# TNM087 - Image Processing and Analysis

## Task 4 – White Point Correction

### Background:

From the moment a camera sensor captures an image until it is stored, or displayed, the image is processed in several stages of a processing pipeline. This is shown in the following figure from Richard Szeliski, Computer Vision:

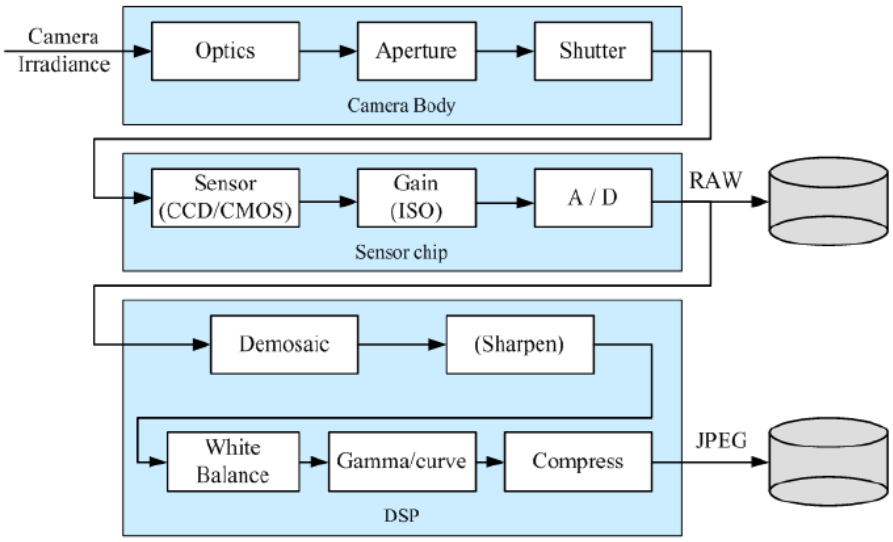


**Figure 2.23** Image sensing pipeline, showing the various sources of noise as well as typical digital post-processing steps.

One of the steps is White Balancing or White Point Correction. This is often used to improve the overall color impression of the image or in computer vision it is used to compensate the influence of the illumination on the final image. A simple way to do this is to pick one (or several) point(s) in an image and map them to white. The simplest method is to scale the pixel values in the three R, G and B channels using global constants. It is usually assumed that similar processes are used by the human visual system. This is known as von-Kries adaptation. https://en.wikipedia.org/wiki/Von_Kries_Coefficient_Law

### Task:

Use ginput to pick one point in the image OImage. In the new image CImage this pixel will have the maximum pixel value in each channel (so that appears to be a white point). The type of the output image is given by the input argument ‘type’.

### Syntax:

```
function CImage = WhitePoint(OImage,type)
```

**Hints:**

OImage is the numerical array and not the filename!

A minimal solution must be able to handle the cases where OImage is of type uint8 or a double array and CImage is of the type uint8 or double. Read the description of im2double and the simple conversion double(..) which are two tools to convert images to double arrays.

In an advanced version you may consider other datatypes for OImage and CImage like uint16.

The processing is done in the

```
% Here you need several lines of code where rgbvec defines the scaling
% after the scaling you have to truncate the pixel values
% Finally you have to convert the result to the datatype given by
otype

CImage =

section of the template. For-loops over pixels are not accepted but
otherwise there are several possible solutions. You can loop over the
channels (R,G,B) or you can use reshape to do it in one line.
For-loops over pixels to truncate the pixel values are also not
accepted.
```

Processing of input parameters related to output variables:

I usually process all input arguments first. Like in this part of the template

```
if (nargin < 2)
    otype = 'b';
else
    if strcmp(type,'b')
        …
end;

…
switch otype

…
```

If you think it is more natural to do this at the end when you actually compute the result you may change the template.