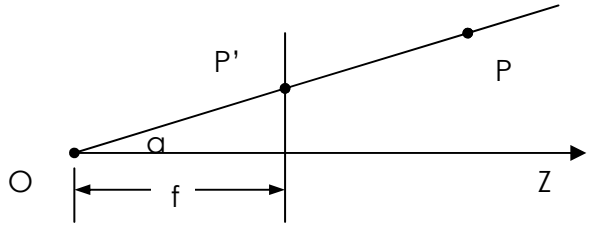
Problem 1 CCD to Camera Transformation

(a) Field of View (FOV)

i. The following diagram shows the perspective projection of a camera



If the width of the image is *L,* The Field of View

FOV = 2a = 2atan(L/2f)

ii. For the given camera,

FOV-vertical = 2atan(8/24) = 36.9°

FOV-horizontal = 2atan(6/24) = 28.1°

iii. There are only finite numbers of pixels in an image. The larger the FOV, the more scene is projected to the image. Hence the resolution for every pixel is decreased.

(b) Application

i. Let the camera-frame coordinates of a point be (X_c, Y_c, Z_c) and the image plane coordinates be (x, y), then

$$\begin{bmatrix} sx \\ sy \\ s \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix}$$

Let the pixel coordinates **w** = (*u, v*), then

$$\begin{bmatrix} su \\ sv \\ s \end{bmatrix} = \begin{bmatrix} k_u & 0 & u_0 \\ 0 & k_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} sx \\ sy \\ s \end{bmatrix}$$

Combine together:

$$\begin{bmatrix} su \\ sv \\ s \end{bmatrix} = \begin{bmatrix} k_u & 0 & u_0 \\ 0 & k_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix}$$

ii. Assume the origin of camera frame is at the center of the image, then

$(u_0, v_0) = (250, 250)$.

$(k_u, k_v) = (500/12, -500/16) = (41.67, -31.25)$

It can be computed that **w** = (*u*, *v*) = (367, 199)

Problem 2 Camera Projection

(a) Let $\mathbf{X}_c = \mathbf{a} + \lambda\mathbf{b}$, then

$$\mathbf{x} = f\left(\frac{a_x + \lambda b_x}{a_z + \lambda b_z}, \frac{a_y + \lambda b_y}{a_z + \lambda b_z}\right)$$

$$= f\left(\frac{a_x}{a_z} + \frac{\lambda}{(a_z+\lambda b_z)a_z}(b_x a_z - b_z a_x),\ \frac{a_y}{a_z} + \frac{\lambda}{(a_z+\lambda b_z)a_z}(b_y a_z - b_z a_y)\right)$$

$$= f\left(\frac{a_x}{a_z} + \Lambda(b_x a_z - b_z a_x),\ \frac{a_y}{a_z} + \Lambda(b_y a_z - b_z a_y)\right)$$

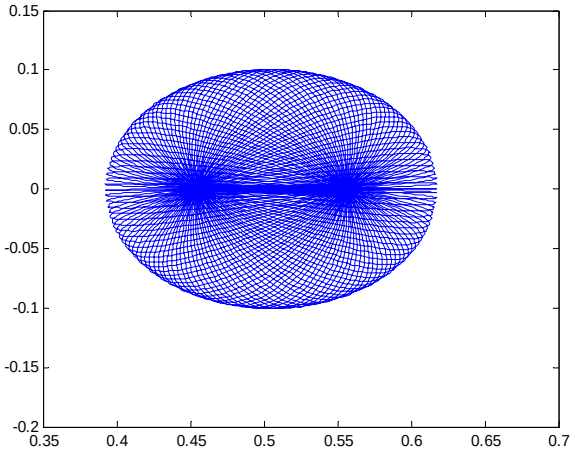
It represents a line in a 2D space.

(b) The camera frame coordinates of sphere is

$\mathbf{X}_c = (r\sin\varphi\cos\theta + x_0, r\sin\varphi\sin\theta, r\cos\varphi + z_0)$, then

$$\mathbf{x} = f\left(\frac{r\sin\varphi\cos\theta + x_0}{r\cos\varphi + z_0}, \frac{r\sin\varphi\sin\theta}{r\cos\varphi + z_0}\right)$$

Simulated in Matlab, the projection is shown as



(sphere_project.m)

It is not a circle on the image plane.

3a.

$$\text{CamMatrix} = \begin{pmatrix} -0.2905 & -0.0532 & 0.1866 & 0.6283 \\ 0.0881 & -0.3264 & 0.0881 & 0.6010 \\ -0.0002 & -0.0002 & -0.0002 & 0.0021 \end{pmatrix}$$

After defining a 2Nx12 matrix P as given in the reading and can be seen explicitly in my code, I found the desired projection matrix by finding the eigenvector corresponding to the smallest eigenvalue of P'*P, and reshaped this vector into a 3x4 matrix, as in the method described in section 3.1 of the text. Printing out the input (u,v) values and those generated by obtained matrix, both are seen to be projections of equivalent cubes in space, so the matrix appears to be valid.

I decomposed my matrix into K[R,T], but I am afraid that my K matrix is not correct. From my notes, the readings, and an outside article, I have not been able to discern what I am missing, but the product of my decomposed matrices is incorrect. Generating code is in the file LinearCalib.mat. I was, however, able to find several of the relevant parameters:

$$\begin{aligned} &\text{rho} = 3.3134\text{e+}003 \\ &\text{u0} = 300.5 \\ &\text{v0} = 300.5 \\ &\text{theta} = \tfrac{\pi}{2} \\ &\text{alpha} = 1.1177\text{e+}003 \\ &\text{beta} = 1.1177\text{e+}003 \\ &\text{R} = \begin{pmatrix} 0.7071 & 0.4150 & -0.5725 \\ 0.0000 & -0.8097 & -0.5869 \\ -0.7071 & 0.4150 & -0.5725 \end{pmatrix} \\ &\text{T} = \begin{pmatrix} 0.0000 \\ -0.0810 \\ 6.9277 \end{pmatrix} \end{aligned}$$

3b.

The points $(0,0,0), (1,1,1)$, and $(5,5,5)$ all transform to approximately $(300.5, 287.4363)$, the desired result, for they are on the same line. However, the second coordinate takes on values $287.4361, 287.4363$, and 287.4337, respectively, demonstrating a small amount of error already by the third decimal place. Other points:

$(-1, 1, 1) \rightarrow (549.7931, 143.4970)$
$(1, -1, 1) \rightarrow (300.5, 574.0207)$
$(-1, -1, 1) \rightarrow (510.8559, 408.8933)$.

All three of these corner points transformed to the exact values given (up to the decimal places inspected), implying that my matrix is accurate to a reasonable precision, considering that any radial distortion was ignored.

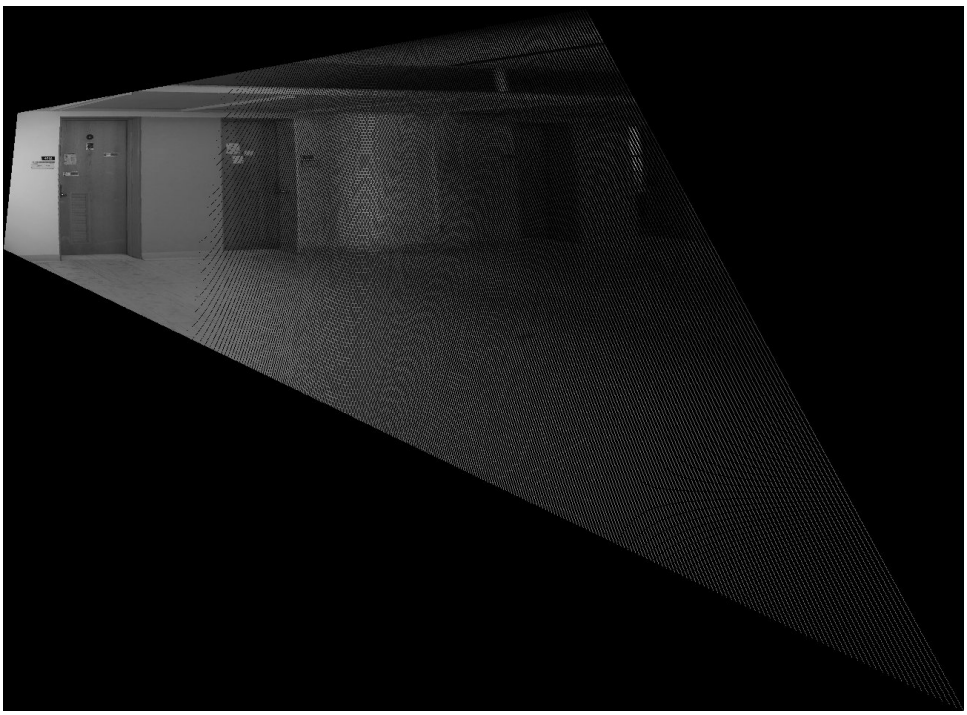
4a.
For the image hall1.pgm, the four corners of the door in image coordinates and world coordinates are:

u = [183, 361, 205, 358];
v = [109, 155, 917, 805];
x = [0, 91, 0, 91];
y = [182, 182, 0, 0];

These points define an 8x9 matrix as in the reading, whose eigenvector corresponding to the smallest eigenvalue defines the desired H matrix. Reshaped to a 3x3 matrix, we find:

$$\mathrm{H} = \begin{pmatrix} 0.0028 & -0.0003 & 0.2182 \\ 0.0009 & -0.0048 & 0.9759 \\ 0.0000 & -0.0000 & 0.0011 \end{pmatrix}$$

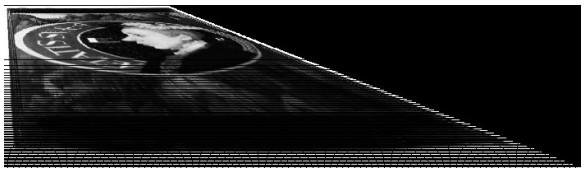
Applying the inverse of this matrix to the input image, the following is obtained, with visible aliasing as the image size is stretched.



Using the same method on edwardVI.pgm, with the point data:

u = [0,607,0,170];
v = [0,0,170,170];
x = [200, 0, 200, 0];
y = [200, 200, 0, 0];

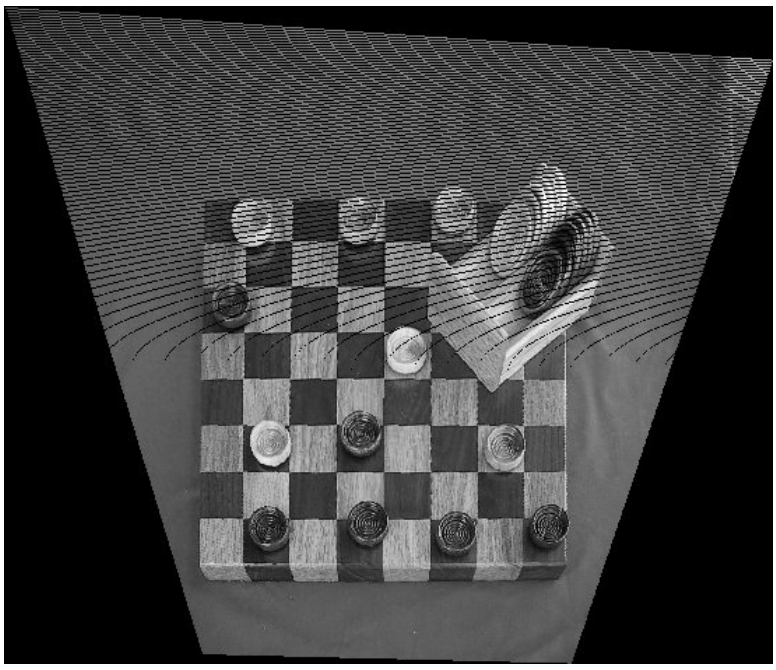
I get the following image, where the aliasing occurs as the image is stretched towards the viewer. I would have expected the circle to be more fully circular than my results present.



Applying the method to Checkerboard.pgm, with data:

u = [113, 144, 504, 471];
v = [305, 183, 295, 174];
x = [0, 0, 200, 200];
y = [0, 100, 0, 100];

Results in the image:



4b.
This method uses the same H matrix as in part A, but is premultiplied by a matrix T^{-1} and postmultiplied by a matrix U, where T rotates and translates the image data to be centered around (0,0) and have an average distance of $\sqrt{(2)}$ from the origin, and U does the same for the scene points. For the image hall1.pgm,

$$\mathrm{T} = \begin{pmatrix} 0.0038 & 0 & 0.2182 \\ 0 & 0.0038 & -1.8793 \\ 0 & 0 & 1.0000 \end{pmatrix}$$

and

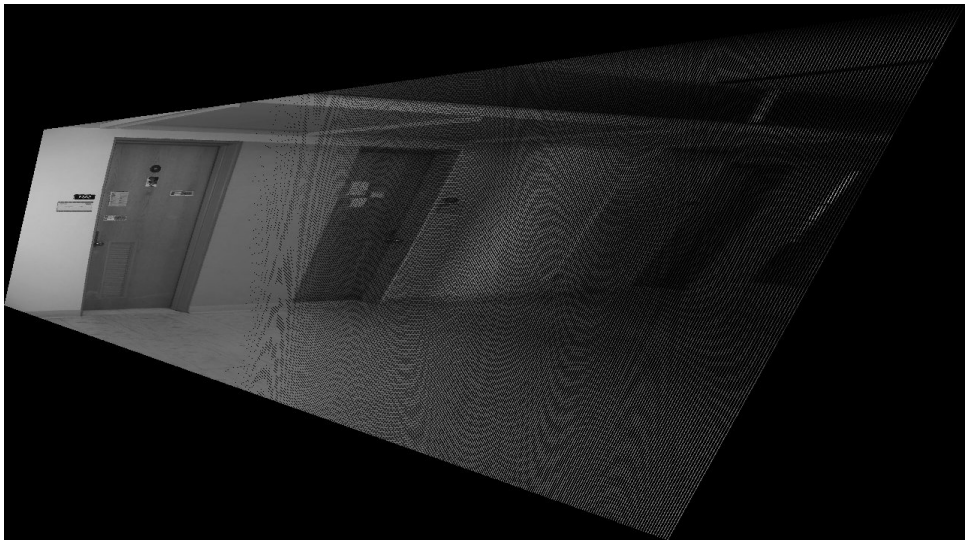
$$\mathrm{U} = \begin{pmatrix} 0.0139 & 0 & -0.6394 \\ 0 & 0.0139 & -1.2649 \\ 0 & 0 & 1.0000 \end{pmatrix}$$

The resulting images look exactly the same as in part (a) for all three cases, and are not reprinted here (all three were checked).

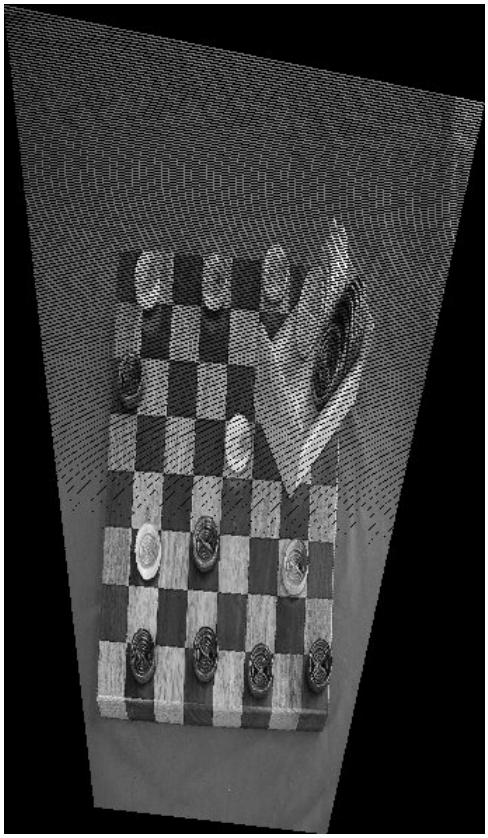
4c.
This method incorporated more errors than parts (a) and (b). For teach image, I found four points on the four corners of a rectangle on a plane in the image, and used the two sets of parallel lines this rectangle defined to find two vanishing points. The line connecting these points is the vanishing line, represented $ax + by + c = 0$, and then the given matrix formula (with fixed type-os) was used to rectify the image. I found that the value of f I used played a huge role in the accuracy of the resulting image. Large focal lengths resulted in image coordinates many times larger than the desired resulting image, but when f was taken to be in the range 0.3 to 0.5, a reasonable image was generated. The images produced by this method were not as accurate as those from parts (a) and (b), even when the pixel values were centered around the origin. However, the images do approximate the desired results. For the image hall1.pgm, the generated matrix is:

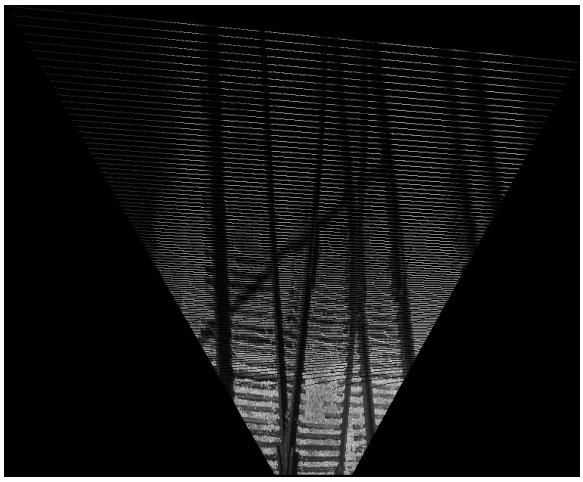
$$\begin{pmatrix} -0.3419 & 0.1040 & 0.0005 \\ 0.1040 & 0.9919 & -0.0000 \\ -0.0005 & 0.0000 & -0.3500 \end{pmatrix}$$

The resulting image for hall1.pgm is below, and we see that it is not as accurate as the previous two methods.



The same method was applied to Checkerboard.pgm and Railroad.pgm, each time chosing four points on a plane in the scene to determine the vanishing line. The Railroad image, in particular, took a very long time to generate (I actually had to cancel it before it finished the top 100 rows of pixels, as the matrix vector multiplications were somehow taking up too much memory, so the image below is missing the top 100 rows of pixels in the scene). Similar to hall1, we see that the checkerboard example is not as square as it was in parts (a) and (b). The results are below:



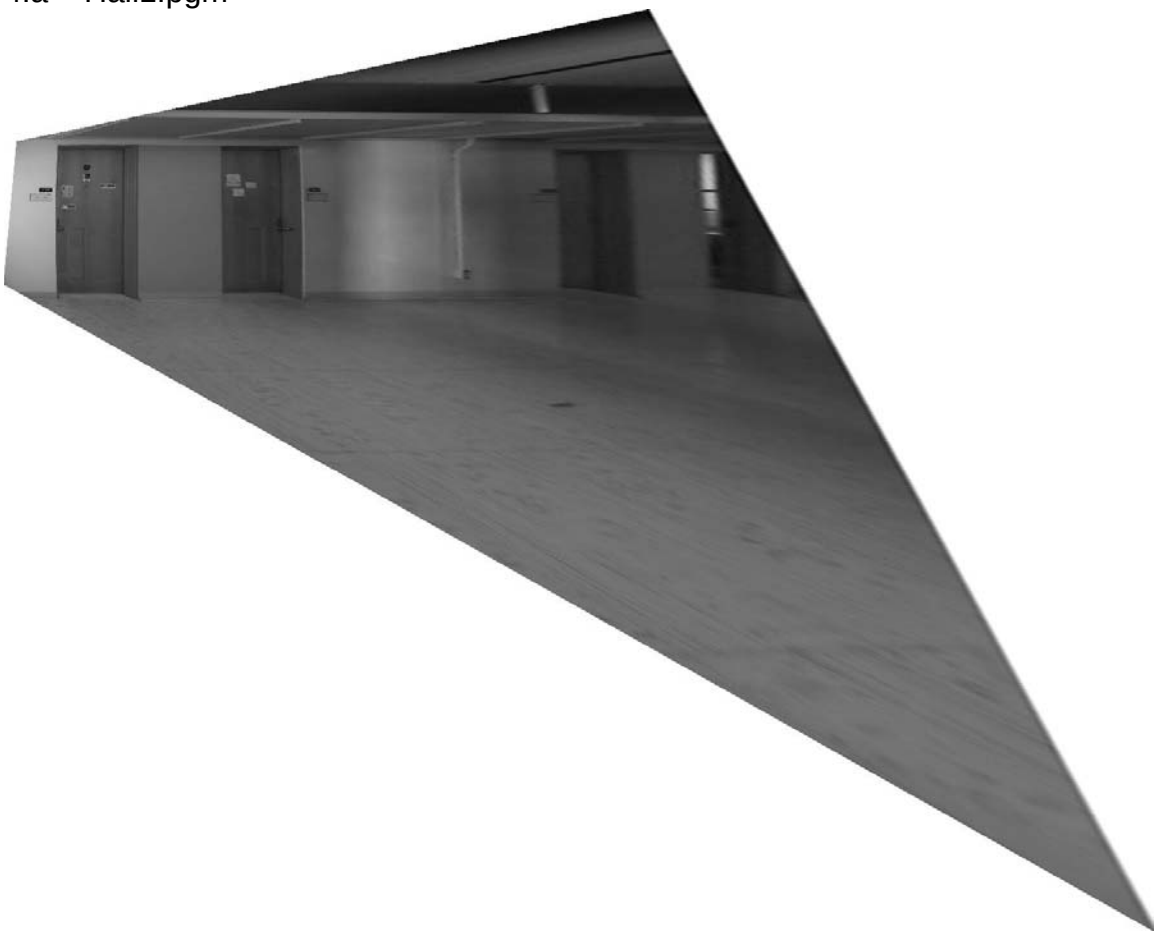


4d.
If we used an affine instead of a projective transformation to describe a plane-to-plane transformation, we would have 6 degrees of freedom. An affine transformation preserves collinearity and ratios of distances, which means it preserves parallel lines, unlike a projective transformation, and the projected midpoint of any line remains the midpoint of the projected line, again a feature not preserved by projective transformations, and affine transformations do not have lines or planes at infinity. Affine transformations can be represented as combinations of rotations, translations, dilations, and shears. A general affine transformation (for the given situation) has 8 degrees of freedom, compared to the 11 of the projective transformation, so although the projective transformation loses 3 degrees of freedom in the plane-to-plane transformation (from allowing $z = 0$ without loss of generality), the affine transformation only loses 2 degrees of freedom, as it is already limited in the transformations it can perform.

With projective transformations, it is possible for a planar square to project to a triangle in the limit as two of the parallel sides meet at a vanishing point. As affine transformations preserlve parallelism, this cannot be the case in an affine transformation. Affine transformations are appropriate models when the distance between the "front" and "back" of the scene is small, so that size changes due to perspective are small enough to be irrelevant.

Appendix – Images

4.a – Hall1.pgm



4.a – edwardVI.pgm

4.a – filecab.pgm