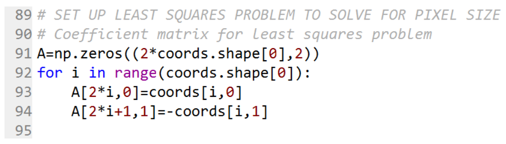
1 Python code walkthrough

At this point in our code, the variables *coords* contains the image plane coordinates of the free-throw rectangle and the backboard white box. As explained in our Theory Writeup, the free-throw rectangle and backboard box are points on the court for which we know the real-life coordinates (and thus the image plane coordinates) as well as the pixel coordinates to a reasonable degree of accuracy. Following the notation in our Theory Writeup, we create a matrix $A = \begin{pmatrix} x_1 & 0 \\ 0 & -y_1 \\ x_2 & 0 \\ 0 & -y_2 \\ \dots & \dots \end{pmatrix}$ using the image plane coordinates.



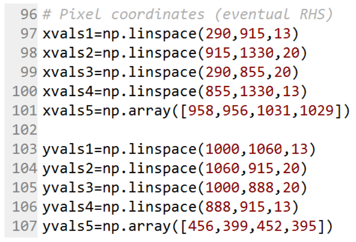
```
# SET UP LEAST SQUARES PROBLEM TO SOLVE FOR PIXEL SIZE
# Coefficient matrix for least squares problem
A=np.zeros((2*coords.shape[0],2))
for i in range(coords.shape[0]):
    A[2*i,0]=coords[i,0]
    A[2*i+1,1]=-coords[i,1]

```

Figure 1: Code that creates the Coefficient Matrix of our least-squares problem

We then create variables *xvals1, ..., xvals5* and *yvals1, ..., yvals5* that contain the pixel x-coordinates and y-coordinates of our points. There are 5 variables because I broke the free-throw rectangle into 4 parts (one for each side) and made a fifth set of coordinates for the backboard box. Each array contains hard-coded coordinates that I estimated using MS Paint. For example, *xvals1* corresponds to the pixel x-coordinates of the free-throw line. Using Paint, we see that the pixel x-coordinate of the left end of the free-throw line is (approximately) 290 and that the pixel x-coordinate of the right end of the free-throw line is (approximately) 915. We then use linspace to get the pixel x-coordinates of some number of intermediate points on the line (in this case, 13 total points). Similarly, *yvals1* represents the fact that the pixel y-coordinate of the left end of the free-throw line is 1000, while the y-coordinate of the right end is 1060.

Note: This was all hard-coded because I couldn't think of a good way to systematically identify the pixel coordinates of the free-throw rectangle and backboard box.

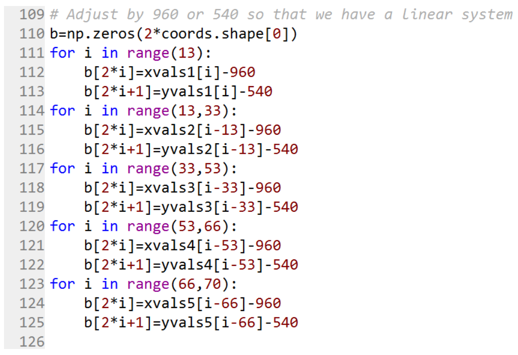
Variable	Real-life Object
xvals1, yvals1	Free-throw line
xvals2, yvals2	Right side of free-throw box
xvals3, yvals3	Left side of free-throw box
xvals4, yvals4	Baseline side of free-throw box
xvals5, yvals5	Backboard box



```python
# Pixel coordinates (eventual RHS)
xvals1=np.linspace(290,915,13)
xvals2=np.linspace(915,1330,20)
xvals3=np.linspace(290,855,20)
xvals4=np.linspace(855,1330,13)
xvals5=np.array([958,956,1031,1029])

yvals1=np.linspace(1000,1060,13)
yvals2=np.linspace(1060,915,20)
yvals3=np.linspace(1000,888,20)
yvals4=np.linspace(888,915,13)
yvals5=np.array([456,399,452,395])
```

Figure 2: Code that creates arrays of the x and y pixel coordinates of points

The next section of our code takes our x and y pixel coordinates and turns them into the RHS $\begin{pmatrix} v_1 - 960 \\ u_1 - 540 \\ v_2 - 960 \\ u_2 - 540 \\ \dots \end{pmatrix}$ from our Theory Writeup. We then solve the least-squares problem and our solution vector $\vec{s}$ contains the pixel height and pixel width.



```python
# Adjust by 960 or 540 so that we have a linear system
b=np.zeros(2*coords.shape[0])
for i in range(13):
    b[2*i]=xvals1[i]-960
    b[2*i+1]=yvals1[i]-540
for i in range(13,33):
    b[2*i]=xvals2[i-13]-960
    b[2*i+1]=yvals2[i-13]-540
for i in range(33,53):
    b[2*i]=xvals3[i-33]-960
    b[2*i+1]=yvals3[i-33]-540
for i in range(53,66):
    b[2*i]=xvals4[i-53]-960
    b[2*i+1]=yvals4[i-53]-540
for i in range(66,70):
    b[2*i]=xvals5[i-66]-960
    b[2*i+1]=yvals5[i-66]-540
```

Figure 3: Code that creates the RHS of the least-squares problem.