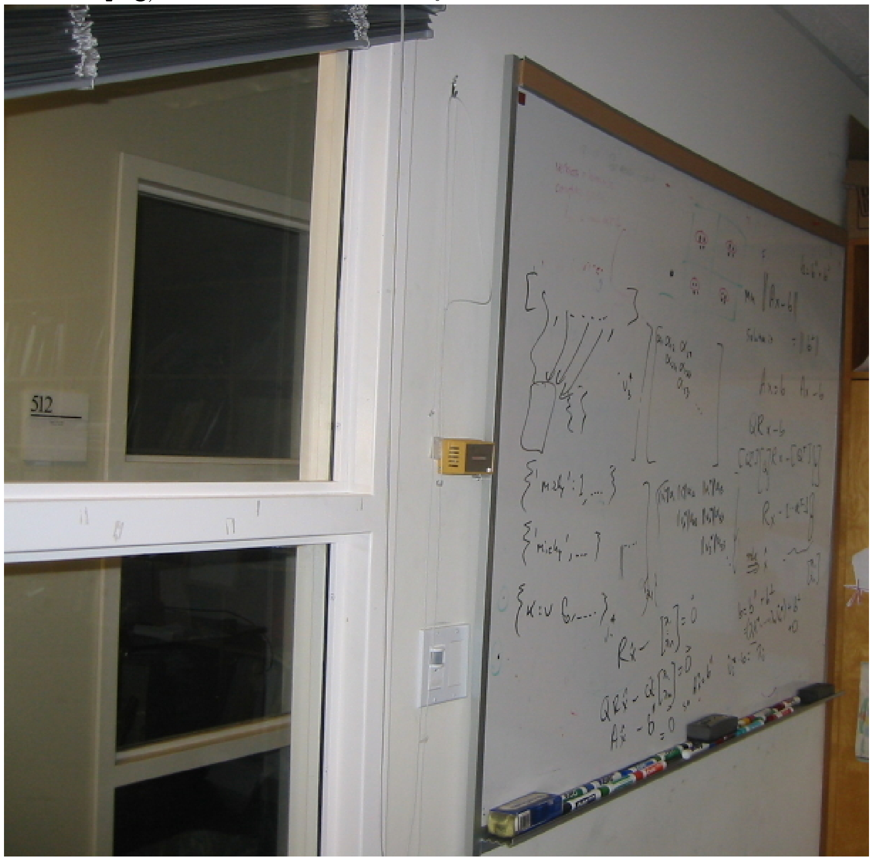
perspective lab

Coding the Matrix, 2015

`For auto-graded problems, edit the file perspective_lab.py to include your solution.`

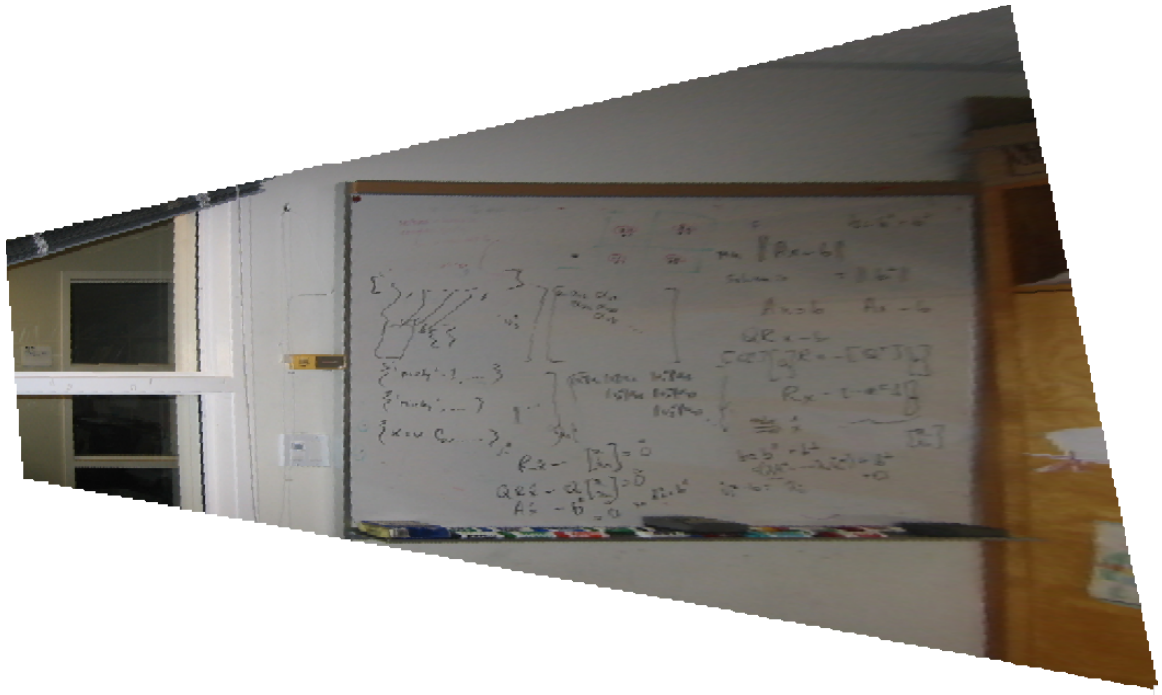
1 *Lab: Perspective rectification*

The goal for this lab is to remove perspective from an image of a flat surface. Consider the following image (stored in the file `board.png`) of the whiteboard in my office:



Looks like there's some interesting linear algebra written on the board!

We will synthesize a new image. This new image has never been captured by a camera; in fact, it would be impossible using a traditional camera since it completely lacks perspective:



The technique for carrying out this transformation makes use of the idea of a coordinate system. Actually, it requires us to consider *two* coordinate systems and to transform between a representation within one system and a representation within the other.

Think of the original image as a grid of rectangles, each assigned a color. (The rectangles correspond to the pixels.) Each such rectangle in the image corresponds to a parallelogram in the plane of the whiteboard. The perspective-free image is created by painting each such parallelogram the color of the corresponding rectangle in the original image.

Forming the perspective-free image is easy once we have a function that maps pixel coordinates to the coordinates of the corresponding point in the plane of the whiteboard. How can we derive such a function?

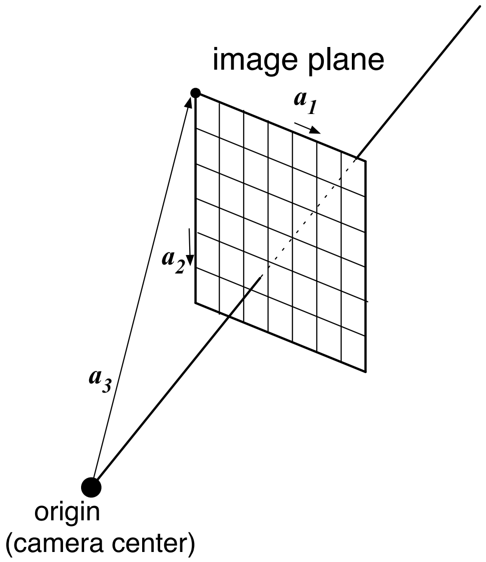
The same problem arises in using a Wiimote light pen. The light from the light pen strikes a particular sensor element of the Wiimote, and the Wiimote reports the coordinates of this sensor element to the computer. The computer needs to compute the corresponding location on the screen in order to move the mouse to that location. We therefore need a way to derive the function that maps coordinates of a sensor element to the coordinates in the computer screen.

The basic approach to derive this mapping is by example. We find several input-output pairs—points in the image plane and corresponding points in the whiteboard plane—and we derive the function that agrees with this behavior.

1 *The camera basis*

We use the camera basis $\boldsymbol{a}_1, \boldsymbol{a}_2, \boldsymbol{a}_3$ where:

- The origin is the camera center.
- The first vector $\boldsymbol{a}_1$ goes horizontally from the top-left corner of the top-left sensor element to the top-right corner.
- The second vector $\boldsymbol{a}_2$ goes vertically from the top-left corner of the top-left sensor element to the bottom-left corner.
- The third vector $\boldsymbol{a}_3$ goes from the origin (the camera center) to the top-left corner of sensor element (0,0).

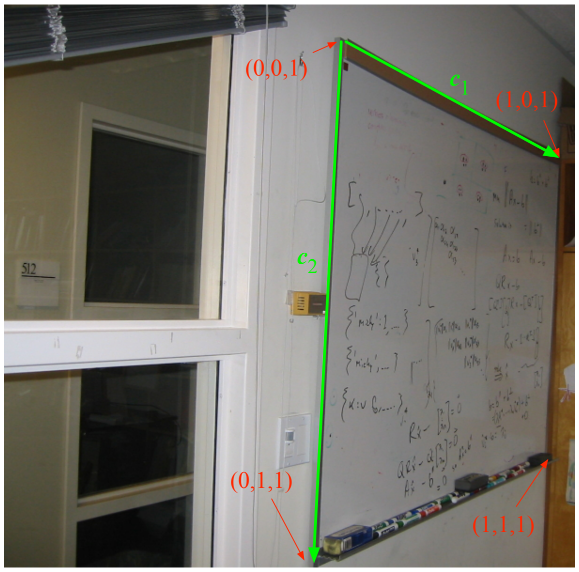


This basis has the advantage that the top-left corner of sensor element (x_1, x_2) has coordinate representation $(x_1, x_2, 1)$.

2 *The whiteboard basis*

In addition, we define a *whiteboard basis* $\boldsymbol{c}_1, \boldsymbol{c}_2, \boldsymbol{c}_3$ where:

- The origin is the camera center.
- The first vector $\boldsymbol{c}_1$ goes horizontally from the top-left corner of whiteboard to top-right corner.
- The second vector $\boldsymbol{c}_2$ goes vertically from the top-left corner of whiteboard to the bottom-left corner.
- The third vector $\boldsymbol{c}_3$ goes from the origin (the camera center) to the top-left corner of the whiteboard.



As we will see, this basis has the advantage that, given the coordinate representation (y_1, y_2, y_3) of a point $\boldsymbol{q}$, the intersection of the line through the origin and $\boldsymbol{q}$ with the whiteboard plane has coordinates $(y_1/y_3, y_2/y_3, y_3/y_3)$.

3 *Mapping from pixels to points on the whiteboard*

Our goal is to derive the function that maps the representation in camera coordinates of a point in the image plane to the representation in whiteboard coordinates of the corresponding point in the whiteboard plane.

At the heart of the function is a change of basis. We have two coordinate systems to think about, the camera coordinate system, defined by the basis $\boldsymbol{a}_1, \boldsymbol{a}_2, \boldsymbol{a}_3$, and the whiteboard coordinate system, defined by the basis $\boldsymbol{c}_1, \boldsymbol{c}_2, \boldsymbol{c}_3$. ~~This gives us two representations for a point~~. Each of these representations is useful:

1. *It is easy to go from pixel coordinates to camera coordinates:* the point with pixel coordinates (x_1, x_2) has camera coordinates $(x_1, x_2, 1)$.

2. *It is easy to go from the whiteboard coordinates of a point $\boldsymbol{q}$ in space to the whiteboard coordinates of the corresponding point $\boldsymbol{p}$ on the whiteboard:* if $\boldsymbol{q}$ has whiteboard coordinates (y_1, y_2, y_3) then $\boldsymbol{p}$ has whiteboard coordinates $(y_1/y_3, y_2/y_3, y_3/y_3)$.

In order to construct the function that maps from pixel coordinates to whiteboard coordinates, we need to add a step in the middle: mapping from camera coordinates of a point $\boldsymbol{q}$ to whiteboard coordinates of the same point.

To help us keep track of whether a vector is the coordinate representation in terms of camera coordinates or is the coordinate representation in terms of whiteboard coordinates, we will use different domains for these two kinds of vectors. A coordinate representation in terms of camera coordinates will have domain C=`{'x1','x2','x3'}`. A coordinate representation in terms of whiteboard coordinates will have domain R=`{'y1','y2','y3'}`.

Our aim is to derive the function $f : \mathbb{R}^C \longrightarrow \mathbb{R}^R$ with the following spec:

- *input:* the coordinate representation $\boldsymbol{x}$ in terms of camera coordinates of a point $\boldsymbol{q}$
- *output:* the coordinate representation $\boldsymbol{y}$ in terms of whiteboard coordinates of the point $\boldsymbol{p}$ such that the line through the origin and $\boldsymbol{q}$ intersects the whiteboard plane at $\boldsymbol{p}$.

There is a little problem here; if $\boldsymbol{q}$ lies in the plane through the origin that is parallel to the whiteboard plane then the line through the origin and $\boldsymbol{q}$ does not intersect the whiteboard plane. We'll disregard this issue for now.

We will write f as the composition of two functions $f = g \circ h$, where

- $h : \mathbb{R}^C \longrightarrow \mathbb{R}^R$ is defined thus:
  - *input:* a point's coordinate representation with respect to the camera basis
  - *output:* the same point's coordinate representation with respect to the whiteboard basis
- $g : \mathbb{R}^R \longrightarrow \mathbb{R}^R$ is defined thus:
  - *input:* the coordinate representation in terms of whiteboard coordinates of a point $\boldsymbol{q}$
  - *output:* the coordinate representation in terms of whiteboard coordinates of the point $\boldsymbol{p}$ such that the line through the origin and $\boldsymbol{q}$ intersects the whiteboard plane at $\boldsymbol{p}$.

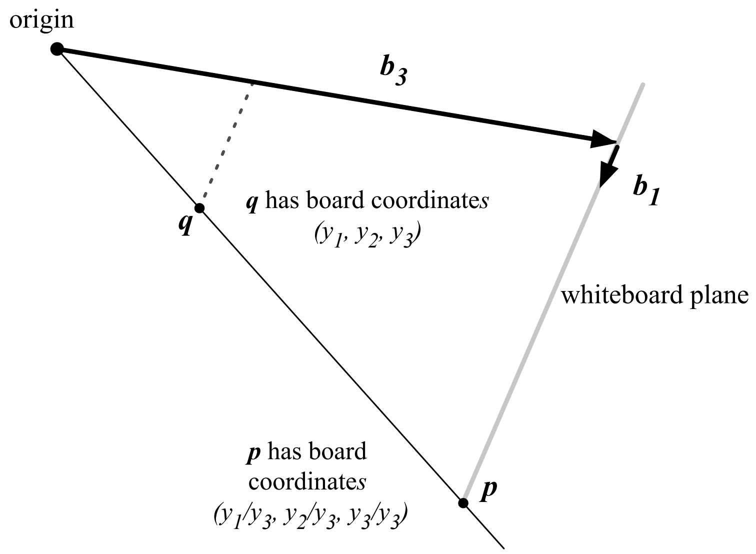
4 *Mapping a point not on the whiteboard to the corresponding point on the whiteboard*

In this section, we develop a procedure for the function g.

We designed the whiteboard coordinate system in such a way that a point on the whiteboard has coordinate y_3 equal to 1. For a point that is closer to the camera, the y_3 coordinate is less than 1.

Suppose $\boldsymbol{q}$ is a point that is not on the whiteboard, e.g. a point closer to the camera. Consider the line through the origin and $\boldsymbol{q}$. It intersects the whiteboard plane at some point $\boldsymbol{p}$. How do we compute the point $\boldsymbol{p}$ from the point $\boldsymbol{q}$?

This figure shows a top-view of the situation.


origin
b_3
b_1
q
q has board coordinates (y_1, y_2, y_3)
whiteboard plane
p has board coordinates $(y_1/y_3, y_2/y_3, y_3/y_3)$
p

In this view, we see the top edge of the whiteboard, a point $\boldsymbol{q}$ not on the whiteboard, and the point $\boldsymbol{p}$ on the whiteboard that corresponds to $\boldsymbol{q}$ (in the sense that the line between the origin and $\boldsymbol{q}$ intersects the whiteboard plane at $\boldsymbol{p}$).

Let the whiteboard-coordinate representation of $\boldsymbol{q}$ be (y_1, y_2, y_3). In this figure, y_3 is less than 1. Elementary geometric reasoning (similar triangles) shows that the whiteboard-coordinate representation of the point $\boldsymbol{p}$ is $(y_1/y_3, y_2/y_3, y_3/y_3)$. Note that the third coordinate is 1, as required of a point in the whiteboard plane.

Task 1: Write a procedure `move2board(y)` with the following spec:

- *input:* a {'y1','y2','y3'}-vector $\boldsymbol{y}$, the coordinate representation in whiteboard coordinates of a point $\boldsymbol{q}$
(Assume $\boldsymbol{q}$ is not in the plane through the origin that is parallel to the whiteboard plane, i.e. that the y3 entry is nonzero.)
- *output:* a {'y1','y2','y3'}-vector $\boldsymbol{z}$, the coordinate representation in whiteboard coordinates of the point $\boldsymbol{p}$ such that the line through the origin and $\boldsymbol{q}$ intersects the whiteboard plane at $\boldsymbol{p}$.

5 *The change-of-basis matrix*

You have developed a procedure for g. Now we begin to address the procedure for h.

Writing a point $\boldsymbol{q}$ in terms of both the camera coordinate system $\boldsymbol{a}_1, \boldsymbol{a}_2, \boldsymbol{a}_3$ and the whiteboard coordinate system $\boldsymbol{c}_1, \boldsymbol{c}_2, \boldsymbol{c}_3$, and using the linear-combinations definition of matrix-vector multiplication, we have

$$\left[\begin{array}{c} \\ \boldsymbol{q} \\ \\ \end{array}\right] = \left[\begin{array}{c|c|c} & & \\ \boldsymbol{a}_1 & \boldsymbol{a}_2 & \boldsymbol{a}_3 \\ & & \end{array}\right] \left[\begin{array}{c} x_1 \\ x_2 \\ x_3 \end{array}\right] = \left[\begin{array}{c|c|c} & & \\ \boldsymbol{c}_1 & \boldsymbol{c}_2 & \boldsymbol{c}_3 \\ & & \end{array}\right] \left[\begin{array}{c} y_1 \\ y_2 \\ y_3 \end{array}\right]$$

Let $A = \left[\begin{array}{c|c|c} & & \\ \boldsymbol{a}_1 & \boldsymbol{a}_2 & \boldsymbol{a}_3 \\ & & \end{array}\right]$ and let $C = \left[\begin{array}{c|c|c} & & \\ \boldsymbol{c}_1 & \boldsymbol{c}_2 & \boldsymbol{c}_3 \\ & & \end{array}\right]$. Since the function from $\mathbb{R}^3$ to $\mathbb{R}^3$ defined by $\boldsymbol{y} \mapsto C\boldsymbol{y}$ is an invertible function, the matrix C has an inverse C^{-1}. Let $H = C^{-1}A$. Then a little algebra shows

$$\begin{bmatrix} & & \\ & H & \\ & & \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}$$

This is just a recapitulation of the argument that change of basis is matrix-multiplication.

6 *Computing the change-of-basis matrix*

Now that we know a change-of-basis matrix H exists, we don't use the camera basis or the whiteboard basis to compute it because we don't know those bases! Instead, we will compute H by observing how it behaves on known points, setting up a linear system based on these observations, and solving the linear system to find the entries of H.

Write $H = \begin{bmatrix} h_{y_1,x_1} & h_{y_1,x_2} & h_{y_1,x_3} \\ h_{y_2,x_1} & h_{y_2,x_2} & h_{y_2,x_3} \\ h_{y_3,x_1} & h_{y_3,x_2} & h_{y_3,x_3} \end{bmatrix}$.

Let $\boldsymbol{q}$ be a point on the image plane. If $\boldsymbol{q}$ is the top-left corner of pixel x_1, x_2 then its camera coordinates are $(x_1, x_2, 1)$, and

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} h_{y_1,x_1} & h_{y_1,x_2} & h_{y_1,x_3} \\ h_{y_2,x_1} & h_{y_2,x_2} & h_{y_2,x_3} \\ h_{y_3,x_1} & h_{y_3,x_2} & h_{y_3,x_3} \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ 1 \end{bmatrix}$$

where (y_1, y_2, y_3) are the whiteboard coordinates of $\boldsymbol{q}$.

Multiplying out, we obtain

$$y_1 = h_{y_1,x_1}x_1 + h_{y_1,x_2}x_2 + h_{y_1,x_3} \tag{1}$$
$$y_2 = h_{y_2,x_1}x_1 + h_{y_2,x_2}x_2 + h_{y_2,x_3} \tag{2}$$
$$y_3 = h_{y_3,x_1}x_1 + h_{y_3,x_2}x_2 + h_{y_3,x_3} \tag{3}$$

If we had a point with known camera coordinates and known whiteboard coordinates, we could plug in these coordinates to get three linear equations in the unknowns, the entries of H. By using three such points, we would get nine linear equations and could solve for the entries of H.

For example, by inspecting the image of the board, you can find the pixel coordinates of the bottom-left corner of the whiteboard. You can do this using an image viewer such as The GIMP by opening the image and pointing with your cursor at the corner and reading off the pixel coordinates.[1] I get the coordinates $x_1 = 329$, $x_2 = 597$. Therefore the sensor element that detected the light from this corner is located at $(x_1, x_2, x_3) = (329, 597, 1)$ in camera coordinates.

Plugging in these values for x_1, x_2, we get

$$y_1 = h_{y_1,x_1}329 + h_{y_1,x_2}597 + h_{y_1,x_3} \tag{4}$$
$$y_2 = h_{y_2,x_1}329 + h_{y_2,x_2}597 + h_{y_2,x_3} \tag{5}$$
$$y_3 = h_{y_3,x_1}329 + h_{y_3,x_2}597 + h_{y_3,x_3} \tag{6}$$

Pretend we knew that the whiteboard coordinates for the same point were $(0.2, 0.1, 0.3)$. Then we could plug these values into the equations and obtain three equations in the unknown values of the entries of H. By considering other points in the image, we could get still more equations, and eventually get enough equations to allow us to solve for the entries of H. This approach consists in learning the function h from input-output pairs $(\boldsymbol{x}, \boldsymbol{y})$ such that $h(\boldsymbol{x}) = \boldsymbol{y}$.

The bad news is that we don't know the whiteboard coordinates for these points. The good news is that we can use a similar strategy, learning the function f from input-output pairs such that $f(\boldsymbol{x}) = \boldsymbol{y}$. For example, if $\boldsymbol{x}$ is $(329, 597, 1)$ then $\boldsymbol{y} = f(\boldsymbol{x})$ is $(0, 1, 1)$.

[1] There are simpler programs that can be used for the same purpose. Under Mac OS, you don't need to install anything: the tool for making a screenshot of a rectangular region can be used.

How can we use the knowledge of input-output pairs for f to calculate the entries of H? We need to do a bit of algebra. Let (y_1, y_2, y_3) be the whiteboard coordinates of the point $\boldsymbol{q}$ whose camera coordinates are $(329, 597, 1)$. We don't know the values of y_1, y_2, y_3 but we do know (from the discussion in Section 4) that

$$\begin{aligned} 0 &= y_1/y_3 \\ 1 &= y_2/y_3 \end{aligned}$$

so

$$\begin{aligned} 0y_3 &= y_1 \\ 1y_3 &= y_2 \end{aligned}$$

The first equation tells us that $y_1 = 0$. Combining this with Equation 4 gives us a linear equation:

$$h_{y_1,x_1}329 + h_{y_1,x_2}597 + h_{y_1,x_3} = 0$$

The second equation tells us that $y_3 = y_2$. Therefore, combining Equations 5 and 6 gives us

$$h_{y_3,x_1}329 + h_{y_3,x_2}597 + h_{y_3,x_3} = h_{y_2,x_1}329 + h_{y_2,x_2}597 + h_{y_2,x_3}$$

Thus we have obtained two linear equations in the unknown entries of H.

Recall that a linear equation can be expressed as an equation stating the value of the dot-product of a coefficient vector—a vector whose entries are the coefficients—and a vector of unknowns. In this case, the unknowns are the entries of H. We therefore define a vector $\boldsymbol{h}$ to consist of these unknowns. What should the domain of $\boldsymbol{h}$ be? We define its domain to be the Cartesian product $R \times C$ where R and C are the sets defined in Section 3, namely R={'y1','y2','y3'} and C={'x1','x2','x3'}.

Task 2: Assign to the variable D the Cartesian product $R \times C$.

It might seem strange to you that the entries of a matrix can be rearranged to form the entries of a vector, and vice versa.

The equations we have obtained so far can be written as

$$\begin{aligned} \boldsymbol{u} \cdot \boldsymbol{h} &= 0 \\ \boldsymbol{v} \cdot \boldsymbol{h} &= 0 \end{aligned}$$

where $\boldsymbol{u}$ is the vector

```
  Vec(D, {('y1','x1'): 329, ('y1','x2'):597, ('y1','x3'):1})
```

and $\boldsymbol{v}$ is the vector

```
  Vec(D, {('y3','x1'): 329, ('y3','x2'):597, ('y3','x3'):1,
('y2','x1'):-329, ('y2','x2'):-597, ('y2','x3'):-1})
```

However, two equations will not suffice. By considering the other three corners of the whiteboard, we can get six more equations. In general, suppose we know numbers x_1, x_2, w_1, w_2 such that

$$f([x_1, x_2, 1]) = [w_1, w_2, 1]$$

Let $[y_1, y_2, y_3]$ be the whiteboard coordinates of the point whose camera coordinates are $[x_1, x_2, 1]$. The whiteboard-coordinate representation of the original point $\boldsymbol{p}$ is $(y_1/y_3, y_2/y_3, 1)$. This shows

$$\begin{aligned} w_1 &= y_1/y_3 \\ w_2 &= y_2/y_3 \end{aligned}$$

Multiplying through by y_3, we obtain

$$w_1 y_3 = y_1$$
$$w_2 y_3 = y_2$$

Combining these equations with Equations 1, 2, and 3, we obtain

$$w_1(h_{y_3,x_1}x_1 + h_{y_3,x_2}x_2 + h_{y_3,x_3}) = h_{y_1,x_1}x_1 + h_{y_1,x_2}x_2 + h_{y_1,x_3}$$
$$w_2(h_{y_3,x_1}x_1 + h_{y_3,x_2}x_2 + h_{y_3,x_3}) = h_{y_2,x_1}x_1 + h_{y_2,x_2}x_2 + h_{y_2,x_3}$$

Multiplying through and moving everything to the same side, we obtain

$$(w_1x_1)h_{y_3,x_1} + (w_1x_2)h_{y_3,x_2} + w_1h_{y_3,x_3} - x_1h_{y_1,x_1} - x_2h_{y_1,x_2} - 1h_{y_1,x_3} = 0 \tag{7}$$
$$(w_2x_1)h_{y_3,x_1} + (w_2x_2)h_{y_3,x_2} + w_2h_{y_3,x_3} - x_1h_{y_2,x_1} - x_2h_{y_2,x_2} - 1h_{y_2,x_3} = 0 \tag{8}$$

Because we started with numbers for x_1, x_2, w_1, w_2, we obtain two linear equations with known coefficients.

Task 3: Write a procedure `make_equations(x1, x2, w1, w2)` that outputs a list $[\boldsymbol{u}, \boldsymbol{v}]$ consisting of two D-vectors $\boldsymbol{u}$ and $\boldsymbol{v}$ such that Equations 7 and 8 are expressed as

$$\boldsymbol{u} \cdot \boldsymbol{h} = 0$$
$$\boldsymbol{v} \cdot \boldsymbol{h} = 0$$

where $\boldsymbol{h}$ is the D-vector of unknown entries of H.

By using the four corners of the whiteboard, we obtain eight equations. However, no matter how many points we use, we cannot hope to exactly pin down H using only input-output pairs of f.

Here is the reason. Suppose $\hat{H}$ were a matrix that satisfied all such equations: for any input vector $\boldsymbol{x} = [x_1, x_2, 1]$, $g(\hat{H}\boldsymbol{x}) = f(\boldsymbol{x})$. For any scalar α, an algebraic property of matrix-vector multiplication is

$$(\alpha\hat{H})\boldsymbol{x} = \alpha(\hat{H}\boldsymbol{x})$$

Let $[y_1, y_2, y_3] = H\boldsymbol{x}$. Then $\alpha(\hat{H}\boldsymbol{x}) = [\alpha y_1, \alpha y_2, \alpha y_3]$. But since g divides the first and second entries by the third, multiplying all three entries by α does not change the output of g:

$$g(\alpha\hat{H}\boldsymbol{x}) = g([\alpha y_1, \alpha y_2, \alpha y_3]) = g([y_1, y_2, y_3])$$

This shows that if $\hat{H}$ is a suitable matrix for H then so is $\alpha\hat{H}$.

This mathematical result corresponds to the fact that we cannot recover the scale of the whiteboard from the image. It could be a huge whiteboard that is very far away, or a tiny whiteboard that is very close. Fortunately, the math also shows it doesn't matter to the function f.

In order to pin down *some* matrix H, we impose a scaling equation. We simply require that some entry, say the `('y1', 'x1')` entry, be equal to 1. We will write this as $\boldsymbol{w} \cdot \boldsymbol{h} = 1$

Task 4: Write the D-vector $\boldsymbol{w}$ with a 1 in the `('y1', 'x1')` entry.

Now we have a linear system consisting of nine equations. To solve it, we construct a $\{0, 1, \ldots, 8\} \times D$ matrix L whose rows are the coefficient vectors and we construct a $\{0, 1, \ldots, 8\}$-vector $\boldsymbol{b}$ whose entries are all zero except for a 1 in the position corresponding to the scaling equation.

Task 5: Assign to $\boldsymbol{b}$ the $\{0, 1, \ldots, 8\}$-vector $\boldsymbol{b}$ whose entries are all zero except for a 1 in position 8.

Task 6: Write a procedure `make_nine_equations(corners)`. The input is a four-element list `corners` consisting of

- the (x_1, x_2) pair of coordinates for the top-left corner of the whiteboard,
- the (x_1, x_2) pair of coordinates for the bottom-left corner,
- the (x_1, x_2) pair of coordinates for the top-right corner, and
- the (x_1, x_2) pair of coordinates for the bottom-right corner.

The output is the eight-element list `veclist` consisting of the following vectors, in order.

- the vector $\boldsymbol{u}$ and the vector $\boldsymbol{v}$ from `make_equations(x1, x2, w1, w2)` applied to the top-left corner,
- the vector $\boldsymbol{u}$ and the vector $\boldsymbol{v}$ from `make_equations(x1, x2, w1, w2)` applied to the bottom-left corner,
- the vector $\boldsymbol{u}$ and the vector $\boldsymbol{v}$ from `make_equations(x1, x2, w1, w2)` applied to the top-right corner,
- the vector $\boldsymbol{u}$ and the vector $\boldsymbol{v}$ from `make_equations(x1, x2, w1, w2)` applied to the bottom-right corner,
- the vector $\boldsymbol{w}$ from Task 4.

Task 7: Here are the pixel coordinates for the corners of the whiteboard in the image.

top left	$x_1 = 358, x_2 = 36$
bottom left	$x_1 = 329, x_2 = 597$
top right	$x_1 = 592, x_2 = 157$
bottom right	$x_1 = 580, x_2 = 483$

Assign to `veclist` the result of applying `make_nine_equations(corners)` to the above data.

Assign to L the matrix whose rows are the vectors comprising `veclist`. (Use the procedure `rowdict2mat` from the module `matutil`.)

Task 8: Assign to `hvec` the solution to the equation $L * \boldsymbol{h} = \boldsymbol{b}$. Verify for yourself that it is indeed a solution to this equation.

Assign to H the matrix whose entries are given by the vector `hvec`.

You can check that the matrix H is correct. The pixel corresponding to the top-right corner of the whiteboard has coordinates (592,157), which corresponds to the point in the camera plane whose coordinate representation in terms of the camera basis is

```
Vec({'x1','x2','x3'}, {'x1':592, 'x2':157, 'x3':1})
```

Multiply that vector by the matrix H. The result should be a vector something like

```
Vec({'y1', 'y3', 'y2'},{'y1': 240.2549, 'y2': 2.842170943040401e-14, 'y3': 240.25490196078434})
```

which is the coordinate representation of the same point in terms of the whiteboard basis. To find the corresponding point in the plane of the whiteboard, divide each of the coordinates by the `'y3'` coordinate, obtaining

```
Vec({'y1', 'y3', 'y2'},{'y1': 1.0, 'y2': 1.1829814583780332e-16, 'y3': 1.0})
```

This is the point whose whiteboard coordinates are (1,0,1), which is indeed the top-right corner of the whiteboard.

7 *Image representation*

Recall the image representation used in the 2D geometry lab. A *generalized image* consists of a grid of generalized pixels, where each generalized pixel is a quadrilateral (not necessarily a rectangle).

The points at the corners of the generalized pixels are identified by pairs (x, y) of integers, the pixel coordinates.

Each corner is assigned a location in the plane, and each generalized pixel is assigned a color. The mapping of corners to points in the plane is given by a matrix, the *location matrix*. Each corner corresponds to a column of the location matrix, and the label of that column is the pair (x, y) of pixel coordinates of the corner. The column is a `{'x','y','u'}`-vector giving the location of the corner. Thus the row labels of the location matrix are `'x'`, `'y'`, and `'u'`.

The mapping of generalized pixels to colors is given by another matrix, the *color matrix*. Each generalized pixel corresponds to a column of the color matrix, and the label of that column is the pair of pixel coordinates of the top-left corner of that generalized pixel. The column is a `{'r','g','b'}`-vector giving the color of that generalized pixel.

The module `image_mat_util` (from the Geometry Lab) defines the procedures

- `file2mat(filename, rowlabels)`, which, given a path to a `.png` image file and optionally a tuple of row labels, returns the pair `(points, colors)` of matrices representing the image, and
- `mat2display(pts, colors, row_labels)`, which displays an image given by a matrix `pts` and a matrix `colors` and optionally a tuple of row labels. There are a few additional optional parameters that we will use in this lab.

As in the 2D geometry lab, you will apply a transformation to the locations to obtain new locations, and view the resulting image.

8 *Synthesizing the perspective-free image*

Now we present the tasks involved in using `H` to create the synthetic image.

~~**Ungraded Task:**~~ Construct the generalized image from the image file `board.png`:

```
(X_pts, colors) = image_mat_util.file2mat('board.png', ('x1','x2','x3'))
```

~~**Ungraded Task**~~**:** The columns of the matrix `X_pts` are the camera-coordinates representations of points in the image. We want to obtain the board-coordinates representations of these points. Multiply the matrix `H` by each column of `X_pts`, obtaining a matrix `Y_pts` whose columns are the results. Hint: Use the matrix-vector definition of matrix-matrix multiplication.

Task 9: Each column of `Y_pts` gives the whiteboard-coordinate representation (y_1, y_2, y_3) of a point $\boldsymbol{q}$ in the image. We need to construct another matrix `Y_board` in which each column gives the whiteboard-coordinate representation $(y_1/y_3, y_2/y_3, 1)$ of the corresponding point $\boldsymbol{p}$ in the plane containing the whiteboard.

Write a procedure `mat_move2board(Y)` with the following spec:

- *input:* a Mat each column of which is a 'y1','y2','y3'-vector giving the whiteboard coordinates of a point $\boldsymbol{q}$
- *output:* a Mat each column of which is the corresponding point in the whiteboard plane (the point of intersection with the whiteboard plane of the line through the origin and $\boldsymbol{q}$

Here's a small example:

```
>>> Y_in = Mat(({'y1', 'y2', 'y3'}, {0,1,2,3}),
     {('y1',0):2, ('y2',0):4, ('y3',0):8,
      ('y1',1):10, ('y2',1):5, ('y3',1):5,
      ('y1',2):4, ('y2',2):25, ('y3',2):2,
      ('y1',3):5, ('y2',3):10, ('y3',3):4})
>>> print(Y_in)

        0  1  2  3
      ------------
 y1  |  2 10  4  5
 y2  |  4  5 25 10
 y3  |  8  5  2  4

>>> print(mat_move2board(Y_in))

           0 1    2    3
      ------------------
 y1  |  0.25 2    2 1.25
 y2  |   0.5 1 12.5  2.5
 y3  |     1 1    1    1
```

Once your `mat_move2board` procedure is working, use it to derive the matrix `Y_board` from `Y_pts`:

```
>>> Y_board = mat_move2board(Y_pts)
```

One simple way to implement `mat_move2board(Y)` is to convert the Mat to a column dictionary (coldict), call your `move2board(y)` procedure for each column, and convert the resulting column dictionary back to a matrix.

Ungraded Task: Finally, display the result of

```
>>> image_mat_util.mat2display(Y_board, colors, ('y1', 'y2', 'y3'),
scale=100, xmin=None, ymin=None)
```

Ungraded Task: If you have time, try the process with some other images. There are pictures of Brown University's Computer Science building (`cit.png`), a plaque at Brown University describing a Newtonian tree (`Newton_tree.png`), and a building in Heidelberg where Kirchhoff and Bunsen did research (`House_of_the_Giant.png`). For each image, you can select a rectangle and define a coordinate system that assigns coordinates $(0,0,1),(1,0,1),(0,1,1),(1,1,1)$ to the corners of the rectangle. Then find out the pixel coordinates corresponding to these points, and so on. Note: Since the transformation has some numerical problems, you might have to set limits when calling `image_mat_util.mat2display`. Try setting `xmin = -2, xmax=2, ymin = -2, ymax=2`.