Film Composition

Title Filter Tools
Goal Learn how to perform image filtering in Conrad
Audience People who want to learn filtering in Conrad
Prerequisite Basics of Grid2D and ImageJ
Chapters Introduction Linear filter (mean filter) Nonlinear filter (median filter) Filtering in ImageJ Write your own filter
Duration 8 - 10 mins

Code

```
import ij.ImageJ;
import ij.IJ;
import ij.ImagePlus;
import ij.process.ImageProcessor;
import ij.process.FloatProcessor;
import ij.gui.Roi;
import ij.plugin.filter.Convolver;
import edu.stanford.rsl.conrad.filtering.MeanFilteringTool;
import edu.stanford.rsl.conrad.filtering.MedianFilteringTool;

public class videoTutorialsFilterTools {
	public static void main(String[] args) {

		String filename = "vessel.jpg";

		// Open the image using ImageJ
		ImagePlus imp = IJ.openImage(filename);

		// Wrap the ImagePlus instance to a Grid2D image
		Grid2D gridImage = ImageUtil.wrapImagePlus(imp).getSubGrid(0);

		// Grid2D images for saving the filtered images
		Grid2D filteredMean = new Grid2D(gridImage);
		Grid2D filteredMedian = new Grid2D(gridImage);



		// Mean filtering
		// Create a mean filter instance
		MeanFilteringTool meanFilter = new MeanFilteringTool();
```

```
// Create the mean filter and configure the kernel function
meanFilter.configure(5,5);

// Perform the filtering and save the filtered image
meanFilter.applyToolToImage(filteredMean);

// Display and compare the filtered image with the original one
//gridImage.show("Original");
//filteredMean.show("Mean filtered");



// Median filtering
// Create a median filter instance
MedianFilteringTool medianFilter = new MedianFilteringTool();

// Configure the kernel width and height
medianFilter.configure(5,5);

// Perform the filtering and
filteredMedian = medianFilter.applyToolToImage(gridImage);

// Compare the result with the original image
//gridImage.show("Original");
//filteredMedian.show("Median filtered");



// Gaussian filtering
// Using ImageJ to perform Gaussian filtering
IJ.run(imp, "Gaussian Blur...", "sigma=1.5");
//gridImage.show("Original");
//imp.show("");



// We can also create our own kernel function and then perform convolution
// Set kernel width and height
int kw = 3;
int kh = 3;

// Float array for storing the kernel data
float[] kernel = new float[kw*kh];

// Define the kernel
for(int i = 0; i < kernel.length; i++)
{
	kernel[i] = 1.f / (kw*kh);
}

// Compute the convolution of the image with the previously defined kernel
// Perform the convolution
FloatProcessor ip = ImageUtil.wrapGrid2D(new Grid2D(gridImage));
Convolver conv = new Convolver();
conv.convolve(ip, kernel, kw, kh);

Grid2D convolvedImage = ImageUtil.wrapFloatProcessor(ip);
gridImage.show("Original");
convolvedImage.show("Convolved Image");
```

```
        }
}
```

Syllabus

No.	Chapter	Text	Code snippet	Duration	Total time
1	Introduction	Hello everyone. Welcome to the Conrad Tutorials class. In this video, we are going to learn how to perform image filtering in Conrad and ImageJ.			
2	Linear filter (mean filter)	We start with linear filters. As we know, for a linear filter, the output is a linear function of the input. One widely used linear filter is the mean filter, which is already implemented in Conrad. In order to perform mean filtering, we first need to create an instance of the class MeanFilteringTool. Then we configure the kernel function by calling the member function *configure()*, which has two parameters specifying its width and height. This function will automatically generate the weights of the kernel matrix. After the preparation is done, we can now start the filtering process using the function *applyToolToImage()*, the input parameter of which is just a copy of the original Grid2D image. At last, we can compare the filtered image with the original one. <code running> As we can see, the resulting image was blurred.	`MeanFilteringTool meanFilter = new MeanFilteringTool();` `meanFilter.configure(5,5);` `meanFilter.applyToolToImage(filteredMean);` `gridImage.show("Original");` `filteredMean.show("Mean filtered");`		

3	Nonlinear filter (median filter)	Sometimes, mean filters do not perform well in case of, for example salt and pepper noise. In this case, some nonlinear filter, such as median filter, can achieve descent results due to its edge preserving. To use a median filter in Conrad is very similar to that of a mean filter. We need to first create a MedianFilteringTool instance, then configure the kernel width and height, and finally call the function *applyToolToImage()* to trigger the filtering process. We can display the original and filter images to see the results. <code running>	`MedianFilteringTool medianFilter = new MedianFilteringTool();` `medianFilter.configure(5,5);` `filteredMedian = medianFilter.applyToolToImage(gridImage);` `gridImage.show("Original");` `filteredMedian.show("Median filtered");`		
4	Filtering in ImageJ	In Conrad, you can also execute ImageJ commands to filter an image. You can do that by running the command „Gaussian Blur...“, which has an input parameter sigma that specifies the standard deviation. <code running> We can see that the original image has been smoothed after we ran this command. You can set different values to sigma to see how the image would be influenced.	`// Gaussian blur` `IJ.run(imp, "Gaussian Blur...", "sigma=1.5");`		
5	Write your own filter	In Conrad, there is a class called Convolver, which can be used to perform convolution between a kernel and an image. We first set the kernel width and height. Then we define a float array for storing the kernel. Now we can specify our own weights of the kernel. Since the Convolver needs a FloatProcessor as its input parameter, we need to wrap the	`int kw = 3;` `int kh = 3;` `float[] kernel = new float[kw*kh];` `for(int i = 0; i < kernel.length; i++) kernel[i] = 1.f / (kw*kh);` `FloatProcessor ip = ImageUtil.wrapGrid2D(new Grid2D(gridImage));`		

		Grid2D image to a FloatProcessor instance. Then we create a Convolver instance by calling its constructor. The convoluton is triggered if we call the function *convolve()*, which has four input parameters, i.e. the FloatProcessor, the kernel function, the kernel width and the kernel height, respectively. After the convolution is done, we can wrap the FloatProcessor instance back to a Grid2D image for displaying. This can be done by using the static function *wrapFloatProcessor()* in the class ImageUtil. Finally we can show the images. <code running>	``` Convolver conv = new Convolver(); conv.convolve(ip, kernel, kw, kh); Grid2D convolvedImage = ImageUtil.wrapFloatProcessor(ip); gridImage.show("Original"); convolvedImage.show("Convolved Image"); ```		