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...
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 likely 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].

```python
# Import the module
import win32com.client

# Connect to the application
com_handle = win32com.client.Dispatch('PSV.AcquisitionInstance')

# Get the application object
application_object = h.GetApplication(com_handle, 10000)

# Query the application object for additional data and operations
acquisition_object = application_object.Acquisition
settings_object = application_object.Settings

...  
```

**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
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.
It gives a very optimistic\textsuperscript{2} estimate of the time required to complete.

The script is located at $\texttt{dZ\textbackslash dprohe\textbackslash Python\textbackslash rectangle_click_spread.py}$ 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 $\texttt{python dZ\textbackslash dprohe\textbackslash Python\textbackslash rectangle_click_spread.py -h}$ 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.

\textsuperscript{2}This 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:\dprohe\Python\rectangle_click_spread.py [-h] -x \x_interval\ -y \y_interval\n
- -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 script is located at D:\dprohe\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:\dprohe\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:\probe\Python\ on the 3D SLDV computer owned by Sandia National Laboratories Department 01522, and are also reported in the appendices.
References


# rectangle_click_spread.py

```python
# -*- coding: utf-8 -*-

Created on Tue Apr 28 11:39:07 2015
@author: dprohe

import win32api, win32con
import time
import sys
import getopt

click_delay = 0.01
release_delay = 0.1
move_delay = 0.1
x_interval = 10
y_interval = 10

help_string = ''
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.
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 'r__file__'+ [h] -x <x_interval> -y <y_interval>

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

try:
    opts, args = getopt.getopt(sys.argv[1:], "hx:y:")
except getopt.GetoptError:
    print(help_string)
for opt, arg in opts:
    if opt == '-h':
        print(help_string)
        sys.exit()
    elif opt == '-x':
        x_interval = int(arg)
    elif opt == '-y':
        y_interval = int(arg)

def click(x, y):
    win32api.SetCursorPos((x, y))
    time.sleep(click_delay)
    win32api.mouse_event(win32con.MOUSEEVENTF_LEFTDOWN, x, y, 0, 0)
    time.sleep(release_delay)
```

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.
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 'r__file__'+ [h] -x <x_interval> -y <y_interval>
```

-h: displays this string
-x: sets the number of clicks along the horizontal axis
-y: sets the number of clicks along the vertical axis
Listing 2: Python script that clicks in a quadrilateral shape.
B  drive_lasers_3D.py

```python
# -*- coding: utf-8 -*-

Created on Tue Jun 30 12:38:02 2015

@author: Dan Rohe

instructions = '''
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
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.''

print(instructions)

speed_index = 7
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
laser_position = [-150.0, 150.0, 0.0] # mm, starting laser position

import win32com.client
from msvcrt import getch
import os
import sys

if '--h' in sys.argv:
    sys.exit()

h = win32com.client.Dispatch('PSV.AcquisitionInstance')
app = h.GetApplication(True, 10000)
acq = app.Acquisition
settings = app.Settings
info = acq.Infos
alignments = info.alignments
scanhead_devices = info.ScanHeadDevicesInfo.ScanHeadDevices
```
for i in range alignments3D.count:
    scanX, scanY, distance = alignments3D[i].Coord3DToScanner(laser_position[0]/1000,
        laser_position[1]/1000, laser_position[2]/1000, 0.0, 0.0, 0.0)
    scanhead_devices[i].ScanHeadControl.ScannerControl.SetBeamPosition(scanX, scanY)

while True:
    displacement = speeds[speed_index]
    key = ord(getch())
    if key == 27:  # Esc, exit
        break
    elif key == 48:  # 0, Z-
    elif key == 53:  # 5, Z+
    elif key == 50:  # 2, Y-
        laser_position[1] -= displacement
    elif key == 56:  # 8, Y+
        laser_position[1] += displacement
    elif key == 52:  # 4, Z-
        laser_position[0] -= displacement
    elif key == 54:  # 6, Z+
        laser_position[0] += displacement
    elif key == 43:  # +, speed up
        speed_index += 1
        if speed_index == len(speeds):
            speed_index = 0
            print("Speed changed to " + str(speeds[speed_index]) + " mm")
    elif key == 45:  # -, slow down
        speed_index -= 1
        if speed_index < 0:
            speed_index = 0
            print("Speed changed to " + str(speeds[speed_index]) + " mm")
    elif key == 113:  # enter, place measurement point
        print("Adding Measurement Point at " + str(laser_position))
        # According to the PSV examples, I need to save and rewrite it to add a
        # measurement point.
        settings.Save(save_location)
        h_fas = win32com.client.Dispatch("Polyfile.Polyfile")
        h_fas.Readonly = False
        h_fas.Open(save_location)
        meas_point = h_fas.Infos.MeasPoints.Add()
        vidx = 0
        vidy = 0
        for i in range alignments3D.count:
            mirror_x, mirror_y = scanhead_devices[i].ScanHeadControl.ScannerControl.
            GetBeamPosition(0.0, 0.0, 0.0)
            thisvidx, thisvidy = alignments2D[i].ScannerToVideo(mirror_x, mirror_y, 0.0, 0.0)
            vidx += thisvidx alignments3D.count
            vidy += thisvidy alignments3D.count
            meas_point.SetVideoXY(vidx, vidy)
            meas_point.SetCoordXYZ(*[1/1000 for l in laser_position])
        h_fas.Save()
        h_fas.Close()
        settings.Save(save_location, 312+68)
        os.remove(save_location)
    elif key == 102:  # F, Focus
        print("Focusing...")
    else:
        print("Focusing...")
    for i in range alignments3D.count:
        info.Vibrometers[i].Controllers[i].SensorHeads[i].StartAutoFocus()
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