Mini-project 2018: Image processing

SOLUTIONS

1 Rotation

```
function output_image = rotate_image(input_image, theta)

    % Define the tranformation matrix
    theTransform = inv([cos(theta) sin(theta);...
                        -sin(theta) cos(theta)]);

    % Get size of input image
    [ny, nx] = size(input_image);

    % Determine centre of input image
    xCent = round(nx/2); yCent = round(ny/2);

    % Preallocate output image
    output_image = zeros(ny, nx);

    % Preallocate source pixel
    src = zeros(2, 1);

    % Loop through each pixel of output image and find
    % coordinates of input image for source pixel
    for ix = 1: nx
        for iy = 1: ny
            src=round(theTransform*([ix; iy]-[xCent; yCent])+...
                                                [xCent; yCent]);
            xSrc = src(1);
            ySrc = src(2);

            % Check if source coordinates are valid
            if (~(xSrc<1 || ySrc<1 || xSrc>nx || ySrc>ny))
                output_image(iy, ix) = input_image(ySrc, xSrc);
            end
        end
    end
end
```

2 Edge detection

2.1 Loops

```
function output_image = detect_edges(input_image)

    % Get size of input image
    [ny, nx] = size(input_image);

    % Preallocate output image
    output_image = zeros(ny-1,nx-1);

    % Loop though each pixel of the output image and apply the edge
    % detection formula to the input image
    for iy = 1:ny-1
        for ix = 1:nx-1
            output_image(iy,ix) = (abs(input_image(iy,ix) -...
                                       input_image(iy+1,ix)) + ...
                                   abs(input_image(iy,ix) -...
                                       input_image(iy,ix+1)))/2;
        end
    end
end
```

2.2 Vectorised

```
function output_image = detect_edges_vectorised(input_image)

    % get size of input image
    [ny, nx] = size(input_image);

    % Create vectors used to index input image
    yv = 1:ny-1;
    xv = 1:nx-1;

    % Apply edge detection formula
    output_image=...
                (abs(input_image(yv, xv) -input_image(yv, xv+1)) +...
                 abs(input_image(yv, xv)-input_image(yv+1, xv)))/2;

end
```

3 Blurring

3.1 Loops

```
function output_image = blur_image(input_image)

    % get size of input image
    [ny, nx] = size(input_image);

    % Preallocate output image
    output_image = zeros(ny-2,nx-2);

    % Loop through each pixel of the output image and apply the
    % blurring formula to the input image (by looping through
    % neighbouring pixels)
    for iy = 1:ny-2
        for ix = 1:nx-2
            for i = 0:2
                for j = 0:2
                    output_image(iy,ix) = output_image(iy,ix) +...
                                          input_image(iy+i,ix+j)/9;
                end
            end
        end
    end
end
```

3.2 Vectorised

```
function output_image = blur_image_vectorised(input_image)

    % Ensure input is double precision, if it is uint
    % pixel values will clip at 255
    input_image = double(input_image);

    % get size of input image
    [ny, nx] = size(input_image);

    % Create vectors used to index input image
    yv = 1:ny-2;
    xv = 1:nx-2;

    % Apply edge blurring formula
    output_image=...
        (input_image(yv, xv) + input_image(yv+1, xv) +...
         input_image(yv + 2, xv) + input_image(yv, xv+1) +...
         input_image(yv, xv+2) + input_image(yv+1, xv+1) + ...
         input_image(yv+1, xv+2) + input_image(yv+2, xv+1) +...
         input_image(yv+2, xv+2))/9;

end
```