EE 569 3D Vision: Assignment 2

Furkan Gül, Student ID: 28795

November 28, 2020

1 Experiments

There exist several methods to extract edge points or edgels in a given input image. These methods can be divided into two categories of 1st derivative and 2nd derivative edge detectors. Prewitt, Roberts, Canny are common types of 1st derivative edge detectors. LoG (Laplacian of Gaussian) is a 2nd derivative edge detector. In this assignment, the objective is to find out which edge detectors perform better in which conditions on images. To observe and compare the performance of 1st derivative and 2nd derivative edge detectors, there will be three images to be processed through those four types of derivative edge detectors. These input images are a ball, a Rubik's cube, and a valve. They can be seen in Figures 1, 2, and 3 respectively.

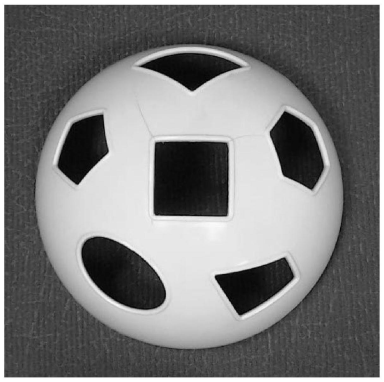


Figure 1: Ball Image

This first part of the report is an introductory step to understand a script, and a function implemented for assignment 2 deeply.

Before diving into the results of using different edge detectors, it will be better to mention the main script that calls all the required functions benefited in the assignment. It is named "assign2", which can be found in the Appendix. Because there exist fundamentally three input images that will be addressed in the assignment, the main script is divided into three sections by starting the code line with two comment characters (%%). Each section has one

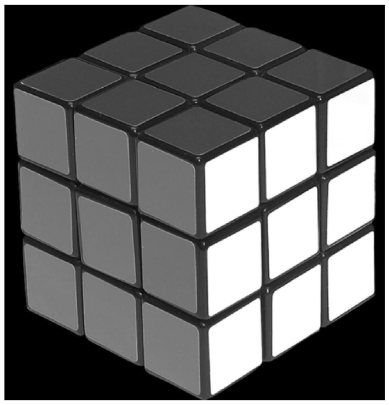


Figure 2: Rubics Cube Image

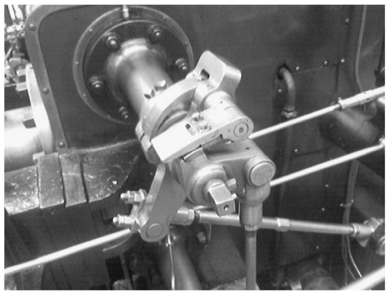
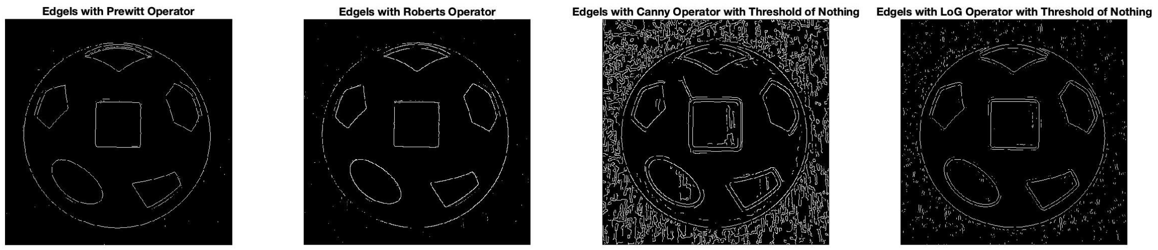


Figure 3: Valve Image


Edgels with Prewitt Operator
Edgels with Roberts Operator
Edgels with Canny Operator with Threshold of Nothing
Edgels with LoG Operator with Threshold of Nothing
Figure 4: Edgels in Ball Image without Thresholds

function in common for all three images. Its name is ”edgeDetectComp” which falls into the Appendix. At the beginning of this function, there are a couple of lines of codes to make the function generic. It starts by checking the input images if it is RGB or grayscale. If it is RGB, the function will convert it into a grayscale format. To smoothen the input image and eliminate noise in the input image, imgaussfilt Matlab built-in function is used with a sigma of 1.5 for just Canny and LoG edge operators since those two operators are more sensitive to noise. Then, using Matlab’s built-in edge function, four edge detector operators are easily applied to the corresponding input image with some thresholds for just both Canny and LoG detectors to find ”strong edgels”. If one does not want to specify any thresholds, then those two thresholds must be set to 0. Lastly, the four edge images obtained with four different edge detectors will be displayed in the Matlab figure. The results of this process for the ball image without giving any threshold values can be seen in Figure 4.

2 Results & Discussions

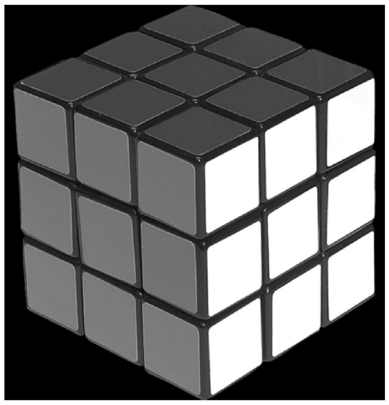


Figure 5: Rubics Cube Input Image

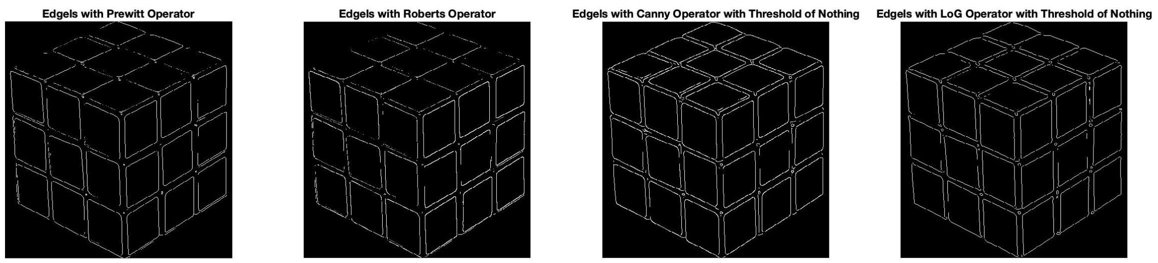

Edgels with Prewitt Operator
Edgels with Roberts Operator
Edgels with Canny Operator with Threshold of Nothing
Edgels with LoG Operator with Threshold of Nothing

Figure 6: Edgels in Rubics Cube Images with Four Different Derivative Edge Detectors without any Thresholds

To get a better understanding of how accurately Prewitt, Roberts, Canny and Laplacian of Gaussian(LoG) derivative filters will extract edge pixels from a given input image, three different input images found online will be processed with those four derivative filters with different hyperparameters like thresholds. These input images are Rubik's cube, a ball, and a valve.

Rubik's cube (look at Figure 5) is given as an input image to "edgeDetectComp" function without any thresholds for detecting edge pixels. The resulting edge images for each of those four derivative operators can be seen in Figure 6. In Figure 6, since there is not much noise in Canny and LoG edge images, no need to apply thresholds to Canny and LoG derivative edge operators. So, just give T1 and T2 threshold values as 0 so that there will be no thresholding. When those Prewitt and Roberts Edge images on the left side of Figure 6 are examined, it is clear that the edges in Roberts Edge image are thicker than the edges in Prewitt Edge image. It is also obvious that the discontinuities in edges in Prewitt Edge image occur many times compared to Roberts Edge image. Those two deductions arise from the fact that Roberts derivative filter is giving equal weights to all pixels in a window not giving zeros to central pixels in a mask as it is in Prewitt operator. By doing so, in contrast to Prewitt derivative filters, Roberts filters can capture more edge elements with fewer discontinuities in edges, circles, or any feature you want to extract from images. On the other hand, when Canny and LoG edge images are considered, LoG edge image is more accurate at localizing edges but Canny is better at linking the edge elements thanks to non-maximum suppression, linking, and thresholding steps in Canny edge detector. Thanks to better linking of Canny and higher noise sensitivity of LoG operator, discontinuities in Canny edge image occurs less than that of LoG edge image. But it can be said that there are almost no discontinuities in Canny and LoG edge images compared to many discontinuities in Prewitt and Robert edge images. By looking at Figure 8, it can be observed that those deductions are also valid for the valve image (see Fig. 7). Note that since there is also not much noise in Canny and LoG edge images of valve image, there is no need to give threshold values to the function.

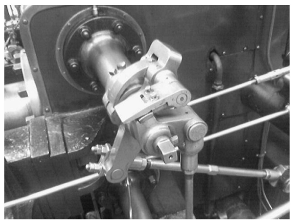


Figure 7: Valve Input Image

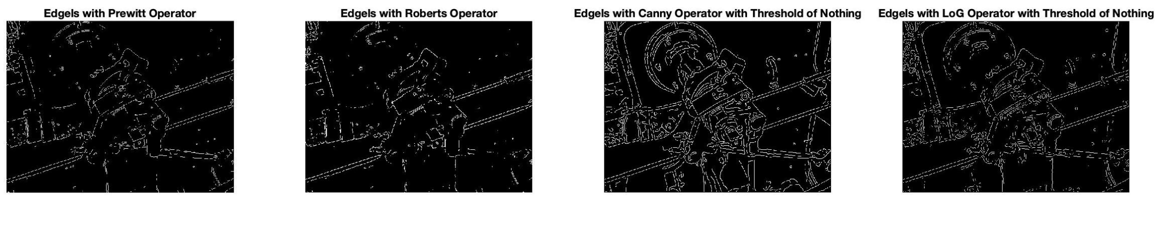

Edgels with Prewitt Operator
Edgels with Roberts Operator
Edgels with Canny Operator with Threshold of Nothing
Edgels with LoG Operator with Threshold of Nothing

Figure 8: Edgels in Rubics Cube Image with Four Different Derivative Edge Detectors without any Thresholds

Let's try the same comparison with another input image, which is a kind of ball image (see Fig.9). Since there seems much noise at the outside of the ball shape in the original

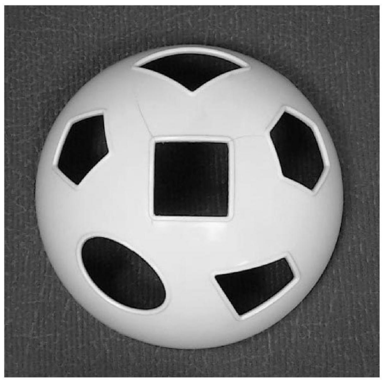


Figure 9: Ball Input Image

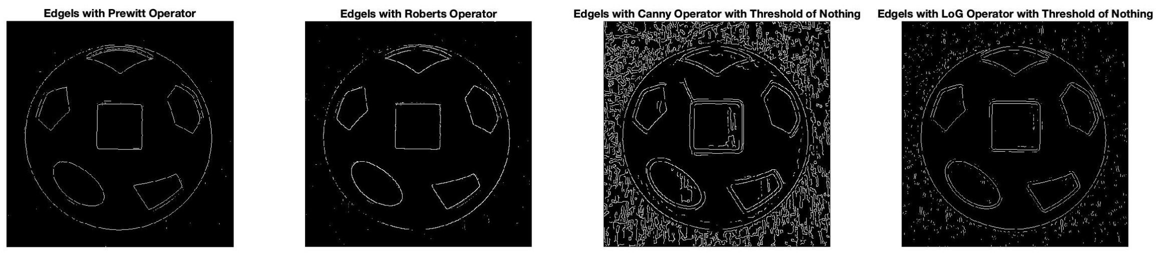

Edgels with Prewitt Operator
Edgels with Roberts Operator
Edgels with Canny Operator with Threshold of Nothing
Edgels with LoG Operator with Threshold of Nothing

Figure 10: Edgels in Ball Image with Four Different Derivative Edge Detectors without any Thresholds

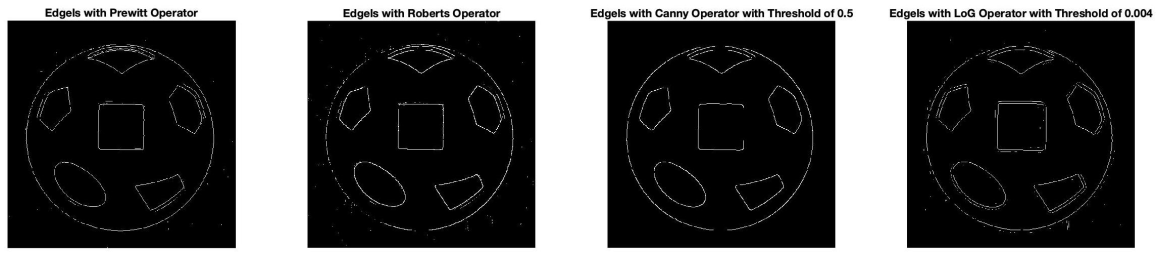

Edgels with Prewitt Operator
Edgels with Roberts Operator
Edgels with Canny Operator with Threshold of 0.5
Edgels with LoG Operator with Threshold of 0.004

Figure 11: Edgels in Ball Image with Four Different Derivative Edge Detectors with Thresholds for Canny and LoG

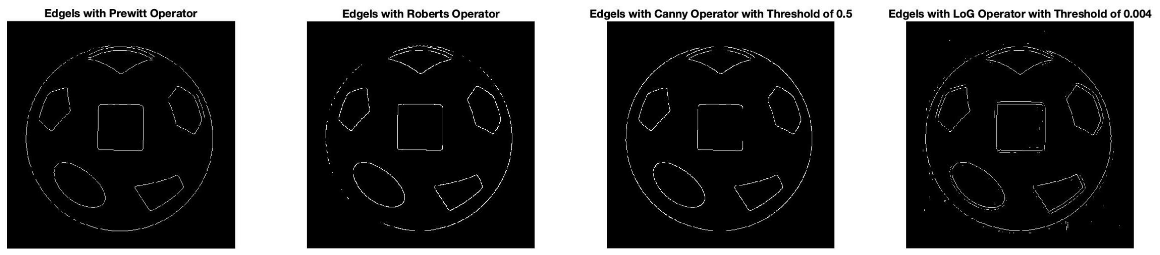

Edgels with Prewitt Operator
Edgels with Roberts Operator
Edgels with Canny Operator with Threshold of 0.5
Edgels with LoG Operator with Threshold of 0.004

Figure 12: Edgels in Ball Image with Four Different Derivative Edge Detectors with Thresholds for Canny and LoG & Gaussian Filtering for Prewitt and Roberts too

ball input image in Figure 9, there can be noise in edge images this time. Let's examine what the result of four edge images of the ball image is without any thresholds by looking at Figure 10. In Figure 10, there is a lot of noise in Canny and LoG edge images. I also expect much more noise in Prewitt and Roberts edge images, but there is not much. The reason for this may arise from the fact that the built-in Edge function in Matlab automatically adjusts the threshold value for those two Prewitt and Roberts operators to avoid noise as much as possible. That's why it makes sense to apply threshold values for detecting only "strong edge elements" in the case of Canny and LoG operators. By making some manual optimization over two thresholds for those Canny and LoG operators to remove all unnecessary noises from edge images, I ended up with 0.5 and 0.004 respectively. Threshold values are so small since the built-in edge function takes the threshold as a normalized value between 0 and 1. Thresholded Canny and LoG edge images can be seen in Figure 11. Furthermore, to remove all noises in Prewitt and Roberts edge images in Figure 11, the built-in imgaussfilt function will be applied to the input ball image for all those four operators, not just Canny and LoG as it is the case earlier. The resulting noise-clean four edge images can be seen in Figure 12. It seems that the best edge images come from using Prewitt and Canny edge detector. But when you consider better linking as a performance metric, it would be wise to choose Canny edge image as the best as it is also seen in Figure 12. But the threshold for Canny can be optimized further to eliminate remaining discontinuities in Canny edge image.

A Appendix

All the below codes and their corresponding outputs were obtained as latex output format without any manual intervention by using the "Publish" Method in Matlab.

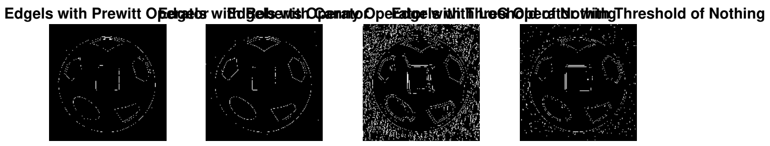
assign2 Script

Contents of assign2 Script

Ball Image

```
clc;clear;close;
% Let's get Monastry image
img = imread("ball.jpg");
imshow(rgb2gray(img))
% Set T1 and T2 to 0 in order to not have any threshold for edgels
```

```
T1 = 0;
T2 = 0;
edgeDetectComp(img, T1, T2);
% Optimized thresholds for edgels in Canny and LoG respectively
% T1 = 0.5;
% T2 = 0.004;
% edgeDetectComp(img, T1, T2);
```

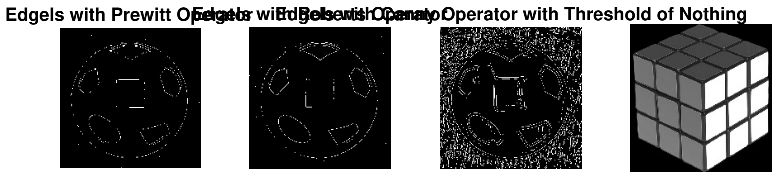

Edgels with Prewitt OpEdgels witEdgels with CannyOpeEdgels with LoG Operator with Threshold of Nothing

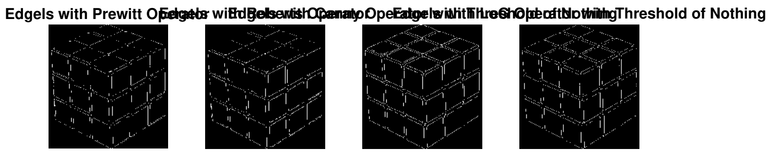
Rubics Cube Image

Let’s get Rubics Cube image

```
img = imread("rubicsCube.png");
imshow(rgb2gray(img))
% Set T1 and T2 to 0 in order to not have any threshold for edgels
T1 = 0;
T2 = 0;
edgeDetectComp(img, T1, T2);
```

```
% Optimized thresholds for edgels in Canny and LoG respectively
% T1 = 0.5;
% T2 = 0.004;
% edgeDetectComp(img, T1, T2);
```

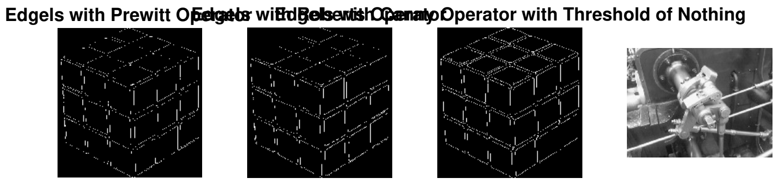


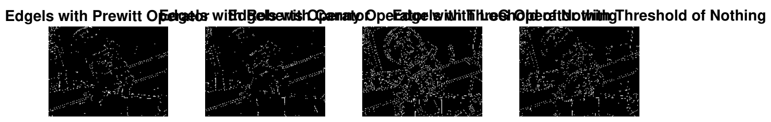


Valve Image

Let's get Valve image

```
img = imread("Valve_original.PNG");
imshow(rgb2gray(img))
% Set T1 and T2 to 0 in order to not have any threshold for edgels
T1 = 0;
T2 = 0;
edgeDetectComp(img, T1, T2);
% Optimized thresholds for edgels in Canny and LoG respectively
% T1 = 0.5;
% T2 = 0.004;
% edgeDetectComp(img, T1, T2);
```





edgeDetectComp Function

```
function [] = edgeDetectComp(img, T1, T2)
% Chech the original image if it is RGB or monocolor
```

```
[~, ~, c] = size(img);

% If the image is RGB (colored), then convert it to grayscale image
if c == 3
    img = rgb2gray(img); % It is overwritting. Not good in general.
end

img = imgaussfilt(img, 1.5);

prewitt = edge(img, "prewitt");
roberts = edge(img, "roberts");

% For just ball image
%img = imgaussfilt(img, 1.5);

if T1 == 0 && T2 == 0
    canny = edge(img, "canny");
    log = edge(img, "log");
    T1 = "Nothing";
    T2 = T1;
else
    canny = edge(img, "canny", T1);
    log = edge(img, "log", T2);
end


figure()
subplot(1,4,1)
imshow(prewitt)
title("Edgels with Prewitt Operator")

subplot(1,4,2)
imshow(roberts)
title("Edgels with Roberts Operator")

subplot(1,4,3)
imshow(canny)
title("Edgels with Canny Operator with Threshold of " + num2str(T1))

subplot(1,4,4)
imshow(log)
title("Edgels with LoG Operator with Threshold of " + num2str(T2))

end
```