Due April 20

Homework 5: CMSC 426

1. **60 points.** Stereo Matching: Implement a stereo matching algorithm that computes the disparity for a pixel in the left image by matching it to nearby pixels in the right image, for rectified pairs of images (epipolar lines are horizontal scan lines). Use Sum of Square Differences (SSD) between a region around the pixel in the left image and a region around each of the possible matching pixels in the right image. To make your algorithm run faster, you may also assume that the disparity will always range between ±16. This means, for example, that the pixel (17,23) in the left image will match, in the right image, one of the pixels (17,23), (17, 22), …(17,8) (or be occluded). Your function should look something like:

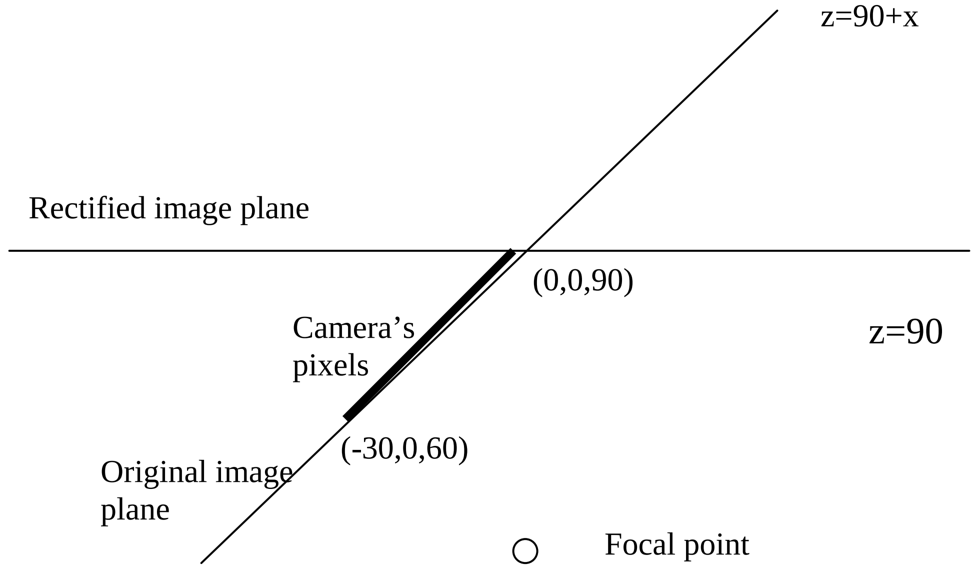
```
D = stereo_ssd(left, right, n),
```

where D is a disparity map, left and right are the two images, and an nxn window is used for comparison. That is, D is the same size as left and right, and gives the disparity for every pixel in left. Test your algorithm on the image pairs I1L, I1R, and the pair T3bw, T4bw. Print images of the resulting disparity maps.

2. Image Rectification.

a. **10 points.** Pencil and paper warm-up. Suppose we have a camera with a focal point at (0,0,0) and an image plane described by z = 90 + x. We want to rectify this image, so that it appears as it would if taken with a camera with a focal point at (0,0,0) and an image plane at z = 90 (see picture below).

i. Suppose the camera takes a picture with pixels extending from (-30, 0 60) to (0,0,90). Where would these pixels appear in the rectified image? (Part b, below, contains a hint about the answer).



ii. If we want to find the intensity in the rectified image at the position (-25,0,90), where would we need to look in the original image?

b. **20 points** Now you're going to write a program to perform rectification for a 1D image in the y=0 plane. This will have the form:

```
   [O,s] = rectify1D(I,xs,zs,xe,ze).
```

We will assume that the focal point is always at (0,0,0). We will also assume that all coordinates are written in units of pixels. The pixels in the original image are contained in the vector I. In world coordinates, this image appears on the image plane beginning at the point (xs, 0, zs), and extends to the point (xe, 0, ze). This means that we must have length(I) >= sqrt((xs-xe)^2 + (zs-ze)^2), so that the region in the image plane that contains the image is no longer than the number of pixels we have in the image. If I contains extra pixels, they don't need to be used in the rectified image.

Our goal is to rectify the image so that it appears as if it were taken by a camera with a focal point at (0,0,0), and an image plane at z=100. So to rectify an image according to the geometry shown in the picture above, we would need I to be a vector 50 long containing 50 intensities, and we would call:

```
[O,s] = rectify1D(I, -30, 60, 0, 100).
```

rectify1D will produce two outputs. O is a vector of pixel intensities for the rectified image. s indicates the x coordinate in the image plane where the rectified image starts.

Test your function by executing the calls:

```
  [O,s] = rectify1D(ones(1,50), -30, 60, 0, 100).
```

This should return for O a vector containing 50 1s.

```
 [O,s] = rectify1D(1:50, -30, 60, 0, 100).
```

Turn in your code and the results you get for these two test images. Now load I3 from the class web page and run your code on the first line of this 2D image. E.g., `[O,s] = rectify1D(I3(1,:), 0,100, 98,51).`
Display the results by plotting them: plot(O), and turn it in.

c. **10 Points.** Now extend this code to rectify a 2D image. Assume that the original 2D image differs from the rectified image only by a rotation about the y axis, to keep things relatively simple. This means that we have the same geometry shown above, except we could draw the same picture for any plane with a constant y value. Your program, rectify2D should take as input an unrectified 2D image, along with the (x,y,z) coordinates in the world of the lower left-hand corner of the image, and of the upper right hand corner. To be more explicit, if the lower left corner of the image is (xs,ys,zs), and the upper right corner is (xe,ye,ze), then you may assume that the other two corners of the rectangular image are (xs,ye,zs), and (xe,ys,ze).
You may assume, again, that the focal point is at (0,0,0), and that the rectified image should be in the z = 100 plane. Run this code on the image I3.jpg, with the lower left-hand corner designated at (0,0,100), and the upper right-hand corner at (100,100,50). Print and turn in the result.