Unit 4. Frame Processes

4.1 Why Frame Operations are Needed

There are three basic reasons for operations on two or more *frames* (images). First, there may be multiple images of the same scene that each contains 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 using the pixelwise medians or other typical or representative values for each pixel. An example of this 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 can be found in electron microscopy and medical imaging systems.

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, taking ratios, etc., we can process these multiple images from multiple spectral bands into a single image may better expose certain features.

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 taken by satellite are in different optic bands or may be from high resolution synthetic aperture radar.

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 different materials on the ground and its signature should show that mixture.

The third type of frame processing is for special purposes, such as differencing two consecutive frames from a surveillance video camera to detect motion (if any objects have moved). Other processes remove or add objects to a scene or superimpose features such as texture.

4.2 Frame Registration for Image Fusion

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* one (*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 total pixel differences are the smallest in some sense. 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 yield a difference image

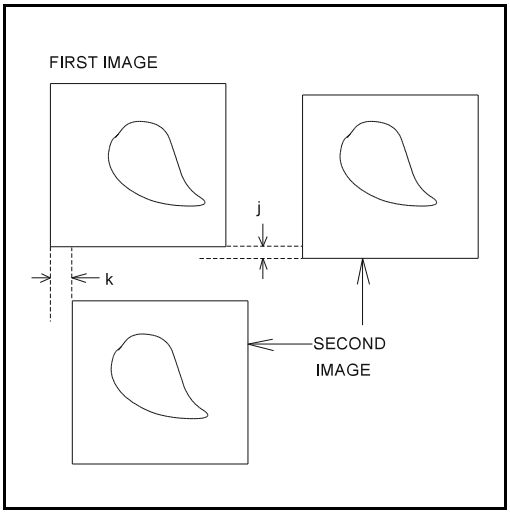
$$f_{diff}(m,n) = [f_{in}(m,n) - f_{base}(m,n)]^2 \tag{4.1}$$

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)

$$\in(j,k) = \{\sum_{(n=0,N-1)}\sum_{(m=0,M-1)} [f_{in}(m,n) - f_{base}(m-j,n-k)]^2 \,/\, \sum_{(m=0,M-1)} [f_{base}(m,n)]^2\}^{1/2} \tag{4.2}$$

If two images are aligned exactly, then there may be differences due to noise, which is usually zero-mean Gaussian noise. However, the sum-squared error 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 and i is the current. As the misalignment increases, this alignment error adds to the noise power. 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. Translational misalignment.

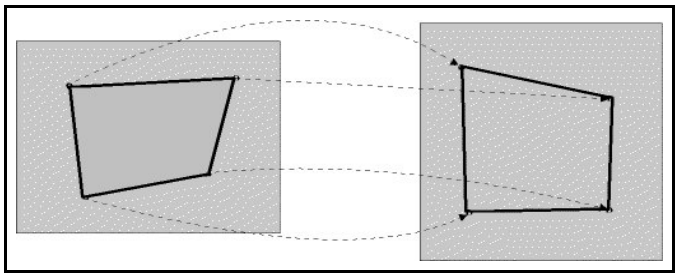


Let $f_{in}(m,n)$ be the input and let $f_{base}(m,n)$ be the base image. The two images may have been taken by the same satellite at the same time in different wavelength bands, at different times by the same satellite, or even on different days. The images could be two medical images taken by two different machines or be taken by 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.

Registration for Nonlinear Misalignment. Two images of the same scene may have more serious differences. They may have been captured from different spatial points or with different magnifications, in which case the first image 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 one corresponding point in the base image where each point in the pair represents the same physical point in the scene. More point pairs require higher degree polynomials that introduce more local optima and "wiggles" that could distort the mapping. Figure 4.2 displays a situation where four points are mapped into four points in another image in a nonlinear fashion.

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 here). The output image $\{f_{out}(m\sim,n\sim)\}$ is the registered version of the input image (registered with respect to the base image). Figure 4.2 shows a polynomial mapping of four points.

Figure 4.2. A 4-point polynomial mapping.



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 input to the base image that maps each (m,n) to $(m\tilde{},n\tilde{})$ in the output image via

$$m\tilde{} = a_1m + b_1n + c_1mn + d_1$$

$$n\tilde{} = a_2m + b_2n + c_2mn + d_2 \qquad (4.3a)$$

Substituting in the four pairs of points we obtain the following 8 equations.

$$m_1^{\sim} = a_1 m_1 + b_1 n_1 + c_1 m_1 n_1 + d_1, \qquad n_1^{\sim} = a_2 m_1 + b_2 n_1 + c_2 m_1 n_1 + d_2 \tag{4.3b}$$

$$m_2^{\sim} = a_1 m_2 + b_1 n_2 + c_1 m_2 n_2 + d_1, \qquad n_2^{\sim} = a_2 m_2 + b_2 n_2 + c_2 m_2 n_2 + d_2 \tag{4.3c}$$

$$m_3^{\sim} = a_1 m_3 + b_1 n_3 + c_1 m_3 n_3 + d_1, \qquad n_3^{\sim} = a_2 m_3 + b_2 n_3 + c_2 m_3 n_3 + d_2 \tag{4.3d}$$

$$m_4^{\sim} = a_1 m_4 + b_1 n_4 + c_1 m_4 n_4 + d_1, \qquad n_4^{\sim} = a_2 m_4 + b_2 n_4 + c_2 m_4 n_4 + d_2 \tag{4.3e}$$

***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()* that lets the user select the pairs of control points, where the arguments of the function 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. 4.8 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 input window is on the left 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 parts. There is also a *hand-slider* (like Adobe Acrobat Reader uses) for clicking on and sliding the images for better viewing. Checking the Lock Ratio box locks magnification of the input and base images so that changing one changes the other proportionately.

To register the input image with the base we select four points (four here, but we could select more) in the detail images, one point at a time either in the input or base image and for each point selected, we must next select one that corresponds to the same physical point in the scene in the next window. Thus we obtain four pairs of corresponding points. To begin selecting the points, we click the toolbar (at the top, under the menu bar) on the button that contains a circled plus sign (⊕). 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 to create a pair. We select four pairs here, but we could select more. It is difficult to select exactly corresponding points in small images unless we first magnify them.

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

A* Matlab *Registration Example. We first bring up Matlab and then we will type in the appropriate commands. We resize the original image to serve as the base image with a different size and we also rotate the original image to form the input image.

```
>> Iorig = imread(`shuttle.tif´);              // read in original shuttle image as Iorig
>> Ibase = imresize(Iorig, 1.25);              // resize it to serve as base image
>> Input = imrotate(Iorig, 15, `bilinear´);    // rotate 15 deg., bilinear interpolate for input im.
>> cpselect(Input(:,:,1), Ibase);              // call Control Point Selection Tool
```

The *Control Point Selection Tool* window comes up here. We select 4 pairs of control points as described in the above discussion and close the menu. Figure 4.3 shows the original image *shuttle.tif* and Figure 4.4 displays the rotated image that serves as the input image to be registered with the base image. Figure 4.5 is the resized base image that is 25% larger than the original. We now define a transformation that we give the name *mytform* that uses the control points. To do this we use the function *cp2tform()*.

```
>> mytform = cp2tform(input_points, base_points, 'projective'); // create transf. w/control points
>> Ireg = imtransform(Input, mytform);         // transf. input image w/mytform
>> imshow(Input);                              // show input image, Fig. 4.4
>> figure, imshow(Ibase);                      // show base image, Fig. 4.5
>> figure, imshow(Ireg);                       // show registered image, Fig. 4.6
```

When we register the input image with respect to the base image we obtain the image shown in Figure 4.6. It transforms to be somewhat like the base image, but it is not transformed exactly because of error in the points selected. Therefore, we will register the image of Figure 4.6 again with respect to Figure 4.5. This time we obtain Figure 4.7, which appears to be a reasonable registered image.

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

```
>> cpselect(Ireg(:,:,1), Ibase);                          // call cp routine w/registered image as input
>> mytform = cp2transform(input_points, base_points, `projective´);  //create transformation
>> Ireg2 = imtransform(Ireg, mytform);                    // transform/register previously reg. image
>> figure, imshow(Ireg2);                                 // show new figure with new regstered image
```

Fig. 4.3. The original image.

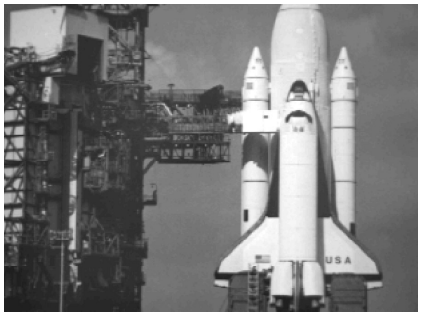


Figure 4.4. The rotated image as *input*.

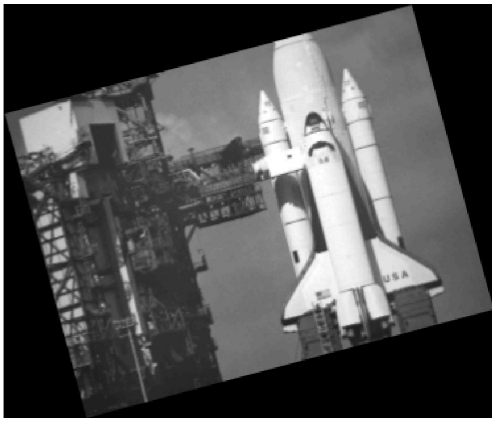


Fig. 4.5. The *base* image.

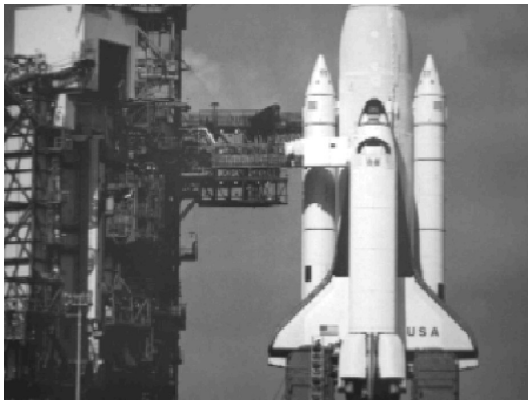


Figure 4.6. Registered rotated image.

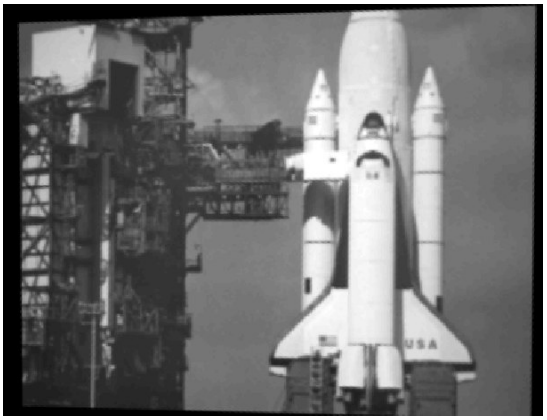


Fig. 4.7. The twice registered image.

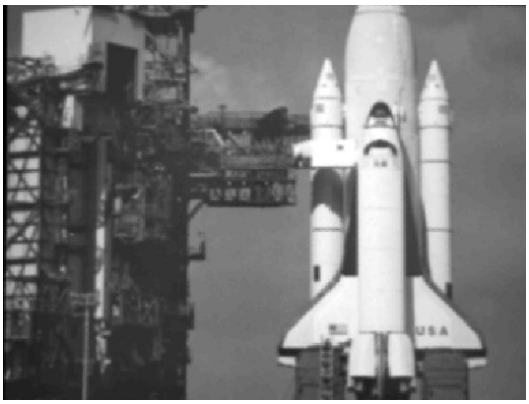


Figure 4.3 shows the original image *shuttle.tif* and Fig. 4.4 shows the result of rotating it 15° in the counter-clockwise direction to form the input image. Figure 4.5 displays the enlargement of the original image by a factor of 1.25 that is the base image. After putting in the commands for Matlab listed above we obtained Figure 4.6, 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. 4.6 to arrive at Fig. 4.7. 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. 4.8. The Matlab *Control Point Selection Tool* window.

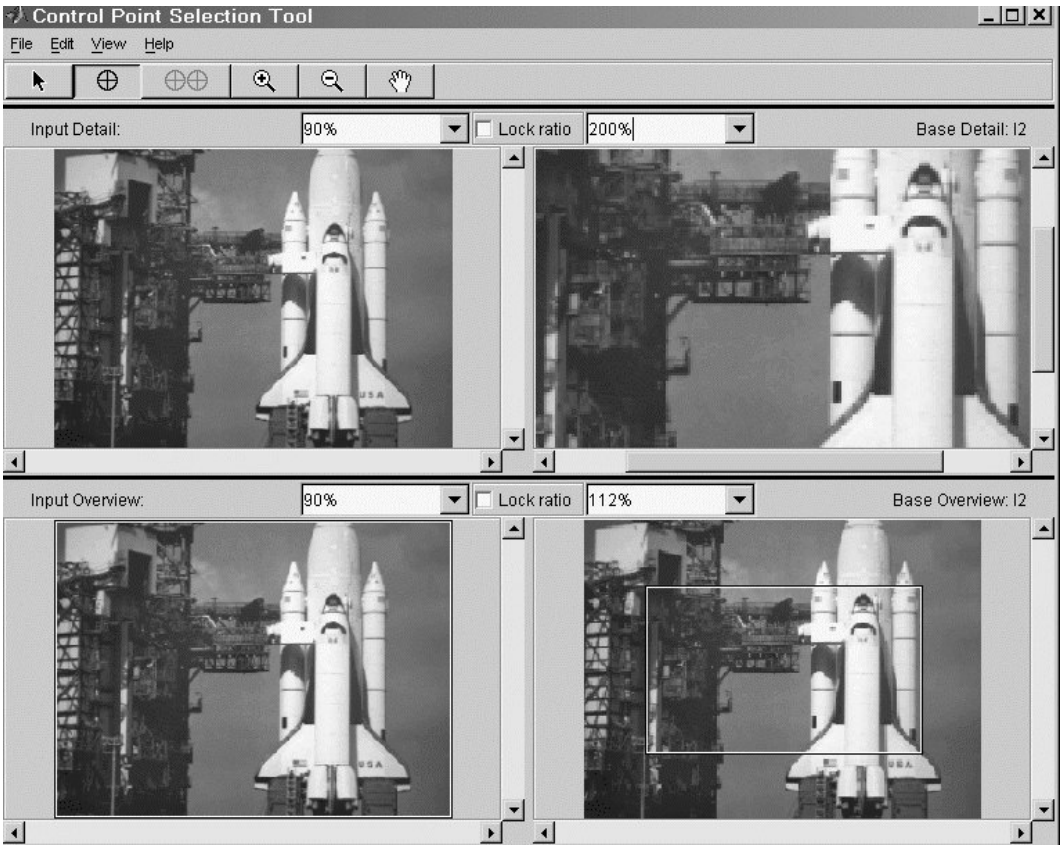


Figure 4.8 shows the Control Point Selection Tool window with the original *shuttle.tif* as the input at 90% of its original size in the detail input window (upper left) and in the overall input window (lower left). The base image here is shown at 200% of its orginal size in the detailed base window (upper right) and at 112% of its original size in the overview base window (lower right). In this latter window, the section that is magnified above is shown in a rectangle.

The circled *plus* button is down for selecting points in either the input base or detail base images, but whichever is used, the next selection must be from the opposite detail 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 the variable *mytform* above. This mapping is then applied with the function *imtransform()* as was done above.

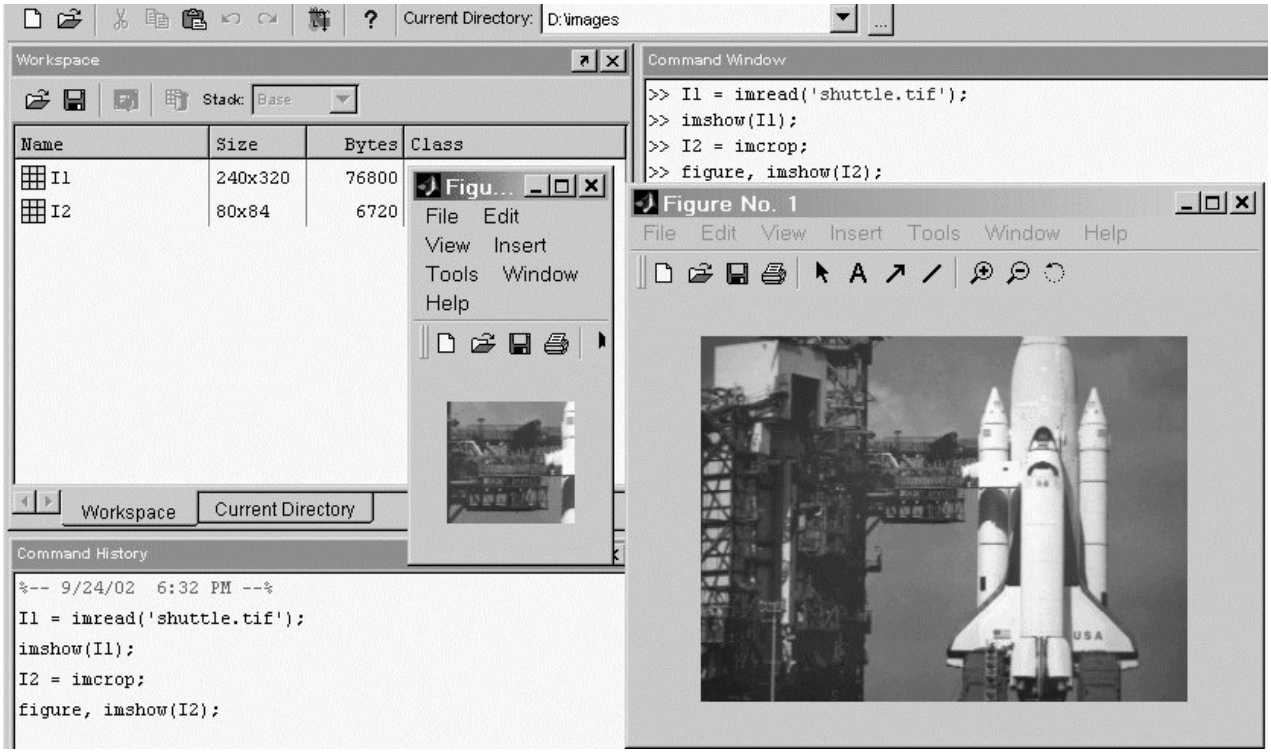
Generating Examples with Matlab by Cropping. We created an example above by resizing and rotation. These are critical to creating good examples, but we also need to be able to crop an image with Matlab. The following example shows the use of the *imcrop* tool in *Matlab's Image Processing Toolbox*.

```
>> I1 = imread(`shuttle.tif´);          // read in image to location I1
>> imshow(I1);                          // show the rotated image
>> I2 = imcrop;                         // call tool to crop displayed image I1
```

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 I2 in memory. Now we display the new image I2 and enlarge it by clicking and pulling down the bottom left corner for better viewing. Figure 4.9 shows the result.

```
>> figure, imshow(I2);                  // show cropped image, which may be small
>> I3 = imresize(I2, 2.0);              // double size of I2 and store as I3
>> figure, imshow(I3);                  // show the enlarged cropped image in new figure
```

Figure 4.9. A cropped rectangle from an image.



4.3 Pointwise Fusion of Pixels in Different 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 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.

Matlab *Example to Blank the Frame 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 with 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 gray shade. To complete the process, we stretch the contrast. Figures 4.10 and 4.11show the original and black and white images while Figures 4.12 and 4.13 display the multiplied and contrast stretched images. Images can be added, subtracted, multiplied and divided by either another image (arithmetic on the two corresponding pixels) or by a constant (arithmetic of the constant with each pixel).

```
>> 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. 4.10. Original image.

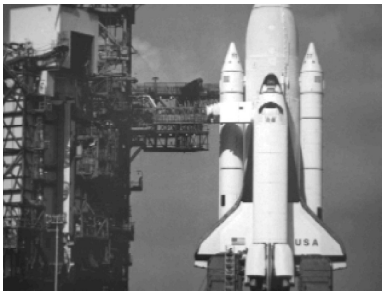
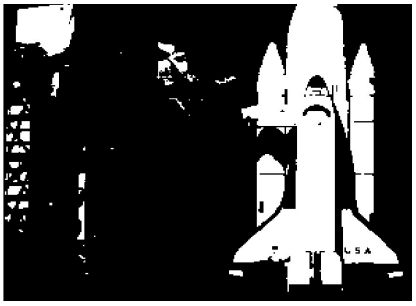


Fig. 4.11. Black and white image.



To reduce the background further, we could multiply the image in Figure 4.13 by, say, 0.25 to reduce the contrast and zero (blank) out many of the moderately light shades of gray and follow this by a contrast stretching or multiplication by 4 (why will multiplication by 0.25 followed by multiplication by 4 not yield the same image back?).

Fig. 4.12. Orig. multiplied by b&w.

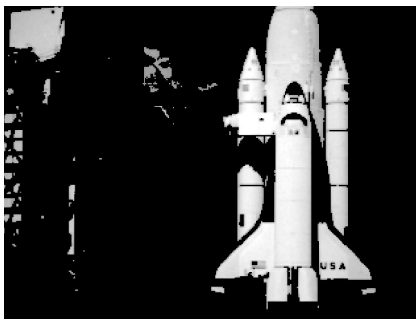
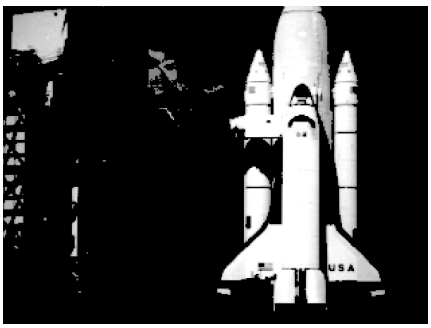


Fig. 4.13. Contrast stretched final image.



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 $\{f_{out}(m,n)\}$. Then

$$f_{out}(m,n) = af_1(m,n) + (1-a)f_2(m,n), \quad 0 \le a \le 1 \tag{4.5}$$

When a = 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 a can be increased to, say, 0.6, etc. For R frames, a 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, Multiplying and Dividing. This is a useful operation in certain situations. We take

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

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.

Ratio-ing 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. We use the log() function to compress the drastic range of values of the ratio of pixel values.

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

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

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 resulting values reasonable, but these values must then be mapped scaled into the interval [0, 255] to display a properly contrasted image.

A* Matlab *Example of Frame Arithmetic. The *Matlab* commands given below are an example of converting the original image *shuttle.tif* into one that appears to be at night with bright lights shining on the space shuttle. Figure 4.14, 4.15 and 4.16 are respectively the original, inverted and darkened inverted images. Figures 4.17 and 4.18 show the respective subtraction of Figure 4.16 from 4.14 and the multiplication of Fig. 4.16 by the constant 2.

Figure 4.14. Original shuttle.

Figure 4.15. Inverted shuttle.

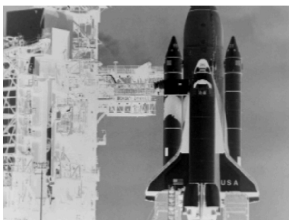
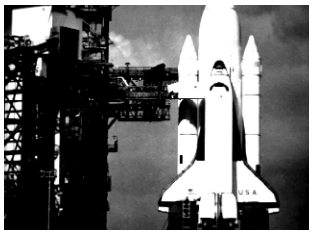


Figure 4.16. Darkened inverse.

Figure 4.17. Subtracted image.

Figure 4.18. Multiplied image.



```
>> i1 = imread('shuttle.tif');      //read in image
>> i2 = imcomplement(i1);           //invert it by taking its complement
>> i3 = immultiply(i2, 0.5);        //multiply the result by 0.5 to darken it further
>> imshow(i1);                      //show original image (Fig. 4.14)
>> figure, imshow(i2);              //show inverted image (Fig. 4.15)
>> figure, imshow(i3);              //show darkened inverted image (Fig. 4.16)
>> i4 = imsubtract(i1, i3);         //subtract darkened inverted image from original
>> figure, imshow(i4);              //show result (Fig. 4.17)
>> i5 = immultiply(i4, 2);          //multiply result by 2 to whiten light pixels
>> figure, imshow(i5);              //show whitened results (Fig. 4.18)
```

Image compositing. This is a process where a part (usually an object) of an image $f_1(m,n)$ can be segmented or isolated. All pixels outside of the boundary are set to zero in the output image $g_1(m,n)$ and this image is then multiplied by the original image. 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 top 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, an entertainer's head can be put on a different body.

4.4 Pixel Fusion Based on Areas

The Area Fusion Process. In this process we consider two pixels at the same location in 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 p 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 T of the nbhds of the pixels at that location.

$$f_{out}(m,n) = T[N\{f_{in}(m,n)\}], \quad \text{where } N\{f_{in}(m,n)\} \text{ is a nbhd of } f_{in}(m,n) \tag{4.9}$$

Spatial Frequency Fusion. The $f_{out}(m,n)$ value can be chosen from two input pixels $f_{in1}(m,n)$ and $f_{in2}(m,n)$ by their *pointwise spatial frequency* (psf), which is defined as the sum of the magnitudes of differences between a pixel and each other pixel in its neighborhood. For a 3x3 neighborhood the psf is

$$psf(p_5) = \Sigma_{k \neq 5} |p_k - p_5| \tag{4.10}$$

As we can see, the psf includes all directional absolute differences and reflects the degree to which the center pixel p_5 differs from its neighbors overall. For fusion, the psf's for $\{f_{in1}(m,n)\}$ and $\{f_{in2}(m,n)\}$ are calculated respectively at every pixel location (m,n). During the fusion process, we compare the psf values at each pixel (m,n). If $psf(f_{in1}(m,n))$ is larger than $psf(f_{in2}(m,n))$, $f_{out}(m,n) = f_{in1}(m,n)$, and so forth. This is because a greater psf indicates a greater variation at the pixel, which in turn indicates greater detail. In Figures 4.19 and 4.20 the respective right tree and left tree were blurred. Figure 4.21 shows the result of fusing them with pointwise spatial frequency to eliminate the blurring.

Fig. 4.19. Right tree is blurred. **Fig. 4.20. Middle tree is blurred.** **Fig. 4.21. Fused image is clear.**

Maximum Difference and Related Fusion. In these types of methods we consider two pixels $f_1(m,n)$ and $f_2(m,n)$ in two different images but at the same location that represents the same physical point in the scene, i.e., the images are registered. We consider a block neighborhood of each corresponding pixel in the two images, and also a larger outer block about the inner block. Figure 4.22 shows the situation. The process can take different forms, but essentially it is that the inner blocks are processed to compute representative values and similarly for the outer blocks. The difference between the inner and outer block representative values for each image determines the weights for a weighted average of the center pixels to form the output pixel.

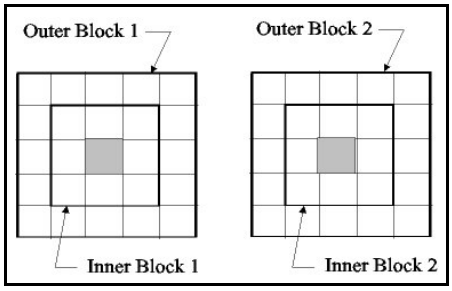


Figure 4.22. Nbhds inside nbhds for fusing pixels.

For example, let us take the averages u_1 and u_2 of the inner blocks and also the averages of the outer blocks v_1 and v_2. The differences are d_1 and d_2 where

$$d_1 = |u_1 - v_1|, \quad d_2 = |u_2 - v_2| \tag{4.11}$$

Upon comparing these we determine which one, say d_1, is the largest. Then we can select the first pixel, denoted by p_1, as the output pixel. This is an option where we want to obtain the image with the greatest differences. Another method is to compute weights, standardize them and then take a weighted average.

$$W_1 = |u_1 - v_1|, \quad W_2 = |u_2 - v_2| \tag{4.12a}$$

$$w_1 = W_1 / (W_1 + W_2) \tag{4.12b}$$

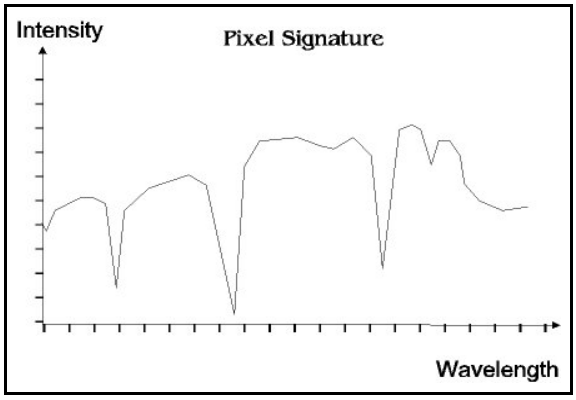
$$w_2 = W_2 / (W_1 + W_2) \tag{4.12c}$$

$$p_{out} = w_1 p_1 + w_2 p_2 \tag{4.12d}$$

Variations of this type of algorithm include the use of medians or trimmed means in place of the averages. We could also use pointwise spatial frequencies at the center pixel with respect to the inner block and with respect to the outer block, provided that speckle noise has been removed and light smoothing applied. It goes without saying that more than two images can be fused at each pixel in such a manner.

4.5 Multispectral and Hyperspectral Images

Multispectral Image Fusion. Here we have two or more images of the same scene in different bands of optic or infrared wavelengths. We consider the case of optic and infrared images that are to be fused (two images) with the objective of detecting humans or man-made objects among natural backgrounds. The infrared image will show metal as extra broght (emitting more heat) or extra dark from a background of foliage, depending on the ambient temperature. Humans aare extra warm, where warmer implies greater intensity in the grayscale image. For optic images, the objects of interest are determined by grayscale edges.



Hyperspectral Images and Fusion. Hypersepctral images are what we call a set of multiple images taken at the same time instant from the same point in space in different short bands of wavelength. For example the AVIRIS captures over 200 images of the same scent in different bands of light frequencies. For any given pixel, there are, say, 200 intensity values for the respective bands. If we graph these intensity values on the vertical axis over the 200 bands on the horizontal axis, we obtain a graph called a *signature* of the pixel. Figure 4.22 displays a hypothetical signature for a single pixel. Materials can be detected by their signatures (e.g., concrete, asphalt, metal, water, foliage).

Figure 4.22. A pixel signature.

Often is is useful to fuse red, green and blue representative frames of lower resolution with a higher resolution grayscale image. The *Brovey* algorithm (*Brovey transform*) is used for this. The basic idea is to use the R, G and B intensity images to merge with the higher resolution intensity image and convert back to color. We will consider this further when we discuss color models in a later unit.

4.6 Motion Detection.

Time Sequences of Frames. A video type of camera produces a sequence of consecutive frames over time. By comparing the frames by a type of 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 difference two consecutive frames at a time to blank out all pixels that are the same. The output magnitude of differences is essentially a zero (black) 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 from the background.

$$f_{diff1}(m,n) = f_r(m,n) - f_{r+1}(m,n) \tag{4.13a}$$

$$f_{diff2}(m,n) = f_{r+1}(m,n) - f_r(m,n) \tag{4.13b}$$

$$f_{diff}(m,n) = (1/2)(f_{diff1}(m,n) + (1/2)f_{diff2}(m,n) \tag{4.13c}$$

Note that $f_{diff1}(m,n) \neq f_{diff2}(m,n)$ in general although it may on many pixels. The reason that we need to obtain two differences is that the object that moves may be either lighter or darker than the background, so one of the two differences may blank out the object.

Tracking. Another useful process is to use thresholding on the two images first to blank out the image or the background. Then by using a sequence of consecutive image pairs, and computing the center of gravity of a connected blob in the difference image at each time step, a moving object can be *tracked*, that is, a trajectory of the moving object can be constructed. If the camera rotates to follow a moving blob, then the corrective adjustments must be made to accurately track the blob. The blob must be located, the left, right, upper, lower, upper left, lower left, upper right and lower right edges determined and then the approximate center of gravity computed. A similar technique is the fit an ellipse to the blob and use the center of the ellipse as the center of gravity.

An Example of Motion and Tracking. Figures 4.23 and 4.24 show two images of the same scene where an object (the cup in this example) has moved. First we subtracted the image of Fig. 4.24 from that of Fig. 4.23 to obtain Figure 4.25 shown below. We also subtracted Fig. 4.23 from Fig. 4.24 with the resulting image shown in Figure 4.26. Figures 4.27 and 4.28 are the inverses of Figures 4.25 and 4.26. It is clear that the cup object has moved. To show the objects in two different locations in a single image, we add the two images of Figures 4.25 and 4.26, noting that the black pixels sum to black while black and a specific gray sum to the specific gray. Figure 4.29 is the resulting image.

Figure 4.23. First image with cup.

Figure 4.24. Second image with cup.

Figure 4.25. Fig. 4.23 minus Fig. 4.24.

Figure 4.26. Fig. 4.24 minus Fig. 4.23.

Figure 4.27. Inverse of Fig. 4.25.

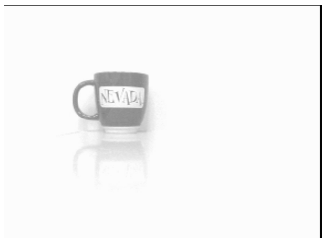


Figure 4.28. Inverse of Fig. 4.26.

Figure 4.29. Sum of Figs. 4.25 and 4.26.

4.8 Exercises

4.1 Use *Matlab* to crop the lower 2/3 of the central area of the image *shuttle.tif*. Take care to obtain as few gray levels as possible. Now enhance the contrast and sharpen this image to expose the cable that goes from the left bottom of the overdeck (between the platform and the shuttle) downward and rightward to the ground. Show all of the images.

4.2 Crop the overdeck in the shuttle image, resize it to be twice as wide and twice as high and then rotate it 30 degrees. Now register it as the input image with the original as the base image using *Matlab*. Show all images.

4.3 At the end of Unit 3 is a C program for mask transformations. Modify it so it computes the pointwise spatial frequency of an image. Use it to compute the psf image of the shuttle image (be sure to standardize the psf image). Select a good threshold to show all edges of moderate or greater strength and then invert the result using a tool. Show all images and describe all uses of tools and your own program.

4.4 Use the original shuttle image, a thresholded shuttle image converted to black and white and image arithmetic to get a bright shuttle object against a very dark background. Do not obliterate the detail of the shuttle object by making it too bright. Show the images at the various steps.