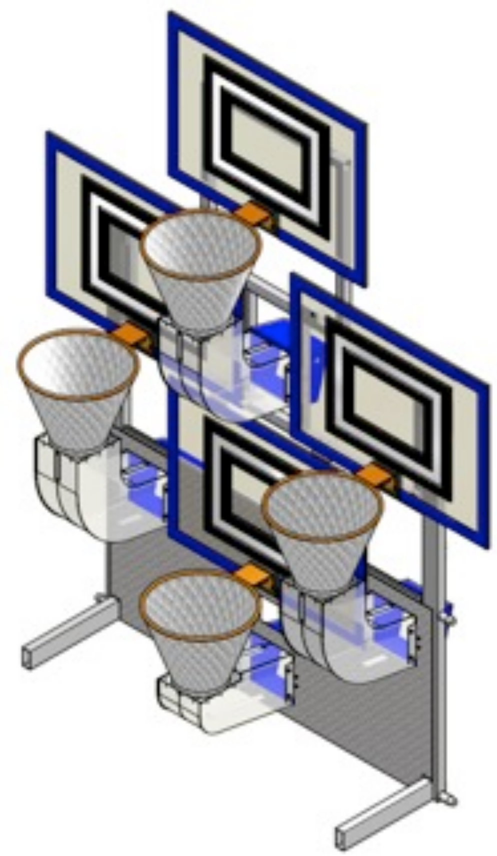
FIRST Robotics 2012

How to Determine Camera Heading and Location from 2 Known Boxes on a Wall



Mike Stitt

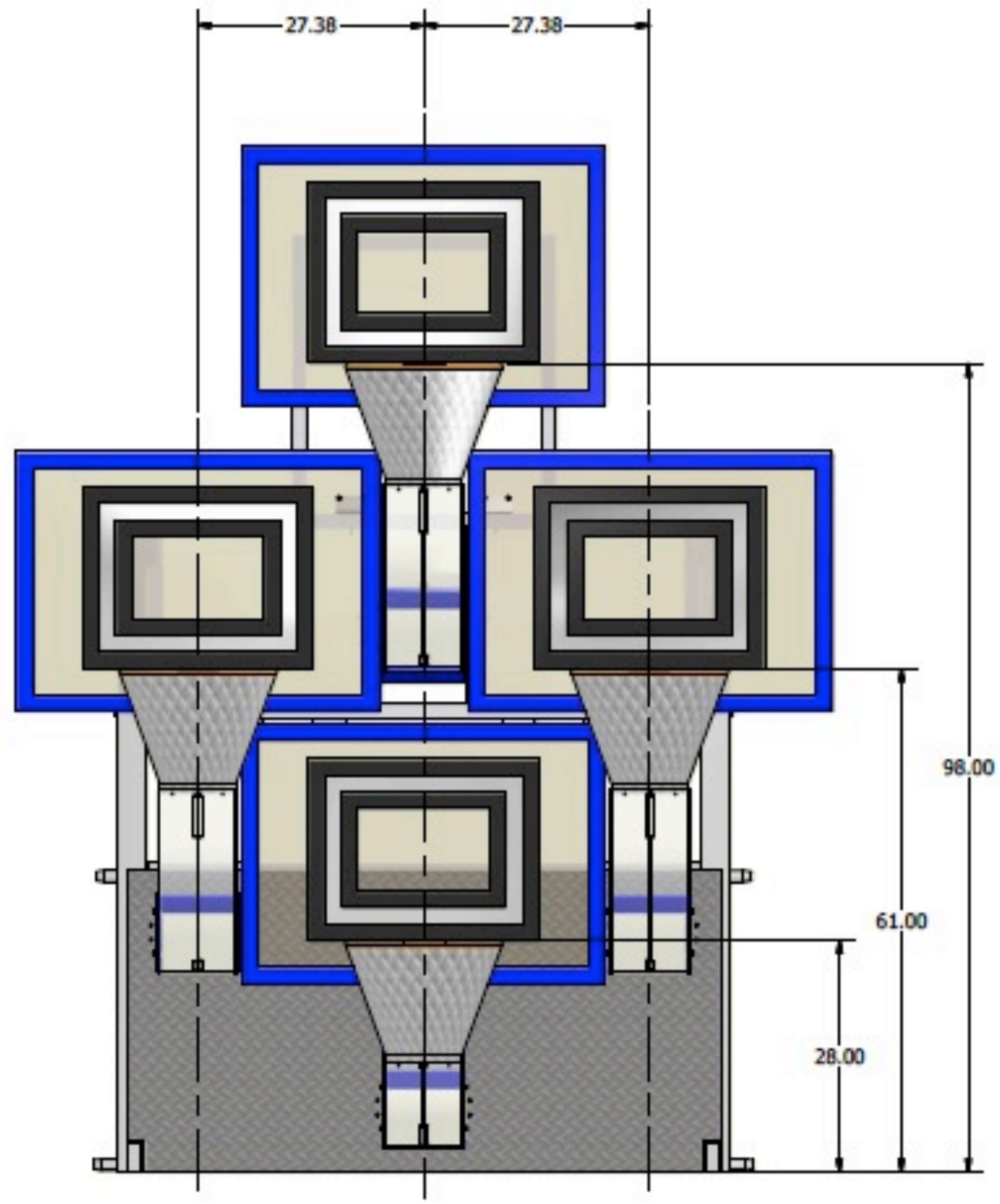
26 February 2012

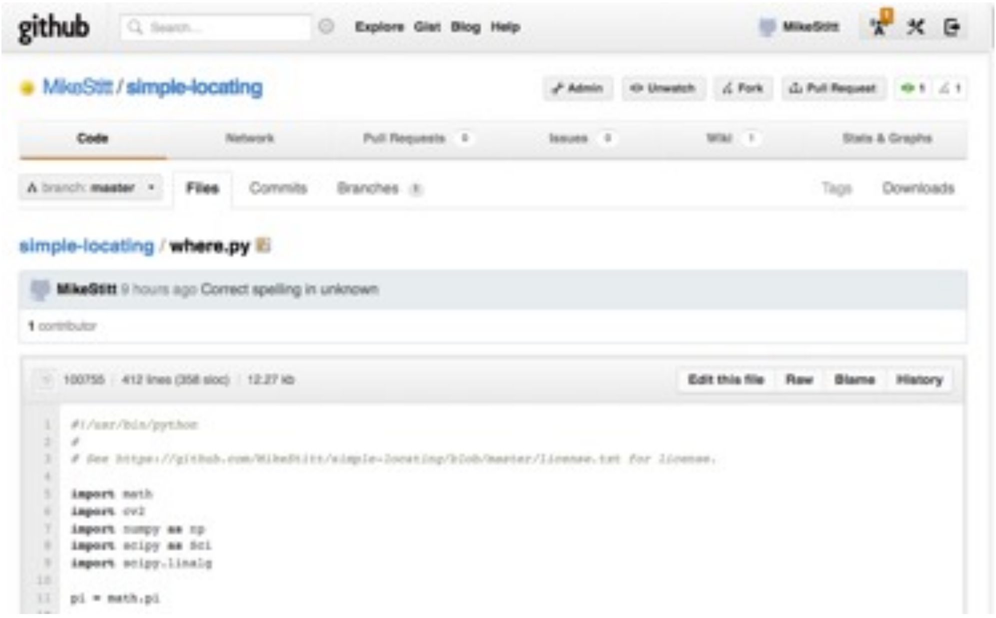
rev 3

See https://github.com/MikeStitt/simple-locating-docs/blob/master/howto.pdf for latest version.

The Problem Space

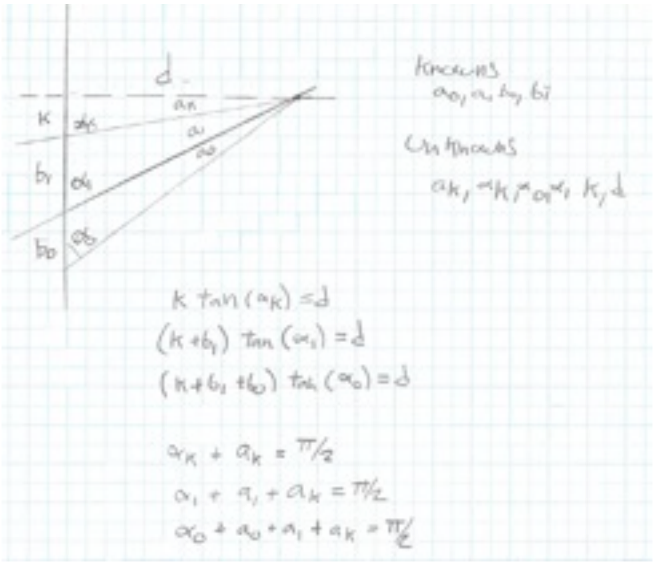
The FIRST Robotics 2012 Backboard Targets





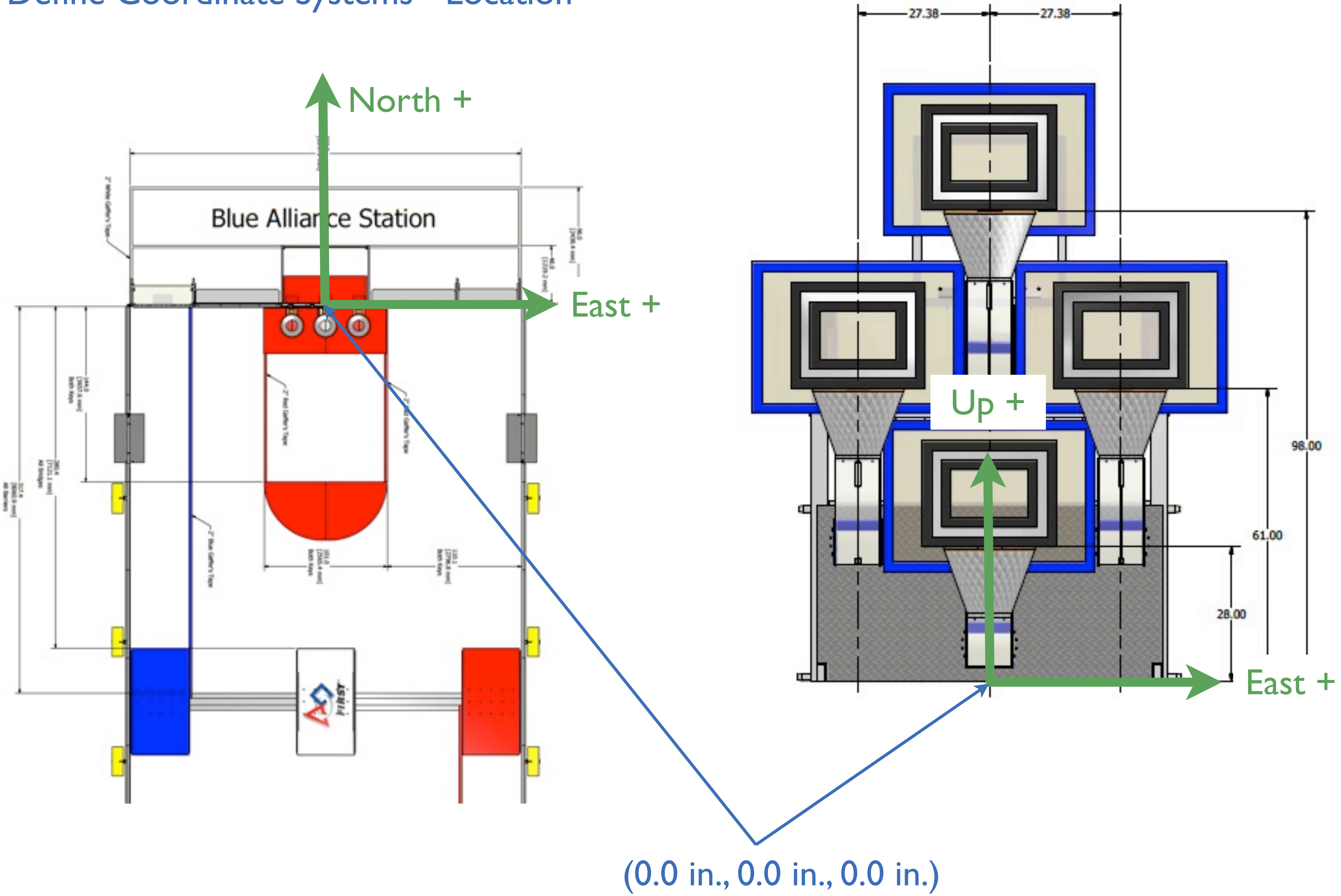
See:
https://github.com/MikeStitt/simple-locating
for sample code in python.

See
http://python.org/
for information on python.

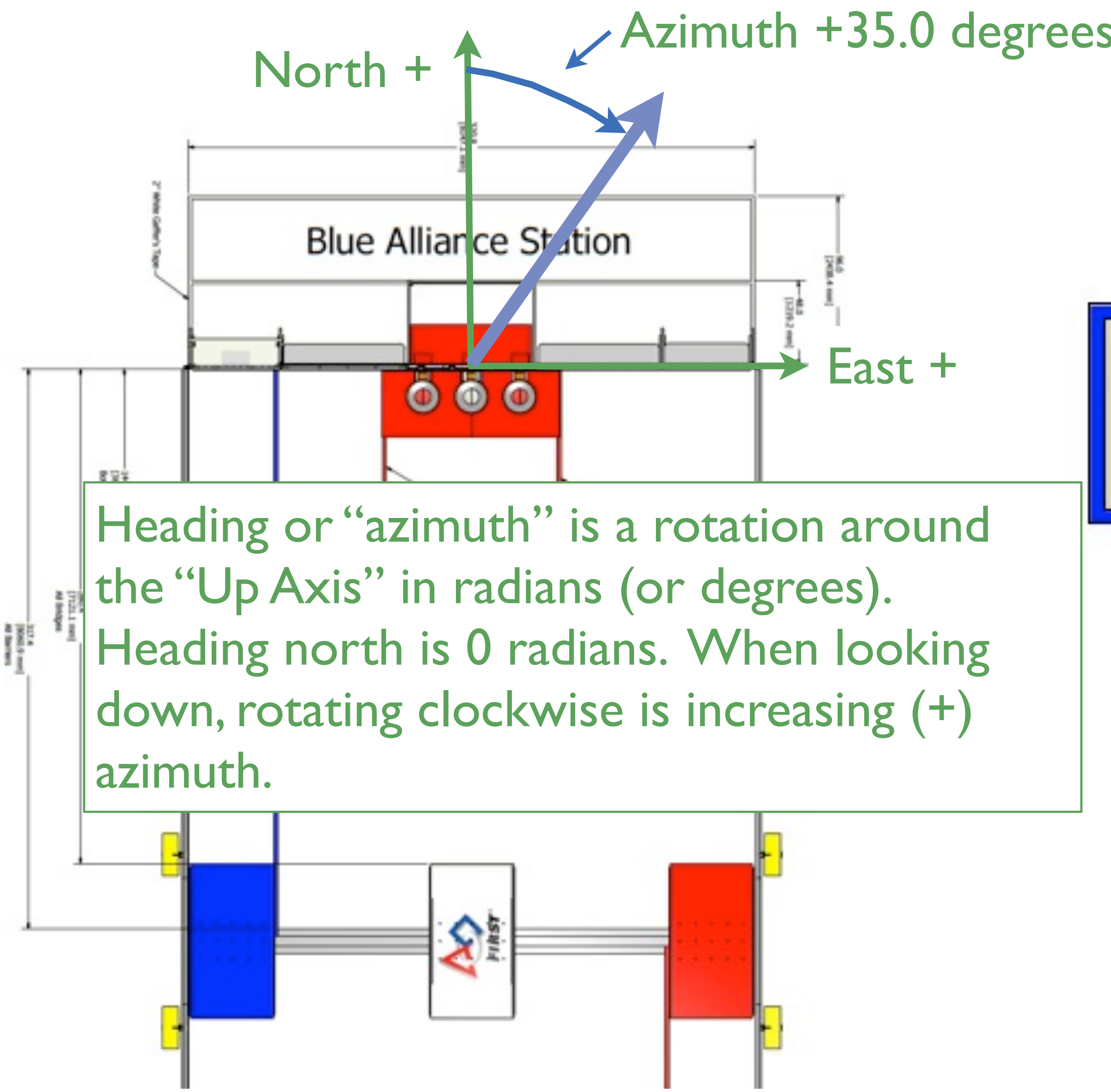


See:
https://github.com/MikeStitt/simple-locating-docs
for this how-to.

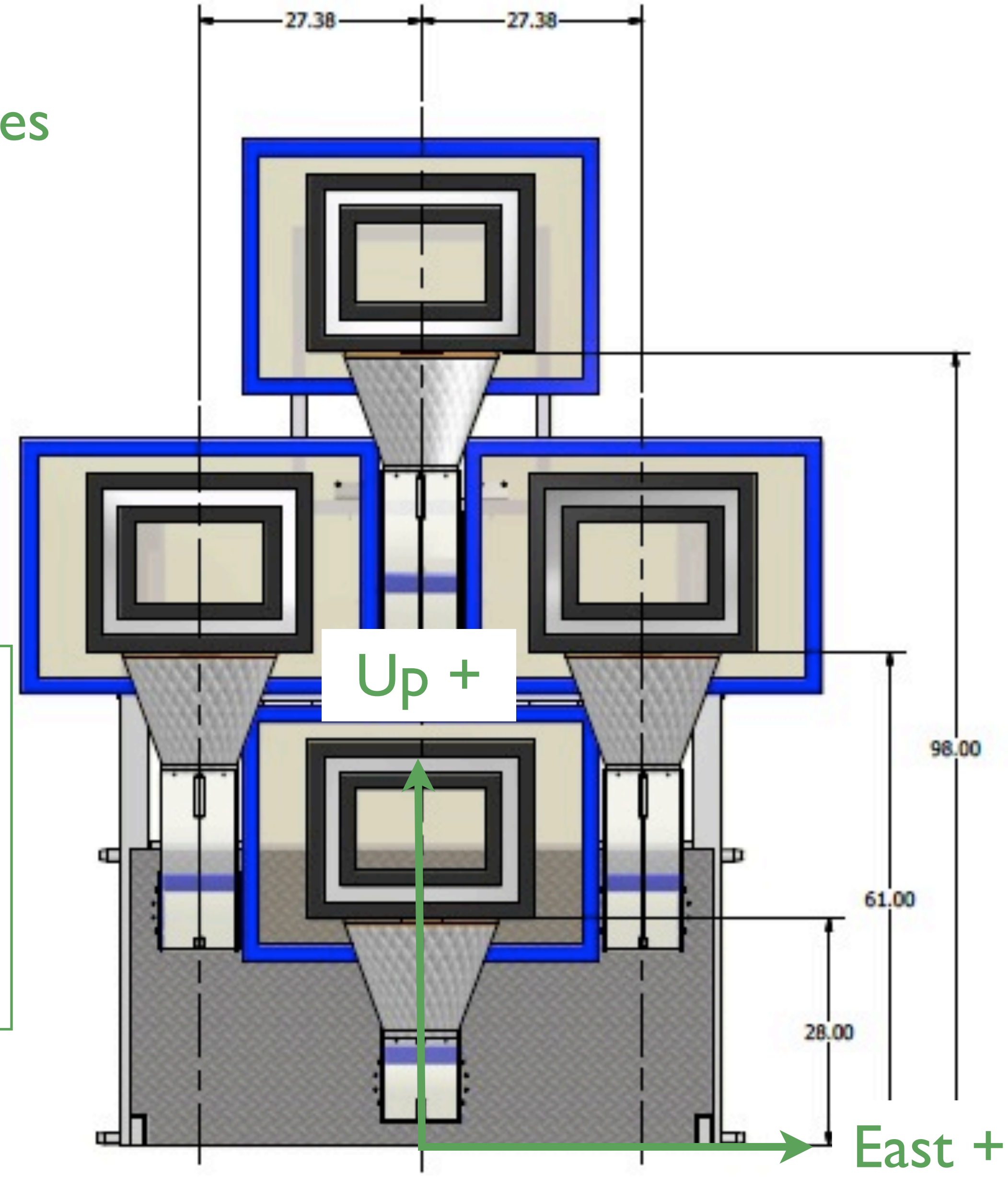
Step 0a
Define Coordinate Systems - Location



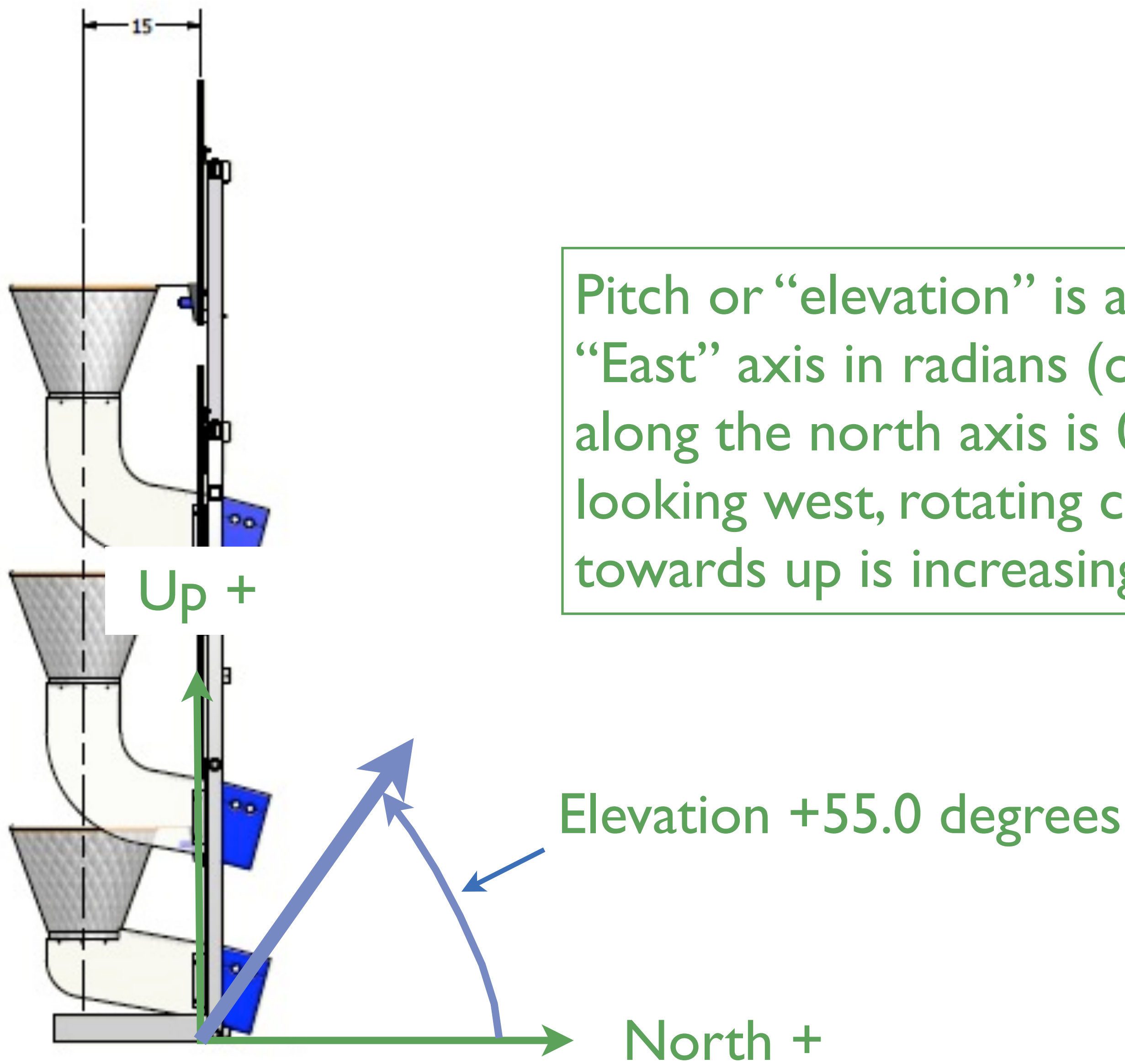
Step 0b
Define Coordinate Systems - Heading or Azimuth



Heading or "azimuth" is a rotation around the "Up Axis" in radians (or degrees). Heading north is 0 radians. When looking down, rotating clockwise is increasing (+) azimuth.



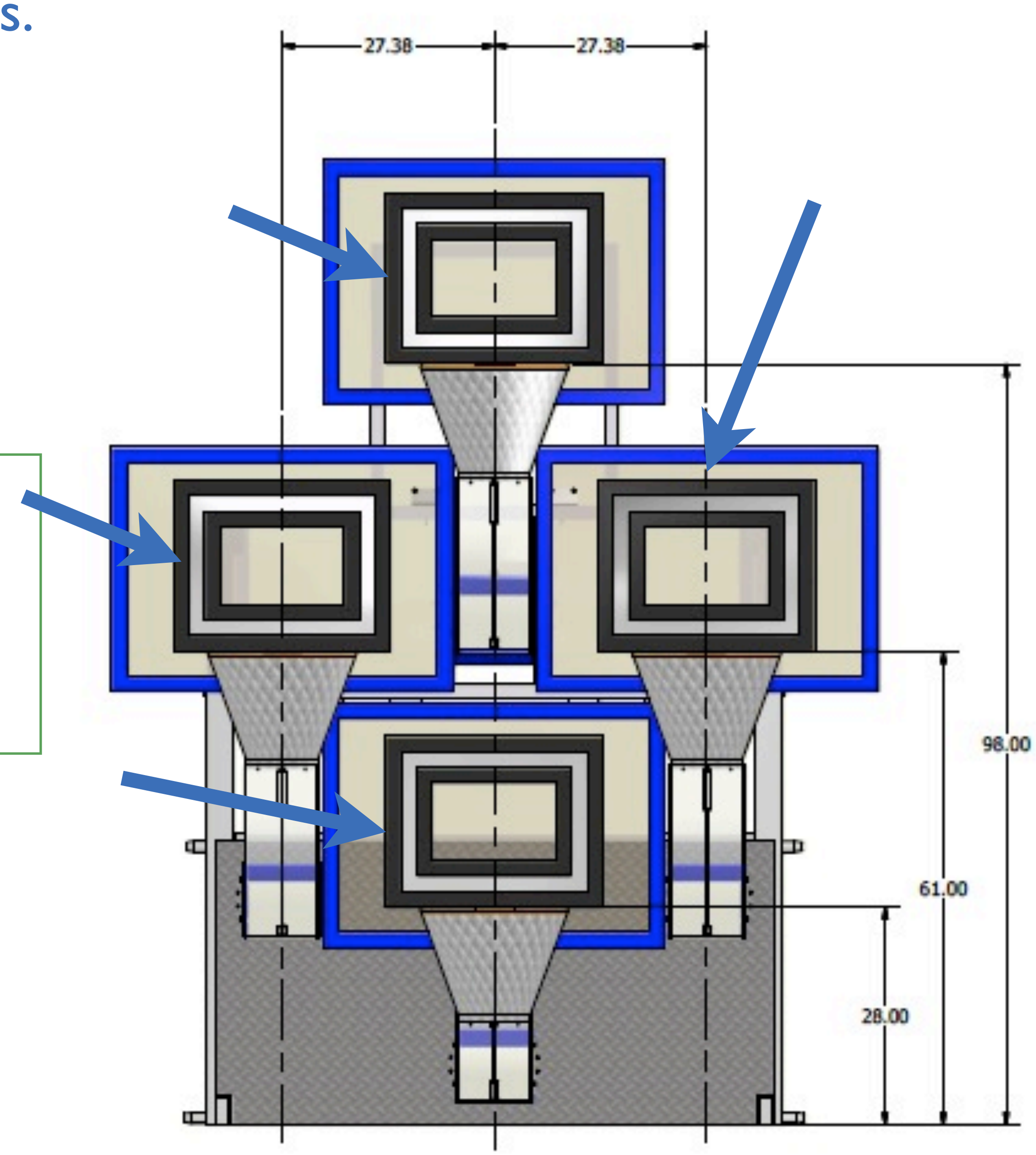
Step 0c
Define Coordinate Systems - Pitch or Elevation



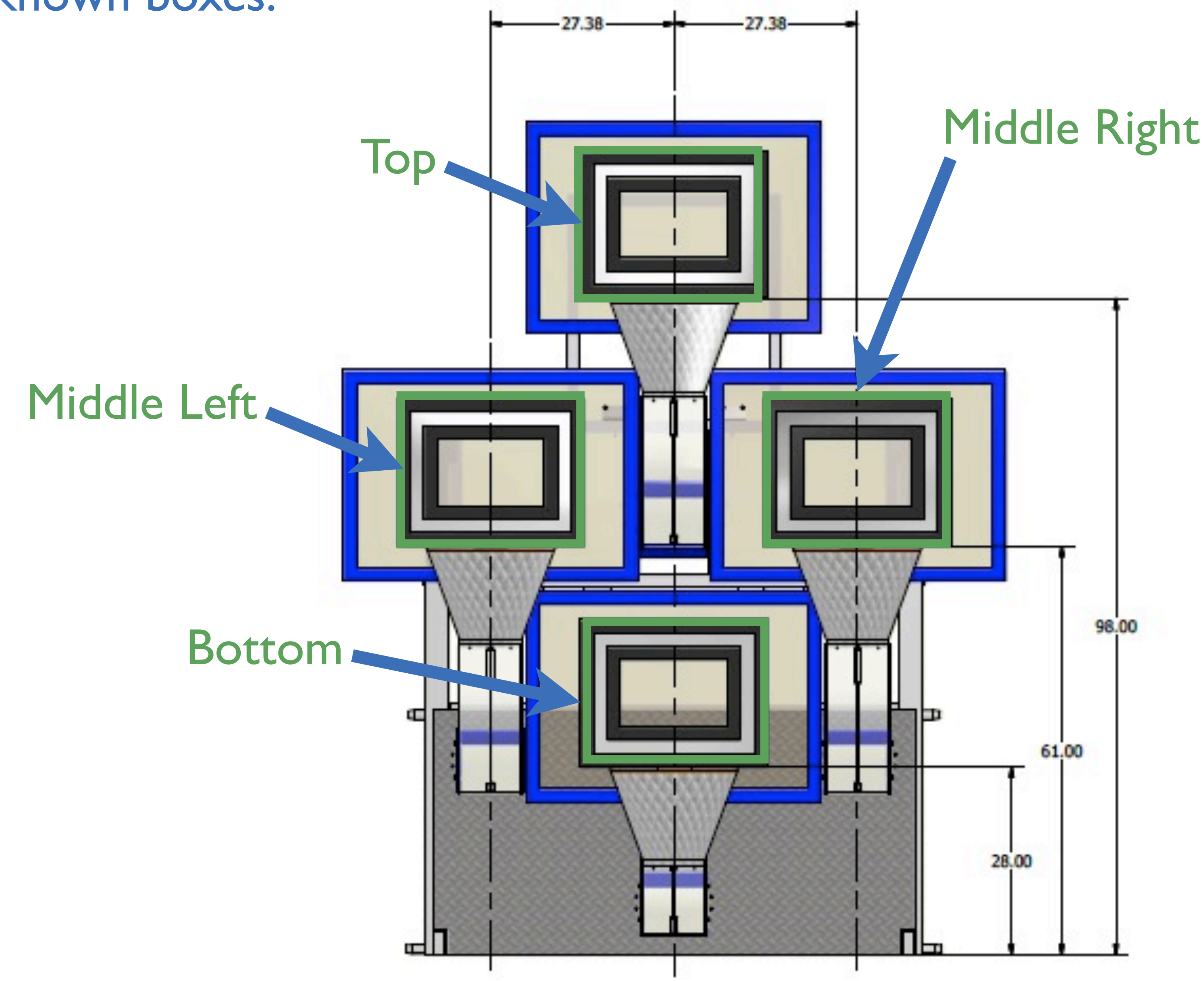
Pitch or "elevation" is a rotation around the "East" axis in radians (or degrees). Pitched along the north axis is 0 radians. When looking west, rotating counter clockwise towards up is increasing (+) elevation.

Step 0d
Define a table of locations of the known boxes.

Using the high contrast white boxes surrounded by black boxes as good objects for a computer vision system to recognize we have 4 targets.



Step 0d
Define a table of locations of the known boxes.



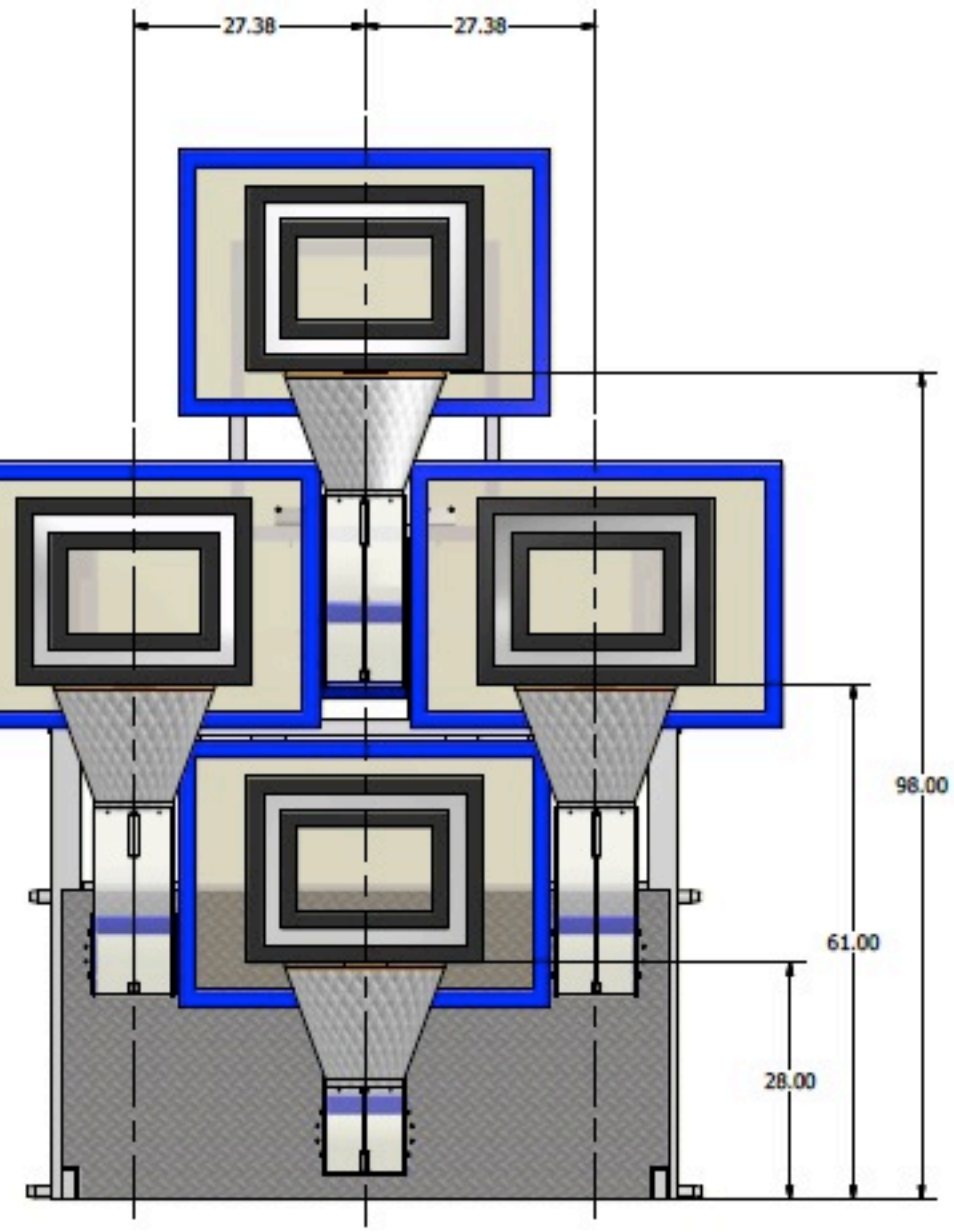
In where.py we define a target_position class that assumes the box targets are along the east and up axis at north 0.0 in”

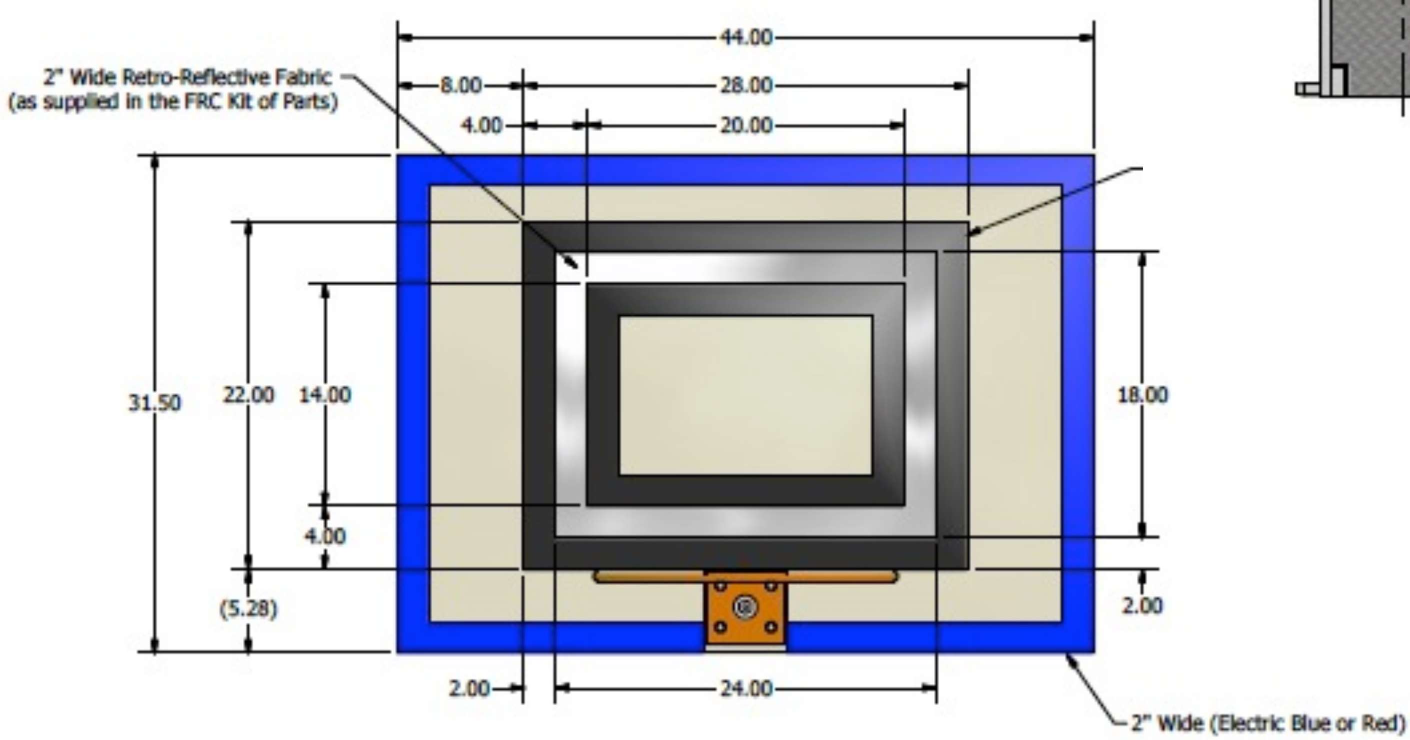
```
class target_position:
     def __init__(self,l,r,t,b,h):
          self.left_inches = l            # east coordinate of left edge
          self.right_inches = r           # east coordinate of right edge
          self.top_inches = t             # up coordinate of top edge
          self.bottom_inches = b          # up coordinate of bottom edge
          self.hoop_height = h            # up coordinate of hoop
          self.center_east = (l+r)/2.0    # east coordinate center
          self.center_up = (t+b)/2.0      # up coordinate
```

Step 0d
Define a table of locations of the known boxes.

Define some useful constants

```
#
# Height of hoop above ground
#

LOW_HOOP_UP = 28.0   # inches
MID_HOOP_UP = 61.0   # inches
TOP_HOOP_UP = 98.0   # inches

#
# Center of middle hoop
#
MID_LEFT_HOOP_EAST  = -27.38  # inches
MID_RIGHT_HOOP_EAST = +27.38  # inches

#
# Target edges from center of hoop
#
TARGET_LEFT_DELTA   = -12.0 #inches
TARGET_RIGHT_DELTA  = +12.0 #inches
TARGET_TOP_DELTA    = +20.0 #inches
TARGET_BOTTOM_DELTA =  +2.0 #inches
```

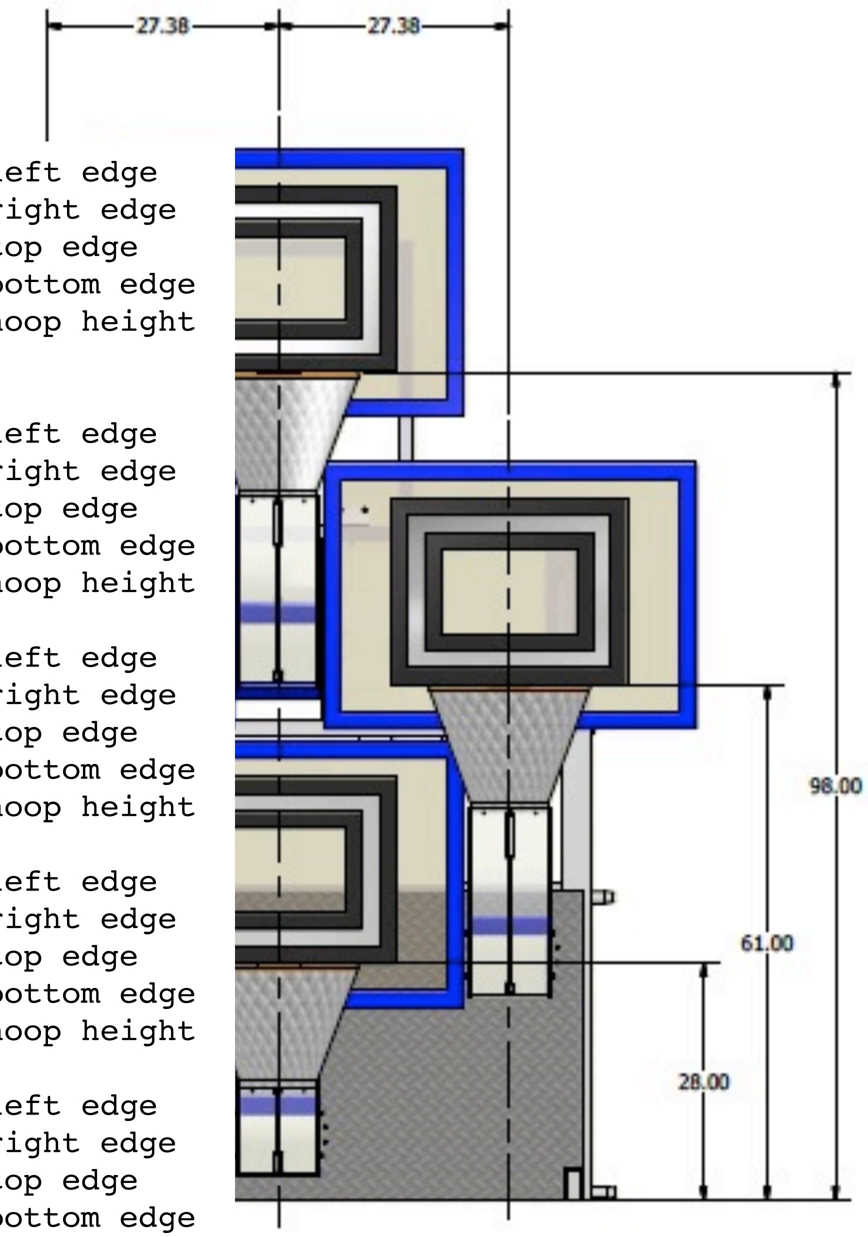


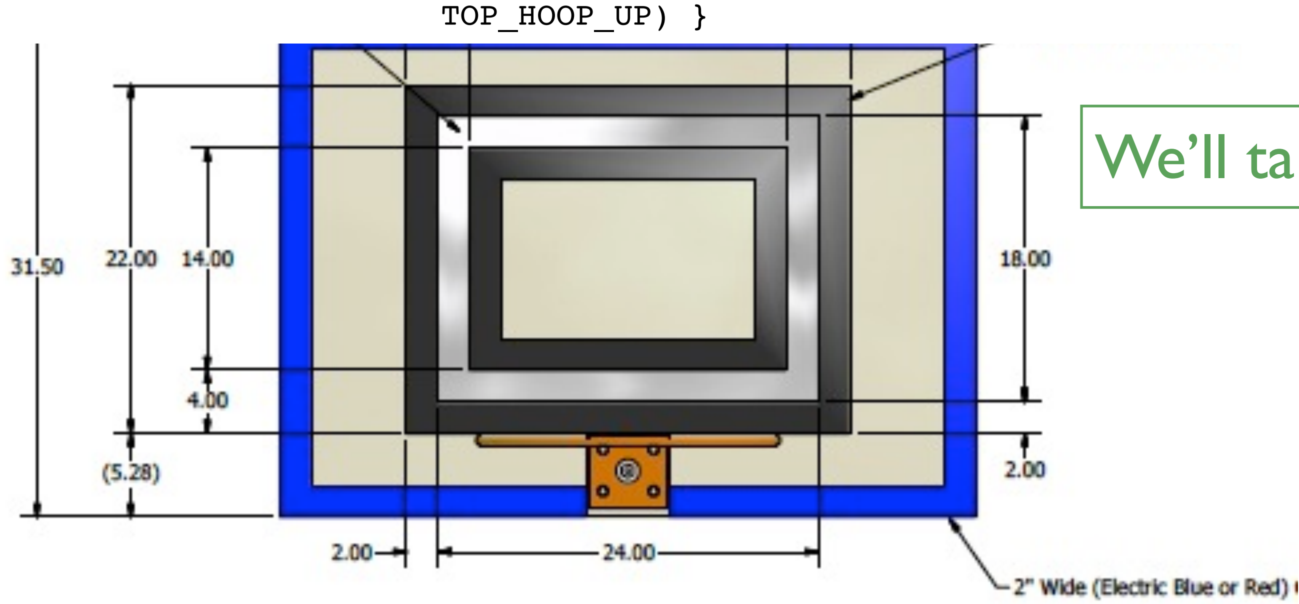


Step 0d
Define a table of locations of the known boxes.

Finally build the table.

```
target_locs = { LOW:         target_position( 0.0+TARGET_LEFT_DELTA,              # l = left edge
                                    0.0+TARGET_RIGHT_DELTA,                       # r = right edge
                                    LOW_HOOP_UP+TARGET_TOP_DELTA,                 # t = top edge
                                    LOW_HOOP_UP+TARGET_BOTTOM_DELTA,              # b = bottom edge
                                    LOW_HOOP_UP),                                 # h = hoop height

        # Default an unknown middle level hoop to be the left hoop
        MID_UNKNOWN: target_position( MID_LEFT_HOOP_EAST+TARGET_LEFT_DELTA,       # l = left edge
                                    MID_LEFT_HOOP_EAST+TARGET_RIGHT_DELTA,        # r = right edge
                                    MID_HOOP_UP+TARGET_TOP_DELTA,                 # t = top edge
                                    MID_HOOP_UP+TARGET_BOTTOM_DELTA,              # b = bottom edge
                                    MID_HOOP_UP),                                 # h = hoop height

        MID_LEFT:    target_position( MID_LEFT_HOOP_EAST+TARGET_LEFT_DELTA,       # l = left edge
                                    MID_LEFT_HOOP_EAST+TARGET_RIGHT_DELTA,        # r = right edge
                                    MID_HOOP_UP+TARGET_TOP_DELTA,                 # t = top edge
                                    MID_HOOP_UP+TARGET_BOTTOM_DELTA,              # b = bottom edge
                                    MID_HOOP_UP),                                 # h = hoop height

        MID_RIGHT:   target_position( MID_RIGHT_HOOP_EAST+TARGET_LEFT_DELTA,      # l = left edge
                                    MID_RIGHT_HOOP_EAST+TARGET_RIGHT_DELTA,       # r = right edge
                                    MID_HOOP_UP+TARGET_TOP_DELTA,                 # t = top edge
                                    MID_HOOP_UP+TARGET_BOTTOM_DELTA,              # b = bottom edge
                                    MID_HOOP_UP),                                 # h = hoop height

        TOP:         target_position( 0.0+TARGET_LEFT_DELTA,                      # l = left edge
                                    0.0+TARGET_RIGHT_DELTA,                       # r = right edge
                                    TOP_HOOP_UP+TARGET_TOP_DELTA,                 # t = top edge
                                    TOP_HOOP_UP+TARGET_BOTTOM_DELTA,              # b = bottom edge
                                    TOP_HOOP_UP) }                                # h = hoop height
```



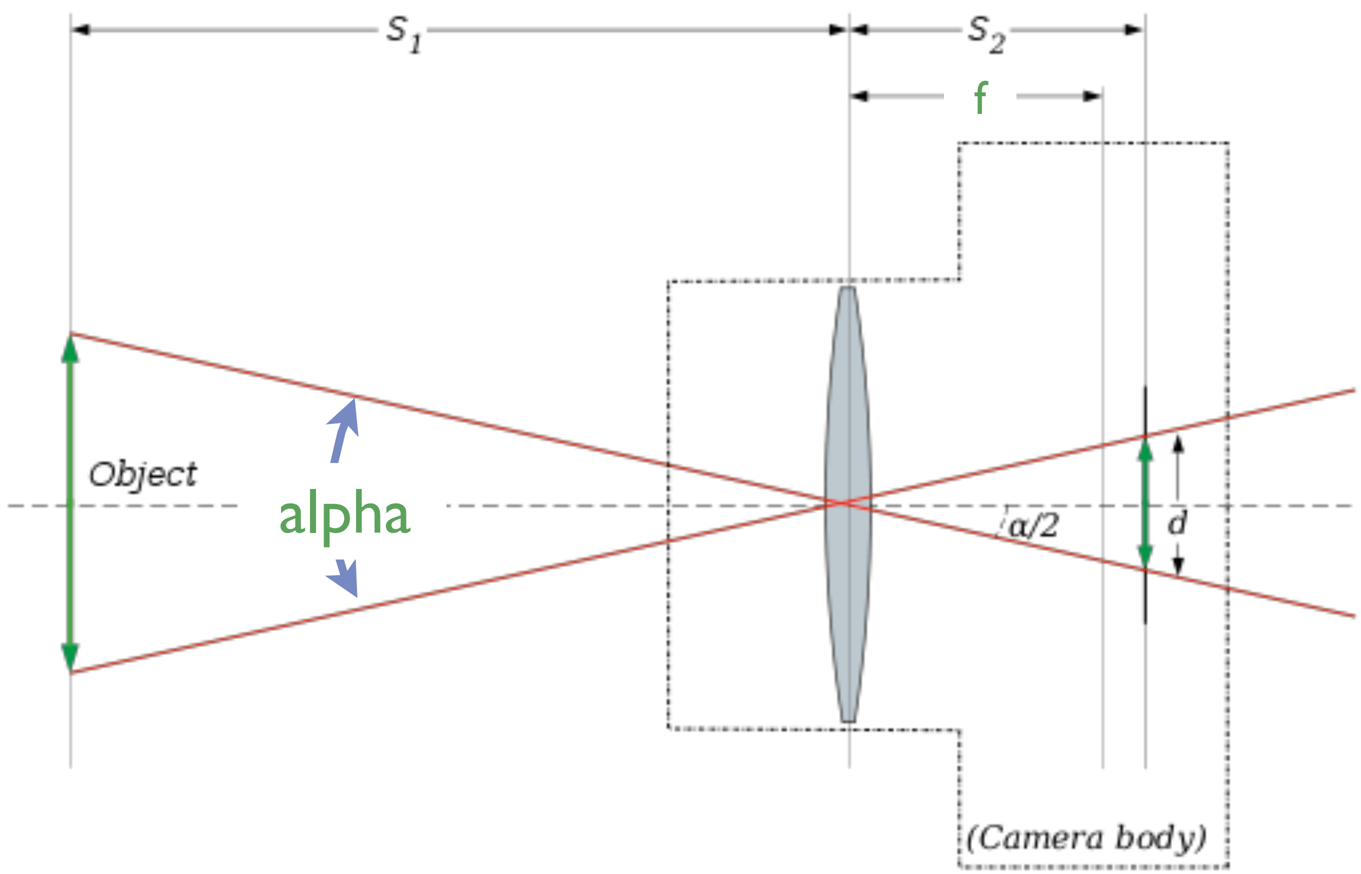


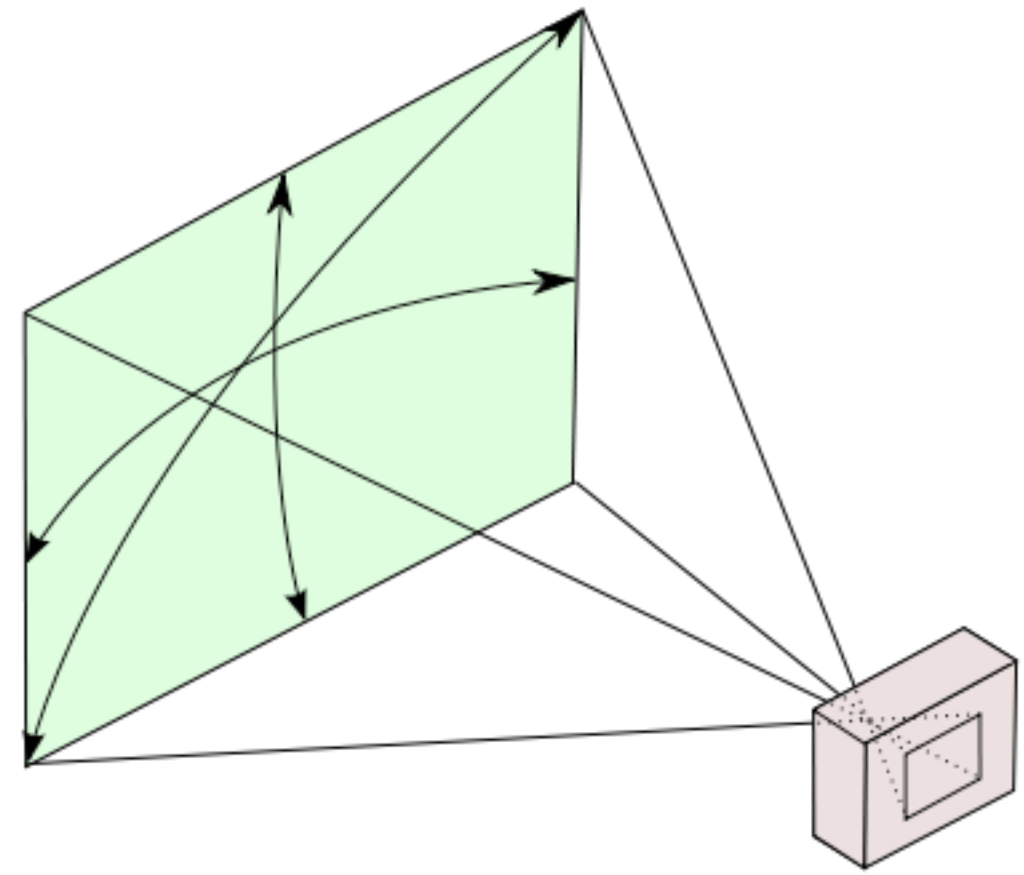
We'll talk about MID_UNKOWN later

Step 0e
Get to know our camera and angle of view equations

See http://en.wikipedia.org/wiki/Angle_of_view

When symmetrical across the center of the camera:
alpha = 2 atan(d / 2f)
where:
- alpha is the angle of view
- d is the distance on the image sensor
- f is the focal length

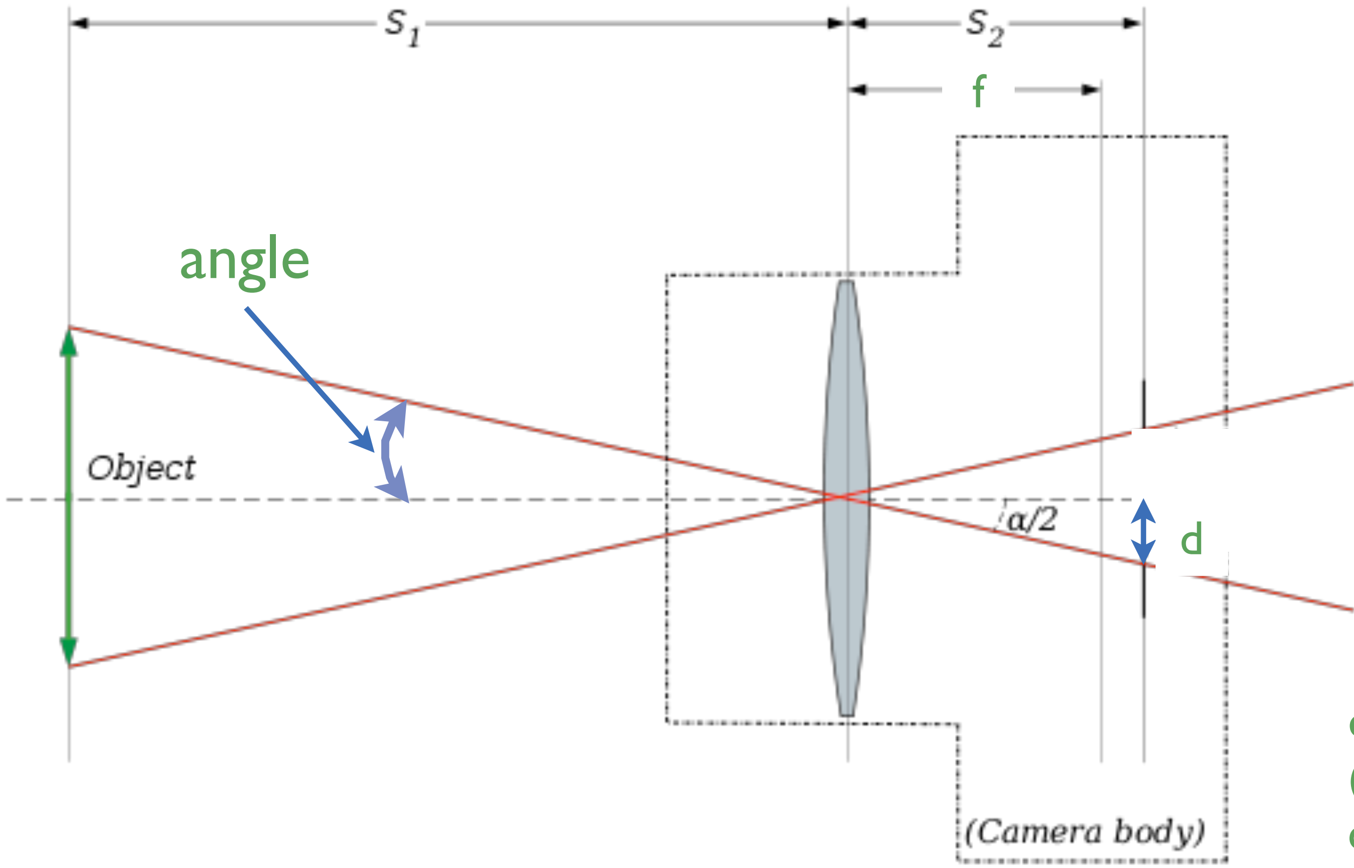


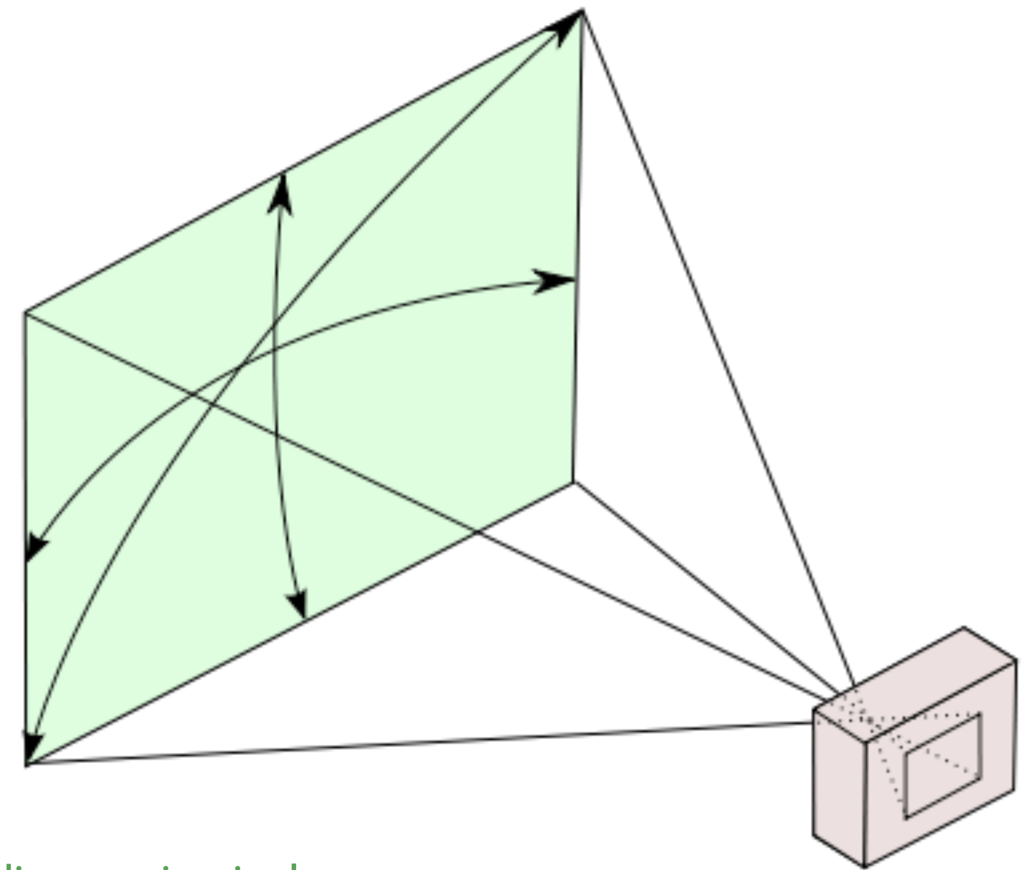


We know our width angle of view and camera pixels
so we can solve for focal length in pixels.

```
# Solve angle = 2 atan( d/(2f ) for f:
# f = d / ( 2*tan(angle/2))
#
camera_focal_len = camera_x_pixels / ( 2.0 * math.tan( camera_x_fov_rad / 2.0 ))   # pixels
```

Step 0f
Turn our camera into an angle measuring device





d is the distance in pixels (left,right,up, or down) from the center pixel in the camera field of view

angle is the correspond angle from the boresight of the camera

```
#
# Use the 1/2 angle form of angle = 2 atan( d/(2f))
#  angle = atan( d/f )
#
# This equation assumes d and angle is from the center of the field of view.
#
def pixel2rad( pixel ):
     return math.atan( pixel / camera_focal_len )  # + is up or right
```

Step 0g
Acquire the location of target boxes from a camera

In simple-locating, test.py simulates a camera taking a picture of rectangles on a wall and returns a list of rectangles with the pixel locations of the sides of the rectangles.

```
#
# Step 0g
#
# Project the image on to the camera, identify the complete targets in the field of view
#
constructed_rectangles = construct_test_image(
     math.radians(float(az)), # Rotate Right - (Azimuth) - radians
     0.0,                     # Tilt Up      - (Elevation) - radians
     float(east),             # Shift Right  - (East) - inches
     54.0,                    # Shift Up     - (Up) - inches
     float(-south) )          # Shift Forward- (North) - inches
```

We’re ready for Step 1!

Step 1
When we find a target record where we found the edges in pixels

test.py does this by appending a target object 'where.target' to the list of target objects, "targets"

```
#
# Start with an empty list of targets
targets = []
#
# Perform Step 1 on all the target rectangles in the field of view
#
for r in constructed_rectangles:
        #                      edges: left, right,  top, bottom      : in pixels
        targets.append( where.target( r[0],  r[1], r[2],    r[3] ) )

# Perform Steps 2 through 12 on the target set of rectangles in the field of view
#
calc_az, calc_east, calc_south = where.where( targets )
```

In python, the self parameter is automatically added to the where.target function call to the __init__ constructor to make the object.

where.py records the edges in target object constructors.

```
class target:
#
# Step 1:
# When we find a target record where we found the edges in pixels:
#
     def __init__(self,l,r,t,b):
          self.left_pixels = l
          self.right_pixels = r
          self.top_pixels = t
          self.bottom_pixels = b
          self.pos = UNKNOWN
```

Key Concept. For each target object:

```
target.left_pixels   = l = r[0] = location of left edge
target.right_pixels  = r = r[1] = location of right edge
target.top_pixels    = t = r[2] = location of top edge
target.bottom_pixels = b = r[3] = location of bottom edge
```

Step 2
For each target, convert the pixel locations to angles from the center line of the camera.

In where.py the function "where.where" does this by calling est_initial_angles.

```
#
#  Given a list of found rectangles,
#  invoke steps 2 through 12 on the rectangles
#
def where( rectangles ):
     global debug_found

# Invoke steps 2 through 6 for each target found
#
     for r in rectangles:
          r.est_initial_angles()
```

est_initial_angles converts the pixel locations to angles.

```
#
# Step 2:
# Convert the pixel locations to angles from the center line of the camera:
#
     def est_initial_angles(self):
          self.left_rad   = pixel2rad( self.left_pixels     - camera_x_pixels / 2.0 )
          self.right_rad  = pixel2rad( self.right_pixels    - camera_x_pixels / 2.0 )
          self.top_rad    = pixel2rad( self.top_pixels      - camera_y_pixels / 2.0 )
          self.bottom_rad = pixel2rad( self.bottom_pixels   - camera_y_pixels / 2.0 )
```

We created this pixel to angle conversion "`pixel2rad()`" at Step 0f.

"`self.left_pixels-camera_x_pixels/2.0`" is horizontal distance in pixels from the center of camera sensor.

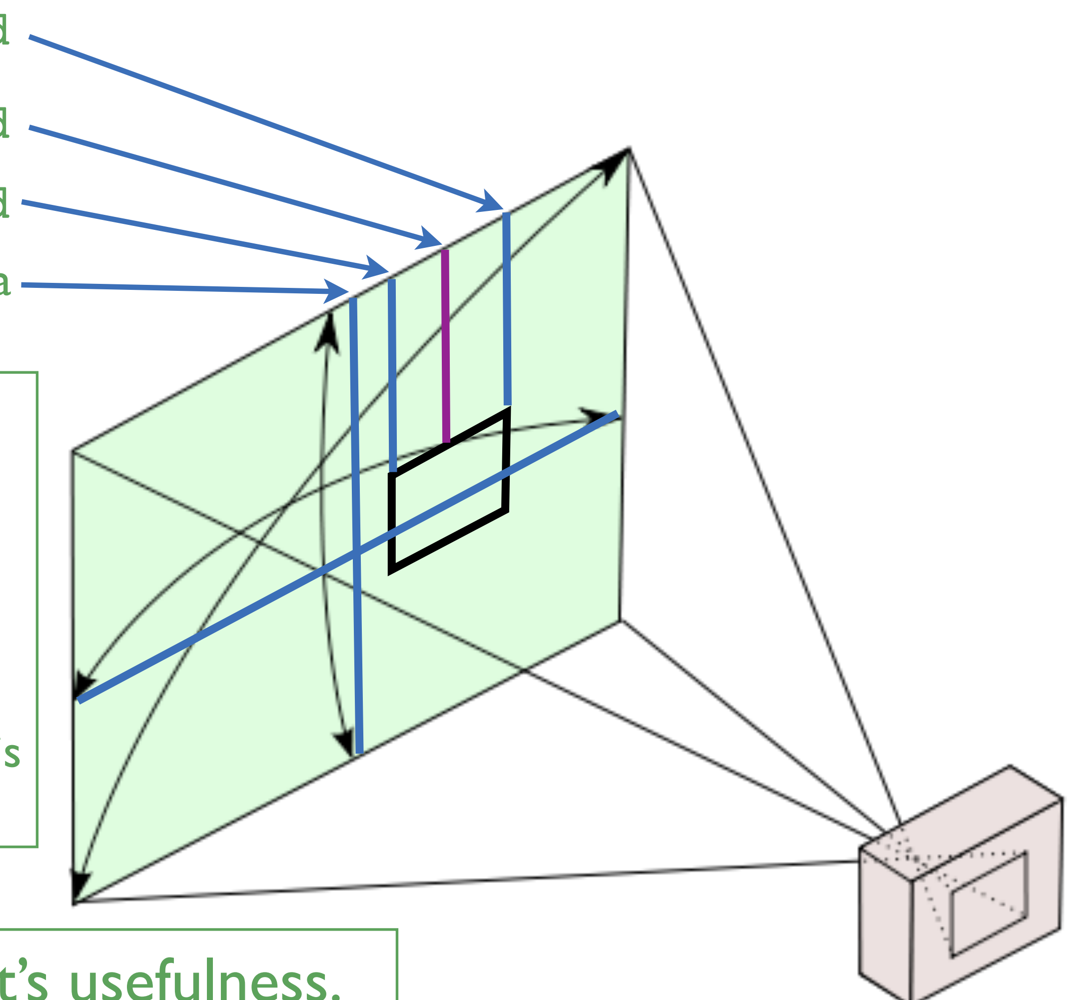
`pixel2rad()` the returns positive angles for objects that are above or right of the camera center line, negative angles for objects that are below or left of the center line.

Step 3
Estimate the azimuth and elevation from camera on the robot to the center of each target

In est_initial_angles(self):

```
#
# Step 3:
# Azimuth is left to right angle from the center line of the camera. +angles are to the right.
# Elevation is down to up angle from the center line of the camera. +angles are up.
#
# Estimate the Azimuth and Elevation from the camera to the center of the target.
#
        self.azimuth_rad   = (self.left_rad + self.right_rad) / 2.0          # + is right
        self.elevation_rad = (self.top_rad  + self.bottom_rad) / 2.0 - camera_pitch_error # + is up
```

This azimuth and elevation is relative to the robot, not the field.



This is a hueristic to estimate roughly where the center of the target is. It ignores several problems. Including:

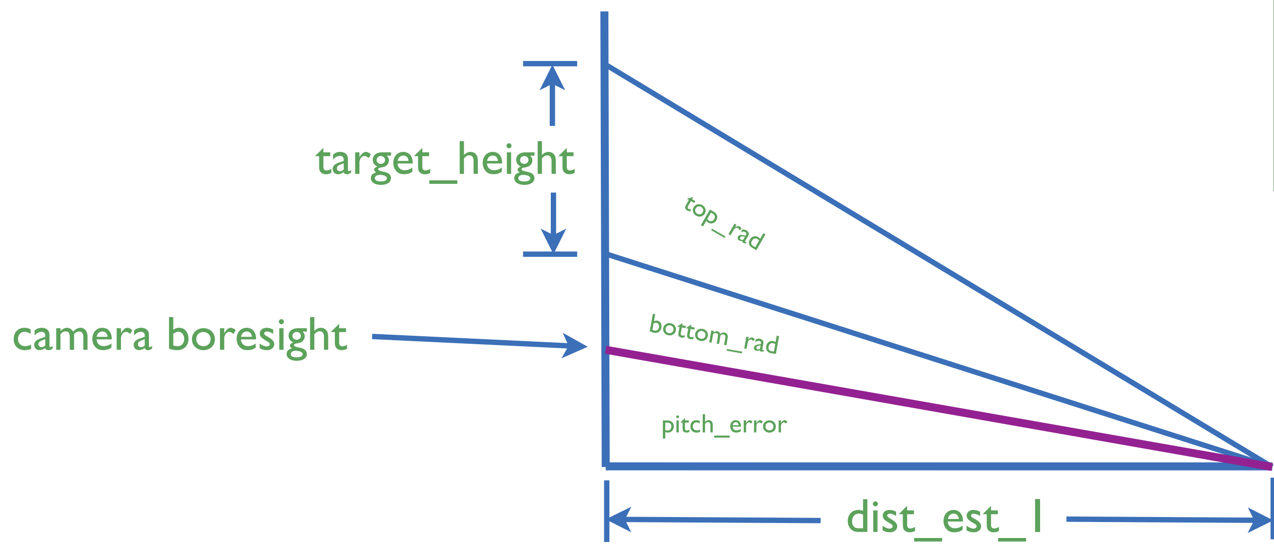
- We are averaging circular coordinates to estimate the center of an object in linear coordinate system.
- The camera is not orthogonal to the wall.
- The computer vision bounding box is rectangular, but the target is a trapezoid, in camera space.

This is fine. We just can't use this estimate beyond it's usefulness.

Step 4
Initial estimate of the distance to the targets.

In est_initial_angles(self):

```
# Step 4:

self.dist_est_1 = target_height / ( math.tan(self.top_rad+camera_pitch_error)
                                   -math.tan(self.bottom_rad+camera_pitch_error) )
```



We use target_height for our initial distance estimate because it is relatively immune to variations in robot azimuth, unlike target_width.

target_height = dist_est_1 * tan(top_rad+pitch_error) - dist_est_1 * tan(bottom_rad+pitch_error)

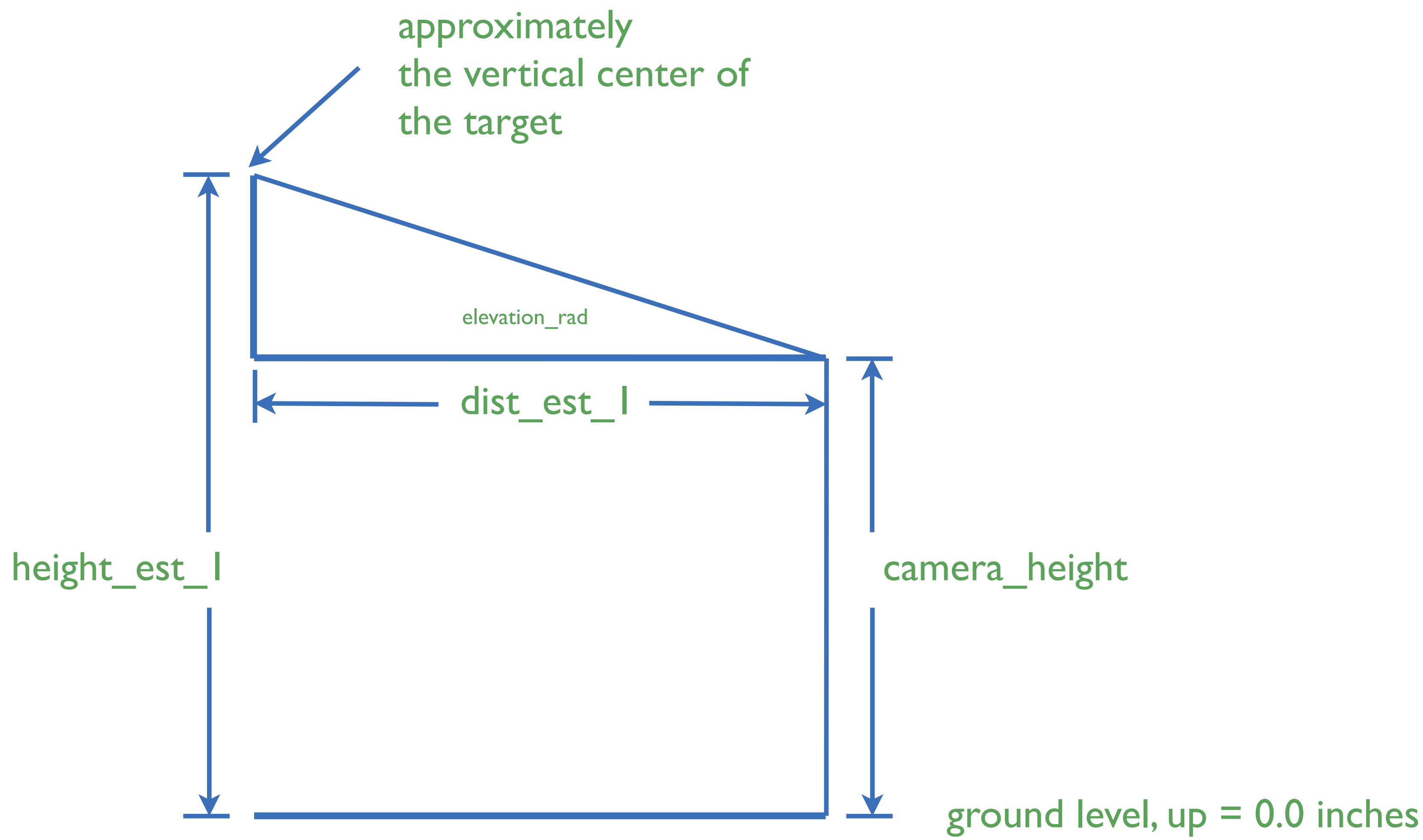
solving for dist_est_1....

target_height / (tan(top_rad+pitch_error) - tan(bottom_rad+pitch_error)) = dist_est_1

Step 5
Initial estimate of the vertical center of the targets.

In est_initial_angles(self):

```
# Step 5:
# Initial estimate of the height of the center of this target above ground based upon the
# distance to the target, the angle of the target, and the camera height
#

          self.height_est_1 = self.dist_est_1 * math.tan(self.elevation_rad) + camera_height
```



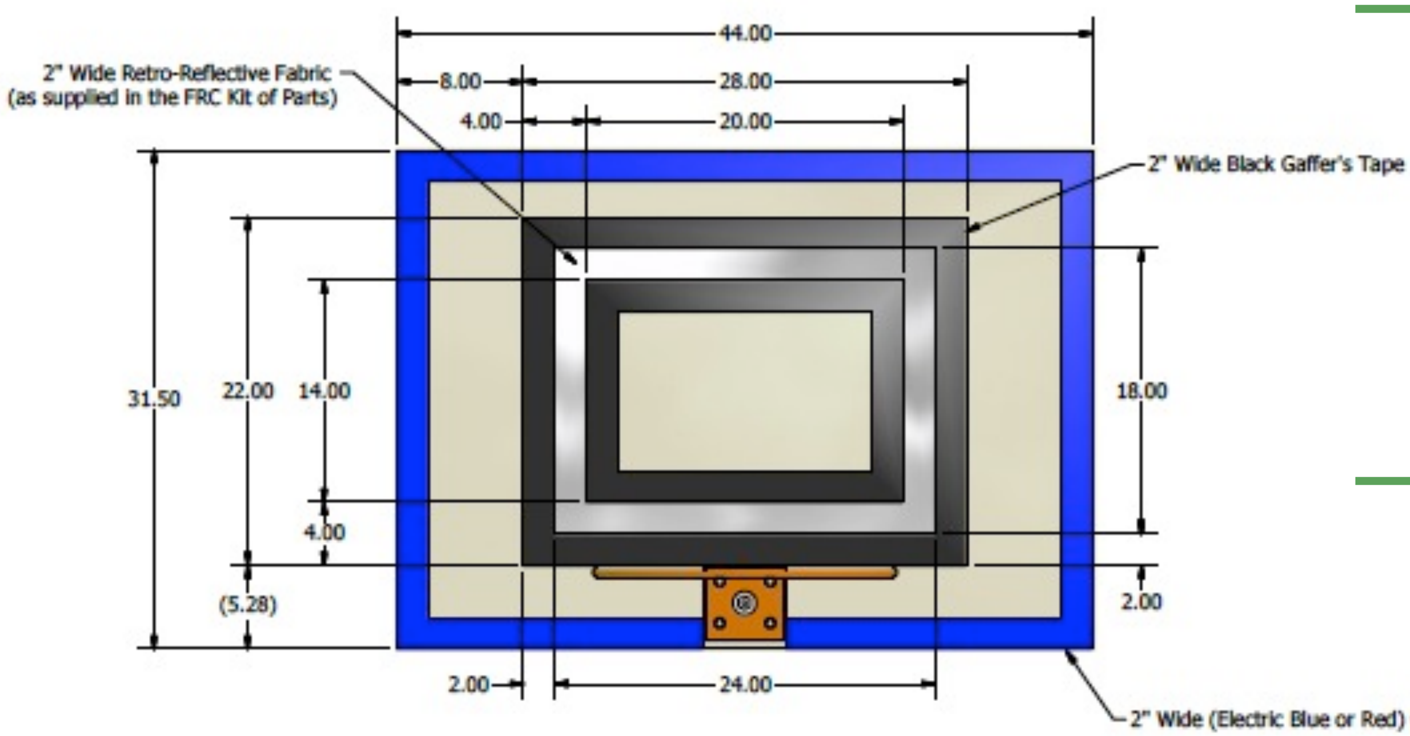
Step 6
Classify the targets as low, middle, or top based upon their height_est_1.

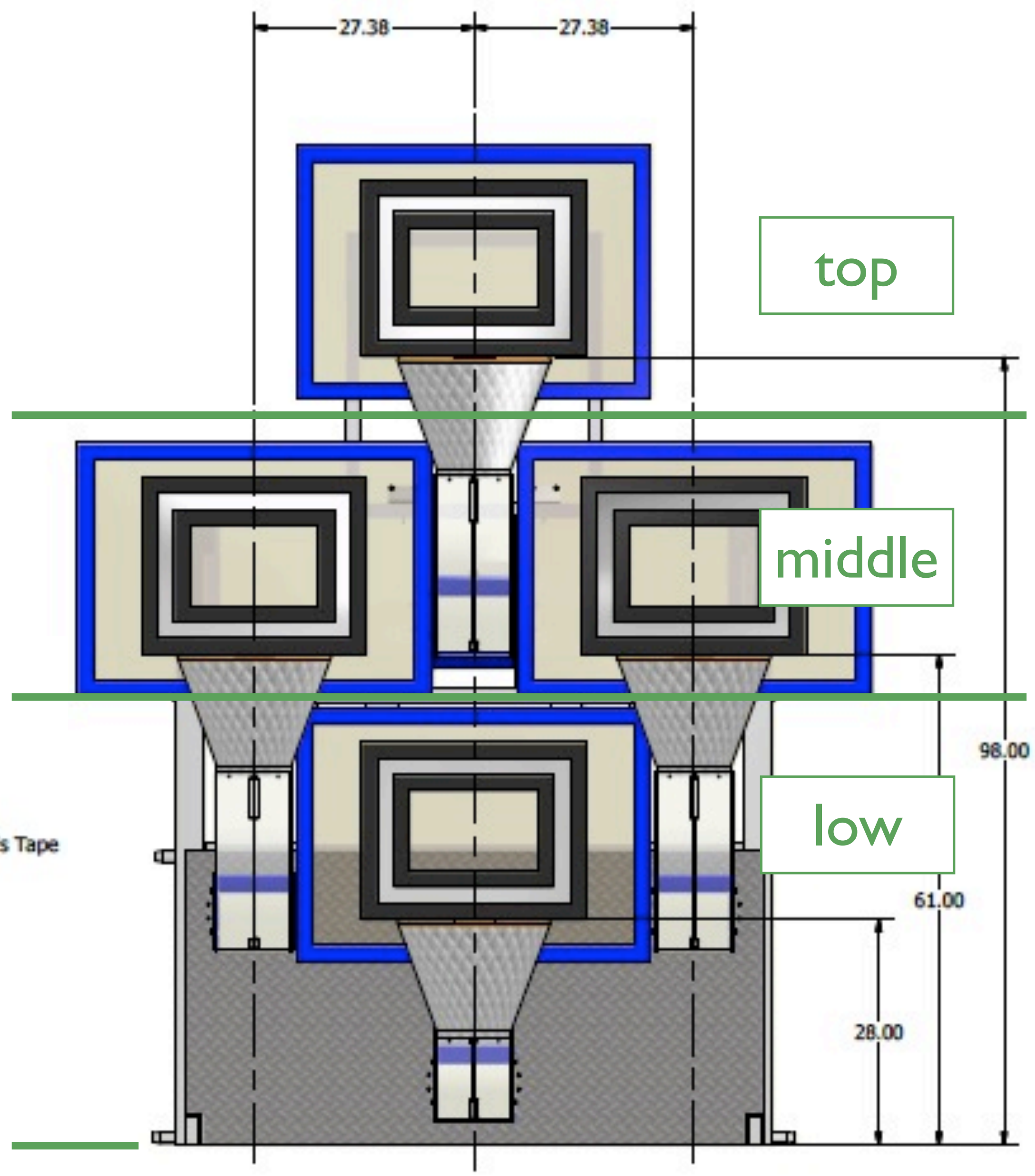
In est_initial_angles(self):

```
#
# Step 6:
# Classify the target as a low, middle or top target based upon it's height above ground.
#
          if ( self.height_est_1 < 55.5 ):
               self.level = LOW
          elif ( self.height_est_1 < 90.5 ):
               self.level = MID_UNKNOWN
          else:
               self.level = TOP
```

center low = 28 + 2 + 18/2 = 39 in.
center middle = 61 + 2 + 18/2 = 72 in.
center top = 98 + 2 + 18/2 = 109 in.

low to middle threshold = (39 + 72)/2 = 55.5 in.
middle to top threshold = (72+109)/2 = 90.5 in.





Step 7
Find the center target azimuth that is most left and most right.

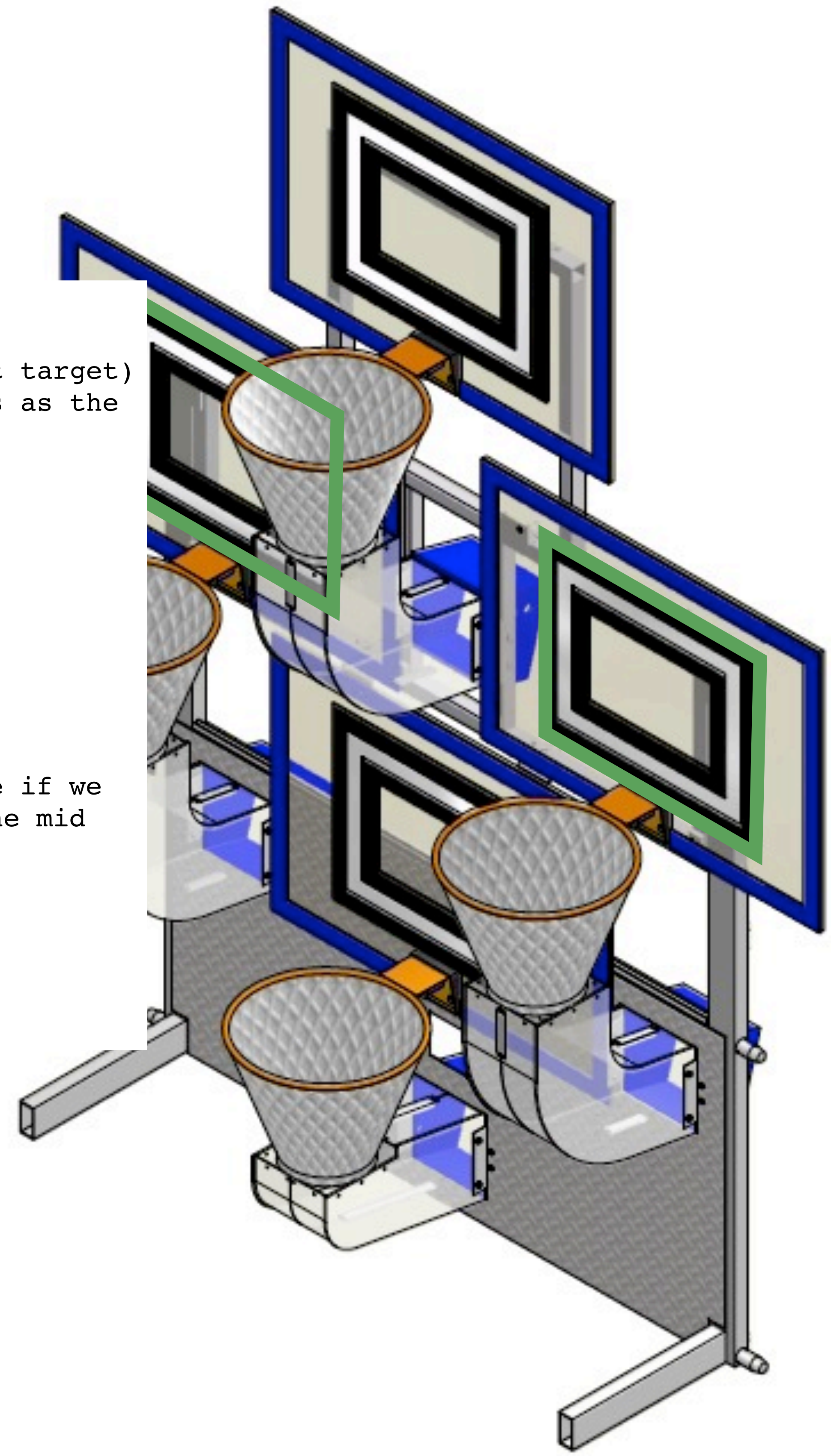
In where.py the function “where.where”:

```
# Step 7
# Find the center target azimuth that is most left and most right
#
     min_azimuth = +pi   # start at +180 which is out of view to right
     max_azimuth = -pi   # start at -180 which is out of view to left

     for r in rectangles:
          min_azimuth = min( min_azimuth, r.azimuth_rad )
          max_azimuth = max( max_azimuth, r.azimuth_rad )
```

Step 8
Classify the middle targets as MID_LEFT and MID_RIGHT

In where.py classify_pos() does this:

```
#
# Step 8:
# Given the minimum azimuth (most left target), and maximum azimuth (most right target)
# if we have identified more than one target, classify the middle level targets as the
# left or the right middle.
#
     def classify_pos( self, min_az, max_az ):
          if self.level == MID_UNKNOWN:
               if min_az == max_az:
                    self.pos = MID_UNKNOWN
               elif self.azimuth_rad == min_az:
                    self.pos = MID_LEFT
               elif self.azimuth_rad == max_az:
                    self.pos = MID_RIGHT
               else:
                    self.pos = MID_UNKNOWN # should not reach this line, becaue if we
                                           # found a mid and another target, the mid
                                           # should be min_az or max_az
          else:
               self.pos = self.level
```



We classify the middle targets as MID_LEFT and MID_RIGHT based upon their center azimuth being at the left or right edge of the azimuth's we've found.

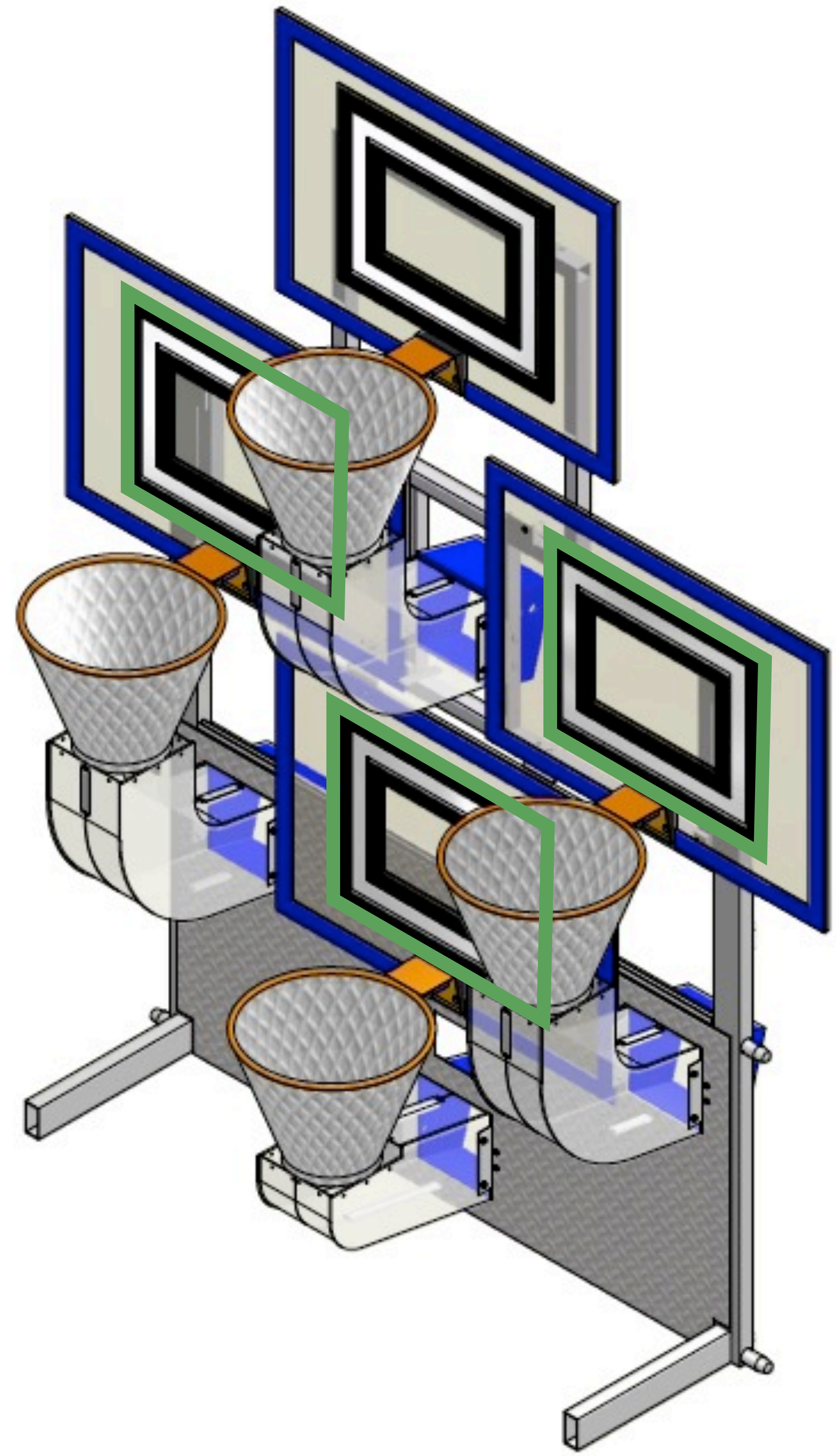
Now, if we've found at least two targets we should know which ones they are and where they are on the wall.

It's getting time to estimate where the camera is.

Step 9
Find the leftmost and rightmost targets

In where.py the function "where.where" runs step 9 for each target

```
# Step 9
# Identify the left most and right most targets.

     leftmost = MID_RIGHT+1
     rightmost = MID_LEFT-1
     for r in rectangles:
          if r.pos < leftmost :
               leftmost = r.pos
               left = r

          if r.pos > rightmost :
               rightmost = r.pos
               right=r
```



For example, if we found the MID_LEFT, LOW, and MID_RIGHT targets, select the MID_LEFT and MID_RIGHT targets.

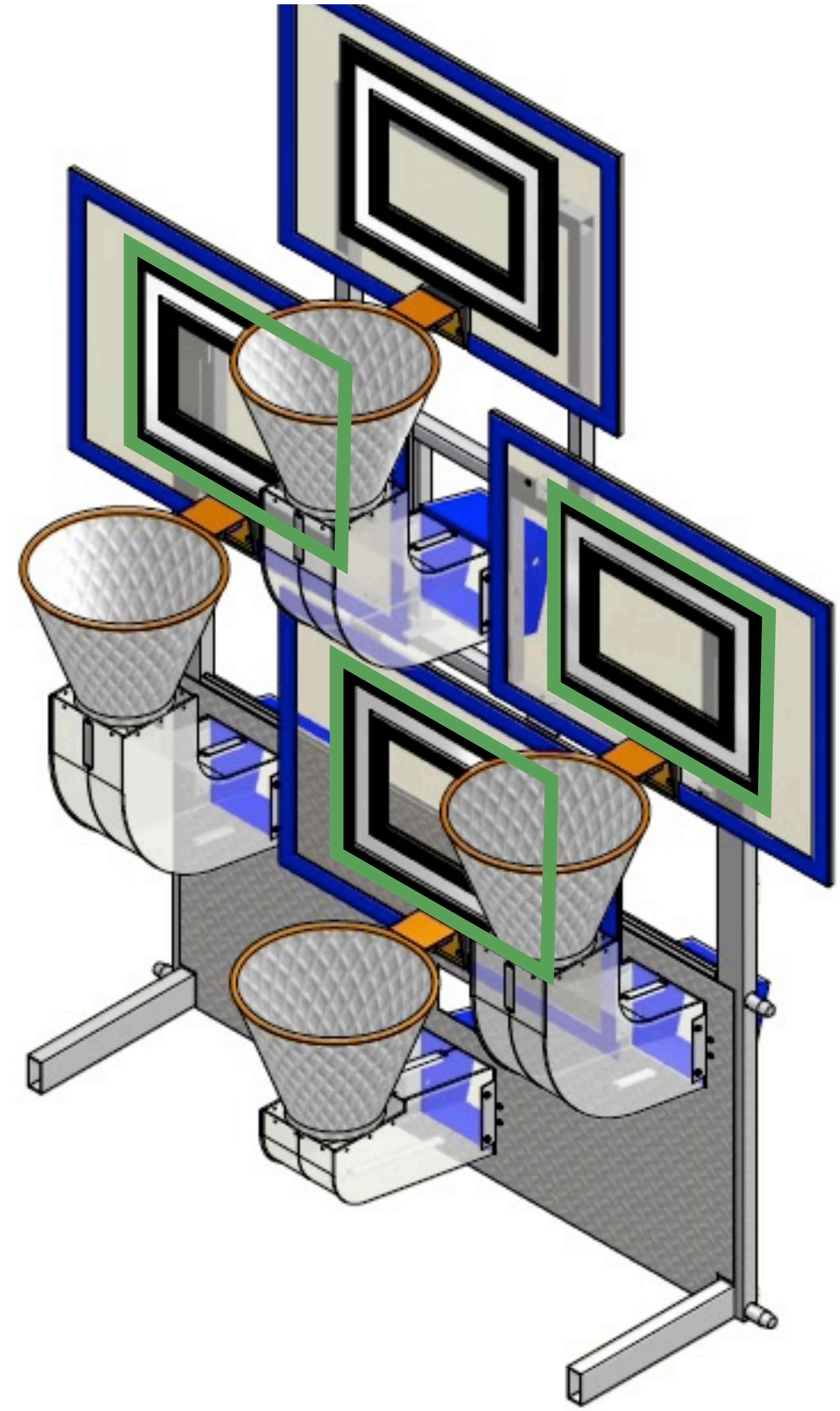
We're doing this because we're looking for vertical lines in the camera field of view that are far apart so we have lots of pixels of to calculate where the camera is at in step 11.

Step 10
Check if we found targets to know the angles to 3 vertical lines.

In where.py the function "where.where" runs step 10 for each target

```
# Step 10
# If we have found two different targets, and they are not
# just the top and bottom targes.
# Then perform step 11 on two sets of 3 angles angles
# two the 3 targets.
#
     if (leftmost != MID_RIGHT+1) and (leftmost != rightmost)
          and not((leftmost==LOW) and (rightmost==TOP)) :
```

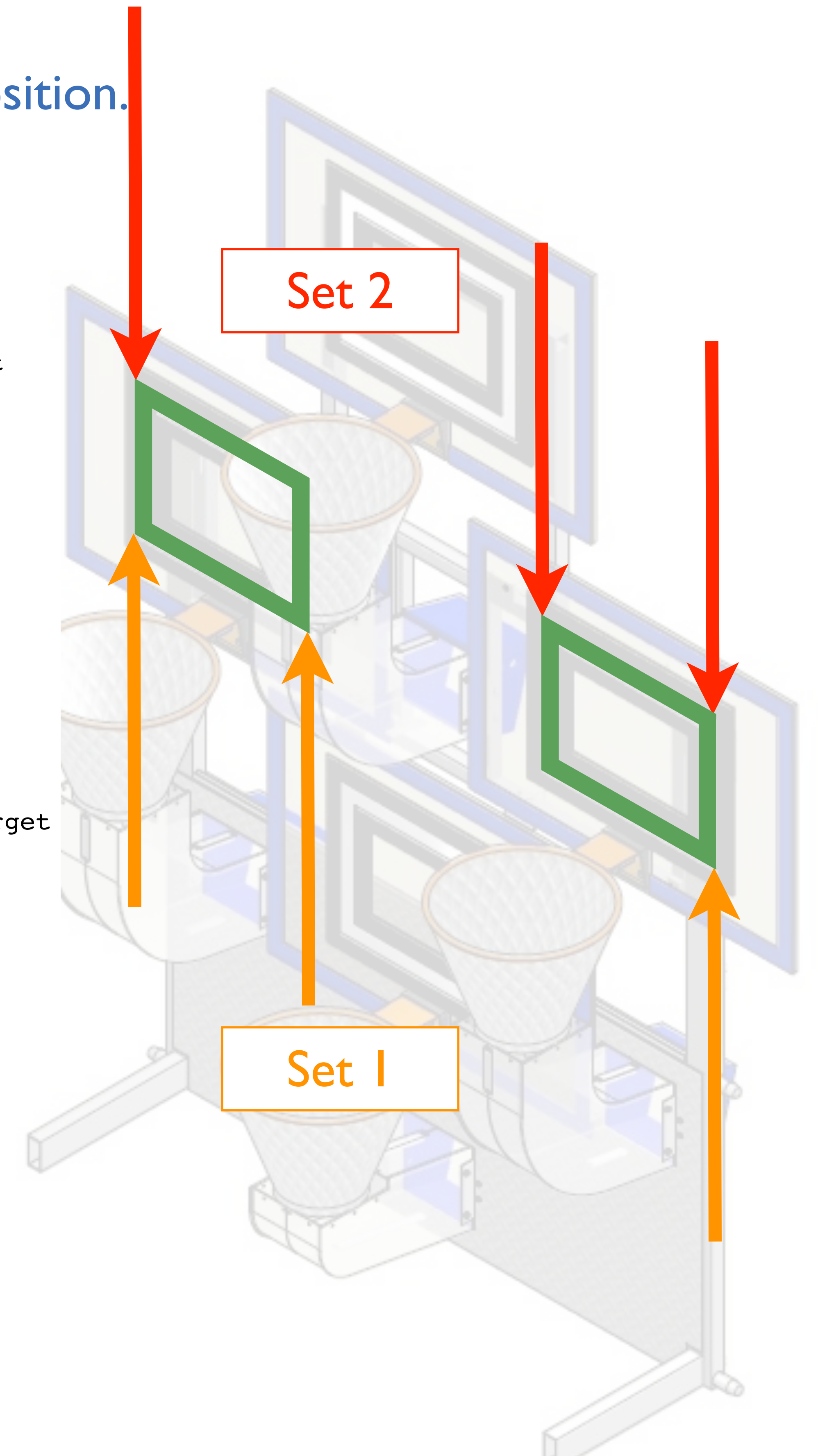
If we have just found the top and bottom targets, because they are directly above each other we don't have angles to 3 different vertical lines.



Step 11a.1
Select two sets of 3 vertical lines to use to estimate our position.

In where.py, the function "where.where" runs step 11a for each target

```
# Perform step 11 using the using both vertical edges of the far left target
# and the right edge of the far right target.
#
# See definition of field coordinate system.
# az1 and az2 are estimates of the camera heading in azimuth in radians
# east1 and east2 are estimates of the camera east position
# south1 and south2 are estimates of the camera south position
#
az1, east1, south1 = estimate_pos_3_sep_hrz_angles( left.left_rad,
                                  left.right_rad,
                                  right.right_rad,
                                  target_locs[left.pos].left_inches,
                                  target_locs[left.pos].right_inches,
                                  target_locs[right.pos].right_inches)

# Perform step 11 using the using the left vertical edge of the far left target
# and the both vertical edges of the far right target.
az2, east2, south2 = estimate_pos_3_sep_hrz_angles( left.left_rad,
                                  right.left_rad,
                                  right.right_rad,
                                  target_locs[left.pos].left_inches,
                                  target_locs[right.pos].left_inches,
                                  target_locs[right.pos].right_inches)
```

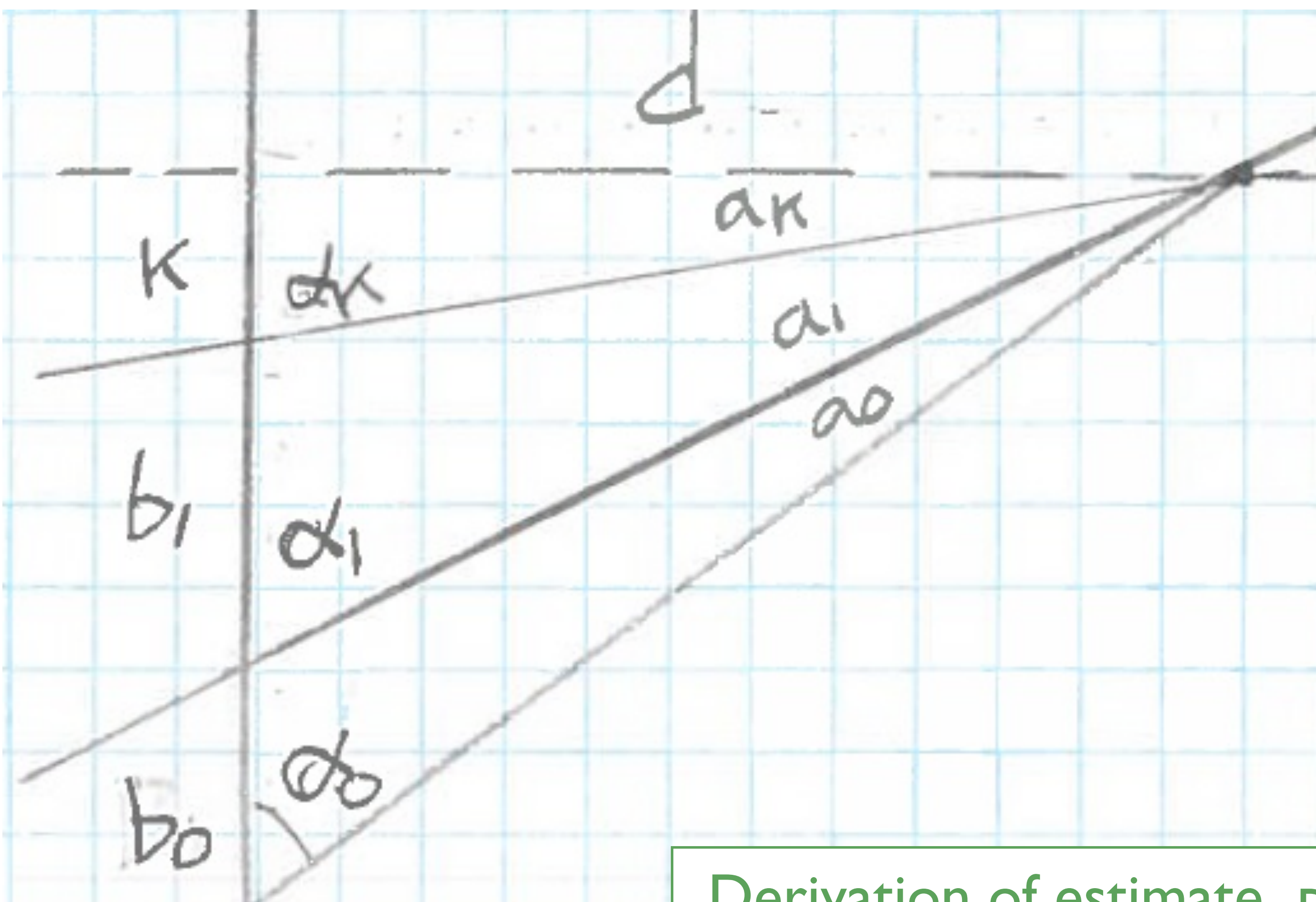


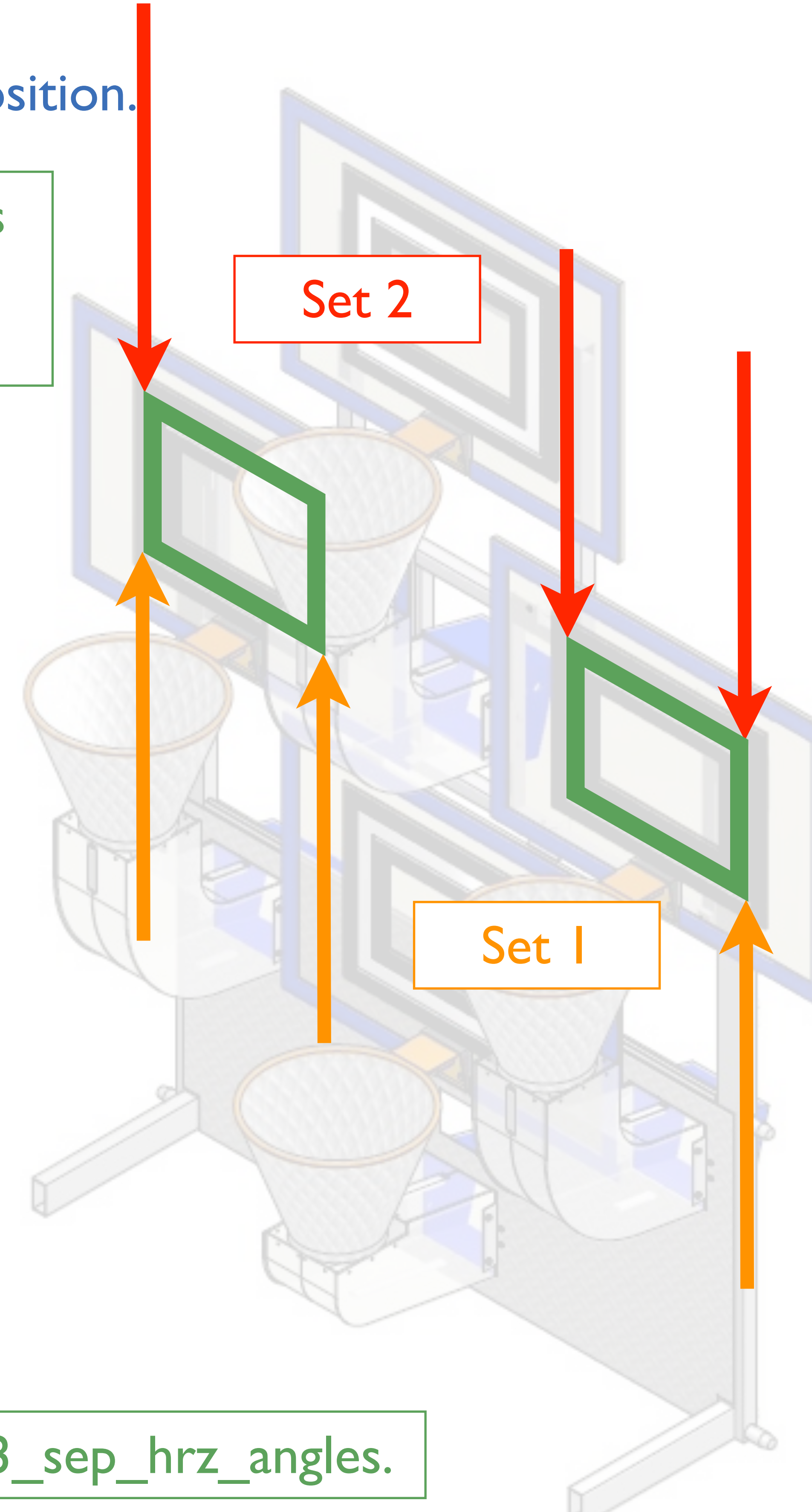
We use the two vertical lines that are farthest from each other on the targets. Then on the first set we use the inner vertical line on the left target. On the second set we use the inner vertical line on the right target.

Step 11a.2

Select two sets of 3 vertical lines to use to estimate our position.

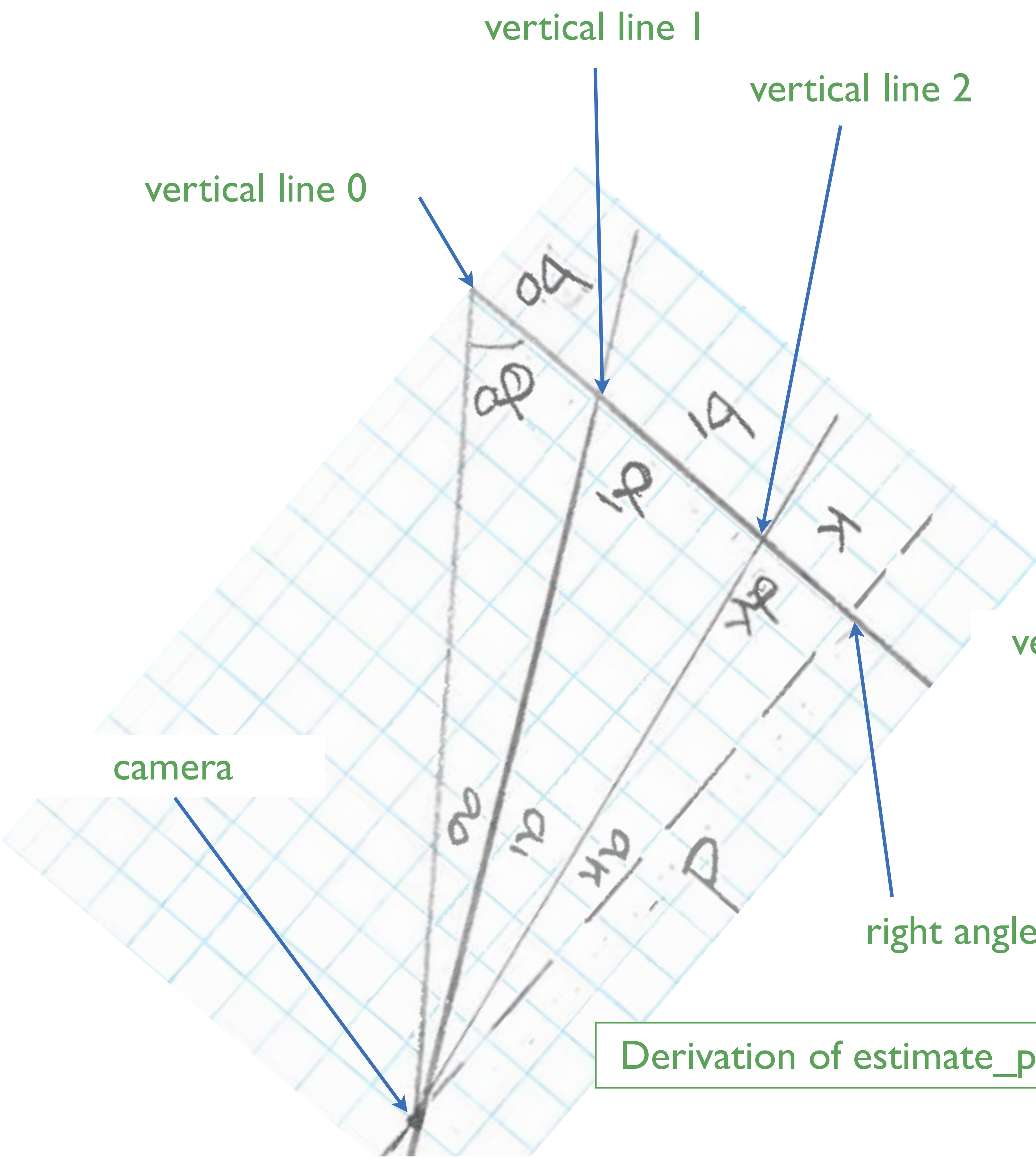
In where.py, the function estimate_pos_3_sep_hrz_angles calculates the camera azimuth from the angles to 3 points on a line, and the positions of those points on the line.

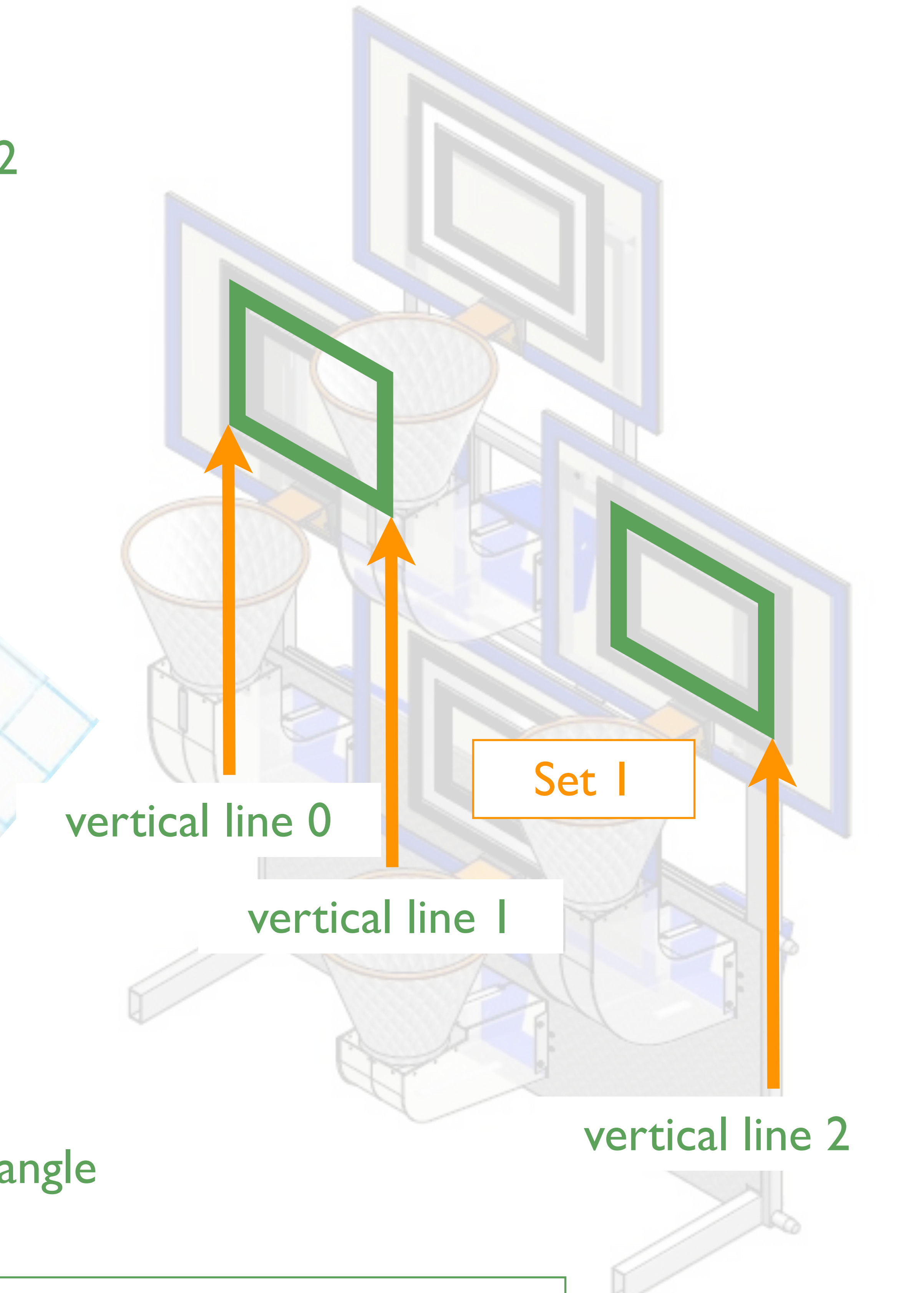




Derivation of estimate_pos_3_sep_hrz_angles.

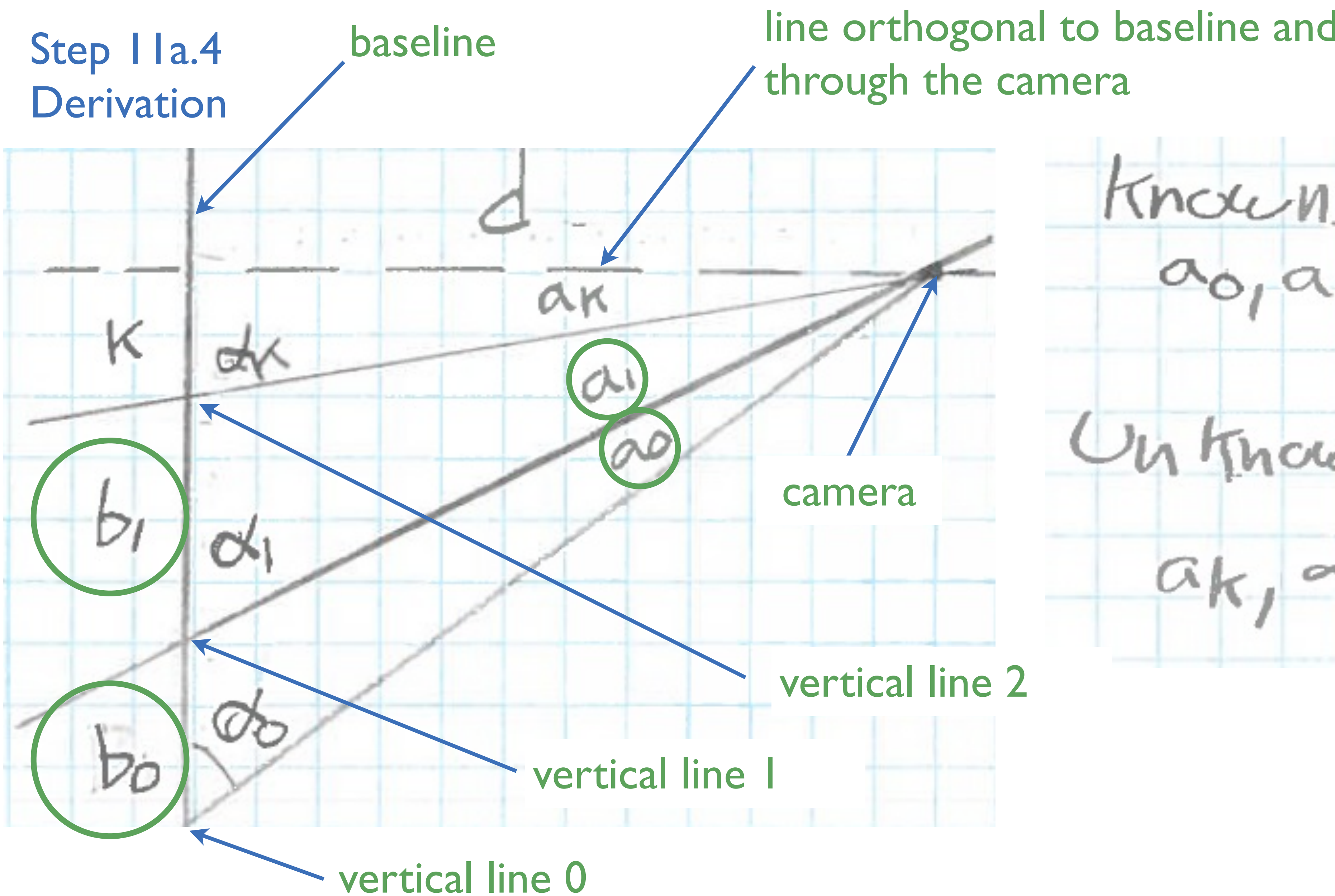
Step 11a.3
Derivation

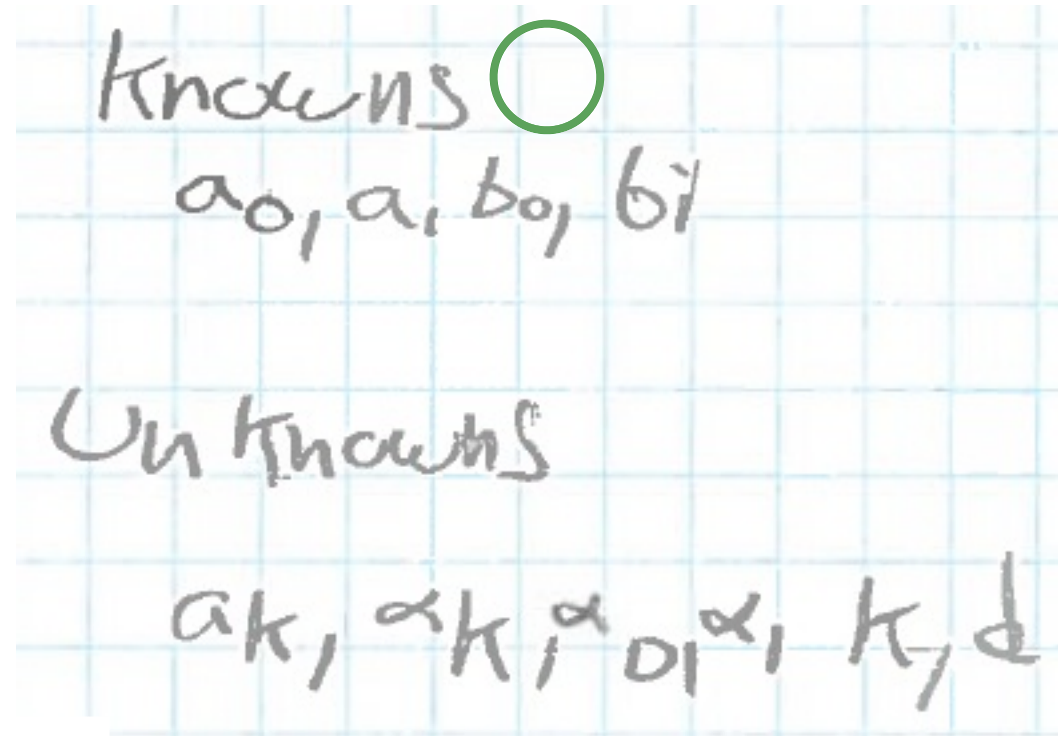




Derivation of estimate_pos_3_sep_hrz_angles.

Step 11a.4 Derivation





Knowns:

b_0 is distance from vertical line 0 to vertical line 1

b_1 is distance from vertical line 1 to vertical line 2

a_0 is angle from vertical line 0 to vertical line 1

a_1 is angle from vertical line 1 to vertical line 2

Unknowns:

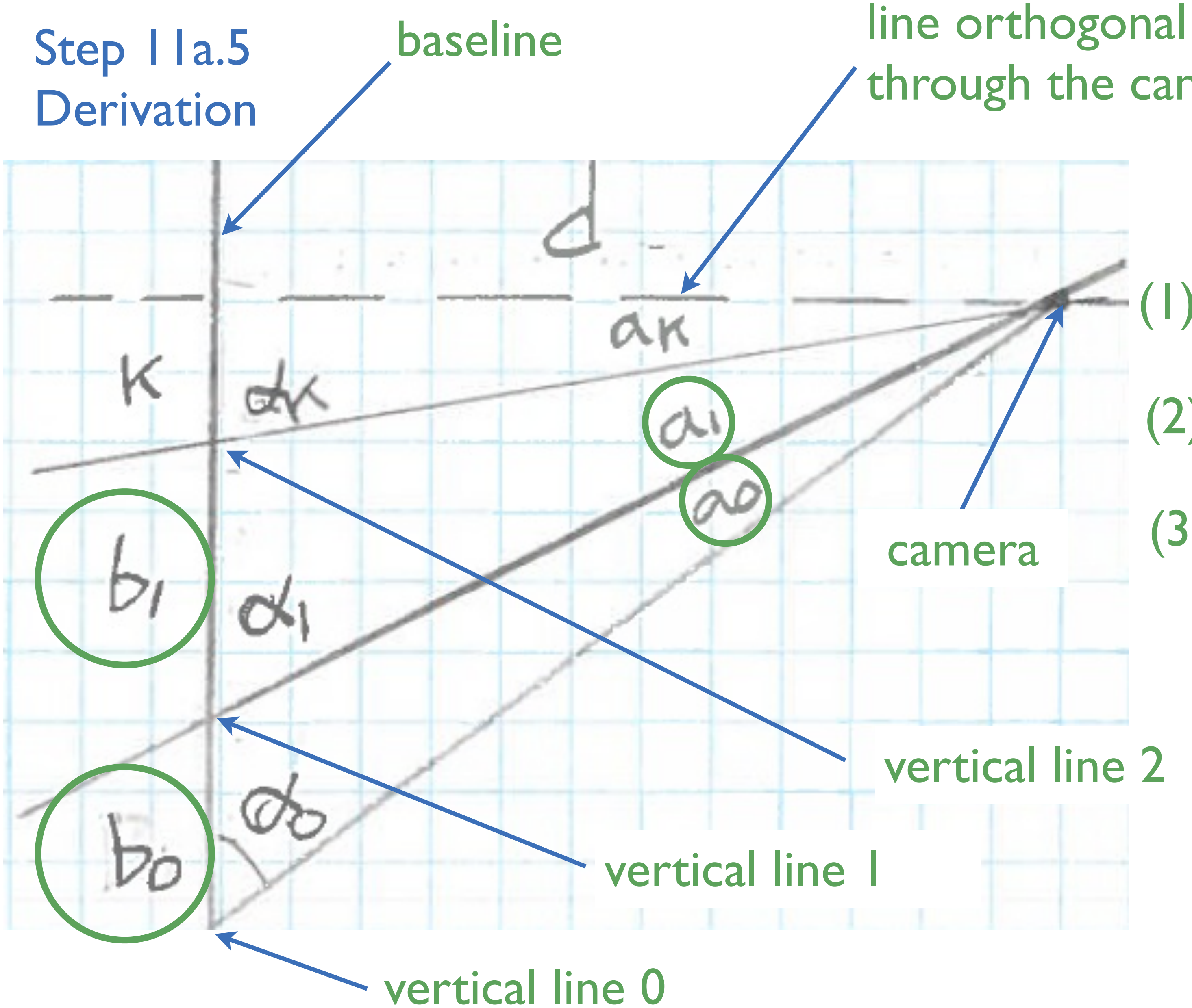
α_0, α_1, α_k are angles between the baseline and the camera

a_k is angle from line perpendicular to baseline to vertical line 2

k is distance along baseline from vertical line 2 to point on orthogonal line to camera

d is orthogonal distance from camera to baseline

Step 11a.5 Derivation



tangent equations for triangles with d as a side

(1) $k \tan(\alpha_k) = d$

(2) $(k + b_1) \tan(\alpha_1) = d$

(3) $(k + b_1 + b_0) \tan(\alpha_0) = d$

angles of triangles total 180 degrees

(4) $\alpha_k + a_k = \pi/2$

(5) $\alpha_1 + a_1 + a_k = \pi/2$

(6) $\alpha_0 + a_0 + a_1 + a_k = \pi/2$

re-order angle equations to solve for variables in tangent equations

(7) $\alpha_k = \pi/2 - a_k$

(8) $\alpha_1 = \pi/2 - a_1 - a_k$

(9) $\alpha_0 = \pi/2 - a_0 - a_1 - a_k$

(10) let $A = \pi/2 - a_k$

Step 11a.6
Derivation

tangent equations d=d=d, and substitute A into tangent angles

(11) $$d = k\tan(A) = (k+b_1)\tan(A-a_1) = (k+b_0+b_1)\tan(A-a_0-a_1)$$

We now have two equations in two variables (A) and (K) !

use identity tan(a-b) = [tan(a)-tan(b)]/[1+tan(a)tan(b)]

(12) $$d = k\tan(A) = (k+b_1)\frac{\tan(A)-\tan(a_1)}{1+\tan(A)\tan(a_1)} =$$

(13) $$= (k+b_0+b_1)\frac{\tan(A)-\tan(a_0+a_1)}{1+\tan(A)\tan(a_0+a_1)}$$

Step 11a.7
Derivation

multiple both sides of equation (12) by the denominator

(14)
$$k\tan(A)\left(1+\tan(A)\tan(a_1)\right) = (k+b_1)\left(\tan(A)-\tan(a_1)\right)$$

(15)
$$\cancel{k\tan(A)} + k\tan^2(A)\tan(a_1) = \cancel{k\tan(A)} - k\tan(a_1) + b_1\tan(A) - b_1\tan(a_1)$$

(16)
$$k\tan^2(A)\tan(a_1) + k\tan(a_1) = b_1\left(\tan(A)-\tan(a_1)\right)$$

(17)
$$k\tan(a_1)\left[\tan^2(A)+1\right] = b_1\left(\tan(A)-\tan(a_1)\right)$$

use identity $\tan^2(a) + 1 = \sec^2(a)$; and divide both sides by $\tan(a_1)$

(18)
$$k\left[\sec^2(A)\right] = \frac{b_1\left(\tan(A)-\tan(a_1)\right)}{\tan(a_1)}$$

Step 11a.8
Derivation

(19) Divide both sides of (17) by $\sec^2(A)$

$$(19)\quad k = \frac{b_1\left(\dfrac{\tan(A)}{\tan(a_1)} - 1\right)}{\sec^2(A)}$$

by similarity of algebra

$$(20)\quad (b_0+b_1)\left(\frac{\tan(A)}{\tan(a_0+a_1)} - 1\right)\cancel{\cos^2 A} = b_1\left(\frac{\tan(A)}{\tan(a_1)} - 1\right)\cancel{\cos^2(A)}$$

Apply steps (14) through (19) to equation (13) by similar algebra, and then substitute $1/\sec^2(A) = \cos^2(A)$ to obtain expression (20) which is also equal to k.

We now have one equation in one variable (A) !

Step 11a.9 Derivation

Divide both sides by $\cos^2(A)$, and then subtract by the left side to move all terms to the right side

(21) $$0 = \frac{b_1 \tan(A)}{\tan(a_1)} - b_1 - \frac{b_0 \tan(A)}{\tan(a_0+a_1)} - \frac{b_1 \tan(A)}{\tan(a_0+a_1)} + b_0 + b_1$$

gather terms

(22) $$0 = \frac{b_1 \tan(A)}{\tan(a_1)} - \frac{(b_0+b_1)\tan(A)}{\tan(a_0+a_1)} + b_0$$

factor tan(A)

(23) $$-b_0 = \tan(A)\left(\frac{b_1}{\tan(a_1)} - \frac{b_0+b_1}{\tan(a_0+a_1)}\right)$$

Step 11a.10
Derivation

Divide both sides by the same term

(24) $$\frac{-b_0}{\left(\frac{b_1}{\tan(a_1)} - \frac{b_0+b_1}{\tan(a_0+a_1)}\right)} = \tan(A)$$

Take the arc tangent of both sides, and re-apply equation (10)

(25) $$\tan^{-1}\left(\frac{-b_0}{\frac{b_1}{\tan(a_1)} - \frac{b_0+b_1}{\tan(a_0+a_1)}}\right) = A = \pi/2 - a_k$$

We can solve for a_k !

$a_k = \pi/2$ - atan($-b_0/(b_1/\tan(a_1)$ - $(b_0 + b_1)/\tan(a_0+a_1)$)

use atan2 see http://en.wikipedia.org/wiki/Atan2 to keep from dividing by zero and to place the angle in the right quadrant.

$a_k = \pi/2$ - atan2($-b_0$, $(b_1/\tan(a_1)$ - $(b_0 + b_1)/\tan(a_0+a_1)$)

Step 11a.11
Derivation

Identity from equations (1) and (2)

(26) $$k \tan(\alpha_k) = d = (k + b_1)\tan(\alpha_1)$$

Expand terms and subtract from both sides

(27) $$k \tan(\alpha_k) - k \tan(\alpha_1) = b_1 \tan(\alpha_1)$$

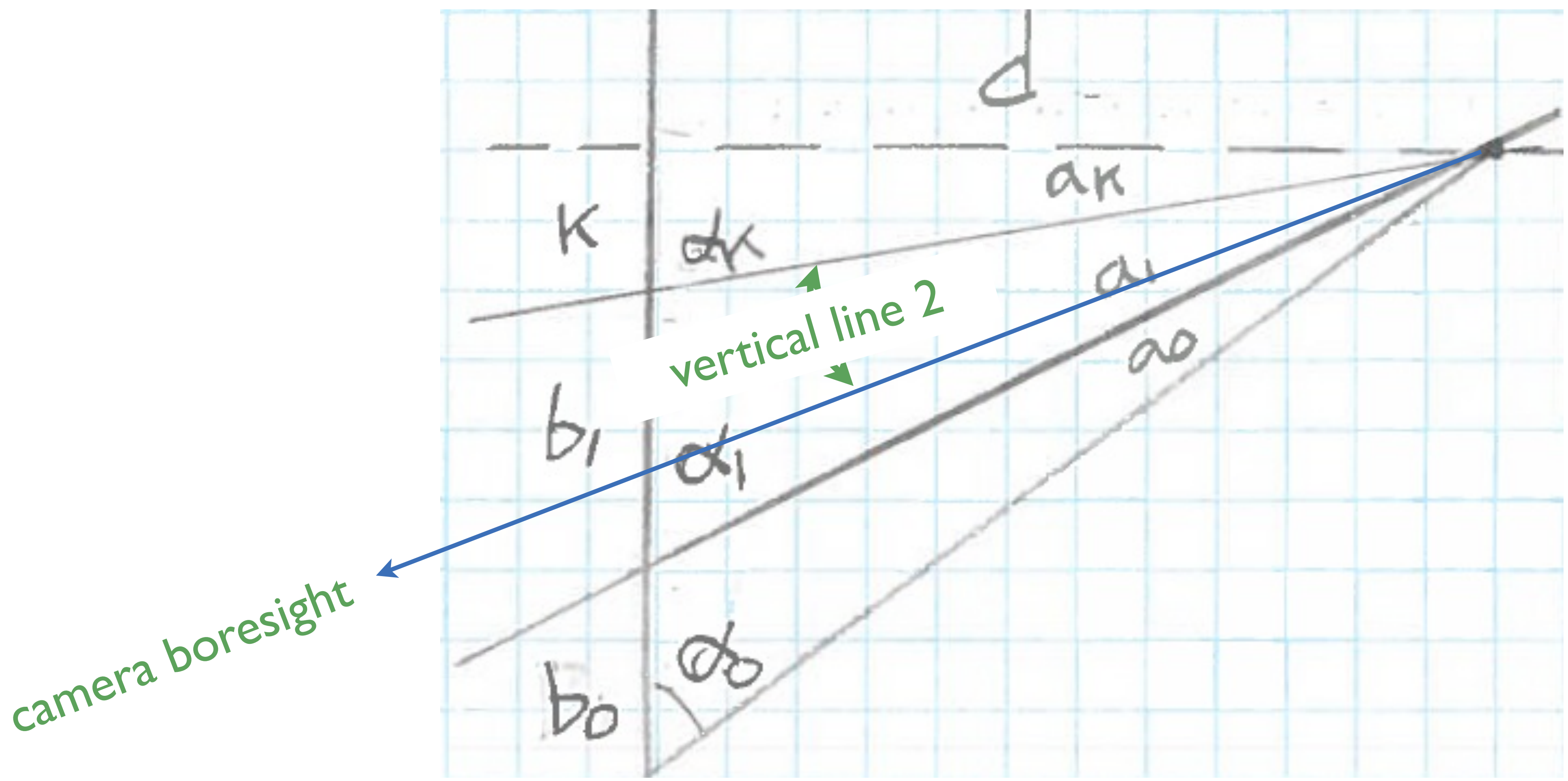
Factor k and divide both sides by [tan(α_k)-tan(α_1)]

(28) $$k = \frac{b_1 \tan(\alpha_1)}{\tan(\alpha_k) - \tan(\alpha_1)}$$

Use equation (7) to solve for α_k, equation (8) to solve for α_1 and equation (28) to solve for k, and then equation (1) to solve for d.

Step 11a.12
Derivation

Calculate Azimuth



azimuth should be = -(angle_of_vertical_line2+a_k)

but empirically we need to add π, TBD investigate the quadrant atan2 is returning:

azimuth = -(angle_of_vertical_line2+a_k) + π

Step 11b
Call estimate_pos_3_sep_hrz_angles one time for each of two sets of angles

```
#Step 11
# Given 3 camera angles to 3 veritcal lines along the wall of backboards, estimate the
# camera heading (azimuth), east position, and south position
#
# See https://github.com/MikeStitt/simple-locating-docs/blob/master/mathToFindLocationFromAnglesTo3PointsOnALine.pdf?raw=true
#
#
def estimate_pos_3_sep_hrz_angles( left_rad, mid_rad, right_rad, left_pos, mid_pos, right_pos ):
     a0 = mid_rad - left_rad
     a1 = right_rad - mid_rad

     b0 = mid_pos - left_pos
     b1 = right_pos - mid_pos

     A = math.atan2( -b0 , ( (b1/math.tan(a1)) - (b1+b0)/math.tan(a1+a0)))

     ak = pi/2-A
     alpha_k = pi/2-ak
     alpha_1 = pi/2-ak-a1
     alpha_2 = pi/2-ak-a1-a0

     k = b1 * math.tan(alpha_1) / (math.tan(alpha_k)-math.tan(alpha_1))

     d = k * math.tan(alpha_k)

     #      (            azimuth,        east, south )
     return (-(right_rad+ak-pi), right_pos+k,     d )
```

Step 12
Average the results of the two azimuth and position estimates.

```
# take two estimates of position with 3 angles

# Perform step 11 using the using both vertical edges of the far left target
# and the right edge of the far right target.
#
# See definition of field coordinate system.
# az1 and az2 are estimates of the camera heading in azimuth in radians
# east1 and east2 are estimates of the camera east position
# south1 and south2 are estimates of the camera south position
#
az1, east1, south1 = estimate_pos_3_sep_hrz_angles( left.left_rad,
                                  left.right_rad,
                                  right.right_rad,
                                  target_locs[left.pos].left_inches,
                                  target_locs[left.pos].right_inches,
                                  target_locs[right.pos].right_inches)

# Perform step 11 using the using the left vertical edge of the far left target
# and the both vertical edges of the far right target.
az2, east2, south2 = estimate_pos_3_sep_hrz_angles( left.left_rad,
                                  right.left_rad,
                                  right.right_rad,
                                  target_locs[left.pos].left_inches,
                                  target_locs[right.pos].left_inches,
                                  target_locs[right.pos].right_inches)
#
# Step 12.
# Average the two passes of passes for an estimate of the camera position and heading
#
return ( (az1+az2)/2.0, (east1+east2)/2.0, (south1+south2)/2.0 )
```

Step 14
Calculate the range to the center of the hoop.

```
#
# Step 14.
# For a target rectangle calculate the range along the floor from the center
# of the camera to the center of the hoop.
#
def target_range( target, east, south ):
     target_east = target_locs[target.pos].center_east
     target_south = -15.0

     return math.sqrt( math.pow(target_east-east,2) + math.pow(target_south-south,2) )
```

TBD need diagram here.

Step 13
Calculate the azimuth offset from the center of the backboard to the center of the hoop.

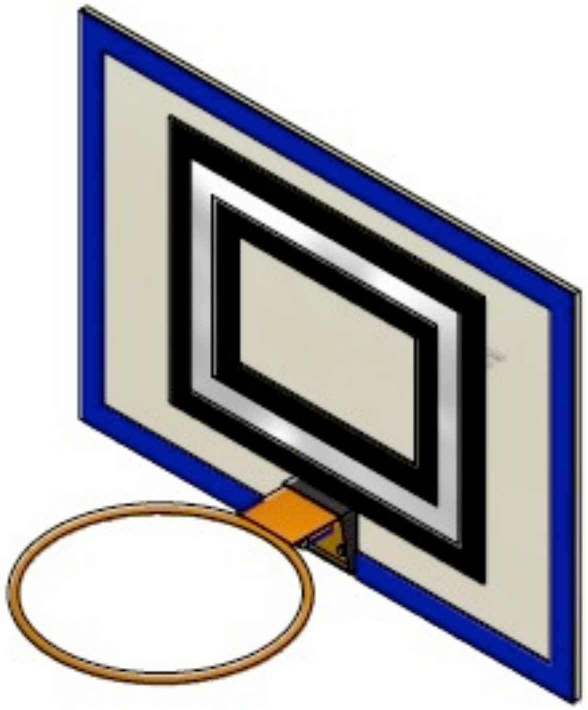
```
#
# Step 13.
# For a target rectangle calculate the azimuth offset from the center of the
# center of the backboard to the center of the hoop.
#

def target_backboard_az_and_az_offset( target, east, south ):
     #
     # Hoop Center is 15 inches south of center of the backboard.
     #
     target_east = target_locs[target.pos].center_east
     target_south = -15.0                                    # inches

     backboard_center_az = math.atan2(target_east-east,-south)
     target_center_az = math.atan2(target_east-east,target_south-south)

     az_offset = target_center_az - backboard_center_az

     return backboard_center_az, az_offset
```

TBD need diagram here.

Step 15
Operator selects commands target hoop system closes loop.

If the operator selects or pre-selects a target hoop, the system should iterate over a PID (Proportional Integral Derivative) loops to null out the azimuth error and shooter velocity error.

When the error is low enough the system should launch the shooter.



Step n+1
Backup Information

See https://github.com/MikeStitt/simple-locating/blob/master/test.py for a simulation test bench.
See https://raw.github.com/MikeStitt/simple-locating/master/output.txt for simulation results.