ENME 489Y – Remote Sensing: Spring 2018

Department of Mechanical Engineering

Due Date	Thursday, April 5th, 2018 by 9:30 am
Submission Information	• Submit email response to Question #1 via mitchels@umd.edu • Submit .pdf response to Question #4 via Gradescope

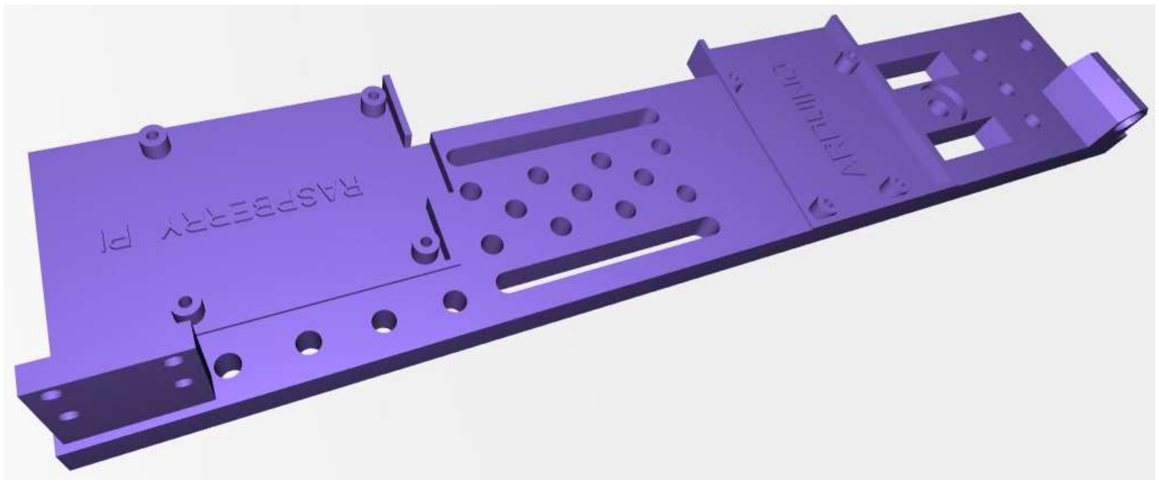
Introduction to field deployable lidar, Inertial Measurement Unit (IMU), and pySerial

***General Reminder*: keep recording pictures & videos of each aspect of your project to edit/stitch together in your semester project video and final project presentation!**

Question #1

In the Spring 2017 edition of 489Y, students were permitted free creative reign to design and integrate their lidar sensor. This semester, students are **once again free** to design and integrate their sensor as they desire. However, we are also providing a standardized, 3D-printable mount design should students prefer to use a more direct method. Solid model .stl files are provided via ELMS > Modules > Project.

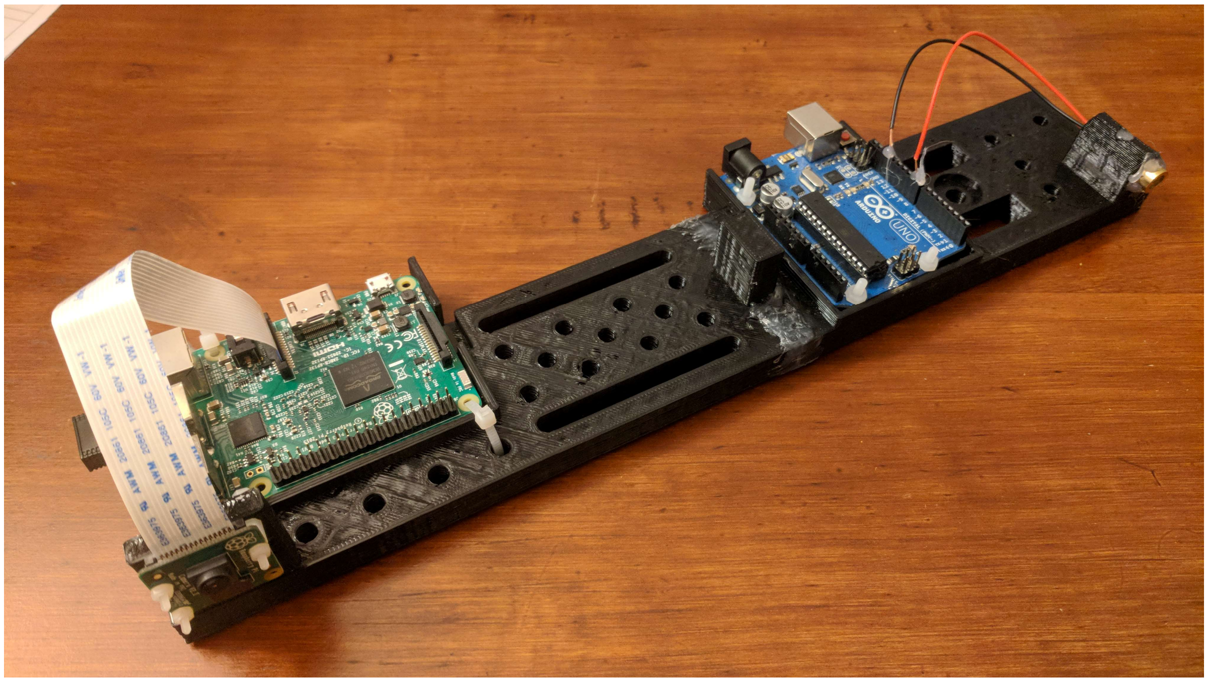
The standardized mount consists of a single holder plate. This plate has mounting holes for an Arduino Uno, Raspberry Pi, Raspberry Pi camera, and a LED line laser:

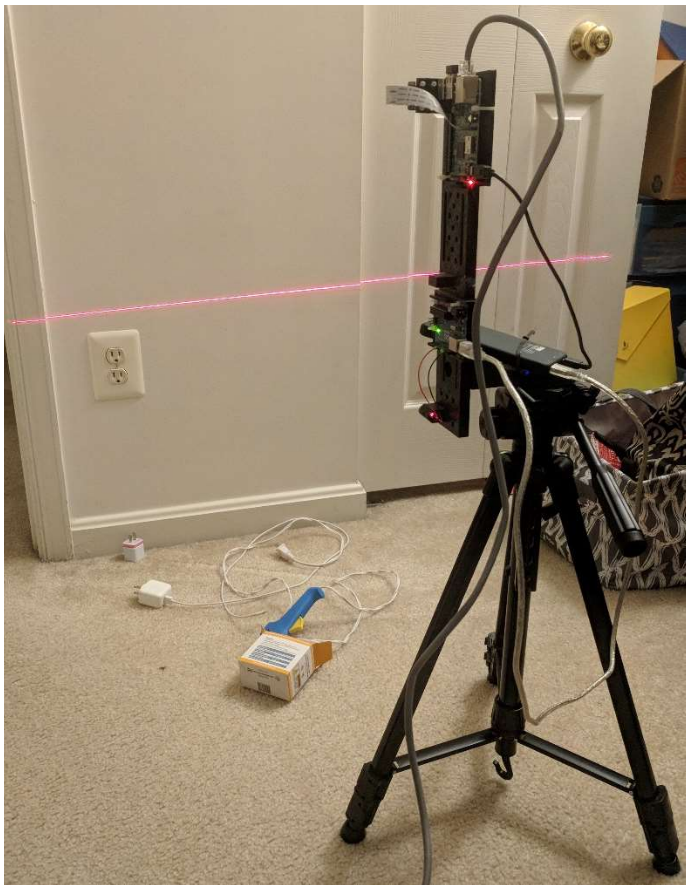


The plate also has miscellaneous holes and slots allocated as cable tie downs, etc., along with two rectangular cutouts that can be used to tie the plate into a tripod via the 3D-printable connecting plate shown at right.

While it is ***preferable*** to print the plate as **one solid piece** (think structural rigidity), students also have access to a 2-part solid model file(s) that can be epoxied together, in the event your 3D printer cannot handle the size of the single plate design.

Here is one incarnation of the assembled, aligned instrument. Again - students are **not required** to use the standardized mount should they prefer more creative approaches to integrating their lidar sensor.





Assemble your field-deployable lidar sensor (Dr. Mitchell's is shown at right).

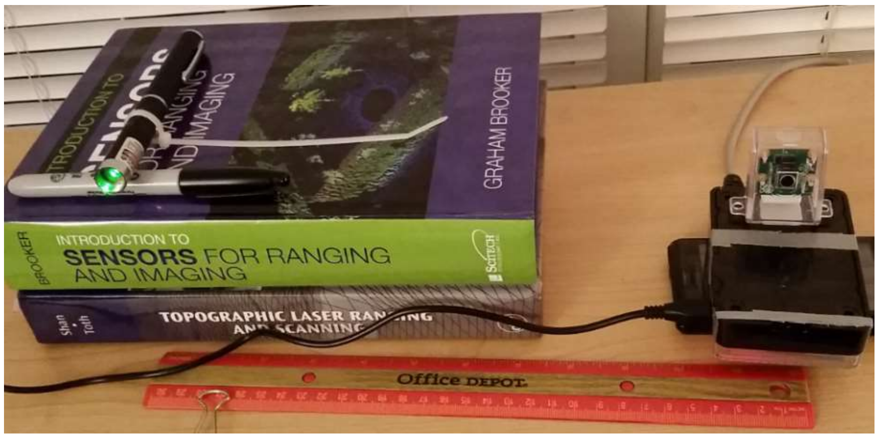
Record a **minimum 30 second video clip** of yourself describing the sensor configuration and demonstrating any pertinent features (feel free to use your cell phone camera at this point). Be sure to point out the essential hardware components (e.g. transmitter, receiver, software interfaces, etc.) and consider discussing why you designed the sensor the way you did.

Upload the video to your YouTube account, then email mitchels@umd.edu the link to the video.

Consider taking multiple videos **as you assemble the lidar, to edit/stitch together in your semester project video!**

<u>Question #2</u> (nothing to submit)

In Assignment #4 we explored the geometrical range measurement using a single beam LED laser and a Raspberry Pi camera. By calibrating the bistatic setup we were able to collect rudimentary range measurements. Here we expand upon the single beam technique, replacing the single beam laser with a line laser, which dramatically improves the data acquisition rate of the lidar.



Single beam laser bistatic configuration (left) vs. line laser bistatic configuration (right)

This sensor configuration requires angular alignment of the line laser with the Raspberry Pi camera. To begin, head over to GitHub and download *laser_alignment_image.py*:

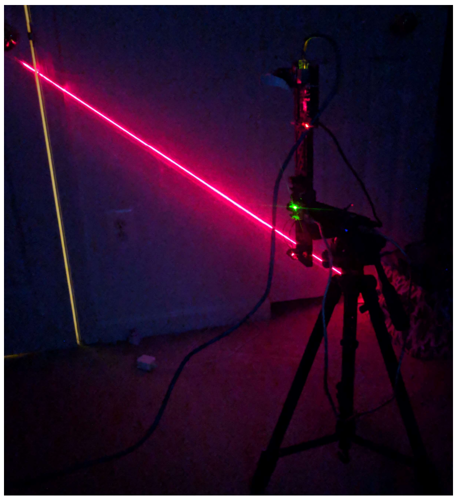
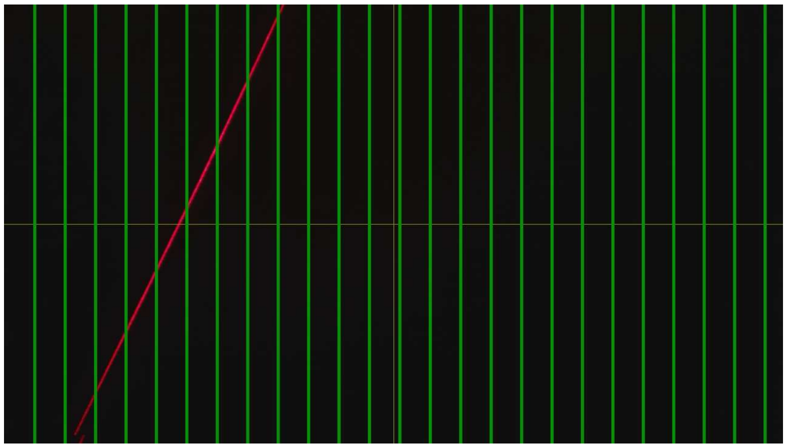
https://github.com/oneshell/enme489y

This script opens the RPi camera feed to support the alignment process. The *laser_alignment_image.py* script plots a series of vertical green lines along the camera frame. The spacing of these lines can be adjusted as desired. Begin the alignment process by connecting the LED line laser to a 5V power source, such as the Arduino Uno or a GPIO pin on the RPi.

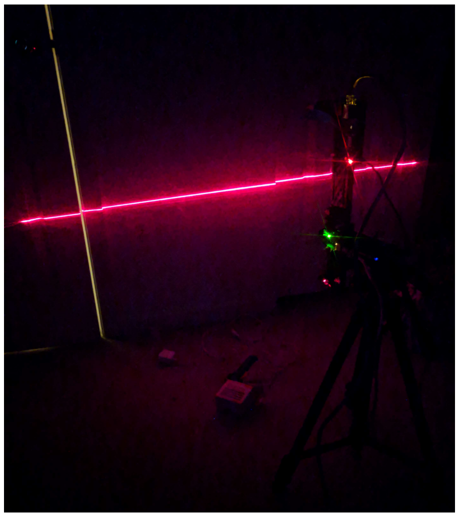
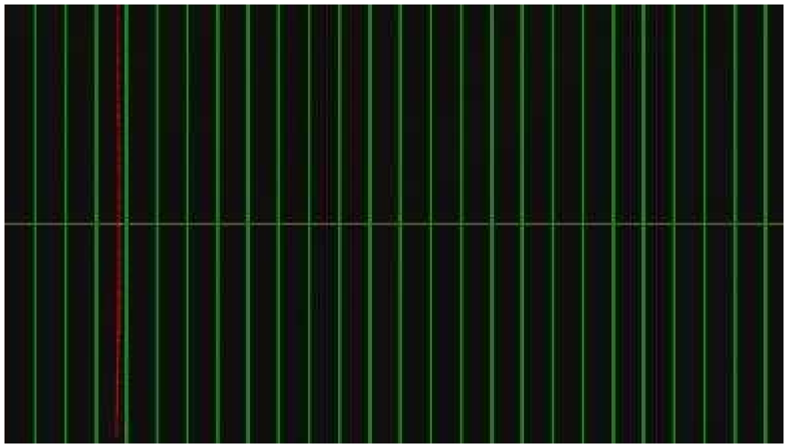
Note: regardless of how you have mechanically integrated your sensor, **it is imperative that the RPi camera be *rigidly* mounted prior to beginning the alignment process**. Once the lidar is aligned, the axis of the camera and the axis of the laser should remain aligned and parallel throughout the semester, so ensure the RPi camera is securely mounted before continuing!

While monitoring the camera's video feed, rotate and align the line laser such that the red laser line is parallel to the green alignment lines. See example images on the next page. Leave the video stream running and **secure the laser in place** (e.g. using epoxy).

After properly securing the laser, ensure the alignment of the laser line has not shifted. Double check that both the RPi camera and line laser are rigidly secured in their respective mounts. Pressing the **q** key will end the video feed.

Line laser orientation at the beginning of the alignment process (not currently aligned). Note the lack of parallelism between the red laser line and the green vertical alignment lines.

Line laser aligned. Red laser line now parallel with green vertical alignment lines.

It is **imperative that the RPi camera and laser be *rigidly* mounted** to maintain their alignment and ensure proper functioning of the lidar sensor throughout the semester.

Compared to the single laser beam configuration of Assignment #4, integration of the line laser is equivalent to stacking multiple single beam LED lasers on top of each other to form an array of beams or, in this case, a line.

In effect, each row of pixels of the camera frame is now a single laser beam lidar!

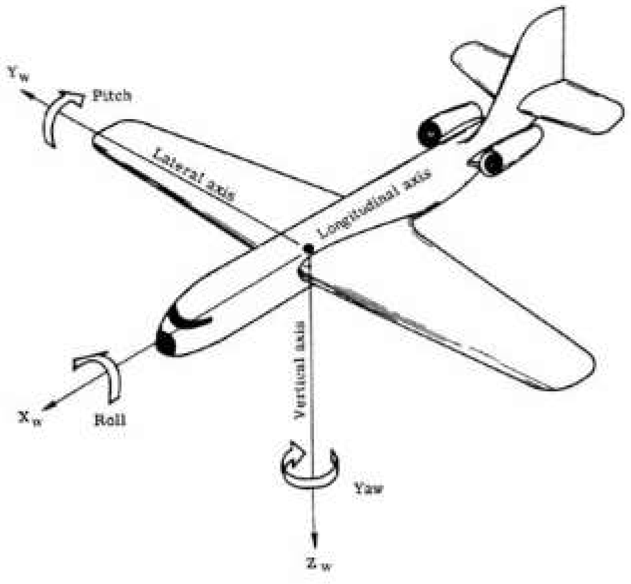
Thought: consider **recording videos** of the laser alignment process, both with your cell phone *and* with your RPi, to use in your semester video!

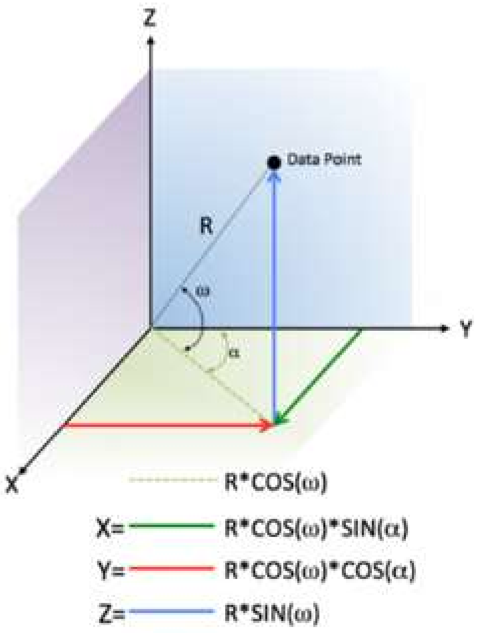
Question #3 (nothing to submit)

As discussed in lecture, to geolocate lidar data requires the sensor to record:

1. Range data, describing the distance between the sensor and the target
2. GPS data, describing the latitude-longitude-altitude position of the sensor
3. IMU data, describing the pitch-roll-yaw pointing of the sensor

With these sources of information, each lidar range measurement can be transformed into a XYZ coordinate in 3-dimensional space.





For 489Y, we collected GPS data in Assignment #1 to identify the location of each target on campus. ***No further GPS data is required***.

Initial range measurements were recorded with the prototype single beam lidar in Assignment #4. Now that we have the line laser installed and aligned, let's evaluate how to record IMU data with our lidar sensor. To measure the pointing angle of the RPi lidar, ***we have two options***:

1. Use the accelerometer in your cell phone via an app such as Bubble Level (for Android) or equivalent.
2. Use an accelerometer chip such as the Analog Devices ADXL327, which can be found in the Digilent Analog parts kit (Dr. Mitchell also has a very limited supply of chips available for loan).

If using your cell phone, ***securely*** mount the phone to the lidar mounting plate. Enter the angle reading provided by the app when saving range data in Python (more information on this coming in Assignment #6).

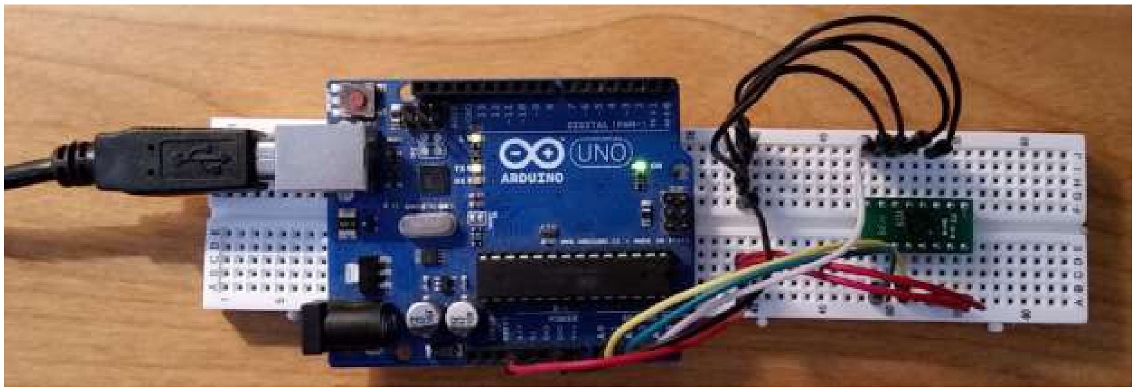
If using the Analog Devices chip, begin by reading through the datasheet available at:

http://www.analog.com/media/en/technical-documentation/data-sheets/ADXL327.pdf

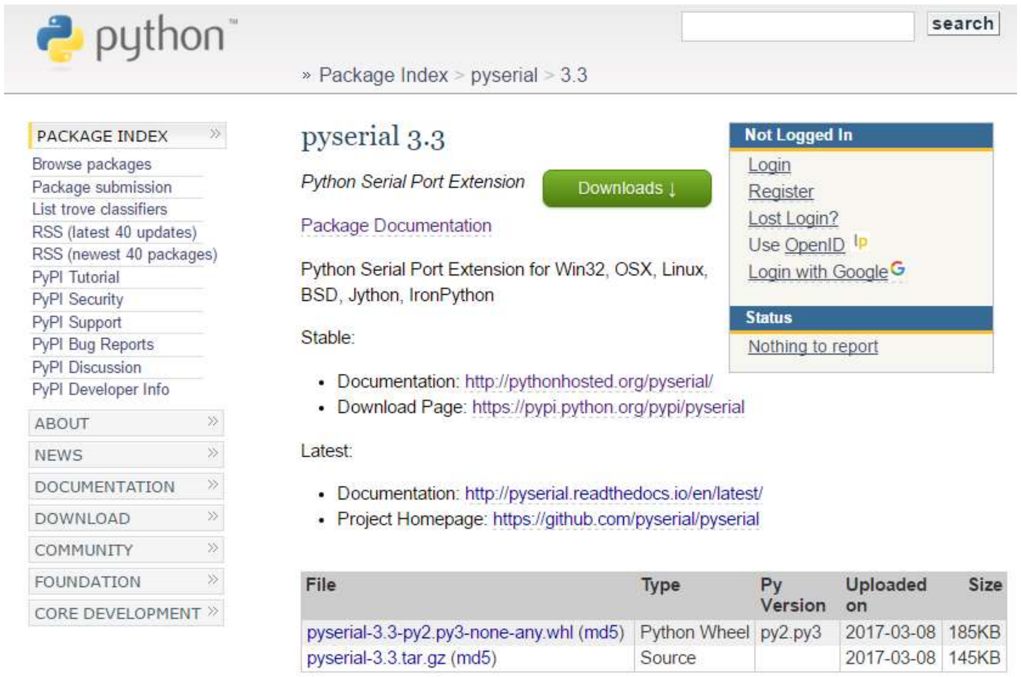
To better understand how the accelerometer chip functions, checkout:

http://www.digikey.com/videos/en/vp/5026329017001

Let's begin by connecting the chip to an Arudino and ensuring we can read the Arduino serial stream via the RPi using Python. Download and read through Assignment #5 – Supplemental.pdf for a description of the ADXL327 accelerometer, including how to properly wire up the chip to an Arduino. Using the Arduino code provided at the end of this assignment, ensure that the accelerometer chip is functioning properly by opening the Serial Monitor and viewing the x, y, z angle outputs.



Open up a browser in your RPi and navigate to https://pypi.python.org/pypi/pyserial, then download **pyserial-3.3.tar.gz** to your RPi. The file will be located in the \Downloads folder on your RPi.


» Package Index > pyserial > 3.3

PACKAGE INDEX
Browse packages
Package submission
List trove classifiers
RSS (latest 40 updates)
RSS (newest 40 packages)
PyPI Tutorial
PyPI Security
PyPI Support
PyPI Bug Reports
PyPI Discussion
PyPI Developer Info
ABOUT
NEWS
DOCUMENTATION
DOWNLOAD
COMMUNITY
FOUNDATION
CORE DEVELOPMENT

pyserial 3.3

Python Serial Port Extension

Package Documentation

Python Serial Port Extension for Win32, OSX, Linux, BSD, Jython, IronPython

Stable:

- Documentation: http://pythonhosted.org/pyserial/
- Download Page: https://pypi.python.org/pypi/pyserial

Latest:

- Documentation: http://pyserial.readthedocs.io/en/latest/
- Project Homepage: https://github.com/pyserial/pyserial

File	Type	Py Version	Uploaded on	Size
pyserial-3.3-py2.py3-none-any.whl (md5)	Python Wheel	py2.py3	2017-03-08	185KB
pyserial-3.3.tar.gz (md5)	Source		2017-03-08	145KB

Not Logged In
Login
Register
Lost Login?
Use OpenID
Login with Google

Status
Nothing to report

To complete the install of pySerial, checkout: https://www.youtube.com/watch?v=I7fyFb7gNEQ

Alternatively, consider running a *pip install pyserial* to install pySerial on your RPi.

Due to the time lag associated with OpenCV, there is no need to transfer data quickly over the serial stream. Try setting your baud rate to 1200 in both the Arduino and pySerial programs.

The following is a good tutorial describing serial communication between Arduino and USB:

https://oscarliang.com/connect-raspberry-pi-and-arduino-usb-cable/

Here is a similar Arduino reference: http://playground.arduino.cc/Interfacing/Python

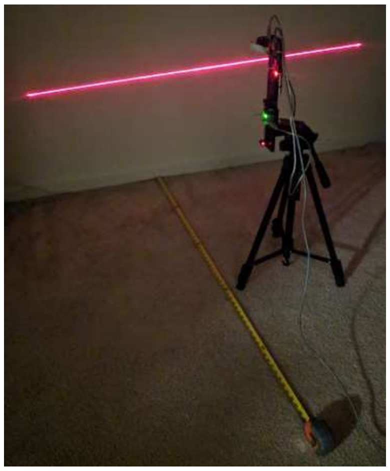
With the ADXL327 powered and serial communication between the ADXL327 and Arudino established, create a new file called *serialtest.py* by typing *nano serialtest.py* into a terminal window. Input the following code and save the file (press 'Y') when exiting:

```
import serial

ser = serial.Serial('/dev/ttyACM0', 1200)

while 1:
        print(ser.readline())
```

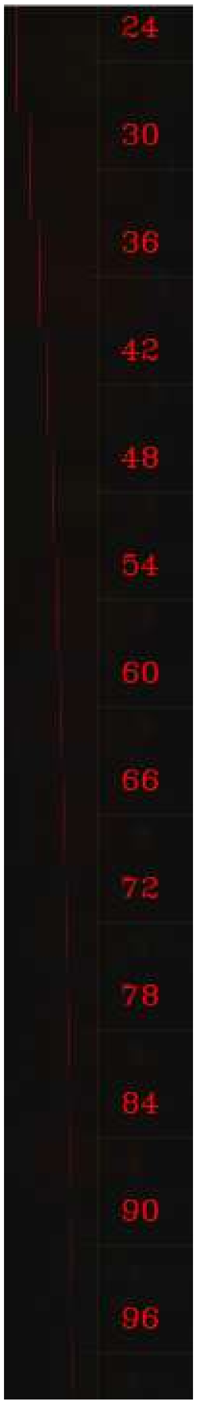
Execute the code by typing *python serialtest.py* into the command line and ensure that you are properly reading and displaying the Arduino output serial stream properly by tiling the accelerometer.

With the bistatic lidar and the ADXL327 (or cell phone equivalent), each lidar range measurement can be transformed into a XYZ coordinate!

Question #4



Here we utilize the triangulation lidar range measurement from Assignment #4 and upgrade the sensor calibration from a single beam laser to the line laser. Identify a wall free of obstructions and setup the lidar orthogonal to the wall surface (image at right).



Head over to GitHub and download *laser_alignment_image_blank.py*:

https://github.com/oneshell/enme489y

This script opens the RPi camera feed to support the alignment process. The *laser_alignment_image_blank.py* script asks the user to "**Please enter distance from wall, in inches:** ". Enter the distance measured with a tape measure and hit the Enter key. The script will echo the distance value you entered into the terminal window (to confirm the number) then opens the video feed.

Notice that the distance number has now been added to the top right corner of the video feed via the **cv2.putText( )** command.

With the instrument aligned orthogonal to the wall, hit the **m** key. This will save a .jpg file on your RPi, the filename of which is defined by the distance from the wall. To confirm you have successfully recorded the .jpg, in a terminal window type:

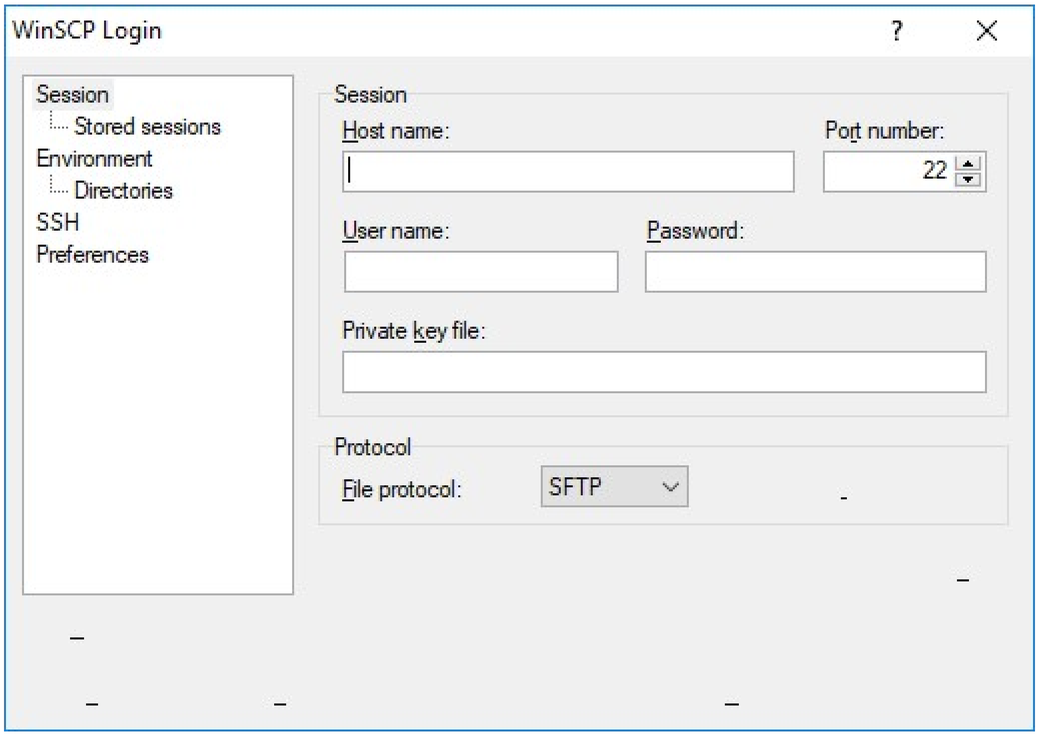
```
ls

xdg-open distance_value_here.jpg
```

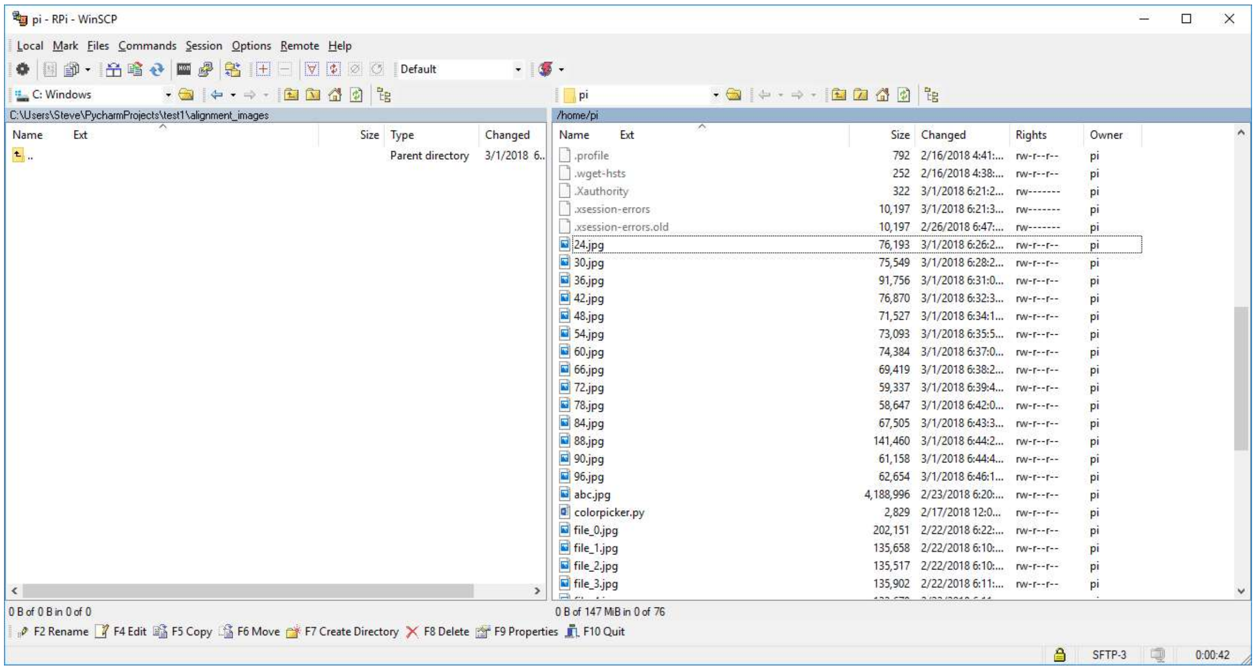
This will open the saved .jpg file.

Run the *laser_alignment_image_blank.py* script and save .jpg files while ranging to the wall, for known distances from 0 to 10 feet, in increments of 6 inches. An example of this sequence is shown in the collage at left. Once complete, type **ls** into a terminal window and confirm the .jpg files have successfully saved to your RPi.

Transfer all of the .jpg files created during this exercise from your RPi to your laptop. This can be accomplished in several ways:

1. Upload images to Gmail and email files
2. Insert a USB into your RPi and copy files
3. My favorite: ***FTP into the RPi*** from your laptop using an FTP client such as WinSCP. The Host name will be the IP address of your RPi. The User name and Password are the same as when using Putty and VNC Viewer (typically **pi** and **raspberr** or **raspberry**).





C:\Users\Steve\PycharmProjects\test1\alignment_images

Name	Ext	Size	Type	Changed
..			Parent directory	3/1/2018 6..

0 B of 0 B in 0 of 0

/home/pi

Name	Ext	Size	Changed	Rights	Owner
.profile		792	2/16/2018 4:41:...	rw-r--r--	pi
.wget-hsts		252	2/16/2018 4:38:...	rw-r--r--	pi
.Xauthority		322	3/1/2018 6:21:2...	rw-------	pi
.xsession-errors		10,197	3/1/2018 6:21:3...	rw-------	pi
.xsession-errors.old		10,197	2/26/2018 6:47:...	rw-------	pi
24.jpg		76,193	3/1/2018 6:26:2...	rw-r--r--	pi
30.jpg		75,549	3/1/2018 6:28:2...	rw-r--r--	pi
36.jpg		91,756	3/1/2018 6:31:0...	rw-r--r--	pi
42.jpg		76,870	3/1/2018 6:32:3...	rw-r--r--	pi
48.jpg		71,527	3/1/2018 6:34:1...	rw-r--r--	pi
54.jpg		73,093	3/1/2018 6:35:5...	rw-r--r--	pi
60.jpg		74,384	3/1/2018 6:37:0...	rw-r--r--	pi
66.jpg		69,419	3/1/2018 6:38:2...	rw-r--r--	pi
72.jpg		59,337	3/1/2018 6:39:4...	rw-r--r--	pi
78.jpg		58,647	3/1/2018 6:42:0...	rw-r--r--	pi
84.jpg		67,505	3/1/2018 6:43:3...	rw-r--r--	pi
88.jpg		141,460	3/1/2018 6:44:2...	rw-r--r--	pi
90.jpg		61,158	3/1/2018 6:44:4...	rw-r--r--	pi
96.jpg		62,654	3/1/2018 6:46:1...	rw-r--r--	pi
abc.jpg		4,188,996	2/23/2018 6:20:...	rw-r--r--	pi
colorpicker.py		2,829	2/17/2018 12:0...	rw-r--r--	pi
file_0.jpg		202,151	2/22/2018 6:22:...	rw-r--r--	pi
file_1.jpg		135,658	2/22/2018 6:10:...	rw-r--r--	pi
file_2.jpg		135,517	2/22/2018 6:10:...	rw-r--r--	pi
file_3.jpg		135,902	2/22/2018 6:11:...	rw-r--r--	pi

0 B of 147 MiB in 0 of 76

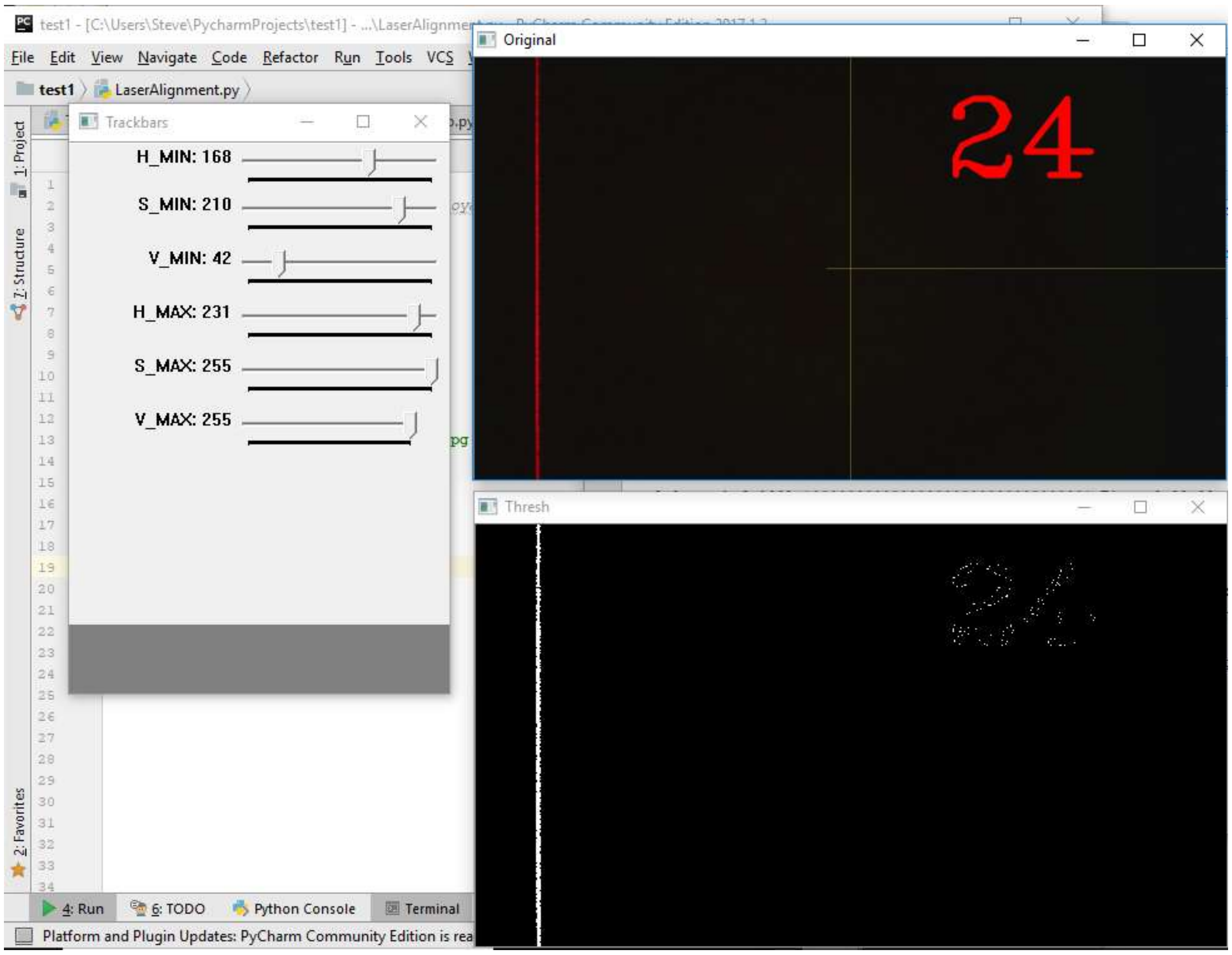
With all .jpg files transferred to your laptop, specifically your working directory in PyCharm, head over to GitHub and download *laser_alignment.py*:

https://github.com/oneshell/enme489y

Let's explore this script, which will be executed in PyCharm. When writing this Assignment, Dr. Mitchell placed his alignment images in a folder on his laptop named **alignment_images**. Therefore, the *laser_alignment.py* script initially finds all .jpg files located in the /alignment_images folder. Consider updating the code as required for your setup:

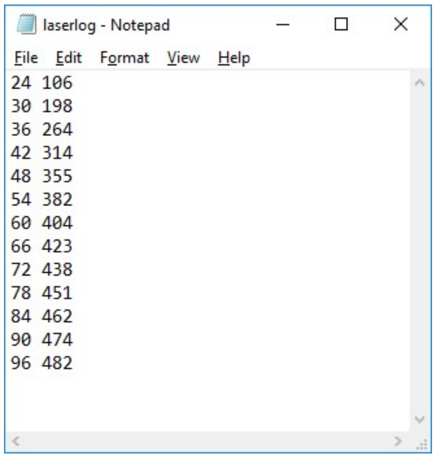
```
files = glob.glob('alignment_images/*.jpg')
```

The script then defines the lower and upper color bounds to apply when masking each image. These can be tuned using the *colorpicker.py* script:


test1 - [C:\Users\Steve\PycharmProjects\test1] - ...\LaserAlignment.py
File Edit View Navigate Code Refactor Run Tools VCS
test1
LaserAlignment.py
Trackbars
Original
Thresh
24
1: Project
7: Structure
2: Favorites
4: Run
6: TODO
Python Console
Terminal
Platform and Plugin Updates: PyCharm Community Edition is rea

H_MIN: 168
S_MIN: 210
V_MIN: 42
H_MAX: 231
S_MAX: 255
V_MAX: 255

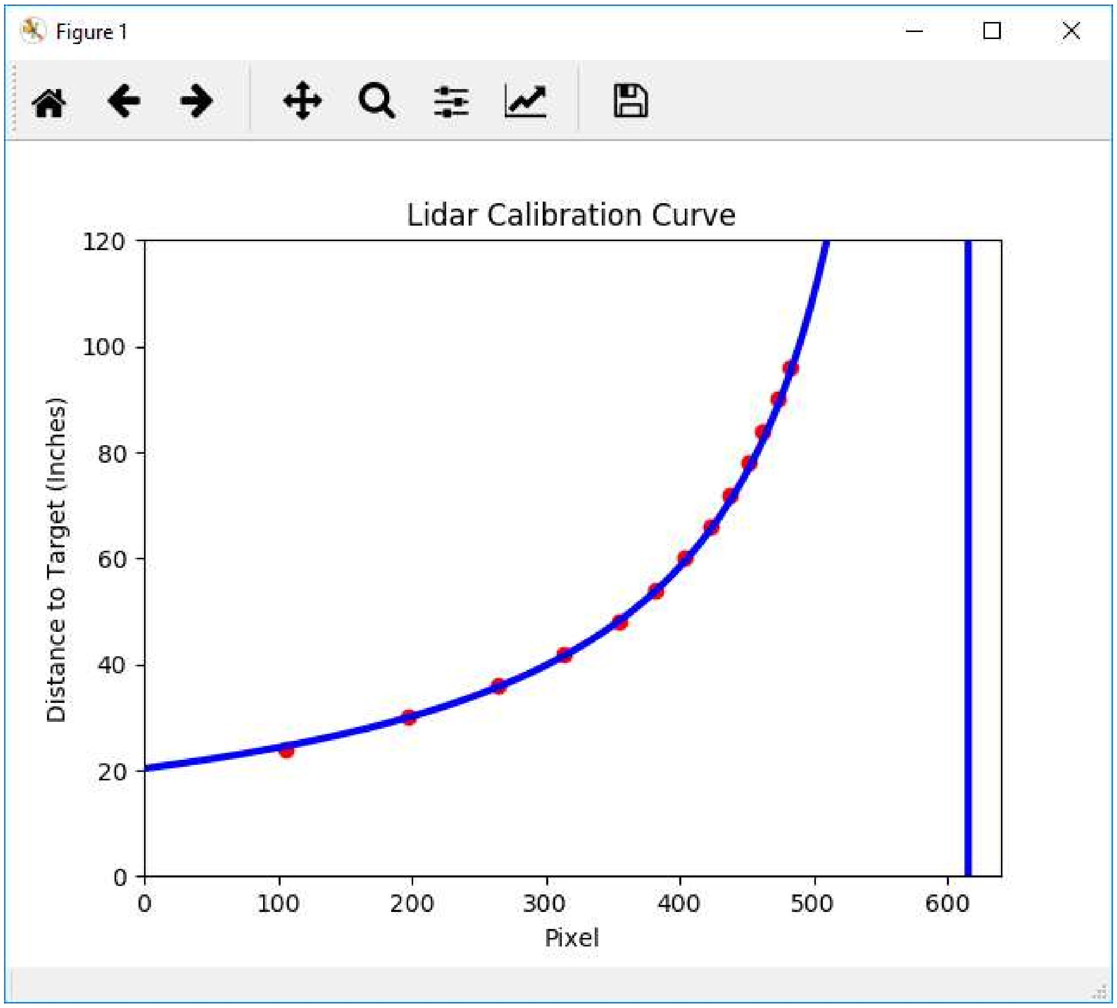
Execute the *laser_alignment.py* script in PyCharm. The script loops through each .jpg file in the directory, identifies the red laser pixels in the center row of the image (y = 360 pixels), averages the pixels together to identify the center pixel, and then creates a ***laserlog.txt*** file which stores the distance and corresponding pixel coordinates to file.


laserlog - Notepad

File Edit Format View Help

```
24 106
30 198
36 264
42 314
48 355
54 382
60 404
66 423
72 438
78 451
84 462
90 474
96 482
```

Similar to the calibration exercise of Assignment #4, the script then plots the acquired range data (red) on top of the theoretical output (blue). **Adjust the values of *ro* and *rpc* as required for your lidar**.



As with Assignment #4, ***paste your combined plot into a .doc file, generate 1-2 paragraphs describing how well your data aligns with the theoretical values, and upload a .pdf of the file to Gradescope.***

***Congratulations*!** Your field deployable lidar system is aligned, calibrated, and ready to head out on campus to map your target.

In Assignment #6 we will explore how to create 3D point clouds with the acquired range and IMU data, but plan to begin mapping your target *asap*!

Remember to take pictures & videos of yourself as you are on campus mapping for your semester project video and final presentation!

Assignment 5 – Supplemental.pdf
Credit: Original code written by Tommy Jones, UMD '17
Available on GitHub: https://github.com/oneshell/enme489y/blob/master/IMUchipv2.ino

Arduino Code to communicate with IMU

```
// ENME 489Y: Remote Sensing
// Spring 2017

// ADXL327 Accelerometer
// http://www.analog.com/media/en/technical-documentation/data-sheets/ADXL327.pdf

// Note: 3.3V required to power the accelerometer chip!

// ADC map
const int x = A1;         // x-axis yellow
const int y = A2;         // y-axis green
const int z = A3;         // z-axis white wire
const int samples = 400;   // number of samples averaged per 100ms
const int gap = 100;      // time for each reading

void setup() {
 Serial.begin(1200);
}

void loop() {

 // initialize voltage sum variables
 float totalx = 0;
 float totaly = 0;
 float totalz = 0;
 // initialize angle variables
 int xAngle = 0;
 int yAngle = 0;
 int zAngle = 0;

 // loop and record voltages
 for (int i =1; i<= samples; i++)
 {

   int xAngleRead = analogRead(x); // angle for x
   int yAngleRead = analogRead(y); // angle for y
   int zAngleRead = analogRead(z); // angle for z

  //first number of map is 0 degrees, second number is at 90 degrees; less than 0 is negative degrees
   // x
   // adds up all the voltages
```

```
    totalx = xAngleRead + totalx;
    totaly = yAngleRead + totaly;
    totalz = zAngleRead + totalz;


    delay(gap / samples); // loop delay
  }

  // average voltages
int xAngleAve = totalx / samples;
int yAngleAve = totaly / samples;
int zAngleAve = totalz / samples;

  // convert average voltages to angles, given calibration map
xAngle = map( xAngleAve, 342, 435, 0, 90); //angle for x
yAngle = map( yAngleAve, 333, 425, 0, 90); //angle for y
zAngle = map( zAngleAve, 333, 425, 0, 90); //angle for z

// Send angle measurements to the serial stream to be read by the Raspberry Pi
// Format: x y z
String xyzString = String(xAngle) + " " + String(yAngle) + " " + String(zAngle);
//Serial.print(xyzString);
Serial.println(xyzString);

// The following code can be uncommented to print analog measurements to the screen
// This information is useful when performing the initial angle calibration
// Otherwise, leave this section commented out
//String xyzString = String(analogRead(x)) + "," + String(analogRead(y)) + "," + String(analogRead(z));
//Serial.print(xyzString);
//Serial.println();

}
```