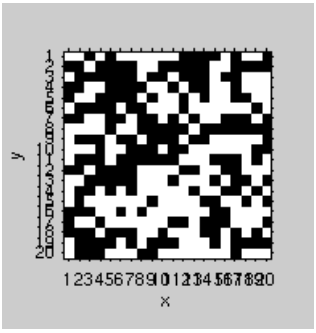
Assignment 1 - Signal and Image Processing 2014

Martin Jørgensen, tzk173

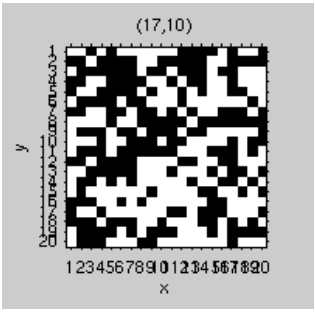
September 4, 2014

Contents

Task 1.1



(a) The randomly generated image.



(b) The randomly generated image with a blackened pixel as well as the coordinates.

This task was solved by generating a 20×20 matrix of ones and zeros. Then using ginput to retrieve the coordinates for a click and setting that pixel to value 0. Source code is in the appendix.

Task 1.2

To solve this I generated a 60×60 image in the same way as Task 1.1, and displayed it using the three different methods `imshow`, `image` and `imagesc`. They are descriped in the list below.

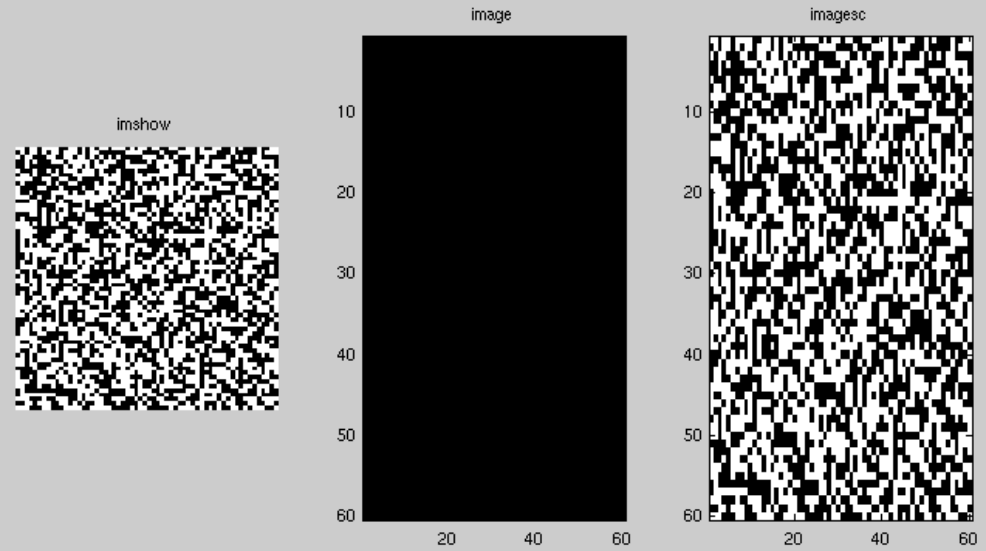


Figure 2: The same image displayed using the different methods.

`imshow` Can take either a string with a filename/path, an image handle or a matrix as input and displays it as an image. It has a number of default attributes, such as disabling axis labels and ticks, and forcing square pixels by making the x and y axis have the same distance between ticks. It will also try to guess whether an image/matrix us in grayscale, RGB or binary.

`image` Takes a matrix as input which it will interpret as an image. The resulting image object will attempt to stretch to fill whatever window it is nested in, resulting in rectangular pixels. It will keep the tick marks along the images. Will use the active colormap instead of trying to guess what the data means.

`imagesc` Have the same behaviours as image, but will scale the image data to make use of the full colormap.

The source code is in the appendix.

Task 1.3

This task was solved by using a for loop and an array function to extract a given bit for all pixels at once and order it into sub plots. See appendix for source

code.

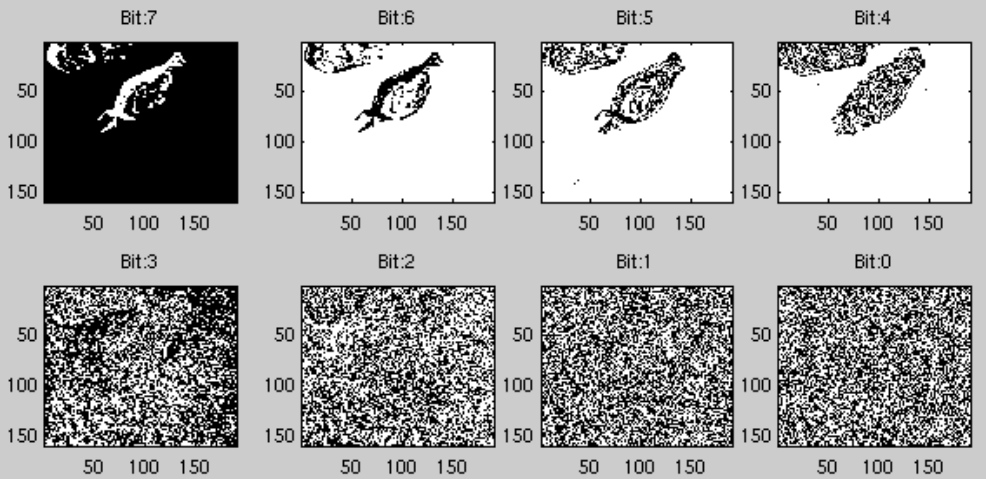


Figure 3: The bit plances obtained from splicing *cell.tif*

Just like in the books Figure 1.3 on page 5, we see that the most significant bits contain most of the coarse image information so the lower bits resembles noise. Since this is a grayscale image where objects that er visible are whiter (intensity close to 255) it makes sense that objects are visible in the higher bit planes. Thw lower bit planes encompas only small variations in the visiblity but a change in for isntance the 8'th bit will change the intensity with 128, or half of the possible intensity.

Task 1.4

For this task I chose one of the sample pictures, the cookie monster picture. (*monster.jpg*). The code is very simple and can be found in the appendix.

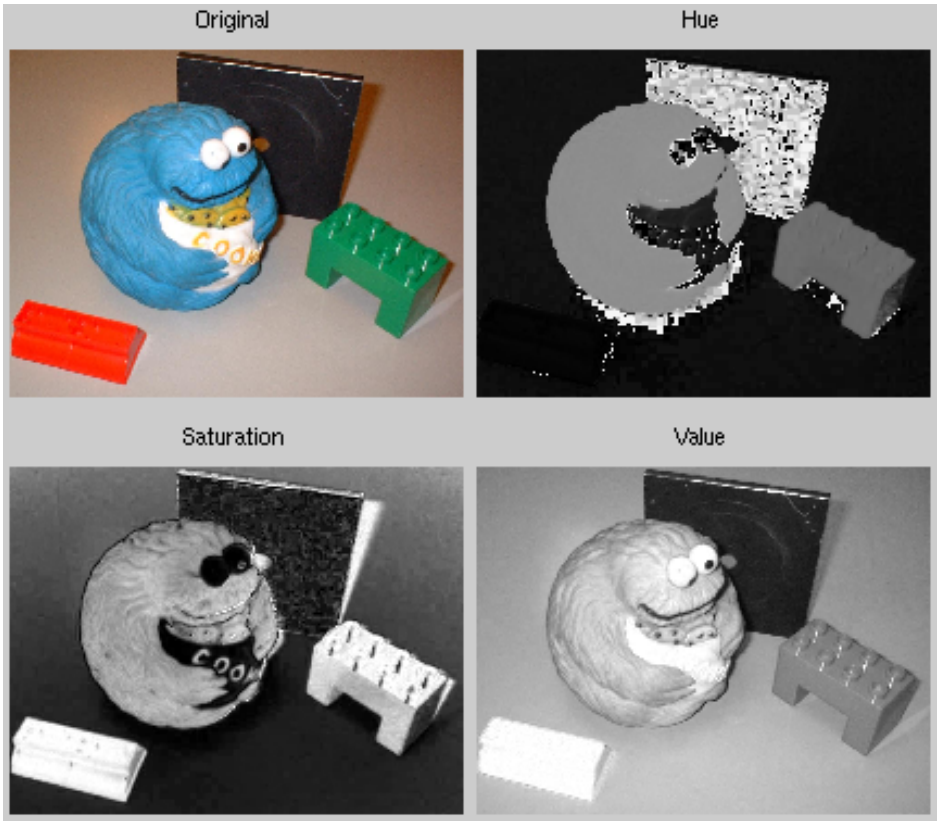


Figure 4: The HSV channels extracted from the original image.

Task 1.5

The code used to generate the figures can be found in the appendix.

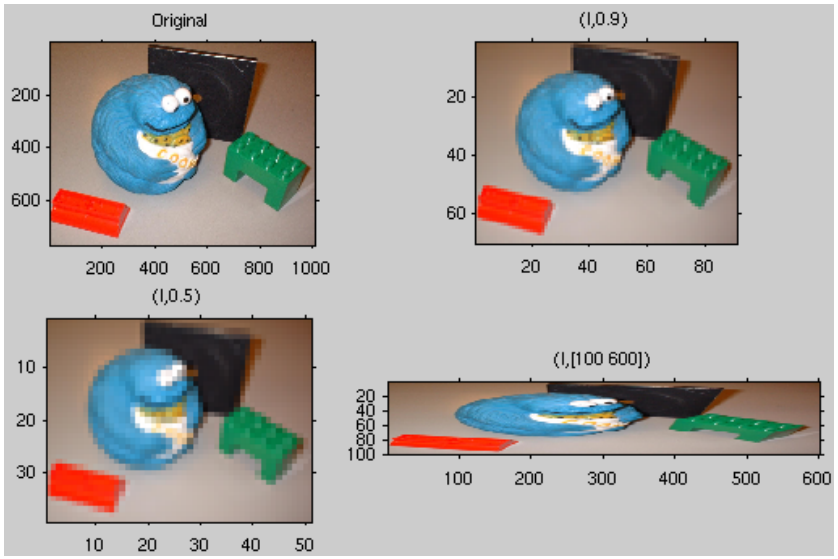


Figure 5: The Cookie monster image sized down using three different scales. The last one also ruins the aspect ratio of the image.

In the upper right image, ther eis clear aliasing as the resolution falls and “smooth” edges can no longer be drawn. It is still possible to make out all the details, evne though the curls in Cookie Monsters fur and the ligh-reflections on the green lego brick is harder to make out. In the lower left image on Figure 5 several details are gone, the pupils in Cookie Monsters eyes as well as the reflections on the lego block are almost impossible to make out. The text on the jar is unreadable. The lower right image in the same figure does not loose the details, but because the aspext-ratio is skewed, some details like the text on the jar is a bit hard to make out.

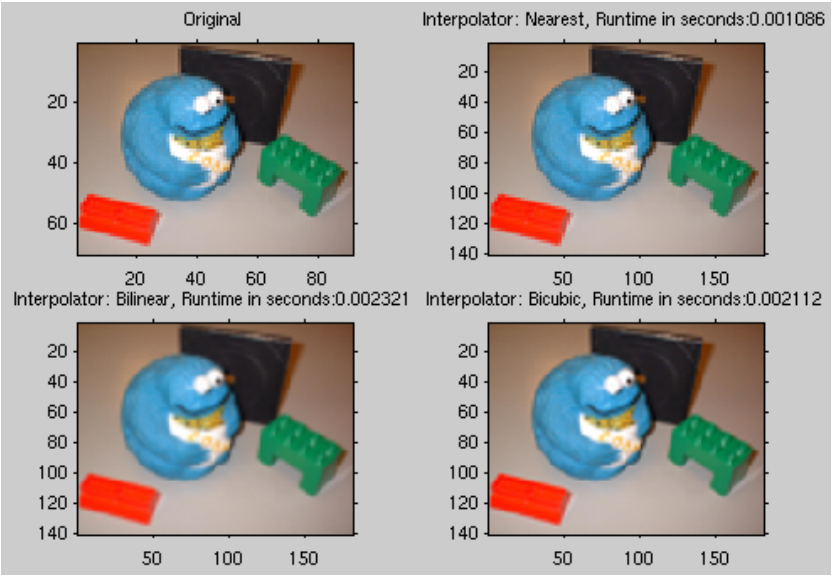


Figure 6: Results when scaling up a low resolution image, in this case the original is the cookie monster picture scaled down.

The upper right image in Figure 6 is created by giving all the new pixels the same value as the pixel that is nearest their position in the original image. This essentially "stretches" all the pixels and gives a very jagged look. When comparing it to the original they're identical since they're displayed in the same size. The two lower figures are taken by giving new pixels a value based on the average value in their neighborhood in the original image. (2×2 and 4×4 neighborhoods respectively.) The edges gets a lot softer and less jagged, but the image gets a very fussy quuality to it, as if the camera was out of focus.

Task 1.6

The code for this task is shown in the appendix.

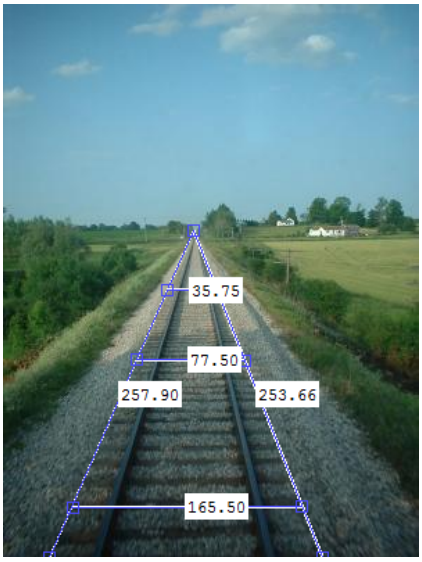


Figure 7: Measurements (in pixels) of the railway sleepers.

Figure 7 shows the measurements of the sleepers on the railway. 2 parallel lines were added at the end of the sleepers in order to facilitate easier measurements of the sleepers that are further into the background, since these were blurry.

The first measures sleepers is 165,5 pixels across. The second is 77.5 px and the third is 35.75 px. It is impossible to recreate the (X, Y, Z) positions of the sleepers without any more information. We lack both focal length, and some information about positions in the scene. We can use the rate at which the sleepers get shorter, i.e. how fast the lines that go to the vanishing point converges, to make some assumptions about how "wide" things in the picture are along the x-axis, but apart from that there is little we can do.

Example, I measured the distance between the two "parallel" lines to be 4.5 px, at the same distance as the first white house, the wall of the white house is roughly 4.5 px tall and 25.38 px wide. Knowing that for every 4.5 pixel we have roughly 2.5 meters, the wall must be around 2.5 meters tall and 14.1 meters long. ($\frac{25.38px}{4.5px}2.5m = 14.1m$). This assumes that the wall is somewhat parallel to the camera.

Task 1.7

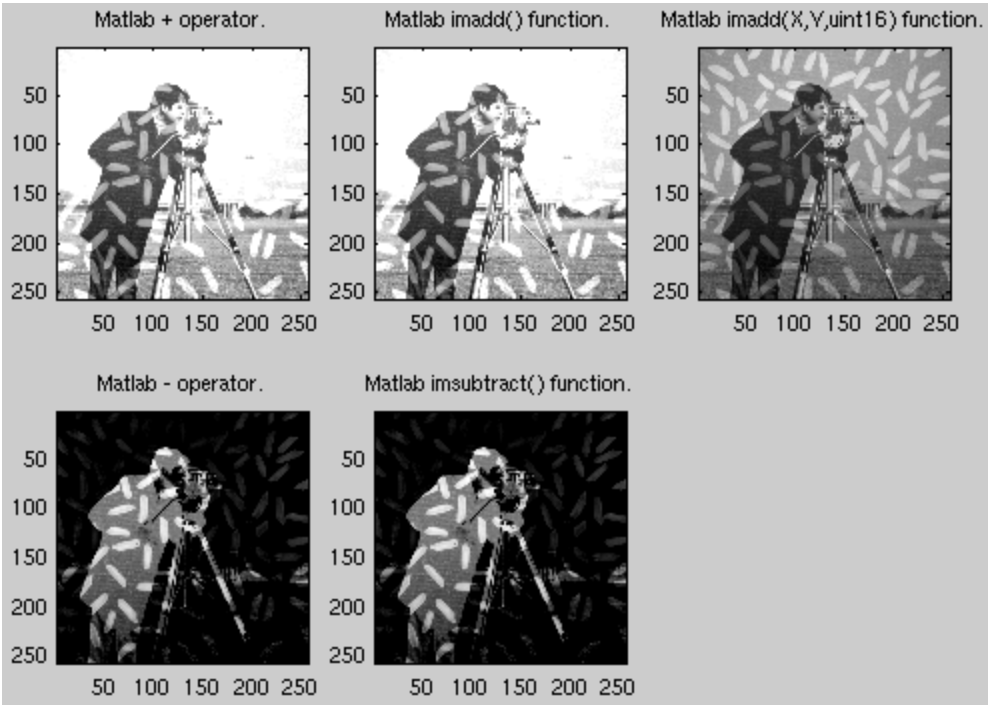


Figure 8: The results fo the different methods and operations.

The two first pictures in Figure 8 is the result of two methods of adding images. Since the image is bound by the 8bit intensity limit, they produce the same result. The add the intensities of the images together, bound it at 255 and displays it. When letting `imadd` (upper right image) give 16 bit output, the result is vastly different. The increased number of different shades means that values that become higher than 255 does not jost go white. The images are showed with `imagesc` in order to increase the visibility of the number of intensities.

For the last two images, just like the first, there is no difference. Intensities cannot go below 0, and as such they just go black. It is not possible to specify 16bit output for `imsubtract`, but it would not help, since we cannot display a negative intensity.

Task 1.8

The code for this task can be found in the appendix.

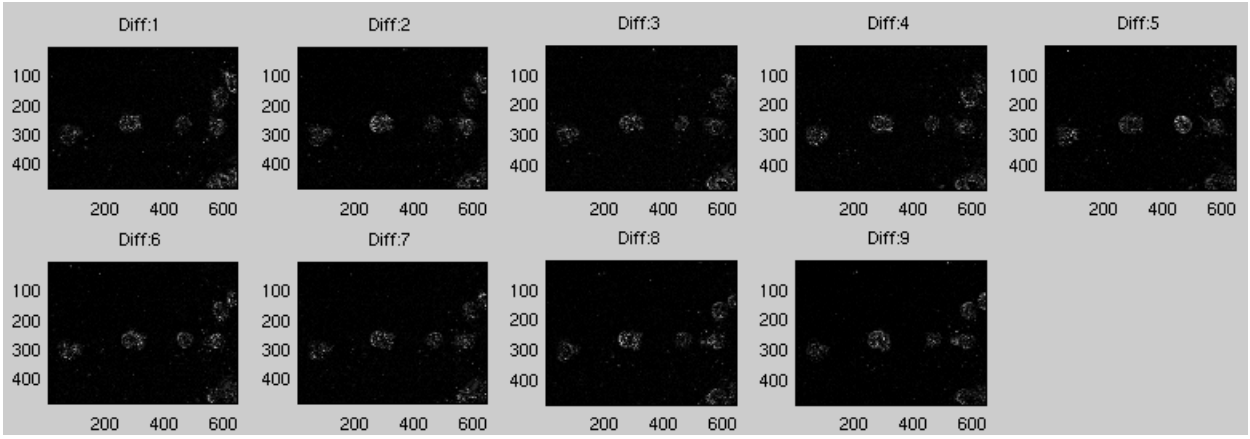


Figure 9: The differences between frames of the cell “animation”.

The difference is maxed by showing the images using `imagesc` which will scale it to use the entire grayscale colormap. This allows us to see details more clearly. The result from the operation is that pixels that change a lot, gets a stronger (whiter) color, so areas with much change will “light up”. This can be usefull for motion tracking, if the background is largely static, like in this image, it i easy to track objects moving over it, simply follow the white areas around.

Task 1.9

The code for both the multiplication function and the experiment can be found in the appendix.

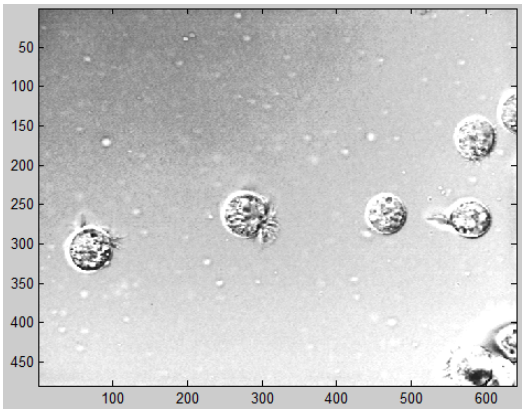


Figure 10:

Appendix

Task 1.1 Code

```
% Solution for part 1.1 of Assignment 1.
% Written by: Martin Jørgensen, tzk173

clear all;



% Generate figure and image.
figure(111);
pixels = randi([0,1],20,20);
h = imshow(pixels); axis on;

% Fix the looks.
set(gca,'XTick',0:20); xlabel('x');
set(gca,'YTick',0:20); ylabel('y');

% Get input and add the coordinate to the title, then color the
% pixel and update the figure data.
[x,y] = ginput(1);
```

```
title(strcat('(' , num2str(round(x)) , ',' , num2str(round(y)) ,')'))
pixels(round(y),round(x)) = 0; set(h,'CData',pixels);
```

Task 1.2 Code

```
% Solution for part 1.2 of Assignment 1.
% Written by: Martin Jørgensen, tzk173

clear all;

% Create figure and image data.
h = figure(121);
pixels = randi([0,1],60,60);

% Create sub figure for imshow.
subplot (1,3,1)
imshow(pixels)
title('imshow')

% Create sub figure for image.
subplot (1,3,2)
image(pixels)
title('image')

% Create sub figure for imagesc.
subplot (1,3,3)
imagesc(pixels)
title('imagesc')
```

Task 1.3 Code

```
% Solution for part 1.3 of Assignment 1.
% Written by: Martin Jørgensen, tzk173
```

```
clear all;

% Read the selected test image.
I = imread('cell.tif');

% Create parent figure.
h = figure(131);

% Go through the bits from least significant to most.
for i=1:8
    subplot(2,4,9-i); % Ordered most-significant leftmost.
    colormap(gray);   % Force colormap.
    imagesc(arrayfun(@(x) bitget(x,i),I)); % apply bitget to each pix
    axis image; title(strcat('Bit:', num2str(i-1)));
end
```

Task 1.4 Code

```
% Solution for part 1.4 of Assignment 1.
% Written by: Martin Jørgensen, tzk173

clear all;

% Read the selected test image.
I = imread('monster.jpg');

% Convert it.
[H,S,V] = rgb2hsv(I);

figure; % Create a new figure
subplot(2,2,1); imshow(I); title('Original');  axis image;
subplot(2,2,2); imshow(H); title('Hue'); axis image;
subplot(2,2,3); imshow(S); title('Saturation'); axis image;
subplot(2,2,4); imshow(V); title('Value'); axis image;
```

Task 1.5 Code

```
% Solution for part 1.5 of Assignment 1.
% Written by: Martin Jørgensen, tzk173

clear all;

% Read the selected test image.
I = imread('monster.jpg');

% Make Throw away resolution.
I1 = imresize(I,0.09);
I2 = imresize(I,0.05);
I3 = imresize(I,[100 600]);

figure(151); % Create a new figure.
subplot(2,2,1); imshow(I); title('Original');  axis image on;
subplot(2,2,2); imshow(I1); title('(I,0.9)'); axis image on;
subplot(2,2,3); imshow(I2); title('(I,0.5)'); axis image on;
subplot(2,2,4); imshow(I3); title('(I,[100 600])'); axis image on;


% Upscaling experiments.

% Warmup the image cache to get better times.
for i=1:50
    I4 = imresize(I1,2,'nearest');
end

% Get the proper timings.
tic
I4 = imresize(I1,2,'nearest');
t1 = toc;
```

```
tic
I5 = imresize(I1,2,'bilinear');
t2 = toc;

tic
I6 = imresize(I1,2,'bicubic');
t3 = toc;


figure(152);
subplot(2,2,1); imshow(I1); title('Original');  axis image on;
subplot(2,2,2); imshow(I4); axis image on;
title(strcat('Interpolator: Nearest, Runtime in seconds: ',num2str(t1
subplot(2,2,3); imshow(I5); axis image on;
title(strcat('Interpolator: Bilinear, Runtime in seconds: ',num2str(t
subplot(2,2,4); imshow(I6); axis image on;
title(strcat('Interpolator: Bicubic, Runtime in seconds: ',num2str(t3
```

Task 1.6 Code

```
% Solution for part 1.6 of Assignment 1.
% Written by: Martin Jørgensen, tzk173

clear all;

% Read the selected image.
I = imread('railway.png');

% Sleeper length is 2.5m
% Do proper math in report.
imtool(I);
```

Task 1.7 Code

```
% Solution for part 1.7 of Assignment 1.
```

```
% Written by: Martin Jørgensen, tzk173

clear all;

% Read the selected image.
I1 = imread('rice.png');
I2 = imread('cameraman.tif');

% Add the images.
I3 = I1 + I2;
I4 = imadd(I1,I2);
I5 = imadd(I1,I2,'uint16');

% Subtract them.
I6 = I1 - I2;
I7 = imsubtract(I1,I2);

% Display them.
figure(171);
colormap(gray);

subplot(2,3,1);
imagesc(I3); axis image;
title('Matlab + operator.');

subplot(2,3,2);
imagesc(I4); axis image;
title('Matlab imadd() function.');

subplot(2,3,3);
imagesc(I5); axis image;
title('Matlab imadd(X,Y,uint16) function.');

subplot(2,3,4);
```

```
imagesc(I6); axis image;
title('Matlab - operator.');

subplot(2,3,5);
imagesc(I7); axis image;
title('Matlab imsubtract() function.');
```

Task 1.8 Code

```
% Solution for part 1.8 of Assignment 1.
% Written by: Martin Jørgensen, tzk173

clear all;

% Read the selected images.
imgs = cell(1,10);
for i=1:9
    imgs{i} = imread(strcat('AT3_1m4_0',num2str(i),'.tif'));
end
imgs{10} = imread(strcat('AT3_1m4_10.tif'));

figure(181);
colormap(gray);
for i=1:9
    subplot(2,5,i);
    imagesc(imabsdiff(imgs{i},imgs{i+1})); axis image;
    title(strcat('Diff: ',num2str(i), '-', num2str(i+1)));
end
```

Task 1.9 Code

Blending function

```
function [ C ] = imgBlend( A, B, wA, wB )
  %imgBlend Performs a weighted blend of the images A and B.
```

```
  %C = A.*wA + B.*wB;
  C = imadd(immultiply(A,wA),immultiply(B,wB),'uint16');
end
```

Experiment/Use

Insert when done.

```
% Solution for part 1.9 of Assignment 1.
% Written by: Martin Jørgensen, tzk173

clear all;

% Read the selected images.
imgs = cell(1,10);
for i=1:9
    imgs{i} = imread(strcat('AT3_1m4_0',num2str(i),'.tif'));
end
imgs{10} = imread(strcat('AT3_1m4_10.tif'));

res = imgBlend(imgs{1},imgs{2},2.5,4);
for i=2:9
    res = imgBlend(imgs{i},imgs{i+1},2.5,4);
end

% Show result.
figure(191);
colormap(gray);
imagesc(res);
```