CS223b, Winter 2005
HW2 Solutions

Let's do the short answer questions first...

Problem 2

1. Camera Baseline (3 pts): Assume you want to use two of the webcams from last week's assignment for stereo. Your project requirements state that you should be able to distinguish a point that is 10m from your camera from one 9m away. How far away do you have to place your two cameras horizontally if you want their viewing directions to be parallel?

This problem presents the complication that it's not entirely possible to predict how a *continuous* disparity will translate into a *discrete* disparity without more information... so we'll accept multiple solutions to this problem:

Solution 1

Another approach is to assume that the disparity between the 9m object and the 10m object will be exactly one pixel. In other words:

$(d_9 - d_{10}) = 1$

...using the formula from the text and substituting for disparities:

$(fT/9) - (fT/10) = 1$

$(fT)(1/9 - 1/10) = (fT)(0.0111) = 1$

$T = 1 / (f * 0.0111) = 1/ (462 * .01111) = .195m$

The catch with this solution is that if I plug back in to get the actual disparities, I see that:

$d_9 = fT/9 = 10.0$
$d_{10} = fT/10 = 9.0$

So the disparities are indeed different by one pixel... but I know that given a real system computing real disparities, there is *some* point where I could get d_9 equal to one and d_{10} equal to zero, if you imagine just gradually widening the baseline from 0 until the disparity of the 9m object popped from zero to one.

So this is the "conservative" solution...

Solution 2

A good trick here is to assume that the 9m object will be the farthest possible object that *just barely* gives us one pixel of disparity. This lets me avoid doing any computation with the 10m object… the important thing is that decreasing the baseline decreases disparity, so if I do find the baseline that *just barely* gives me *exactly* one pixel

of disparity at 9m, I know that I have the smallest acceptable baseline. And by definition I'll have zero pixels of disparity at 10m (or 9.00001m, for that matter), so I know I can distinguish the 9m object from the 10m object.

Now page 144 in the book tells us the relationship between depth and disparity:

$Z = f * (T/d)$

...where Z is object distance, f is camera focal length, T is the baseline, and d is the observed disparity.

I want a disparity of 1 pixel (i.e. s_x) at a Z (distance from the camera plane) of 9m, and f/s_x is 462 (from your first homework, or from the assignment FAQ). Doing the substitutions, I get:

$9m = f * T / s_x = (f / s_x) * T = 462 * T$

…and solving for T, I get 0.0195m.

Note that if I made T any smaller, I would need a larger depth to make $d = s_x$, so I wouldn't be able to tell the 9m object from the 10m object. So this really is the smallest acceptable baseline…

The catch with this solution is that this basically assumes that disparities are continuous (which is an aggressive assumption, but you have no information to the contrary), or that I can estimate disparities with fantastic subpixel accuracy.

2. Depth recovery (2 pts): Can depth be recovered from a stereo pair taken under the following circumstances? Briefly justify your answers.

(a) Two images taken by orthographic projection from cameras with parallel optical axes?

No. An orthographic camera only captures rays that are parallel to its optical axis, so none of the rays captured by the two cameras could possibly intersect, making depth triangulation impossible.

(b) Two images taken by a single perspective proj. camera rotated about its optical center?

No. Capturing two images from the same location at two orientations is equivalent to having a stereo rig with a baseline of zero; triangulating depth is impossible in this configuration.

Matlab Solutions

A simple solution is presented as ‘stereo.m’, along with a tester program called ‘test_stereo.m’.

An optimized solution that uses convolution to replace loops, performs more extensive precomputation, and performs left-right consistency checking is presented as ‘stereoOpt.m’, along with a tester program called ‘test_stereoOpt.m’. The original m-files are available on the course web page at cs223b.stanford.edu.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% stereo.m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% function [d] = stereo(corrL, corrR)
%
% CS223b
% Winter '05
%
% Compute disparity from two rectified grayscale images.  Input parameters
% are rectified left and right images.
%
% Adapted by Dan Morris from:
% http://robots.stanford.edu/cs223b04/homework/hw3/hw3_sol.pdf
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [d] = stereo(corrL, corrR)

% The maximum allowable disparity...
max_d = 10;

% The _radius_ of the search window... a radius of 2, for example,
% gives a 5x5 window
rad = 3;

% Figure out the iimage and window size
[rows cols] = size(corrL);
imL = zeros(rows+(2*rad),cols+(2*rad));
imR = zeros(rows+(2*rad),cols+(2*rad));
imL(rad+1:rad+rows,rad+1:rad+cols) = corrL;
imR(rad+1:rad+rows,rad+1:rad+cols) = corrR;

% Initialize the disparity matrix
d = zeros(size(imL));

% For each row...
for u = rad+1:rows+rad

  % For each column we can search...
  for v=max_d+rad+1:cols+rad

  % The 2*rad x 2*rad + 1 window we want to _find_ in the right image
  smL = imL(u-rad:u+rad,v-rad:v+rad);

  % A vector of correlation values; the minimum will be our winner...
  %
  % So we initialize it to the largest possible value...
  vals = 99e9*ones(1,cols+2*rad);

  % Search back to the left in the right image...
  for mv=v-max_d+1:v

    % The current window in the right image...
    smR = imR(u-rad:u+rad,mv-rad:mv+rad);

    % The correlation value for this window...
    vals(mv) = sum(sum((smL-smR).^2));
```

```
  end

  % The best correlation value and the corresponding disparity
  [mn, dp] = min(vals);

  % Put this in the output vector
  d(u,v) = v-dp;
  end

end

% Clip away the useless parts of our disparity map
d = d(rad+1:rad+rows, rad+1:rad+cols);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% test_stereo.m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% CS223b
% Winter '05
%
% Dense stereo correlation driver program
%
% Dan Morris, Stanford University
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Close all figures
close all;

disp('Reading images...');

% Pick a set of images to read...
%im_l = imread('real7L.pgm');
%im_r = imread('real7R.pgm');
im_l = imread('toy3L.pgm');
im_r = imread('toy3R.pgm');

disp('Calling correlation routine...');

% Start a timer; we want to know how long our program is taking...
tic;

% Compute a disparity map...
d = stereo(im_l,im_r);

% Tell us how long the whole process took
toc

disp('Normalizing...');

% Normalize the disparity map to make it look pretty...
d = d * (255/max(max(d)));
figure(1);
imagesc(im_l);
colormap('gray');
figure(2);
imagesc(im_r);
colormap('gray');
figure(3);
imagesc(d);
colormap('gray');
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% stereoOpt.m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% CS223b
% Winter '05
%
% function [disparity_map, consistency, right_disp_map] =
% stereoOpt(left_im, right_im, maxdisp, window_size, check_consistency)
%
% Optimized dense stereo correlation function
%
% Dan Morris, Stanford University
%
% Computes a disparity map based on the stereo images 'left' and 'right',
% scanning for maxdisp pixels from each pixel in the left image.  If
% the original image is size (h,w), the disparity map will have size
% (h,w-maxdisp).
%
% The second return value is a consistency-checked version of the
% disparity map, in which all disparities that were not identical in
% the left-to-right-matched map and the right-to-left-matched map
% are set to 0.
%
% window_size is the edge size of the (two-dimensional) neighbor-averaging
% filter that will be applied to the difference maps between the two images
% before computing the disparity maps.
%
% maxdisp defaults to 15, window_size defaults to 5
%
% The third matrix returned, right_disp_map, is the right-to-left-matched
% disparity map used in consistency checking
%
% If 'check_consistency' is zero, no consistency check is performed and the
% second and third return values are not computed.  'check_consistency'
% defaults to zero.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [disparity_map, consistency, right_disp_map] = ...
  stereoOpt(left_im, right_im, maxdisp, window_size, check_consistency)

% Check arguments and create defaults where we need to...
if (nargin < 2)
    fprintf(1,'correlation_match requires at least two arguments...\n');
    return;
end

if (nargin < 3)
    maxdisp = 15;
end

if (nargin < 4)
    window_size = 5;
end

if (nargin < 5)
  check_consistency = 0;
end
```

```
% Make sure the input images are the same size
eqim = (size(left_im)==size(right_im));
if (find(eqim==0))
    fprintf(1,'Images are not the same size...\n');
    return;
end

% Create a matrix of size [height,width,maxdisp] to cache all
% the squared differences
dcache = zeros(size(left_im,1),size(left_im,2)-maxdisp,maxdisp+1);
if (check_consistency)
  % A second difference cache for the reverse disparity map
  dcache_r = dcache;
end

% Compute all left-to-right and right-to-left squared differences...
%
% I reverse the indexing in the right-to-left matched computation so the indices
% will have the same intuitive meaning as those used for the left-to-right matched
% differences (higher index into the resulting matrix is a higher disparity)
for(d=0:maxdisp)
    dcache(:,:,d+1) = (left_im(:,d+1:size(left_im,2)-maxdisp+d) - ...
      right_im(:,1:(size(left_im,2)-maxdisp))).^2;
    if (check_consistency)
      dcache_r(:,:,maxdisp+1-d) = (right_im(:,d+1:size(left_im,2)-maxdisp+d) - ...
        left_im(:,1+maxdisp:(size(left_im,2)))).^2;
    end
end

% Convolve each isodisparity plane with an averaging filter (this is the
% windowing operation)
avg_filter = ones(window_size,window_size);
for(d=1:maxdisp+1)
    dcache(:,:,d) = conv2(dcache(:,:,d),avg_filter,'same');
    if (check_consistency)
      dcache_r(:,:,d) = conv2(dcache_r(:,:,d),avg_filter,'same');
    end
end

% Find the minimum difference at each x,y position; this is the disparity at
% that point
[diffs,inds] = min(dcache,[],3);
if (check_consistency)
  [diffs_r,inds_r] = min(dcache_r,[],3);
end

% Subtract 1 from each index, since the lowest possible disparity should be zero,
% not 1
inds = inds - 1;
if (check_consistency)
  inds_r = inds_r - 1;
end

% Do the consistency check if the caller asked us to...
if (check_consistency)

  % Compute a ramp vector for left-right consistency checking
  h = size(left_im,1);
  w = size(left_im,2) - maxdisp;
```

```
  ramp = linspace(1,h*w,h*w);
  ramp = reshape(ramp,h,w);

  % Evaluating the right disparity map at this vector _should+ yield the left
  % disparity map
  crosscheck = ramp - inds;
  crosscheck(find(crosscheck<1)) = 1;
  right_check = inds_r(crosscheck);

  % Anywhere we didn't find a match, we have a 'bad' disparity
  bad_inds = find(abs(right_check - inds) > 0);

  checked_left = inds;
  checked_left(bad_inds) = 0;

  % Set the output variables
  consistency = checked_left;
  right_disp_map = inds_r;
else
  consistency = 0;
  right_disp_map = 0;
end

% ...and return the disparity map...
disparity_map = inds;
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% test_stereoOpt.m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% CS223b
% Winter '05
%
% Driver for optimized dense stereo correlation function
%
% Dan Morris, Stanford University
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Start a timer; we want to know how long our program is taking...
tic;

% Pick a set of images to read...
left_im = double(imread('real7L.pgm'));
right_im = double(imread('real7R.pgm'));
%left_im = double(imread('toy3L.pgm'));
%right_im = double(imread('toy3R.pgm'));

% Pick a maximum disparity and window size (note that the size here is the
% edge size, not the radius, of the window)
maxdisp = 45;
window_size = 13;

% Should we ask for a left-right consistency check also?
do_consistency = 0;

disp('Starting correlation routine...');
[disp_map, consistency, disp_map_r] = ...
  stereoOpt(left_im,right_im,maxdisp,window_size,do_consistency);
disp('Finished correlation routine...');

% Tell the user how long that took
toc

% Show the user all the pretty pictures
close all;

figure(1);
imagesc(left_im);
colormap(gray);
title('Original left image');

figure(gcf+1);
imagesc(right_im);
colormap(gray);
title('Original right image');

figure(gcf+1);
imagesc(disp_map);
colormap(gray);
title('Left-to-right matched disparity map');

if (do_consistency)
  figure(gcf+1);
  imagesc(disp_map_r);
```

```
  colormap(gray);
  title('Right-to-left matched disparity map');

  figure(gcf+1);
  imagesc(consistency);
  colormap(gray);
  title('Consistency-checked Left-to-right matched disparity map');
end
```