3.3. Image Analysis

In some applications (particularly in research and process optimisation), more information is required than a single numeric value – but information that may be difficult to visually extract from slices or 3D visualisations. In these situations, image processing must be performed in order to extract the desired information from the reconstruction.

One such example may be measuring the flow rate of a stirred tank (or detecting the circulation loops). This could be achieved by adding some highly viscous blob to the tank, and using the tomography to follow its position within the tank. Measuring the circulation time would however require a person to review several frames, and the time intervals between when each was acquired. Blob detection (which enables fast motion tracking) can be a simple task for a computer to perform, provided there is sufficient contrast between the "blob" value* and the "background" value.

Descriptions and examples of some potentially useful algorithms are given below in the following subchapters. A practical combination of them is demonstrated. The following image (Figure 3-F) will be used as a starting point for demonstrations of various algorithms:

Figure 3-F: RAF Lynx wildcat (starting image for image processing demonstrations).

* In this case, the value of interest is conductivity.

3.3.1. Edge detection: Laplacian convolution kernel

There are various algorithms and kernels for edge-detection, one of the simplest is convolution with the Laplacian kernel:

$$\mathbf{L} \propto \begin{pmatrix} & -1 & \\ -1 & 4 & -1 \\ & -1 & \end{pmatrix} \qquad 3.1$$

Other more optimal kernels exists for edge-detection, such as the Sobel[52] and Scharr[53] kernels, however the Laplacian kernel is sufficient for this demonstration.

For each pixel in an image, convolution with the Laplacian kernel averages the values of the pixels immediately adjacent to the source pixel then subtracts that average from the source pixel, to produce the value for the corresponding pixel in the output image. Applying this kernel to a monochrome version of the image in Figure 3-F produces the following:

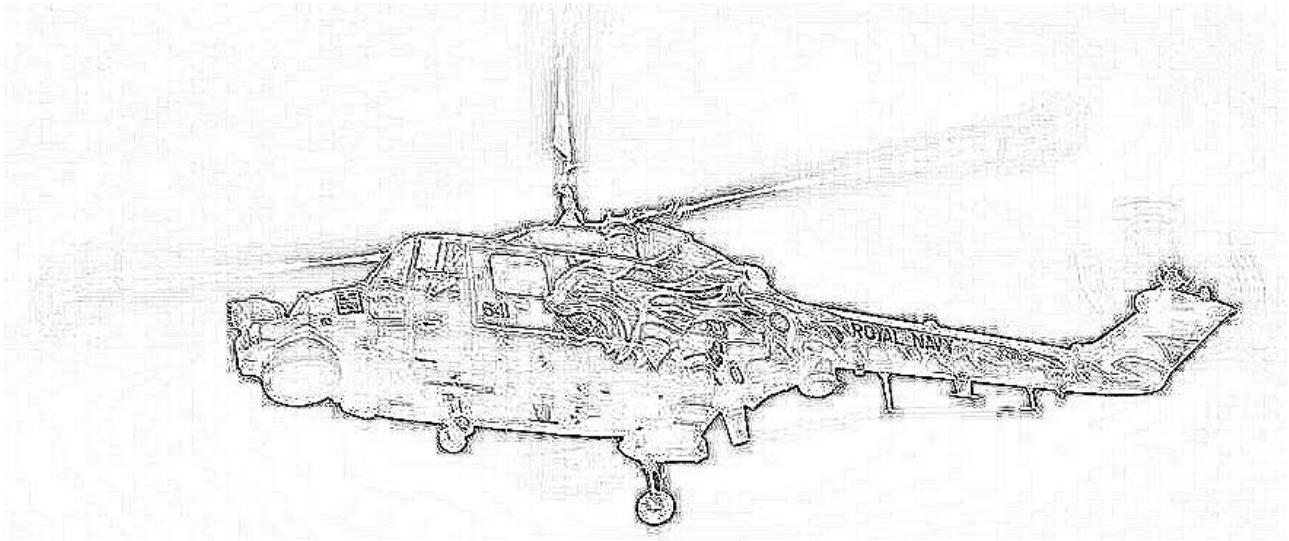

Figure 3-G: The sample image, with a Laplacian convolution kernel applied.

Because this filter compares pixels to their immediate neighbours, the resolution and sharpness of the input image strongly effects the output of the filter. A blurry, high-resolution image will result in weaker (lower contrast) edges than a scaled down version of the same image, when convolution-based edge detection is used. As reconstructed ERT images tend not to be sharp, this algorithm alone is therefore a poor choice for analysing ERT images. This problem can be resolved by using a high-pass spectral filter on the image instead of a convolution matrix. This is typically achieved with Fourier transform–based convolution. These apply a spectral envelope to the spatial frequency spectrum of the image.

3.3.2. Object-detection: Threshold

Provided there is sufficient contrast between the background and the objects of interest, or the background can be removed from the image*, a fast method to isolate objects is to apply a threshold to the image, set at a brightness level between those of the background and foreground – any pixels below this value become black, while any above become white. Determining where to set the threshold has a strong effect on the quality of the output. One way to determine this is to take a weighted average of the source image – weighted by the Laplacian-filtered image. This method calculates the "edge" value by averaging the pixels of the source image, and weighting each pixel by how intense it becomes when the Laplacian "edge-finding" filter is applied, thus rejecting values from smooth areas in the image (areas lacking edge detail).

Figure 3-H: Sample image with threshold filter applied.

This filter still produces a 2D image; it has not yet reduced the image to object positions, sizes, shapes or velocities. Additionally, since some pixels of the foreground are of similar intensity to the background, some of the "object" has not been detected by the filter, and the real "object" has been split into several pieces in the image. A human can infer the actual boundary from prior knowledge of how vehicles typically appear, however the algorithms presented in this chapter avoid requiring any prior knowledge of what the source images are supposed to represent.

* Temporal filtering is one such way to achieve this, provided the background is static. For photographic images, stereoscopic cameras or large apertures may be used to respectively provide depth information or to blur edges outside the focal plane of the image.

3.3.3. Blob-detection: Hoshen-Kopelman union-find algorithm

The image data forms two-dimensional scalar field over the real numbers. Each pixel falls into one of two none-overlapping (disjoint) sub-sets: the foreground or the background. The union-find class of algorithms group such data according to which sub-set each datum is a member of. In this case, we use a union-find algorithm (Hoshen-Kopelman) to group contiguous pixels (from the threshold-filtered image) of each subset, forming blobs of "foreground" and "background". Small blobs* (commonly termed "specks" or "holes") are removed, resulting in few large blobs being produced for each object:

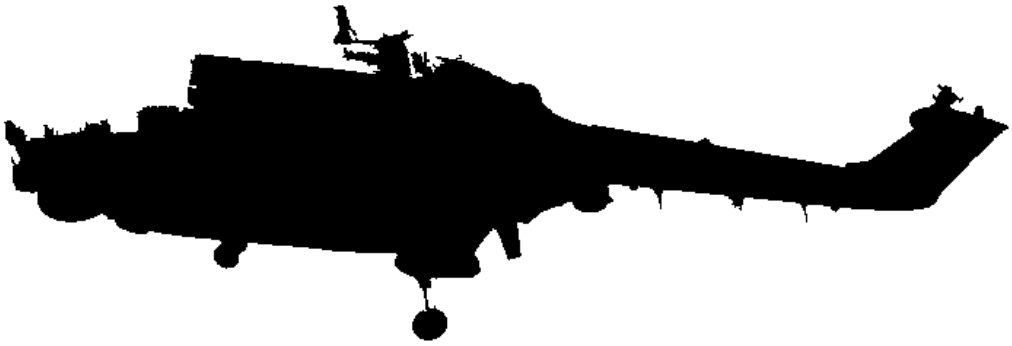

Figure 3-I: The Threshold-filtered image has been grouped by a Hoshen-Kopelman algorithm, and small groups have been removed from the image.

Notice that some parts of the foreground (which produce small blobs) have been removed. The blob-culling algorithm requires some customisation if it is to produce accurate results. Even with a simple blob-culling algorithm that imposes a minimum-area limit on detected blobs, most of the helicopter's shape has been recovered from the image. The computer is no longer holding image data – this image is rendered from a list, listing the pixels that were detected as "foreground", or tracing the outlines of foreground regions. To track an object across frames. the mean or the centroid of object pixel co-ordinates could be used to give a "visual centre of mass" for the position of the object, and the total pixel count (for the object) gives some indication of the object's size. Given a set of N pixel co-ordinates (x_i, y_i) for a foreground blob's pixels, the centroid of that blob is given by:

$$\left(c_x, c_y\right) = \left(\sum_{i=1}^{N} \frac{x_i}{N}, \sum_{i=1}^{N} \frac{y_i}{N}\right) \qquad 3.2$$

* i.e. blobs containing less than some arbitrary minimum amount of pixels

3.3.4. Demonstration of an object isolation technique based on the previous algorithms

To visualise the accuracy of the algorithm, the outline(s) of the detected blob(s) may be superimposed on the source image (Figure 3-J).

Figure 3-J: Sample image, with outline of what the computer "detects" as the object superimposed.

The rotors and windshield were removed by the blob-culling stage, as they resulted in small blobs after the Hoshen-Kopelman algorithm had been applied. The blob-culling stage is justified though, since most of the fuselage shape has been accurately represented despite some of the "foreground" within this region being of comparable brightness to the background (see Figure 3-H).

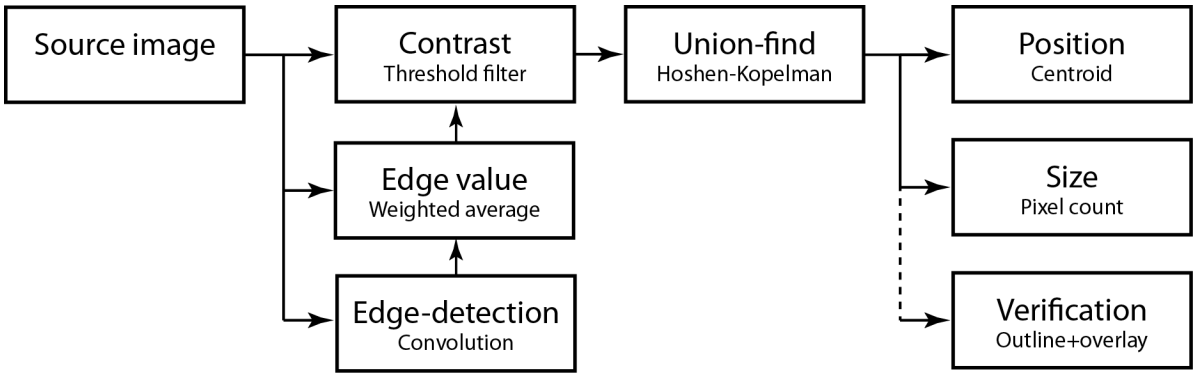

Figure 3-K: Flowchart for the described blob analysis method.

By applying this technique to a range of frames representing different times during a process (i.e. video), and analysing how blob positions have changed, motion tracking can be implemented. Inspection of blob sizes and analysis of how they change may also allow quantification of dissolution or mixing.

3.3.5. Feature-point tracking algorithms

Once suitable pre-processing has been performed on a sequence of images, to identify features of interest in them, the motion of these features may be inferred and analysed by *motion-estimation* or* *feature-tracking* algorithms. If the pre-processing stage provides information that can be directly used to identify and distinguish between various features of interest, then motion can be tracked simply by analysing the positions of features in different frames. If multiple features of the same type can appear (e.g. bubbles of a certain size), or different features cannot be distinguished reliably after pre-processing then more advanced algorithms may be used that, given a certain number of initialisation frames, will choose which of a set of features in a subsequent frame corresponds to the feature identified in the previous frames based on how physically probable the implied motion path is. Amongst other assumptions, this often includes imposing a limit on the maximum acceleration between frames, which is reasonable provided that extremely strong but short-term impulses are unlikely to occur in the physical system.

Many of these algorithms were developed for computer vision, so are designed to process 2D projections of 3D worlds – hence they also handle consequences of the projection such as occlusion, out-of-view, and may also provide depth estimation.[54] As tomography provides full three-dimensional images of closed systems, these extra features are not necessary, and therefore not relevant for comparisons between the algorithms. Whilst removal of these features may reduce the computation cost of these algorithms, they will be analysing full three-dimensional images which typically contain considerably more data than two-dimensional images.

A detailed description, comparison, and animated demonstration of some motion estimation algorithms is presented in source #54. One such demonstration shows two different algorithms being (separately) used to create a vector field (presumably, a velocity field) from a video of particles being carried by a slightly turbulent fluid flow.

* These names are interchangeable.