Image Processing in MatDeck

Image processing, a subbranch of digital signal processing, is the use of computer algorithms to perform processing of a digital image. Digital images are defined over two dimensions, which makes all these algorithms multidimensional. Image processing tasks implemented in MatDeck are the most often used algorithms for image enhancement, and image manipulation.

Image Acquisition

In MatDeck, there is a function called image read() which is used to read an image from a graphics file. This function works with common bitmap graphic files: .png, .gif, .jpg, .bmp and tif. The following line shows how to read an image file given by name, inferring the format of the file from its contents and extension. Through this example we use the standard test images, such as tulips.png below.

```
1  img := image read("tulips.png")
```

The image can be displayed in a MatDeck document using image widget() and embed widget():

```
pic:=image widget(0 , img)
set size(pic , 300 , 200)    The function sets the size of widget.
```

The second option for image acquisition is image capture() which uses the system camera to make digital photography. In this case, the resolution of the image is determined via the camera device. When the processing is done, the resulting image can be saved by using the function image write() providing the file name.

```
2  image write("after.png", img)
```

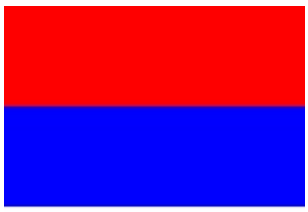
Image Creation

It is possible to convert any two dimensional matrix into a image using the function matrix2image(). There is also a reverse function which can transform the image matrix into a regular matrix, which is usually done when there is need to use regular matrix operations. We can check anytime if a matrix variable is an image or not using is image() function. Besides that, it is possible to create an image from scratch by creating an image with a given number of pixels, i.e. absolute resolution. In an image matrix it is possible to set the pixels values manually. In a further segment, we illustrate simple image creation using these options. Function to bgra() is used to combine RGB values and transparency into pixel.

```
3  imgc := image create(1, 3)
4  imgc := set value at(imgc, to bgra(0, 0, 255, 255), 0, 0)
5  imgc := set value at(imgc, to bgra(255, 0, 0, 255), 1, 0)
```

```
6
7 imgc := set value at(imgc, to bgra(255, 255, 255, 255), 2, 0)
```

```
w1 := image widget(0 , imgc)
set size(w1 , 150 , 150)
```



```
image dpi(img) = 72
img := image set dpi(img , 90)
image dpi(img) = 90

is image(img) = true
```

Further, it is possible to set the relative resolution of an image by using the function image set dpi(), and check current using image dpi() as illustrated above.

Basic Image Manipulations

MatDeck contains functions for basic operations with digital images. It is possible to extract each color component into a separate matrix for further processing using the functions image red(), image green(), image blue(). There is also a function which extracts the information regarding transparency, which is used in .png images, image alpha().

```
8  Red := image red(img)
9  Green := image green(img)
10 Blue := image blue(img)
11 Transparency := image alpha(img)
```

Functions above take image as a matrix and return matrices of the same size containing a single color channel. In MatDeck, there is a function which combines the four components into a single image bgra() as a whole.

```
12 Green = matrix2image(floor(0.5 * Green))
13 img1 := image bgra(Blue, Green, Red, Transparency)
```

The image img contains the weakened Green component.

```
pic1 := image widget(0 , img1)
set size(pic1 , 300 , 200)
```

The function sets the size of widget.

In the previous part, there are functions which are matrix based. In some situations, image processing can be pixel based, for that reason there are MatDeck functions which are pixel based and extract color

components from a single pixel. These functions are to red(), to green(), to blue() and to alpha(). Function to bgra() is used to combine RGB values and transparency into a single pixel. Let us return to the image imgc from above, which is a simple image of width one and height three. Recall, the pixel imgc[0] is red.

```
to red(imgc[0]) = 255
to green(imgc[0]) = 0
to blue(imgc[0]) = 0
to alpha(imgc[0]) = 255
```

Color Manipulation

In MatDeck there are built in functions for basic manipulation with color content and transparency within an image. These functions, image red mul(), image green mul(), image blue mul(), image alpha mul(), allow the multiplication of color components with a constant. If a multiplicative constant is greater than one, the color content will be increased. Further, if the multiplicative constant is smaller than one, the color content will be decreased. In a special case when the multiplicative constant is zero, the selected color component is set to zero. Here are several examples.

```
// Red component of image Tulips
imgr := image blue mul(img, 0)
imgr := image green mul(img, 0)
 // Enchanced blue component in image Tulips
imgb := image blue mul(img, 2)
```

```
picr := image widget(0 , imgr)
set size(picr , 300 , 200)
```

```
picb := image widget(0 , imgb)
set size(picb , 300 , 200)
```

It is possible to experiment with transparency, alpha, in an image and in that way one can change the shape of image.

```
imga := image alpha mul(img, 0.2)
```

In order to see the effect of transparency, the canvas with a background color is used. Next, we change the shape of the image. The shape of the image can be defined using loops which allows us to access pixels of the image. For that purpose we use the functions image width() and image height().

```
wi := image width(img)
he := image height(img)
alpham := triangular(Transparency, "upp")
alpham := matrix2image(alpham)
imgs := image bgra(Blue, Green, Red, alpham)
```

pica1 := image widget(0 , imga)

set size(pica1 , 300 , 200)

pica2 := image widget(0 , imgs)

set size(pica2 , 300 , 200)

There is a function which creates the mirror image of a given image in MatDeck. The user can select the orientation of the mirror image.

```
25 imgm := image mirror(imgs, true, false)
26 imgm1 := image mirror(imgs, false, true)
```

pica3 := image widget(0 , imgm)

set size(pica3 , 300 , 200)

pica4 := image widget(0 , imgm1)

set size(pica4 , 300 , 200)

Image fill function is used to enhance the color quality further.

```
27 imgf := image fill(img, to bgra(200, 100, 50, 255))
28 picm := image widget(0, imgf)
29 set size(picm, 100, 100)
```

Further Processing

We have presented the basic image operations including color manipulations on image matrix, or on pixel. Further processing in MatDeck includes image scaling and rotation, and image filtering.