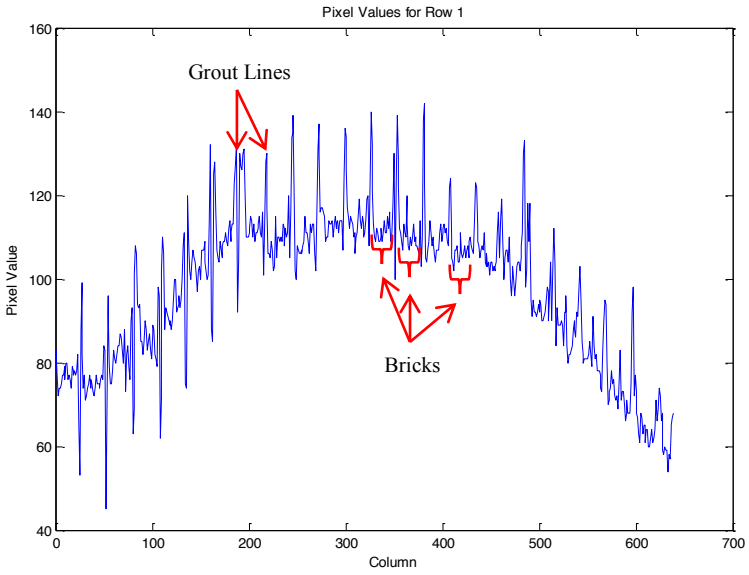
Lab 4: Edge Detection in Images

A. Background Information

Edge detection is a common, but very important method for analyzing images. It can be used to help identify objects in a scene or features of an object. It is widely used in biometrics security applications to identify facial features or fingerprints. The example below shows the process of edge detection applied to a picture of a bike next to a brick wall. As can be seen, edge detection easily shows the outline of the bike, the bricks in the wall, and even the writing on the bike.

Edge Detection Example: Original Image (left) and Edge Detected Image (right)

So how does edge detection work? The first step in understanding the process is to understand what an edge is in an image. If you were to look at the values of in an image along a single row or column, you would see something that looks like this:



This is actually the graph of the pixel values from the first row of the original picture shown above. What you are seeing here are the individual dark colored bricks that make up the top border of the image. The bricks themselves are the lower and relatively flat sections, while the lighter colored grout between the bricks is represented by the sharp peaks. With this

understanding, we can see that an edge (a sharp change in color) is represented by a large change in the pixel value.

In order to locate the areas that have these drastic changes in color, we need to apply a mathematical operation to the image. In this case, since we want to know the change in the color values, we will be using a derivative, since a derivative is large when there is a drastic change (an edge) and small when there is little change (not an edge).

So far, we've only been applying derivatives to 1-dimensional data using the 2 point and 3 point derivative estimates. So how do we apply a derivative to a 2-dimensional image? The process is essentially the same. However, in this case, we will be applying the 3 point derivative estimate to three rows (or columns) and adding the results together to determine whether there is an edge at the location in the image. The next section describes the process by which we will apply the derivative to an image.

B. Understand the Process

The process we will use to perform edge detection is what is called 2-D filtering, but in essence, we will be applying the 3 point derivative estimate over a 3x3 section of our image to check and see if there is an edge present. The filter that we will be using is called a Prewitt Filter, and looks like this:

$$Prewitt_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \quad Prewitt_y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

The filter is applied as follows:

1) Access a 3x3 portion of the image
2) Multiply each value in the filter by the corresponding value from the image
3) Add up the results of the multiplications and take the absolute value (this is done because all image values are positive, but we want to find any large change, be it from low to high or high to low pixel values)
4) Store the resulting value into a new image at the location of the center of the portion from the original image

Assume we have the image portion shown below, which was taken from rows 4-6 and columns 3-5 of the full image. If we were to look at what is happening mathematically when applying the $Prewitt_x$ filter, we would calculate a value for NewImage(5,4) as follows:

$$image(4{:}6,3{:}5) = \begin{bmatrix} 50 & 75 & 150 \\ 52 & 74 & 154 \\ 51 & 76 & 149 \end{bmatrix}$$

$$P = \begin{bmatrix} 50 & 75 & 150 \\ 52 & 74 & 154 \\ 51 & 76 & 149 \end{bmatrix} .* \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} -50 & 0 & 150 \\ -52 & 0 & 154 \\ -51 & 0 & 149 \end{bmatrix}$$

$$NewImage(5,4) = |Sum\ of\ Values\ in\ P| = |(150 - 50) + (154 - 52) + (149 - 51)|$$

This is the same thing as the sum of three 3 point derivative estimates (with $2\Delta t = 1$). To apply this filter to the whole image, we simply pull out a different part of the image and apply the filter, storing the result into the new image. The one thing to note is that the filter cannot be applied to border of the image, since you cannot pull out a 3x3 matrix centered along the border. Typically, the border of an edge-detected image is simply left black (value = 0).
Assume that we have an image as shown below. Complete the table by applying the edge detection algorithm described above, using the Prewitt_x filter and the Prewitt_y filter.

Original Image:

0	0	255	0	0
0	0	255	0	0
0	0	255	0	0
0	0	255	0	0
0	0	255	0	0

Prewitt_x:

0	**0**	**0**	**0**	**0**
0	**765**	**0**	**765**	**0**
0	**765**	**0**	**765**	**0**
0	**765**	**0**	**765**	**0**
0	**0**	**0**	**0**	**0**

Prewitt_y:

0	**0**	**0**	**0**	**0**
0	**0**	**0**	**0**	**0**
0	**0**	**0**	**0**	**0**
0	**0**	**0**	**0**	**0**
0	**0**	**0**	**0**	**0**

What do you notice about the values when you use the Prewittx and Prewitty filters? What effect do you think this will have when you apply this to an actual image?

When the Prewitt_x is applied, the vertical line is detected, whereas with the Prewitt_y, the line is not detected. This means that when applied to an image, the filters will only find vertical (x) or horizontal (y) lines.

C. Implementation of Edge Detection

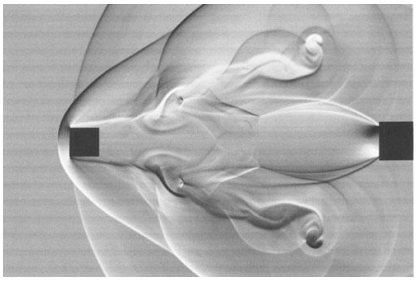
Write a script that implements edge detection using the Prewitt_x filter. Your script should follow the procedure outlined below:

1. Load an image into MATLAB using the `imread` command.
2. Check to see whether the image is a color image or a grayscale image by checking the number of planes along the third dimension (i.e. 1 plane is grayscale, 3 planes is color).
3. If the image is a color image, convert it to grayscale using the conversion shown below where X is the original color image and Pic is the resulting grayscale image.

```
Pic = 0.299*X(:,:,1) + 0.587*X(:,:,2) + 0.114*X(:,:,3);
```

4. Display the grayscale image using the `imshow` command.
5. Convert Pic to a double.
6. Create a new array filled with zeros that is the same size as your image.
7. Apply the Prewitt$_x$ filter to the image as follows:
    a. Pull out a section of the original image.
    b. Multiply each value in the section of the image by the corresponding values in the filter.
    c. Add all of the values together.
    d. Take the absolute value of the final value.
    e. Store the final value into your new image array at the center of the section you just filtered.
    f. Repeat for all possible sections of the original image (remember, you cannot do this for the border values!).
8. Scale the new image so that the minimum value is 0 and the maximum value is 255.
9. Display your edge-detected image using the `imshow` command. ***You will first need to convert the image back into uint8 format using the `uint8` command.***

Once you have your script written and working, select one of the two provided images. The Arm_Fracture.jpg image is an x-ray image of a broken arm. As you can see, this person has a transverse fracture of the radius. It is fairly easy to identify where the fracture is, but it would be very difficult for a computer to identify this fracture in an automated way. Applying edge detection will help to show the outline of the bones and make it easier for a computer to automatically locate the fracture, making the job easier for the physician.



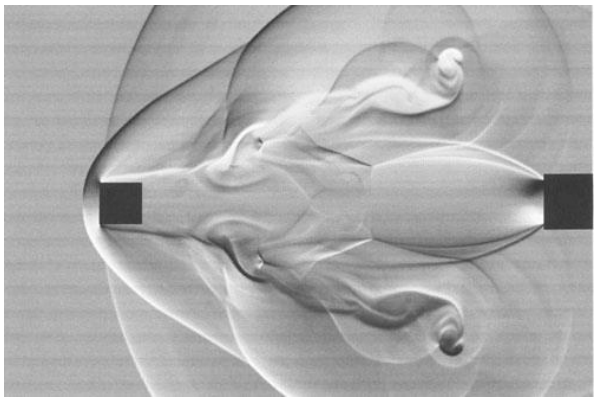
The Schockwave.jpg image shows the pattern produced by an object moving at a high rate of speed through a medium. As you can see from the image, there is quite a complicated pattern produced as the object moves, but there is also some noise in the picture (horizontal repeating lines) that might make it difficult to analyze. Again, by applying edge detection, we can emphasize the pattern produced by the object as it travels through the medium and remove some of the unwanted noise.

Run your script using your selected image and paste a copy of your results below:

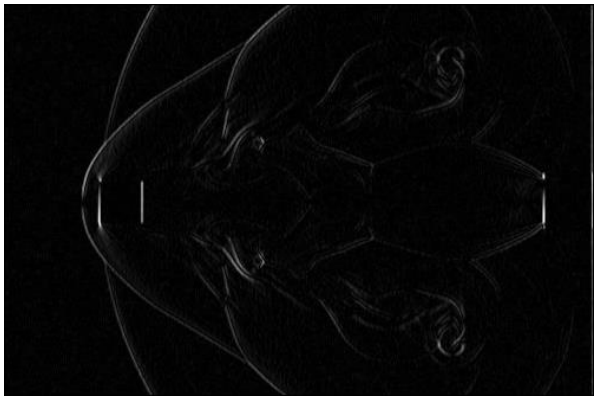
Original Image: Arm Fracture

Edge Detection with Prewitt$_x$: Arm Fracture

Original Image: Shockwave



Edge Detection with Prewitt$_x$: Shockwave



Paste your script file here:

```
Pic = imread('Shockwave.jpg');
[rows, cols, planes] = size(Pic);
% if the image is a color image, convert to grayscale
if planes > 1
    Pic = 0.299*Pic(:,:,1) + 0.587*Pic(:,:,2) + 0.114*Pic(:,:,3);
end
% display original image using imshow:

% Convert Pic to a double:

% Create a matrix of zeros same size as Pic to hold new image:

% Apply Prewitt filter:
Fx = [-1 0 1;-1 0 1;-1 0 1];
```

```
% Need a double for loop here - careful not to walk off end of the image
for r =
      for c =
 Filtered_Image(r,c) = abs(sum(sum(Fx.*Pic(?:?,?:?))));


% Scale Filtered Image so it ranges from 0 to 255

% Convert the Filtered image back to uint8

% Display final edge-detected image using imshow
```

Have students refer back to the calculations at the bottom of page 2 to figure out how to set this up!

Looking at the image you produced, which types of edges were you able to find?

The edges that were predominantly vertical were detected by the Prewitt$_x$ filter.

What do you think you should do in order to find other edges in your image?

Apply filters that detect edges in different orientations from the Prewitt$_x$, such as the Prewitt$_y$.

D. Improved Edge Detection

Modify your script file so that it applies three more filters to the image and adds the results together as follows:

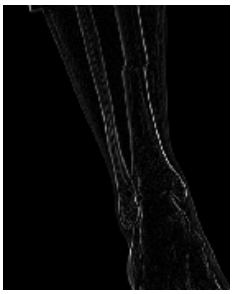
1. Add code to create three additional arrays filled with zeros that are the same size as your original image.
2. Add code to apply the Prewitt$_y$ filter shown previously and two diagonal Prewitt filters, shown below:

$$Prewitt_{d1} = \begin{bmatrix} -1 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} \quad Prewitt_{d2} = \begin{bmatrix} 0 & 1 & 1 \\ -1 & 0 & 1 \\ -1 & -1 & 0 \end{bmatrix}$$

3. Add code to scale the three new images so that the minimum value in each image is 0 and the maximum value is 255.
4. Add code to display your new edge-detected images using the `imshow` command. You can either display all four images in separate figures or use subplot to display them all in one window. ***Don't forget to convert to uint8 first.***
5. Add code to add each of the four edge-detected images together.
6. Add code to scale the summed image so that it again within the range of 0 to 255.
7. Add code to display the summed image using the `imshow` command.

Run your new script using your selected image and paste a copy of your results for the four individual and the final summed edge-detected images below:

Edge Detection with Prewitt X: Arm Fracture



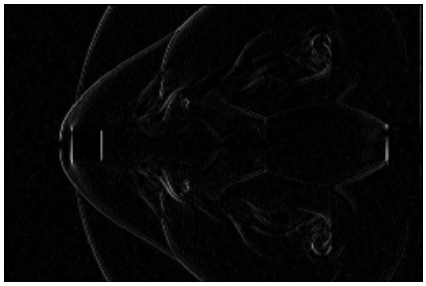
Edge Detection with Prewitt YArm Fracture

Edge Detection with Prewitt Diagonal DownArm Fracture

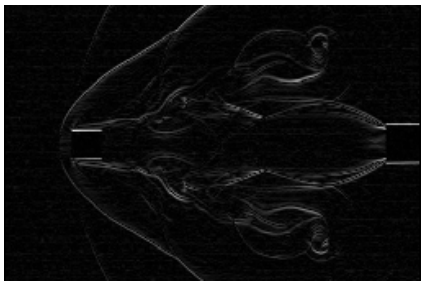
Edge Detection with Prewitt Diagonal UpArm Fracture

Edge Detection with Combined Prewitt:Arm Fracture

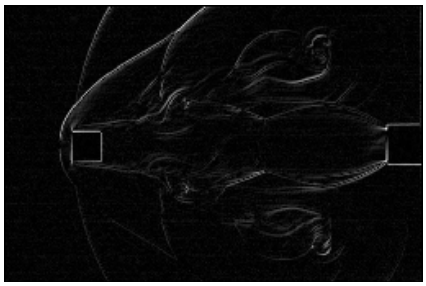
Edge Detection with Prewitt X: Shockwave



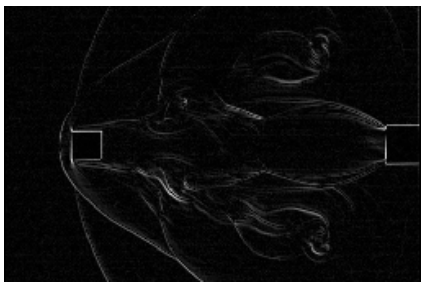
Edge Detection with Prewitt YShockwave



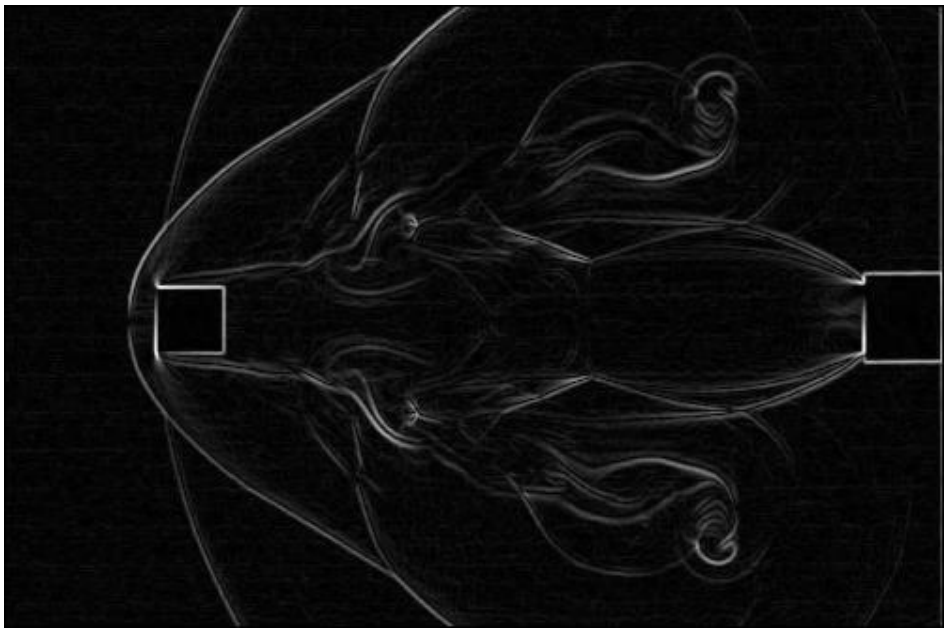
Edge Detection with Prewitt Diagonal DownShockwave



Edge Detection with Prewitt Diagonal UpShockwave



Edge Detection with Combined Prewitt:Shockwave



Paste your improved script file here:

Looking at the results of the four individual Prewitt filters, what do you notice about the types of edges found? What is it about these filters that allow them to find these types of edges?

Each filter finds a different type of edge, where the type of edge found is perpendicular to the gradient of the filter.

How does the summed image compare to the individual images?

The summed image contains all the detected edges.

If you chose the Arm_Fracture.jpg image and you are designing a computer system to identify the fracture automatically, what procedure might you use now that you have the edge-detected image?

OR

If you chose the Shockwave.jpg image, now that you have the edge-detected image, what features might you want to investigate if you are trying to analyze the pattern produced?

Arm Fracture Response:
Look for discontinuities or shifts in a long and relatively straight line. This would indicate a break in the bone, causing a break in the line. ***Other similar type responses are also appropriate.***

Shockwave Response:
Look at the patterns in front of and behind the moving object to determine how it will react when moving in the atmosphere. ***Other similar type responses are also appropriate.***

E. For Fun!

As was mentioned in the introduction, one of the applications for edge detection is to identify facial features for the purposes of biometric security and facial recognition. Take a selfie and run the image through your edge detection code.

Paste a copy of your original (grayscale) and edge-detected selfie below:

Original Image: Selfie

Edge Detection with Combined Prewitt:Selfie

F. To be turned in:

- You will need to upload this word document with all requested tables, questions, and figures included and the m-file for your final script.