Assignment A11: Range Data

Clinton Fernandes

u1016390

04/27/2016

A11

CS 6320

1. Introduction

The purpose of this assignment is to develop and study range data analysis. Given the synthetic range data image and its corresponding ground truth data defining the topo class at each point, it is needed to study the impact of Gaussian noise in the range value [0 1] on the topo classification.

As an additional exercise, the topo function (classification) has to be applied onto the actual range image found in scene1.mat (data provided).

The question that can be answered after the completion of this assignment is, how does the Gaussian noise affect the topo classification?

2. Method

Range images (depth maps), instead of brightness or color information, store the depth at which the ray associated with each pixel first intersects the scene observed by a camera. After having obtained the range data through any of the different range-sensing technology, image segmentation and classification can be performed onto it. This is an approach to extract features from imagery based on objects, where pixels in close proximity and having similar characteristics are grouped together into a segment.

The segmentation and classification process requires the development of the following functions:

CS5320_range2pts, CS5320_normals, CS5320_planes, CS5320_extract_data, CS5320_topo

CS5320_range2pts

At first, the range may not be in the x,y,z points image form. Hence we need to the convert range image to full x,y,z points image. The CS5320_range2pts function does this task.

Input to the function is the range image: range_im (mxnxd array)

Here d is either 1 or 3. If d = 1 then the image is just a height map and if d=3 is already an x,y,z image in which case, just return the same image.

Output of the function is points_im (mxnm3 array): x,y,z points in image format.

Algorithm:

- [rows,cols,d] = size(range_im);
- If d ==3, then return the same image, else continue
- For all rows (r) and columns (c)
    - points_im(r,c,1) = c;
    - points_im(r,c,2) = rows - r + 1;
    - points_im(r,c,3) = range_im(r,c);
- End

CS5320_normals

This function computes the normals on range data (at every point position)

Input to the function is : im (mxnx3 array): range data image

: k (int): uses 2k+1 by 2k+1 neighborhood for normal computation

Output of the function is: normals (mxnx3 array): normals at each point

Note: As we define a neighborhood around each pixel to compute the normal at that particular position, the normals at the boundaries (k pixels width) of the image will not be computed, as there are not enough or no neighborhood pixels for these positions. Hence the normals will be set to 0 by default.

Algorithm:

- [rows,cols,planes] = size(pts_im);
- for yr=1+k:rows-k
    - for xc = 1+k:cols-k
        - for pl=1:3
            - xdiff(1,pl) = pts_im(yr,xc+k,pl) - pts_im(yr,xc-k,pl);
            - ydiff(1,pl) = pts_im(yr-k,xc,pl) - pts_im(yr+k,xc,pl);
        - end
        - N = cross(xdiff,ydiff);
        - N = N/norm(N);
        - for pl=1:planes
            - normals(yr,xc,pl) = N(pl);
        - end
    - end
- end

CS5320_planes

This function is used to fit planes at every pixel by looking at the neighboring positions. The parameters in the plane equation (a, b, c, d) are given as the output of the function at every position of the image. Also, the mean square error (between the fitted plane considered points in a patch) is given as the output.

NOTE: This function of plane fitting can be implemented by various methods. I have developed 2 functions, **CS5320_planes** and **CS5320_planes_LS.** The function CS5320_planes_LS follows the Least squares method to fit the plane and has been stored in this assignment folder. However it will not be described in this section. Given below is the information about the CS5320_planes function which uses the normal components as the plane parameters (a,b,c).

On input: points (mxnx3 array): x,y,z points

normals (mxnx3 array): surface normal at each point

k (int): uses 2k+1 by 2k+1 window to fit plane

On output: planes (mxnm5 array): plane parameters and error of fit

channels 1-4: a,b,c,d plane parameters

channel 5: mean error of fit in window

Algorithm:

- for i = 1+k:rows-k
  - for j = 1+k:cols-k
    - % assign the normal components as a,b,c
    - planes(i,j,1) = normals(i,j,1);
    - planes(i,j,2) = normals(i,j,2);
    - planes(i,j,3) = normals(i,j,3);
    -
    - % get xyz values of neighborhood
    - x = points(i-k:i+k,j-k:j+k,1);
    - y = points(i-k:i+k,j-k:j+k,2);
    - z = points(i-k:i+k,j-k:j+k,3);
    -
    - % now find the value of d
    - planes(i,j,4) = -(normals(i,j,1)*mean(x(:))...
    - + normals(i,j,2)*mean(y(:)) + normals(i,j,3)*mean(z(:)));
    -
    - %now compute the error
    - temp = (planes(i,j,1)*x + planes(i,j,2)*y...
    - + planes(i,j,3)*z + planes(i,j,4)*ones(2*k+1)).^2;
    - planes(i,j,5) = sum(sum(temp))/(2*k+1)^2;

    - end
- end

CS5320_extract_data

This function pulls segment of data (in between 2 endpoints) from an image.

Input: im (mxnxp array): p-dimensional array

: Endpoint rows and column values: r1, c1, r2, c2

: channels (1xk vector): channel indexes to extract

Output: v (qxs array): extracted data from channels on the segment from [r1,c1] to [r2,c2]

: q is the number of pixels from [r1,c1] to [r2,c2]

: s is the number of channels extracted

Algorithm:

- set, s = size(channels,2);
- STEP = 0.1;
- temp_im = zeros(max(r1,r2),max(c1,c2)); %to track visited rows, columns

%now initiate the output variable (v) be storing the data belonging to r1, c1

- for ch = 1:s
    - v(1,ch) = im(r1,c1,channels(1,ch));
- end
- breakLoop = 0; %this is to break the while loop when required
- r = r1;
- c = c1;
- temp_im(r1,c1) = 1; % set r1,c1 location to 1 to note that this position is visited
- theta = posori(atan2(r2-r1,c2-c1)); % angle of the segment to be extracted from image
- d_row = STEP*sin(theta); %set step increment for row direction
- d_col = STEP*cos(theta); %set step increment for column direction
- dist_1_2 = norm([r1;c1]-[r2;c2]); %distance between r1c1 and r2c2
- while breakLoop == 0
    - r = r + d_row; %new row to check
    - c = c + d_col; %new column to check
    - d = norm([r1;c1]-[r;c]); %distance between new rc to r1c1
    - if d>dist_1_2%if distance betn current rc and r1c1 is more than max_dist
        - breakLoop = 1; %here, break the loop
    - else
        - ri = max(1,floor(r)); %lowest integer for row
        - ci = max(1,floor(c)); %lowest integer for column
        - if temp_im(ri,ci)==0 %if image rc position havn't been checked

- temp_im(ri,ci) = 1; %set image check position to 1
- temp_v = zeros(1,s);
- for ch = 1:s
  - temp_v(1,ch) = im(ri,ci,channels(1,ch));
- end
- v = [v;temp_v];

        - end
    - end
- end
- if temp_im(r2,c2)==0 % if r2c2 position is not checked
    - temp_v = zeros(1,s);
    - for ch = 1:s
        - temp_v(1,ch) = im(r2,c2,channels(1,ch));
    - end
    - v = [v;temp_v];
- end

CS5320_topo

This function is used to determine the topographical classes of range image. To check the topographical features, here, I have considered the depth values (or z values) from the range image.

On input: points (mxnx3 array): x, y, and z channel range image

normals (mxnm3 array): normals at each point

planes (mxnx5 array): plane info at each point (a, b, c, d, err)

k (int): uses 2k+1 by 2k+1 window

On output: topo (mxnx7 array): topo class likelihoods (in range [0,1])

channel 1: FLAT

channel 3: PIT

channel 4: RIDGE

channel 5: RAVINE

channel 6: HILLSIDE

channel 7: JUMP_EDGE

Algorithm:

- initialize topo = zeros (mxnx7)
- At every pixel i, j

- extract 4 directions of data values across the pixel's neighborhood, along 0, 45, 90, 135 degrees.
- Use polyfit and check if all 4 sets are linear, then the pixel is FLAT
  - Update the topo (i, j, 1) = 1
- else if all go down on either side of the pixel (z values of pixel is more than its neighborhood), then the pixel is a peak.
  - Update the topo (i, j, 2) = 1
- else if all go up on either side of the pixel (z values of pixel is less than its neighborhood), then the pixel is a pit.
  - Update the topo (i, j, 3) = 1
- else if one direction is linear (using polyfit) and the others go down on either side (z values of pixel is more than its neighborhood), then the pixel is a ridge
  - Update the topo (i, j, 4) = 1
- else if one direction is linear and the others go up on either side, then the pixel is a ravine
  - Update the topo (i, j, 5) = 1
- Else if there is any discontinuity/jump edge at the pixel {}
  - Update the topo (i, j, 7) = 1
- For the rest of the positions, mark them as hill sides
  - Update the topo (i, j, 6) = 1

- end

After having developed the functions, we need to use them for image segmentation and for topographical classification.

Also, now we need to study the impact of Gaussian noise in the range value on the topo classification. For this, we will add the Gaussian noise to the original range image in the following manner.

```
im_w_noise = im + randn(size(im))*sigma
```

here, sigma is the standard deviation.

To Study the effect of nose on the topo classification, we will vary sigma and see the output of CS5320_topo.

Also, we will apply the topo function to the actual range image found in scene1.mat (scene1.depthImage) in the class data/A11 directory.

3. Verification

CS5320_range2pts

A test range data array with dimensions 3x3x1 was given to the function.

rp_im1 = [1 2 3; 8 4 2; 5 3 2];

Code:

```
%Verification for CS5320_range2pts
%d==1
rp_im1 = [1 2 3; 8 4 2; 5 3 2];
rp_im1_pts = CS5320_range2pts(rp_im1);
```

```
K>> rp_im1_pts

rp_im1_pts(:,:,1) =

     1     2     3
     1     2     3
     1     2     3


rp_im1_pts(:,:,2) =

     3     3     3
     2     2     2
     1     1     1


rp_im1_pts(:,:,3) =

     1     2     3
     8     4     2
     5     3     2
```

It was seen that the output (figure on right) have the x coordinates, y coordinates and the z coordinates of the points in the range data correctly.

Thereafter, a range data array with dimensions 3x3x3 was given to the function.

rp_im3(:,:,1) = randn(3);

rp_im3(:,:,2) = randn(3);

rp_im3(:,:,3) = randn(3);

Code:

```
rp_im3(:,:,1) = randn(3);
rp_im3(:,:,2) = randn(3);
rp_im3(:,:,3) = randn(3);
rp_im3_pts = CS5320_range2pts(rp_im3);
```

Here, the output should have been the same. This was obtained. Left figure: input, Right figure: output

```
K>> rp_im3

rp_im3(:,:,1) =

    0.3252   -1.7115    0.3192
   -0.7549   -0.1022    0.3129
    1.3703   -0.2414   -0.8649


rp_im3(:,:,2) =

   -0.0301    1.0933    0.0774
   -0.1649    1.1093   -1.2141
    0.6277   -0.8637   -1.1135


rp_im3(:,:,3) =

   -0.0068    0.3714   -1.0891
    1.5326   -0.2256    0.0326
   -0.7697    1.1174    0.5525
```

```
K>> rp_im3_pts

rp_im3_pts(:,:,1) =

    0.3252   -1.7115    0.3192
   -0.7549   -0.1022    0.3129
    1.3703   -0.2414   -0.8649


rp_im3_pts(:,:,2) =

   -0.0301    1.0933    0.0774
   -0.1649    1.1093   -1.2141
    0.6277   -0.8637   -1.1135


rp_im3_pts(:,:,3) =

   -0.0068    0.3714   -1.0891
    1.5326   -0.2256    0.0326
   -0.7697    1.1174    0.5525
```

CS5320_normals

To verify this function let us take the range image, ht_im. After passing this to CS5320_range2pts, we get ht_im_pts (xyz points image).

Now give this as input to CS5320_normals.

Below figures include the Normal vector values shown at particular locations on certain planes/objects in the image. It was found that the normal values (shown by [UVW]) were correct.

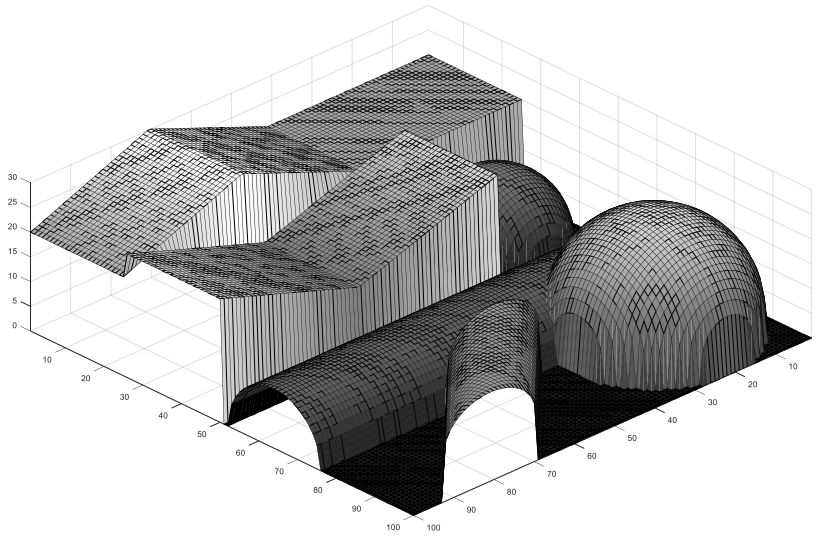


figure: surf(ht_im)

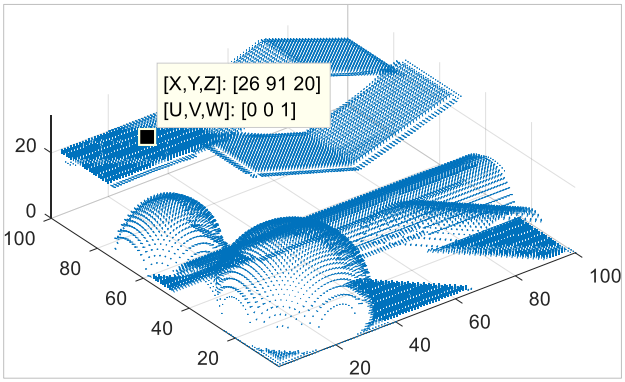


figure: normal on flat plane [0 0 1]

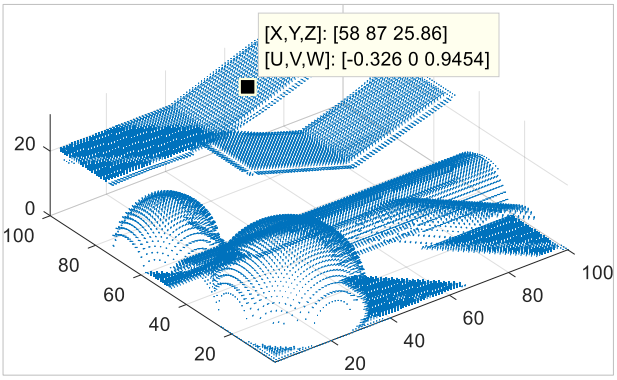


figure: on inclined plane [-0.326 0 0.9456]

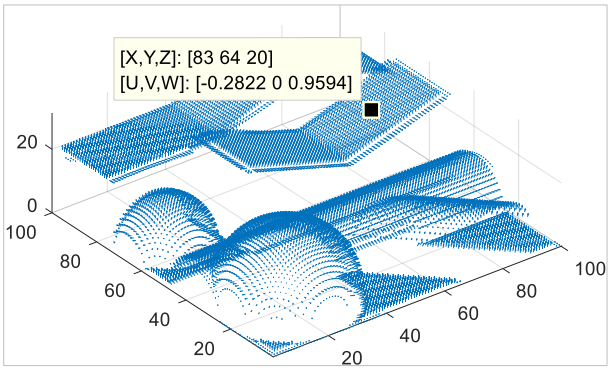


figure: on inclined plane [-0.2822 0 0.9594]

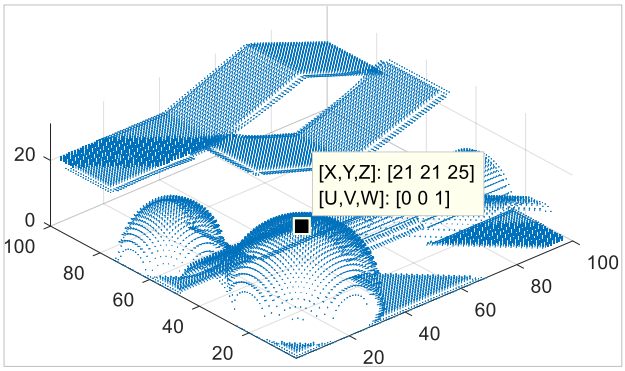


figure: at sphere top center [0 0 1]

CS5320_planes

The plane parameters (a, b, c, d) that were obtained (at every image position) as the output of the CS5320_planes function should satisfy the equation: ax +by + cz +d = 0 at every position in the image.

Hence this can be used as a verification method.

We, can for every location of the image frame, we will take the x, y, a, b, c values and find the z value through the equation. If the function is working properly then we should get the same/almost same values of z that were used as the input to find a, b, c, d.

For this, we will first plot original ht_im_pts (x, y, z) as **green** dots. And then plot the z values obtained through the equation in **red** and see whether the match.

Code:

```
%verification of CS5320_planes
[rows,cols, planes] = size(ht_im_pts);
figure;hold on;
plot3(ht_im_pts(:,:,1),ht_im_pts(:,:,2),ht_im_pts(:,:,3),'g.');
for x = 1:cols
    for y = 1:rows
        rw = rows - y+1;
        cl = x;
        a = im_pl(rw,cl,1);
        b = im_pl(rw,cl,2);
        c = im_pl(rw,cl,3);
        d = im_pl(rw,cl,4);
        z = -(a*x + b*y + d)/c;
        if isnan(z)
            continue;
        end
        plot3(x,y,z,'r.');
    end
end
axis equal;
```

The test were carried out onto 2 different range data sets, and it was seen that the function was working properly.

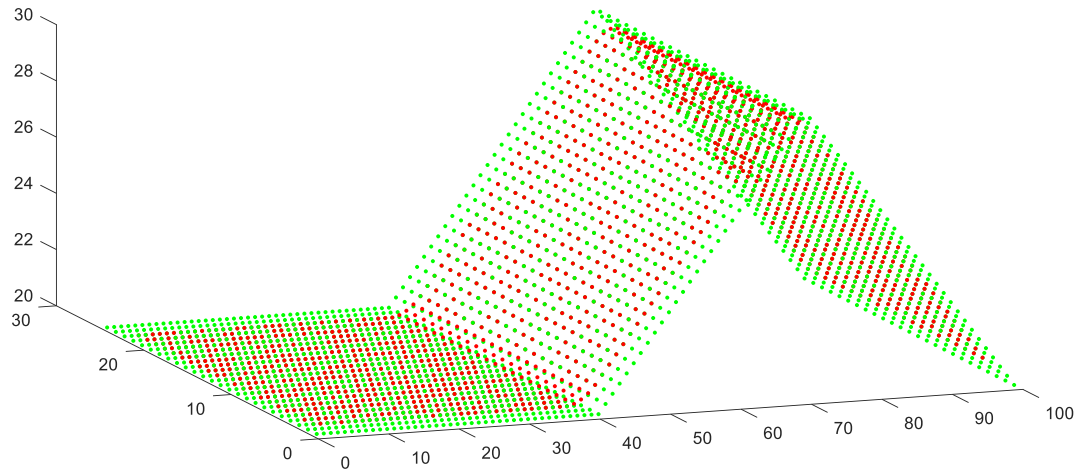


Figure: Verification of CS5320_planes

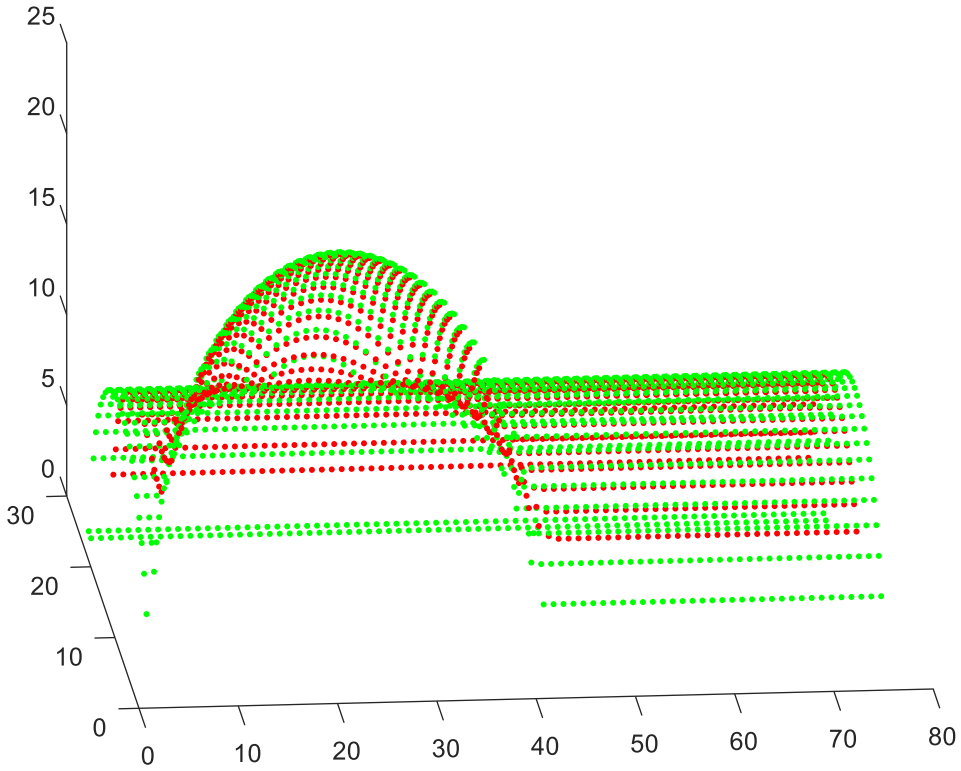


Figure: Verification of CS5320_planes

CS5320_extract_data

To verify, we will run this function with the call: v = CS5320_extract_data (ht_im_pts,51,21,51,27, [1:3])

Here, we have to extract 3 channel data from [r1, c1] = [51,21] to [r2, c2] = [51,27] from ht_im_pts.

The output was as shown below, and was similar to what was required.

```
v =

    21    50     0
    22    50     0
    23    50     0
    24    50     0
    25    50     0
    26    50     0
    27    50     0
```

If the data was to be extracted from the diagonal of a 1 channel image (3x3):

temp_im = round(randn(3)*10);

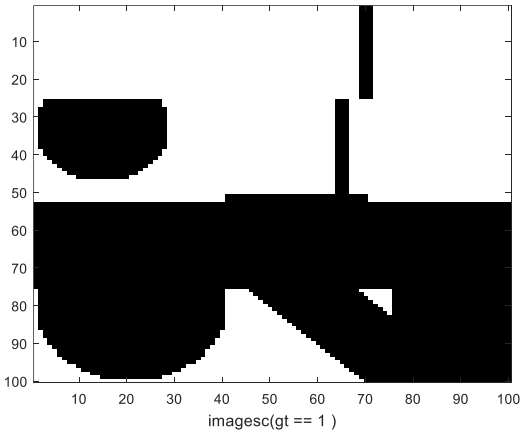
Call: v = CS5320_extract_data(temp_im,1,1,3,3,1);

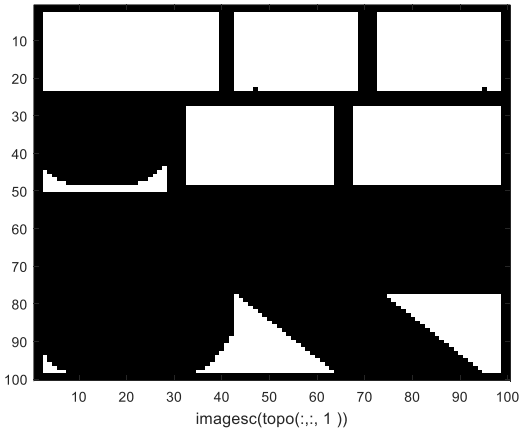
```
temp_im =

     2    -8     2
     0     6    14
     5    13     2
```

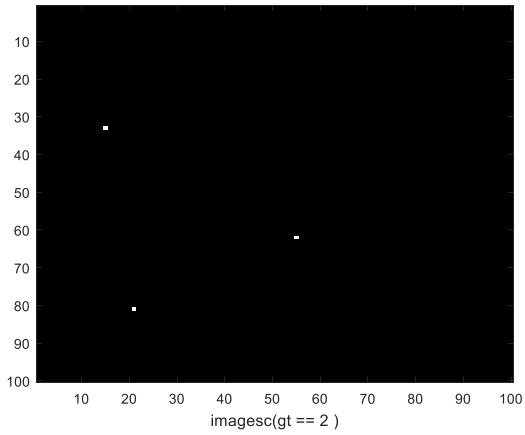
```
v =

     2
     6
     2
```

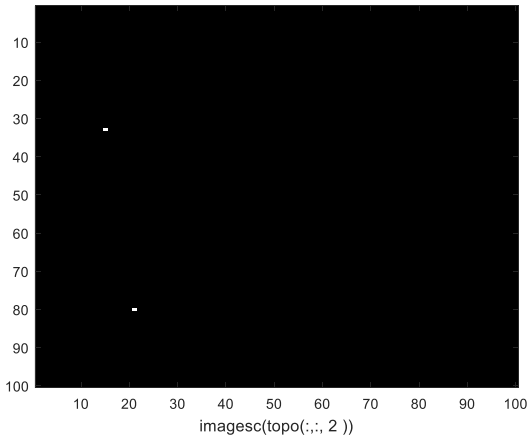
CS5320_topo

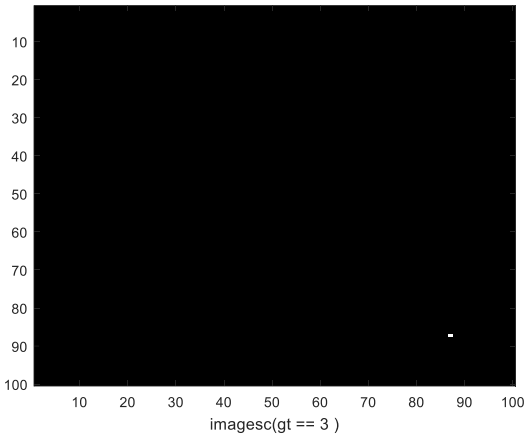
In order to verify the function, the function was run using k =2, where 2k+1 was the window size. To check the correctness of the function, the output of the function was compared with the corresponding ground truth image. It was seen that the output was good, and it seems that it was better in some cases like the RIDGE (4) and RAVINE (5).



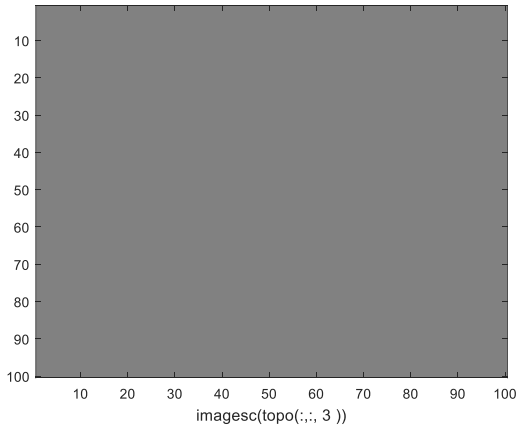




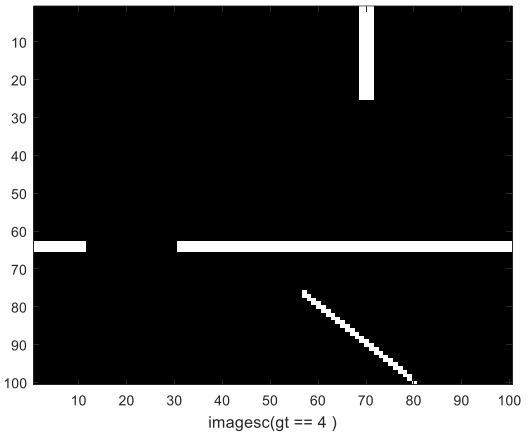




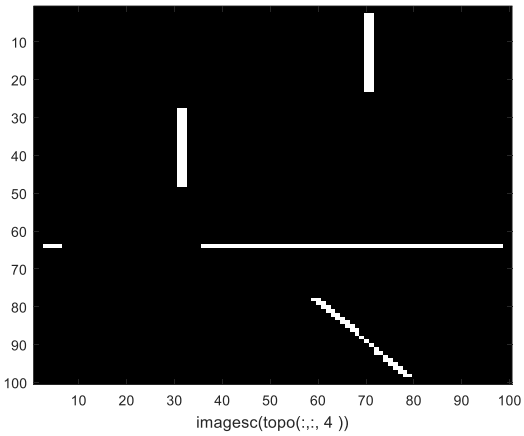
imagesc(gt == 3)



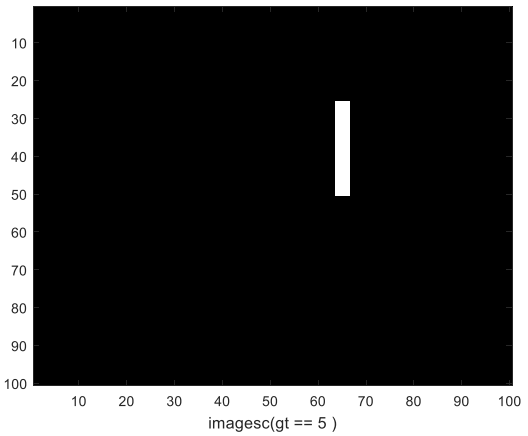
imagesc(topo(:,:, 3))



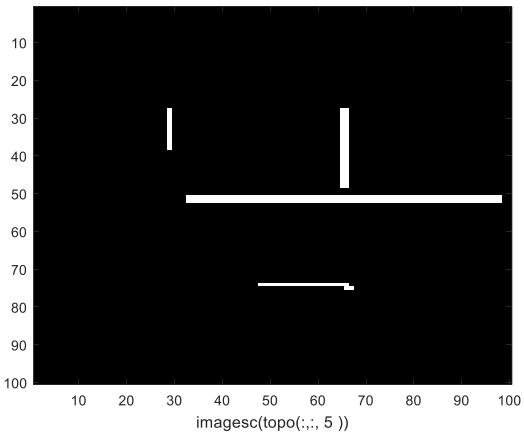
imagesc(gt == 4)



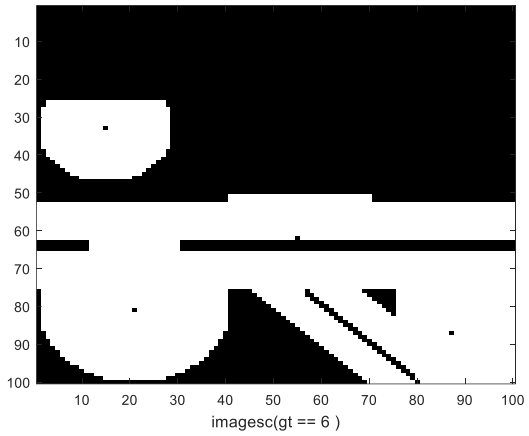
imagesc(topo(:,:, 4))

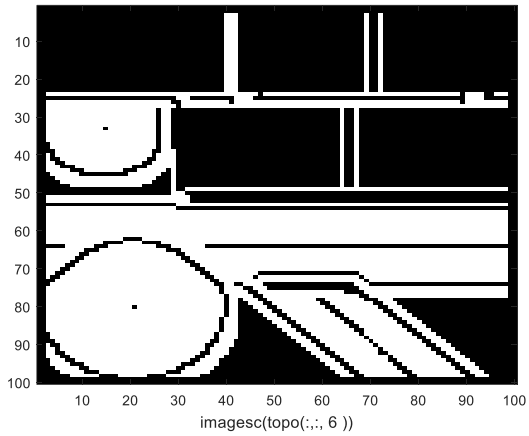


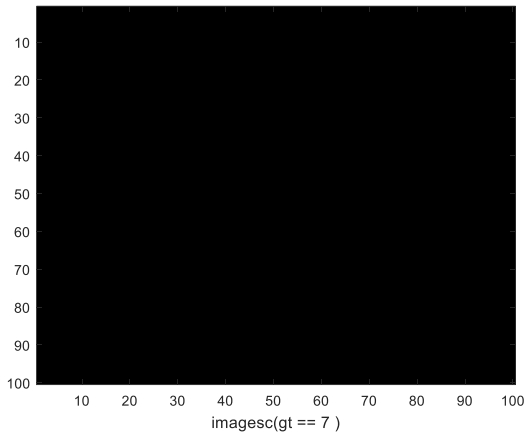
imagesc(gt == 5)

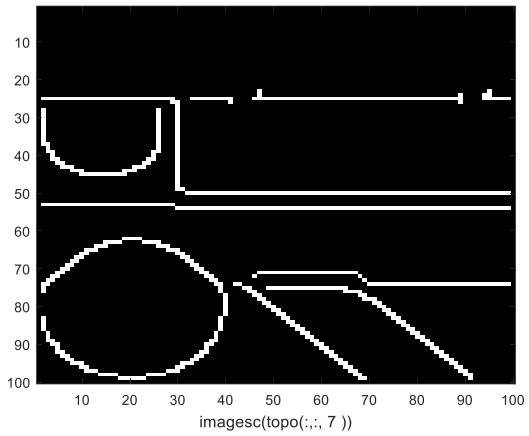


imagesc(topo(:,:, 5))









4. Data

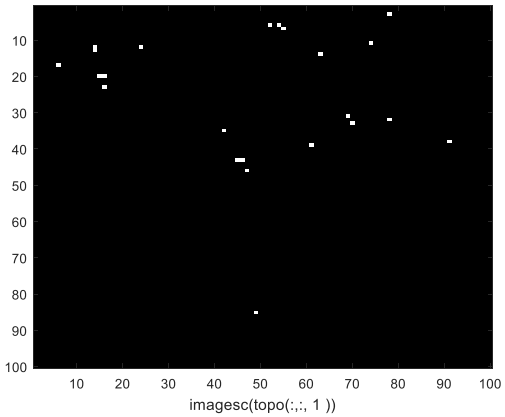
As we need to study the impact of Gaussian noise in the range value on the topo classification. We will add the Gaussian noise to the original range image in the following manner.

im_w_noise = im + randn(size(im))*sigma

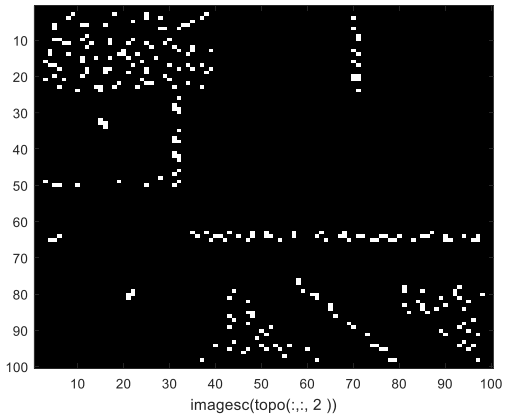
here, sigma is the standard deviation. we will vary sigma and see the output of CS5320_topo.

When the noise was added to the image, it was seen that the performance of the CS5320_topo function was very bad. Except for the JUMP_EDGES (which were found through canny edge detector), it would not detect any of the features correctly from the image even with sigma of 0.00001.

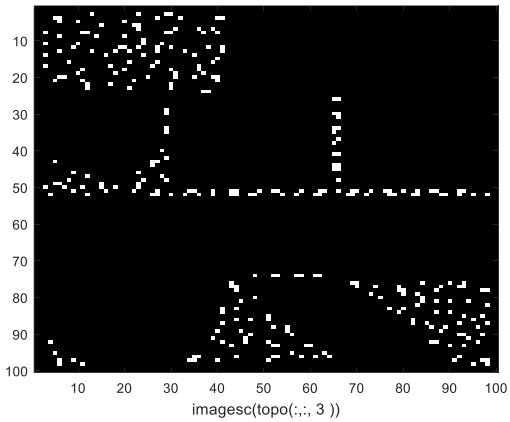
The following outputs were obtained:



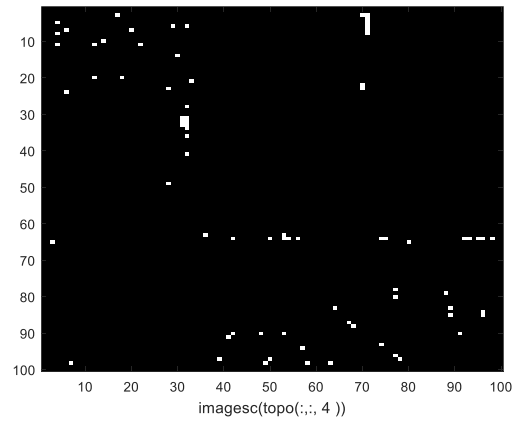
imagesc(topo(:,:, 1))



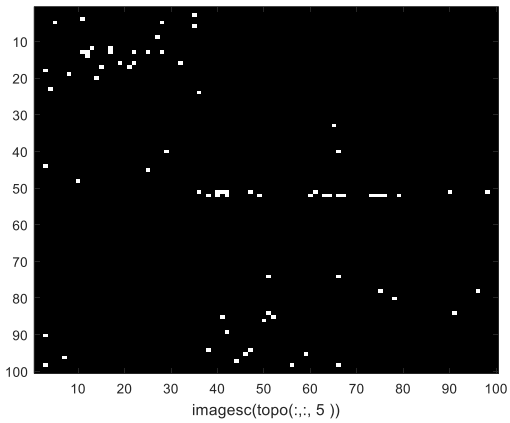
imagesc(topo(:,:, 2))



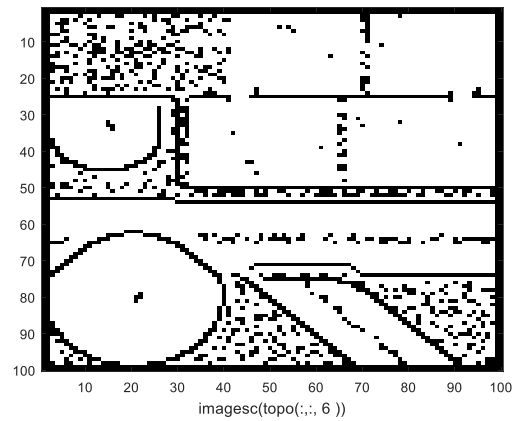
imagesc(topo(:,:, 3))



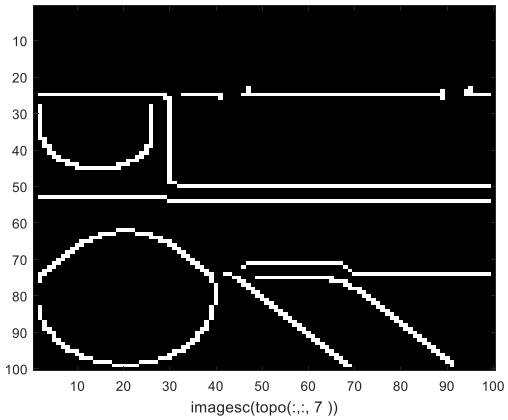
imagesc(topo(:,:, 4))



imagesc(topo(:,:, 5))



imagesc(topo(:,:, 6))



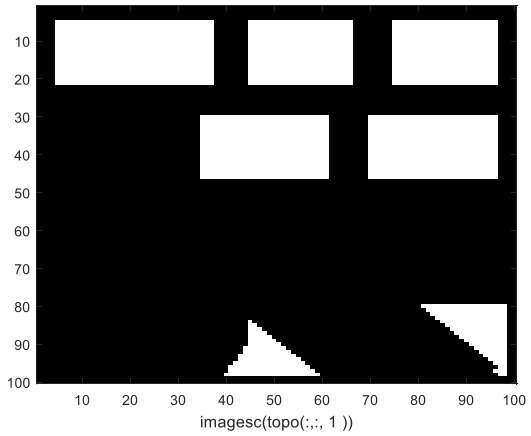
Hence, I thought of changing the CS5320 topo function. Instead of taking the z values to check for the features in the range data, the information from the planes (input to CS5320_topo) was used.

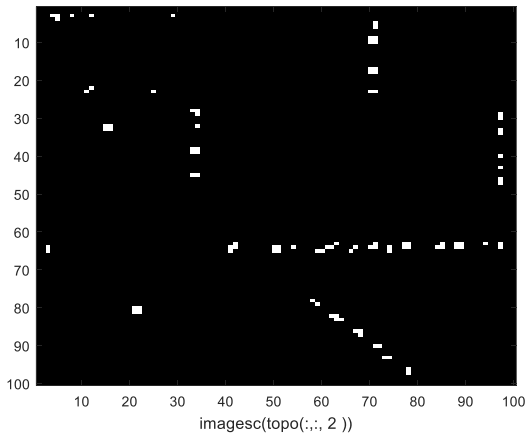
Now, instead of taking the z value from points (input to CS5320_topo) directly, the parameters of planes at each point were used to find the z at a point, using the equation,

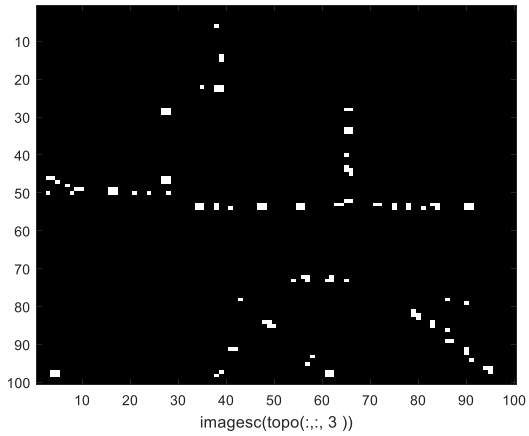
$$z = -(ax + by + d)/c$$

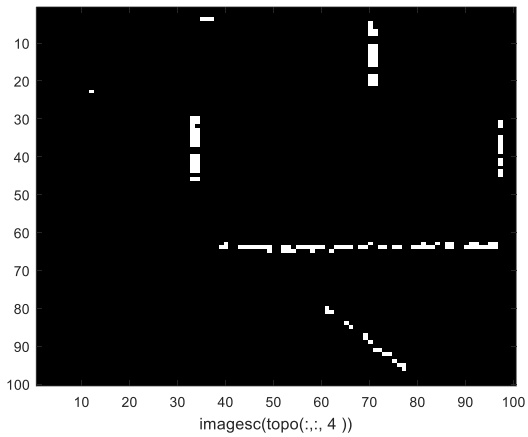
And these z values were used in the function foe classification. The new function developed is saved as 'CS5320_topo_fromfitplanes'.

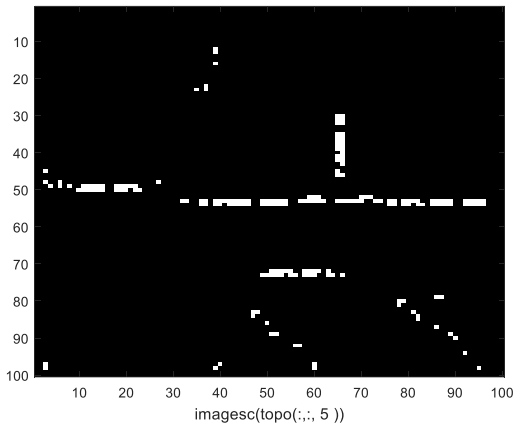
Now, upon testing the function with ht_im, the classification was found to be similar. When a Gaussian noise with sigma = 0.00001 was used, the following output classification was obtained.

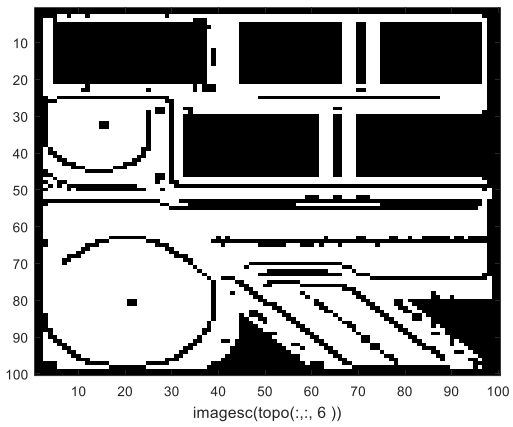


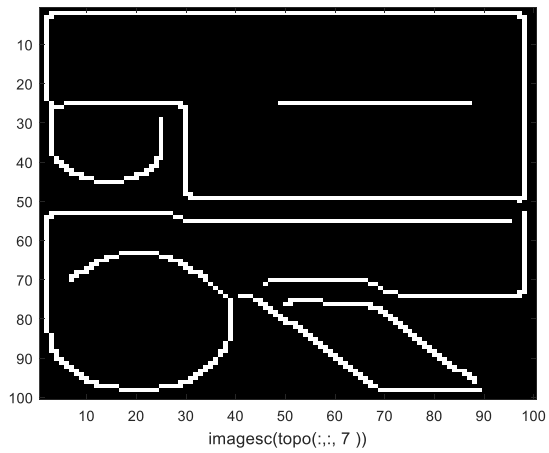






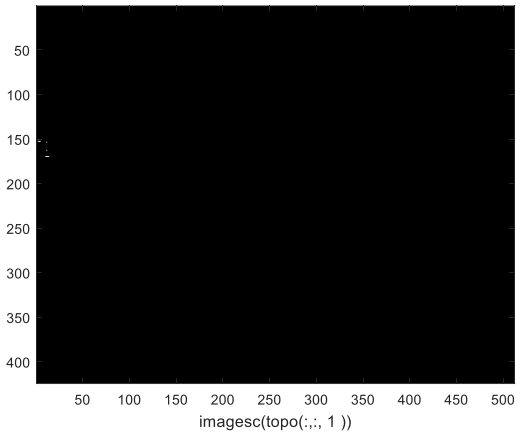




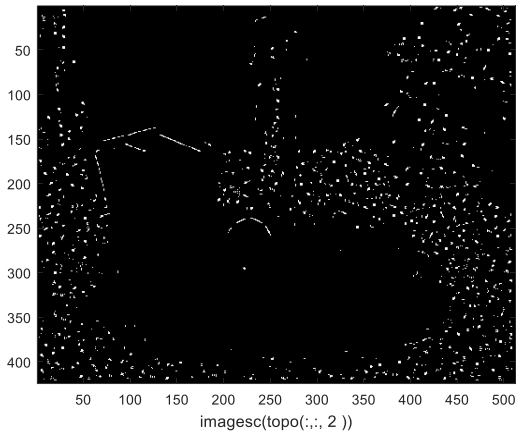


Now, in this case it was seen that the classification was improved, then that found by CS5320_topo. But, on the increase of noise further from sigma = 0.00001 to 0.0001, It was found that the function could not classify properly.

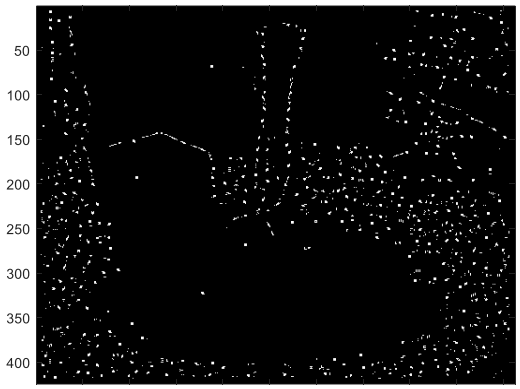
Data: Applying topo function to the actual range image found in scene1.mat



imagesc(topo(:,:, 1))



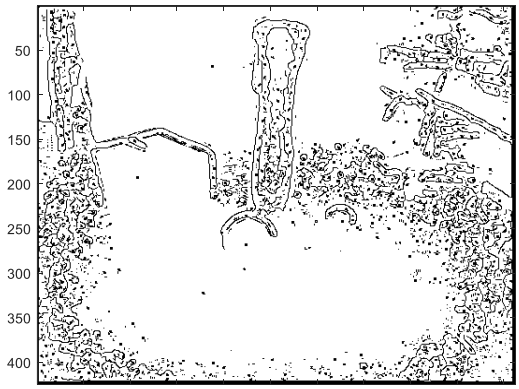
imagesc(topo(:,:, 2))



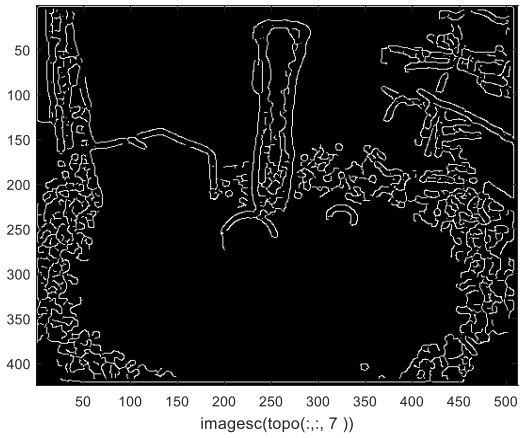
imagesc(topo(:,:, 3))

imagesc(topo(:,:, 4))

imagesc(topo(:,:, 5))



imagesc(topo(:,:, 6))



Analysis

Some analysis, has already been discussed along with data in previous section.

In general, the performance of CS5320_topo is bad due to noise.

5. Interpretation

It is seen that the noise affects the topographical classification. As a new function was developed which fitted planes with a noisy image, the classification was improved. But when the noise was further increased from sigma = 0.00001 to 0.0001, The classification got worse with increase in noise.

I assume that fitting a plane fitting to the noisy image before classification function would help.

6. Critique

I could have developed the CS5320_topo function in some other way, by using the different input data to the function (normals, planes, points). I tried different ways but, the current functions: CS5320_topo and CS5320_topo_fromfitplanes were the only ones that I could come up with.

Also, it is to be noted that my C5320_topo function takes about 6 minutes to work on the scene1.mat data.

To get good results from CS5320_topo, for the noisy range data and for scene1.mat data, I tried to fit planes to the depth images, tried Gaussian smoothening, however I didn't get the required results.

7. Log

Coding: 23 hrs

Report: 4 hrs