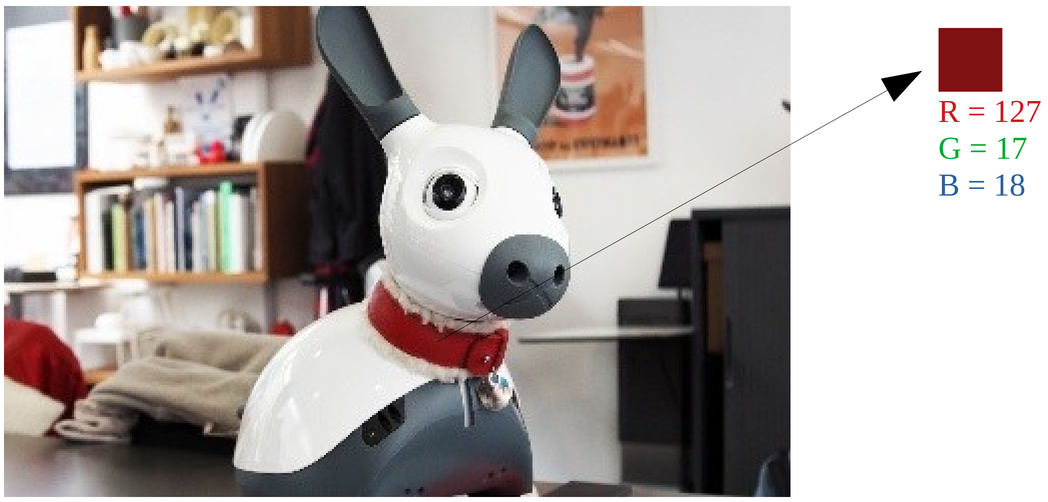
Colour Segmentation

Colour segmenting is perhaps one of the simplest computer vision processes. Each pixel in an image has associated with it three values between 0 and 255 (for an 8-bit image, since $2^8 = 256$). Each value indicates the intensity for that pixel's red, green and blue colours, with the final pixel colour determined by a linear combination of these three values.

Let's extract one pixel from MiRo's collar in below image and show its RGB values:



As expected, there is a stronger intensity of the red pixel value vs. green and blue.

When we segment the image according to some RGB boundaries, we're selecting only those pixels with which the RGB values exist within those boundaries.

E.g., let's set a lower boundary of R = 0, G = 0, B = 0 and an upper boundary of R = 255, G = 150, B = 150. Our boundaries are therefore:

Lower limit = [0, 0, 0]
Upper limit = [255, 150, 150]

We pass through each pixel in the image, test whether it's three values exist within this range, and remove (i.e. set its pixel value to [0, 0, 0]) any pixel outside of this range. What should we expect to see with the above limits applied? Try it in the GUI!

Colour segmentation can be a useful pre-processing task in order to extract regions of interest for which further processing can be performed on.

Edge Detection

The edge detector deployed here is called the Canny Edge Detector, after its founder John Canny in 1986. It consists of 3 principle steps (often more can be used but these are the most important). The steps are as follows:

1. *Filter out noise.* To filter out noise in an image, a number or convolutions can be used. In this case the Gaussian filter is employed (see the help page on smoothing and blurring).

2. *Intensity gradients*. This involves determining how the intensity of pixel values change as you move across the pixels in an image. The method used for this operation is called Sobel, and the gradients in the *x* and *y* directions are a summed computation of the following:

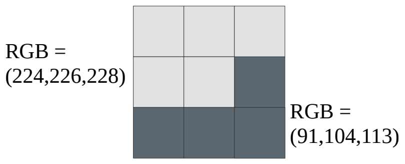
$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}, G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix}$$

With this, the gradient strength and direction can be found:

$$G = \sqrt{G_x^2 + G_y^2} \quad \theta = \arctan\left(\frac{G_y}{G_x}\right)$$

Direction is rounded to the nearest multiple of 45^o (i.e. 0^o, 45^o, 90^o or 135^o).

E.g. let us take a 3x3 window at a point that marks the boundary between MiRo's white outer shell and its grey underbelly. We may have the following (simplified and with lines added to differentiate pixels):



in which there are two distinct colours with RGB values as shown above. We have therefore 3 *planes*, or dimensions, with which we could perform the operation on, relating to each colour channel (alternatively one could convert the image first to greyscale and then perform these operations with only one colour channel). Let's try with the red colour channel. Performing the convolutions as proposed and summing the weightings would give:

$$G_x = \begin{bmatrix} -1\times 224 & 0\times 224 & +1\times 224 \\ -2\times 224 & 0\times 224 & +2\times 91 \\ -1\times 91 & 0\times 91 & +1\times 91 \end{bmatrix} = -266$$

$$G_y = \begin{bmatrix} -1\times 224 & -2\times 224 & -1\times 224 \\ 0\times 224 & 0\times 224 & 0\times 91 \\ +1\times 91 & +2\times 91 & +1\times 91 \end{bmatrix} = -308$$

$$G = \sqrt{(-266)^2 + (-308)^2} = 407$$

$$\theta = \arctan\left(\frac{308}{266}\right) = 45^o \ (rounded)$$

3. *Thresholding*. The two thresholds you entered in the main window are used here:
   a) If pixel gradient is above the upper threshold, it’s considered an edge.
   b) If pixel gradient is below the lower threshold, it’s rejected.
   c) If pixel gradient is between the two thresholds, it will only be considered an edge if a neighbouring pixel is above the threshold.

There are various ways in which edge detection can be performed, and a number of factors can affect the final result, from the selection of the smoothing operation, convolution weights, threshold boundaries, as well as the colour channels and/or whether you choose to operate on the greyscale image.

Average Smoothing and Gaussian Blurring

Smoothing/blurring is the process of determining a pixel value as a function of its neighbouring pixel values. It is often used as a pre-processing task in order to remove noise from an image. Two of the most popular methods for this is average smoothing and Gaussian blurring. Let us start with the simpler, average smoothing.

In average smoothing, the simplest of all smoothing operations, the candidate pixel is set to be the average of itself and its neighbouring pixels. To do this we can convolute an image with a kernel of the following form:

$$\frac{1}{k^2}\begin{bmatrix} 1_{11} & \cdots & 1_{1k} \\ \vdots & \ddots & \vdots \\ 1_{k1} & \cdots & 1_{kk} \end{bmatrix}$$

where *k* is the size of the kernel (and should be odd to give a centre pixel). As an example, suppose we choose our kernel to be of size 3, then *k = 3* and our convolution kernel will be:

$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

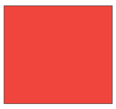
Let’s make the hypothetical case that a 3x3 window in an arbitrary image gives us the following pixel (RGB) values:

(243,123,112)	(243,97,82)	(240,65,61)
(237,23,36)	(245,180,120)	(240,65,61)
(237,28,36)	(237,28,15)	(237,12,13)

We see that the centre pixel is somewhat different to its neighbouring pixels; perhaps it have picked up some noise in the green and blue channels. But let us apply the above average convolution for each colour channel:

$R_{average}$ = *(243 + 243 + 240 + 237 + 245 + 240 + 237 + 237 + 237) / 9 =* ***240***
$G_{average}$ = *(123 + 97 + 65 + 23 + 180 + 65 + 28 + 28 + 12) / 9 =* ***69***
$B_{average}$ = *(112 + 82 + 61 + 36 + 120 + 61 + 36 + 15 + 13) / 9 =* ***60***

Giving us the final result of (240, 69, 60), thus removing some of the noise:



Gaussian smoothing is of a very similar process, but rather than giving equal weight to all neighbouring pixels, the weights are distributed in relation to their distance from the centre pixel: those that are further away are given less weight, and those that are closer more weight. Mathematically speaking, we adjust a parameter known as the *Standard Deviation (SD)* of the Gaussian. A larger SD means we apply more weight to those pixels further from the centre; an SD of 0 would mean we only consider the value of the centre pixel (i.e. there is no change in its value). Thus, for a 3x3 window again we may apply the following convolution (for those aware of the Gaussian function, the SD used in this case was 1):

$$\frac{1}{25}\begin{bmatrix} 2 & 3 & 2 \\ 3 & 5 & 3 \\ 2 & 3 & 2 \end{bmatrix}$$

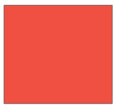
The *1 / 25* factor is used to normalise the convolution (i.e. ensure all values sum to 1). Applying this to the example image window shown above would result in the following:

$$R_{Gaussian} = (243 \text{ x } 2 + 243 \text{ x } 3 + 240 \text{ x } 2 + 237 \text{ x } 3 + 245 \text{ x } 5 + 240 \text{ x } 3 + 237 \text{ x } 2 + 237 \text{ x } 3 + 237 \text{ x } 2) / 25 = \mathbf{240}$$

$$G_{Gaussian} = (123 \text{ x } 2 + 97 \text{ x } 3 + 65 \text{ x } 2 + 23 \text{ x } 3 + 180 \text{ x } 5 + 65 \text{ x } 3 + 28 \text{ x } 2 + 28 \text{ x } 3 + 12 \text{ x } 2) / 25 = \mathbf{80}$$

$$B_{Gaussian} = (112 \text{ x } 2 + 82 \text{ x } 3 + 61 \text{ x } 2 + 36 \text{ x } 3 + 120 \text{ x } 5 + 61 \text{ x } 3 + 36 \text{ x } 2 + 15 \text{ x } 3 + 13 \text{ x } 2) / 25 = \mathbf{65}$$

Giving a final result of (240, 80, 65):



With this in mind, what differences would you expect on the amount of blurring between the average filter and the Gaussian, and why? At what value of SD does the Gaussian smoothing operation become an average smoothing operation?

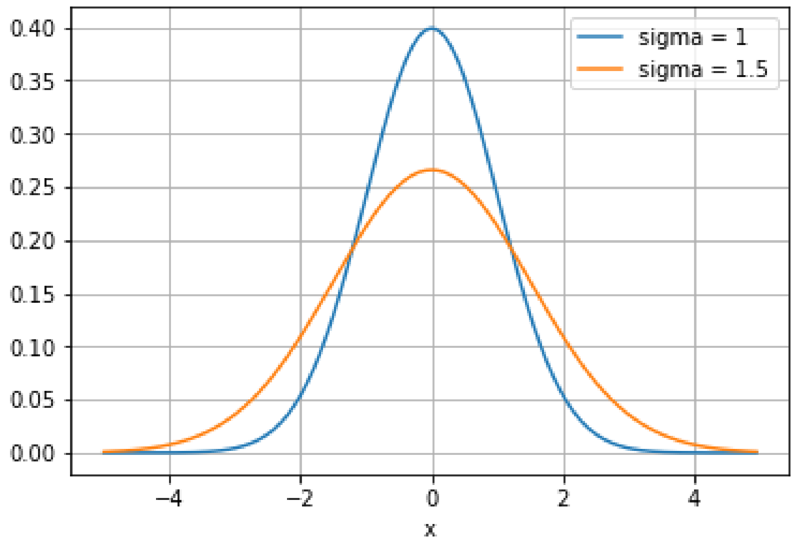
Difference of Gaussians

The Difference of Gaussians, or DoG, is a convolution that extends upon the standard Gaussian blurring operator. As the name suggests, the operator works by taking the difference of two Gaussian convolutions, both requiring different standard deviations (if they had the same, the difference would just be zero).

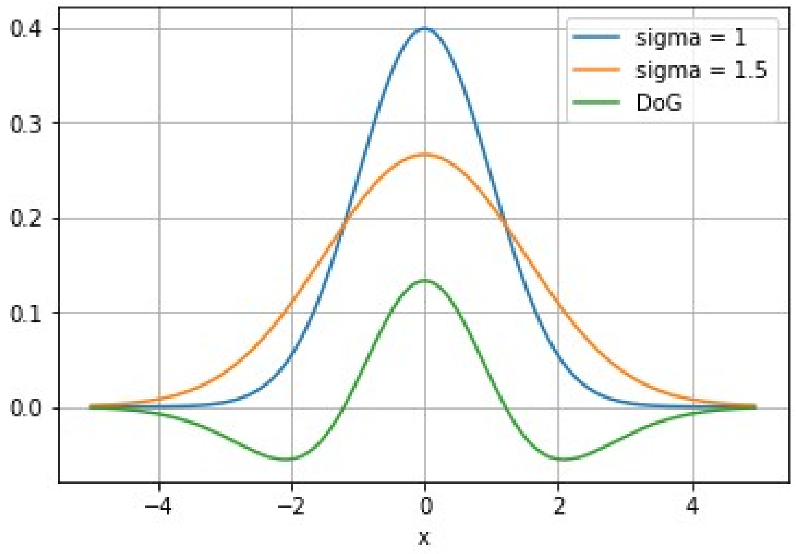
To understand this operator it's important to understand more about the Gaussian function generally. A one-dimensional Gaussian function centred on 0 takes the following form:

$$G_\sigma = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}}$$

σ is what we call the standard deviation, and determines the shape of the Gaussian, as you'll shortly see. If we plot the Gaussian as a function of *x*, and do this for two different values of σ, say $\sigma = 1$ and $\sigma = 1.5$, we get the following:



Taking the difference of these two Gaussians then gives us the iconic "Mexican Hat" plot. This is overlaid onto the original two plots, show below:



To convolve an image with a DoG filter, it’s simple enough to convolute the image independently with two standard Gaussians and then take the difference of the resulting two images.

What would be the result on the shape of the DoG if the standard deviation of the first Gaussian was larger than the standard deviation of the second Gaussian? I.e., if DoG = G(σ=1.2) – G(σ=1), for instance.

How would this affect the resulting convoluted image? (Tip, try using the GUI!)