Unit 4. Frame Processes

4.1 Why Frame Fusion is Needed

There are two basic reasons for fusing two or more frames into a single image frame. First, there may be multiple images of the same scene that each contain noise. By averaging these frames pixelwise into a single one, the noise is reduced and the details are strengthened. Other methods of combining them are possible, such as the pixelwise medians or other typical or representative values for each pixel.

Second, there may be available images taken in different bands of visible light or infrared.Although the scene is the same, each image is different because different wavelenghts of light are absorbed or reflected by the different objects in the scene. By adding, subtracting, takeing ratios, etc., we can process these multiple images from multiple spectral bands into a single image that is enhanced for seeing certain features. An example of the first type of situation would be the combination of multiple ultrasonic images of the same object into an image that has less noise and more detail. Other examples are in electron microscopy and medical imaging systems.

Other examples of the second type of fusion arise when cameras aboard aircraft or satellites are equipped with filters that each passes a particular band of wavelength. The Landsat satellites that orbit the earth take pictures for detecting the presence of crops, minerals, and other land uses. Other satellites, such as the GOES, gather images that are used for weather monitoring and prediction. Orbiting satellites scan every square mile of the earth regularly to detect frost, analyze crop yields, detect the formation of storms, etc. The earth images are processed for enhancement, restoration and analysis.

Another example of the second type uses *hyperspectral* images that are taken at the same time from the same point in time and space but in several different short bands of wavelength. The AVIRIS (Airborne Visible/Infrared Imaging Spectrometer) captures over 200 images in different bands. Each pixel has an intensity in each of these bands, so a graph can be made of the intensity in each band, which is called a *signature* for that pixel. A pixel may cover a square meter or a few square meters and so its intensities in the different bands should show a mixture of materials on the ground and its signature should show that.

4.2 Frame Registration for the Fusion of Images

Translational Misalignment. While two images may contain the same scene, they may not be exactly identical. The second may be shifted leftward, rightward, upward or downward one or more pixels or a combination of these shifts. Such a translation of one image from the other is the simplest type of difference. When we *register* the first (*input*) image with respect to the other (*base*) image, we find the x and y shifts that yield the best match according to some criterion. In other words, we shift an image up and down, left and right with respect to another one to align them so that the pixel differences are the smallest on the average. This does not work for more complex differences between the images.

For this simple misalignment, we can adjust the input image by shifting it in the x and y directions until an error measure is minimized. One useful error measure computes the pixelwise squared difference of the two images to yields a third error image as

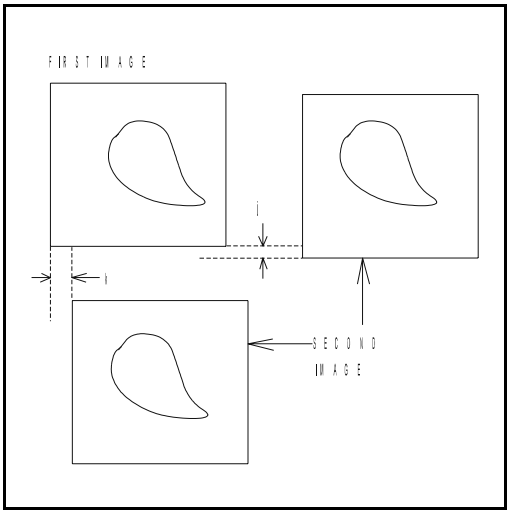
$$f_e(m,n) = [f_{in}(m,n) - f_{base}(m,n)]^2 \tag{4.1}$$

Upon summing these over m and n we obtain the sum-squared error, which may be rather large for images out of alignment. Thus we standardize closeness of fit by using the *relative root summed-squared error* (RRSSE)

$$\epsilon_{12}(j,k) = \{\alpha_{(n=0,N-1)}\alpha_{(m=0,M-1)} [f_1(m,n) - f_2(m-j,n-k)]^2 \,/\, \alpha_{(m=0,M-1)} [f_1(m,n)]^2\}^{1/2} \tag{4.2}$$

If two images are aligned exactly, then they are the same except for noise, which is usually zero-mean Gaussian noise. However, the square of this measure of difference is of second order and under perfect alignment is a measure of the relative power of the noise. This is analogous to the squaring of a voltage v to obtain the power P in watts per ohm resistance via $P = vi = v(v/R) = v^2/R = v^2$, where R = 1 ohm). As the misalignment increases, this measure adds error to the noise power and thus grows larger. When $j = j_0$ and $k = k_0$ such that the RRSSE is minimal, then this shift provides the optimal alignment in the mean-square sense. Figure 4.1 shows the translational misalignment of two simple images.

Figure 4.1. Registration of images.



Let $f_{in}(m,n)$ be the input and let $f_{base}(m,n)$ be the base image, where both are of the same size MxN. The two images may have been taken by the same satellite at the same time in different wavelength bands, at different times by the same satellite, or even on different days. The images could be medical images taken by two different machines or be the same one at two different times. One could be an X-ray image before the injection of dyes and the other one could be an X-ray taken after the injection of dyes. With small distortions allowable, the differences are shifts. A more substantial example is a sequence of X-rays that must be aligned to show the sequence of heart beats. Most registration problems, however, are more than translational misalignment.

Nonlinear Misalignment. But two images of the same scene may have more serious differences. They may have been captured from different spatial points in different directions with different magnifications, in which case the first one is a nonlinear distortion of the base image. This is the most difficult type of registration problem. To register the input image with the base image, we can use polynomial transformations of four pairs of points where each pair contains one point in the input and a corresponding point in the base image that are the same geophysical point in the scene. More points would introduce more local optima and "wiggles" that could distort the map.

Given the four corresponding pairs of points in the two images, we obtain a mapping $(m_1,n_1),...,(m_4,n_4)$ from the inpujt to the base image that maps each (m,n) to $(\tilde{m},\tilde{n})$ in the output full block via

$$\tilde{m} = a_1m + b_1n + c_1mn + d_1, \qquad \tilde{n} = a_2m + b_2n + c_2mn + d_2 \tag{4.3}$$

The two equations for each of 4 pairs of points makes 8 linear equations in the 8 unknowns $a_1, b_1, c_1, d_1, a_2, b_2, c_2, d_2$ ($mn = p$ is a coefficient). More terms and points should not be necessary except in very distorted cases. The output image f(m~,n~) is not the registered version of the input image with the base image.

Registration with Matlab. Matlab has functions that can be used to perform nonlinear registration of the input image with a base image. The most important function is *cpselect()* where the arguments are the names of the input and base images. When we enter this at the command line in the command window of Matlab, the *Control Point Selection Tool* window comes up (see Fig. 4.7 below) with its four sub-windows: the two input and base *detail* windows on top and the two input and base *overview* windows on the bottom. The base window is on the right in both cases. The size of the detail images can be changed by selecting at the top and the scroll bars can be used to move the image to see all of them. There is also a *hand-slider* (like Adobe Acrobat Reader uses) for sliding the images for better viewing. Clicking the lock box at the top causes any movement of one image to apply to the other.

To register the input image with the base we must select (usually) four points in one detail image, either the input or base image, one at a time, and for each one we select, we must next select one in the other image that corresponds to the same physical point in the scene. Thus we obtain four corresponding pairs of points. To begin selecting the points, we must find on the toolbar at the top under the menu bar the button that contains a circled plus sign ("+"). When we click this then Matlab goes into the point selection mode and we can click

on a feature point in the input image and then click on a point in the base image that is the same feature point. This creates a pair. We select four pairs in the usual case.

The user must now save the control points with names for the input and base images, which is done by selecting on the menu bar *File*, and then *Save Points to Workspace*. In the ensuing pop-up box the user must type in names for the saved points, which we call *input-points* and *base-points*. We then select the *Structure with all points* square and enter the name of the data structure that holds the point: we use *cpstruct*. Next, we click on the **OK** button to close the pop-up menu and continue with the nonlinear transformation to register the input with the base image.

A Matlab Registration Example. We first bring up Matlab and then type in the following commands. Note that we resize the first image to serve as the base.

```
>> Input = imread('shuttle.tif');              // read in shuttle image
>> Ibase = imresize(Input, 1.25);              // resize it to serve as base image
>> Input2 = imrotate(Input, 15, 'bilinear');   // rotate 15 degrees, bilinear interpolate
>> cpselect(Input(:,:,1), Ibase);              // call control point routine
```

[The Control Point Selection Tool window comes up here. We select 4 pairs of control points as described above and close the menu.]

```
>> mytform = cp2tform(input_points, base_points, 'projective');
                                               // create transformation with control points
>> Ireg = imtransform(Input2, mytform);        // transform/register input image
>> imshow(Input);                              // show input image
>> figure, imshow(Ibase);                      // show base image as new figure
>> figure, imshow(Ireg);                       // show registered image as new figure
```

[The first registration did an approximate job, but we now register the result again with the same base image for a better approximation]

```
>> cpselect(Ireg(:,:,1), Ibase);               // call cp routine w/registered image as input
```

[Again, we work with the Control Point Selection Tool window and then close it]

```
>> mytform = cp2transform(input_points, base_points, 'projective');
                                               // create new transformation with new cntrl pts.
>> Ireg2 = imtransform(Ireg, mytform);         // transform/register previously reg. image
>> figure, imshow(Ireg2);                      // show new figure with new regstered image
```

Fig. 4.2. The original image.

Figure 4.3. The rotated image.

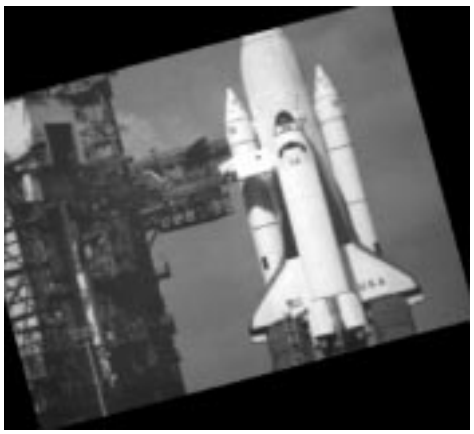


Fig. 4.4. The enlarged base image.

Figure 4.5. Registered rotated image.

Fig. 4.6. The twice registered image.

Figure 4.2 shows the original image *shuttle.tif* and Fig. 4.3 shows the result of rotating it 15° in the counter-clockwise direction to form the input image. Figure 4.4 displays the enlargement of the original image by a factor of 1.25 that is the base image. After putting in the commands for Matlab listed above we obtained Figure 4.5, which is a good first approximation to the enlarged base image (the rotation has been taken out by the polynomial mapping). However, the left side is not as high as the right side so the right side is stretched. We apply another round of registration to Fig. 4.5 to arrive at Fig. 4.6. We could do another round to tune the registration, but we have made the point that repeated registration (up to a point) improves it.

Fig. 4.7. The Matlab *Control Point Selection Tool* window.

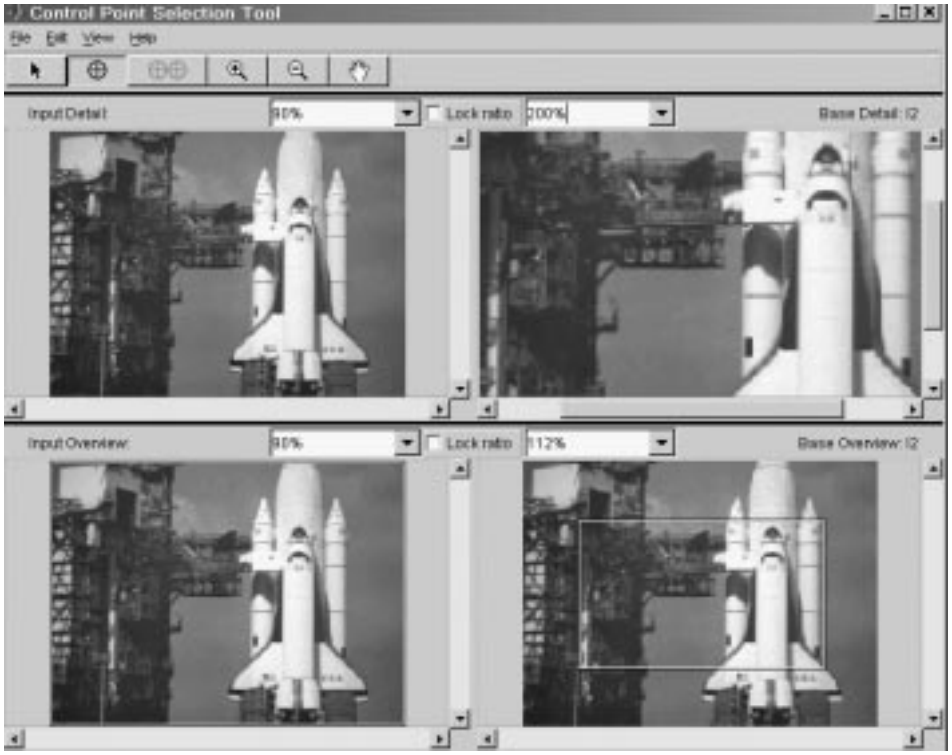


Figure 4.7 shows the Control Point Selection Tool window with the original *shuttle.tif* as the input and a copy that is reduced in size to 0.8 of the original as the base image. The input and base windows, as well as the detailed and overview ones are shown. The circled plus button is down for selecting points in either the input or base image, but whichever is used, the next selection must be from the opposite image to create a pair of corresponding points. After this window is closed, the function *cp2tform()* must be used to create a transformation (mapping) that we called mytform above. This mapping is then applied with the function *imtransform()* as was done above.

Generating Examples with Matlab by Cropping. We created an example above by resizing and rotation. These are critical to creating good examples, but we also need to be able to crop an image with Matlab. The following example shows the use of all three functions mentioned here.

```
>> I1 = imread('shuttle.tif');          // read in image to location I1
>> I2 = imrotate(I1, 30);               // rotate image 30 degrees and store at I2
>> imshow(I2);                          // show the rotated image
>> I3 = imcrop;                         // call tool to crop displayed image I2
```

[At this point we click a point in the image to be the upper lefthand corner of a rectangle and drag down to the lower righthand corner and release the mouse button. The rectangular area is stored at location I3 in memory. Now we display the new image I3 and enlarge it.]

```
>> figure, imshow(I3);                  // show cropped image, which may be small
>> I4 = imresize(I3, 2.0);              // double size of I3 and store as I4
>> figure, imshow(I4);                  // show the enlarged cropped image in new figure
```

Figure 4.8. A cropped rectangle from an image.

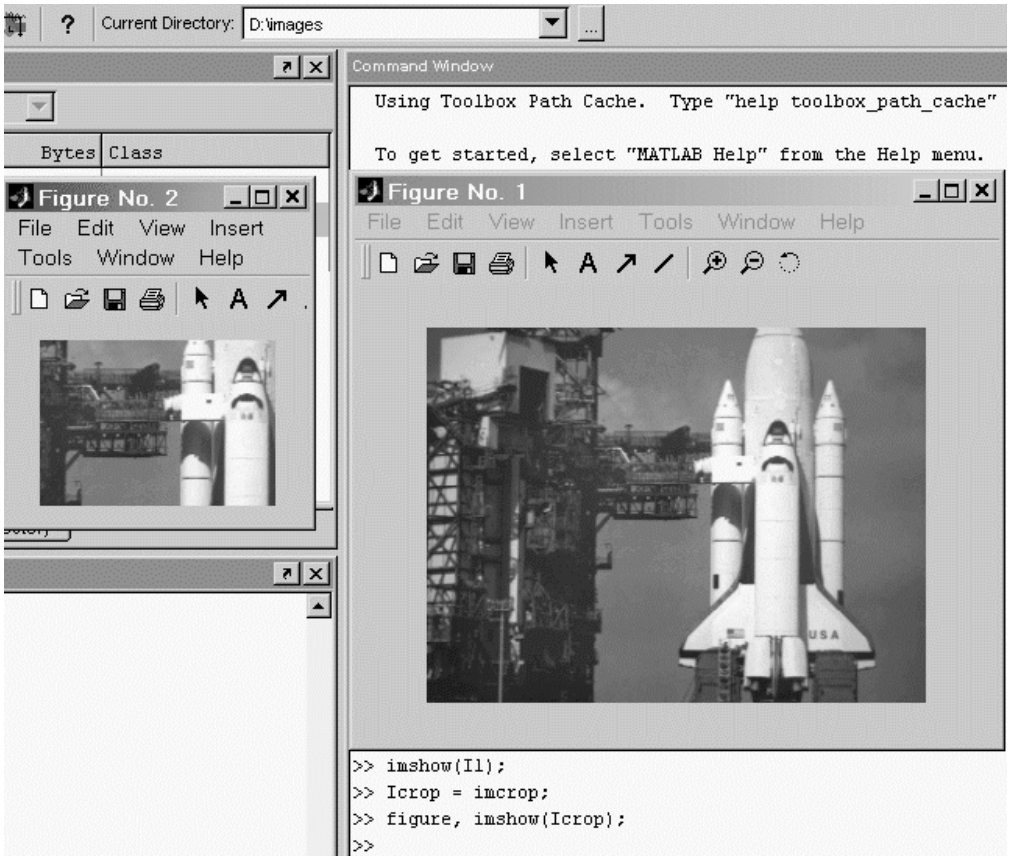


Figure 4.8 shows a section of the screen when the function *imcrop* is used on an image I that is displayed on the screen and is in focus (multiple images and windows may be displayed on the screen at one time, but only one is in focus, denoted by its menu bar being brightened). Immediately after the imshow(I) function is executed, the image I is in focus (otherwise we must click on it), in which case its top bar (above the menu bar) is brightened.

4.3 Pointwise Fusion of Images

The Pointwise Process. Given two registered images $\{f_1(m,n)\}$ and $\{f_2(m,n)\}$ there are many ways to combine two pixels at the same locations in the two different images. At pixel location (i,j) in each image we can use most any kind of arithmetic or logical operation to combine the two pixels $f_1(i,j)\}$ and $\{f_2(i,j)$ into an output pixel, where the operation is denoted by "$\diamond$."

$$f_{out}(i,j) = f_1(i,j) \diamond f_2(i,j) \tag{4.4}$$

The operation $\diamond$ can represent such operations as: i) addition; ii) weighted averaging; iii) subtraction; iv) multiplication; v) minimum; vi) maximum; vii) ratio of logarithm of pixel values; viii) AND, OR or XOR logic (for binary, or black and white, images); and others that may include combinations of these operations. Also, decisions can be made at each pixel location as to which of the two pixels from the two different images at that location is to be used as the output pixel.

Example Using Matlab to Remove Background. In the following example we have the original *shuttle.tif* image. We employ the function *graythresh()* to find a good greylevel to use as the threshold, followed by the function *im2bw()* to convert the image to black and white. All pixels below the threshold are converted to black while all pixels above the threshold are converted to white. Using this black and white image to multiply the original image, we obtain a new image that has all background pixels below the threshold blanked out, while all pixels above the threshold are an appropriate light shade. To complete the process, we stretch the contrast.

```
>> clear, close all;                    // clear all data structures from memory
>> I1 = imread('shuttle.tif');          // read in original image
>> imshow(I1);                  // show image on screen
>> T = graythresh(I1);                  // find good graylevel T for threshold
>> Ibw = im2bw(I1, T);                  // convert I1 to black & white with threshold T
>> figure, imshow(Ibw);                 // show the black and white image
>> I2 = immultiply(I1, Ibw);            // multiply original by black and white image
>> figure, imshow(I2);                  // show result of mulltiplication
>> I3 = imadjust(I2, [0.3 0.9], [0 1]); // contrast stretch image I2
>> figure, imshow(I3);                  // show final image as new figure
```

Fig. 4.9. Original image.

Fig. 4.10. Black and white image.

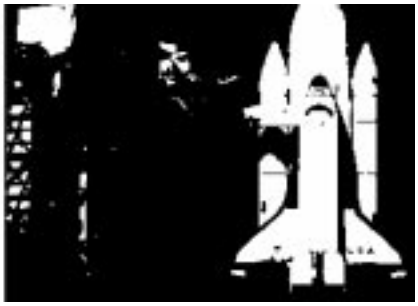
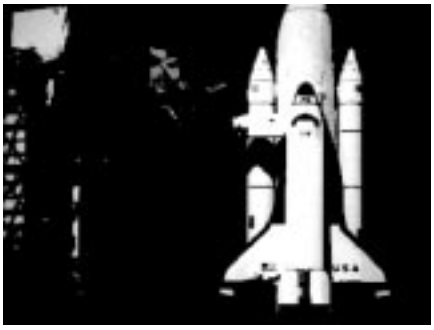


Fig. 4.11. Orig. multiplied by b&w.

Fig. 4.12. Contrast stretched final image.



Frame Averaging. A simple way to combine two or more frames is to average them pixelwise. We use two frames $f_1(m,n)$ and $f_2(m,n)$ here, but the process is the same for more than two images. We assume that the second image has been registered against the first one and designate the output image by $f_3(m,n)$. Then

$$f_{out}(m,n) = \alpha f_1(m,n) + (1-\alpha)f_2(m,n) \tag{4.5}$$

When $\alpha = 0.5$ the output image is just the pixelwise arithmetic mean. However, if the first image is considered to be a little better than the second one, then α can be increased to, say, 0.6, etc. For R frames, α can vary from 1/R. There are other ways to average the pixels at each location. For example, the fuzzy weighted average that is described later. For more than two frames, the pixelwise median can be used in place of averages.

The *alpha-trimmed mean* can also be used for R > 2: for small integer α, throw out the highest α values and the lowest α pixel values at each pixel location, and then average the remaining ones. This eliminates the "snake swallowing an egg" effect of outliers and provides an average of the "good" pixel values. We modify the method and count the middle pixel twice to help stabilize the result.

Differencing. This is a useful operation in certain situations. We take

$$f_{out}(m,m) = f_1(m,n) - f_2(m,n) \tag{4.6}$$

as the difference. One purpose is to *remove background* from an image to better expose details. Another is to *equalize illumination* in microscopic images, where a background image that shows the nonuniformly distributed illumination is subtracted. The process called *digital subtraction angiography* subtracts an image of heart tissue from another where a dye has been added to the blood. This takes out the obscurring parts to enhance the blood vessels in the image. Another differencing application is motion detection given later.

Ratioing. This is used in special situations. An example is in the detection of alive or dead vegetation among vegetation that is both dead and alive. It is known that alive vegetation gives off considerable energy in the infrared band but reflects very little energy in the red color band. On the other hand, it is also known that dead vegetation reflects red band energy but gives off very little infrared. Let $\{f_1(m,n)\}$ be an image taken in the infrared band, and let $\{f_2(m,n)\}$ be an image taken in the red color (visible light) band. Let

$$g_1(m,n) = \log[f_1(m,n)/f_2(m,n)] = \log[f_1(m,n)] - \log[f_2(m,n)] \tag{4.7a}$$

$$g_2(m,n) = \log[f_1(m,n)/f_2(m,n)] = \log[f_1(m,n)] - \log[f_2(m,n)] \tag{4.7b}$$

The first image, $g_1(m,n)$, is increased in illumination (intensity) because the pixels $f_1(m,n)$ of greater infrared intensity are divided by the low level pixel values of $f_2(m,n)$ that do not radiate much infrared. Thus the first image enhances the alive vegetation. Analogously, the second image enhances the dead vegetation. The logarithm keeps the results reasonable, but the output images must be scaled so as not to exceed 255.

Image compositing is a process where a part (usually an object) of an image $f_1(m,n)$ can be segmented. All pixels outside of the boundary are set to zero in the output image $g_1(m,n)$. Thus the output image contains only the object on a black background. This image can be added to any other image $f_2(m,n)$ that forms a background. The new output image is then

$$g_2(m,n) = f_2(m,n) + g_1(m,n) \tag{4.8}$$

Such techniques are used to show, for example, a new car sitting on a mountain with steep cliffs all around it where the car could not be put except with a freight helicopter. We can also physically cut part of an image with scissors and scan it on a black background. The result can be added to another image that contains the desired backgound. For example, a city could be put on an underwater background.

4.4 Pixel Fusion Based on Areas

The Area Fusion Process. In this process we have two pixels at the same location from two registered images. Here, however, we use the areas around the two pixels to make a decision as to what the output pixel in the fused image should be. This provides much more power than the pointwise pixel fusion methods. The area is usually a nbhd of a pixel that occupies the central location of the nbhd. In other words, we compute the output pixel value at location (m,n) to be a function of the nbhd of the pixel at that location.

$$f_{out}(m,n) = T[nbhd(f_{in}(m,n))] \tag{4.9}$$

Spatial Frequency Fusion.

Maximum Difference Fusion.

Multispectral Image Fusion.

4.6. Hyperspectral Image Fusion

4.7 Motion Detection.

Time Sequences of Frames. A video type of camera produces a sequence of frames over time. By comparing the frames by differencing, motion of any objects in the scene can be detected.

Motion detection can be performed by working with consecutive images taken over a sequence of time points. Given the sequence $\{f_1(m,n)\}$, $\{f_2(m,n)\}$,..., $\{f_p(m,n)\}$ of images, we subtract each one from the next consecutive one to take out all pixels that are the same. The output magnitude of differences is essentially a zero image except for noise and the pixels in objects that have moved. A connected blob of pixels that are not zeroed out is an object that has moved. This works very well for objects of different intensity than the background.

A test can be made for the significance of the difference in power (using the sum of the pixel values or their squares). The threshold for the decision (it moved or did not move) can be adjusted by observing the operation online and incrementing the threshold. By using a sequence of consecutive image pairs, and computing the center of gravity of connected blobs in the difference image, moving objects can be *tracked*, that is, a trajectory of the moving object can be constructed.

4.8 Exercises