Vanishing Points

The image on the right shows a bridge receding into the distance. The planks of the bridge have a fixed width and the handrails have a fixed height, but they both appear to shrink with distance. If this bridge had infinite length, these lines would converge to a single point - the vanishing point. This goal of this assignment is to estimate the vanishing point from user-specified lines.

Methods

In the simplest case, a vanishing point is computed from the intersection of two lines.

A vanishing point may be more accurately computed from three or more lines. While two non-parallel lines must intersect, three or more non-parallel lines may not have a single intersection (as shown below). In such situations we seek the point that is closest to all lines. We will use the perpendicular distance between the point and each line, as opposed to the vertical or horizontal distance, so that the estimation is not dependent on the orientation of the lines.
The perpendicular distance between a point \( \vec{v} \) and a line \( l \) defined by the points \( \vec{p} \) and \( \vec{q} \) is:

\[
d = \| \vec{n}^T (\vec{v} - \vec{p}) \|, \tag{1}
\]

where \( \| \cdot \| \) denotes vector norm and \( \vec{n} \) is perpendicular to the line:

\[
\vec{n} = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \frac{(\vec{q} - \vec{p})/\|\vec{q} - \vec{p}\|}{\|\vec{q} - \vec{p}\|}. \tag{2}
\]

The \( 2 \times 2 \) matrix on the left rotates by 90 degrees a unit vector along the line \( l \).

The point \( \vec{v} \) that minimizes the average distance to \( m \) lines is determined by defining a quadratic error function to be minimized:

\[
E(\vec{v}) = \sum_{i=1}^{m} \| \vec{n}_i^T (\vec{v} - \vec{p}_i) \|^2, \tag{3}
\]

where the point \( \vec{p}_i \) and vector \( \vec{n}_i \) parameterize the \( i^{th} \) line. The point \( \vec{v} \) that minimizes the average distance to all \( m \) lines is determined by differentiating this error function with respect to \( \vec{v} \):

\[
\frac{dE}{d\vec{v}} = \sum_{i=1}^{m} 2\vec{n}_i^T (\vec{n}_i^T (\vec{v} - \vec{p}_i))
\]

\[
= \sum_{i=1}^{m} 2\vec{n}_i^T (\vec{n}_i^T \vec{v} - \vec{n}_i^T \vec{p}_i), \tag{4}
\]

setting the result equal to zero and solving for \( \vec{v} \):

\[
\sum_{i=1}^{m} 2\vec{n}_i^T (\vec{n}_i^T \vec{v} - \vec{n}_i^T \vec{p}_i) = 0
\]

\[
\sum_{i=1}^{m} 2\vec{n}_i^T \vec{v} - \sum_{i=1}^{m} 2\vec{n}_i^T \vec{p}_i = 0
\]

\[
\sum_{i=1}^{m} \vec{n}_i^T \vec{v} = \sum_{i=1}^{m} \vec{n}_i^T \vec{p}_i
\]

\[
\vec{v} = \left( \sum_{i=1}^{m} \vec{n}_i^T \vec{n}_i \right)^{-1} \sum_{i=1}^{m} \vec{n}_i \vec{p}_i. \tag{5}
\]
The $2 \times 2$ matrix on the right-hand side of the above solution will be invertible unless all $m$ lines are parallel.

**Results**

Show in Figure 1 are six successive screen captures of a user specifying lines in an image. Shown in Figure 2 is the estimated vanishing point (magenta circle). Per our optimization, the vanishing point minimizes the perpendicular distance to each line and therefore lies in the middle of the non-intersecting user specified lines.

**Source Code**

The main script is vp.m. This script allows a user to specify $n \geq 2$ of lines in the image from two points on the line. The x- and y-coordinates of these two points is stored in a $n \times 4$ data structure dat. The vanishing point is computed using Equation (5) and drawn atop the image and specified lines.

There are three auxiliary functions imshow.m, drawmarker.m, and drawline.m that, respectively, neatly displays an image adjusting the axis and aspect ratio, draws a marker of specified location, symbol, color, and size on the current plot, and draws a line specified by two points.

```matlab
%%
%% vp.m
%%
clear;
clf;

%% Load and display image
set( gcf, 'Renderer', 'zbuffer' );
im = imread('bridge.png');
imshow( uint8(im) );
hold on;
[ydim,xdim,zdim] = size(im);

%% UI for specifying lines
hold on;
```

I am using the matlab latex package mcode that may be found at www.mathworks.com/matlabcentral/fileexchange/8015-m-code-latex-package
Figure 1: A user specifies six lines in an image.
Figure 2: The estimated vanishing point (magenta circle) from six user specified lines (green). Shown in the lower panel is a magnified view of the region around the vanishing point.
c = 1;
while(1)
    p1 = ginput(1);
    if( isempty(p1) )
        break; % done inputting lines
    end
    drawmarker( p1(1), p1(2), 'o', 'g', 12 );
    p2 = ginput(1);
    drawmarker( p2(1), p2(2), 'o', 'g', 12 );
    drawline( p1, p2, 'g', 1 );
    dat(c,:) = [p1 p2];
    c = c + 1;
end
n = c - 1; % total number of lines
hold off;

%% Estimate vanishing point from n lines
R = [0 -1; 1 0];
for k = 1 : n
    u = (dat(k,1:2) - dat(k,3:4))';
    N(k,:) = R * ( u / norm(u) ); % normal to line
end

%% Estimate vanishing point
A = zeros(2,2);
b = zeros(2,1);
for k = 1 : n
    A = A + N(k,:)'; * N(k,:);
    b = b + N(k,:)'; * N(k,:) + dat(k,1:2)';
end
vEst = inv(A)*b;

%% Plot vanishing point
hold on;
h = plot( vEst(1), vEst(2), 'm.' );
set( h, 'MarkerSize', 24 );
hold off;

%%
%% DISPLAY AN IMAGE (NICELY)
%%
function[] = imshow( im )
[ydim,xdim,zdim] = size(im);
if( zdim == 1 ) % grayscale
    imagesc( im, [0 255] );
    axis image off;
    colormap gray;
else
    imagesc( uint8(im) );
    axis image off;
end

p = get( gcf, 'Position' );
set( gcf, 'Position', [p(1) p(2) xdim, ydim] );
set( gca, 'Position', [0 0 1 1] );
drawnow;

%%% DRAW A MARKER
%%%
%%% function[h] = drawmarker( x, y, symbol, color, sz )

for k = 1 : size(x,1)
    h = plot( x(k), y(k), symbol );
    set( h, 'Color', color, 'MarkerSize', sz );
end
drawnow;

%%% DRAW A LINE
%%% function[h] = drawline( p1, p2, color, width )

t = -1e3; q1 = (1-t)*p1 + t*p2;
t = 1e3; q2 = (1-t)*p1 + t*p2;

h = line( [q1(1) q2(1)], [q1(2) q2(2)] );
set( h, 'Color', color, 'LineWidth', width );
drawnow;