Gesture Recognition

Andrea Dittadi, Davide Magrin, Michele Polese

July 29, 2016

Introduction

In this report we will introduce two descriptors that can be used for hand gesture recognition by means of machine learning techniques, and we will provide some performance analysis of such descriptors. We used a support vector machine (SVM) as classifier, training it with pairs consisting of some image descriptor and the corresponding desired output class. After the training phase, the SVM is used in this case to classify new hand gestures. In particular we used the `scikit-learn` library for Python from [1], [2].

In the following section we describe the input of our feature extraction procedure, and give a brief outline of the preprocessing that the raw images went through. In the next section, the idea behind the descriptors and the method to compute them is described. Finally, we give some results in order to assess the quality of the extracted features.

Preprocessing

The dataset contains 25 images for each of 22 different gestures, from 5 different subjects who repeated the gestures 5 times each. The images have information both on colour and depth, and the preprocessing phase is based on these data. For each image the preprocessing output consists of 2 files, described below.

The `preproc` files contain a basic structure of the hand, isolated from the background and with pixels classified with 3 labels (hand pixels, wrist and background). The estimated radius and center of the palm and an estimate of the orientation of the hand are also stored in this file.

The `clusters` files instead are the result of a clustering performed both on colour and depth information, aimed at isolating the fingers from the other parts of the hand. For each image, each pixel is classified with a label that specifies the cluster that the pixel belongs to. Unfortunately in most of the cases some fingers are split in more than one cluster, thus making more difficult the recognition of gestures in which fingers are arranged in a complicated pattern (see Fig. 1). For this reason, the descriptors we adopted do not rely too much on the clustering.

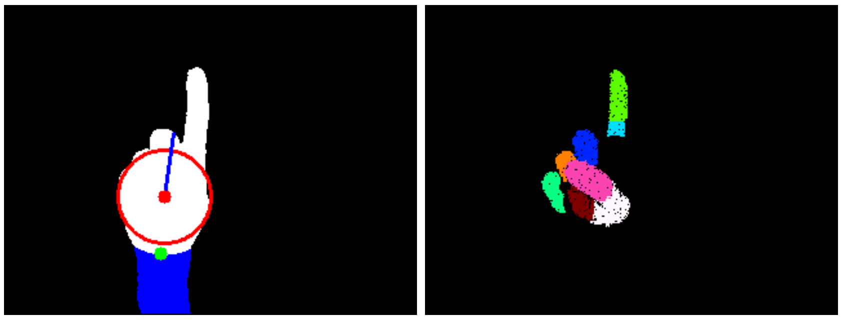


Figure 1: An example of not informative clustering

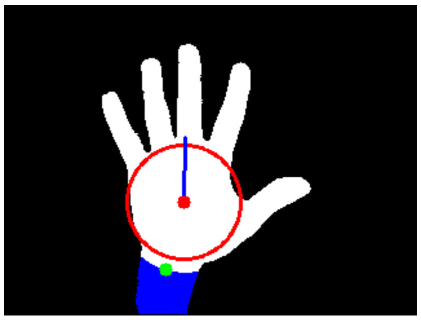
Features

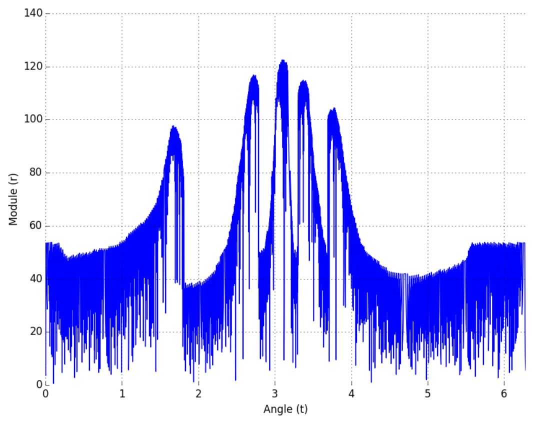
Perimeter features

This approach tries to extract a 1D signal from the available data, only keeping the most meaningful information contained in the 2D signal provided by the preprocessing. Since for each gesture the "state" of each finger can only be "open", "half closed" or "fully closed", the outer perimeter of the hand contains almost all of the information needed to distinguish between two different gestures. In some cases, there may be additional information that is not picked up by the perimeter (e.g. when a closed thumb may be visible or hidden behind other closed fingers), however these cases do not happen in the given dataset.

The starting point for the perimeter extraction is a list of cartesian coordinates `hand_pixels`, that represents the locations of the pixels recognized by the preprocessing to belong to the hand. These cartesian coordinates (x, y) are converted to polar by a function `cart2polar` that outputs the module and angle of each coordinate (r, t), using the center of the palm (estimated during the preprocessing) as the origin of the coordinate system. The function also supports a normalization of the angle coordinate with respect to the direction of the hand if this is provided. Once every cartesian coordinate has been converted to polar, a quantization of the angles is performed, using a rounding of the angle values. After this operation, for each quantized angle $\vartheta \in [0, 2\pi)$ there are multiple coordinates (r, t) for which $t = \vartheta$. For each ϑ, the perimeter method keeps the pixel with the maximum module r. This means that only the most distant point remains in the set, thus describing the perimeter of the hand. Since it doesn't matter what coordinate system is used by the features as long as this choice is coherent throughout the whole computation, a conversion back to the cartesian system is not necessary.

The result of the above computations can be visualized as a 1D function, for which the domain is $[0, 2\pi)$ and the value at each point is the module r corresponding to each angle. The perimeter of a sample gesture has been plotted in Fig. 2.


(a) Gesture



(b) Perimeter

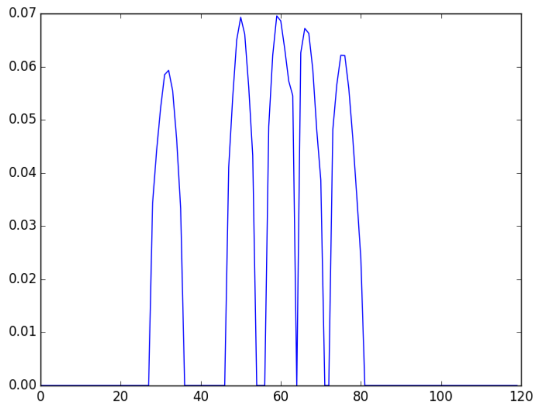

(c) Perimeter after filtering

Figure 2: Perimeter of gesture 1

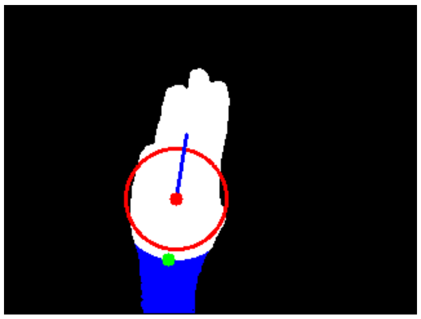
Since this computation of the perimeter yields a very noisy function, some filtering is necessary. First of all, in order to reduce the number of samples of the function, the domain is divided into 120 bins and the mean of the samples contained in each bin is used as the new signal, that is hence composed of 120 samples. After this preliminary step, the following thresholding is applied to the function:

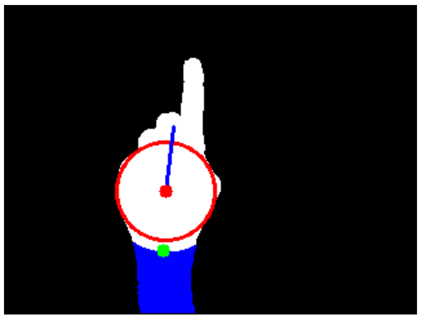
$$f(x) = \begin{cases} f(x) & \text{for } f(x) > \dfrac{M_f + 2m_f}{3}, \\ 0 & \text{otherwise,} \end{cases} \tag{1}$$

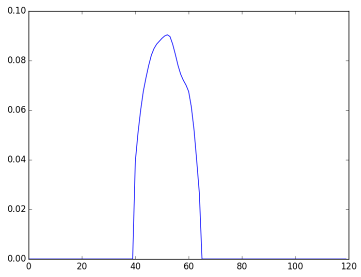
where M_f and m_f are respectively the maximum and minimum value of the function. Finally, the signal is filtered using a gaussian bell of length 9 and variance $\sigma^2 = 4$ and the values that were set as 0 by the threshold are set as 0 once again. This yields the final result represented in Fig. 2.

After these preliminary filtering operations, the signal is smooth enough to perform a search for local maxima that would yield a reasonably low number of false positives. The locations of every local maximum in the signal is memorized, and then only those peaks whose value is above a certain threshold are kept. This is done in order to minimize the occurrence of false positives, that have been observed to occur mainly when the fingers are closed, because of the noisiness of the signal near the knuckles. As the feature vector's length must be the same for all samples, if more than 5 peaks are found only the first 5 are kept as they most likely represent the fingers. If, on the other hand, less than 5 peaks are found, their locations are memorized in the first positions of the vector, that is padded with zeros to make it exactly 5 elements long.

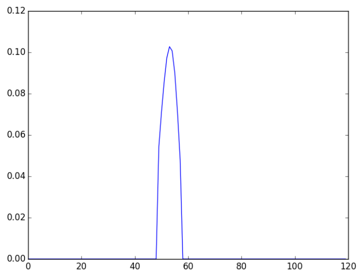
Next, for each of the peaks a corresponding "width" is computed in the following manner: first of all, the algorithm starts from the peak index, and it searches for the index of the nearest local minimum or zero both at the right and at the left of the peak. Then, the width is simply computed as the difference of the two indices. The procedure is described in Algorithm 1. Similarly to the procedure followed for the peak locations, if less than 5 peaks are found, the rest of the width feature vector is padded with zeroes. This metric is especially useful when the SVM needs to distinguish between gestures such as those depicted in Fig. 3.


(a) Gesture 2


(b) Gesture 3



(c) Gesture 2 perimeter



(d) Gesture 3 perimeter

Figure 3: Shapes and perimeters of two gestures that are identical if no peak width information is used

Finally, also the peak height is memorized, in an attempt to capture the "open", "half closed" or "fully closed" state of each finger. Once again, this vector is padded with zeroes to obtain a length of 5. The final feature vector is obtained by merging the three features described earlier, and has a length of 15. Prior to merging, in order to give the different metrics the same weight, they are all normalized so that all the feature values vary inside a predefined range.

If H is the number of pixels of the `hand_pixels`, the conversion from cartesian to polar is an $O(H)$ operation, as is the choice of the maximum module after the quantization of the angles. Since after the final quantization of the angle

Algorithm 1 Peak width

```
Require: maxima = array containing indices of local maxima
 1: for currentPeak = 0 to 4 do
 2:     idx = maxima[currentPeak]
 3:     if idx $\neq$ 0 then
 4:         i = idx
 5:         while i > 0 and filtered[i-1] $\leq$ filtered[i] and filtered[i-1] $\neq$ 0 do    ▷ Move to the left
 6:             i = i - 1
 7:         end while
 8:         w1 = i                                    ▷ Memorize the leftmost index of the peak width
 9:         i = idx
10:         while i < len(filtered)-1 and filtered[i+1] $\leq$ filtered[i] and filtered[i+1] $\neq$ 0 do    ▷ Move to the right
11:             i = i + 1
12:         end while
13:         w2 = i                                    ▷ Memorize the rightmost index of the peak width
14:         width[currentPeak] = w2 - w1              ▷ The width is the distance of the two indices found previously
15:     end if
16: end for
```

domain there are only 120 bins, the signal has constant length, and since it does not depend on the number of hand pixels anymore, all operations on this signal are $O(H)$. This yields a final computational complexity of $O(H)$.

Circular grid

The idea behind this kind of approach is to help the previous descriptor to distinguish gestures with similar perimeter but with fingers positioned on the palm in different ways. This difference cannot be captured properly by the perimeter, consider for example gestures 2 and 13. As it can be seen in Figures 4, 5 and 6 they share a similar perimeter, but the positions of the clusters of some fingers are quite different. Therefore the idea is to try to describe the clusters' positions in a hand-size-invariant, rotation-invariant way. The complete descriptor will be a vector with the two descriptors concatenated. Actually before the concatenation the vector of the second descriptor is multiplied by the max of the perimeter descriptor vector, in order to have values in the same range.

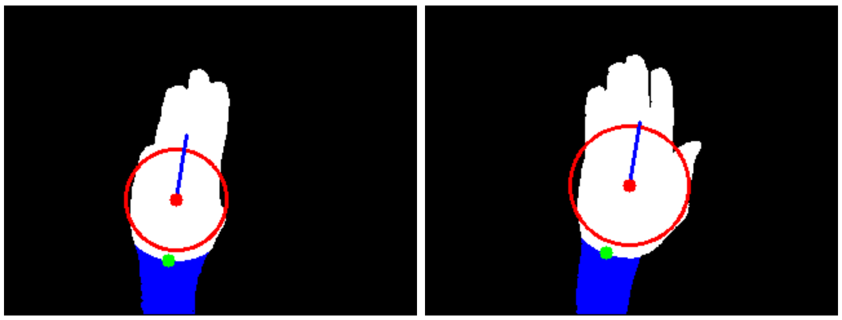
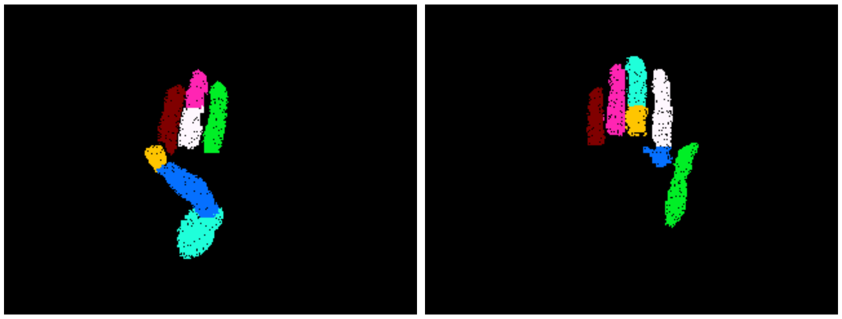

(a) 2 (b) 13

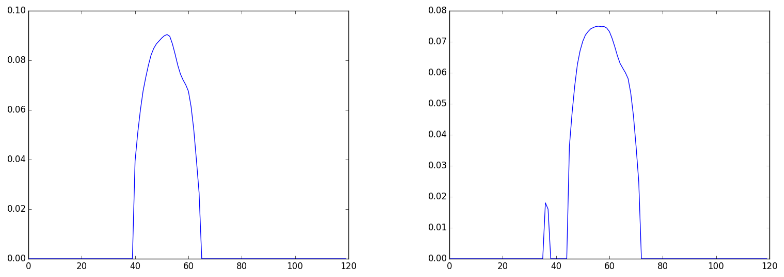
Figure 4: Shape of gestures 2 and 13

The first step is to divide the hand in a polar grid, with circular rings with a difference between radii proportional to the palm radius R_{palm} (hand-size invariance) and an angular width δ for each circular sector. In order to reach invariance to rotation the polar grid is rotated so that the direction of the hand orientation corresponds to an angle of 0 degrees. Let M be the number of circular rings, M_{inner} the number of desired circular rings which lie entirely inside the palm circle and $N = \frac{2\pi}{\delta}$ the number of circular sectors. Then $(M+1)N$ is the total number of sectors, including the external ones, i.e. the ones with external radius equal to infinity. For example, in Figure 7 there is a basic grid with $M = 3$, $N = \frac{2\pi}{2\pi/3} = 3$.



(a) 2 (b) 13

Figure 5: Clusters of gestures 2 and 13



(a) 2 (b) 13

Figure 6: Perimeter of gestures 2 and 13

The first circular ring has radii $R_1 = 0, R_2 = R_{palm}/M_{inner}$, and in general the k-th one has radii

$$R_1 = \frac{(k-1)R_{palm}}{M_{inner}}, \qquad R_2 = \frac{k\,R_{palm}}{M_{inner}} \tag{2}$$

Let $R_{circ} = R_{palm}/M_{inner}$. Then, as in the previous method, the polar coordinates (r,t) of each hand pixel are returned by the `cart2polar` function, with t normalized to the hand orientation. Each pixel is assigned to the proper sector, which is computed as

$$\frac{t}{\delta}\ \min\left\{\frac{r}{R_{circ}},\ M-1\right\}. \tag{3}$$

The last step cycles on all the clustered pixels and computes the total number of clustered pixels of each sector. Finally the descriptor is a vector with $(M+1)N$ entries, each of them being the total number of clustered pixels in a sector, normalized to the area A_{sec} of the sector. If the sector's radii are R_1 and R_2 with $R_1 < R_2$, then its area is computed as

$$A_{sec} = \frac{\delta}{2}(R_2^2 - R_1^2). \tag{4}$$

We observed that including in the descriptor the sectors which are external to the palm circle does not bring any improvement, on the contrary it worsens the performances with respect to the classifiers trained only with the perimeter descriptor. This is probably due to the fact that this information is already captured by the perimeter descriptor, thus the complete descriptor contains many correlated entries. Therefore only the sectors that cover the palm circle are considered for the computation of the complete descriptor. In particular the best combination is given by $\delta = \frac{2\pi}{3}$ and a single circle inside the palm, i.e. with $r = 0$ and R equal to the palm radius. Therefore the length of this descriptor is 3, and the complete descriptor will have length equal to 18.

An example is given in Figure 8, where only the two dark red and the dark green sectors of each image are considered. Notice the difference in the quantity of clustered pixels in the palm: gesture 2 has some fingers that overlap with the palm circle, therefore the number of clustered pixels in the sectors of interest is greater for gesture 2 than for gesture 13. This helps the SVM to properly classify the gestures: more comments on the improvement in term of performances will be given later.

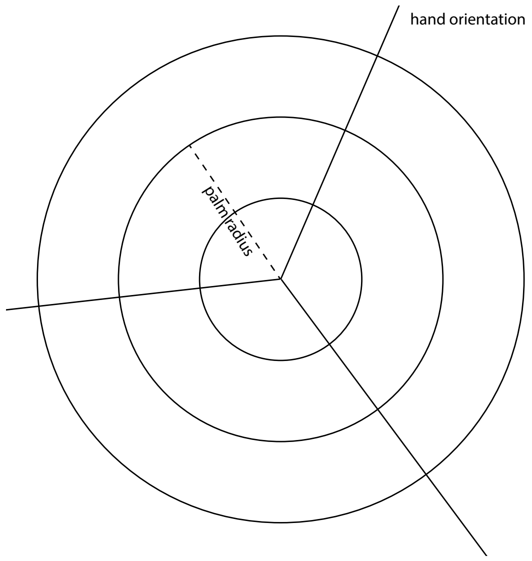
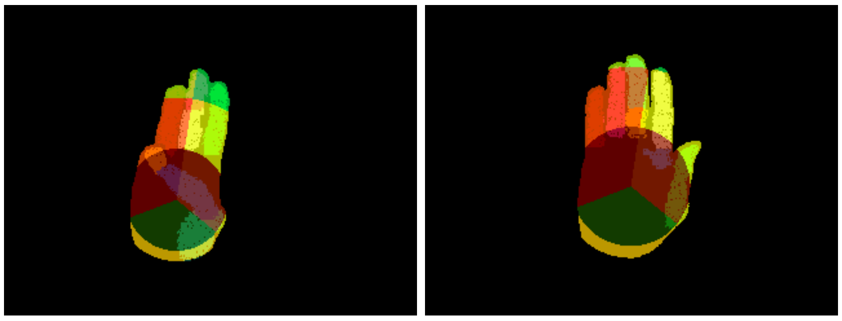


Figure 7: Polar grid example



(a) Gesture 2 (b) Gesture 13

Figure 8: Example of polar grid applied to two images

As a final consideration, the computation of this descriptor is fast, since once the H pixels of the hand are in polar coordinates it cycles through each of these only once, and then through each of the clustered ones another time, therefore the computational complexity is $O(H)$.

Results

In this section we evaluate the performances of the perimeter-based descriptors, and the improvement brought about by the additional set of descriptors based on the presence of clusters in the palm region. We performed feature extraction on the whole dataset, with the perimeter-based method and then with both methods. The quality of these descriptors is analyzed with Algorithm 2, which evaluates the average success rate for a given number of gestures m.

The success rate is computed for a given set $\mathcal{G}$ of m randomly chosen gestures, for all repetitions of each gesture, with training on 4 subjects and testing on 1. The training and testing sets of iteration i, for $i = 1, \dots, 5$, are respectively

$$S_{tr}^{(i)} = \{(s, g, r) : \ s \in \mathcal{S} \setminus i, \ g \in \mathcal{G}, \ r \in \mathcal{R}\} \tag{5}$$

$$S_{test}^{(i)} = \{(s, g, r) : \ s = i, \ g \in \mathcal{G}, \ r \in \mathcal{R}\} \tag{6}$$

where $\mathcal{S} = \{1, \dots, 5\}$ is the set of subjects and $\mathcal{R} = \{1, \dots, 5\}$ is the set of repetition indices. The average success rate for a set of gestures $\mathcal{G}$ is the average of the success rates defined above for $i \in \mathcal{S}$. The result is once again averaged over all the chosen sets $\mathcal{G}$ with m elements, to obtain an estimated average success rate for a subset of the dataset with m gestures. This metric is summarized in Figure 9. Note that in Algorithm 2 the value of N depends on m and the actual values are in Table 1.

m	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
N	6000	4800	3600	2400	2160	1920	1680	1440	1200	1080	960	840	720	600	480	360	240	15	10	5	1

Table 1: Number of iterations N for each number of gestures m

Algorithm 2 Success rate analysis

```
1:  for $m = 2$ to 22 do
2:      for $n = 1$ to $N$ do
3:          Draw a set $\mathcal{G}$ of $m$ gestures at random without replacement from the pool of 22 gestures
4:          for $i = 1$ to 5 do
5:              Train the SVM with the training set $S_{tr}^{(i)}$
6:              Test the SVM with the testing set $S_{test}^{(i)}$
7:          end for
8:      end for
9:      The success rate for $m$ gestures is the average over the success rates of all $5N$ iterations
10: end for
```

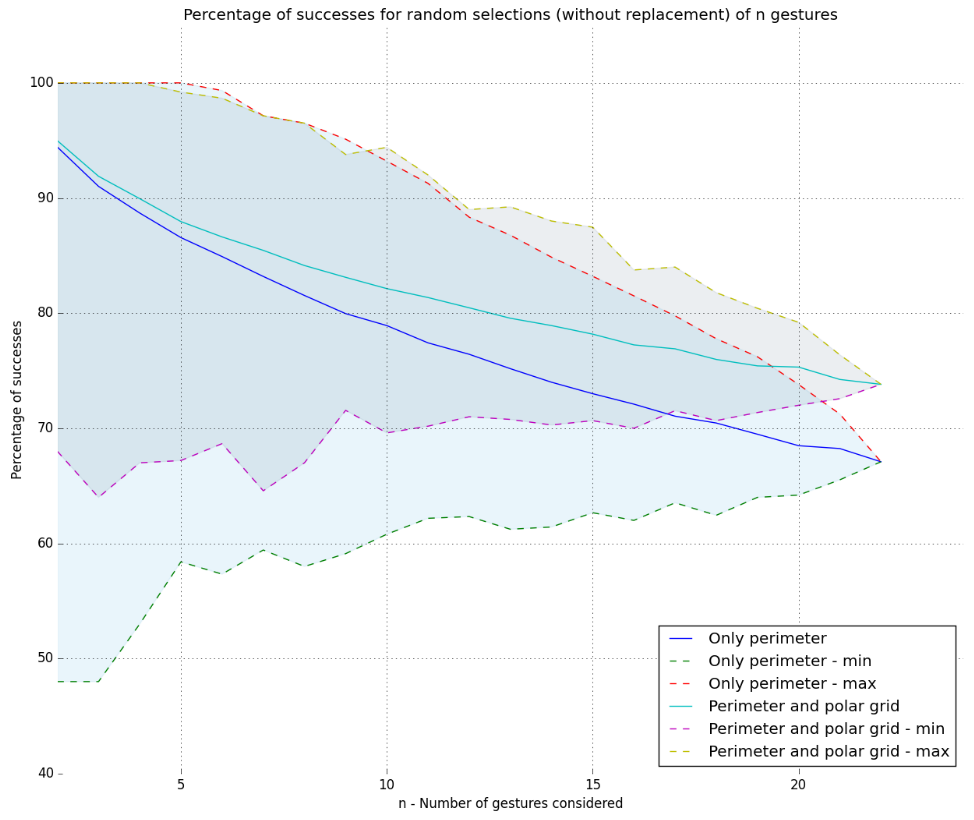


Figure 9: Success rate as in Algorithm 2

In Tables 2 and 3, instead, are reported the performance tables of the descriptors for training/testing with all the 22 gestures. Each row represents an actual gesture, and the j-th element of row i counts how many times gesture i was classified as j in the last iteration ($m = 22$, $N = 1$) of Algorithm 2.

If we consider gestures 2 and 13, introduced in the previous section in order to explain how the polar grid works, we obtain that:

- with the perimeter descriptor gesture 2 is correctly recognized in 8% of the trials (and it is recognized as gesture 13 in the 60% of the cases)
- with both the descriptors gesture 2 is correctly recognized in 60% of the cases, and never mistaken as gesture 13, yielding an improvement of 750%.

Gesture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
0	25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	24	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
2	0	1	2	0	0	0	0	0	0	6	1	0	0	15	0	0	0	0	0	0	0	0
3	0	0	0	14	0	0	0	0	1	0	0	2	0	0	0	0	0	0	0	0	0	8
4	0	1	0	0	23	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	10	9	0	0	0	0	0	0	1	5	0	0	0	0	0	0	0
6	0	0	0	0	0	2	16	0	1	0	0	0	0	0	6	0	0	0	0	0	0	0
7	0	0	5	0	0	0	0	12	0	1	4	0	0	0	2	0	0	1	0	0	0	0
8	0	0	0	0	3	0	0	0	10	0	0	2	0	0	0	0	5	0	0	0	0	5
9	0	0	3	0	0	0	0	0	0	19	0	0	0	0	0	0	0	0	0	0	3	0
10	0	0	2	0	0	0	0	1	0	0	21	0	0	0	0	0	1	0	0	0	0	0
11	0	0	1	1	3	0	0	0	0	0	0	16	0	0	0	0	0	0	0	0	0	4
12	0	0	0	0	1	0	0	0	0	0	0	0	21	0	0	0	0	0	3	0	0	0
13	0	0	8	0	0	1	0	0	0	0	0	0	0	14	1	0	0	1	0	0	0	0
14	0	0	0	0	0	3	2	0	0	0	0	0	0	2	16	0	0	0	0	2	0	0
15	0	5	2	0	0	0	0	0	0	0	0	0	0	0	0	18	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	20	3	0	0	0	0
17	0	0	1	0	0	0	0	4	0	0	1	0	0	0	0	1	3	15	0	0	0	0
18	0	0	0	0	1	0	0	0	0	0	4	0	1	0	0	0	1	0	16	0	2	0
19	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	1	0	22	0	0
20	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	22	0
21	0	0	0	8	0	0	0	0	0	0	0	3	0	0	0	1	0	0	0	0	0	13

Table 2: Perimeter descriptor, performance map: rows are the actual gesture, columns are the prediction

Gesture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
0	25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	23	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	16	0	0	0	0	0	0	6	1	0	0	0	0	2	0	0	0	0	0	0
3	0	0	0	18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	7
4	0	1	0	0	24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	13	7	0	0	0	0	0	0	0	5	0	0	0	0	0	0	0
6	0	0	0	0	0	4	15	0	0	0	0	0	0	0	6	0	0	0	0	0	0	0
7	0	0	5	0	0	0	0	12	0	1	0	0	0	5	2	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	21	0	0	1	0	0	0	0	3	0	0	0	0	0
9	0	0	3	0	0	0	0	0	0	16	0	0	0	0	0	0	0	0	0	0	6	0
10	0	0	3	0	0	0	0	0	0	0	22	0	0	0	0	0	0	0	0	0	0	0
11	0	0	1	0	0	0	0	0	2	0	0	20	0	0	0	0	1	0	0	0	0	1
12	0	0	0	0	1	0	0	0	0	1	0	0	20	0	0	0	0	0	3	0	0	0
13	0	0	1	0	0	0	2	0	0	0	0	0	0	19	2	0	0	0	1	0	0	0
14	0	0	0	0	0	3	2	0	0	0	0	0	0	1	17	0	0	0	0	2	0	0
15	0	4	3	0	0	0	0	0	0	0	0	0	0	0	0	18	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	1	18	3	1	0	0	0
17	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	4	19	0	0	0	0
18	0	0	0	0	2	0	0	0	0	0	0	0	2	0	0	0	3	0	18	0	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	0	0	1	0	20	0	0
20	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	22	0
21	0	0	0	8	0	0	0	0	3	0	0	3	0	0	0	1	0	0	0	0	0	10

Table 3: Both descriptors, performance map: rows are the actual gesture, columns are the prediction

References

[1] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python. 2001–.

[2] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.