# Algorithms & Code

For the application we took an adapted version of the Canny Edge Detection algorithm and combined it with our own approach inspired by the paper written by Son Lam Phung titled ‘Detecting people in images: An Edge Density Approach’.

## Gaussian Blur

We start with a Gaussian blur on each image in the test set which is written as a standalone SPU program. We use it like the Canny Edge Detection algorithm to remove background noise in the image which a Sobel operation is sensitive to. While we get the desired effect, a reduction of noise allowing real edges to be detected more easily we do suffer from the problem that a traditional Gaussian blur is computationally expensive and is actually the most expense SPU program we have running in our benchmarks. However, it does remove a lot of the noise making the next step easier.

## Sobel Filter

Once we have filtered out the noise in the image we perform a Sobel operation using a 3x3 kernel to measure pixel change magnitude to use it as a basis for detecting edges. The provided SPU program does this calculation and outputs a texture with all edges in white and anything else in black.

Even though Sobel is susceptible to noise we use it for many factors. First it is a simple algorithm that is very easily parallelised over multiple processing units. Also it involves matrix multiplication which allows us to take advantage of SIMD instructions to get even greater performance from the application.

Like the Canny Edge Detection algorithm, we do apply thresholding to remove so called weak edges and strengthen those that remain. The result is an image with all edges identified an output image we call the edge map.

## Window Edge Density

For calculating the size of the regions of interest in the edge map we use a method inspired by the paper referenced above. Here we scan through the image with a certain window size. We then calculate the edge density in total for that region. If it is above a certain threshold we mark the window as a region of interest. In our program we scan through the image using 45x45 pixel windows with step intervals of 12 pixels. Once completed we have a texture with regions painted in white with non-regions painted in black allowing us to now highlight the areas by overlaying it over the original image.

## Overlay

The final step is to overlay the detected regions of interest over the original image. This is a very simple operation and just involves darkening pixels that are not within regions of interest. Once that is done we write the new pixel data to disk as a 24-bit bitmap where you see the region of interests highlighted.

## Additional Point

There is an additional point to cover, why didn’t we implement the Canny Edge detector algorithm in full? Well put simply we sought to minimise the amount of SPU programs as creating an SPU context is an expensive process. Also a former student who was made available to question said that Hysteresis, the process of allowing weak edges to appear in the image if they are next to strong edges did little or nothing on the test set given. In short we choose to simplify the overall approach allowing for more optimisation and to reduce expensive SPU context creation.