Department of
NUCLEAR ENGINEERING
University of California, Berkeley
Thermal Hydraulics Laboratory

XPETS Experimental Code

By Grant Buster

July 17, 2015

FHR
Fluoride Salt Cooled High Temperature Reactor

NEUP Nuclear Energy University Programs
U.S. Department of Energy

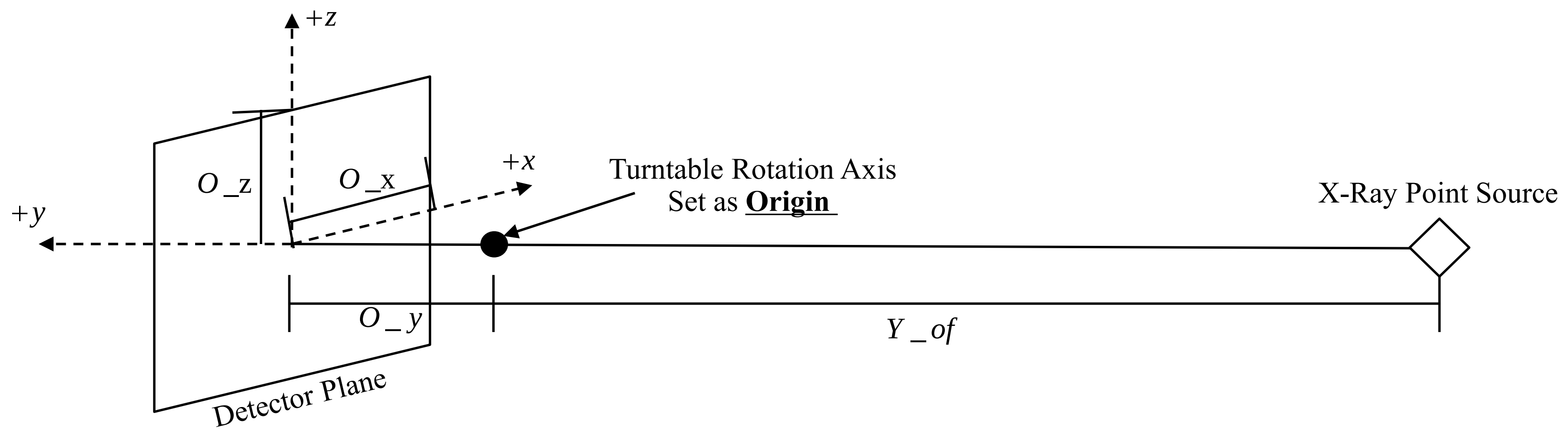
XPETS Calibration

Geometry Solver

(same as non-experimental Module 2 code)

Importance of Geometry

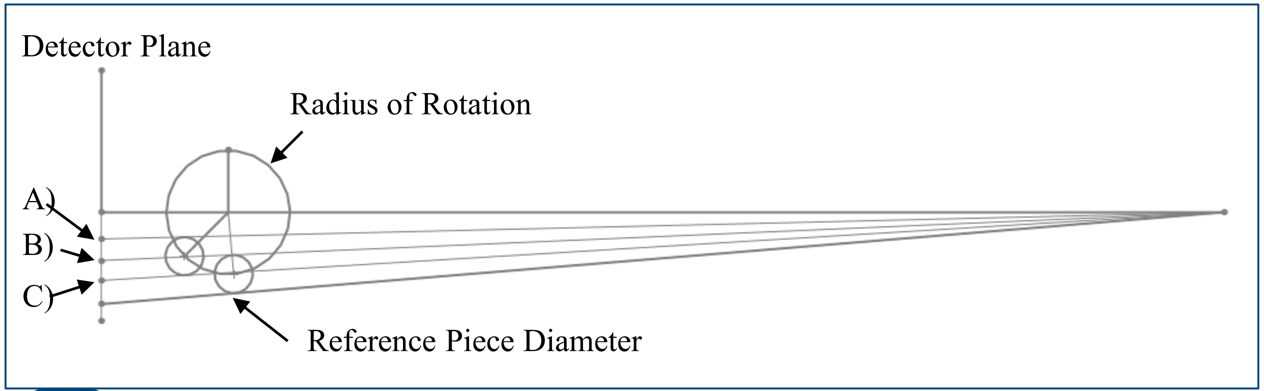
- An accurate 3D reconstruction is impossible without a good geometry solution
- What does this mean?
- 4 key geometric parameters need to be defined:
  - Y_of (y axis distance from origin to focal/source point)
  - O_x (x axis distance from right hand side of detector to origin)
  - O_y (y axis distance from detector plane to origin)
  - O_z (z axis distance from top of detector to origin)



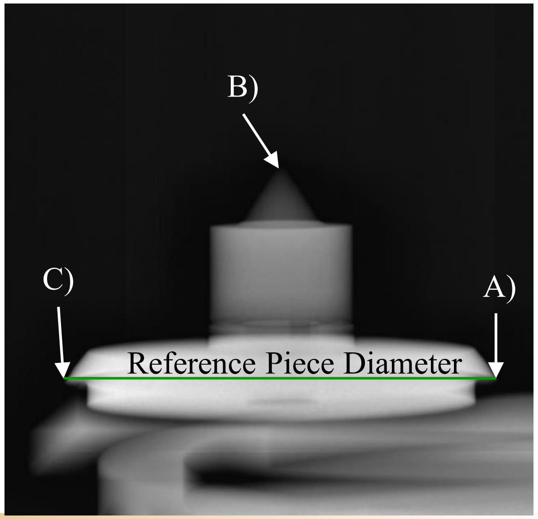
O_y and Y_of

- O_y and Y_of have been defined using a machined reference geometry and Solidworks geometry solver
  - O_y = 0.2283 (m)
  - Y_of = 1.8453 (m)
- These values should be considered constant unless:
  - The turntable is moved
  - The columnator or detector are moved in the y-direction (but vertical shifts should not affect O_y and Y_of)

SolidWorks Geo Solver



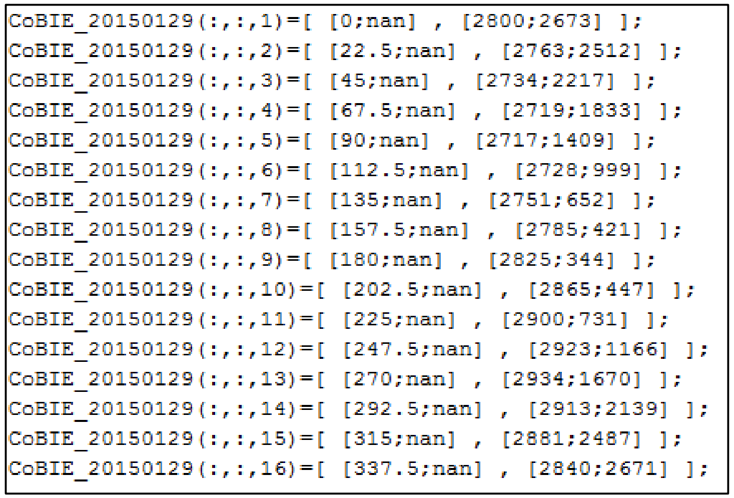
Machined Reference Piece



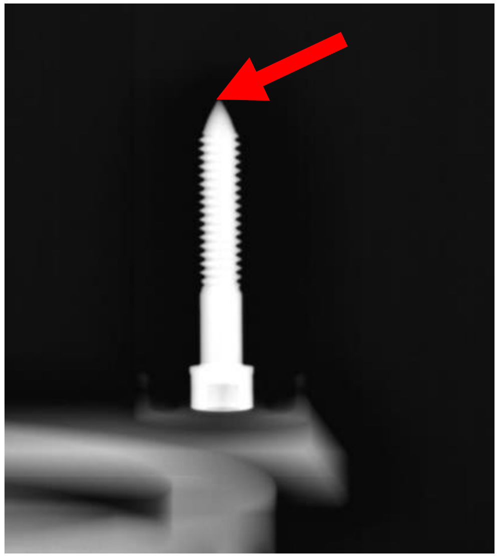
Defining O_x

- **We define O_x using an x-ray image set of a single point (usually a thumb screw) rotated through 360 degrees. NOTE: the screw should be placed as far off-center as possible (away from the center of the detector in z and x directions).**
- **The angle and the i,j tip location (in pixels) (i=column#, j=row#) for each image need to be entered in the format:**
  - Ox_DATA(:,:,1)=[[angle;nan] , [j ; i]];
- **Into the code: *XPREX_20141029_Full_Geo_Solver.m***
- **Example:**

XPREX_20141029_Full_Geo_Solver.m

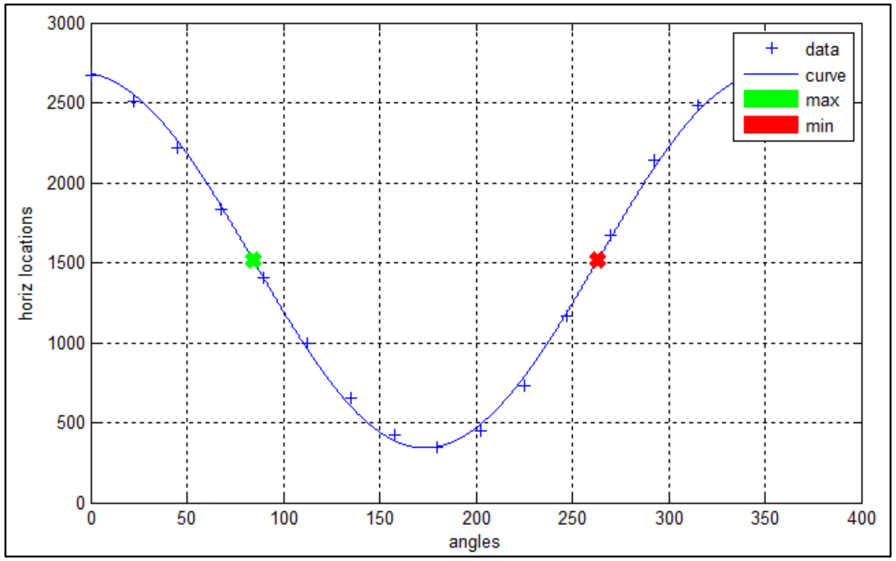


```
CoBIE_20150129(:,:,1)=[ [0;nan] , [2800;2673] ];
CoBIE_20150129(:,:,2)=[ [22.5;nan] , [2763;2512] ];
CoBIE_20150129(:,:,3)=[ [45;nan] , [2734;2217] ];
CoBIE_20150129(:,:,4)=[ [67.5;nan] , [2719;1833] ];
CoBIE_20150129(:,:,5)=[ [90;nan] , [2717;1409] ];
CoBIE_20150129(:,:,6)=[ [112.5;nan] , [2728;999] ];
CoBIE_20150129(:,:,7)=[ [135;nan] , [2751;652] ];
CoBIE_20150129(:,:,8)=[ [157.5;nan] , [2785;421] ];
CoBIE_20150129(:,:,9)=[ [180;nan] , [2825;344] ];
CoBIE_20150129(:,:,10)=[ [202.5;nan] , [2865;447] ];
CoBIE_20150129(:,:,11)=[ [225;nan] , [2900;731] ];
CoBIE_20150129(:,:,12)=[ [247.5;nan] , [2923;1166] ];
CoBIE_20150129(:,:,13)=[ [270;nan] , [2934;1670] ];
CoBIE_20150129(:,:,14)=[ [292.5;nan] , [2913;2139] ];
CoBIE_20150129(:,:,15)=[ [315;nan] , [2881;2487] ];
CoBIE_20150129(:,:,16)=[ [337.5;nan] , [2840;2671] ];
```



Defining O_x

- *XPREX_20141029_Full_Geo_Solver.m* then utilizes the fact that the horizontal location of the screw tip will move in a sinusoidal path:



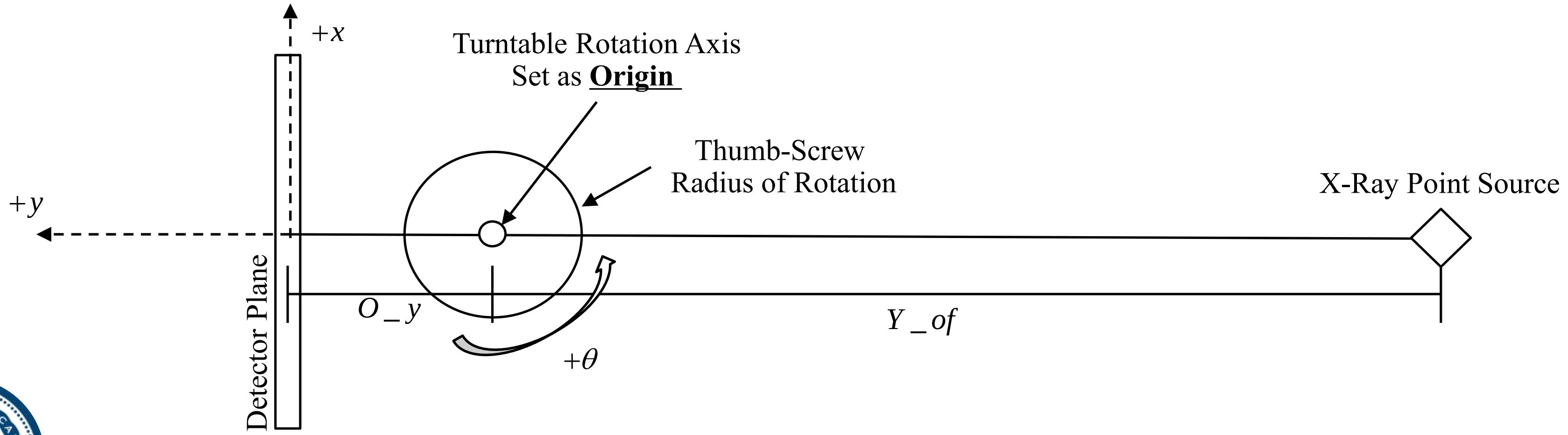
- *XPREX_20141029_Full_Geo_Solver.m* will automatically calculate any angle offset and O_x
  - Angle offset is caused by the screw not being placed perfectly at 0-degees when the turntable is at 0-degrees. This is accounted for and it is okay to place the screw at a random angle.

Defining O_z

- This is the (algebraically) difficult part
- Start from a top-down view, in order to calculate the radius of rotation of the thumb screw tip
- Radius of rotation is calculated by the following equation:

$$\boxed{R_{rot} = \frac{Y\_of \times abs\left(i - O\_x\right)}{\left(Y\_of + O\_y\right)\cos\left(\theta\right) - abs\left(i - O\_x\right)\sin\left(\theta\right)}}$$

- Where i is the horizontal pixel location (column index) of the thumbscrew tip in one image. This is usually taken when the screw is far right or far left. Theta in the above equation is theta of the turntable plus the theta offset



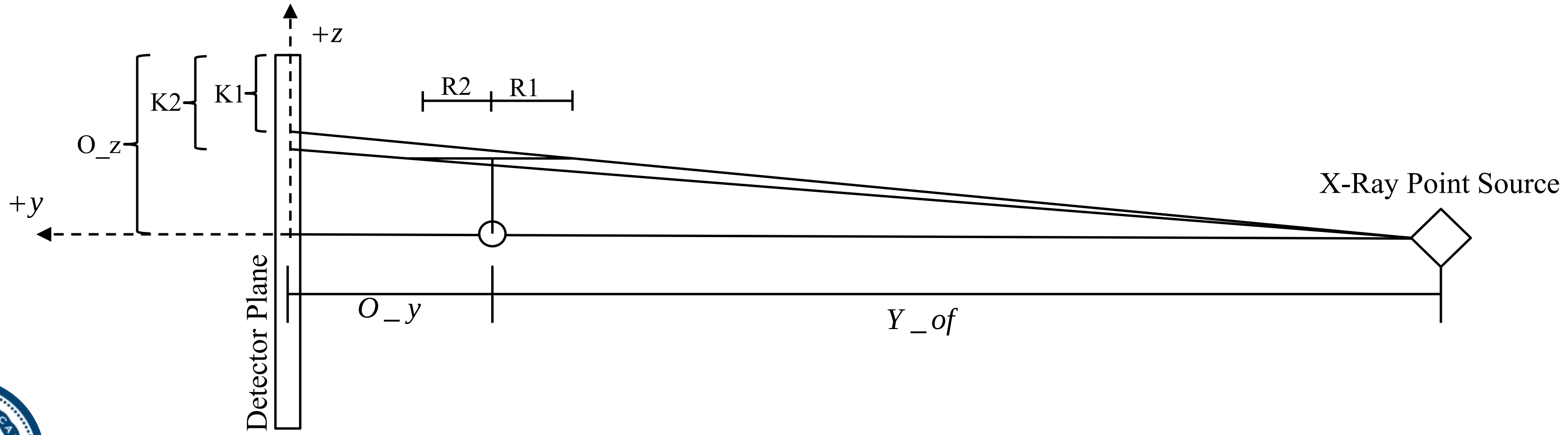
Defining O_z

- We move to a side-view, in order to calculate O_z
- Two images are picked, usually the two where the thumbscrew tip is highest and lowest in the image
- R1 and R2 are calculated using R_rot*sin(theta), in the domain:

$$R1 < 0$$
$$R2 > 0$$

- O_z is calculated as follows:

$$\boxed{O\_z = \frac{(K2)(Y\_of) + (K2)(R2) - (K1)(Y\_of) - (K1)(R1)}{R2 - R1}}$$



Geometry Solved!

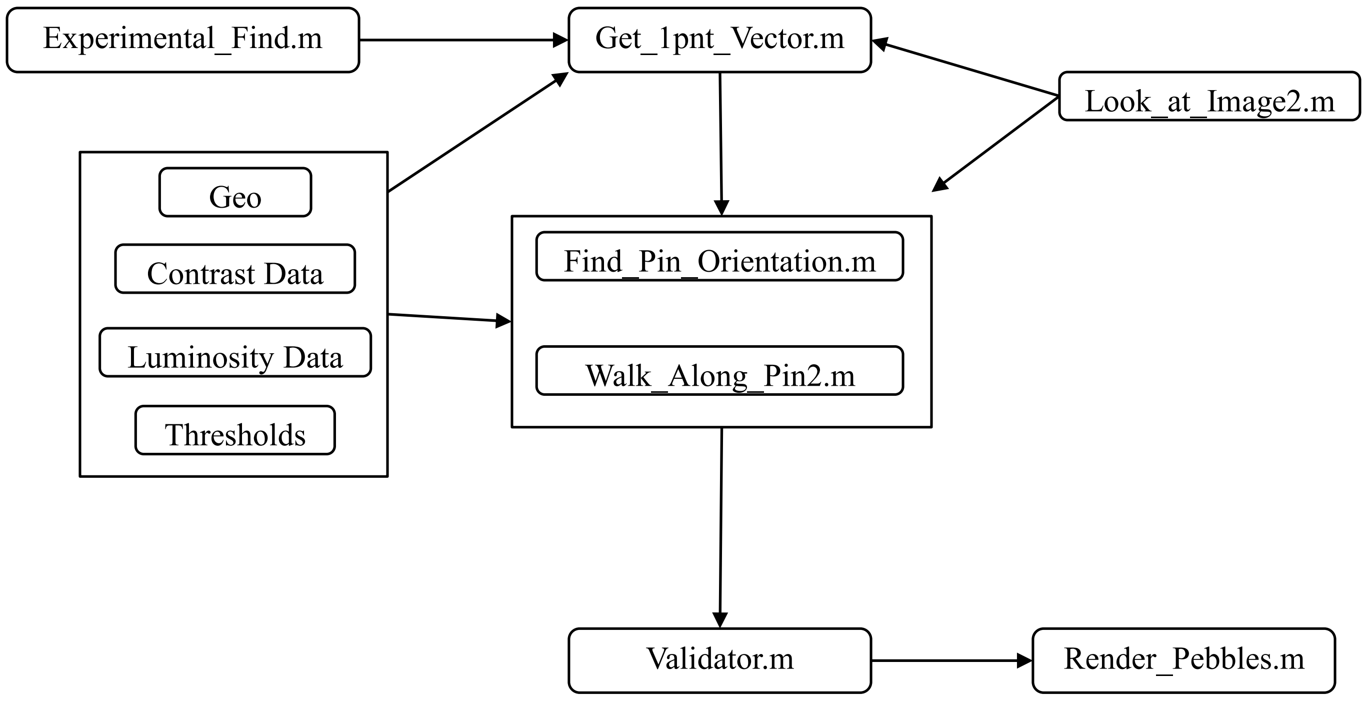
- The geo.mat file should now be good to go!
- Two other important parameters:
  - The detector has 6993 pixels per meter (PPM)
  - A pin/pebble diameter is 0.01257 (m)
- The geo.mat file should be placed in the active directory, and loaded before module2 starts to run

XPETS Experimental 3D Bed Reconstruction

Experimental Search Algorithm

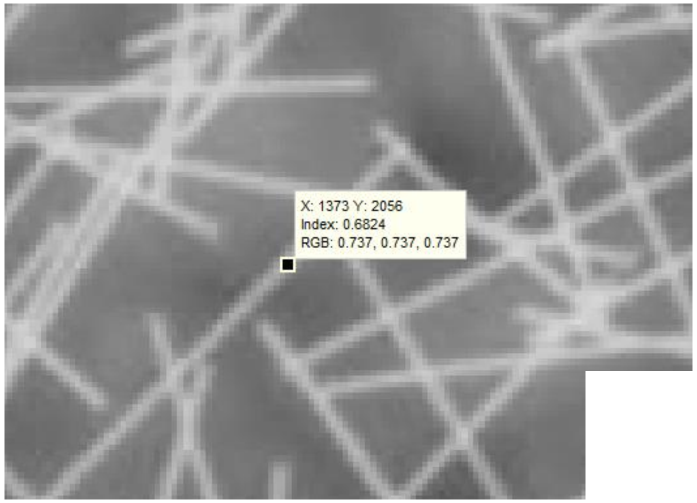
- The experimental code operates on the principle that it should be easier to search for an instrumented pebble in a 3D space
- In broad strokes this is the idea:
  - Assume you can find a point in 3D where you think a tungsten pin was in the actual test section (we'll go through how you find this point in a minute)
  - You should be able to scan around that point and find two other points that appear (in the x-ray and contrast images) to be on the same pin
  - Those two points should give you a pin orientation
  - You can then "walk along" that orientation, verifying that you are still on a pin in your images
  - Once you "step off" of the pin, you can verify that you have found a pin of an appropriate length
- Let's see how this is done…

Module 2 Code Architecture

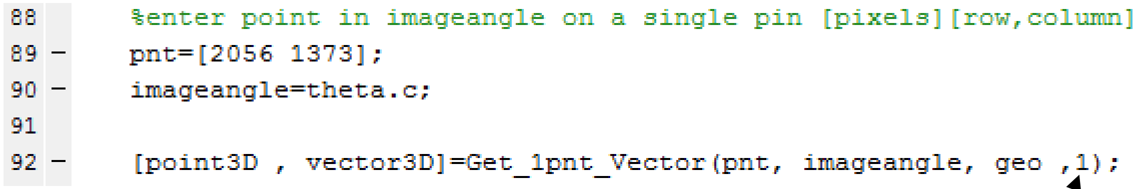


The Starting Point

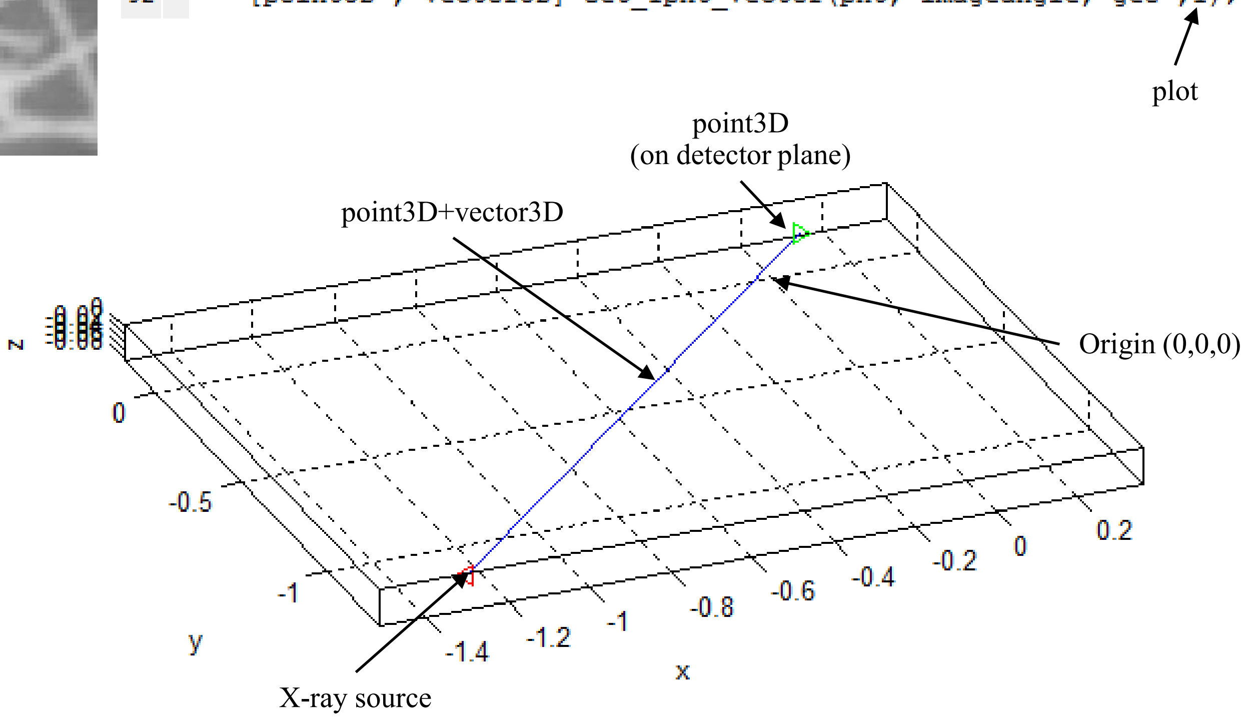
- **Start with a point that you know is on a single pin in an x-ray image. This is currently a user input, but in theory it shouldn't be hard to automate**



In Experimental_Find_05.m:

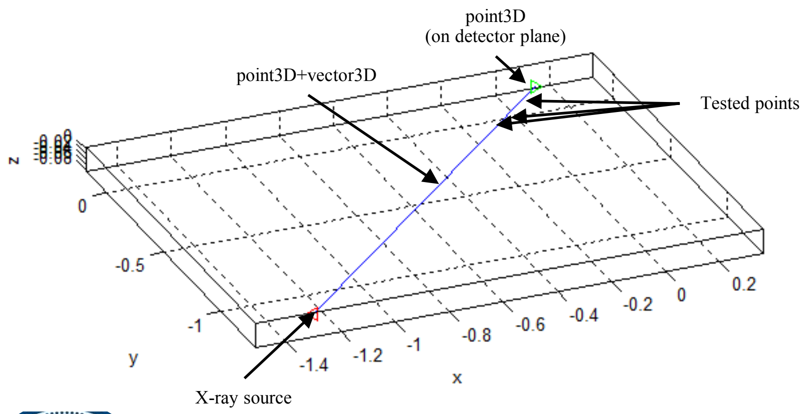


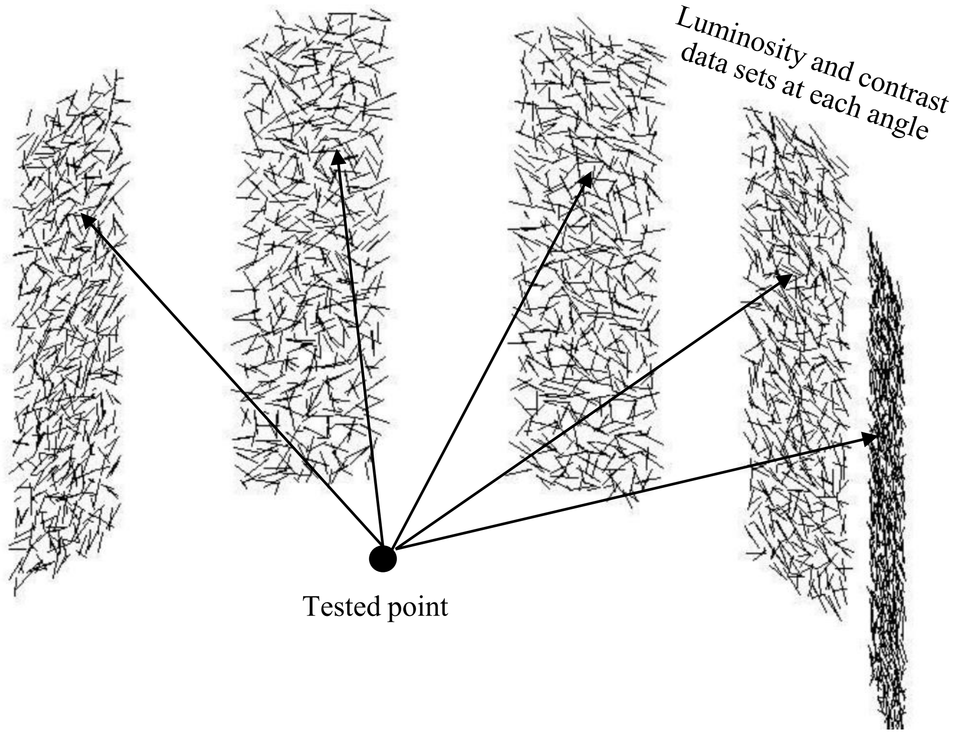
```
    %enter point in imageangle on a single pin [pixels][row,column]
    pnt=[2056 1373];
    imageangle=theta.c;

    [point3D , vector3D]=Get_1pnt_Vector(pnt, imageangle, geo ,1);
```



The Starting Point

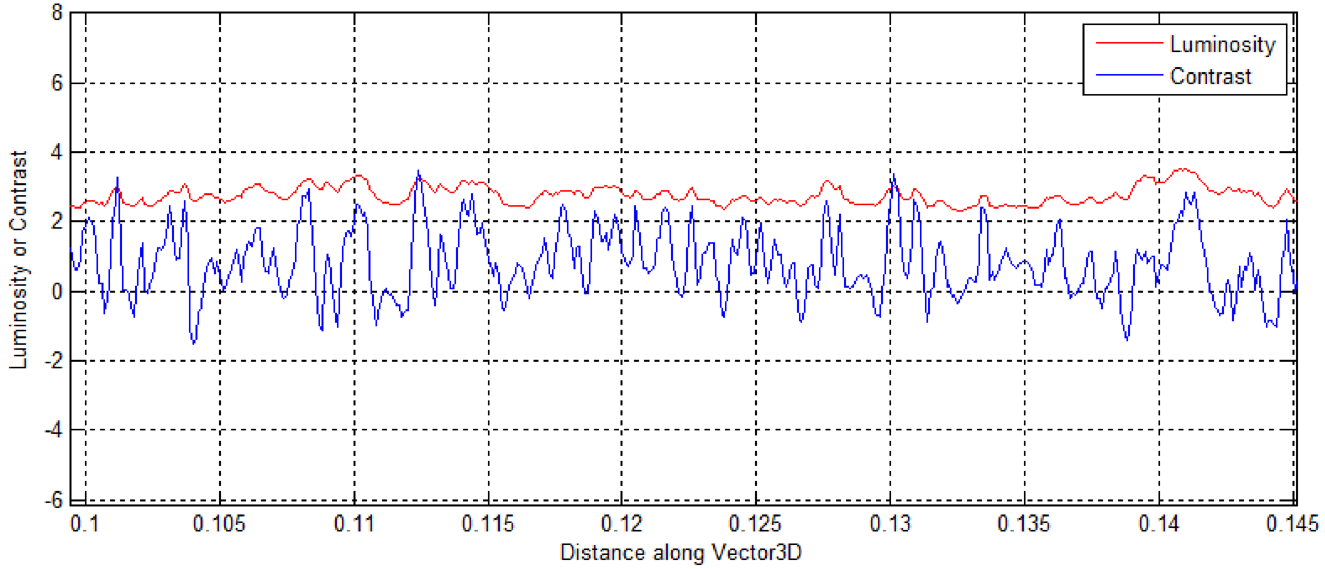
- Experimental_Find_5.m will then iterate from point3D along vector3D to a user-input endpoint
- The code uses Look_at_Image2.m to check each point along vector3D against luminosity and contrast sets
- Contrast and luminosity data at each point along vector3D are then stored in "point_assessments"





The Starting Point

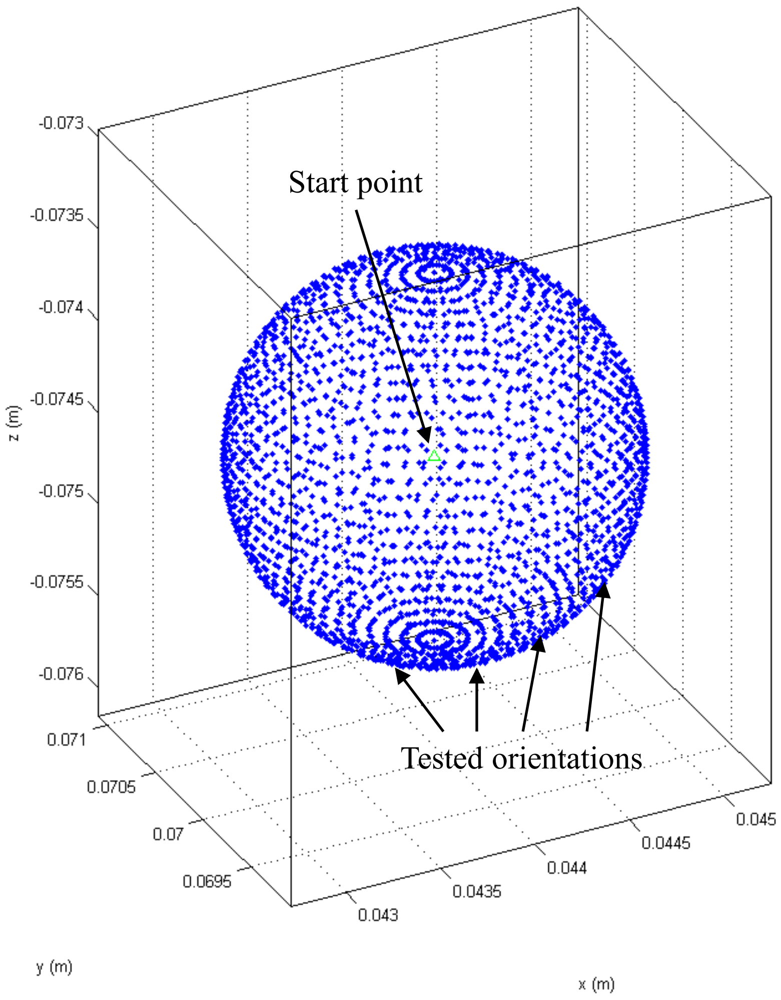
- Contrast and luminosity data at each point along vector3D are then stored in “point_assessments” and can be seen plotted here:



- The code has to pick a select few of the points along vector3D to test. The selection method certainly deserves improvements
- Keep computational cost in mind
- Once you pick points to try, you have your starting points

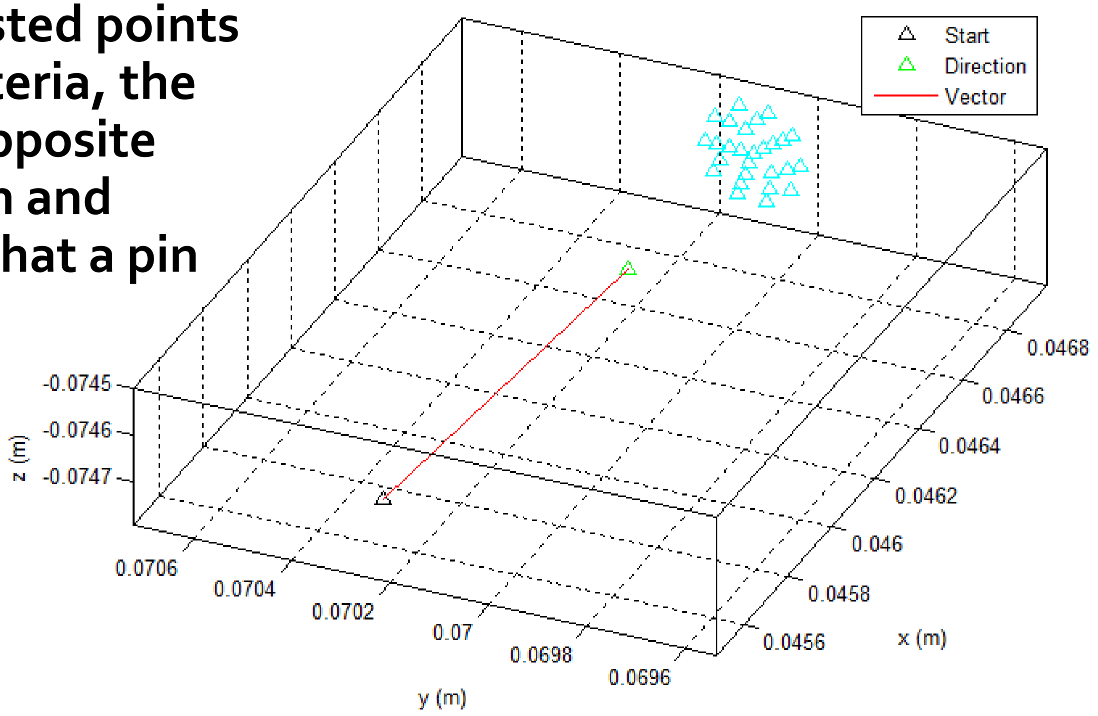
Finding the Orientation of a Pin

- Now that you have a starting point in 3D that you think may belong to an actual tungsten pin in the test section, you can try and find the orientation of that pin
- Find_Pin_Orientation.m checks a ton of points in a sphere around your starting point
- The function currently returns a single direction based on the best contrast or luminosity data



Tracking a Pin

- Once an orientation has been determined, the code "walks along" the pin in both the positively tested direction and the opposite direction (forward and backward)
- The code tests points in the asserted direction, allowing for a certain angular error. This creates a cone-shaped projection
- Once the code "walks off" of a pin, i.e. none of the tested points fulfill the threshold criteria, the code saves the most opposite points, tests the length and hopefully determines that a pin was indeed found



Conclusions

- This method has been shown to work very well for isolated pins (perfectly ideal cases) and also with mixed results for pins in beds with depth up to 12 pebble diameters
- By testing multiple points along a pin in the 2D image, the 3D pin should certainly be findable
- The full algorithm is very clearly half-baked, but the methodology deserves more exploration
  - The full algorithm would iterate through 2D locations that have very high contrast ridges (Mike's module 1 code does a great job of determining these locations)
  - Found pins could eliminate many possibilities

Conclusions

- Here are some proposed improvements:
  - The algorithm that selects from "point_assessments" needs to combine normalized luminosity and contrast data and also account for contrast "ridges" (strong positive surrounded by negative contrast)
    - » This is possibly most important. Testing only one point is surprisingly fast. If less than 5 points could be identified as possibilities, this code would be speedy.
  - The starting point in the 2D image should be selected so it is <u>not</u> the endpoint of a pin
    - » The orientation should then be found such that the opposite orientation is also verified at the same time in the same sphere of tested points
  - Several spheres of tested points (w/ different radii) could be tested at the same time to try to find a consistent orientation