Computer Engineering Master Degree

Computer Vision

# Report about Project

Raffaele Di Nardo Di Maio 1204879

September 9, 2020

## 1 Project details

The goal of the experience was to estimate in an image the best match from a series of views of an object that we are trying to localize.

1. **Computation of $\gamma$-transform**
   this transform was applied to increase the contrast and stretch very high values in the images. In fact the values of the gamma values are all greater than 1. This transformation was applied only to views because of very strong brightness in them.

2. **Computation of Canny detection**
   Canny detector is applied to test images and also to $\gamma$-transformed views.

3. **Matching phase**
   this phase is performed using two different approaches, depending on specification of optional argument `-dist` (see Section 2.2). These approaches are:

   - **Template Matching with distance transform**
     The distance transformed is applied to each test image and then the function `cv::matchTemplate()` is applied to the result of the transform using Canny detected view as template. The metric used is the *Template Matching Correlation Coefficient (TM_CCOEF* in opencv*)*. Then I search the max value in the resulting matrix and using the position of that pixel as the position of the template for the match. With this approach I take only the ten best matches for each image, as specified in the assignment. The score value used to establish the best matches is the value contained in the max entry of result matrix.
   - **Template Matching with histogram refinement**
     The template matching is performed looking to TM_CCOEF as in the previous approach on the test image with detected edges and using view image with detected edges as template. To improve the results obtained, I don't take only the 10 best matches but the 50 best results and then I apply a refinement algorithm based on histogram. For each previously computed match I compute a sub-window, related to the position of found match and with size equal to the related view, and I filter it with the corresponding mask. This RGB sub-window of the original test image is then converted to the HSV space and its histogram of the Hue channel is computed. The resulting histogram is then compared with the Hue histogram of the original view image using `cv::compareHist()`. The final score of a match is the following:

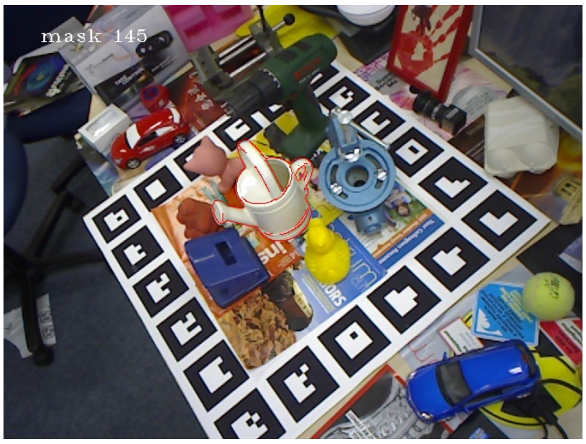
     $$final\_score = \alpha * match\_score + \beta * hist\_score$$

     where $match\_score$ is the score obtained by applying only `cv::matchTemplate()` and $hist\_score$ is the score obtained by `cv::compareHist()`. Hence from the set of 50 best matches only 10 best matches survive.

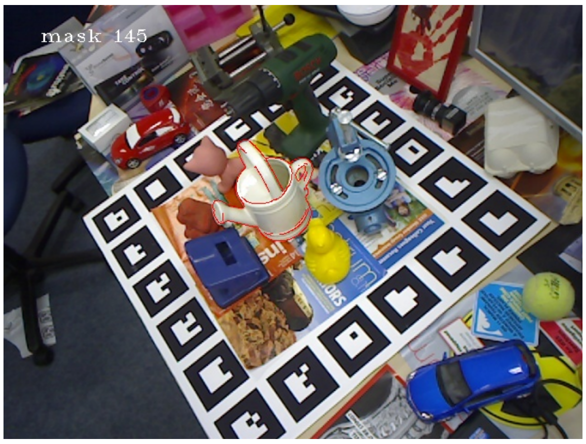
4. **Generation of result text files and result images**
   for each mask of 10 best matches, its egdes are highlighted in red and printed in position evaluated before over the image (see Figure 1). Each result image has a name with this format:

```
datasetNum_matchNum.jpg
```

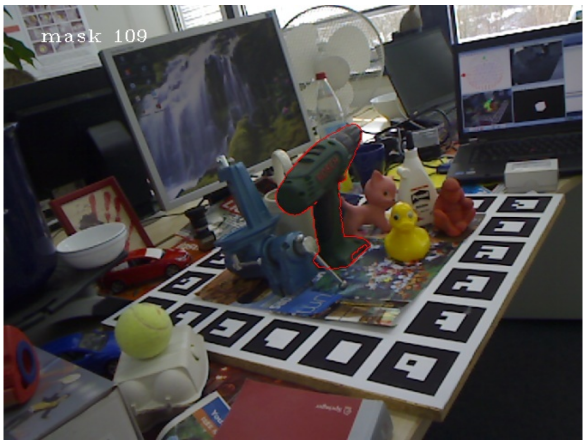
where `datasetNum` is the name of the dataset folder (it can be *can*, *driller* or *duck*) followed by the number of the corresponding test image and **matchNum** is the number of the corresponding mask/view. For each dataset the matches are written in the corresponding result text file with format specified in the assignment.



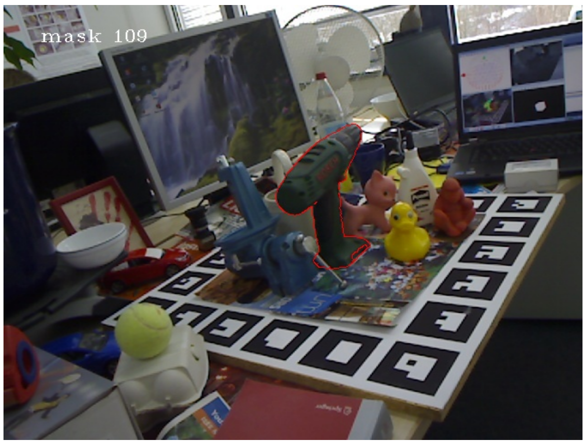
(a) can0_0.jpg



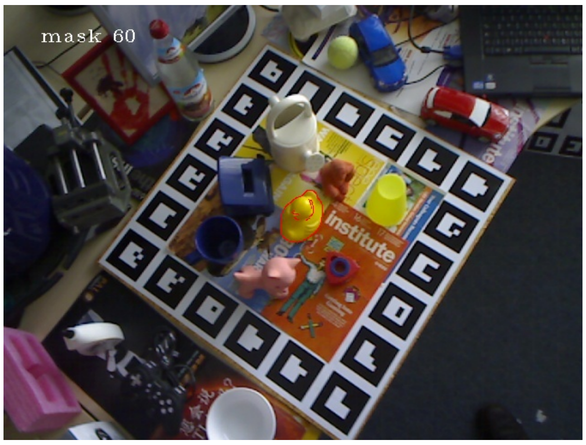
(b) can0_0.jpg



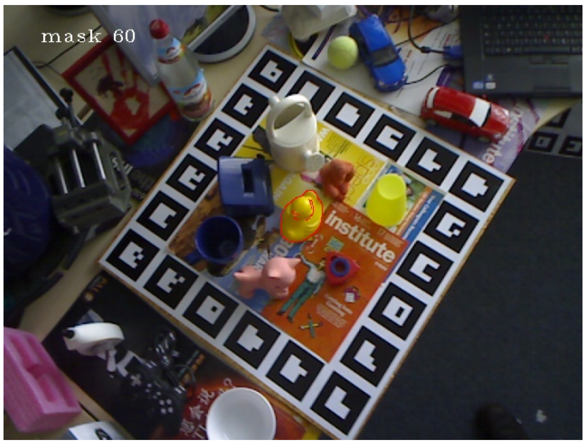
(c) driller0_0.jpg



(d) driller0_0.jpg



(e) duck0_0.jpg



(f) duck0_0.jpg

Figure 1: Example of result images with highlighted detected edges of views (left for Distance Transform approach, right for Histogram Refiniment approach).

## 1.1 Experimental results

Looking to details of previous analysed steps of the implementation, I explain the reasons and the results obtained with the previous operations:

1. **Computation of $\gamma$-transform**
The $\gamma$-transform was applied after different trials because of the precence of too many week edges and the colours of the views. Instead of using this transform, I tried also to equalize images before but the results were worst than obtained ones without any single-pixel operations. After different trials, the final values of $\gamma$ in $\gamma$-transform applied to each set of views are the following:

|  | **can** | **driller** | **duck** |
|---|---|---|---|
| $\gamma$ | 1.4 | 1.3 | 1.4 |

Table 1: $\gamma$ values for $\gamma$-transform applied on views of each dataset.

2. **Computation of Canny detection**
For each dataset I defined two different high threshold of Canny detector for each dataset: the first for the views and the second for test images. Then looking to the results obtained, I assigned to low thresholds of Canny detector the value of the high thresholds decreased of a specific quantity: 30 for views and 40 for test images. The values for high thresholds are the following:

|  | **can** | **driller** | **duck** |
|---|---|---|---|
| **test images** | 150.0 | 150.0 | 100.0 |
| **views** | 100.0 | 160.0 | 100.0 |

Table 2: High thresholds for Canny detector.

The values of high thresholds are both smaller than other ones for *duck* dataset because of need of more details for detection of shape of wing and beak.

3. **Matching phase**
In this phase for each image of each dataset, the two methods obtain both all the best feasible matches. For both approaches, the matching was more simple on **can** dataset and very complex for several images of **driller** and **duck** datasets. In particular the method with distance transform is more efficient in terms of memory occupation because it stores less results. In many images the histogram refinement approach obtains best results with wrong matches, positioning matches around the zone of the object to search even if they have a little bit wrong shapes.
The main problem of first method based on histogram refinement was to find the best thresholds for Canny detection for `driller` and `duck` because of some problems based on the shapes. In particular for `driller` matching, too small values for high Canny threshold help me to identify better the object in some images but loosing the correct matches on other ones. The same happens with too big values for high Canny threshold.
The reason comes from the level of contrast of different images and the smoothness of someone, that was very highlighted in several **can** images. The histogram comparison was performed only on Hue channel because of difference saturation and value of test image object and the corresponding model. For the same reason, to enhance the accuracy of comparison I did not work in RGB colour space. I tried another approach to estimate final score from histogram comparison without combining it with the previous one. I try to use only the histogram comparison score to discover from the set of 50 best matches the new 10 best ones. I used two approaches:

   - Take only matches with the score of histogram comparison greater than a specified threshold
   - Take the 10 matches with highest score of histogram comparison

For both methods, I notice that having higher scores of histogram comparison doesn't mean that there will not be false positive. In fact there were a lot of wrong matches and for this reason I decide to give an higher weight to previous Template Matching score and combine them in the following final score:

$$final\_score = 100 * match\_score + hist\_score$$

## 2 Code

### 2.1 Performance

The program can run analysing only one dataset or all the datasets in parallel. The management of this behaviour was done by using threads. Each thread performs the template matching on a dataset and the terminal window, for log printing, is the only resource managed using mutual exclusion.
Referencing to Section 1, I'm going to discuss the little tricks that I used to improved efficiency of the phases of the algorithm:

1. **Computation of $\gamma$-transform**
   To reduce the number of computation performed by the gamma transform, I compute only once, at the creation of the object related to Gamma Transform, a vector of size equal to the number of levels that has in position $i$ the value of the $\gamma$-transform for the input $i$.

2. **Computation of Canny detection**
   The computation of Canny detection for each view image is performed before the Matching phase and the results are stored. In this way then we need to compute the edges only once for each test image.

3. **Matching phase**
   To improve the efficiency of the buffer containing the best matches for each image of each dataset, I use a sorted vector (sorted by increasing score of matching) and I insert every element without loosing the sort. Every time that I obtain the best position/score of a mask in the template matching, I try to insert it in the best results buffer following this approach:

   **Algorithm 1: Insertion**

   ```
   1 if buffer is full then
   2     if match.score < buffer[0].score then
   3         buffer.insert_ordered(element)
   4 else
   5     buffer.insert_ordered(element)
   ```

   The insertion is based on the search of the position in the buffer for which the new match has a score that maintains sorted the vector. The manage of best matches for the created methods is done in the following way:

   - **Template Matching with distance transform**
     for each test image a buffer of max size equal to 10 is created. In the first buffer the best match of each view is inserted in the buffer with score obtained by `cv::matchTemplate()`.
   - **Template Matching with histogram refinement**
     for each test image a buffer of max size equal to 50 and then a buffer of max size equal to 10 are created. In the first buffer the best match of each view is inserted in the buffer with score obtained by `cv::matchTemplate()`. After inserting all the best matches for that image, I insert all of them in the second buffer with the score equal to final_score, previously described.

4. **Generation of result text files and result images**
   If the user decides to analyse all the datasets in parallel, each thread performs the creation of text result files and writes in each of them. The same happens for the creation of the images with 10 best detected matches. The use of threads was first of all added to write in parallel these files without loosing too much time.

### 2.2 Command line parameters

The program needs to have, as command line arguments, the following ones with the specified format:

```
-i input_path [-r results_path] [-o output_path] [-h] [-dist]
```

| | |
|---|---|
| **-i input_path** | *Mandatory argument*<br>input_path is the existing folder that contains the sub-folders of the three datasets(`can/`, `driller/` and `duck/`). Each one of this folders must contains also, as specified in the assignment of the project, the two sub-folders `models/` and `test_images/`. |
| **-r results_path** | *Optional argument*<br>results_path is the existing folder that will contain the text files describing the ten best matches for each image (`can_results.txt`, `driller_results.txt` and `duck_results.txt`).<br>If this argument isn't specified by the user, the text files are going to be stored in "default" path `../../../` |
| **-o output_path** | *Optional argument*<br>output_path is the existing path in which the program will save all the test_images, modified by printing the edges detected through Canny of the mask of the 10 best matches.<br>These are written on the image using red color and the name of the related model is shown in the high left corner of the image.<br>If this argument isn't specified, the result images with highlighted matches are not going to be store on disk. |
| **-h** | *Optional argument*<br>if this argument is specified, even if there are other arguments, the program exits printing the description of possible command line arguments. |
| **-dist** | *Optional argument*<br>if this argument is specified, distance transform method will be used in the computation of best matches, otherwise it will be used histogram refinement. |

The parser that process the command line arguments, detects also if some parameter is present twice and if "-i", "-r", "-o" are typed without the specified paths. An example of possible command on terminal on Windows can be the following:

```
./Project.exe -i ../../../dat -r ../../../results -o ../../../results -dist
```

After running the program, a Menu page will be display and the user can interact with it, selecting if he wants to apply Template Matching on a single dataset or in parallel to all the datasets. The user can also exit from the program using this menu that will be displayed again at the end of the computation desired.

## 2.3 Code Organization

The program is organized in 13 files that we can organized, looking to their functionalities, into the following sets:

- **Template matching**
  It's composed by `TemplateMatching.hpp` and `TemplateMatching.cpp` files that implement the two methods used in the program to estimate the best matches.
- **Gamma transform**
  It's composed by `GammaTransform.hpp` and `GammaTransform.cpp` files that implement the gamma transform, used in the program to change the contrast of test images.
- **Menu**
  It's composed by `Menu.hpp` and `Menu.cpp` files that implement the parser of the command line arguments and manage the lines to be shown on the screen.
- **Main activity**
  It's composed by `Project.hpp` and `Project.cpp` that manage threads that perform Template Matching.
- **Canny detection**
  It's composed by `CannyDetector.hpp` and `CannyDetector.cpp` that compute Canny Detection for a specified image.
- **Storing of best matches**
  It's composed by `BestResults.hpp` and `BestResults.cpp` that implement the classes:
  - `Result`
    it constructs the entity of result of match and implements the methods needed to access to its fields.
  - `BestResults`
    it constructs the buffer of results and manages the update of best matches for each image.
- **Utility constants**
  It's composed by `Utility.hpp` file and implements some useful values:

- Colour used in log printing
- Colour used in Menu
- Colour used for printing the mask name and the edges of the matched mask in the result image
- Gamma values for Gamma Transform
- Threshold values for Canny Detection
- Threshold values for Canny Detection
- Name of sub-folders needed to load images of datasets provided by user