Sobel filter:

```
% Read Input Image
input_image = imread('butterfly.jpg');

% Displaying Input Image
input_image = uint8(input_image);
figure, imshow(input_image); title('Input Image');

% Convert the truecolor RGB image to the grayscale image
input_image = rgb2gray(input_image);

% Convert the image to double
input_image = double(input_image);

% Pre-allocate the filtered_image matrix with zeros
filtered_image = zeros(size(input_image));

% Sobel Operator Mask
Mx = [-1 0 1; -2 0 2; -1 0 1];
My = [-1 -2 -1; 0 0 0; 1 2 1];

% Edge Detection Process
% When i = 1 and j = 1, then filtered_image pixel
% position will be filtered_image(2, 2)
% The mask is of 3x3, so we need to traverse
% to filtered_image(size(input_image, 1) - 2%, size(input_image, 2) - 2)
% Thus we are not considering the borders.
for i = 1:size(input_image, 1) - 2
 for j = 1:size(input_image, 2) - 2

 % Gradient approximations
 Gx = sum(sum(Mx.*input_image(i:i+2, j:j+2)));
 Gy = sum(sum(My.*input_image(i:i+2, j:j+2)));

 % Calculate magnitude of vector
 filtered_image(i+1, j+1) = sqrt(Gx.^2 + Gy.^2);

 end
end

% Displaying Filtered Image
filtered_image = uint8(filtered_image);
figure, imshow(filtered_image); title('Filtered Image');

% Define a threshold value
thresholdValue = 100; % varies between [0 255]
output_image = max(filtered_image, thresholdValue);
output_image(output_image == round(thresholdValue)) = 0;
```

```
% Displaying Output Image
output_image = im2bw(output_image);
figure, imshow(output_image); title('Edge Detected Image');Prewitt filter:
% Read Input Image
input_image = imread('butterfly.jpg');

% Displaying Input Image
input_image = uint8(input_image);
figure, imshow(input_image); title('Input Image');

% Convert the truecolor RGB image to the grayscale image
input_image = rgb2gray(input_image);

% Convert the image to double
input_image = double(input_image);

% Pre-allocate the filtered_image matrix with zeros
filtered_image = zeros(size(input_image));

% Prewitt Operator Mask
Mx = [-1 0 1; -1 0 1; -1 0 1];
My = [-1 -1 -1; 0 0 0; 1 1 1];

% Edge Detection Process
% When i = 1 and j = 1, then filtered_image pixel
% position will be filtered_image(2, 2)
% The mask is of 3x3, so we need to traverse
% to filtered_image(size(input_image, 1) - 2
%, size(input_image, 2) - 2)
% Thus we are not considering the borders.
for i = 1:size(input_image, 1) - 2
 for j = 1:size(input_image, 2) - 2

 % Gradient approximations
 Gx = sum(sum(Mx.*input_image(i:i+2, j:j+2)));
 Gy = sum(sum(My.*input_image(i:i+2, j:j+2)));

 % Calculate magnitude of vector
 filtered_image(i+1, j+1) = sqrt(Gx.^2 + Gy.^2);

 end
end

% Displaying Filtered Image
filtered_image = uint8(filtered_image);
figure, imshow(filtered_image); title('Filtered Image');
```

```
% Define a threshold value
thresholdValue = 100; % varies between [0 255]
output_image = max(filtered_image, thresholdValue);
output_image(output_image == round(thresholdValue)) = 0;

% Displaying Output Image
output_image = im2bw(output_image);
figure, imshow(output_image); title('Edge Detected Image');Robert filter:
% MATLAB Code | Robert Operator from Scratch

% Read Input Image
input_image = imread('butterfly.jpg');

% Displaying Input Image
input_image = uint8(input_image);
figure, imshow(input_image); title('Input Image');

% Convert the truecolor RGB image to the grayscale image
input_image = rgb2gray(input_image);

% Convert the image to double
input_image = double(input_image);

% Pre-allocate the filtered_image matrix with zeros
filtered_image = zeros(size(input_image));

% Robert Operator Mask
Mx = [1 0; 0 -1];
My = [0 1; -1 0];

% Edge Detection Process
% When i = 1 and j = 1, then filtered_image pixel
% position will be filtered_image(1, 1)
% The mask is of 2x2, so we need to traverse
% to filtered_image(size(input_image, 1) - 1
%, size(input_image, 2) - 1)
for i = 1:size(input_image, 1) - 1
 for j = 1:size(input_image, 2) - 1% Gradient approximations
 Gx = sum(sum(Mx.*input_image(i:i+1, j:j+1)));
 Gy = sum(sum(My.*input_image(i:i+1, j:j+1)));

 % Calculate magnitude of vector
 filtered_image(i, j) = sqrt(Gx.^2 + Gy.^2);

 end
end

% Displaying Filtered Image
```

```
filtered_image = uint8(filtered_image);
figure, imshow(filtered_image); title('Filtered Image');

% Define a threshold value
thresholdValue = 100; % varies between [0 255]
output_image = max(filtered_image, thresholdValue);
output_image(output_image == round(thresholdValue)) = 0;

% Displaying Output Image
output_image = im2bw(output_image);
figure, imshow(output_image); title('Edge Detected Image’
```