UNIVERSIDAD PABLO DE OLAVIDE SEVILLA
ESCUELA POLITÉCNICA SUPERIOR

Robótica y Visión Artificial
Ingeniería Informática en Sistemas de Información

EPD 5: ROS and OpenCV (II). Template matching and depth information

Objectives

- Continue working with OpenCV: Template Matching
- Running the Kinect sensor under ROS
- Running EPD4 code over real-time data from the Kinect.
- Using the depth information from the Kinect

Assignments

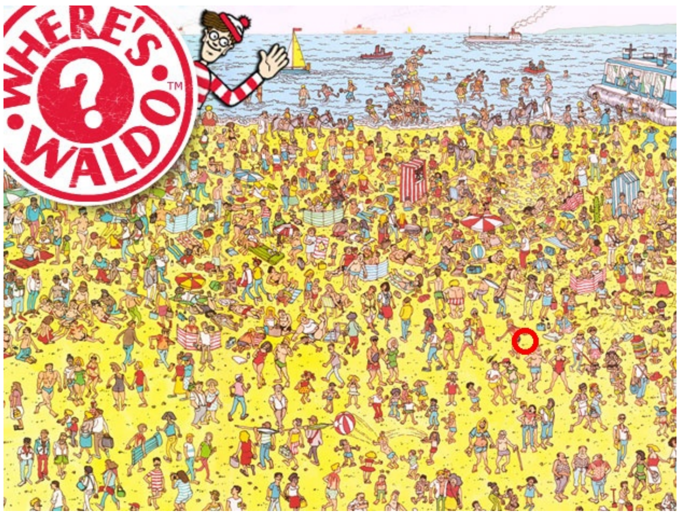
1 Materials

In this EPD we will continue working with ROS and OpenCV. See the EPD4 for further information.

Furthermore, we will use the Kinect sensors that are available in several computers in the lab.

2 Template matching

One function that we have seen in theory is template matching. The objective is to look for a template on an image, and determine the most likely location of the template.



The circle indicates the location on the image of the template on the right

OpenCV has a function to look for a pattern on an image:

C++: void **matchTemplate**(InputArray **image**, InputArray **templ**, OutputArray **result**, int **method**)

http://docs.opencv.org/modules/imgproc/doc/object_detection.html?highlight=matchtemplate#matchtemplate

This function returns in **result** a similarity map, in the form of an image. The maximum of this map will indicate where is the pattern located. To look for this maximum we can use the function:

C++: void **minMaxLoc**(InputArray **src**, double* **minVal**, double* **maxVal**=0, Point* **minLoc**=0, Point* **maxLoc**=0, InputArray **mask**=noArray())

http://docs.opencv.org/modules/core/doc/operations_on_arrays.html?highlight=minmaxloc#minmaxloc

Create a package (in your workspace src folder) called epd5, and depending on the packages roscpp, cv_bridge and image_transport.

Download the source files and create an executable called epd5_c1 for the node code in file nodec1.cpp

The source code at nodec1.cpp and student.cpp performs this operation. They use the function above for pattern matching.

Analyse the code in nodec1.cpp and the function **searchTemplate** *in student.cpp. Run the code and see how it works. Use the images waldo4.jpg and temp.jpg (this last one as template).*

To know more:
You can find a tutorial on template matching using OpenCV at:
http://docs.opencv.org/doc/tutorials/imgproc/histograms/template_matching/template_matching.html#

3 Drawing functions in OpenCV

OpenCV has several functions to draw on the images. This is quite useful for debugging and to show results.

In the code of the function processImage_c5 it is used the following function:

C++: void **`circle`**(Mat& **img**, Point **center**, int **radius**, const Scalar& **color**, int **thickness**=1, int **lineType**=8, int **shift**=0)

http://docs.opencv.org/modules/core/doc/drawing_functions.html#circle

which draws a circle on the image img, at the position center, with radius radius.

In the same way, it is possible to draw further figures, as a rectangle:

void **`rectangle`**(Mat& **img**, Point **pt1**, Point **pt2**, const Scalar& **color**, int **thickness**=1, int **lineType**=8, int **shift**=0)

http://docs.opencv.org/modules/core/doc/drawing_functions.html#rectangle

a line:

C++: void **`line`**(Mat& **img**, Point **pt1**, Point **pt2**, const Scalar& **color**, int **thickness**=1, int **lineType**=8, int **shift**=0)

http://docs.opencv.org/modules/core/doc/drawing_functions.html#line

etc.

To know more:
You can find a tutorial on drawing within OpenCV at:
http://docs.opencv.org/doc/tutorials/core/basic_geometric_drawing/basic_geometric_drawing.html

4 Running the Kinect sensor under ROS

In order to gather data from the Kinect sensor, and make them available to other ROS nodes, we should install the following ROS package if it is not already installed: openni_launch (http://wiki.ros.org/openni_launch) para la Kinect Xbox 360 o bien openni2_launch para la Asus XTion:

- ROS packages required for Kinect for Xbox 360:

```
sudo apt-get install ros-indigo-freenect-camera ros-indigo-freenect-launch
```

- ROS packages required for Asus XTion:

```
sudo apt-get install ros-indigo-openni2-camera ros-indigo-openni2-launch
```

In the wiki, you can see how to run it. It makes use of a new tool of ROS, roslaunch, which will be explained in future EPDs:

- Running the ROS node driver for the Kinect Xbox 360:

```
roslaunch freenect_launch freenect.launch
```

- Running the ROS node driver for the Asus XTion:

```
roslaunch openni2_launch openni2.launch
```

Once it is launched, you can try to see images from the Kinect running the following lines at a terminal:

```
rosrun image_view disparity_view image:=/camera/depth/disparity
```

to see the depth image, or the following:

```
rosrun image_view image_view image:=/camera/rgb/image_color
```

to see the color image.

Initiate the Kinect/XTion node and see the images from the Kinect/XTion.

It is important to notice that we have now 2 ways to work with the Kinect. We can play a bag file that stores Kinect data, as we did in the last part of EPD4. This is the same as working with the real Kinect. That is, your code will work the same with the real Kinect or with the recorded data.

5 Running your programs over the real-time Kinect data

The real Kinect publishes the same topics that the ones published when replaying the bag files. This means that we can use exactly the same code with the bag files than with the real Kinect.

Thus, in this last part, we will execute the developed code over the real data.

Once you have the Kinect node running, execute your segmentation code from EPD4 (C4) over the real data. Pick an object with a clearly defined colour and try to segment it (modify your code accordingly). Remember that you should modify student.cpp and student.h

We can use pattern matching to track an object on the data from the Kinect. Also, this will illustrate the limitations of the approach

Develop a new node (epd5_c2) that searches for an object on the images from the Kinect by using template matching. You can base your code on the code of question C4 of EPD4, using the functions of C1. Make a photo of some object and use it as pattern for the pattern matching module

6 Processing the depth information from the Kinect

In the code you can find a file called *nodec3.cpp*, which implements a node that subscribes to the colour image and depth information from the Kinect. The depth information is received as an image, which stores the depth for each pixel in meters.

As an example, it performs a thresholding operation over the depth field in order to segment everything that is below 75 cm.

Create a new node called epd5_c3 from the file nodec3.cpp. Compile the code and execute the node. Remember that you should launch first the Kinect-related nodes with openni2_launch or freenect_launch

Previously, go the file openni2.launch (or freenect.launch) in the openni2_launch folder (you can go there by typing `roscd openni2_launch`*) and search for the parameter* `depth_registration`*. Set it to* ***true***

As commented, this node subscribes to the topics that transport the images and depth from the Kinect. The tool rqt from ROS allows us to see the current nodes, and how they are connected:

Launch the tool rqt by typing:

```
rqt
```

in a separate terminal. Then, select, from the Plugins menu, Introspection, Node Graph

By using the depth information it is possible to refine the tracking of a colored object, using the depth to discard objects far or close to the Kinect.

Complete the code from EPD4 and the previous examples with the information from the depth of the Kinect to refine the results.