

**Sandia National Laboratories**

Operated for the U.S. Department of Energy by

Sandia Corporation

Albuquerque, New Mexico 87185

date: August 24, 2015

to: Distribution

from: Dan Rohe, Org. 1522, MS-0557

subject: Documentation and Instructions for Running Two Python Scripts that Aid in Setting up 3D Measurements using the Polytec 3D Scanning Laser Doppler Vibrometer

This memo describes two scripts that were written by the author to aid in setting up a Polytec 3D Scanning Laser Doppler Vibrometer.

1 Introduction

Sandia National Laboratories has recently purchased a Polytec 3D Scanning Laser Doppler Vibrometer for vibration measurement. This device has proven to be a very nice tool for making vibration measurements, and has a number of advantages over traditional sensors such as accelerometers. The non-contact nature of the laser vibrometer means there is no mass loading due to measuring the response. Additionally, the laser scanning heads can position the laser spot much more quickly and accurately than placing an accelerometer or performing a roving hammer impact. The disadvantage of the system is that a significant amount of time must be invested to align the lasers with each other and the part so that the laser spots can be accurately positioned. The Polytec software includes a number of nice tools to aid in this procedure; however, certain portions are still tedious. Luckily, the Polytec software is readily extensible by programming macros for the system, so tedious portions of the procedure can be made easier by automating the process.

The Polytec Software includes a WinWrap (similar to Visual Basic) editor and interface to run macros written in that programming language. The author, however, is much more proficient in Python, and the latter also has a much larger set of libraries that can be used



Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

to create very complex macros, while taking advantage of Python's inherent readability and maintainability.

2 Python and the Polytec System

The author knows very little about how the computer that shipped with the laser system is configured in terms of system registry and drivers, so a conservative approach was taken when selecting the Python distribution to install. Vanilla Python [1], available from www.python.org has an installation procedure that modifies the system registry and changes environment variables such as `PATH`. The author thinks it would be possible that registry changes and environment variable could interfere with the behavior of the laser system. Though unlikely, without knowing the details of how Polytec drives its lasers, it seems to be not worth the risk, especially when better options are available.

2.1 WinPython

WinPython [2] is a Python distribution for Windows that includes the vanilla Python available from the Python website but also includes a number of other nice tools and libraries. Additionally, WinPython does not 'install' itself on the computer, it simply unpacks into a folder. Therefore it is much less likely¹ to cause issues by modifying system information. Another advantage is it contains 64-bit libraries whereas other Python distributions only carry 32-bit libraries.

WinPython comes with many tools that are helpful when programming in Python. It includes an integrated development environment (IDE) called Spyder that can be configured to mirror the MATLAB graphical user interface (GUI) if the user desires. This may be helpful in migrating to Python from MATLAB. WinPython also includes a special command prompt that can be used to run Python scripts, called `WinPython Command Prompt.exe`. Since WinPython does not modify the operating system environment, Python is not on the system path and therefore cannot be called from the command prompt by simply typing in `python`. To get around this, WinPython includes this command prompt in which all of the paths are set correctly. These and a number of other tools are located in the main folder in which WinPython was unpacked. The author suggests pinning `Spyder.exe` and `WinPython Command Prompt.exe` to the start menu for easy access, as he uses these two tools most often.

2.2 Interfacing WinPython with the Polytec Software

The Polytec Acquisition software has a Component Object Model (COM) interface that can be used to interact with the software using external tools. The `win32com` module included

¹Note that the author does not provide any warranty or guarantee that WinPython will not cause harm to the computer or laser system. Polytec mentions in their manual that installing other software may disrupt function of the laser and is not advised, therefore we are going slightly contrary to their instructions here. Python is a programming language, and by using it improperly, a user can delete files, modify the system, and do other harm to the operating system if not careful. As with any tool downloaded from the Internet, one should perform the usual anti-virus scans to ensure the file is clean.

in the WinPython distribution is designed to work with COM objects. To connect to the Acquisition software, one must dispatch a win32com client to connect to the instance of the Acquisition software. The Application object is the COM object that is used access data in the Acquisition software. Code in Listing 1 can be used as an entry point to the Acquisition application. The object hierarchy and available functions and data are documented in the PSV Basic Engine Manual, which is found in the PSV folder in the start menu. The data and options available are similar but not identical to those exposed through the Polytec File Access Software interface [3].

```
1 # Import the module
2 import win32com.client
3 # Connect to the application
4 com_handle = win32com.client.Dispatch('PSV.AcquisitonInstance')
5 # Get the application object
6 application_object = h.GetApplication(com_handle,10000)
7 # Query the application object for additional data and operations
8 acquisition_object = application_object.Acquisition
9 settings_object = application_object.Settings
10 ...
11 ...
```

Listing 1: Example code to access data in the Polytec Acquisition software via the Application object.

With the interfaces exposed through the application object and its children, fairly complex operations can be driven with Python, including toggling the laser beams on and off, pointing and focusing the laser beams at locations, creating 2D and 3D alignments, creating measurement points, and starting a scan.

3 2D Alignment Script

The script documented in this section is not necessarily specific to the Polytec software, as it basically moves the mouse and clicks in a pattern. This script is useful when performing a 2D alignment. It is presented in Appendix A in Listing 2.

Historically, the 2D alignment has been one of the most tedious and labor intensive parts of setting up a scan with the laser. The operator had to move the lasers and click on the laser spot on the camera image, mapping laser deflections to positions on the camera image. For complicated or curved parts a large number of points was usually needed to get good triangulation over the entire part.

With more recent versions of the software, much of the labor intensive nature of the 2D alignment was alleviated; the software included a ‘laser spot finder’ which could automatically detect the spot on the camera image. The user was still required to move the laser, however, and this process remained tedious if a large number of points were required. The user may move and click the mouse upwards of 100 times for each laser head. To the author, it seemed that the Polytec software had solved the hard problem (detecting the laser spot) but neglected the easy problem of moving and clicking the mouse. An initial idea was to just

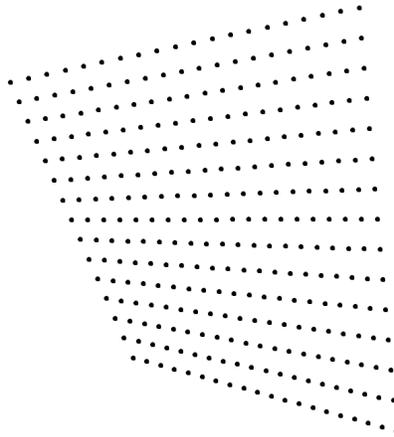


Figure 1: Results of the clicking script on Microsoft Paint. One can see the 20×15 grid of points, which took only a small amount of time to place compared to the amount of time it would require to click those positions manually.

move the mouse and click uniformly over a rectangular area, but this would be limiting for angled parts such as a nose cone, where the projection is more triangular than rectangular.

The author took inspiration from finite elements for this script, and clicked in an area defined by a quadrilateral, using the typical finite element shape functions to warp the clicks so they were approximately parallel with the edges. With this implementation, most shapes could be covered well, with the user filling in sections that were not covered by the script.

The script relies on an adequate 2D alignment already being in place (typically just the four corners of the quadrilateral are sufficient) so laser position due to the click ends up being near where the mouse was clicked. The user then specifies the four corners of the quadrilateral and the script moves the mouse and clicks for the user. Functionality can be demonstrated in any software that records clicks, such as Microsoft Paint, where clicks are recorded by drawing points on the image, see Figure 1.

When using the clicking script in the Polytec Acquisition software, there is no feedback from the software to the script. The script simply clicks and relies on the laser finder to find the spot. If the laser finder has trouble finding the spot, that alignment point may not be placed. This can occur if the surface is too dark or the camera is too far away. The author has had improved laser spot finder performance when using the external camera rather than the camera in the top laser. Note that the clicks are stored in a buffer, so the script may outrun the laser spot finder if it struggles to find a point. The author has found that even if the laser script outruns the laser finder, good results are still obtained. Figure 2 shows the results on a conical test article where the clicking script was used two times.

One should be aware that the clicking script controls the mouse, and when the first click lands in the Acquisition software window, the window that is running the script loses focus. This makes it very difficult to stop the script once it has started. For this reason, the user is queried whether or not they want to proceed before the script takes control of the mouse.

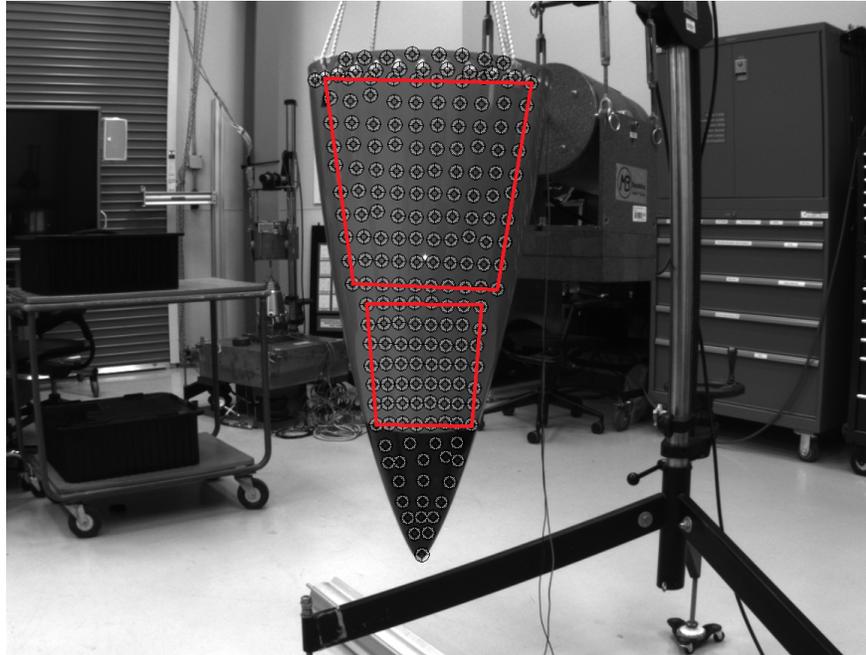


Figure 2: Example of 2D clicking script providing a fine resolution of 2D alignment points. The areas covered by the clicking script are shown outlined in red (two sections), and the remaining points were filled in manually.

It gives a very optimistic² estimate of the time required to complete.

The script is located at `D:\dprohe\Python\rectangle_click_spread.py` on the 3D SLDV computer owned by Sandia National Laboratories Department 01522. To see the instructions for the script, one should open the command prompt discussed in Section 2 and type in `python D:\dprohe\Python\rectangle_click_spread.py -h` into the prompt. The `-h` flag stands for ‘help’, and displays the help instructions. The instructions are reproduced below for completeness:

This script clicks in a quadrilateral pattern, given the four corners. To use this script to automate 2D alignment in the PSV software,

1. Enter 2D alignment
2. Set Laser of interest (Ctrl+1 to Ctrl+3)
3. Create a rough initial alignment (four corners is minimum)
4. Ensure Automatic 2D alignment is activated with High Contrast Display on.
5. Ensure the video of the PSV software is visible below the terminal window.
6. Run script.

²This estimate is the time that would be required if the laser spot finder found each spot instantaneously, which does not happen. Even if all of the spots have not been found by this time, the user should still get control of the mouse back after this time has elapsed.

7. When the script queries for the corners, hover the mouse over the corner when you press enter, DO NOT CLICK as it will remove focus from the terminal window.
8. Use caution when starting the script. It is very difficult to stop once started as it controls the mouse and the terminal window loses focus.

To Run: `python D:\dprobe\Python\rectangle_click_spread.py [-h] -x <x_interval> -y <y_interval>`

- `-h`: displays this help
- `-x`: sets the number of clicks along the horizontal axis
- `-y`: sets the number of clicks along the vertical axis

4 3D Alignment Script

3D alignment is very important when doing a 3D scan because it is what ensures that the laser spots are collocated at a point on the test article. There are two approaches to 3D alignment. The first is performing the 3D alignment directly on the part using known locations on the part. Errors in this alignment will result in errors in the relative positions of the lasers, which will make the spots not be collocated. A second approach is to use the reference object to do the 3D alignment and then transforming the coordinate system to the part coordinate system. This is the approach considered here.

Generally a good 3D alignment can be achieved when using the reference object (on the order of a few tenths of a millimeter). The goal is then to transform the coordinate system of the reference object into the coordinate system of the part. This requires knowledge of points on the part in both the reference coordinate system and the part coordinate system. To get a point in the reference coordinate system, one would generally place a measurement point and then go into the `Modify 3D Coordinates...` dialog box to position the laser at the correct point. This dialog box is very powerful in that it can modify the 3D coordinates, set the distance to the lasers, and initiate a video triangulation. The drawback, however, is that it requires the mouse to interact with the dialog box. If the user is staring closely at the test article to determine whether or not the laser spots are collocated, he or she may end up repeatedly walking back and forth between the test article and the computer monitor to make fine adjustments to the measurement point position, and this can be tedious and time consuming. A better solution would be the user standing in front of the test article with the keyboard in his or her hand, modifying the position of the coordinates as he or she is looking at the part.

The 3D alignment script is meant to provide this functionality. The script positions the lasers at $(x, y, z) = (-150 \text{ mm}, 150 \text{ mm}, 0)$, which is the top left corner of the reference object. The script then lets the user move the spot at which the lasers are pointing using an intuitive set of keys on the keyboard. The user can modify how much the point moves with each keypress, focus the lasers, and place a measurement point all without returning to

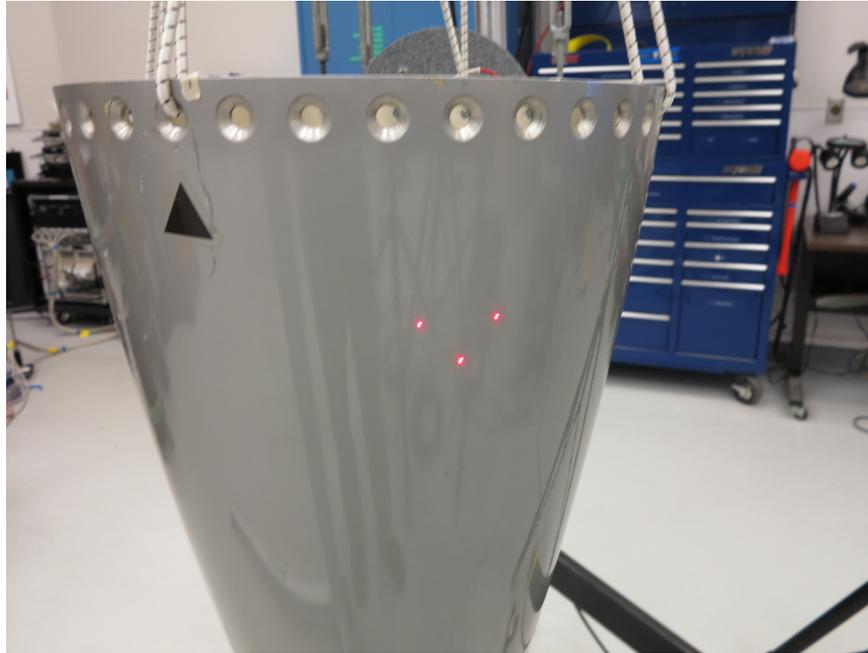


Figure 3: Example of laser spots not yet collocated. The lasers are pointed at a point in space in front of the part, so they appear as three separate spots on the surface behind the point. A similar situation would occur if the point were behind the test article.

the monitor and mouse. These measurement points can then be used by the Polytec-written Macro to transform coordinates.

The script is located at `D:\dprobe\Python\drive_lasers_3D.py` on the 3D SLDV computer owned by Sandia National Laboratories Department 01522. It is also reproduced in Appendix B in Listing 3. To see the instructions for the script, one should open the command prompt discussed in Section 2 and type in `python D:\dprobe\Python\drive_lasers_3D.py -h` into the prompt. The `-h` flag stands for ‘help’, and displays the help instructions. The instructions are reproduced below for completeness:

This script moves the laser beam and places measurements points. It should be run **AFTER** 3D alignment is performed on the reference object. It provides more convenient keys to control the laser position in x,y,z space. The controls are as follows (note: keypad keys require numlock ON as the 5 key is used):

- Num 4: decrease X coordinate
- Num 6: increase X coordinate
- Num 2: decrease Y coordinate
- Num 8: increase Y coordinate
- Num 0: decrease Z coordinate
- Num 5: increase Z coordinate

Figure 4: Animation of aligning laser spots with landmark on the test structure (JavaScript must be enabled to play Animation, and it may be necessary to use Adobe Reader).

- Num .: report current laser location to the command prompt
- Num +: Increases distance the position moves with each key press
- Num -: Decreases distance the position moves with each key press
- f: focus the laser beams
- Enter: Position a measurement point at the coordinates specified
- Esc: Exits the script

Positioning of the measurement point on the video depends on the 2D alignment.

If an instance of the PSV acquisition software is not running, this script will start one and the default settings will be loaded.

Note that if the initial alignment using the reference object is poor, the laser spots will not be on top of one another at any point in space. If the user is having trouble getting all of the spots on the part to be on top of one another, the user should consider improving this initial alignment.

5 Conclusions

This memo documents two scripts written in Python that the author has used successfully to improve the process of aligning the laser beams. The scripts are located at `D:\dprobe\Python\` on the 3D SLDV computer owned by Sandia National Laboratories Department 01522, and are also reported in the appendices.

References

- [1] “Python.” <http://www.python.org/>. Accessed: 2015-07-29.
- [2] “WinPython.” <http://winpython.sourceforge.net/>. Accessed: 2015-07-29.
- [3] “Polytec file access software.” <http://www.polytec.com/us/products/vibration-sensors/vibrometer-software/polytec-file-access-software/>. Accessed: 2015-01-29.

A rectangle_click_spread.py

```
1 # -*- coding: utf-8 -*-
2 """
3 Created on Tue Apr 28 11:39:07 2015
4
5 @author: dprohe
6 """
7
8 import win32api, win32con
9 import time
10 import sys
11 import getopt
12
13 click_delay = 0.01
14 release_delay = 0.1
15 move_delay = 0.1
16
17 x_interval = 10
18 y_interval = 10
19
20 help_string = '''
21 This script clicks in a quadrilateral pattern, given the four corners.
22 To use this script to automate 2D alignment in the PSV software,
23 1. Enter 2D alignment
24 2. Set Laser of interest (Ctrl+1 to Ctrl+3)
25 3. Create a rough initial alignment (four corners is minimum)
26 4. Ensure Automatic 2D alignment is activated with High Contrast
27    Display on.
28 5. Ensure the video of the PSV software is visible below the terminal
29    window.
30 6. Run script.
31 7. When the script queries for the corners, hover the mouse over the
32    corner when you press enter, DO NOT CLICK as it will remove focus
33    from the terminal window.
34 8. Use caution when starting the script. It is very difficult to stop
35    once started as it controls the mouse and the terminal window loses
36    focus.
37
38 To Run:
39 python '+__file__+' [-h] -x <x_interval> -y <y_interval>
40
41 -h: displays this string
42 -x: sets the number of clicks along the horizontal axis
43 -y: sets the number of clicks along the vertical axis
44 '''
45
46 try:
47     opts, args = getopt.getopt(sys.argv[1:], "hx:y:")
48 except getopt.GetoptError:
49     print(help_string)
50
51 for opt, arg in opts:
52     if opt == '-h':
53         print(help_string)
54         sys.exit()
55     elif opt == '-x':
56         x_interval = int(arg)
57     elif opt == '-y':
58         y_interval = int(arg)
59
60 def click(x, y):
61     win32api.SetCursorPos((x, y))
62     time.sleep(click_delay)
63     win32api.mouse_event(win32con.MOUSEEVENTF_LEFTDOWN, x, y, 0, 0)
64     time.sleep(release_delay)
```

```

65     win32api.mouse_event(win32con.MOUSEEVENTF_LEFTUP,x,y,0,0)
66     time.sleep(move_delay)
67
68     raw_input('Move Mouse to Top Left Corner then Press Enter')
69     top_left = win32api.GetCursorPos()
70     #top_left = (1139, 281)
71     print(top_left)
72     raw_input('Move Mouse to Top Right Corner then Press Enter')
73     top_right = win32api.GetCursorPos()
74     #top_right = (1669, 292)
75     print(top_right)
76     raw_input('Move Mouse to Bottom Left Corner then Press Enter')
77     bottom_left = win32api.GetCursorPos()
78     #bottom_left = (1177, 747)
79     print(bottom_left)
80     raw_input('Move Mouse to Bottom Right Corner then Press Enter')
81     bottom_right = win32api.GetCursorPos()
82     #bottom_right = (1697, 714)
83     print(bottom_right)
84
85     time.sleep(1)
86
87     o = bottom_left
88     u = [i-j for (i,j) in zip(bottom_right,bottom_left)]
89     v = [i-j for (i,j) in zip(top_left,bottom_left)]
90     w = [k-l-(i+j) for (i,j,k,l) in zip(u,v,top_right,bottom_left)]
91
92     def shape(xi,yi):
93         return (o[0]+u[0]*xi+v[0]*yi+w[0]*xi*yi,
94               o[1]+u[1]*xi+v[1]*yi+w[1]*xi*yi)
95
96     if 'y' == raw_input('This will take at least '+str((click_delay+release_delay+move_delay)*
97                       x_interval*y_interval)+' seconds, proceed? (y/n)'):
98         points = []
99         put_points = []
100        for i,xi in enumerate([float(k)/(x_interval-1) for k in range(x_interval)]):
101            for j,yi in enumerate([float(k)/(y_interval-1) for k in range(y_interval)]):
102                if (i == 0 and j == 0) or (i == x_interval-1 and j == 0) or (i == 0 and j ==
103                    y_interval-1) or (i == x_interval-1 and j == y_interval-1):
104                    put_points.append([int(k) for k in shape(xi,yi)])
105                    click(*[int(k) for k in shape(xi,yi)])
106                else:
107                    points.append([int(k) for k in shape(xi,yi)])
108        while len(points) > 0:
109            sorted_points = [(sum([1/sum([(float(ppi)-float(pi))**2 for ppi,pi in zip(pp,p)]) for pp
110                in put_points]),index) for index,p in enumerate(points)]
111            min_index = min(sorted_points)[1]
112            pt = points.pop(min_index)
113            click(*pt)
114            put_points.append(pt)

```

Listing 2: Python script that clicks in a quadrilateral shape.

B drive_lasers_3D.py

```
1 # -*- coding: utf-8 -*-
2 """
3 Created on Tue Jun 30 12:38:02 2015
4
5
6
7 @author: Dan Rohe
8 """
9
10 instructions = '''
11 This script moves the laser beam and places measurements points.
12 It should be run *AFTER* 3D alignment is performed on the
13 reference object. It provides more convenient keys to control
14 the laser position in x,y,z space. The controls are as follows:
15
16 *Note: Keypad keys require numlock ON as the 5 key is used*
17 Num 4: decrease X coordinate
18 Num 6: increase X coordinate
19 Num 2: decrease Y coordinate
20 Num 8: increase Y coordinate
21 Num 0: decrease Z coordinate
22 Num 5: increase Z coordinate
23 Num .: report current laser location to the command prompt
24 Num +: Increases distance the position moves with each key press
25 Num -: Decreases distance the position moves with each key press
26 f: focus the laser beams
27 Enter: Position a measurement point at the coordinates specified
28 Esc: Exits the script
29
30 Positioning of the measurement point on the video depends on the
31 2D alignment.
32
33 If an instance of the PSV acquisition software is not running,
34 this script will start one and the default settings will be
35 loaded.'''
36
37 print(instructions)
38
39 speed_index = 7
40 speeds = [0.001,0.01,0.1,0.25,0.5,1.0,2.0,3.0,5.0,10.0,25.0,50.0] # mm
41
42 laser_position = [-150.0,150.0,0.0] # mm, starting laser position
43
44 import win32com.client
45 from msvcrt import getch
46 import os
47 import sys
48
49 if '-h' in sys.argv:
50     sys.exit()
51
52 h = win32com.client.Dispatch('PSV.AcquisitionInstance')
53
54 app = h.GetApplication(True,10000)
55
56 acq = app.Acquisition
57
58 settings = app.Settings
59
60 info = acq.Infos
61
62 alignments = info.alignments
63
64 scanhead_devices = info.ScanHeadDevicesInfo.ScanHeadDevices
```

```

65
66 alignments3D = alignments.alignments3D
67
68 alignments2D = alignments.alignments2D
69
70 for i in range(alignments3D.count):
71     scanX,scanY,distance = alignments3D[i].Coord3DToScanner(laser_position[0]/1000,
72     laser_position[1]/1000,laser_position[2]/1000,0.0,0.0,0.0)
73     scanhead_devices[i].ScanHeadControl.ScannerControl.SetBeamPosition(scanX,scanY)
74
75 save_location = os.environ["Temp"]+"\drive_lasers_3D.set"
76
77 while True:
78     displacement = speeds[speed_index]
79     key = ord(getch())
80     if key == 27: # Esc, exit
81         break
82     elif key == 48: # 0, Z-
83         laser_position[2] -= displacement
84     elif key == 53: # 5, Z+
85         laser_position[2] += displacement
86     elif key == 50: # 2, Y-
87         laser_position[1] -= displacement
88     elif key == 56: # 8, Y+
89         laser_position[1] += displacement
90     elif key == 52: # 4, Z-
91         laser_position[0] -= displacement
92     elif key == 54: # 6, Z+
93         laser_position[0] += displacement
94     elif key == 43: # +, speed up
95         speed_index += 1
96         if speed_index == len(speeds):
97             speed_index = len(speeds)-1
98         print("Speed changed to "+str(speeds[speed_index])+" mm")
99     elif key == 45: # -, slow down
100         speed_index -= 1
101         if speed_index < 0:
102             speed_index = 0
103         print("Speed changed to "+str(speeds[speed_index])+" mm")
104     elif key == 13: # enter, place measurement point
105         print("Adding Measurement Point at "+str(laser_position))
106         # According to the PSV examples, I need to save and rewrite it to add a
107         # measurement point.
108         settings.Save(save_location)
109         h_fas = win32com.client.Dispatch("Polyfile.Polyfile")
110         h_fas.Readonly = 0
111         h_fas.Open(save_location)
112         meas_point = h_fas.Infos.MeasPoints.Add()
113         vidx = 0
114         vidy = 0
115         for i in range(alignments3D.count):
116             mirror_x,mirror_y = scanhead_devices[i].ScanHeadControl.ScannerControl.
117             GetBeamPosition(0.0,0.0)
118             thisvidx,thisvidy = alignments2D[i].ScannerToVideo(mirror_x,mirror_y,0.0,0.0)
119             vidx += thisvidx/alignments3D.count
120             vidy += thisvidy/alignments3D.count
121             meas_point.SetVideoXY(vidx,vidy)
122             meas_point.SetCoordXYZ(*[l/1000 for l in laser_position])
123             h_fas.Save()
124             h_fas.Close()
125             settings.Load(save_location,312+68)
126             os.remove(save_location)
127     elif key == 102: # F, focus
128         print('Focusing...')
129         for i in range(alignments3D.count):
130             info.Vibrometers[0].Controllers[i].SensorHeads[0].StartAutoFocus()
131     elif key == 46: # ., report position

```

```
130     print('Lasers at: '+str(laser_position)+' mm')
131     else:
132         continue
133     for i in range(alignments3D.count):
134         scanX, scanY, distance = alignments3D[i].Coord3DToScanner(laser_position[0]/1000,
135         laser_position[1]/1000, laser_position[2]/1000, 0.0, 0.0, 0.0)
135         scanhead_devices[i].ScanHeadControl.ScannerControl.SetBeamPosition(scanX, scanY)
```

Listing 3: Python script that drives the lasers and points them at a position in space.

Internal Distribution:

MS-0557	Mike Arviso	Org. 1522
MS-0557	Fred Bauer	Org. 1522
MS-0557	David Epp	Org. 1522
MS-0557	Anthony Gomez	Org. 1522
MS-0557	Patrick Hunter	Org. 1522
MS-0557	Randy Mayes	Org. 1522
MS-0557	Adam Moya	Org. 1522
MS-0557	Ben Pacini	Org. 1522
MS-0557	Robert Wauneka	Org. 1522

External Distribution:

Andrew Jessop
Lawrence Livermore National Laboratory
jessop5@llnl.gov

David Macknelly
Atomic Weapons Establishment
David.Macknelly@awe.co.uk