Bone Fracture Detection System

Introduction:

Detecting bone fractures is critical for timely and effective medical treatment. Traditional methods often rely on X-rays interpreted by radiologists, which can lead to potential delays and subjective errors. However, advancements in imaging technology and digital image processing have enabled the development of automated and accurate bone fracture detection systems. This report outlines the development of a bone fracture detection system using MATLAB, leveraging various filters and image processing techniques to enhance, segment, and analyze bone images.

Objectives:

The objectives of this project are:

1. **Image Preprocessing** : Enhance bone images by removing noise and irrelevant structures.

2. **Edge Detection** : Accurately identify and emphasize edges in the bone images.

3. **Feature Extraction** : Extract relevant features to characterize fractures.

4. **Segmentation :** Separate bone regions and identify fractures in bone images.

5. **Performance Evaluation** : Evaluate the accuracy and effectiveness of the detection system.

Purpose:

The main purposes of this project are:

1. **Improved Patient Outcomes**: Enable timely treatment and better patient outcomes through early fracture detection.

2. **Diagnostic Support for Medical Professionals:** Provide tools to assist radiologists in interpreting bone images accurately.

3. **Automation and Efficiency:** Develop an automated system to reduce workload and improve diagnosis efficiency.

4. **Research and Advancements:** Contribute to research in medical imaging and computer-aided diagnosis.

5. **Accessible Healthcare** : Develop cost-effective detection systems for areas with limited medical expertise.

6. **Training and Education :** Provide opportunities for researchers and students to gain experience in medical imaging.

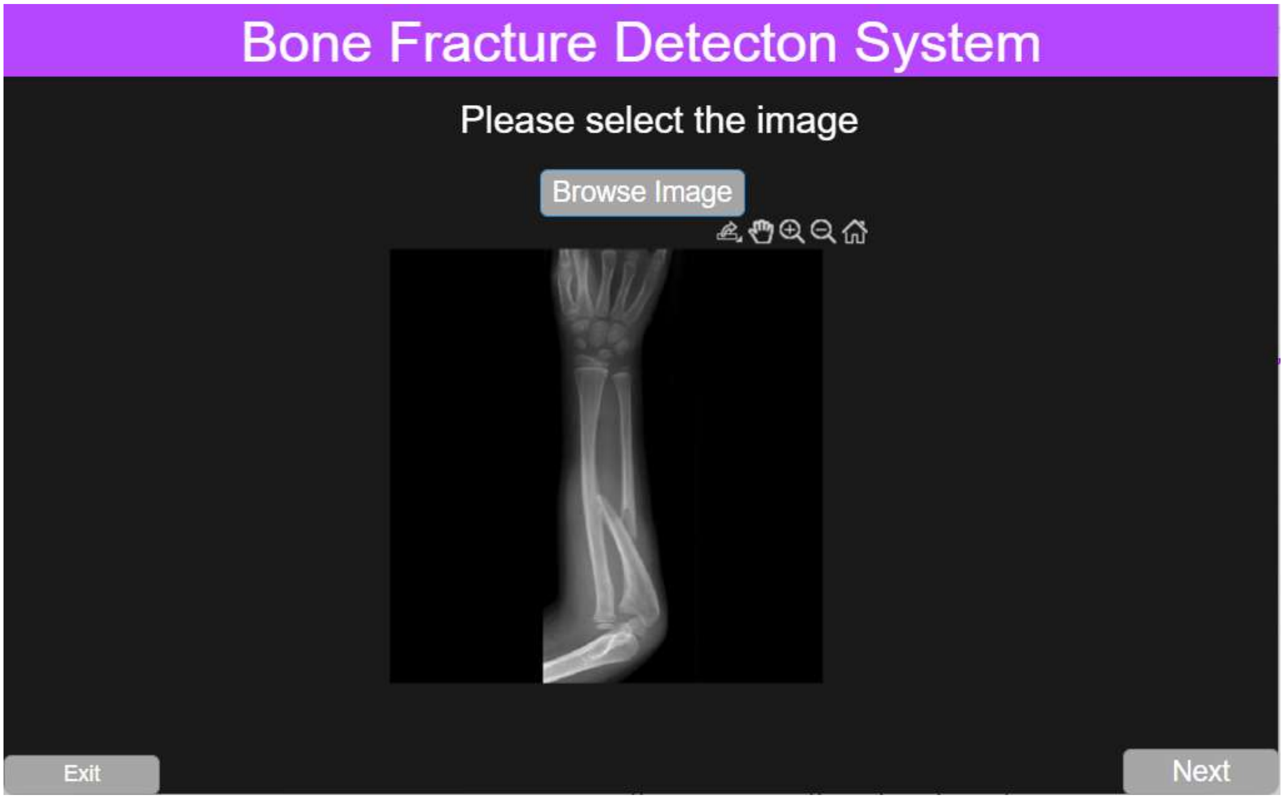
7. **Collaboration and Knowledge Sharing :** Foster collaboration between researchers, medical professionals, and engineers.

Framework:

Step 1: Load Input Image

The process begins by loading the input image, typically an X-ray or CT scan of a bone, which serves as the foundation for the detection process. The quality and resolution of the input image significantly impact the accuracy of subsequent steps.

```
            [file, path] = uigetfile({'*.png;*.jpg;*.jpeg;*.bmp', 'Image Files (*.png, 
*.jpg, *.jpeg, *.bmp)'});
            if isequal(file, 0)
                return;
            end
            imgPath = fullfile(path, file);
            Image = imread(imgPath);
            imshow(Image, 'Parent', app.UIAxes);
            % Assuming you have already loaded the image into 'img' variable
            % Assuming you have already loaded the image into 'img' variable
            setappdata(0, 'imageData', Image);
```



Step 2: Preprocess Image using Gaussian and Median Filters

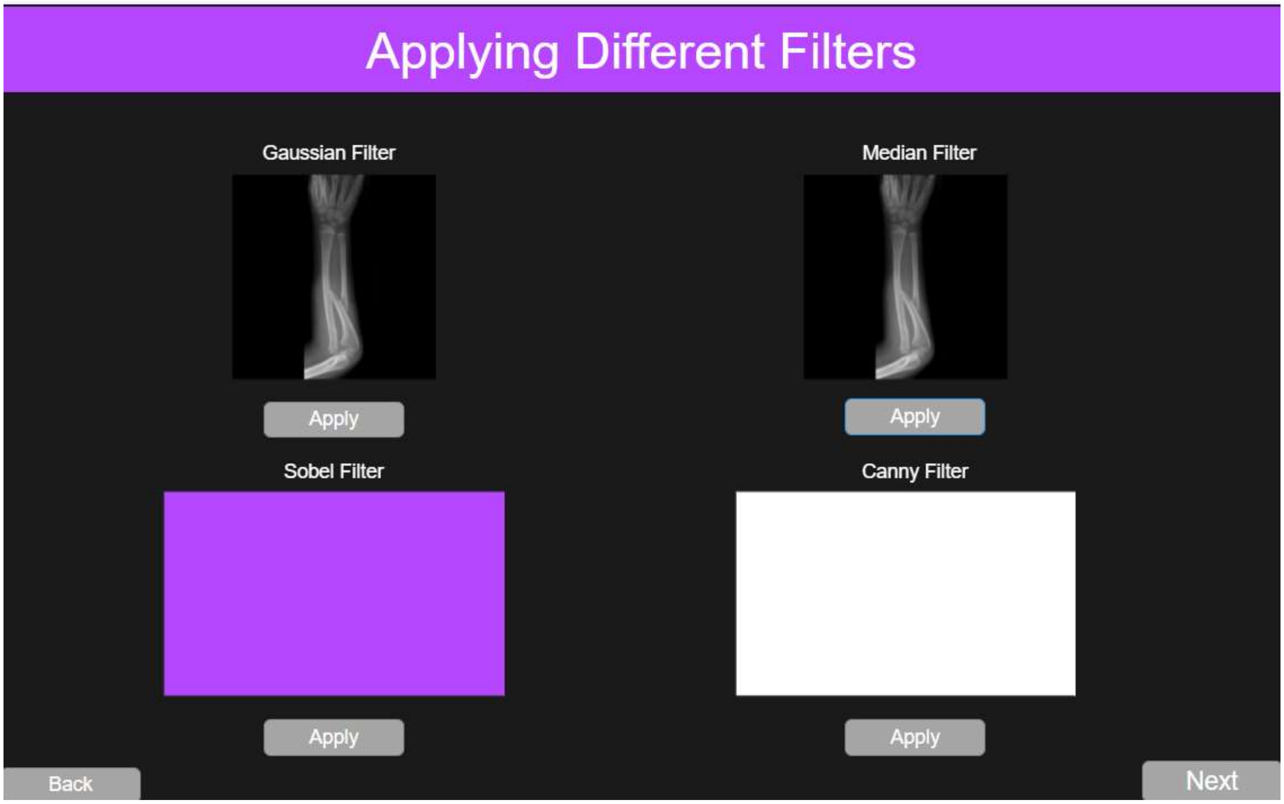
Preprocessing involves using filters to reduce noise while preserving important features. The Gaussian filter smooths the image and reduces high-frequency noise, while the Median filter effectively removes salt-and-pepper noise while preserving edges.

GAUSSIAN:

```
            filteredImg = imgaussfilt(Image, 2);
            imshow(filteredImg, 'Parent', app.UIAxes);
```

MEDIAN:

```
if size(filteredImg, 3) == 3
                filteredImg = rgb2gray(filteredImg);
            end

            % Apply median filter
            denoised_image = medfilt2(filteredImg);
```



Step 3: Convert to Grayscale

If the image is not already in grayscale, it is converted to grayscale for easier processing. Grayscale images contain only shades of gray, simplifying the analysis compared to color images which have a wider range of color information.

```
if size(filteredImg, 3) == 3
  filteredImg = rgb2gray(filteredImg);
```

Step 4: Edge Detection using Sobel and Canny Filters

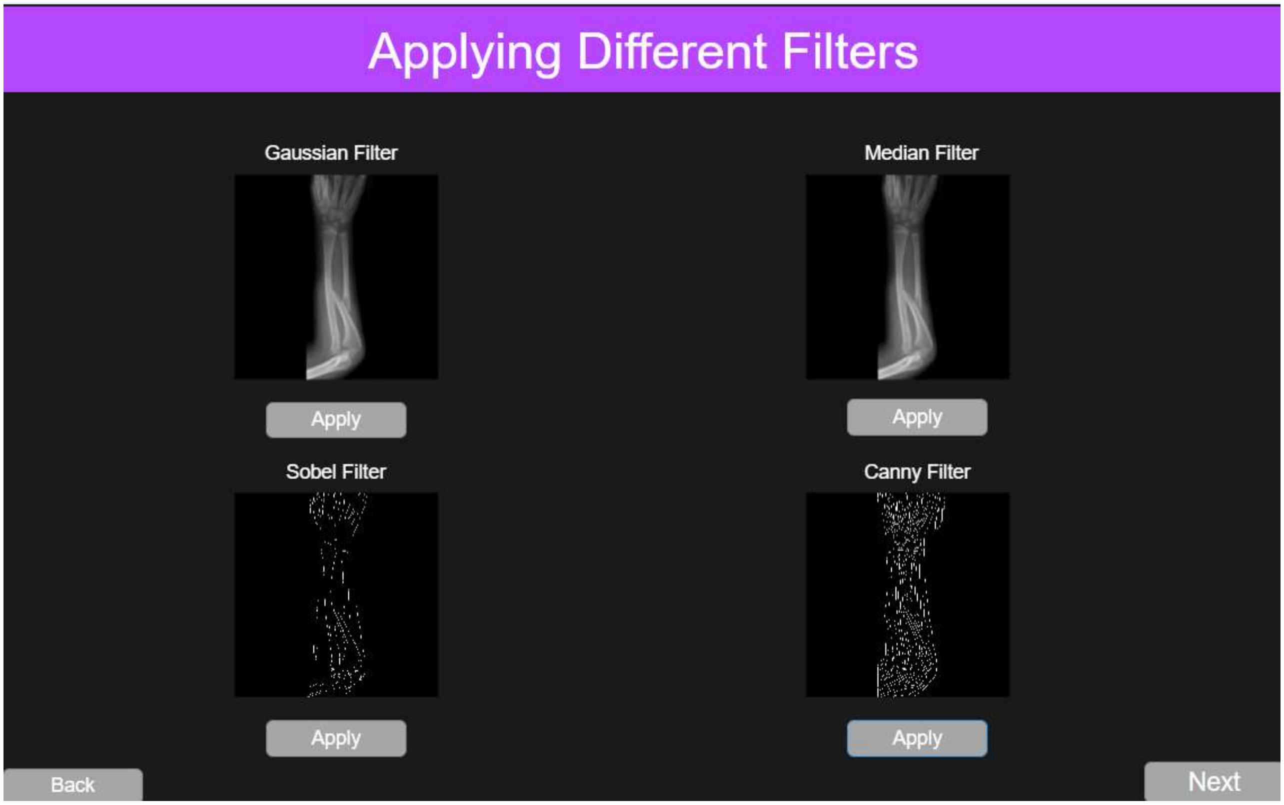
Edge detection highlights the boundaries of bone structures, which are crucial for identifying fractures. The Sobel filter computes the gradient of the image intensity, emphasizing edges, while the Canny edge detector is a multi-stage algorithm that effectively detects a wide range of edges in images.

Sobel:

```
sobel_image = edge(denoised_image, 'Sobel');
imshow(sobel_image, 'Parent', app.UIAxes3);
```

Canny:

```
canny_image = edge(denoised_image, 'Canny');
imshow(canny_image, 'Parent', app.UIAxes4);
```



Step 5: Enhancement using Histogram Equalization

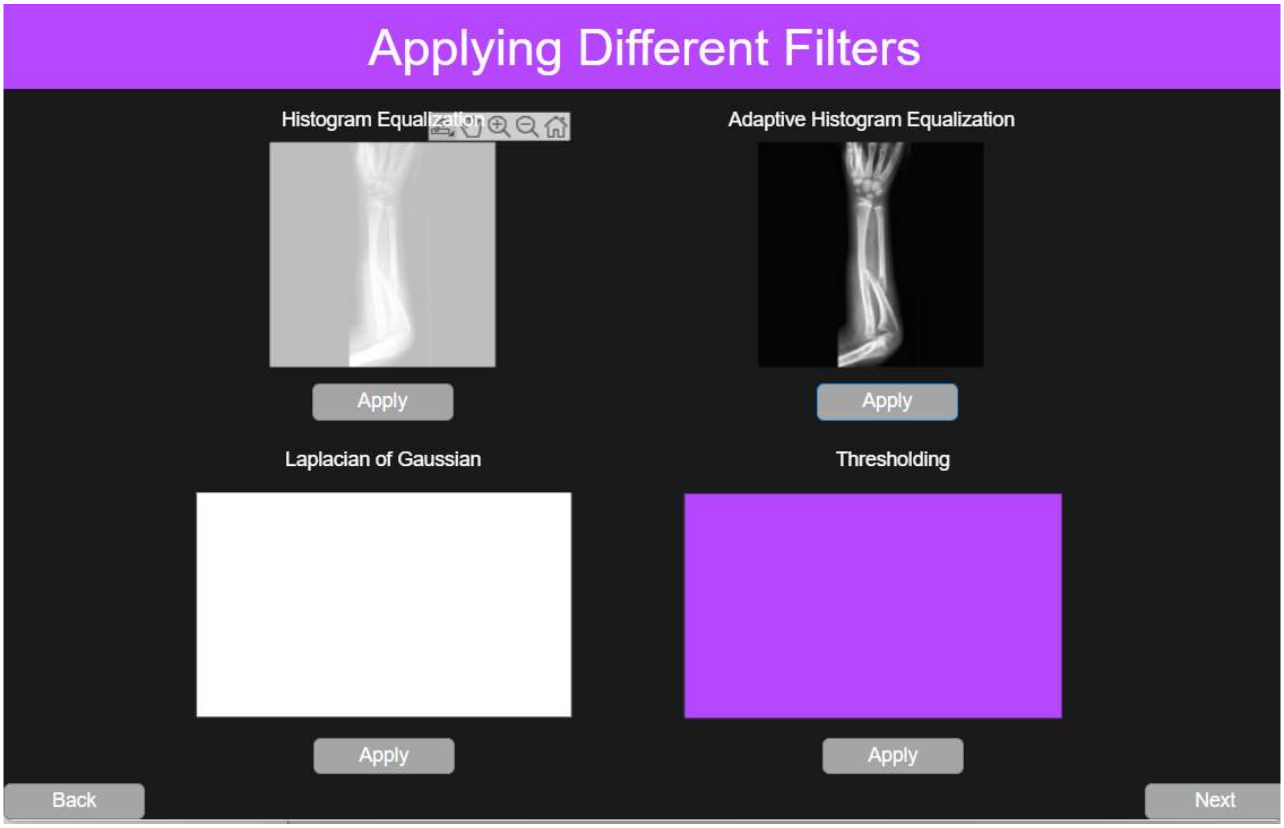
Enhancement techniques such as Histogram Equalization and Adaptive Histogram Equalization (CLAHE) improve the contrast of the image. Histogram Equalization stretches out the intensity range, while CLAHE works on small regions to improve local contrast, making fractures more visible.

Histogram Equalization:

```
equalized_image = histeq(denoised_image);
imshow(equalized_image, 'Parent', app.UIAxes5);
```

adaptive Histogram Equalization:

```
clahed_image = adapthisteq(denoised_image);
imshow(clahed_image, 'Parent', app.UIAxes6);
```

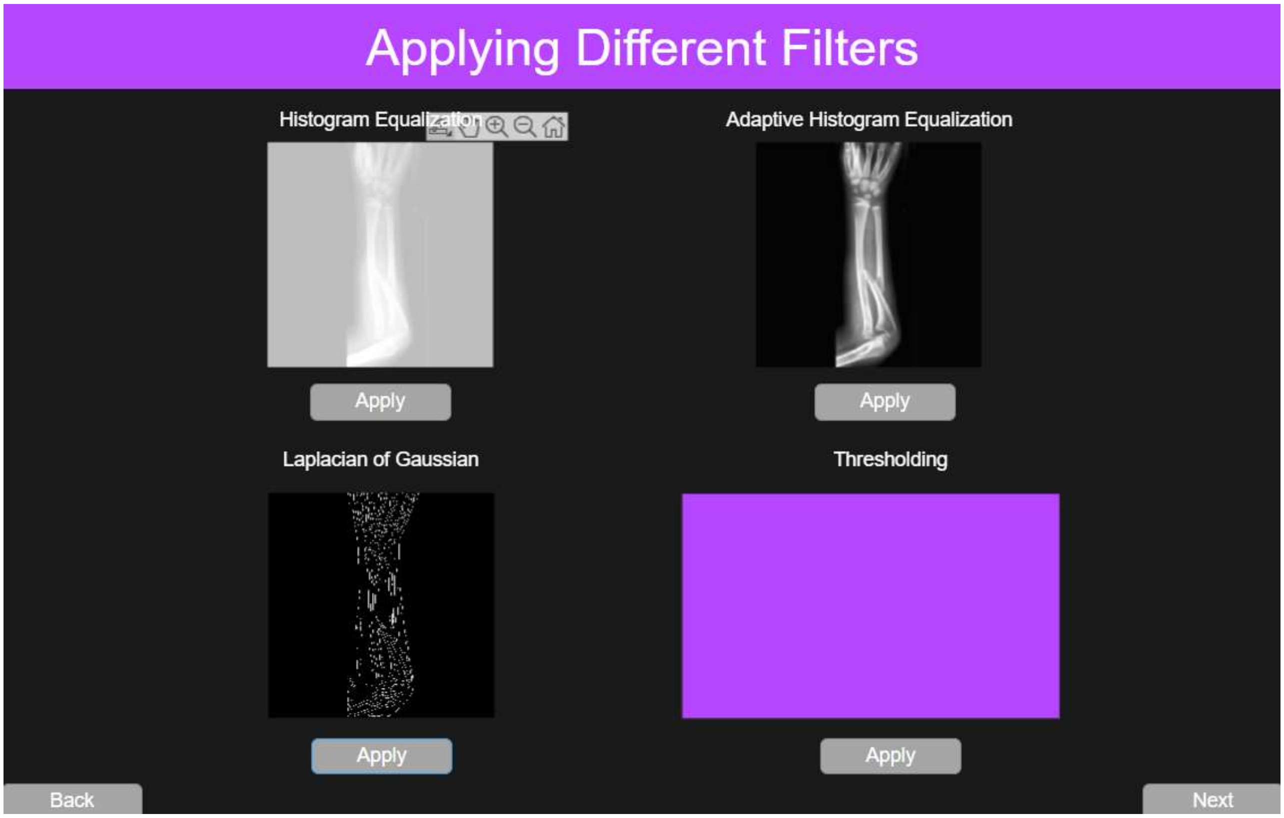


Step 6: Feature Extraction using LoG and Gabor Filters

Feature extraction involves highlighting regions of interest, such as potential fractures. The Laplacian of Gaussian (LoG) filter highlights regions of rapid intensity change and is often used for edge detection. Gabor filters are useful for texture analysis and feature extraction, responding to specific frequencies and orientations in the image.

LoG:

```
log_image = edge(denoised_image, 'log');
 imshow(log_image, 'Parent', app.UIAxes7);
```

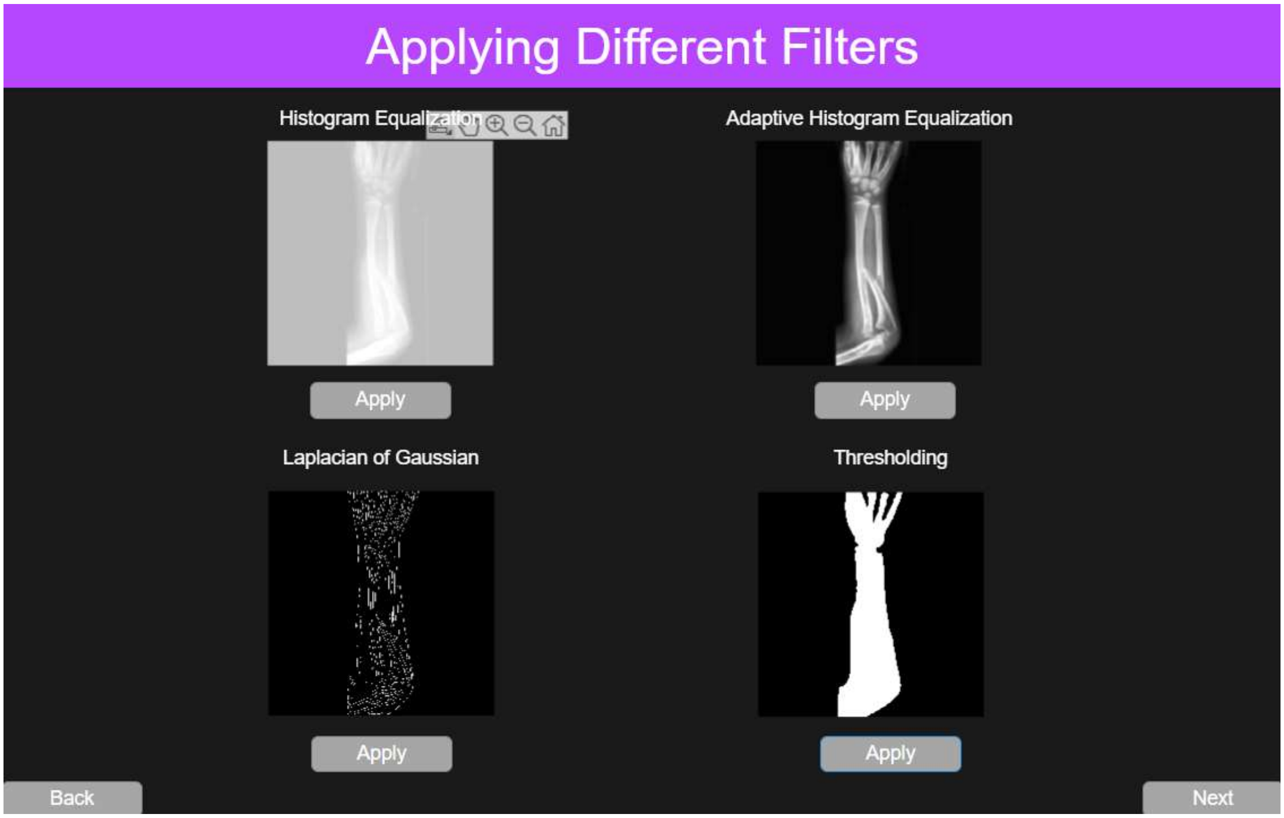


Step 7: Segmentation using Thresholding and Region Growing

Segmentation separates the bone regions and identifies fractures. Thresholding creates binary images from grayscale images by setting a specific intensity value that separates the foreground (bone) from the background. Adaptive thresholding can handle varying lighting conditions. Region growing segments regions based on predefined criteria, useful for separating bone from the background.

Thresholding:

```
threshold_value = graythresh(equalized_image);
thresholded_image = imbinarize(equalized_image, threshold_value);
imshow(thresholded_image, 'Parent', app.UIAxes8);
```

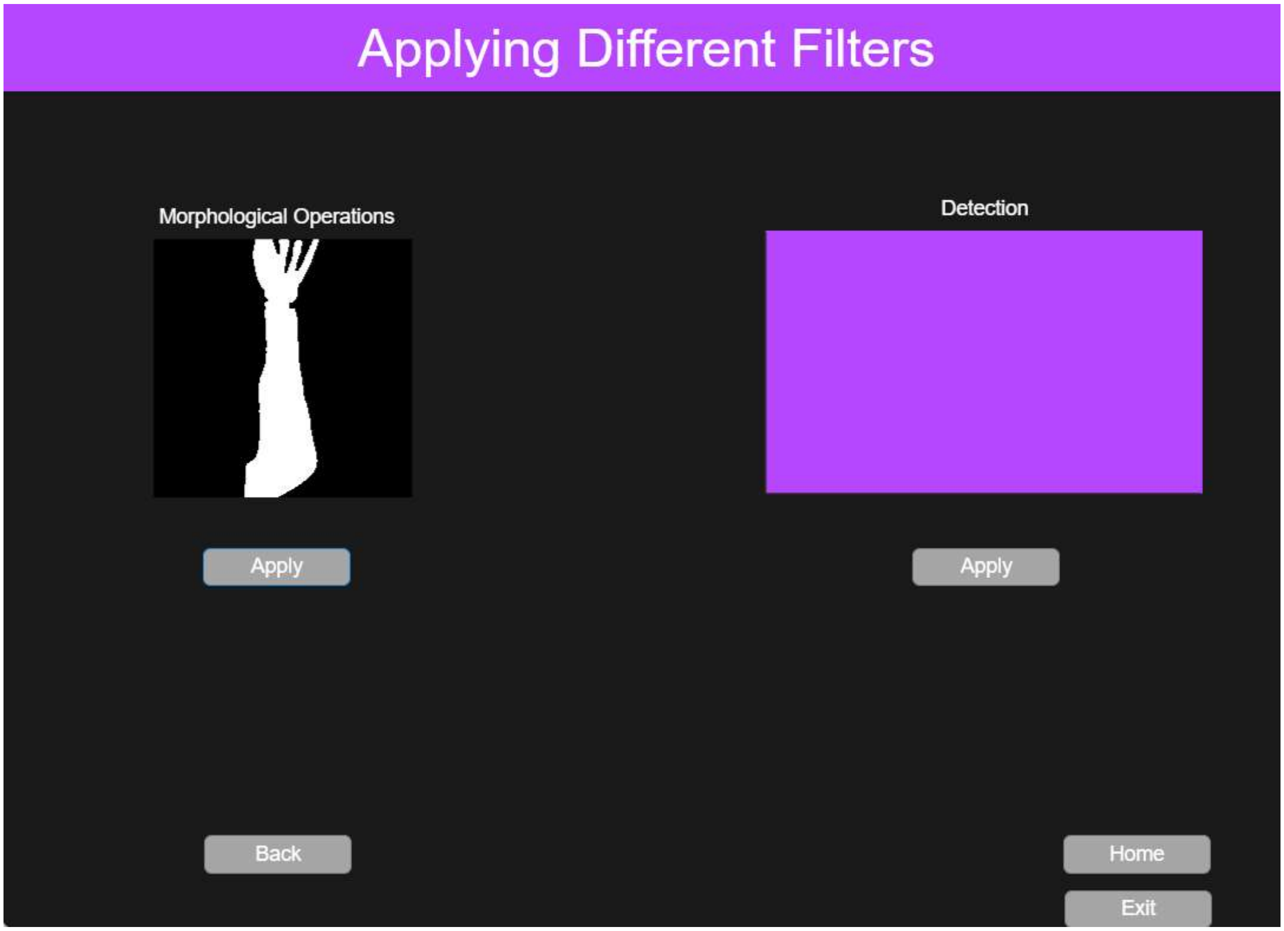


Step 8: Morphological Operations

Morphological operations refine the segmented regions and improve fracture detection. Operations such as erosion and dilation remove noise and fill gaps in detected edges. Opening and closing, which are combinations of erosion and dilation, remove small objects from the foreground or background.

Morphological Operations:

```
threshold_value = graythresh(equalized_image);
thresholded_image = imbinarize(equalized_image, threshold_value);
se = strel('disk', 3);
eroded_image = imerode(thresholded_image, se);
dilated_image = imdilate(eroded_image, se);

opened_image = imopen(dilated_image, se);
closed_image = imclose(opened_image, se);

detected_bones = closed_image;
imshow(detected_bones, 'Parent', app.UIAxes10);
```



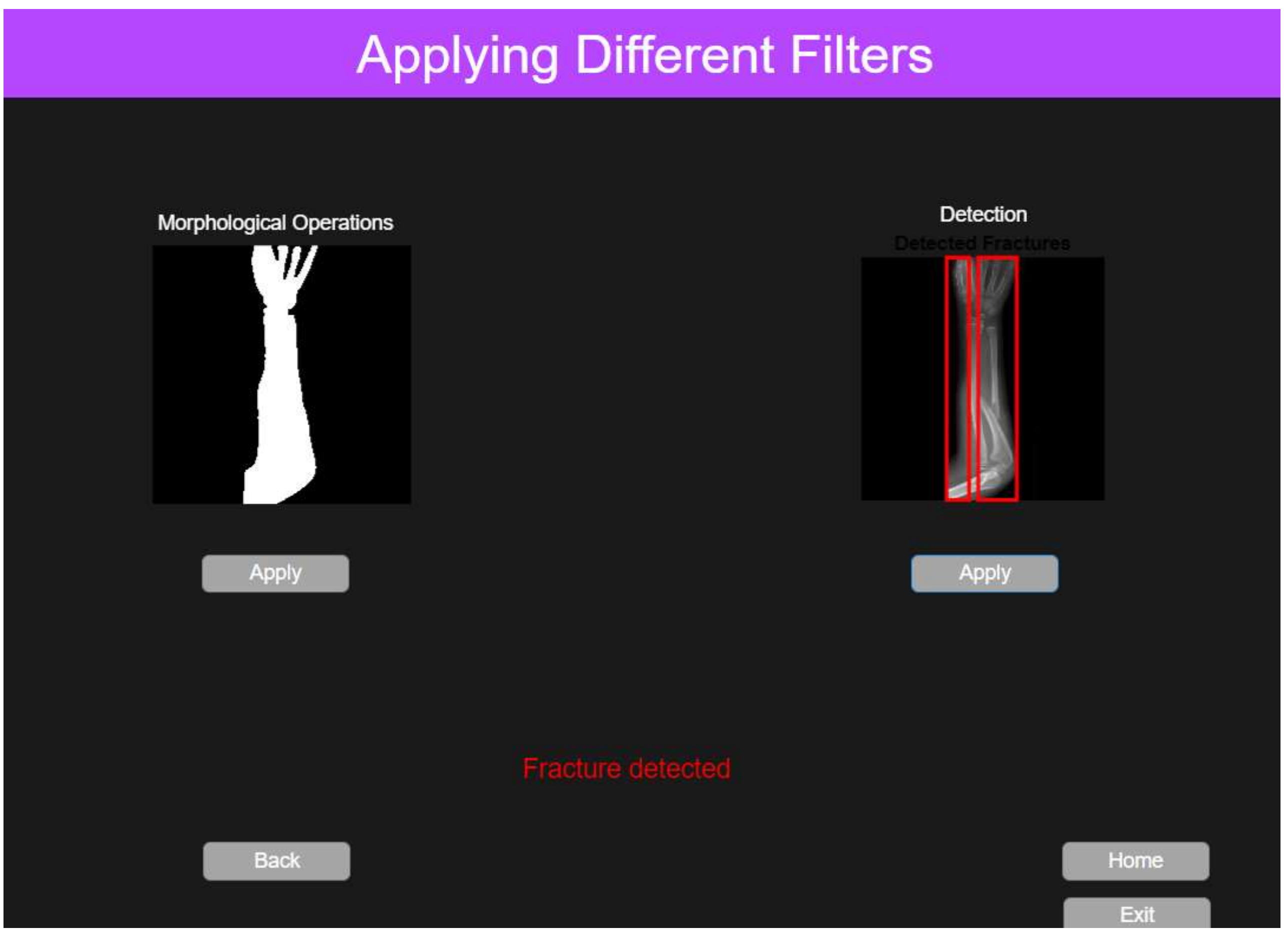
Conclusion:

This project outlines a systematic approach for bone fracture detection using MATLAB, encompassing image preprocessing, edge detection, enhancement, feature extraction, segmentation, and morphological operations. By following these steps, the system can effectively detect and visualize bone fractures, aiding in early diagnosis and improving patient outcomes. The integration of advanced image processing techniques ensures accurate and efficient fracture detection, providing valuable support to medical professionals.

Detection:

```
edges = edge(detected_bones, 'Canny');

% Find Connected Components
CC = bwconncomp(edges);

% Measure properties of image regions
stats = regionprops(CC, 'Area', 'Perimeter', 'BoundingBox', 'Solidity');

% Filter out small regions
area_threshold = 500; % Increase threshold as needed
solidity_threshold = 0.9; % Adjust based on fracture characteristics
```

```
    large_regions = find([stats.Area] > area_threshold & [stats.Solidity] <
solidity_threshold);

    % Display the original image with bounding boxes around large regions
    imshow(Image, 'Parent', app.UIAxes11);
    hold(app.UIAxes11, 'on');
    for idx = large_regions
        rectangle('Position', stats(idx).BoundingBox, 'EdgeColor', 'r', 'LineWidth',
2, 'Parent', app.UIAxes11);
    end
    hold(app.UIAxes11, 'off');
    title(app.UIAxes11, 'Detected Fractures');

    % Update the fracture detection label
    if isempty(large_regions)
        app.Label.Text = 'No fracture detected';
        app.Label.FontColor = [0 1 0]; % Green
    else
        app.Label.Text = 'Fracture detected';
        app.Label.FontColor = [1 0 0]; % Red
    end
```



Future Advancements:

Future advancements in bone fracture detection may include:

1. **Deep Learning Techniques :** Utilizing advanced algorithms like CNNs and RNNs to enhance accuracy.

2. **Multi-Modal Fusion :** Combining different imaging techniques for comprehensive information.

3. **Real-Time Detection :** Optimizing hardware and algorithms for faster real-time analysis.

4. **Automated Segmentation and Localization :** Developing algorithms for automatic and precise localization.

5. **Integration with Clinical Decision Support Systems :** Connecting the detection system with clinical decision support for personalized treatment recommendations.

6. **Interpretability and Explainability** : Developing visualization techniques to provide insights into the system's predictions.

7. **Transfer Learning and Generalization :** Utilizing pre-trained models and transfer learning for efficient adaptation to new datasets.

8. **Validation and Clinical Trials :** Conducting extensive validation and trials to evaluate the system's performance and clinical usefulness.

G. **Integration with Electronic Health Records (EHR)** : Connecting the system with EHR for streamlined workflow and seamless data exchange.

10. **Accessibility and Deployment :** Focusing on user-friendly interfaces, cloud-based solutions, and mobile applications for wider accessibility.

These advancements aim to improve accuracy, efficiency, and accessibility in bone fracture detection, benefiting patients and healthcare professionals.