Introduction

RoVi1 vision exercises

Kim Lindberg Schwaner

University of Southern Denmark
Faculty of Engineering
The Maersk Mc-Kinney Moller Institute

September 1, 2017

Credits

- Based on slides by Stefan-Daniel Suvei

TA

- Kim Lindberg Schwaner (PhD student)
- `kils@mmmi.sdu.dk`
- Office Ø27-604-3
- No fixed office hours, but you are welcome to write me an email or drop by if you need help.

Format

Every session we will

- introduce the next exercise;
- go though hints or important information required to solve the exercise; and
- review the suggested solution to the previous exercise.

Feedback

- Ask for help if you need it.
- Provide feedback if you have ideas for improvement.

OpenCV introduction

- OpenCV is a widely used computer vision software library with API's for many languages.
- Exercise solutions will target C++11 / OpenCV 3.x.
- We will only support the provided virtual machine.
- Find help at
  - `docs.opencv.org`;
  - `answers.opencv.org`; and
  - `docs.opencv.org/master/d9/df8/tutorial_root.html` (tutorials).

Getting started

- Get the virtual machine up and running.
- Download, set up, compile and run the `DisplayImage` example application (download it from BlackBoard).
- Use the `DisplayImage` program as the basis for solving the first exercise.

Exercise 1

- Load the image color.png and change some of its pixel values to draw a black rectangle.
- Rectangle coordinates: $x \in [350, 440]$ and $y \in [100, 220]$.
- Display the modified image.
- Additionally you must
  - complete the task in (preferably three) different ways;
  - set the color of the rectangle to something other than black; and
  - open the image as, or convert it to, grayscale and draw a rectangle in the grayscale image (note the image depth and number of channels, as compared to the color image).
- See the exercise description on BlackBoard for helpful links.

Exercise 1

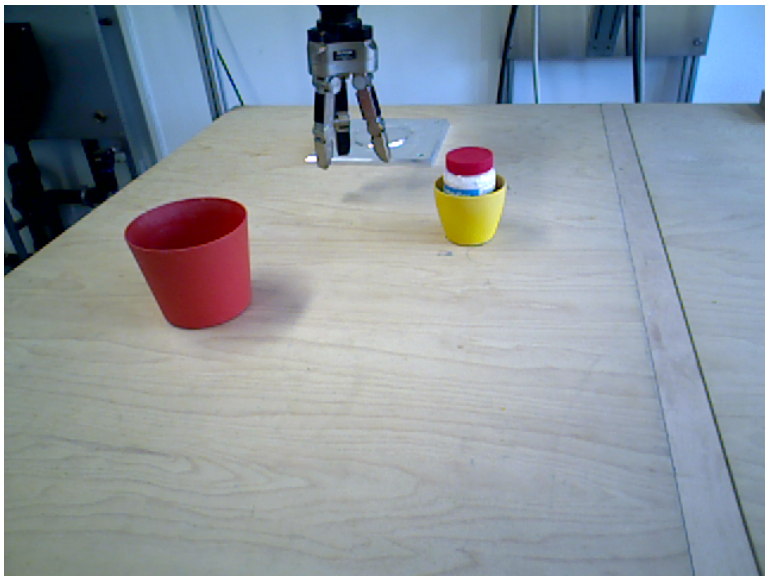


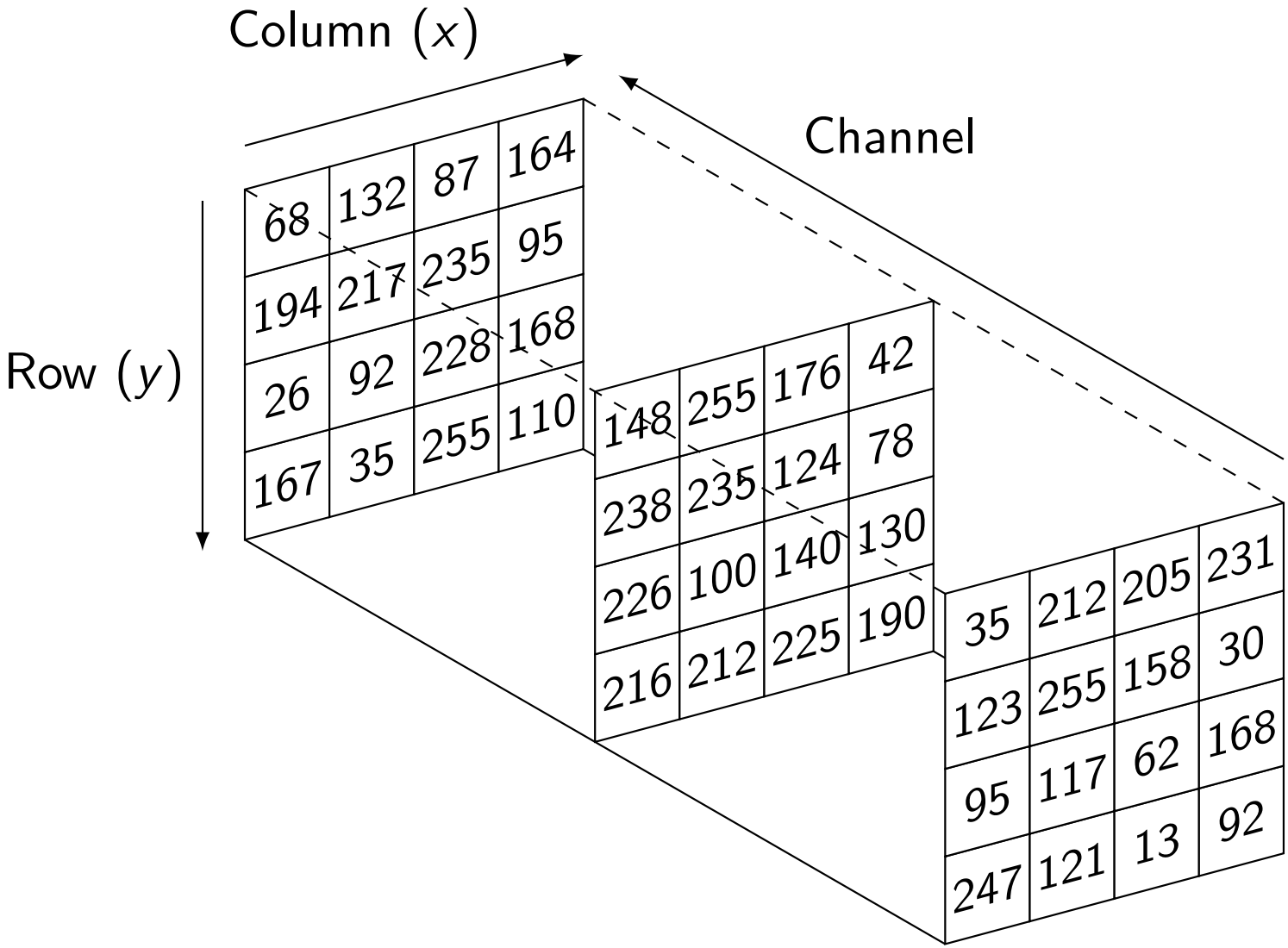
Figure: Original image.

Figure: Image with rectangle.

The OpenCV image container

- Images are stored as a matrices, represented by `cv::Mat`
- `cv::Mat` is a class consisting of
  - a header; and
  - a pointer to the actual image data (pixel intensity values).
- Copy constructors and assignment operators copies only the header (they give a new "view" into the pixel data) and *not* the pixel data itself.
- To duplicate pixel data use the `Mat::clone()` or `Mat::copyTo()` methods.

Logical matrix layout



Depth and channels

- Images are stored in `cv::Mat`'s containing one or more *channels* with pixel value type defined by *depth*.
- The number of channels depend on storage method. There is e.g. 1 channel in grayscale images and 3 channels in RGB images.
- Depth can be one of
  - `CV_8U`: 8-bit unsigned integers
  - `CV_8S`: 8-bit signed integers
  - `CV_16U`: 16-bit unsigned integers
  - `CV_16S`: 16-bit signed integers
  - `CV_32S`: 32-bit signed integers
  - `CV_32F`: 32-bit floating-point numbers
  - `CV_64F`: 64-bit floating-point numbers

Initializing matrices

- Default initialization (initializes just the header):

  ```
  cv::Mat A;
  ```

- Create a 16×16 2-channel (complex) matrix with uninitialized pixel values of floating-point type:

  ```
  cv::Mat B(16, 16, CV_32FC2);
  ```

- Create a 8×8 single channel matrix with zero-initialized pixel values of unsigned integer type:

  ```
  cv::Mat C = cv::Mat::zeros(8, 8, CV_8U);
  ```

- A 8×8 4-channel matrix with uninitialized pixel values of 16-bit signed integer type:

  ```
  cv::Mat_<cv::Vec4s> D(8, 8);
  ```

  (using template wrapper `cv::Mat_<T>` for matrices with type `T` known at compile time)

Accessing matrix elements

- Get a pointer to row i of the 3-channel 8-bit image `img` and then use it to set all channels of element i, j to 0:

```
cv::Vec3b* p_row = img.ptr<cv::Vec3b>(i);
p_row[j] = cv::Scalar::all(0);
```

- Do the same but use the `Mat::at()` method:

```
img.at<cv::Vec3b>(i, j) = cv::Scalar::all(0);
```