UNIVERSITÀ DEGLI STUDI DI PADOVA

Object Pose Estimation and Template Matching

COMPUTER VISION – January 2021 Project

Candidates:
Federico Michelotto 1218322

Summary

To compile the code move in the folder ‘build” and execute the commands `cmake ..` and `make`.
To run the code execute the command `./project` in the “build” folder.
The program takes as optional argument the path of the folder containing the three datasets. By default the path is set to “../../../pose_estimation_dataset”.

Contents

Chapter 1

Used Approach

1.1 Introduction

The goal of this project is to detect in an input image, the 10 best templates (views) of a given object and the corresponding locations in the image. For this problem, the views of an object can be seen as a specific rotation of this object. From the specifications that we have received we do not need to rotate or scale the available views (250 for each object).
For this project I tried to implement the LINE-2D algorithm introduced in class when we talked about robust template matching techniques. This approach belongs to the family of shape-based matching techniques, and it is described in the paper *Gradient Response Maps for Real-Time Detection of Texture-Less Objects* of Hinterstoisser et al. [1].
Initially I tried a simpler approach based on features like SIFT, but I was not able to obtain satisfactory results, and so I decided to try to implement the above mentioned technique.

From an high level view, this technique can be described in the following steps:

1. Quantize the orientation space in $n_0 = 8$ classes.

2. For each template (view) T, extract a small set P of its most discriminant gradient orientations and store their corresponding locations.

3. For each image I compute one gradient image for each channel. For each location select the gradient orientation of the channel with the largest magnitude.

4. For each location c in the image, compute the following similarity measure

$$\mathcal{E}(I, T, c) = \sum_{r \in P} \left(\max_{t \in R(c+r)} |\cos(ori(T, r) - ori(I, t))| \right)$$

   where $R(c + r)$ is the neighborhood centered on the location $c + r$.

The idea is to give an high score to the locations c of the image if the gradient orientations of the image I at locations in $R(c+r)$, for each $r \in P$, are "similar" to the gradient orientations computed in the template T at locations $r \in P$.

Behind this high level schematic there are many implementation details. The most important ones will be discussed in the next section.

1.2 Implementation Details

1.2.1 Spreading the orientations & Precomputing Response Maps

As explained very well in [1], in order to avoid to compute the *max* over a set of quantized orientation, we can first spread the orientations found in the gradient images around their location over a small window, and then exploiting this compact representation of I to precompute n_0 lookup tables, one for each quantized orientation, in which we compute offline the *max* over all possible set of orientations. Note that since we are dealing with $n_0 = 8$ quantized orientations, there are $2^8 - 1$ possible set of orientations. More details can be found at [1]. In the paper the authors suggest a window with size 8. Actually I was not able to understand why an even number. In my implementation the size of this window is 3. This will be discussed later in chapter 2.

1.2.2 Orientation quantization

Note how in LINE-2D it is only considered the orientation of the gradient, not the direction. That is, we are interested only in angles in the range $[0, \pi/2)$. As in the paper, I quantized the orientation in 8 classes, with each class large $\pi/8$. Given an angle in radians between $[0, \pi/2)$, its class is computed as

$$C(\alpha) = \lfloor(\alpha * 8)/\pi\rfloor.$$

Where $C(\alpha)$ is an integer value between 0 and 7. It is then straightforward encoding these values in a byte in a one-hot encoding fashion. Let p be an orientation class, then its encoding $E(p)$ is $1 << p$ (where $<<$ is the left logical shift binary operator). If we convert this binary encoding in base 10, we get that $(E(p))_{10} = 2^p$. The encoding with all zeros, can be used to represent locations with a gradient magnitude too small.
As mentioned earlier, a location of an image can have more orientations associated to it. This can be simply encoded applying a bitwise OR of the single orientations involved.

1.2.3 Gradient computation

In the paper [1], the authors say that to compute the gradient orientation at a specific location, they firstly computed one gradient image for each channel (instead to convert the input image

to gray scale and then computing the gradient image), and then they selected the orientation of the channel with the largest magnitude, discarding the gradients with a low magnitude, and assigning to each location the most frequent orientation in a 3x3 neighborhood, but actually without specifying the order of this operations. Also they don't specify which operator they used to compute the derivatives, if Sobel or something else.
What I did is the following:

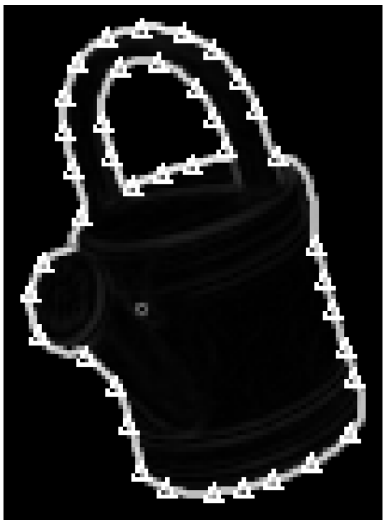
1. Apply a Gaussian blur to each channel separately with a kernel size of 3x3 and a standard deviation equal to 3.
2. Apply the Sobel operator with a kernel size equal to 3 to each channel separately .
3. Normalize the values of the magnitude images to 0-255.
4. Discard the gradient with a magnitude smaller than 10.
5. Select the most frequent quantized orientations in a neighborhood of 3x3.
6. At each location select the orientation of the channel with the strongest magnitude.

1.2.4 Keypoints selection

In the original paper, the authors say that the discriminant locations are selected considering their magnitude, and avoiding an accumulation of keypoints in single area.
Since they don't say how actually they have implemented this strategy, I did the following:

1. Compute the template gradients as explained in 1.2.3, but without applying the Gaussian blur, and instead of applying a threshold of 10, applying a threshold of 90 to the templates of the *can* and the *duck* dataset, and 30 to the templates of the *driller* dataset.
2. Order by decreasing magnitude the remaining locations.
3. Set a target number of keypoints equal to 40, and an initial distance between keypoints *radiust_dist* equal to 7.
4. Initialize a mask with the size of the template.
5. Select one keypoint at the time, starting from the one with the strongest magnitude.
6. Each time a keypoint is selected, disable through the mask all the locations that have a distance less than *radiust_dist*.
7. If there are no remaining keypoints to be selected and the number of selected keypoints is less than 40, restart from 4 but with *radiust_dist* decreased by one.

Some examples are shown in the next page.


(a) can

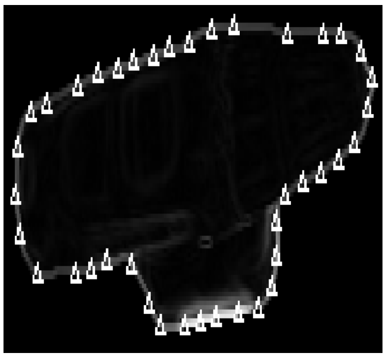
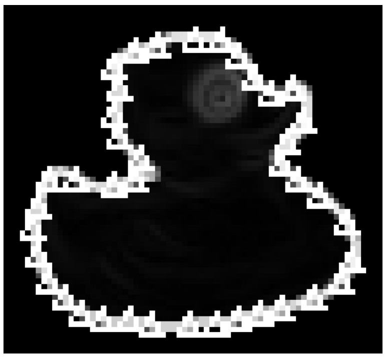

(b) driller

Figure 1.1: Keypoints localization examples.


(a) duck

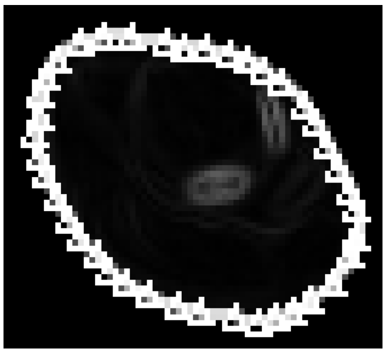

(b) duck

Figure 1.2: Other keypoints localization examples.

Chapter 2

Performances and Evaluations

On my old Intel i5 750, computing the score of a template on all the locations of an image requires something between 3 and 4 seconds. And to compute the scores of 250 templates on 10 test images, for 3 datasets, it took about 8 hours.
To this problem, in [1] is described also a more efficient solution, and the possibility to run this algorithm on a GPU to obtain real-time performances.
For what concerns the results obtained, for most of the images, in the first 5 positions is often present at least one "good" candidate with a reasonable position. But this is not always the case, like for example for the test image "test3.jpg" of the driller dataset (Fig. 2.1), in which none of the first 5 templates returned can be considered valid views.

In this case I think that the poor result is mainly due to many objects behind the target object.

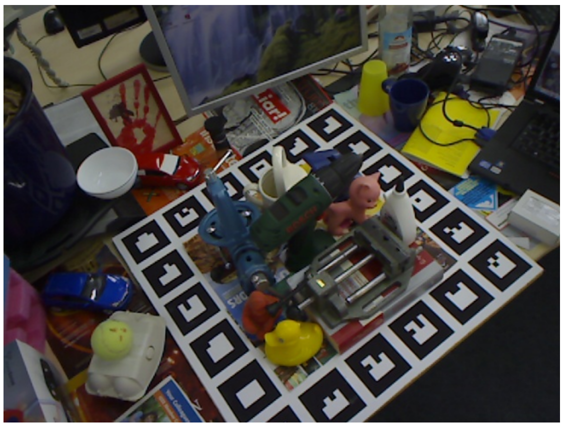


Figure 2.1: test3.jpg of the driller dataset.

Overall, for both the *can* and the *driller* dataset, the first template returned is a "valid" view

for 8 of the 10 test images. While for the *duck* dataset, for only 5 test images the first template returned is a “valid” view.
One thing that surprised me, is that with a maximum score of 40 (the number of keypoints selected in the templates), also the wrong templates got a very high score like 37-38, with a very small difference between the “good” ones. And I noticed that spreading more the orientations exacerbate this behaviour.
Finally I think that setting the parameters properly through more experiments, for example understand better how much Gaussian blur apply, or understand how select the keypoints in an optimal way, could lead to significant improvements.

Bibliography

[1] S. Hinterstoisser et al. “Gradient Response Maps for Real-Time Detection of Textureless Objects”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 34.5 (2012), pp. 876–888. DOI: 10.1109/TPAMI.2011.206.