Parker Bossier

EECE 254

12/11/2011 (revised 5/27/2012)

Browser Zooming Based On Stereopsis

Introduction

This paper describes the approach and results of a Matlab program, *face_zoom.m*, which changes the zoom setting within a Firefox browser based on my distance from the screen. Note that this program uses an already-captured video as its input as opposed to a live stream. To accomplish this task, two Logitech webcams were affixed to the monitor about five inches apart and as coplanar as possible. A 2x2" blue paper square was attached to the user's forehead to serve as a tracker. The per-frame change in depth of the marker (computed with stereopsis as described later) was used to scale the browser window so as to assure a relatively constant apparent text size. All Matlab code can be found in the appendix at the end of this document.

Methods

First, I recorded (and loaded into Matlab) a stereo video in which I moved my head away from the screen and then towards it. See *figure 1* for example frames. I selected a set of eight matching point pairs followed by a color-mask sample used to separate the marker from the scene. The fundamental matrix of the two cameras was then computed with Matlab's *estimatefundamentalmatrix* function and the matching point set described above. The epipolar lines for each point were then calculated using Matlab's *epipolarLine* function. Next, the transformation homographies were computed with Matlab's *estimateUncalibratedRectification* function. See *figure 2*. At this point, all the necessary information had been collected to transform the left and right images so that their epipolar lines were as horizontal as possible. The first frame of the stereo video and the transformed frame are shown in *figure 3* in addition to the cyan epipolar lines through the yellow point pairs.

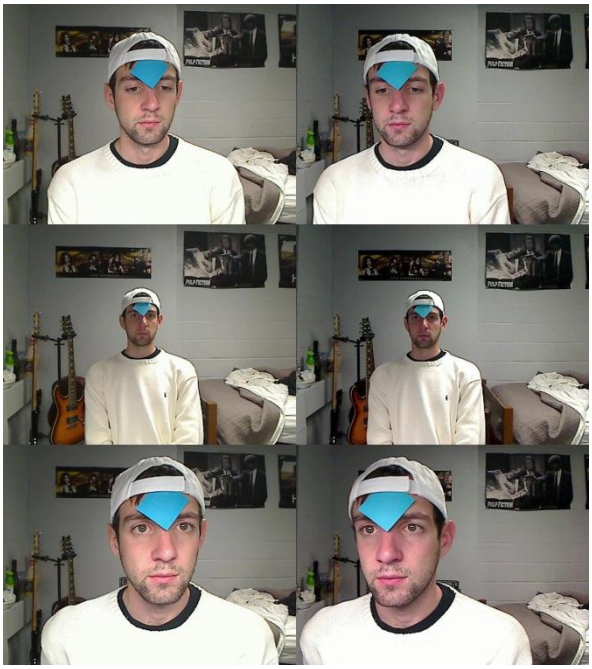

Figure 1. Frames 1, 16, and 32 of the input video.

-0.0000	0.0000	-0.0704
0.0000	-0.0000	0.0041
0.0674	-0.0056	0.9952
1.0341	-0.0869	32.7437
-0.3344	0.9157	13.4316
0.0001	-0.0001	1.0000
1.0779	-0.0548	17.5109
0.1164	1.1447	-49.2284
0.0000	0.0002	1.0000

Figure 2. From top to bottom: fundamental matrix, left homography transform, right homography transform, colormask sample.

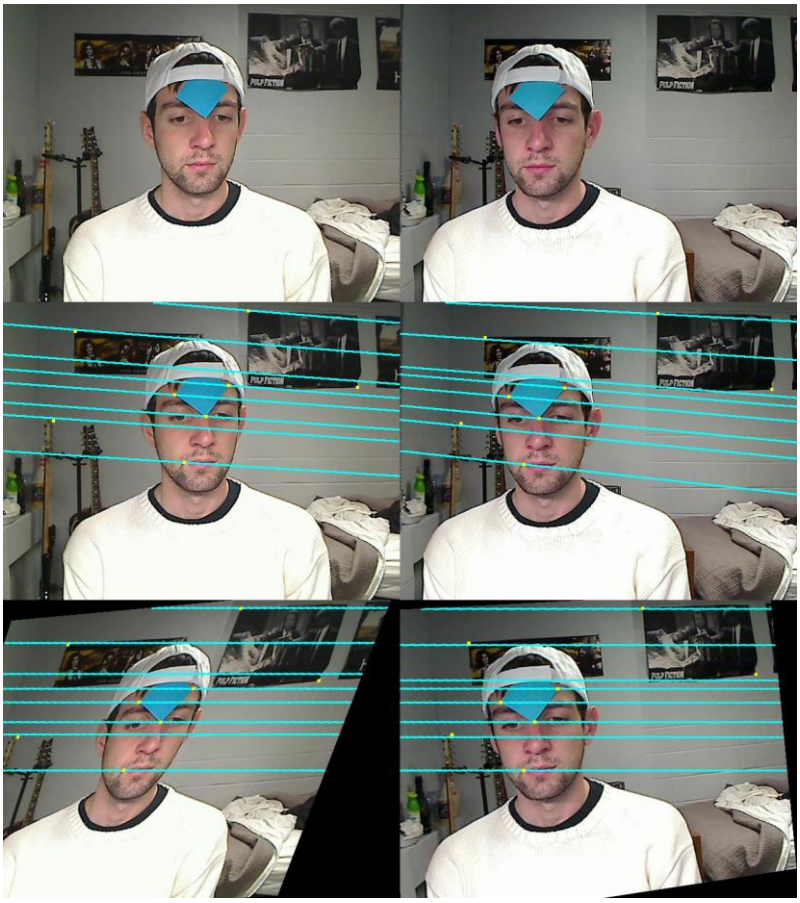

Figure 3. From the top down: original L and R images, original L and R images with epipolar lines drawn for each point, transformed images such that corresponding epipolar lines are collinear.

The next step was to process the video file as a whole. For each frame, the left and right images were color-masked using the *color_mask* function such that only the blue tracker remained in the frame (the background was mapped to black). This function first converted the input images to HSV and created masks such that only the target color (blue) was still visible. These binary masks were then eroded to get rid of some of the noise. Next, connected components in each mask were computed, and all non-largest components were zeroed. This step was necessary in order to ensure that any remaining extraneous noise was removed. Finally, the masks were dilated to recapture information lost in the erosion and then applied to the input images. See *figure 4*.

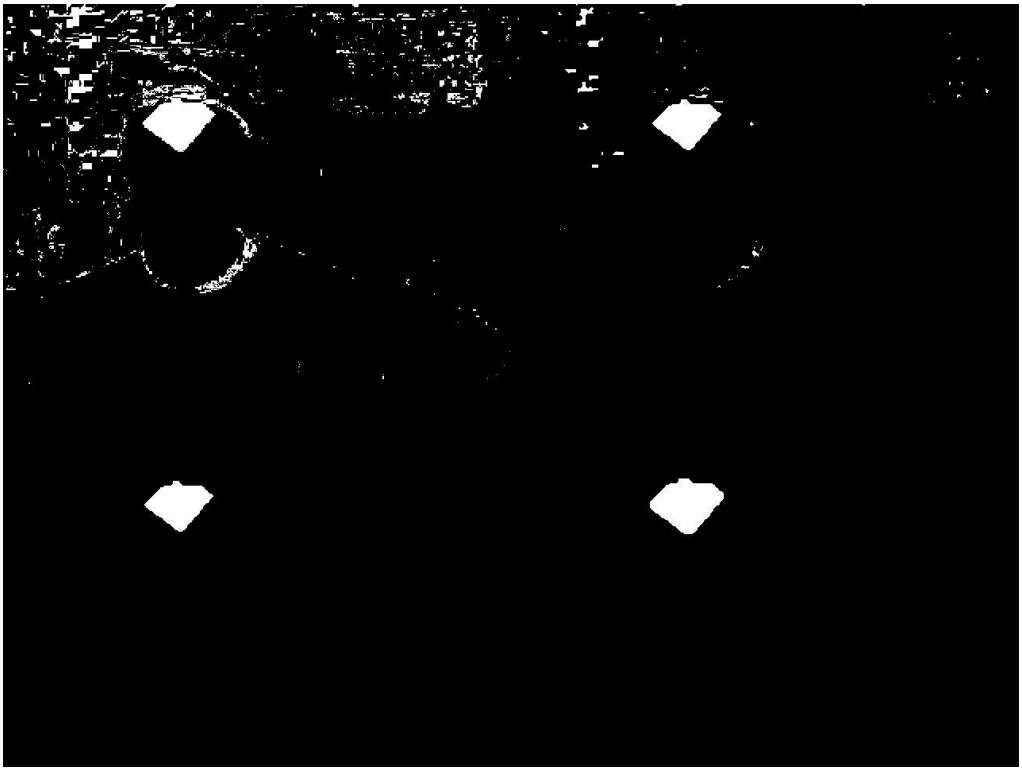

Figure 4. Colormask of the right image of the first frame (visible in figure 3). From left to right, top to bottom: after hue separation, after erosion, after non-largest connected component filtering, after dilation.

The images were then transformed using the transformation homographies computed earlier. The average disparity between points in the transformed left and right images was computed using the *compute_disp*arity function. This function picked a random sampling of non-black pixels in the left image. For each selected pixel, the corresponding pixel in the right image was found by scanning across the corresponding row in the right image. A moving window sum-of-absolute-difference (SAD) algorithm was used to do the scan. The pixel in the right image corresponding to the lowest SAD score was assumed to be the matching pixel. See *figure 5* for a sample set of matching pixels. The average disparity was then computed as the average of the computed disparities with outliers thrown out (*removeoutliers*).

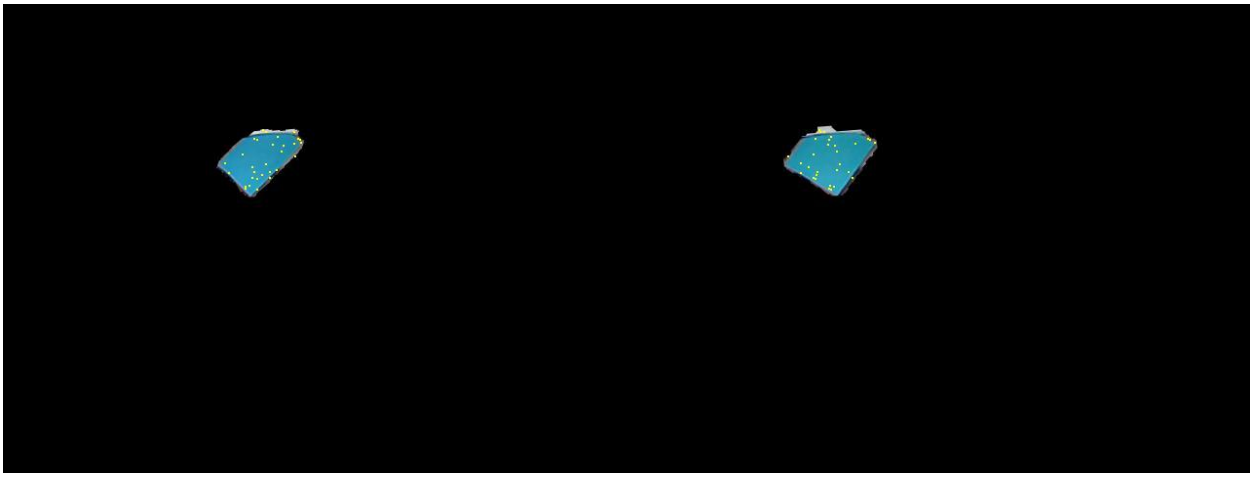

Figure 5. Left: randomly selected points in the left image. Right: right image points corresponding to the random points in the left image.

When repaying the video, the change in disparity between frames was used to scale the Firefox browser. To do this, I used the *SendKeys* functionality of the Windows environment through Matlab to send Control+ and Control- commands.

Results

Several screen captures of the program's output are provided in *figure 6*. As can be seen, when my face moved away from the screen, the magnitude of the disparity increased and the browser zoomed in. When my face moved back towards the screen, the magnitude of the disparity decreased, and the browser zoomed out. The displayed images are, from top to bottom, the original image, the colormasked image, and the transformed colormasked image. The left column is the left camera's image, and the right column is the right camera's image. The plot represents the average disparity of each frame. The red dot on the plot represents the current frame and disparity. The Firefox browser seen on the right of each screen capture started at a zoom level of 0, increased to maximum zoom, and then decreased as my face moved closer to the cameras, eventually reaching a negative zoom level when my face had passed its starting point. The NumLock message was caused by a rather annoying conflict between SendKeys and my Logitech keyboard software. When the program replays the depth information, the zoom response is quite acceptable.

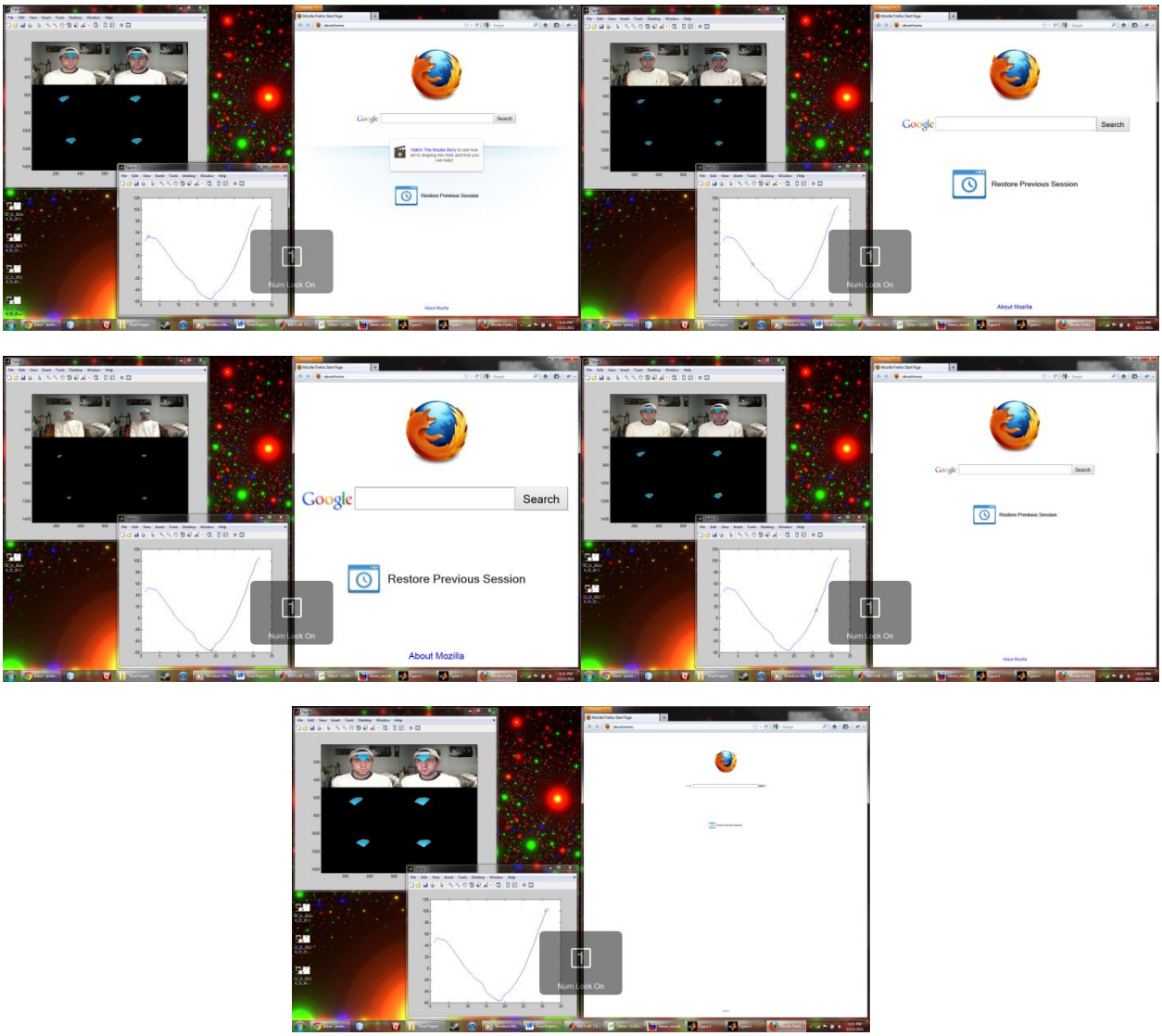

Figure 6. Five screen captures of the program in execution.

Conclusion

This paper explored applying depth from stereopsis techniques to control the zoom setting of a browser. To do this, a fundamental matrix was calculated from a set of eight matching point pairs. Transformation homographies were then calculated using the fundamental matrix. These homographies allowed for the computation of depth from stereopsis. The output of this routine (after initial computation) is fairly fluid and accurate. It should be noted, however, that a finer and more responsive level of browser zoom control would have allowed for a much smoother experience.

Appendix

<face_zoom.m>

```
%% Point capture/calibration

% initial defines
video_source = blue_head;
image_size = size(video_source(:,:,:,1)) ./ [1 2 1];
total_frames = size(video_source, 4);

% extract and split the first frame
demo_frame = video_source(:,:,:,1);
L = demo_frame(:,1:end/2,:);
R = demo_frame(:,end/2+1:end,:);

% three video frame image
imwrite([video_source(:,:,:,1); video_source(:,:,:,16); video_source(:,:,:,32)], 'video_preview.jpg')

% get 8 conjugate pixel pairs
figure(1)
image(demo_frame);
fprintf('Choose 8 matching points, left then right pointwise.\n')
[col_gin row_gin] = ginput(16);

% get the tracker HSV
fprintf('Select the colored tracker.\n');
[col row] = ginput(1);
rad = 2;
target_HSV = rgb2hsv(demo_frame(round(row)-rad:round(row)+rad,round(col)-rad:round(col)+rad,:));
target_HSV = mean(mean(target_HSV));

%% Fundamental matrix computation

% separate the list of points into xL and xR
xL = zeros([2 8]);
xR = xL;
```

```
for i = 1:size(row_gin)
   if mod(i, 2) == 1
      xL(:,ceil(i/2)) = [row_gin(i); col_gin(i)];
   else
      xR(:,i/2) = [row_gin(i); col_gin(i)-image_size(2)];
   end
end

% make x1 and x2 homogeneous
xL(3,:) = 1;
xR(3,:) = 1;

% compute the fundamental matrix
F = estimatefundamentalmatrix(xL(1:2,:), xR(1:2,:), 'Method', 'Norm8Point');

%% Draw epipolar lines on copies of the images (first frame)

% initialize the output images
L_epi = L;
R_epi = R;

% calculate the epipolar lines
epiLinesL = epipolarLine(F', xR(1:2,:));
epiLinesR = epipolarLine(F, xL(1:2,:));

% loop through each point
for i = 1:size(xL, 2)
   % left image
   l = epiLinesL(:,i);

   % slope-intercept calculation
   m = -l(2)/l(1);
   b = -l(3)/l(1);

   % draw the line
   for col = 2:image_size(2)-1
      row = round(m*col+b);
      if row >= 2 && row <= image_size(1)-1
         L_epi(row-1:row+1,col-1:col+1,:) = repmat(reshape([0 255 255], [1 1 3]), [3 3]);
      end
   end

   % draw the point
   point_rad = 2;
   rounded_x = round(xL(1:2,i));
   L_epi(rounded_x(1)-point_rad:rounded_x(1)+point_rad,rounded_x(2)-
point_rad:rounded_x(2)+point_rad,:) = ...
      repmat(reshape([255 255 0], [1 1 3]), [point_rad point_rad]*2+1);
```

```
    % right image
    l = epiLinesR(:,i);

    % slope-intercept calculation
    m = -l(2)/l(1);
    b = -l(3)/l(1);

    % draw the line
    for col = 2:image_size(2)-1
        row = round(m*col+b);
        if row >= 2 && row <= image_size(1)-1
            R_epi(row-1:row+1,col-1:col+1,:) = repmat(reshape([0 255 255], [1 1 3]), [3 3]);
        end
    end

    % draw the point
    rounded_x = round(xR(1:2,i));
    R_epi(rounded_x(1)-point_rad:rounded_x(1)+point_rad,rounded_x(2)-
point_rad:rounded_x(2)+point_rad,:) = ...
        repmat(reshape([255 255 0], [1 1 3]), [point_rad point_rad]*2+1);
end

%% Compute transform homographies

[T1 T2] = estimateUncalibratedRectification(F, xL(1:2,:), xR(1:2,:), image_size);

% transform the epipolar images
L_trans_epi = uint8(transformImagePoints(L_epi, T1));
R_trans_epi = uint8(transformImagePoints(R_epi, T2));

% epi image collage
imwrite([L R; L_epi R_epi; L_trans_epi R_trans_epi], 'first_frame.jpg');

%% Video processing loop

output_video = zeros(size(video_source) .* [3 1 1 1]);
disparities = zeros(total_frames, 1);

for frame_num = 1:total_frames
    % capture the current frame
    cur_frame = video_source(:,:,:,frame_num);

    % seperate the channels
    L = cur_frame(:,1:end/2,:);
    R = cur_frame(:,end/2+1:end,:);

    % color mask
```

```
    [L_masked R_masked] = color_mask(L, R, target_HSV, .08);

    % epipolar transform
    L_trans = uint8(transformImagePoints(L_masked, T1));
    R_trans = uint8(transformImagePoints(R_masked, T2));

    % record the output frame
    output_video(:,:,:,frame_num) = [L R;
                                     L_trans R_trans;
                                     L_masked R_masked];

    disparities(frame_num) = compute_disparity(L_trans, R_trans, 1);

    fprintf('Processing... %d%% complete.\n',uint8(frame_num/total_frames*100))
end

output_video = uint8(output_video);
disparities = smooth(disparities);
disp('Processing complete.')

%% Video replay

% constants
disparity_zoom_thresh = 3;

% launch and select Firefox
handle = actxserver('WScript.Shell');
handle.Run('firefox');
print('Waiting for Firefox...')
pause(5)

for i = 1:total_frames-1
    % display the frame number
    fprintf('Viewing frame %d.\n', i)

    % show the input
    figure(1)
    image(output_video(:,:,:,i));

    % show the plot
    figure(2)
    plot(1:total_frames, disparities)
    hold on
    plot(i, disparities(i), 'or')
    hold off

    % calculate the zoom string
    delta_disparity = disparities(i+1)-disparities(i);
```

```
    if delta_disparity > disparity_zoom_thresh
        key_string = '^{-}'
        handle.AppActivate('firefox.exe');
        handle.SendKeys(key_string);
    elseif delta_disparity < -disparity_zoom_thresh
        key_string = '^{+}'
        handle.AppActivate('firefox.exe');
        handle.SendKeys(key_string);
    end

    pause(.25)
end
```

</face_zoom.m>

<color_mask.m>

```
function [L_out R_out] = color_mask(L, R, target_HSV, threshold)

% expand target_HSV
h = target_HSV(1);
s = target_HSV(2);

% convert to HSV
L_HSV = rgb2hsv(L);
R_HSV = rgb2hsv(R);

% generate masks to zero everything outside of the threshold
mask_L = make_mask(L_HSV, h, s, threshold);
mask_R = make_mask(R_HSV, h, s, threshold);

% mask the input images and output the original, color, masked images
L_out = repmat(mask_L, [1 1 3]) .* L;
R_out = repmat(mask_R, [1 1 3]) .* R;
end

% creates a mask that lets the selected HSV values show through
function mask = make_mask(HSV_im, target_h, target_s, threshold)
    % cut out values out of the hue range
    mask = HSV_im(:,:,1) > (target_h-threshold/2) & HSV_im(:,:,1) < (target_h+threshold/2);

    %foo = mask*255;

    % erode the image to eliminate small noise
    mask = imerode(mask, strel('disk', 2));

    %foo1 = mask;
```

```
    % label the connected components and mask out the non-largest
    % components
    L = bwlabel(mask, 8);
    vector_L = reshape(L, 1, numel(L));
    vector_L(vector_L == 0) = [];
    mask = L == mode(vector_L);

    %foo2 = mask;

    % dilate the mask
    mask = imdilate(mask, strel('rectangle', [7 7]));

    %foo3 = mask;
    %image(uint8(255*[foo foo1; foo2 foo3]));
    %imwrite(uint8(255*[foo foo1; foo2 foo3]), 'mask.jpg');
    %pause

    % cast and return the mask
    mask = uint8(mask);
end
```

</color_mask.m>

<compute_disparity.m>

```
function disparity = compute_disparity(L, R, radius)

% capture the image size
image_size = size(L);

% dispartities index
disparity = [];
disp_index = 1;

% loop through a random set of non-zero pixels
non_zero = find(sum(L, 3) > 0);
non_zero = randsample(non_zero, min(30, numel(non_zero)));
[rows cols] = ind2sub(image_size, non_zero);

for i = 1:size(rows)

    % get the current row/col
    row = rows(i);
    col = cols(i);

    % capture the needle
    needle = L(row,col,:);
```

```
   % initialize the output (infinities)
   result = 1./zeros([image_size(2) 1]);

   % look at each (2*radius+1)^2 window going across the rectified scan line
   for inner_col = radius+1:image_size(2)-radius-1
      window = R(row-radius:row+radius,inner_col-radius:inner_col+radius,:);

      % subtract each window pixel from the needle and return the sum of
      % absolute differences
      result(inner_col) = sum(sum(sum(abs(double(repmat(needle, [2*radius+1 2*radius+1])) -
double(window)))));
   end

   % find the mean location of the minimum value
   found = mean(find(result == min(result)));

   % record the disparity
   disparity(disp_index) = col - found;

   %L(row:row+1,col:col+1,:) = repmat(reshape([255 255 0], [1 1 3]), [2 2]);
   %R(row:row+1,col-round(disparity(disp_index)):col-round(disparity(disp_index))+1,:) = ...
      repmat(reshape([255 255 0], [1 1 3]), [2 2]);

   % increment the index
   disp_index = disp_index + 1;
end

%image([L R])
%imwrite([L R], 'LR_point_matches.jpg')
%pause

% remove disparity outliers is applicable and mean the values
if numel(disparity) >= 3
   disparity = mean(removeoutliers(disparity));
else
   disparity = mean(disparity);
end
end
```

</compute_disparity.m>

<transformImagePoints.m>

```
% Modified and excised from cvexShowStereoImages.m

%==========================================================================
% Transform the image
%==========================================================================
```

```
function I = transformImagePoints(I, t)

htrans = vision.GeometricTransformer('TransformMatrixSource', 'Input port');
t = single(t);
I = step(htrans, single(I), t);
```

</transformImagePoints.m>

<removeoutliers.m>

This code was downloaded from http://www.mathworks.com/matlabcentral/fileexchange/24885-remove-outliers/content/removeoutliers.m

</removeoutliers.m>