COMPUTER VISION – January 2021 Project

Object Pose Estimation and Template Matching

You are asked to develop a system capable of automatically estimating in an image the best match from a series of views (i.e., models or *templates*) of an object that we are trying to localize. Each view corresponds to a position of the object with respect to the camera: this position can be used, for example, as input for a robot aiming to pick up the object. The goal (Fig. 1) is to extract the best matches among the provided views, by comparing each model for each possible x,y location (rotation is not required), and returning the views and the positions with the highest scores (or lowest distances).

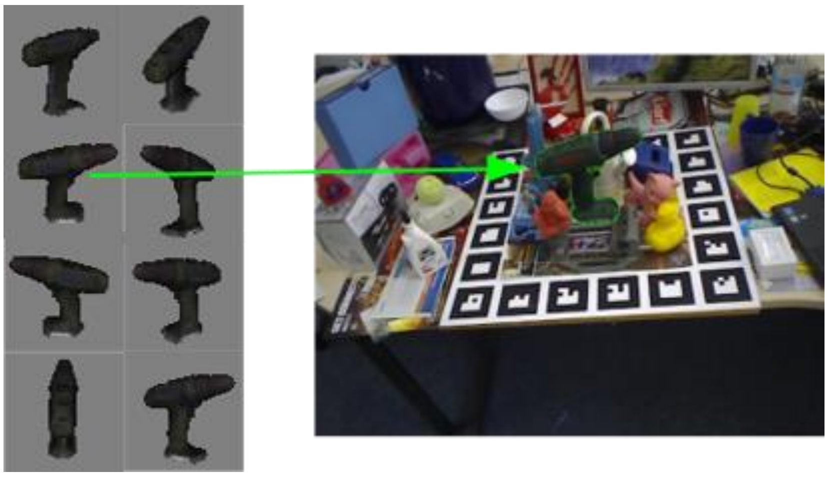

Figure 1: Find and locate inside the image the best matches among a set of views of the object.

The input of the system is a set of synthetic views of the object, extracted from a colored3D reconstruction of such object. In addition to the views, a set of masks are provided (Fig. 2), with same size as the view images, which distinguish in the views the object from the background (with pixel values of 255 and 0, respectively).

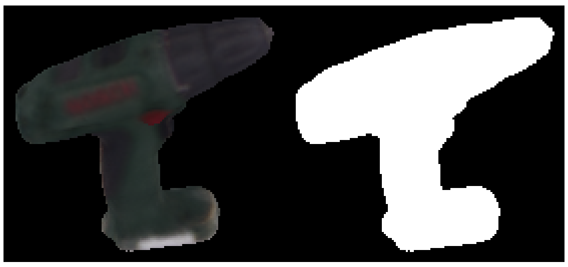

Figure 2: Example of a view (left) ant its corresponding mask (right).

It is suggested to use a robust template matching approach, but alternative approaches (appearance based or machine learning based) are not discouraged. In any case, the use of a simple template matching approach based on direct comparison of the color of the pixels is not allowed. Moreover, you can't use other information that does not derive from the models and the relative masks made available with this project. For example, you can't use any CAD model of the objects.

At this link:

https://drive.google.com/file/d/1qNuSTCsMpxGkVZn1gFncbb-44wOoW78U/view?usp=sharing

it is possible to download the evaluation images (ten for each object) along with the views/masks pairs. For each object (can, driller and duck) you will find 2 folders:

- models/
  - It contains 250 views (model0.png, model1.png,…) of the object and their respective masks (mask0.png, mask 1.png,…), both in png format.
- test_images/
  - It contains 10 test images, in jpg format

In addition to the source code and the project report that describes your approach and comments the obtained performance, three text files should be provided, one for each object, named can_results.txt, driller_results.txt and duck_results.txt, respectively. They summarize the obtained matching results and should be located in the root directory of your project. The format of these files is fixed: for each object, you should provide a file with exactly 10 lines, one for each possible test image. The first entry of each line is actually the name of a test image, followed by 10 tuples with the following structure:

```
<name of a specific template> <position x> <position y>
```

Each tuple represents a match, i.e. the matched template and the translation of such template with respect to the image. **<u>Important</u>**: the template reference frame is in **its top left corner (pixel 0,0)**, not in its center. The ten tuples represent the ten best matches for the corresponding test image, sorted from best to worst., i.e.:

```
<test image name> < first best match template name> < first best match position x> < first
best match position y> … < tenth best match template name> < tenth best match position x> <
tenth best match position y>
```

Here is an example results file for a specific object:

```
test0.jpg  model214.png  212  115  model12.png  433  256  model147.png  212  115  …
...
test9.jpg model11.png 45 277 model241.png 321 28 model88.png 77 600 …
```

To simplify the creation of such results files, a utility class `ResultsWriter` is provided with this project (defined in the sources `results_writer.h` and `results_writer.cpp`, see the `results_writer.zip` archive).

To use this class:

1) Include `results_writer.h`:

```
#include "results_writer.h"
```

2) For each object, declare an instance of `ResultsWriter`, e.g.:

```
ResultsWriter can_res("can"), duck_res("duck"), driller_res("driller");
```

3) For each of the **ten** test images (`test0.jpg...`, `test9.jpg`) of each dataset (can, driller, duck), add the **ten** best matches with the `addResults()` method, e.g., for the object **can** and the first test image (`test0.jpg`):

```
can_res.addResults("test0.jpg", "model123.png", 123,235); // The best match
can_res.addResults("test0.jpg", "model67.png", 87,77);
// …
can_res.addResults("test0.jpg", "model91.png", 348,135); // The 10th best match
```

<u>Important</u>: For each test image, the best matches should be added with `addResults()` strictly in order, from the best match to the 10th best match. In case, have a look to the results file obtained at the end of this procedure to double check the correct order of your best matches.

4) Finally, save the results with the `write()` method, e.g., for the object can:

```
can_res.write();
```

You should find in the same directory of your executable a file (`can_results.txt` in this case) that contains your results. Have also a look to the sample application `test_results_writer.cpp` in the `results_writer.zip` archive.

Submission

You should submit:

- All your code
- A report describing the details of the approach you developed (no page limit, but the evaluation will depend on the content, not on the number of pages!) – the report should include the details outlined above. The report should also include:
    - A discussion about the performance
    - An analysis of critical cases and of the images in which you get wrong results (if any)
- The results files can_results.txt, driller_results.txt and duck_results.txt (see above).

The C++ project should be provided with the CMakeLists.txt file, a directory src/ that includes all the c++ source files (.cpp), a directory include/ that includes all the c++ header files (.h), and a directory python that includes all the Python code (if any). No executables nor object files are needed. You should also submit the trained network – if it is too large to be uploaded on moodle, please share on some online services (e.g. dropbox, google drive, …). You can also attach your Colab notebook to the submission.

Evaluation criteria

The project will be evaluated based on the approach used, the results obtained, the critical analysis of the results. Your code may be tested on additional unknown images, and the results obtained are part of the evaluation. A minor weight will also be given to the quality of the code (e.g., all the code in the main function will result in a penalty).

Remember that the final project is an individual project. You cannot share your code. If you take some ideas from online sources, you should declare this in your code (in a comment) and in the report. Automated tools will be used to verify any plagiarisms.