UNIVERSIDAD PABLO DE OLAVIDE SEVILLA
ESCUELA POLITÉCNICA SUPERIOR

Robótica y Visión Artificial
Ingeniería Informática en Sistemas de Información

EPD 4: ROS and OpenCV introduction. Low-level image processing

Objectives

- First contact with OpenCV
- First low-level image processing working code.

Cuestiones

1 Materials

In this and the remaining EPDs we will use the following tools:

- ROS (Robot Operating System): http://www.ros.org
  - In particular, we will use the ROS Indigo distribution over Ubuntu 14.04 LTS
- OpenCV: http://opencv.org
  - We will use the 2.x version of OpenCV's API. We will intensively refer to the documentation (http://docs.opencv.org/index.html).

We will employ in the EPD a machine with Ubuntu 14.04LTS and ROS Indigo installed. Furthermore, the computer includes an already configured ROS workspace. This corresponds to implement all the following steps (in case you have your own computer, you have to complete all of them to follow the class. In this case, it is recommend to install the full desktop version of ROS):

- http://wiki.ros.org/indigo/Installation/Ubuntu
- http://wiki.ros.org/ROS/Tutorials/InstallingandConfiguringROSEnvironment

2 Using the OpenCV library within ROS

We will create a first Computer Vision program using OpenCV. OpenCV is a third-party library, independent of ROS (although the development teams of both are related).

Create a package called epd4 with a dependence with the ROS package roscpp

Create a package called epd2 within the directory src of your ROS. You can create the package by commanding:

```
catkin_create_pkg epd4 roscpp
```

The source code for this package can be found at Campus Virtual.

We want to add to our package an executable. We will call this executable epd4_c3. Download all the source files in WebCT to the src folder in the epd4 package.

We should add the following line to the file CMakeLists.txt of our package:

```
add_executable(epd4_c3 src/nodec3.cpp src/student.cpp)
```

With this line, we are indicating that we will add an executable called epd4_c3 to our package, and that this executable is made by the compilation of the files nodec3.cpp and student.cpp, located in the folder src.

We have to link our executable with the OpenCV library. This is done by adding the following line after the previous one:

```
target_link_libraries(epd4_c3 ${catkin_LIBRARIES} ${OpenCV_LIBRARIES})
```

The variable ${OpenCV_LIBRARIES} can be used if, at the beginning of the CMakeLists.txt file we add the following lines:

```
find_package(OpenCV)
include_directories(${OpenCV_INCLUDE_DIRS})
```

These lines are used by CMake to find where OpenCV is located in our system

3 A first program with OpenCV

Now we can build our package and play with it.

Make the package epd4 by doing catkin_make. Open the files nodec3.cpp and student.cpp from the src folder

This first software will process one image using OpenCV functions.

Remember that the basic data type that we will use from OpenCV in the Mat class, which represents a multi-dimensional matrix.

http://docs.opencv.org/modules/core/doc/basic_structures.html#mat

```
class CV_EXPORTS Mat
{
public:
    // ... a lot of methods ...
    ...

    /*! includes several bit-fields:
         - the magic signature
         - continuity flag
         - depth
         - number of channels
     */
    int flags;
    //! the array dimensionality, >= 2
    int dims;
    //! the number of rows and columns or (-1, -1) when the array has more than 2 dimensions
    int rows, cols;
    //! pointer to the data
    uchar* data;

    //! pointer to the reference counter;
    // when array points to user-allocated data, the pointer is NULL
    int* refcount;

    // other members
    ...
};
```

The first thing that it is done in the code (at nodec3.cpp) is reading an image from file:

```
cv::Mat input_image = cv::imread(argv[1],0);
```

The functions in OpenCV to read and write images from/to files can be found at:

http://docs.opencv.org/doc/user_guide/ug_mat.html

It is also interesting to be able to visualize the images and the results from the processing. This is done with lines like:

```
cv::imshow("input_image", input_image);
```

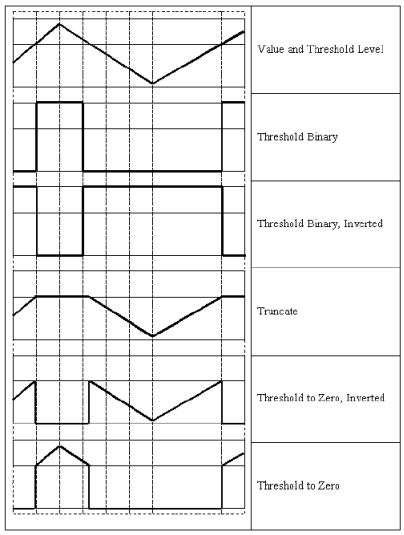
where the first argument is the name of the window, and the second the image to be shown. More information about this can be found at:

http://docs.opencv.org/doc/user_guide/ug_mat.html#visualizing-images

Finally, the processing takes place in the function that you can find at the **student.cpp** file. You can find there some examples of different operations.

- Thresholding: http://docs.opencv.org/modules/imgproc/doc/miscellaneous_transformations.html#threshold

C++: double **threshold**(InputArray **src**, OutputArray **dst**, double **thresh**, double **maxval**, int **type**)¶



- Morphological operations:
  - Dilation: http://docs.opencv.org/modules/imgproc/doc/filtering.html#dilate
  - Erotion: http://docs.opencv.org/modules/imgproc/doc/filtering.html#erode

- 2D general filters: http://docs.opencv.org/modules/imgproc/doc/filtering.html#filter2d

void **filter2D**(InputArray **src**, OutputArray **dst**, int **ddepth**, InputArray **kernel**, Point **anchor**=Point(-1,-1), double **delta**=0, int **borderType**=BORDER_DEFAULT)

- Border detectors: http://docs.opencv.org/modules/imgproc/doc/filtering.html#sobel

Many more processing functions can be found at: http://docs.opencv.org/modules/imgproc/doc/imgproc.html

In order to run the current code, we have the order:

```
rosrun <package> <executable> [params]
```

which it will run a given executable of a particular package.

Download the image aruco.jpg from WebCT and place it where you want. From the same location where the image is located, type the following command:

```
rosrun epd4 epd4_c3 ./aruco.jpg
```

Play with the different functions, commenting and uncommenting them (every time you will have to recompile with rosmake).

4 Processing video from Kinect

In this last assignment, we will use some further ROS functionalities. In particular, ROS allows recording data and re-playing it back with its tool **rosbag**. Then, any ROS node can subscribe to the data being replayed and process it.

Install the following package:

```
sudo apt-get install ros-indigo-vision-opencv
```

Add the following internal dependencies (<build_depend > y <run_depend>) to our package epd2 (in the file package.xml): cv_bridge, image_transport.

For this part, we will create a new executable in the same package, called epd_c4.

Add the following lines to the file CMakeLists.txt of our package:

```
find_package(catkin REQUIRED COMPONENTS roscpp cv_bridge image_transport
)

add_executable(epd_c4 src/nodec4.cpp src/student.cpp)
target_link_libraries(epd_c4 ${catkin_LIBRARIES} ${OpenCV_LIBRARIES})
```

Build the package with catkin_make.

The new code will receive recorded images from a Kinect and will process them. These images are stored in two files called kinect1.bag and kinect2.bag, which will be provided in the class.

Copy the bag files into your machine

To send these video images, we need to run several ROS functionalities:

- First of all, in a different terminal, we need to run the following command:

  ```
  roscore
  ```

  As it will be explained, roscore launches a name server that it is used by ROS whenever we have different nodes communicating. In the previous question we just had one node running, and therefore it was not required. However, in this case we will have two nodes. The **rosbag** process replaying the data, and our node processing the data.

- In a different terminal, if we want to replay the recorded data, we have to issue the following command:

  ```
  rosbag play kinect1.bag
  ```

  This will publish the recorded data over a set of ROS topics. In order to see the topics published, you can type in a terminal:

  ```
  rostopic list
  ```

In order to process the images from the recorded data, you have to launch the epd4 epd4_c4 process first, and then the rosbag play command. You will be able to see the images. The process epd4_c4 can be launched as usually:

```
rosrun epd4 epd4_c4
```

Inspect the code and modify the function processImageColor_c4 in order to get as output a binary image with white on the blue object, and black the background.

Using the second recorded dataset, kinect2.bag, create a new set of functions that are able to segment the bee on the images.

You can use the following functions to enhance the results from the thresholding. They allow to perform the erode and dilate operations on binary images:

C++: void **erode**(InputArray **src**, OutputArray **dst**, InputArray **kernel**, Point **anchor**=Point(-1,-1), int **iterations**=1, int **borderType**=BORDER_CONSTANT, const Scalar& **borderValue**=morphologyDefaultBorderValue())¶

http://docs.opencv.org/modules/imgproc/doc/filtering.html?highlight=erode#erode

C++: void **dilate**(InputArray **src**, OutputArray **dst**, InputArray **kernel**, Point **anchor**=Point(-1,-1), int **iterations**=1, int **borderType**=BORDER_CONSTANT, const Scalar& **borderValue**=morphologyDefaultBorderValue())¶

http://docs.opencv.org/modules/imgproc/doc/filtering.html?highlight=erode#dilate

You can find a tutorial on erosion and dilation using OpenCV at:

http://docs.opencv.org/doc/tutorials/imgproc/erosion_dilatation/erosion_dilatation.html

Furthermore, once the binary image with the object is obtained, it is interesting to obtain the moments of the object to determine its location (the centroid) on the image:

Moments **moments**(InputArray **array**, bool **binaryImage**=false)¶

http://docs.opencv.org/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html?highlight=moments#moments

To know more:

You can find a tutorial on the use of moments in OpenCV at:
http://docs.opencv.org/doc/tutorials/imgproc/shapedescriptors/moments/moments.html