University of Ljubljana, Faculty of Electrical Engineering
Robot Vision (RV)

March 16/18, 2016

Exercise 1: Image Management and Display

Created by: Žiga Špiclin | http://lit.fe.uni-lj.si/RV | Homework deadline: March 23/25, 2016

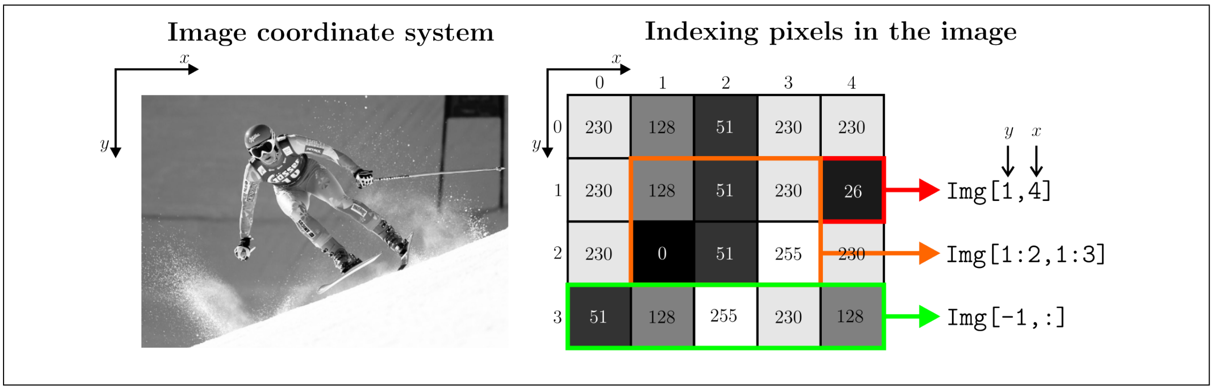
Instructions

During this exercise you will get familiar with the Spyder-Python development environment and learn how to read, write and display digital images using Python programming language. In Python, a grayscale digital image can be represented by a two-dimensional (2D) array of type `ndarray`, which is available in library package `numpy`, whereas a value in each pixel of a digital image can be, for instance, 8-, 16- or 32-bit signed or unsigned integer or a floating-point number (Table 1).

Table 1: Grayscale pixel value data types in numpy library package.

Grayscale value	Data type (`dtype`)	Value range
logical	`'bool'`	{ `False` , `True` }
8-bit unsigned	`'uint8'`	[0, 255]
16-bit unsigned	`'uint16'`	[0, 65535]
32-bit signed	`'uint32'`	[-2^{31}, 2^{31}-1]
32-bit floating point	`'float32'` or `'single'`	[0.0, 1.0]
64-bit floating point	`'float64'` or `'double'`	[0.0, 1.0]

Library package `numpy` can be imported into Python using `import numpy as np`, then, the functions and variables in the library can be accessed using `np.___`. Very useful function to initialize a `ndarray` are `zeros()`, `ones()`, `zeros_like()`, `ones_like()`, `asarray()`, to convert the data type `astype()` or `array( mArray, dtype= ... )`, to get the number of dimensions `ndim()` and the dimensions `shape()` , while `reshape()` and `transpose()` and used to reshape and transpose the array, respectively. Elements of a 2D array named `mArray` can be accessed, for instance, as `mArray[0,3]` (element in first row, fourth column), `mArray[:,1]` (second column), `mArray[-1,:]` (last row), etc. The indices of an array given a logical expression can be determined using function `where()`. The coordinate system of an image based on `ndarray` is shown in the figure below.



	0	1	2	3	4
0	230	128	51	230	230
1	230	128	51	230	26
2	230	0	51	255	230
3	51	128	255	230	128

For reading and writing the `ndarray` based image in raw (uncompressed) format one can use the respective functions `fromfile()` and `tofile()`. Reading and writing images in a standard format like `bmp`, `png`, `gif`, `eps`, `jpeg` can be performed using library package `PIL.Image`. Image is read into Python using function `open()`, which creates a variable of type `Image` and its (grayscale) data format can be obtained using function `Image.getbands()` or converted to other formats using `Image.convert()`. Varibles of type `Image` can be easily converted into `ndarray` using `numpy.array()`, while the inverse conversion can be performed by `Image.fromarray()`. To save an image use `Image.save()`.

Functions for graphical display are available in library package `matplotlib.pyplot`. To display an `ndarray` image use function `imshow()`, while other useful function are `figure()` to open a new display window, `suptitle()`, `xlabel()` and `ylabel()` to set figure and axis captions, and `axes()` to manage the axes. Use `show()` to redraw the display.

1. By using library package `PIL` load the image `slika.jpg`, convert it to grayscale image and then store the image in a `png` file.

2. File `slika-8bit.raw` contains a 2D grayscale image in the form of raw or uncompressed data. The image's width and height are $X \times Y = 975 \times 650$ pixels, whereas each pixel's grayscale intensity is an unsigned 8-bit value. Your task is to write a function that can load an arbitrary raw grayscale image. Consider a function with the following declaration:

   ```
   def loadImageRaw( iPath, iSize, iFormat ):
           return oImage
   ```

   where `iPath` is the full path to the image (directory and filename), `iSize` a vector of image dimensions in pixels and `iFormat` the pixel data format. This function should load the image and return it in variable `oImage`, encoded as a `numpy.ndarray` array. For this purpose you should import the library package `numpy` and make use of the `fromfile()` function.

3. Display an image using function `imshow()`. To correctly display a grayscale image you should adjust parameter `cmap` (set to `cm.Greys_r` by using library package `matplotlib.cm`). The aspect ratio of the display axes should equal the image axes, which can be achieved using `matplotlib.axes().set_aspect( 'equal' , 'datalim' )`. Your task is to write a function that displays an arbitrary grayscale image given as a `numpy.ndarray`:

   ```
   def showImage( iImage, iTitle ):
   ```

   where `iImage` is the image variable to display, while `iTitle` is a title string of the display window.

4. Write a function to store an image given as `numpy.ndarray`:

   ```
   def saveImageRaw( iImage, iPath, iFormat ):
   ```

   where `iImage` is the image variable to be stored, `iPath` the full path to the output image (directory and filename) and `iFormat` the pixel data format. For this purpose you should import the library package `numpy` and make use of the `tofile()` function.

5. File `slika-16bit.raw` contains a 2D grayscale image in raw or uncompressed format with dimensions of $X \times Y = 975 \times 650$, whereas each pixel's value is encoded by an unsigned 16-bit integer. Accordingly modify the functions `loadImageRaw()`, `showImage()` and `saveImageRaw()` so they can be used to read, display and write unsigned 16-bit images.

6. File `slika-24bit-rgb.raw` contains an RGB color 2D image in raw or uncompressed format with dimensions $X \times Y = 975 \times 650$, while each pixel value has $24 = 3 \times 8$ bits. The RGB color image is stored as a sequence of three unsigned 8-bit integer images that encode red (R), green (G) and blue (B) components. Accordingly modify the functions `loadImageRaw()`, `showImage()` and `saveImageRaw()` so they can be used to read, display and write RGB color images.

Homework Assignments

Homework report in the form of a Python script entitled `NameSurname_Exercise1.py` should execute the requested computations and function calls and display requested figures and/or graphs. It is your responsibility to load library packages and provide supporting scripts such that the script is fully functional and that your results are reproducible. The code should execute in a block-wise manner (e.g. `#%% Assignment 1`), one block per each assignment, while the answers to questions should be written in the corresponding block in the form of a comment (e.g. `# Answer: ...`).

1. Write a function to load image in standard formats (e.g. bmp, jpg, png, tif, gif) using library package `PIL.Image`:

   ```
   def loadImage( iPath ):
           return oImage
   ```

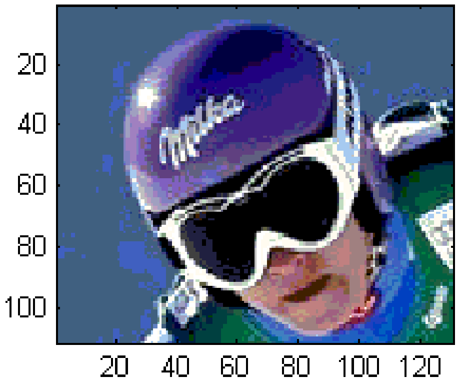


Figure 1: Rectangular area of the color image cropped between vertices $(x_1, y_1) = (260, 70)$ in $(x_2, y_2) = (390, 180)$.

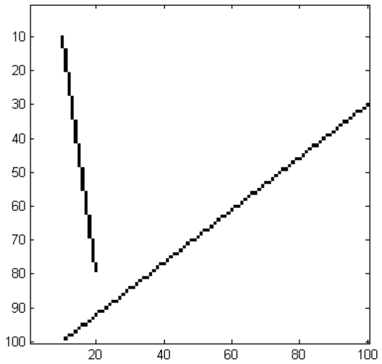


Figure 2: Rendered lines with grayscale value 0 between set vertices $(x_1, y_1) = (10, 10)$ to $(x_2, y_2) = (20, 80)$ and $(x_1, y_1) = (100, 30)$ to $(x_2, y_2) = (10, 100)$.

where `iPath` is the full path to the image (directory and filename). Function should return the image as a `numpy.ndarray` in variable `oImage`.

2. Write a function to store the image in standard formats (e.g. bmp, jpg, png, tif, gif) using library package `PIL.Image`:

```
def saveImage( iPath, iImage, iFormat ):
```

where `iPath` is the full path to the iamge (directory and filename), variable `iImage` is the image given as `numpy.ndarray` and `iFormat` a string denoting a desired output format (e.g. `’bmp’`, `’png’`).

3. Create a new image from the color 2D image by extract a rectangular area between vertices $(x_1, y_1) = (260, 70)$ to $(x_2, y_2) = (390, 180)$. Display the new image and verify the results by comparing to Figure 1.

4. Convert an RGB color image $C = [R,\ G,\ B]$ into a grayscale image S using equation $S = \frac{1}{3}R + \frac{1}{3}G + \frac{1}{3}B$ and compute projections of maximal and mean along the x and y axes of the grayscale 2D image. The projections can be implemented using a `for` loop or using functions `numpy.ndarray.max()` and `numpy.ndarray.mean()`, whereby you should accordingly set the parameter `axis`. To display the projections you can use the function `plot()` in the library package `matplotlib.pyplot`.

5. Modify the color image by setting the values of the red channel with values between 160 and 200 with a value of 255. Use function `where()` in the library package `numpy`. Display the original and the modified color image.

6. Use algorithm 1 to write a function for drawing digital lines into a grayscale `numpy.ndarray` image:

```
def drawLine( iImage, iValue, x1, y1, x2, y2 ):
    return oImage
```

where `iImage` is the image variable, `iValue` the grayscale value assigned to a pixel on the line in the image, while (x_1, y_1) and (x_2, y_2) are the coordinates that define the start and end points of the line. Verify the function by creating a white grayscale image with dimensions $X \times Y = 100 \times 100$ and then drawing two black lines, first from $(x_1, y_1) = (10, 10)$ to $(x_2, y_2) = (20, 80)$ and second from $(x_1, y_1) = (100, 30)$ to $(x_2, y_2) = (10, 100)$. Compare the obtained image with Figure 2.

Algorithm 1(Bresenham's algorithm)

```
function drawLine(image, value, x1, y1, x2, y2)
  dx := abs(x2-x1)
  dy := abs(y2-y1)
  if x1 < x2 then sx := 1 else sx := -1
  if y1 < y2 then sy := 1 else sy := -1
  error := dx-dy
loop
  image(x0,y0) = value
  if x0 = x1 in y0 = y1 then
     break loop
  e2 := 2*error
  if e2 > -dy then
     error := error - dy
         x0 := x0 + sx
  if e2 <  dx then
     error := error + dx
         y0 := y0 + sy
end loop
```

7. Modify the function `drawLine()` such that digital lines can be drawn into an RGB color image with a specified color. Create a 2D color image with dimensions $X \times Y = 300 \times 100$ and color all pixels in white. Use the modified function to draw the letters of your name, whereby you use multiple lines and manually define the start and end points of each line. Draw each letter of your name in a different color.