)
if (errorCode != 0):
self.displayErrorAndClose (socketId, errorCode, 'GatheringStopandSave')
sys.exit()
def smooth(self,x,window_len=10,window='flat'):
# """smooth the data using a window with requested size.
#
#6 This method is based on the convolution of a scaled window with the signal.
#7 The signal is prepared by introducing reflected copies of the signal
```

```
#8 (with the window size) in both ends so that transient parts are minimized
#9 in the begining and end part of the output signal.
#10
#11 input:
#12 x: the input signal
#13 window_len: the dimension of the smoothing window; should be an odd integer
#14 window: the type of window from 'flat', 'hanning', 'hamming', 'bartlett', 'blackman'
#15 flat window will produce a moving average smoothing.
#16
#17 :
#18 the smoothed signal
#19
#20 example:
#21
#22 t=linspace(-2,2,0.1)
#23 x=sin(t)+randn(len(t))*0.1
#24 y=smooth(x)
#25
#26 see also:
#27
#28 numpy.hanning, numpy.hamming, numpy.bartlett, numpy.blackman, numpy.convolve
#29 scipy.signal.lfilter
#30
#31 TODO: the window parameter could be the window itself if an array instead of a string
#32 NOTE: length(output) != length(input), to correct this: return y[(window_len/2-1):-(window_len/2)] instead of just y.
#33 """
if x.ndim != 1:
raise ValueError, "smooth only accepts 1 dimension arrays."
if x.size < window_len:
raise ValueError, "Input vector needs to be bigger than window size."
if window_len<3:
return x
if not window in ['flat', 'hanning', 'hamming', 'bartlett', 'blackman']:
raise ValueError, "Window is on of 'flat', 'hanning', 'hamming', 'bartlett', 'blackman'"
s=np.r_[x[window_len-1:0:-1],x,x[-1:-window_len:-1]]
#print(len(s))
if window == 'flat': #moving average
w=np.ones(window_len,'d')
else:
w=eval('np.'+window+'(window_len)')
y=np.convolve(w/w.sum(),s,mode='valid')
return y
```