Force-Displacement Glenoid Reamer Guide

St Joseph's Health Care London

Western Engineering

Roth | McFarlane Hand & Upper Limb Centre
St. Joseph's Health Care London

Author:
Matthew Stokes

March 7, 2014

Contents

List of Figures

List of Tables

1 Overview

Reamer project was designed to calculate the applied force and resulting displacement of the glenoid during reaming. However due to modern sensor limitations (namely a small wireless load cell) the reamer will not be rotating during measurements.

This document includes information pertaining to setting up the system, recording measurements from the sensors, and interacting with this information. It does not include any background information about the project.

2 Equipment

Equipment has been separated into six distinctly different categories: sensing and tools, 3D printed attachment parts, DAQ & other intermediary electronics, software, supplementary files, and misc.

2.1 Sensing and Tools

Item	Product	Manufacture	Description
1	Load cell	ATI	Nano25 FT 13036
2	Optotrack Certus	NDI	Optical Camera
3	Optical tracker	NDI	
4	Reamer		Surgical reamer

Table 1: Sensing and Tools

2.2 3D Printed Attachment Parts

All parts were designed by Matthew Stokes and printed on his MakerBot Replicator using the settings 100% infill, 0.2mm layer height.

Item	Product	Description
5	Head Couple	Between head of reamer and load cell
6	Back Plate Couple	Between back plate of load cell and reamer shaft
7	Reamer Attachment Plate	Base connecting to the reamer
8	Optical Tracker Attachment Plate	connecting to the reamer attachment plate
9	Standoff	Hidden via thru hole securing reamer attachment plate

Table 2: 3D Printed Attachment Parts

2.3 DAQ & Other Intermediary Electronics

Necessary intermediary products connecting to sensors and involved in setup/calibration.

Item	Product	Manufacture	Description
10	Wireless Strober	NDI	Strobe the optical trackers
11	Stylist	NDI	tracker offset to tip of stylist
12	Optotrak Box	NDI	Sync strober and tracking camera
13	Load Cell DAQ	ATI	filter and amplify
14	Multifunction DAQ	NI	load cell DAQ to USB
15	Computer		Windows XP

Table 3: DAQ & Other Intermediary Electronics

2.4 Software

Item	Program	Author	Description
16	Motion Station	HULC Lab	LabView optical tracker measurements
17	Loadcelly	Stokes	LabView load cell measurements
18	pushpush.py	Stokes	Python interacting with data output
19	ATIDAQFT.NET	ATI	Native software
20	NDI First Principals	NDI	Native software
21	NI Measurement & Automation	NI	Native software

Table 4: Software

2.5 Supplementary Software Files

These files should not be modified.

Item	File Name	Extension	Description
22	nano25calibration	.cal	load cell calibration
23	bsty0213	.rig	rigid body for the stylist
24	smart_02	.rig	rigid body for base NI optical tracker

Table 5: Supplementary Software Files

2.6 Misc Cables, Nuts, Bolts, Wires

Item	Product	Description
25	Threaded Rob	Connecting head couple to load cell
26	4x Bolts	Connecting reamer attachment plate and optical tracker plate
27	4x Nuts	
28	2x Screws	Attach reamer attachment plate to standoff
29	Broken out 24 Pin Cable	Connect Load Cell DAQ to NI-USB-6211
30	Thread	Secure reamer attachment plate to back end of reamer

Table 6: Misc Cables, Nuts, Bolts, Wires

3 Hardware Setup

The project consists of two main systems which are the load cell and the optical tracker. Setup for each of these systems is independent. The reamer itself will be setup as shown in the image below.

Figure 1: Reamer Outfitted with Load Cell and Optical Tracker

Contained in this assembly are the following items:

Item	Product
1	Load Cell
3	Optical Tracker
4	Reamer
5	Head Couple
6	Back Plate Couple
7	Reamer Attachment Plate
8	Optical Tracker Plate
9	Standoff
25	Threaded Rod
26	4X Bolts
27	4X Nuts
28	2X Screws
30	Thread

Table 7: Reamer Assembly Parts List

3.1 Optical System Setup

An overview of the optical system setup is shown in the image below.

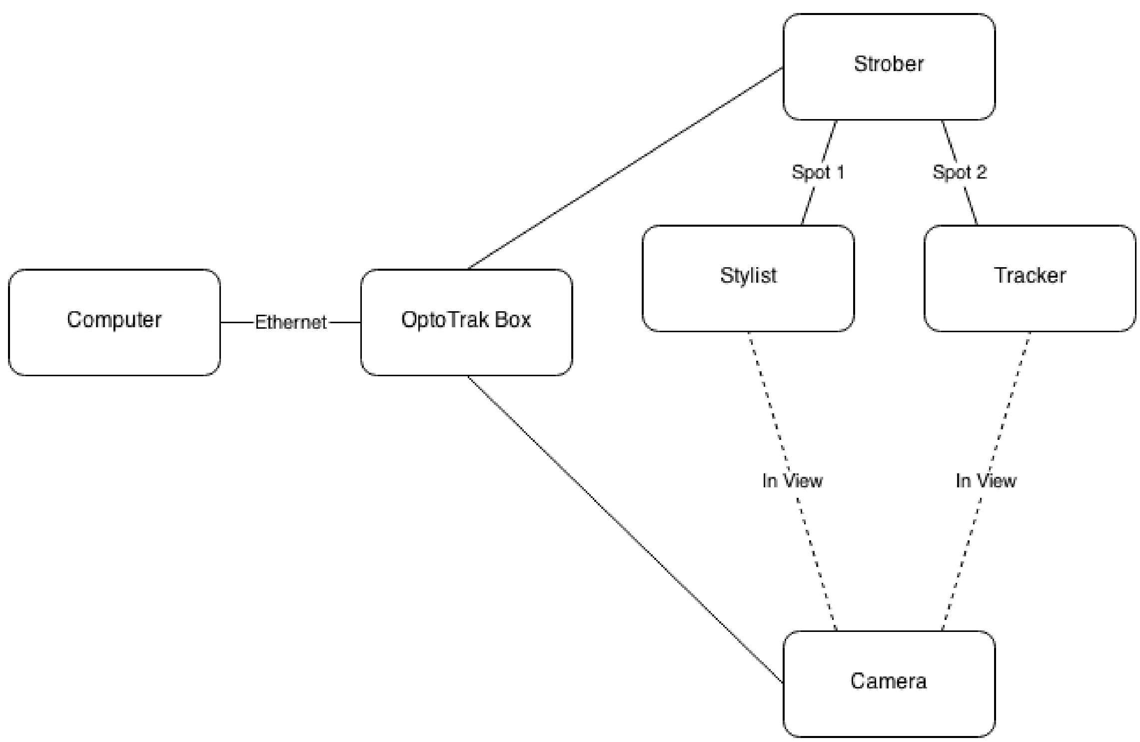

Figure 2: Optical Tracker System Overview

Plugin and power on the optical tracker camera. You should see a green light which indicates the power is on. The optical tracker camera communicates with the Optotrak Box which is connected via Ethernet to the main computer. Power on the Optotrak Box and plug the cable into the optical tracker.

Figure 3: Optotrak Box

Figure 4: Certus Optical Camera

Next we need to wire the tracker on our reamer. The tracker should connect to the strober box which can hold up to 5 trackers. Trackers may also be daisy chained with up to 4 trackers. To begin we will wire the tracker as part of our reamer assembly and also a stylist (which we be used for offset calibration). Note we will wire the stylist in port 1 and the tracker in port 2. This order matters later.

3.1.1 Setup Test

To ensure the system is setup properly open NDI First Principals and click *Create New*

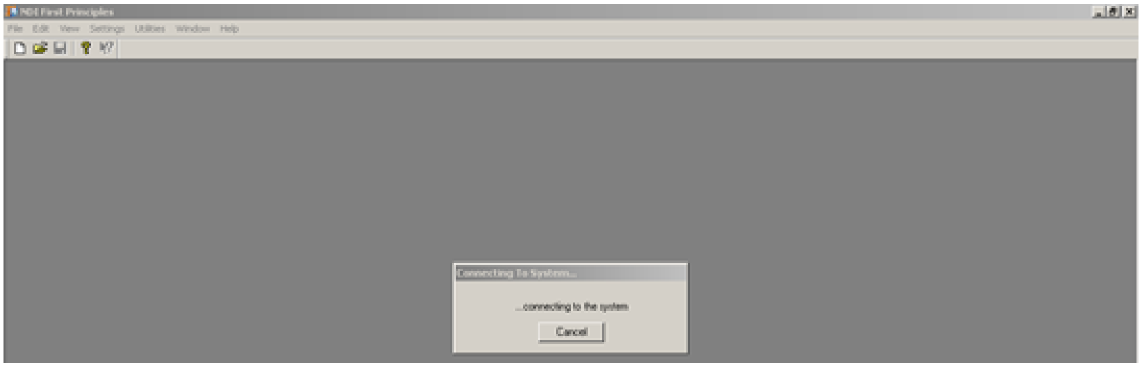

Figure 5: NDI First Principals

If everything is connected we should will be taken to the next screen for setting up a new project seen below. Else go back and try again.

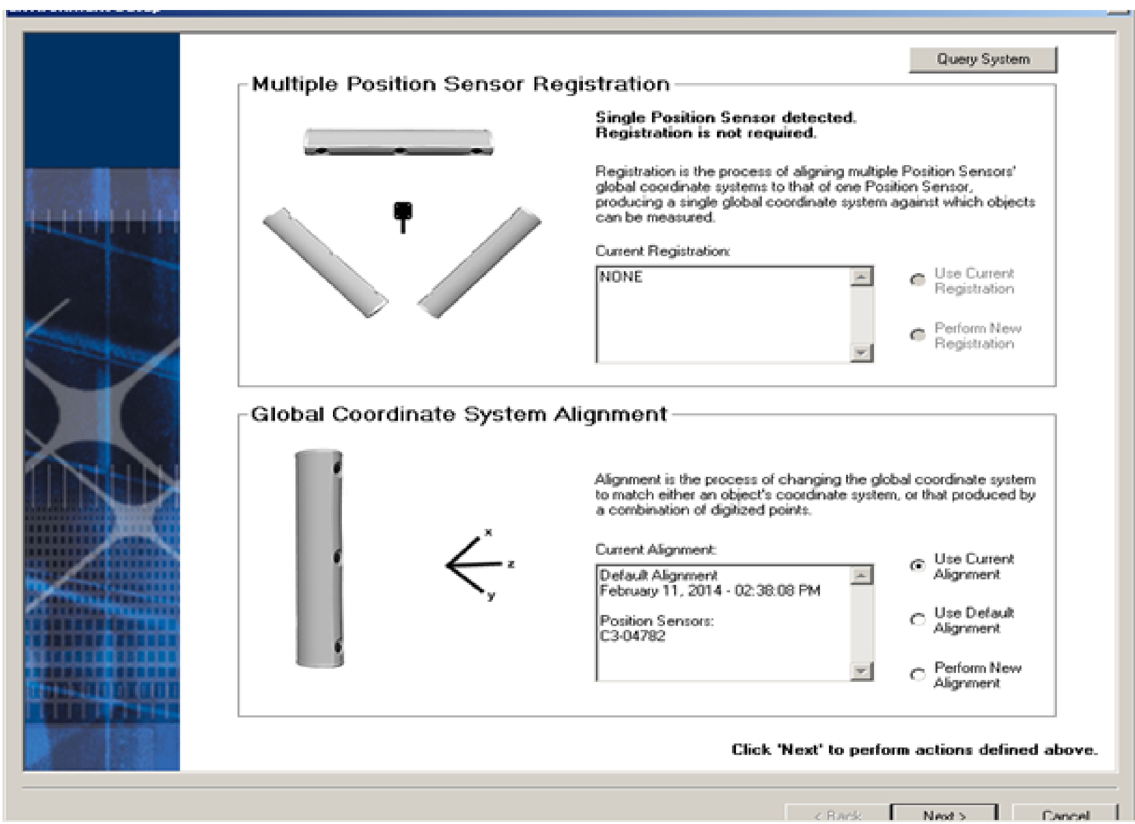

Figure 6: NDI First Principals

3.1.2 Offset Calibration

Now that we know our optical tracker is being read by the camera we will calculate the offset between the center of the tracker and the tip of the reamer. We do because we are interested in the displacement of the tip as opposed to the displacement of the tracker. This allows us to also pivot about the tip without measuring any displacement.

To determine the offset we will position the stylist to touch the tip of the reamer and record the x,y,z values of both the stylist and the other tracker. The difference between these values will be our offset. We will later offset the trackers calibration matrix by these values in return to motion station.

Clicking the *Next* button on the multiple position sensor registration screen should take us to the experiment setup screen shown below.

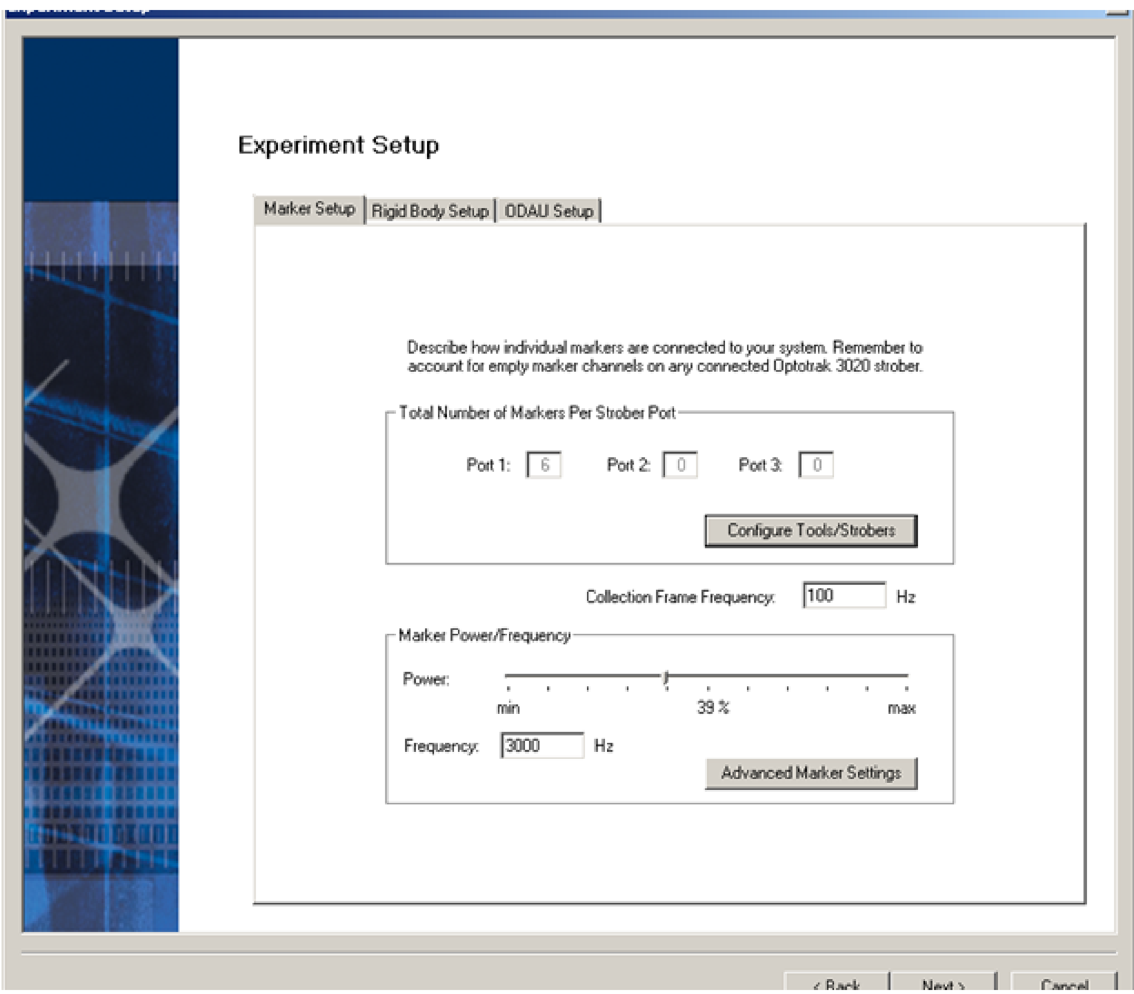

Figure 7: Experiment Setup

Here we should see **6 total markers**- 3 from our stylist and 3 from our tracker. If you do not see six total then something has not been setup correctly.

At this stage we are required to modify some of the settings in order to get better functionality out of our optical system. Modify the following parameters:

Parameter	New Value
Collection Frame Frequency	50
Frequency	3500
Advanced — Duty Cycle	80

Table 8: Reamer Assembly Parts List

These modifications are also shown in the images below.

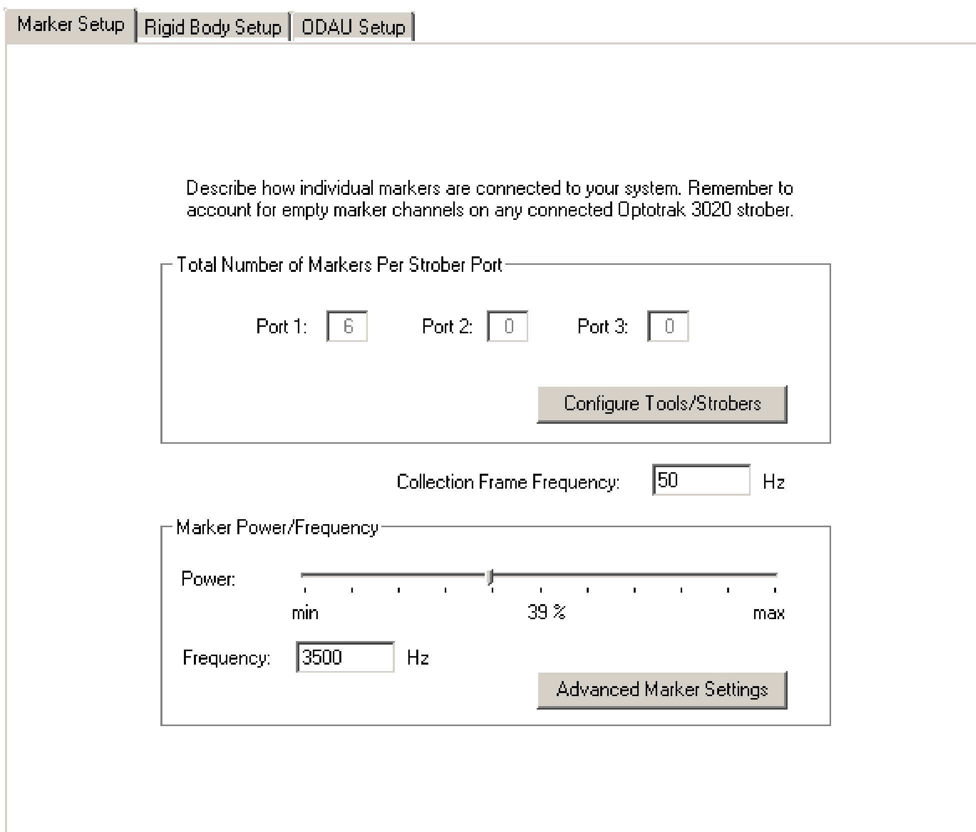

Figure 8: Marker Setup

Once these settings have been changed we need to click on the Rigid Body Setup Tab.

Here we need to add two rigid body files which can be found in Appendix A: **bsty0213.rig** and **smart_02.rig**. Bsty2013 was created in house and smart_02 is the rigid body file for a tracker created by NDI. These files contain information used by the camera to interoperate 3 markers as a rigid body. You must ensure to load the bsty0213 file before the smart_02 file as this is order

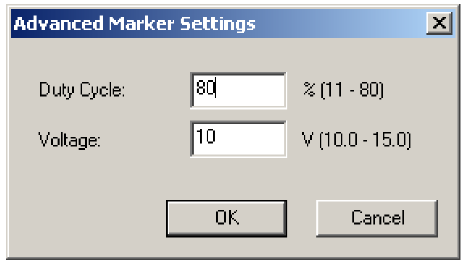

Figure 9: Advanced Marker Settings

Figure 10: Rigid Body Setup Tab

corresponds to our placement earlier. The completed rigid body setup is shown below.

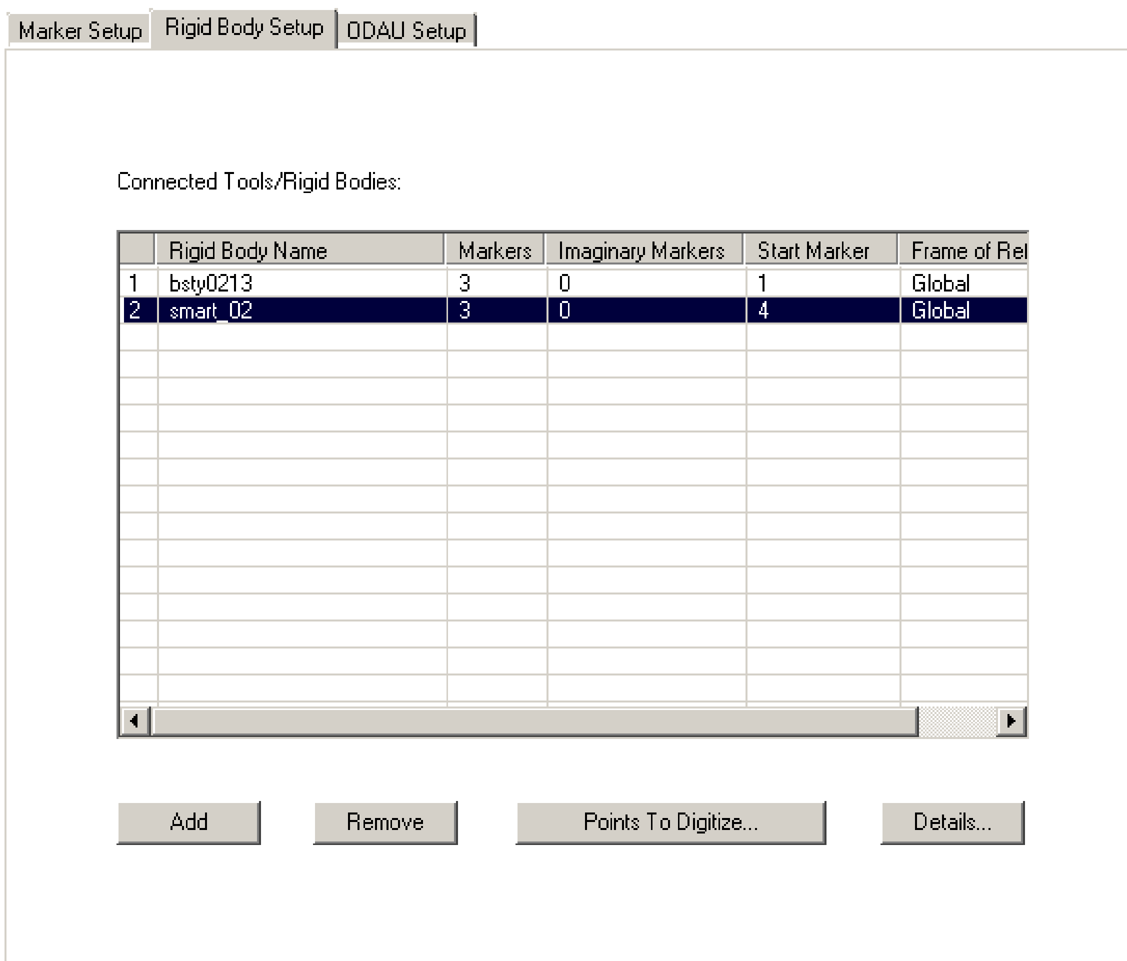

Figure 11: Two Rigid Bodies

We can now click *Next* as we are finished the experimental setup. Clicking *Next* may or may not prompt you with the screen seen below. Click *no*. Just do it. You are lying to yourself if you believe this will actually work in wireless mode.

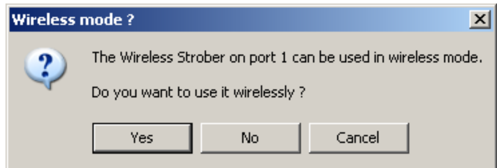

Figure 12: Wireless Mode

Green means the camera can see the markets and I assume you can figure out what red means. Interesting note: if you are in an odd situation where you think it should be reading values but it's not simply put you hand in front of the tracker to hide it from the camera then remove it. This does something along some lines of resetting some internal stuff.

Click the *View* — *Probe View*. This should present you with a screen similar to the one shown below (minus any actual valueswe‘re getting there).

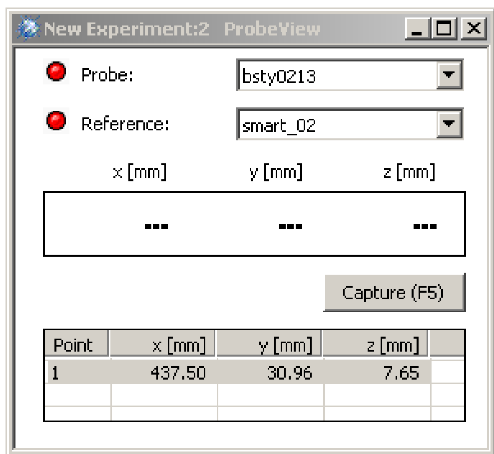

Figure 13: Offset

This is where we determine the offset. We use the smart_02 sensor as the reference and the bsty0213 as the probe. Then once both rigid bodies are green (in sight) it should start showing values which are the difference between the two trackers. It‘s important to note that the stylist has already been configured to offset its tracker to the tip so measurements are from the tip of the stylist to the center of the optical tracker on the reamer.

Touch the tip of the stylist to the tip of the reamer and hit *F5* or the *Capture* button. The resulting values should be similar to the ones shown in the image above else something has really changed which is odd. Record these values somewhere as we will use these soon. It‘s at this time we are ready to move on to using Return of the Motion Station (yay). But before we do this we will take a second to setup our load cell.

3.2 Load Cell Setup

An overview of the optical system setup is shown in the image below.

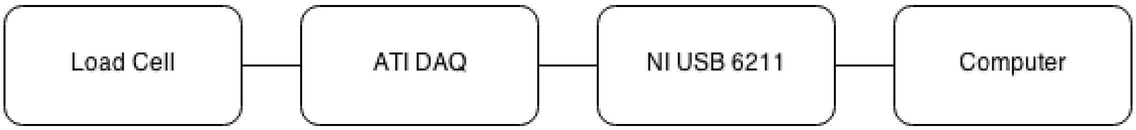

Figure 14: Load Cell Overview

The load cell plugs into the ATI DAQ box. This box connects to the NI USB-6211 device via a special broken out cable.

Figure 15: ATI DAQ

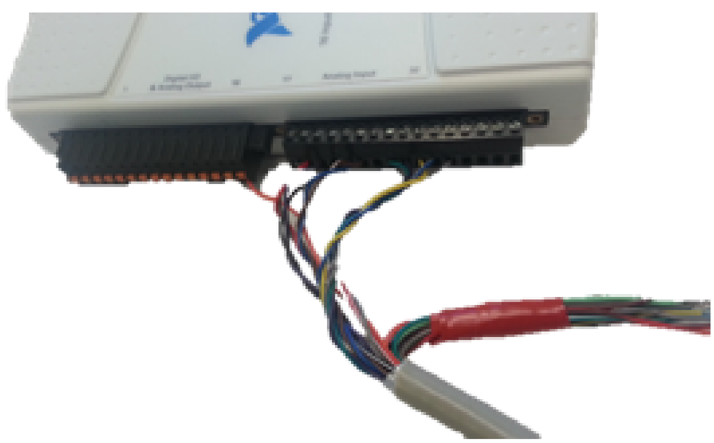

Figure 16: Broken 24 Pin Cable

Finally, the NI USB-6211 box connects to the USB port on the main computer.

Figure 17: NI USB-6211

3.2.1 Setup Test

Open NI Measurement & Automation Explorer which should display the screen shown below.

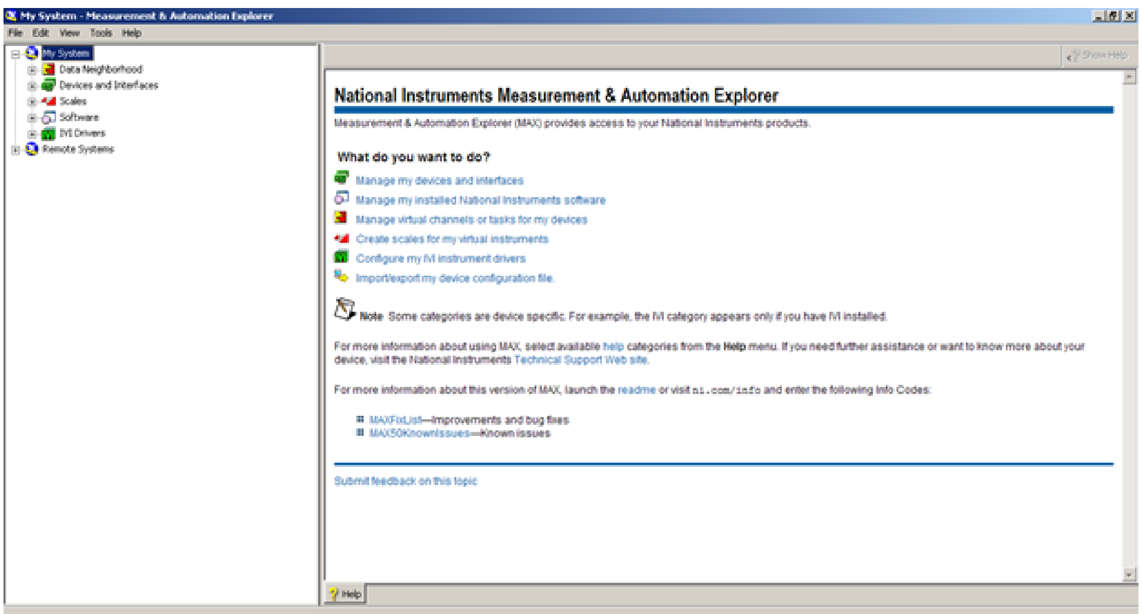

Figure 18: National Instruments Measurement & Automation Explorer

Select *Devices and Interfaces — NI-DAQmx Devices* and make note of the **DEV port** for NI USB-6211 (in our example its DEV 4 but yours may differ).

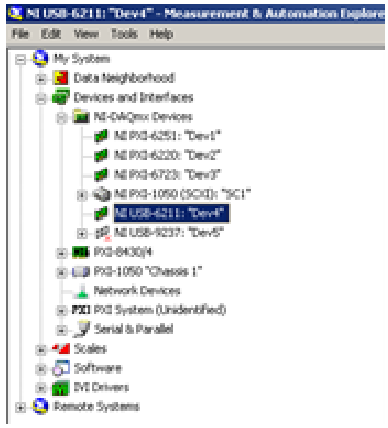

Figure 19: DEV port

Next open ATIDAQFT.NET program shown below.

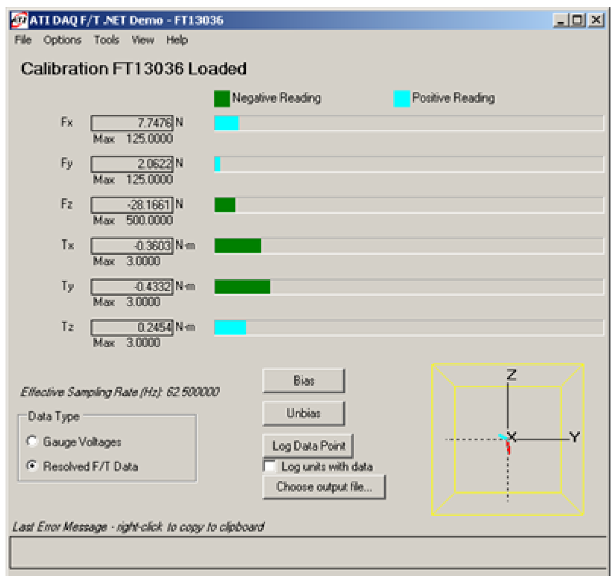

Figure 20: ATIDAQFT.NET

If the load cell values shown on this screen wont stop climbing than you havent selected the proper DEV port for viewing. To change DEV ports click *Options — DAQ Hardware Options* shown below and change the device name to your DEV port. Also important to note is that our load cell works over channels 0-5 (X, Y, Z force and Rx, Ry, Rz torques).

Click *Ok* and now you should see your load cell values as shown previously. If not you have configured something wrong and you need to go back and fix this.

At this stage we have both our load cell and optical tracker setup properly and we are now ready to move on and get acquainted with our software systems for recording sensor measurements and parsing the data.

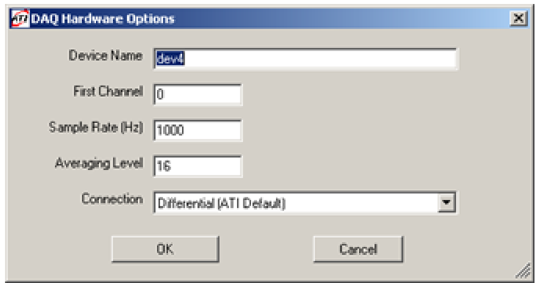

Figure 21: DAQ Hardware Options

4 Sensor Software

You have already been exposed to the native software for the load cell and optical tracker and used these native solutions to ensure our hardware is configured properly. Along with the setup software we have used thus far we will also be using three main pieces of software: return to the motion station- to record optical tracker values, loadcelly- to record load cell values, and pushpush.py to parse all the data super easily.

4.1 Return of the Motion Station

Return to motion station which was developed origionally by Dr. Ferreira as part of his PhD simulator and since then has been expanded by a number of coop students. It enjoys crashing and freezing all the time but there really is no better alternative and as buggy as it is, still provides much added value.

Start by opening Return of the Motion Station. You may notice that one of the main windows doesnt enjoy life, thats fine just ignore it.

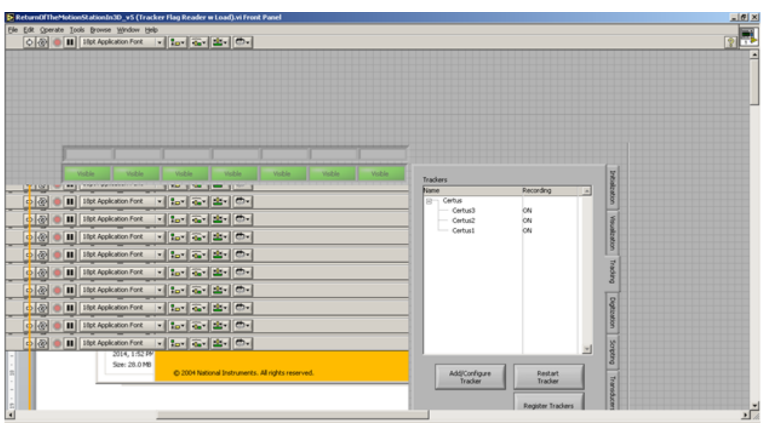

Figure 22: Return to the Motion Station

Run the project and you should be prompted with the screen shown below.
Since this is our first time running the program we should Create New however once you have already setup the experiment you can just open your already existing session folder.

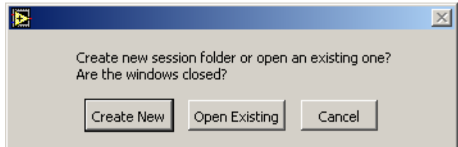

Figure 23: Create New Session Folder

Creating a new session should make the large window go black and pretty much nothing else visible will happen. (It will create all the folders structure behind the scenes among other things).

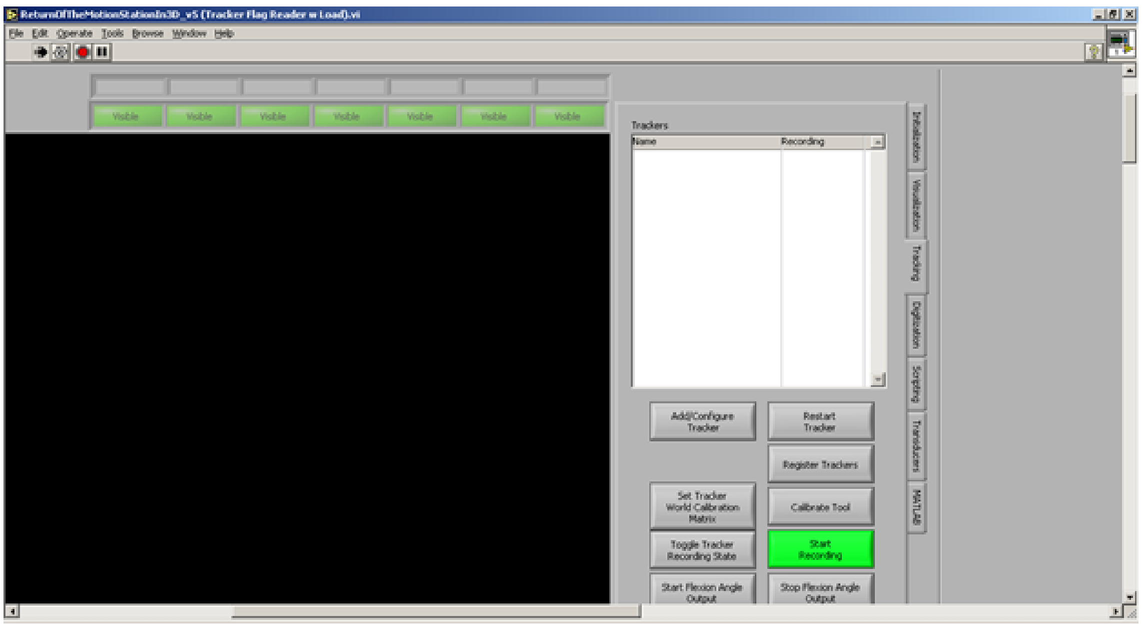

Figure 24: Return to Motion Station Running

Now we want to add our optical tracker. At this point if we havent already we can remove the stylist as we should only have the optical tracker attached to our reamer in slot 1 of our strober. I will repeat. We should only have 1 optical tracker plugged into strobber in slot 1 which is the tracker attached to the reamer.

Click *Add/Configure Tracker* which will display the window shown below. We will select the *Certus* option as we are using a Certus tracker

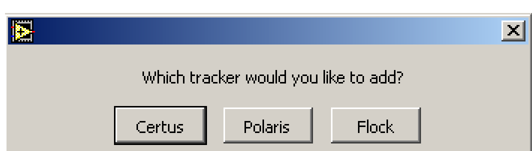

Figure 25: Add/Configure Tracker

Selecting Certus will bring up the Certus Setup Dialog.vi shown below

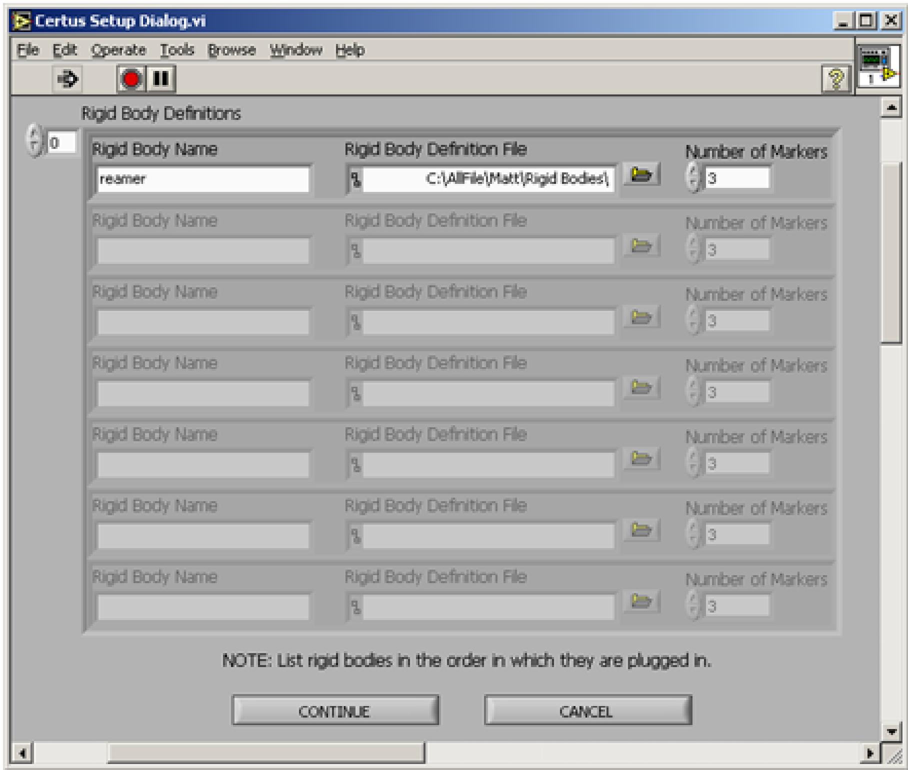

Figure 26: Certus Setup

Here we want to add our smart_02 rigid body and we can name it whatever we like. This is shown in the image above. Click *Continue*.

At this stage we should be brought back to the main screen and the certus optotrak box should start beeping at us. Give this process about 2-3 minutes. Dont touch anything Motion Station is really sensitive right now. Once the process is done we should see the first slot either turn green or red which means weve successfully added our optical tracker into Motion Station.

Next we need to tell motion station to record the tracker. This is easily done with selecting the Certus 1 tracker from the list of trackers and clicking T*oggle Tracker Recording State*. This should add *ON* to the list of trackers column.

The last thing we must do is *Calibrate Tool* to add the offset we calculated earlier. Click *Calibrate Tool* to bring up the *Calibration Dialog.vi*. Insert the values you recorded into the far right column of the matrix and inset 1s along the diagonal. This is shown below.
To ensure our calibration matrix has been set, after clicking *OK* scroll down and look for the vi *Tool Calibrated Offsets* as shown in the image below. If these values arent the same as the matrix you just inputted youve done something wrong.

Wonderful! Now we are all ready to being recording. To do this as you may have guessed click the big green Start Recording button and once we stop recording we will be prompted to enter a

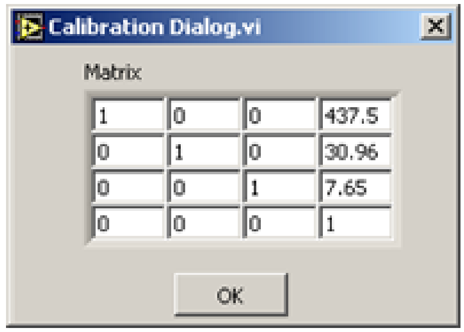

Figure 27: Calibration Matrix

filename. But first before we get underway with our experiment we need to configure our load cell to a state where it too is ready to record on the press of a button.

4.2 Loadcelly

Loadcelly was designed to read input voltages from a load cell on a given DEV, run these through a calibration script (given by ATI), and output values in N to screen with the ability to record these values to a file. General program can be seen below. This is the program you will use to get load cell values.

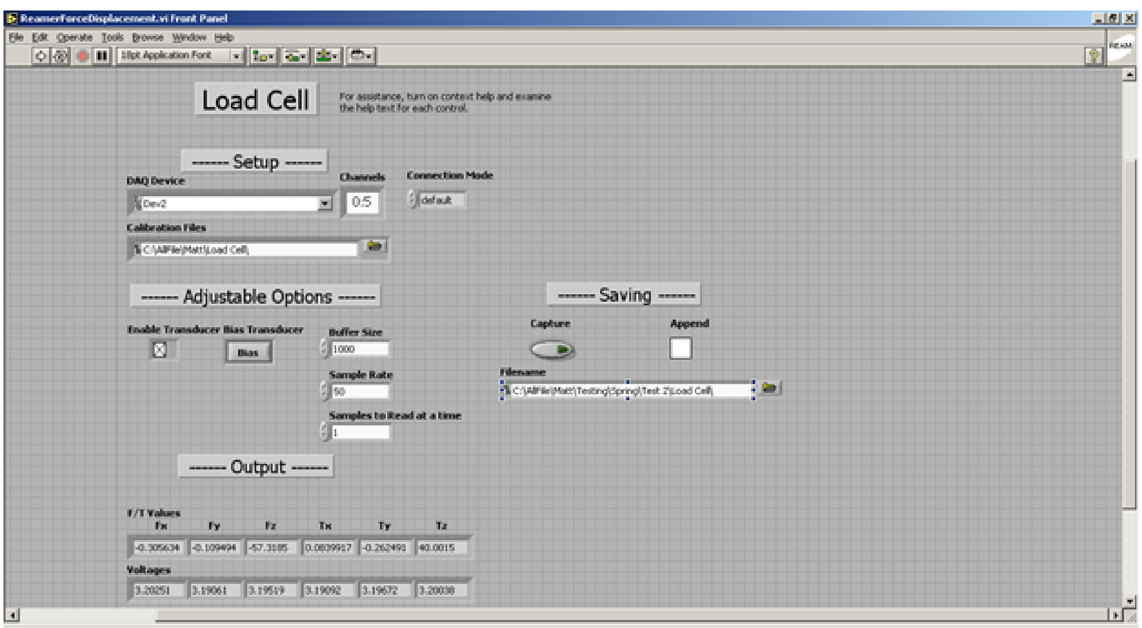

Figure 28: Loadcelly

To use this program first we must select the DEV which our load cell is on. If you dont know what DEV port this is on refer back to the step where we find the NI-6211 DEV.

Next we must load our calibration script (supplied by ATI). If you dont have this calibration script it is included as Appendix A.

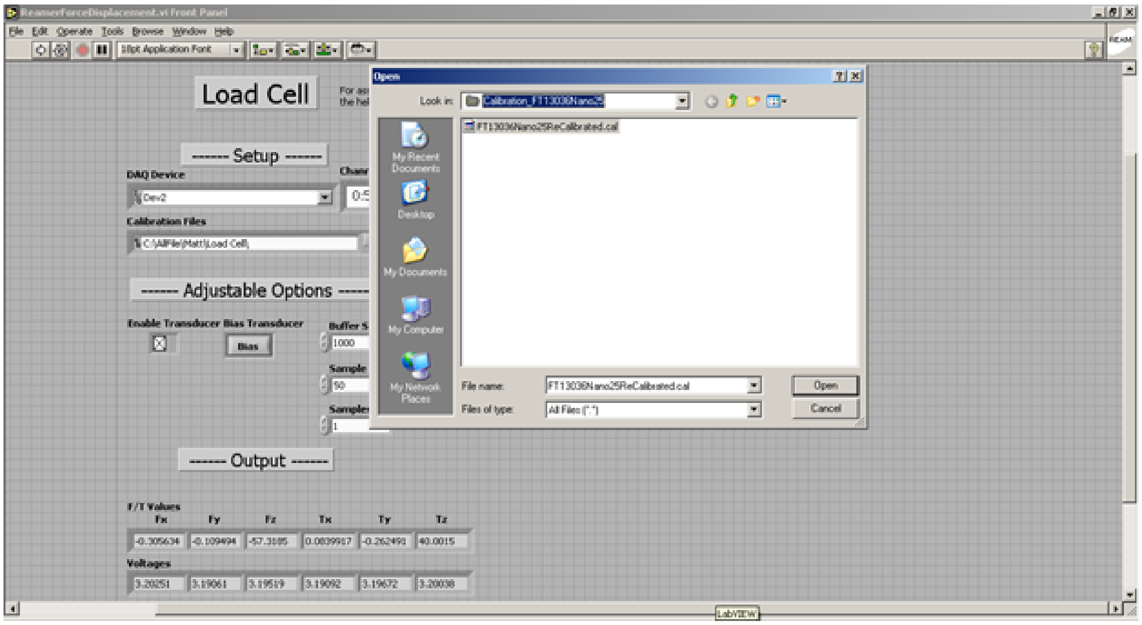

Figure 29: Loadcelly Calibration File

Run the program! You should see the force and voltage values constantly jump which means yay youre reading the load cell sensor values in the program.

An option worth mentioning is the BIAS button which will bias the load cell values. You can press this if you like but my later awesome software program will handle this if you dont.

Lastly when you are ready to record you should first name the file and select a location then hit the Capture button which will turn green and start recording.

4.3 Recording

Recording is easy. Click the record button in each of the two programs then go preform your experiment. Once the experiment is done stop both recordings (motion station will prompt you to save the motion record and the file will be in XXX location)

The optical tracker file is outputted with the following headers:

Timestamp	Dx	Dy	Dz

Table 9: ot.csv

The load cell file is outputted with the following headers:

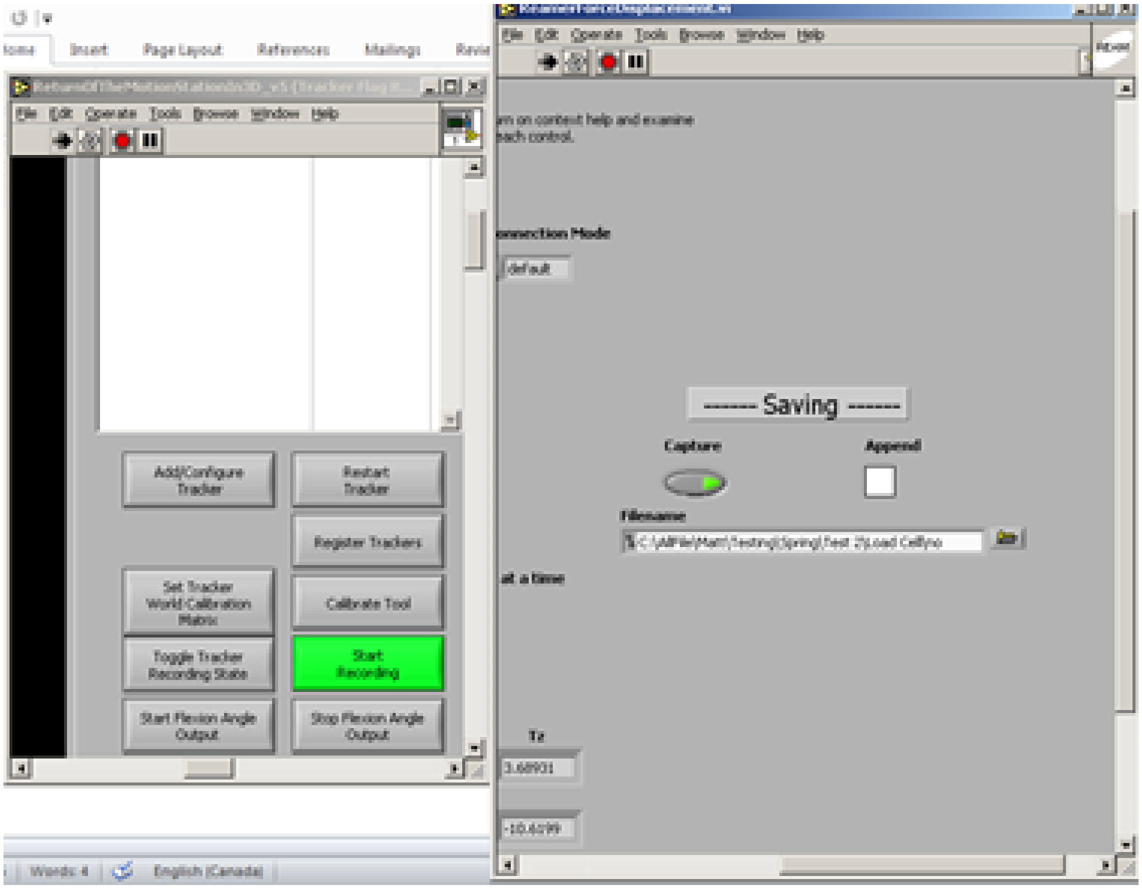

Figure 30: Recording

Timestamp	Fx	Fy	Fz

Table 10: lc.csv

The programs can be combined but greater minds than mine are needed to do this inside of Motion Station. So instead you will have the two programs run simultaneously and you will output two text files which will then be imported into the program pushpush.py which will do many awesome things including automatically trimming the data to remove wait times before or after recording, bias, normalize, filter, and export ready to be plotted.

5 Experimental Procedure

We are looking to measure values from a push. You may record as many pushes as you like however it is recommended that you add a pause ¿ 0.25s between each push so that the events can be considered independent.

6 Post Processing

Pushpush.py is used to extract raw information from the two text files and allow the user access to this in a much more manageable form (Motion Station outputs irrelevant information for our experiment, in a non-consistent formatting). Pushpush.py utilizes abstraction, inheritance, and the visitor design pattern to provide us easy access to our sensor data.

The main.py file is your main interaction with the program. The program should be called in the following manner and requires 3 command line arguments:

```
python main.py  lc inputfilepath  ot inputfilepath  o outputfilepath
```

The output file path will be created if it does not already exist. The program will not execute if improper arguments have been provided. main.py is shown below
code
You use to have to go into this main.py file and change values but that got annoying so I made a little GUI.

```
python main_gui.py  lc inputfilepath  ot inputfilepath  o outputfilepath
```

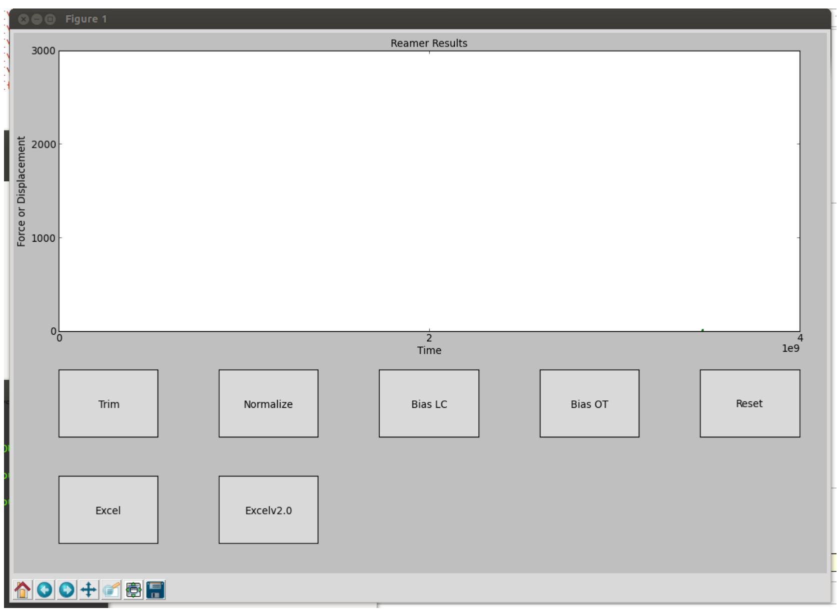

Figure 31: GUI

Now it may appear as if its not working however you need to remember that most of the original timestamp values are 3,000,000,000. Therefore if we ever expect to see anything meaningful we need to normalize. Show below is an example of what the data looks like normalized.

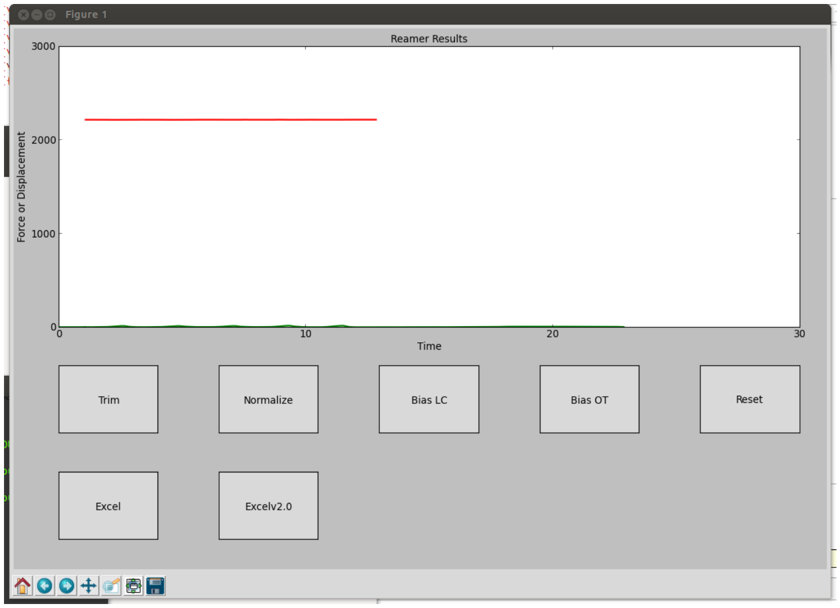

Figure 32: GUI Normalized

Next, let's bias both the load cell and optical tracker as shown below.

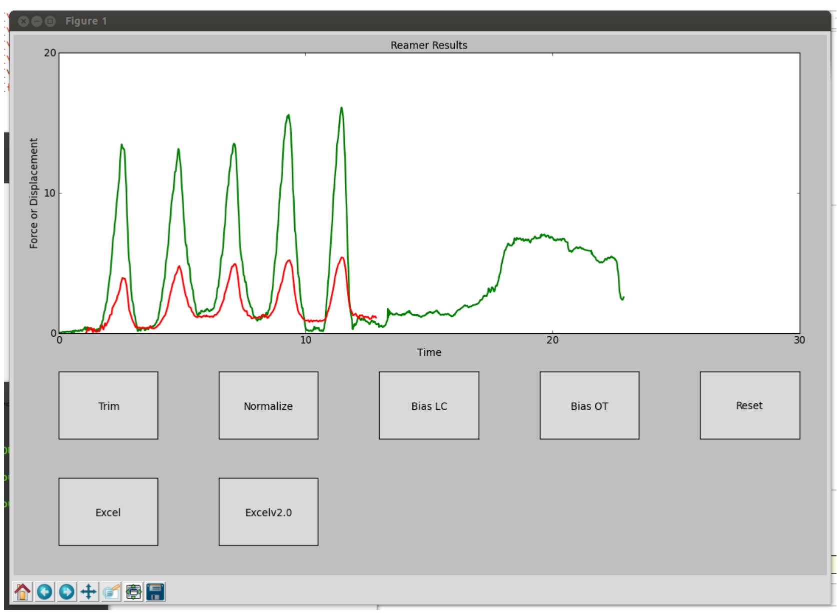

Figure 33: GUI Normalized and Biased

Now this is starting to look like real data! We can remove the secions at the start and end where we have load cell values without optical tracker by clicking the *Trim* button.

Intuitively reset will reset back before we preformed any operations. The last thing to touch on

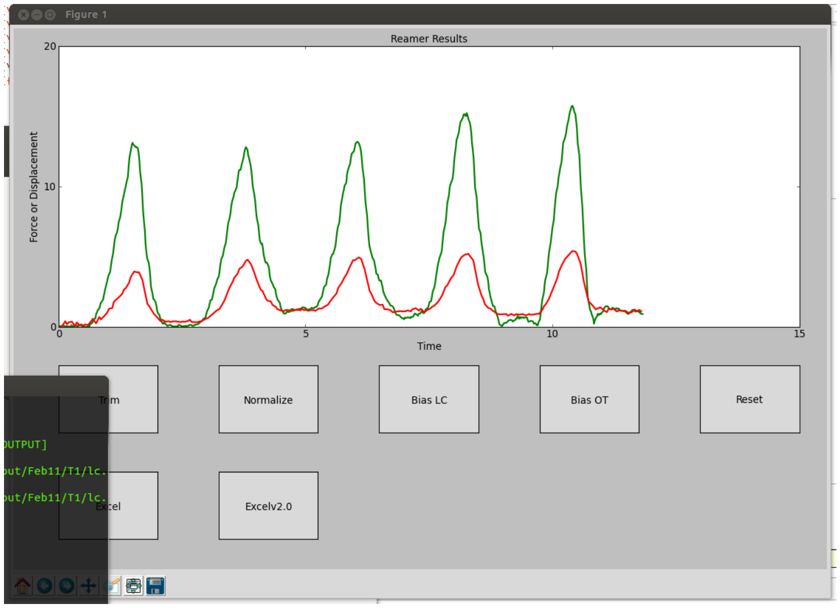

Figure 34: GUI Normalized, Biased, and Trimmed

are these Excel and Excelv2.0 buttons. Excel will output the data to the output file specified in the origional command. We have added some very nice functionality where we will interpolate all missing values to give each load cell and optical tracker measurements a complementary pair. This allows us to calculate stiffness and through interpolation richen our data.

We will export to excel in the following format:

Timestamp	Fr	Dr	K

Table 11: Excel

The excel2.0 button will also remove all values with a Fr below 2.0N. These values can be associated with down time prepping between pushes. This allows us to quickly visually note distincly different pushes.

All code will be included in Appendix B.

A Supplementary Software Files

A.1 smart_02

```
Marker Description File

Real 3D
3          ;Number of markers
1         ;Number of different views
Front
Marker        X            Y           Z        Views
1      -10.0955      -5.8402       0.0000      1
2       10.0645      -5.8402       0.0000      1
3        0.0309      11.6803       0.0000      1

Normals

Marker        X            Y           Z
0.0000       0.0000      -1.0000      1
0.0000       0.0000      -1.0000      1
0.0000       0.0000      -1.0000      1

MaxSensorError
0.20

Max3dError
0.50

MarkerAngle
60

3dRmsError
1.00

SensorRmsError
0.10

MinimumMarkers
3

MinSpread1
0.00

MinSpread2
0.00

MinSpread3
0.00
```

A.2 bsty0213.rig

```
Marker Description File

Real 3D
3        ;Number of markers
1       ;Number of different views
Front
Marker        X            Y           Z       Views
1        1.8639      64.4256      9.1914     1
2     −18.4768      63.0615      6.9789     1
3      −9.9560      81.3896      7.9817     1

Normals

Marker        X            Y           Z
−0.1079      −0.0043       0.9942     1
−0.1079      −0.0043       0.9942     1
−0.1079      −0.0043       0.9942     1

MaxSensorError
0.05

Max3dError
0.10

MarkerAngle
60

3dRmsError
0.10

SensorRmsError
0.05

MinimumMarkers
3

MinSpread1
0.00

MinSpread2
0.00

MinSpread3
0.00
```

A.3 FT13036Nano25ReCalibrated.cal

```xml
<?xml version="1.0" encoding="utf-8"?>
<!-- NOTE: To parse this file for your own software, use the "UserAxis" elements     -->
<!-- to construct the calibration matrix.  The "Axis" elements contain a scaled and  -->
<!-- transformed version of the matrix used for ATI's internal purposes only.  When  -->
<!-- you send your sensor back for recalibration, or replace your sensor with a new  -->
<!-- one, you will receive a new calibration file with a different matrix.  Make     -->
<!-- sure any custom software you write stays up to date with the latest version of  -->
<!-- the calibration matrix.                                                          -->
<FTSensor  Serial="FT13036" BodyStyle="Nano25" Family="DAQ" NumGages="6" CalFileVersion="1.1">
  <Calibration  PartNumber="SI-125-3" CalDate="5/13/2013" ForceUnits="N" TorqueUnits="N-m" DistUnits="m" OutputMode="Ground Referenced Differential" OutputRange="20" HWTempComp="True" GainMultiplier="1" CableLossDetection="False" OutputBipolar="True">
    <Axis Name="Fx" values=" -0.59403  -0.44086   0.17155  32.56920   0.24860 -34.12845 " max="125" scale="2.59826764654692"/>
    <Axis Name="Fy" values="  2.18077 -38.50657  -1.33492  18.50206  -0.69315  19.67996 " max="125" scale="2.59826764654692"/>
    <Axis Name="Fz" values=" 20.55200  -0.12973  20.36407   1.00806  19.39320   0.18228 " max="500" scale="0.775486826825057"/>
    <Axis Name="Tx" values="  0.22784  -0.35271  34.87063   2.31695 -32.82331  -0.58428 " max="3" scale="135.690306256662"/>
    <Axis Name="Ty" values="-40.37538  -0.02850  20.20270   0.80950  19.10862   0.82850 " max="3" scale="135.690306256662"/>
    <Axis Name="Tz" values=" -0.53647 -20.17550   0.95280 -19.28328   1.35746 -19.62813 " max="3" scale="164.97900909492"/>
    <BasicTransform Dx="0" Dy="0" Dz="0.008382" Rx="0" Ry="0" Rz="0"/>
    <UserAxis Name="Fx" values=" -0.22863  -0.16967   0.06602  12.53497   0.09568 -13.13508 " max="125"/>
    <UserAxis Name="Fy" values="  0.83932 -14.82010  -0.51377   7.12092  -0.26678   7.57426 " max="125"/>
    <UserAxis Name="Fz" values=" 26.50206  -0.16729  26.25973   1.29991  25.00777   0.23505 " max="500"/>
```

```
    <UserAxis Name="Tx" values="  0.00871  -0.12682   0.25268   0.07676  -0.24413   0.05918 " max="3"/>
    <UserAxis Name="Ty" values=" -0.29564   0.00121   0.14833  -0.09910   0.14002   0.11620 " max="3"/>
    <UserAxis Name="Tz" values=" -0.00325  -0.12229   0.00578  -0.11688   0.00823  -0.11897 " max="3"/>
  </Calibration>
</FTSensor>
```

B Code

B.1 Packets

B.1.1 Packet.py

```
class Packet(object):
  """ Packet is an abstract class for information packets from
     various devices"""

  def __init__(self, name="Unnamed", timestamp=0):
    """
    Args:
      timestamp: Timestamp from the Epoch. Default 0 (unset)
      name: Name for this packet. Default 'Unnamed'

    Attributes:
      __name: A string containg the name of the packet
      __timestamp: A float containing the timestamp in s from the
         Epoch
    """
    self.__name = name
    self.__timestamp = float(timestamp)

  def __str__(self):
    return "Name: %s, Timestamp: %.5f" % (self.__name, self.
       __timestamp)

  def get_name(self): return self.__name
  def set_name(self, name): self.__name = name

  def get_timestamp(self): return self.__timestamp
  def set_timestamp(self, value): self.__timestamp = value
```

B.1.2 OpticalTrackerData.py

```python
from Packet import Packet
import math

class OpticalTrackerData(Packet):
  """ Subclass of Packet for data captured by an optical tracker.
  Currently holds Dx, Dy, Dz. Modify as needed
  """

  def __init__(self, name="opticaltrackerdata", timestamp=0, dx=0, dy=0, dz=0):
    """ Super class receives timestamp value and default name of 'opticaltrackerdata'

    Args:
      name: packet name. opticaltrackerdata by default.
      timestamp: Timestamp from the Epoch. Default 0 (unset)
      dx: Displacement in X
      dy: Displacement in Y
      dz: Displacement in Z

    Attributes:
      __dx: A float containing displacement in X
      __dy: A float containing displacement in Y
      __dz: A float containing displacement in Z
    """
    Packet.__init__(self, name, timestamp)
    self.__dx = float(dx)
    self.__dy = float(dy)
    self.__dz = float(dz)

  def __str__(self):
    return "Name: %s, Timestamp: %.5f, Dx: %.2f, Dy: %.2f, Dz: %.2f" % (Packet.get_name(self), Packet.get_timestamp(self), self.__dx, self.__dy, self.__dz)

  def get_dx(self): return self.__dx
  def get_dy(self): return self.__dy
  def get_dz(self): return self.__dz
  def get_dr(self): return math.sqrt(math.pow(self.__dx,2)+math.pow(self.__dy,2)+math.pow(self.__dz,2))

  def set_dx(self, value): self.__dx = value
  def set_dy(self, value): self.__dy = value
```

```python
    def set_dz(self, value): self.__dz = value
```

B.1.3 LoadCellData.py

```
from Packet import Packet
import math

class LoadCellData(Packet):
  """ Subclass of Packet for data captured by a load cell.
  Currently holds Fx, Fy, Fz. Modify as needed
  """

  def __init__(self,name="loadcelldata",timestamp=0,fx=0,fy=0,fz=0):
    """ Super class receives timestamp value

    Args:
      name: Name of the entry. Default is "loadcelldata
      timestamp: Timestamp from the Epoch. Default 0 (unset)
      fx: Force in X
      fy: Force in Y
      fz: Force in Z

    Attributes:
      __fx: A float containing force in X
      __fy: A float containing force in Y
      __fz: A float containing force in Z
    """
    Packet.__init__(self, name, timestamp)
    self.__fx = float(fx)
    self.__fy = float(fy)
    self.__fz = float(fz)

  def __str__(self):
    return "Name: %s, Timestamp: %.5f, Fx: %.2f, Fy: %.2f, Fz: %.2f"
        % (Packet.get_name(self), Packet.get_timestamp(self), self.
       __fx, self.__fy, self.__fz)

  def get_fx(self): return self.__fx
  def get_fy(self): return self.__fy
  def get_fz(self): return self.__fz
  def get_fr(self): return math.sqrt(math.pow(self.__fx,2)+math.pow(
     self.__fy,2)+math.pow(self.__fz,2))

  def set_fx(self, value): self.__fx = value
  def set_fy(self, value): self.__fy = value
  def set_fz(self, value): self.__fz = value
```

B.2 Sensors

B.2.1 Sensor.py

```python
class Sensor(object):
  """ Sensor an interface. Sensor instantiations will vary ex. loadcell, optical, thermal.
  A sensor parses data in a text file and stores data packets of useful time stepped information into a list"""

  def __init__(self, name="Unnamed"):
    """
    Args:
      name: Name of the sensor

    Attributes:
      __name: A string containing the name of the sensor. Defaults to "Unnamed"
      __data: A list containing packet objects for each timestamped measurement
    """
    self.__name = name
    self.__data = []

  def __str__(self):
    statement = ""
    for i in range(0,len(self.__data)):
      statement += "Element # "+str(i) + "\n"
      statement += str(self.__data[i]) + "\n"
    return statement

  def get_name(self): return self.__name
  def set_name(self,name): self.__name = name

  def get_data(self): return self.__data
  def set_data(self,data_list): self.__data = data_list
  def addData(self, packet): self.__data.append(packet)

  def size(self): return len(self.__data)
```

B.2.2 LoadCell.py

```
from Sensor import Sensor
from LoadCellData import LoadCellData
import copy
class LoadCell(Sensor):
  """A list containing all LoadCellData related to a specific loadcell"""

  def __init__(self, name="loadcell"):
    Sensor.__init__(self, name)
    self.__data = []

  def __str__(self):
    statement = ""
    for i in range(0,len(self.__data)):
      statement += "Element # "+str(i) + "\n"
      statement += str(self.__data[i]) + "\n"
    return statement

  def get_data(self):
    """ Returns a deep copy of the current data"""
    x = copy.deepcopy(self.__data)
    return x

  def size(self): return len(self.__data)

  def addData(self, packet):
    """ Add packet to load cell. Prints an error message if packet is not of
    type LoadCellData.
    """
    if isinstance(packet,LoadCellData):
      self.__data.append(packet)
    else:
      print "Add FAIL. NOT LOADCELLDATA packet"

  def loadDataFromFile(self, filename):
    """ Load data from an input file filename. Data is expected to be in rows
    in the format: timestamp,fx,fy,fx """
    try:
      reader = open(filename, 'r')
      count = 0
      for row in reader:
```

```
        exploded_row = row.split(",")
        self.__data.append(LoadCellData(filename+"_"+str(count),
           exploded_row[0],exploded_row[1],exploded_row[2],
           exploded_row[3]))
        count = count + 1
    except:
      print "FAIL loading load cell data from file"

  def accept(self, data_exchange):
    """ Visitor design pattern. Allow for a visit. """
    data_exchange.visit(loadcell=self)
```

B.2.3 OpticalTracker.py

```python
from Sensor import Sensor
from OpticalTrackerData import OpticalTrackerData
import copy

class OpticalTracker(Sensor):
  """ Instantiation of Sensor. Stores a list of OpticalTrackerData. """

  def __init__(self, name="Optical Tracker"):
    """ Initializes "Optical Tracker" as name for every new OpticalTracker Sensor

    Attributes:
      __data: List of OpticalTrackerData
    """
    Sensor.__init__(self,name)
    self.__data = []

  def __str__(self):
    statement = ""
    for i in range(0,len(self.__data)):
      statement += "Element # "+str(i) + "\n"
      statement += str(self.__data[i]) + "\n"
    return statement

  def get_data(self):
    """get a deep copy of the current data list"""
    return copy.deepcopy(self.__data)

  def size(self): return len(self.__data)

  def addData(self, packet):
    """ Add a new OpticalTrackerData to the list
    Args:
      packet: OpticalTrackerData to add to the list
    Return:
      Prints message in error
    """
    if isinstance(packet,OpticalTrackerData):
      self.__data.append(packet)
    else:
      print "Add FAIL. Packet NOT OF TYPE OpticalTrackerData"

```

```
  def loadDataFromFile(self, filename):
    """ Loads data from filename into OpticalTrackerData objects
       than stores these objects in the OpticalTracker list.
       Extremely odd format outputted by LABVIEW
    Args:
      loadcell_file: Input file for loadcell
      optical_tracker_file: Input file for optical tracker
    """
    try:
      reader = open(filename, 'r')
      reader.readline() # pass first line
      count = 0
      for row in reader:
        exploded_row = row.split(" ")
        self.__data.append(OpticalTrackerData(filename+"_"+str(count
           ),exploded_row[0].split('\t')[0],exploded_row[3],
           exploded_row[7],exploded_row[11]))
        count = count + 1
    except:
      print "FAIL loading optrical tracker data from file"

  def accept(self, data_exchange):
    """ Visitor design pattern. Allow visit. """
    data_exchange.visit(opticaltracker=self)
```

B.3 Visitor Design Pattern

B.3.1 Reamer.py

```
from OpticalTracker import OpticalTracker
from LoadCell import LoadCell
from DataExchange import DataExchange

class Reamer(object):
  """ A Reamer is a collection of sensors. """

  def __init__(self, opticaltracker=OpticalTracker(), loadcell=
     LoadCell()):
    """
    Args:
      opticaltracker: An optical tracker. Will create empty optical
         tracker by default
      loadcell: A load cell. Will create empty load cell by default

    Attributes:
      __ot: A list of optical trackers
      __lc: A list of load cells
    """
    if not isinstance(opticaltracker, OpticalTracker) or not
       isinstance(loadcell, LoadCell):
      print "Improper initialization of Reamer. Should be types
         OpticalTracker and LoadCell"
    else:
      self.__ot = opticaltracker
      self.__lc = loadcell

  def __str__(self):
    return str(self.__ot) + str(self.__lc)

  def loadDataFromFile(self, loadcell_file, optical_tracker_file):
    """ Loads data from input file into loadcell and opticaltracker
       lists respectfully. Requires one load cell and one optical
       tracker file
    Args:
      loadcell_file: Input file for loadcell
      optical_tracker_file: Input file for optical tracker
    """
    self.__ot.loadDataFromFile(optical_tracker_file)
    self.__lc.loadDataFromFile(loadcell_file)

  def accept(self, data_exchange):
```

```
    """ Visitor design pattern. Allows for a DataExchange object to
       visit to extract information
    Args:
      data_exchange: DataExchange object. Called via: Reamer.visit(
          DataExchange Obj)
    """
    data_exchange.visit(loadcell=self.__lc, opticaltracker=self.__ot
       )
```

B.3.2 DataExchange.py

```
import sys
from LoadCell import LoadCell

class DataExchange(object):
  """ Abstact visitor class"""
  def __init__(self):
    pass

  def visit(self, sensor):
    pass
```

B.3.3 Chart.py

```
import sys
import math
import os
import copy
from LoadCell import LoadCell
from DataExchange import DataExchange
from LoadCellData import LoadCellData
from OpticalTrackerData import OpticalTrackerData
from OpticalTracker import OpticalTracker
from StiffnessElement import StiffnessElement

#from pylab import *
#import matplotlib.pyplot as plt

class Chart(DataExchange):
  def __init__(self):
    self.__lc = [] # list of lists
    self.__ot = [] # list of lists
    self.__x = []
    self.__k = []

  def visit(self, loadcell=None, opticaltracker=None):
    if loadcell != None:
      self.addLoadCell(loadcell.get_data())
    if opticaltracker != None:
      self.addOpticalTracker(opticaltracker.get_data())

  def addLoadCell(self, loadcell):
    self.__lc.append(copy.deepcopy(loadcell))
    for i in range(0,len(loadcell)):
      self.__x.append(copy.deepcopy(loadcell[i]))
    self.__x = sorted(self.__x, key=lambda sensor: sensor.get_timestamp())

  def addOpticalTracker(self, ot):
    self.__ot.append(copy.deepcopy(ot))
    for i in range(0,len(ot)):
      self.__x.append(copy.deepcopy(ot[i]))
    self.__x = sorted(self.__x, key=lambda sensor: sensor.get_timestamp())

  def normalize(self):
    start = float(self.__x[0].get_timestamp())
```

```
    for i in range(0,len(self.__x)):
      self.__x[i].set_timestamp(float(self.__x[i].get_timestamp()) - start)

  def printTimestamp(self):
    for item in self.__x:
      print item.get_timestamp()

  def reset(self):
    self.__x = []

    # add from _ot and _lc
    for loadcell in self.__lc:
      for i in range(0,len(loadcell)):
        self.__x.append(copy.deepcopy(loadcell[i]))
    for opticaltracker in self.__ot:
      for i in range(0,len(opticaltracker)):
        self.__x.append(copy.deepcopy(opticaltracker[i]))

    self.__x = sorted(self.__x, key=lambda sensor: sensor.get_timestamp())

  def printLC(self):
    for item in self.__x:
      if isinstance(item,LoadCellData):
        print item.get_fx()
        print item.get_fy()
        print item.get_fz()

  def printOT(self):
    for item in self.__x:
      if isinstance(item, OpticalTrackerData):
        print item.get_dr()



  def selectRange(self, start, end):
    """ start and end in seconds """
    if start > end:
      print "ERROR: Select Range - Starting value is greater than ending value"
      return
    start_value = float(self.__x[0].get_timestamp())
    end_value = float(self.__x[-1].get_timestamp())

```

```
    start_counter = 0
    end_counter = 0

    for packet in self.__x:
      if packet.get_timestamp() <= start + start_value:
        start_counter = start_counter + 1
      if packet.get_timestamp() <= end + start_value:
        end_counter = end_counter + 1

    self.__x = self.__x[start_counter:end_counter]

  def duration(self):
    return self.__x[-1].get_timestamp() - self.__x[0].get_timestamp()

  def trimStart(self, seconds):
    start = float(self.__x[0].get_timestamp())

    position = 0
    for packets in self.__x:
      if start + seconds < packets.get_timestamp():
        break
      else:
        position = position + 1

    self.__x = self.__x[position:]

  def trimEnd(self, seconds):
    end = float(self.__x[-1].get_timestamp())

    position = 0
    for packets in self.__x:
      if end - seconds < packets.get_timestamp():
        break
      else:
        position = position + 1

    self.__x = self.__x[0:position]

  def autoTrim(self):

    # start point
    start = 1
    if isinstance(self.__x[0], LoadCellData):
      while isinstance(self.__x[start], LoadCellData):
```

```
        start = start + 1
    elif isinstance(self.__x[0], OpticalTrackerData):
      while isinstance(self.__x[start], OpticalTrackerData):
        start = start + 1

    end = len(self.__x) - 1
    if isinstance(self.__x[end], LoadCellData):
      while isinstance(self.__x[end], LoadCellData):
        end = end - 1
    elif isinstance(self.__x[end], OpticalTrackerData):
      while isinstance(self.__x[end], OpticalTrackerData):
        end = end -1

    self.__x = self.__x[start-1:end+2]

  def biasLoadCell(self):
    start_counter = 0
    while not isinstance(self.__x[start_counter], LoadCellData):
      start_counter = start_counter + 1

    start_x = float(self.__x[start_counter].get_fx())
    start_y = float(self.__x[start_counter].get_fy())
    start_z = float(self.__x[start_counter].get_fz())

    for packets in self.__x:
      if isinstance(packets, LoadCellData):
        packets.set_fx(packets.get_fx() - start_x)
        packets.set_fy(packets.get_fy() - start_y)
        packets.set_fz(packets.get_fz() - start_z)

  def biasOpticalTracker(self):
    start_counter = 0
    while not isinstance(self.__x[start_counter], OpticalTrackerData):
      start_counter = start_counter + 1

    start_x = float(self.__x[start_counter].get_dx())
    start_y = float(self.__x[start_counter].get_dy())
    start_z = float(self.__x[start_counter].get_dz())

    for packets in self.__x:
      if isinstance(packets, OpticalTrackerData):
        packets.set_dx(packets.get_dx() - start_x)
        packets.set_dy(packets.get_dy() - start_y)
        packets.set_dz(packets.get_dz() - start_z)
```

```
  def printX(self):
    for item in self.__x:
      print item

  def printOpticalTrackingData(self):
    for sensor in self.__ot:
      for packets in sensor:
        print packets

  def toExcel(self, filename):
    try:

      if not os.path.exists(os.path.abspath(os.path.join(filename, os.pardir))):
        os.makedirs(os.path.abspath(os.path.join(filename, os.pardir)))

      f = open(filename, 'w')
      f.write("Timestamp,Resultant_Force,Resultant_Displacement\n")
      for packet in self.__x:
        line = str(packet.get_timestamp()) + ","
        if isinstance(packet, LoadCellData):
          line += str(packet.get_fr())
        elif isinstance(packet, OpticalTrackerData):
          line += "," +str(packet.get_dr())
        f.write(line+"\n")
    except IOError as e:
      print "Error opening output file for writing...impressive"

  def calculateStiffness(self):
    self.__interpolate()

  def isolate1push(self, threshold):
    if self.__k == []:
      self.__interpolate()

    new_k = []

    for element in self.__k:
      if element.get_fr() > threshold:
        new_k.append(element)

    self.__k = new_k
```

```
def toExcelInterpolated(self, filename):
  try:

    if not os.path.exists(os.path.abspath(os.path.join(filename,
       os.pardir))):
      os.makedirs(os.path.abspath(os.path.join(filename, os.pardir
         )))

    f = open(filename, 'w')
    f.write("Timestamp,Resultant_Force,Resultant_Displacement,
       Stiffness\n")
    if self.__k == []:
      self.__interpolate()

    for element in self.__k:
      line = str(element.get_timestamp()) + ","
      line += str(element.get_fr()) + ","
      line += str(element.get_dr()) + ","
      line += str(element.get_stiffness())
      f.write(line+"\n")

    print "toExcelInterpolated successful"

  except IOError as e:
    print "Error opening output file for writing...impressive"

def toExcelPushSeperated(self, filename):
  try:
    if not os.path.exists(os.path.abspath(os.path.join(filename,
       os.pardir))):
      os.makedirs(os.path.abspath(os.path.join(filename, os.pardir
         )))

    f = open(filename, 'w')
    f.write("Timestamp,Resultant_Force,Resultant_Displacement,
       Stiffness\n")
    if self.__k == []:
      self.__interpolate()

    prev = self.__k[0].get_timestamp()
    for element in self.__k:
      line = str(element.get_timestamp()) + ","
      line += str(element.get_fr()) + ","
      line += str(element.get_dr()) + ","
      line += str(element.get_stiffness())
```

```
        if element.get_timestamp() - prev > .25:
          for i in range(0,5): f.write("\n")
        f.write(line+"\n")
        prev = element.get_timestamp()

      print "toExcelPushSeperated successful"

    except IOError as e:
      print "Error opening output file for writing...impressive"


  def dumpTimeLC(self):
    x = []
    for item in self.__x:
      if isinstance(item,LoadCellData):
        x.append(item.get_timestamp())
    return x

  def dumpTimeOT(self):
    x = []
    for item in self.__x:
      if isinstance(item,OpticalTrackerData):
        x.append(item.get_timestamp())
    return x

  def dumpLC(self):
    x = []
    for item in self.__x:
      if isinstance(item,LoadCellData):
        x.append(item.get_fr())
    return x

  def dumpOT(self):
    x = []
    for item in self.__x:
      if isinstance(item,OpticalTrackerData):
        x.append(item.get_dr())
    return x

  def isolate(self, start, end):
    self.__x = self.__x[start:end]

  ''' 2 pass: 1) loadcell interp 2) optical tracker interp
      returns a list of turpals'''
  def __interpolate(self):
```

```
    stiff_list = []

    # pass 1: loadcell interoplation
    interp1 = False # bool
    interp1_val = 0
    interp2 = False
    interp2_val = 0

    for loc in range(0,len(self.__x)):

      # if found a load cell value
      if isinstance(self.__x[loc], LoadCellData):
        if interp2 == True:
          interp2_val = loc
          self.__interpolateLC(interp1_val, interp2_val, stiff_list)
          interp2 = False

        if interp2 == False:
          interp1 = True
          interp1_val = loc
          stiff_list.append(StiffnessElement(timestamp=self.__x[loc
             ].get_timestamp(), fr=self.__x[loc].get_fr())) # add lc
              data to stiffness element
      else: # optical tracker element
        interp2 = True


    # pass 2: optical tracker interoplation
    interp1 = False
    interp1_val = 0
    interp2 = False
    interp2_val = 0

    for loc in range(0, len(self.__x)):

      # if found an optical tracker value
      if isinstance(self.__x[loc], OpticalTrackerData):
        if interp2 == True:
          interp2_val = loc
          self.__interpolateOT(interp1_val, interp2_val, stiff_list)
          interp2 = False

        if interp2 == False:
          interp1 = True
```

```
          interp1_val = loc
          try:
            stiff_list[loc].set_dr(self.__x[loc].get_dr()) # add lc data to stiffness element
          except IndexError:
            pass

      else: # optical tracker element
        interp2 = True

    self.__k = stiff_list

  ''' interpolate values using timestamp '''
  def __interpolateLC(self, start, end, stiff_list):
    for element in range(start+1, end):

      # y = y_a + (y_b - y_a)[(x-x_a)/(x_b-x_a)]
      interp_fr = self.__x[start].get_fr() + ((self.__x[end].get_fr() - self.__x[start].get_fr()) * (( self.__x[element].get_timestamp() - self.__x[start].get_timestamp()) / (self.__x[end].get_timestamp() - self.__x[start].get_timestamp())))
      #print "Element: " + str(element) + " Value: " + str(interp_fr)

      stiff_list.append(StiffnessElement(timestamp=self.__x[element].get_timestamp(), fr=interp_fr))

  ''' interpolate values using timestamp '''
  def __interpolateOT(self, start, end, stiff_list):
    for element in range(start+1, end):

      # y = y_a + (y_b - y_a)[(x-x_a)/(x_b-x_a)]
      interp_dr = self.__x[start].get_dr() + ((self.__x[end].get_dr() - self.__x[start].get_dr()) * (( self.__x[element].get_timestamp() - self.__x[start].get_timestamp()) / (self.__x[end].get_timestamp() - self.__x[start].get_timestamp())))
      #print "Element OT: " + str(element) + " Value: " + str(interp_dr)
      try:
        stiff_list[element].set_dr(interp_dr)
      except IndexError:
        pass # last element value is undetermined
```

B.3.4 StiffnessElement.py

```

class StiffnessElement(object):

  def __init__(self, timestamp=0, fr=0, dr=0):
    self.__timestamp = timestamp
    self.__fr = fr
    self.__dr = dr

  def __str__(self):
    return "Timestamp: %s, Fr: %.2f, Dr: %.2f" % (self.__timestamp,
       self.__fr, self.__dr)

  def set_fr(self, value):
    self.__fr = value

  def set_dr(self, value):
    self.__dr = value

  def set_timestamp(self, value):
    self.__timestamp = value

  def get_timestamp(self): return self.__timestamp
  def get_fr(self): return self.__fr
  def get_dr(self): return self.__dr

  def get_stiffness(self):
    if self.__dr!=0:
      return (self.__fr/self.__dr) * 1000 # N/m
    else:
      return 0
```

B.4 GUI

B.4.1 gui.py

```python
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.widgets import Button

class Index:

  # takes a chart object and preforms necessary actions
  def __init__(self, chart, ax, output_file):
    self.__chart = chart
    self.__ax = ax
    self.__output_file = output_file
    self.__plot_init()

    # data for line values
    x = self.__chart.dumpTimeLC()
    y = self.__chart.dumpLC()

    x2 = self.__chart.dumpTimeOT()
    y2 = self.__chart.dumpOT()

    # plotting data
    self.__l, = self.__ax.plot(x, y, lw=2)
    self.__l2, = self.__ax.plot(x2, y2, lw=2)

    self.__isolated = 0

  def __reload(self):

    self.__l.set_xdata(self.__chart.dumpTimeLC())
    self.__l.set_ydata(self.__chart.dumpLC())

    self.__l2.set_xdata(self.__chart.dumpTimeOT())
    self.__l2.set_ydata(self.__chart.dumpOT())

    self.__ax.relim()
    self.__ax.autoscale()

    plt.draw()

  def autoTrim(self, event):
    print "Autotrim"
    self.__chart.autoTrim()
```

```
    self.__reload()

  def normalize(self, event):
    print "Normalize"
    self.__chart.normalize()
    self.__reload()

  def biasLoadCell(self, event):
    print "Bias Load Cell"
    self.__chart.biasLoadCell()
    self.__reload()

  def biasOpticalTracker(self, event):
    print "Bias Optical Tracker"
    self.__chart.biasOpticalTracker()
    self.__reload()

  def isolate(self, event):
    print "Isolate"
    self.__chart.isolate1push(4)
    self.__reload()

  def toExcelInterpolated(self, event):
    print "to Excel Interpolated"
    self.__chart.toExcelInterpolated(self.__output_file)
    self.__reload()

  def toExcelPushSeperated(self, event):
    print "to Excel Push Seperated"
    self.__chart.toExcelPushSeperated(self.__output_file)
    self.__reload()

  def reset(self, event):
    print "Reset"
    self.__chart.reset()
    self.__reload()

  def __plot_init(self, fontsize=12):
    self.__ax.plot([1])
    self.__ax.locator_params(nbins=3)
    self.__ax.set_xlabel('Time', fontsize=fontsize)
    self.__ax.set_ylabel('Force or Displacement', fontsize=fontsize)
    self.__ax.set_title('Reamer Results', fontsize=fontsize)

  def isolate(self, val):
```

```
    print int(val)
    if int(val) != self.__isolated:
      self.__chart.isolate1push(val)
      print "change"
      self.__isolated = int(val)

    self.__reload()
```

B.4.2 main_gui.py

```
from Reamer import Reamer
from Chart import Chart

import argparse
import sys

from gui import Index
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.widgets import Button, Slider


############# Command Line Arguments ################

# get command line arguments input, output, loadcel
parser = argparse.ArgumentParser()
parser.add_argument("-lc", "--loadcell", help="file containing load
   cell data")
parser.add_argument("-ot", "--opticaltracker", help="file containing
    optical tracker data")
parser.add_argument("-o", "--output", help="output file")
args = parser.parse_args()

# error check command line arguments
if not args.loadcell:
  parser.error("please include loadcell input file")
if not args.opticaltracker:
  parser.error("please include optical tracker input file")
if not args.output:
  parser.error("pleasae include output file")


############ Visitor Pattern Setup #############

# Create objects reamer and chart
r = Reamer()
c = Chart()


# load data from file(s) into reamer
r.loadDataFromFile(args.loadcell, args.opticaltracker)

# setup visitor
```

```
r.accept(c)


############ GUI Setup #########


fig = plt.figure()

ax1 = plt.subplot2grid((5, 5), (0, 0), colspan=5, rowspan=3)
pos_trim = plt.subplot2grid((5, 5), (3, 0), colspan=1, rowspan=1)
pos_normalize = plt.subplot2grid((5, 5), (3, 1), colspan=1, rowspan
    =1)
pos_bias_lc = plt.subplot2grid((5, 5), (3, 2), colspan=1, rowspan=1)
pos_bias_ot = plt.subplot2grid((5, 5), (3, 3), colspan=1, rowspan=1)
#pos_isolate = plt.subplot2grid((5, 5), (3, 4), colspan=1, rowspan
    =1)
pos_excel_interp = plt.subplot2grid((5, 5), (4, 0), colspan=1,
    rowspan=1)
pos_excel_seperated = plt.subplot2grid((5, 5), (4, 1), colspan=1,
    rowspan=1)
pos_reset = plt.subplot2grid((5, 5), (3, 4), colspan=1, rowspan=1)

plt.tight_layout()

# button names
b_trim = Button(pos_trim, 'Trim',  hovercolor='0.25')
b_normalize = Button(pos_normalize, 'Normalize', hovercolor='0.25')
b_bias_lc = Button(pos_bias_lc, 'Bias LC', hovercolor='0.25')
b_bias_ot = Button(pos_bias_ot, 'Bias OT', hovercolor='0.25')
b_excel_interp = Button(pos_excel_interp, 'Excel', hovercolor='0.25'
    )
b_excel_seperated = Button(pos_excel_seperated, 'Excelv2.0',
    hovercolor='0.25')
b_reset = Button(pos_reset, 'Reset', hovercolor='0.25')

#sl_isolate = Slider(pos_isolate, 'Isolate', valmin=0, valmax=5,
    valinit=0, valfmt='%d')


# button on_click
callback = Index(c, ax1, args.output)

b_trim.on_clicked(callback.autoTrim)
b_normalize.on_clicked(callback.normalize)
b_bias_lc.on_clicked(callback.biasLoadCell)
```

```
b_bias_ot.on_clicked(callback.biasOpticalTracker)
b_excel_interp.on_clicked(callback.toExcelInterpolated)
b_excel_seperated.on_clicked(callback.toExcelPushSeperated)
b_reset.on_clicked(callback.reset)
#sl_isolate.on_changed(callback.isolate)

############### Running GUI ##############

plt.show()
```