Using a Naïve Bayes Classifier based on K-Nearest Neighbors with Distance Weighting for Static Hand-Gesture Recognition in a Human-Robot Dialog System

Pujan Ziaie, Thomas Müller, Mary Ellen Foster, Alois Knoll

Robotics and Embedded Systems Group, Department of Informatics,
Technical University of Munich, Garching, Germany
{ziaie, muelleth, foster, knoll}@in.tum.de

Abstract. We present an effective and fast method for static hand gesture recognition. This method is based on classifying the different gestures according to geometric-based invariants which are obtained from image data after segmentation; thus, unlike many other recognition methods, this method is not dependent on skin color. Gestures are extracted from each frame of the video, with a static background. The segmentation is done by dynamic extraction of background pixels according to the histogram of each image. Gestures are classified using a weighted K-Nearest Neighbors Algorithm which is combined with a naïve Bayes approach to estimate the probability of each gesture type. When this method was tested in the domain of the JAST human-robot dialog system, it classified more than 93% of the gestures correctly into one of three classes.

Keywords: Image Processing; Gesture Recognition; K-Nearest Neighbors; Naïve Bayes; Classification; Human-robot interaction

1 Introduction

When humans interact with one another—and with artificial agents—they make extensive use of a range of non-verbal behavior in addition to communicating via speech. Processing and understanding the non-verbal parts of human communication are crucial to supporting smooth interaction between a human and a robot.

We concentrate on the task of *hand-gesture recognition*: recognizing and classifying the hand shapes and motions of a human user in the context of a cooperative human-robot assembly task. Hand gestures play an important role in this type of interaction, both as an accompaniment to speech and as a means of input in their own right. For example, if a user wants to tell a robot to pick up a certain object among many other objects, it can be difficult to indicate the desired object using only speech. However, if the user combines saying “Pick up that object” with a pointing gesture at the target object, this can be easier to process. Hand gestures can also themselves provide strong indications of the user’s intentions in the absence of speech: for example, the user might move their hand near an object in preparation for

picking it up, or may hold out their hand to indicate that they need the robot to hand over a particular object.

In this paper, we introduce and evaluate a method to recognize the following three types of gestures in a human-robot dialog system: pointing, grasping and holding out (Figure 1). The paper is organized as follows. We begin by discussing related work in the area of gesture recognition: particularly, related approaches to the sub-tasks of image segmentation and classification. Next, we introduce the JAST human-robot dialog system, which supports multimodal human-robot cooperation on a joint construction task, and describe how hand gestures play a role in interactions with the system.

Figure 1: Pointing, grasping, and holding-out gestures

In the main part of the paper, we present a fast, robust and easy-to-implement gesture recognition algorithm which differentiates between three gesture classes mentioned above, and which can be extended to other similar applications. This algorithm proceeds in three main steps. The first step is to segment and label the objects of interest and to extract geometric invariants from them. Next, the gestures are classified using a K-nearest neighbor algorithm with distance weighting algorithm (KNNDW) to provide suitable data for a locally weighted naïve Bayes classifier. The input vector for this classifier consists of invariants of each region of interest, while the output is the type of the gesture. After the gesture has been classified, the final step is to locate the specific properties of the gesture that are needed for processing in the system—for example, the fingertip for a pointing gesture or the center of the hand for a holding-out gesture.

After the algorithm has been described in detail, we describe an experiment in which gestures from the JAST project were recognized and classified with an overall accuracy of 93%. At the end of the paper, we draw some conclusions and discuss possible extensions of this work.

2 Related Work

Many methods have been developed recently to perform successful gesture recognition. Most of these systems consist of two main steps: segmentation and extraction of invariants, and classification of gestures. In this section we discuss how similar applications perform these two steps.

2.1 Extracting Invariants

Invariants are shape descriptors extracted from an image that are independent of the viewpoint [16]. Using invariants for recognition greatly simplifies the process of object recognition because it allows objects to be compared with reference models regardless of the orientation. Before extracting invariants, it is necessary to segment the recognized image to extract the relevant objects or regions of interest and to omit the irrelevant data.

For hand-gesture recognition, some researchers have tried to do the early segmentation process using skin-color histograms [3, 4, 5, 6]. The problem with these methods is that they do not work well in cases when there are some other objects in the scene with the same color as skin color, or where the hand has other colors. In the target JAST application, the background is static and can easily be eliminated, so we concentrate instead on the geometric characteristics of the objects.

Zhou *et al.* [3] extracted invariants for gesture recognition using overlapping sub-windows, and characterized them with a local orientation histogram feature description indicating the distance from the canonical orientation. This makes the process relatively robust to noise, but very time-consuming indeed.

Kuno and Shirai [9] used seven invariants to do hand-gesture recognition, including the position of the finger-tip. This is not practical when we have not only pointing gestures, but also several other gestures, like grasping. However, the invariants they extracted inspired us for some future improvements.

2.2 Classification

Classification is a method to assign a class to a point (vector in spaces of more than two dimensions) in an N-dimensional space. The classes may be predefined and learned beforehand (supervised learning), or may be extracted automatically based on a similarity metric (unsupervised learning).

A *naïve Bayes* classifier assigns the most likely class to a given example given its feature vector, simplifying the task greatly by assuming that the features are independent given a class. Such classifiers are robust, simple to implement and computationally efficient—and, despite the often unrealistic assumption of independence, they are frequently very successful in practice. Many techniques have been developed to improve the performance of naïve Base classifiers; Zheng and Webb [2] provide an overview of efforts in this area.

K-nearest neighbors (KNN) classifiers have a good performance when the attributes of a system are linearly separable. It finds the K nearest (already classified) vectors in the space to the input. The class which has the most vectors in those K neighbors is chosen to be the class of the input vector. K-nearest neighbors with distance weighting (KNNDW) is an improvement which has been proved to perform better than KNN in many cases [8]. In this method, the contribution of each neighbor to the overall classification is weighted by its distance from the point being classified.

The most relevant work to our method has been performed by Frank *et al.* [1] which introduces a locally weighted naïve Bayes (LWNB) classifier. Their evaluation on UCI dataset shows that LWNB outperforms KNN and KNNDW when K is big

enough. Other refinements, like instance cloning local naïve Bayes (ICLNB) [7], have also been introduced which manipulate the training data to get a better performance from the Bayes classifier.

In our implementation, we use a combination of KNNDW and LWNB to get a better performance without manipulating the training data or any complicated modification. The complete explanation can be found in section 5.

3 The JAST Human-Robot Dialog System

The overall goal of the JAST project ("**J**oint **A**ction **S**cience and **T**echnology") is to investigate the cognitive and communicative aspects of jointly-acting agents, both human and artificial. The human-robot dialog system being built as part of the project [14, 15] is designed as a platform to integrate the project's empirical findings on cognition and dialog with research on autonomous robots, by supporting symmetrical, multimodal human-robot collaboration on a joint construction task.

Figure 2: The JAST human-robot dialog system

The robot (**Figure 2**) consists of a pair of mechanical arms with grippers, mounted to resemble human arms, and an animatronic talking head capable of producing facial expressions, rigid head motion, and lip-synchronized synthesized speech. The input channels consist of speech recognition, object recognition, robot sensors, and face tracking; the outputs include synthesized speech, head motions, and robot actions. The user and the robot work together to assemble a wooden construction toy on a common work area, coordinating their actions through speech, gestures, and facial motions. Joint action can take several forms: for example, the robot may ask the user to provide assistance by holding one part of a larger assembly, or by assembling or disassembling components.

Object and gesture recognition in JAST are both performed on the output of a single camera which is installed directly above the table looking downward to take images of the scene. The output of this process is sent to the multimodal fusion component [17], where it is combined with any spoken input from the user to produce combined hypotheses representing the user's requests.

4 Image Processing

The first step in the gesture-recognition process is to process the raw images from the overhead camera to extract the background and to identify Regions of Interest for the gesture-recognition process. Once those regions have been identified, we then extract geometric invariants from them for use in the classification process. In this section, we describe how these image-processing steps are carried out.

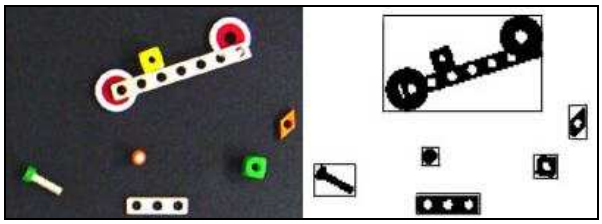

Figure 3: Result of the general segmentation process

4.1 Adaptive Segmentation

Within preprocessing of input images, segmentation is probably one of the most important steps as it is the basis of any fast recognition approach. Thus we need a robust and possibly adaptive method to extract Regions of Interest (ROI) from the image provided by the camera. Generally speaking, segmentation is done by classifying each pixel of the input image and organizing them into groups.

Due to the fact that we have variable lighting conditions and we also have to deal with changing but (mainly) uniform background-colors of the table, we need a segmentation approach that is capable of performing an adaptive loop when

environment parameters are modified (e.g. switching a light on or using a different table). We begin by presenting the basis of our segmentation approach to computing segments from an input image with objects in the foreground and a basically uncluttered background. We then describe a segmentation approach based on high-level abstraction that searches for a coherent set of ROIs.

The first step in segmentation is applying a threshold in order to classify each pixel as object or background. One could use a static or dynamic threshold for this task as described in [11]. In our case, we have a minimum and a maximum threshold based on evaluation of a multi-dimensional color histogram. As a second step, we group the classified pixels into blobs—the segments that will be our regions of interest in further analysis. This is done by connecting neighboring pixels using a search radius with a recursive algorithm. Some additional information that is useful in further processing such as the bounding-box, contour and main-color-components are computed alongside; note that this adds only a constant factor to the run-time of the algorithm.

Crucial to the approach is the *a priori* knowledge of the threshold, which is usually static as long as environmental conditions are stable and thus configured in advance. However, in the JAST setup, the threshold is not stable in this way, and so has to be computed online. For this task the system implements an adaptive loop which is initiated by the system when certain preconditions are complied with. These constraints are computed dynamically and include:

1. High fluctuation in number of regions segmented.
2. Perpetual distortion of regions.
3. Highly non-uniform movements of regions segmented.

Parameters for the preconditions are initiated in advance and adjusted online in a closed loop. When one or more of the preconditions is fulfilled, the following adaptive loop is initiated:

1. Evaluate a multi-dimensional color-histogram of the image to create a reference value r for the threshold, typically the peak in the histogram.
2. Randomly choose a set of parameters $\vec{p}$ for the thresholds.
3. Apply thresholds and grouping with a set of different grouping-search-radii to the original as well as a set of several downscaled versions of the image $N^{r,\vec{p}}$.
4. Compare results for the different images using a rating function (see below).
5. Perform a gradient descent to find a (locally) optimal set of parameters to be used subsequently.
6. Perform a gradient descent to find a (locally) optimal set of precondition-parameters for classifying the old segmentation-parameters as bad and new parameters as good.
7. Save and exit adaptive loop.

Step 4 in this loop makes use of a rating function to enable a high level quality measure. However this is obviously not an easy task as we neither know the number nor size or position of objects or gestures on the table. For this purpose we propose to raise the level of abstraction and evaluate results for a set of segmentation parameters.

First we compute a quadratic error for the selected parameter-set by comparing the predefined features of a segment (such as size, position, color, etc.). As we only want

to consider the closest matching region in two processed images of $N^{r,\vec{p}}$ we have to do a minimum operation after comparing the features f_s of a segment s in an image i to those of another image j.

$$E^{i}_{r,\vec{p}}(s,j) = \min_{s_j \in S_j}\left(\prod_{f \in F} \alpha_f (f_s - f_{s_j})^2 \right) \qquad (1)$$

Within the two best-matching segments of different versions of the original image, we multiply each segment-feature error by a feature-specific weighting factor α_f. Now we have to sum up all minimum quadratic errors to get a global error-result for all segments extracted with the current parameter-set $\vec{p}$ and reference value r in an image.

$$E^{i}_{r,\vec{p}}(j) = \sum_{s \in S_i} \left(E^{i}_{r,\vec{p}}(s,j) \right) \qquad (2)$$

Having computed a global error for the comparison between an image i and j of the set $N^{r,\vec{p}}$ we now sum up considering all images of the set.

$$E_{r,\vec{p}} = \sum_{i,j \in N^{r,\vec{p}}} \left(E^{i}_{r,\vec{p}}(s,j) \right) \qquad (3)$$

As a final step towards an abstract error measurement, we need to specify a quality-function in order to get a rate for the current segmentation parameter choice within [0,1]. We use the sigmoid function $\frac{1}{(1+e^{-x})}$ as a basis for this task:

$$q(r,\vec{p}) = 2\left(\left(1 + e^{-\lambda E_{r,\vec{p}}}\right)^{-1} - \tfrac{1}{2} \right) \qquad (4)$$

Here the global error is weighted with a factor $\lambda \in [0,1]$ that must be chosen carefully, also considering the feature weight α_f. The experimental system in the JAST setup showed good results for λ less than 0.1.

With this quality function we now can easily perform gradient descent or any other optimization method such as Gauss-Newton or Downhill-Simplex [11, 12, 13] to find a suitable set of parameters in our parameter space and so adapt our segmentation to variations in environment conditions.

4.2 Extracting Invariants

Once the regions of interest (ROI) have been identified as described in the preceding section, the next step is to extract the geometric invariants from the binary image for use in classification.

For each ROI we define the following invariants:

1. Number of the points
2. Length of the outer contour

3. Change of gradient in x direction
4. Change of gradient in y direction

Since the user's hands enter the image from outside the camera view, we consider for gesture recognition only those ROIs which end at one of the four sides of the image (table).

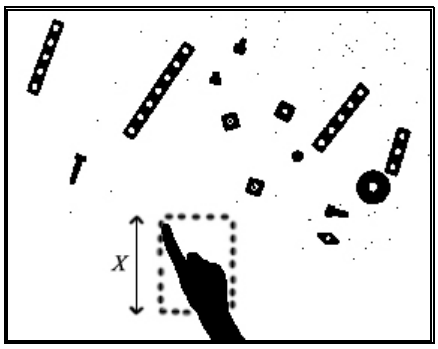

Figure 4: Predefined ROI height

For extracting the invariants, only a specific, predefined area of the end part of the ROI is processed. This way a completely stretched hand will be processed from the wrist to the finger tips. This area is depicted in **Figure 4**. Next, the outer contour of the object is extracted as shown in **Figure 5**. The length of this contour is the second invariant.

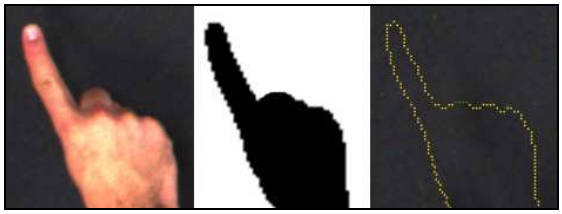

Figure 5: Extracting the outer contour from the segmented image

Then we explore the contour to find the x-y gradient changes, which corresponds to the number of changes in direction. We refer to these points as *gradient points*. To avoid noise, we inspect only the changes in direction which last for a known number of steps (three steps in our application).

Supposing that we have m invariants, we have a vector with m (which is four in our application) dimensions.

$$Inv(m) = \{a_1, a_2, .., a_m\} \quad (5)$$

During the training phase (Section 0), the resulting vector is added to the training pool; during the classification phase (Section 0), it is compared against the three gesture classes for identification.

5 Gesture Recognition

Before performing classification, a training pool is created based on a range of gestures produced by different users, where each training instance is labeled with its gesture class. The invariants from this pool are stored for use in the classification process. In Section 0, we describe the training process, while in Section 0 we describe how classification proceeds. In Section 0, we then show how the class-specific reference points are computed for pointing and grasping gestures.

Note that we are classifying static gestures, while the user's hands could be in motion. We therefore wait for the system to reach a stable state before performing training or classification. A stable state is detected by tracking the coordinates of the ROIs and initiating gesture recognition only once the coordinates remain constant for several frames.

5.1 Training Phase

For the training phase, the user moves his or her hand in different positions and angles for each of the gestures, using both the left and right hands. As stated before, we have three classes of gestures:

$$C(m) = \{c_1, c_2, c_3\} \tag{6}$$

All the extracted invariants are saved in a simple text file. It is recommended that the training is done with a couple of users with different hand size and shape so that the classifier becomes more robust. In our application we used four users' hands. For each gesture around 150 samples is sufficient, so at the end of the training process we have a file consisting of 500 to 600 labeled gestures.

The vectors in the file have one more dimension in comparison with the invariant vector because of the class-id. If we assume that there are n vectors in the file (n samples) then each vector will be:

$$\begin{aligned} &Tr_n(m) = \{i_0, i_1, ..., i_m\} \\ &i_0 \subseteq C \end{aligned} \tag{7}$$

After constructing this pool of labeled invariant vectors, classification is able to proceed.

5.2 Classification: Combining KNNDW & LWNB

To classify the extracted invariant, we first find the K nearest neighbors which are calculated based on the weighted distance of each training vector to the input invariant. Formally, we define the distance-weighting vector as:

$$W_{dist}(m)=\{wDist_1, wDist_2, .., wDist_m\} \tag{8}$$

The distance from the extracted invariant to training vector n can then be computed in Euclidean space as follows:

$$dist_n(Tr, Inv) = \sqrt{\sum_{i=1}^{m} \frac{(Tr_n(m) - Inv(m))^2}{wDist_m}} \tag{9}$$

We also normalize the distance so that all the values will be between 0 and 1.

Next, we choose the K vectors from the training pool which have the shortest distance to the given invariant.

$$\begin{aligned} c(x) &= \{Tr_x(1), dist_x\} \\ x &= \{1..K\} \end{aligned} \tag{10}$$

A normal naïve Bayes probability for the class of the given invariant will then be

$$\tilde{p}(C(j) \mid x) = \frac{\sum_{x=1}^{K} \delta(C(j), c(x))}{K} \tag{11}$$

$$\delta(a,b) = \begin{cases} 1 & if\ (a = b) \\ 0 & otherwise \end{cases} \tag{12}$$

where j is the index of each class (type of gesture).

To improve the result, we add weights to the neighbors. This weight $wB(x) = f(d_x)$ is a function of the Euclidian distance of each vector (d_x), which can be any monotonically decreasing function. In our application, we experimented with function like $f(d_x) = 1 - d_x$ or $f(d_x) = \frac{1}{(d_x)^p}$ for various p and the function which produced the best classification performance was:

$$wB(x) = f(d_x) = \frac{1 - d_x}{1 + d_x} \tag{13}$$

Using these weights, we define our locally weighted naïve Bayes probability by weighting equation (11) as

$$\tilde{p}(C(j) \mid x) = \frac{\sum_{x=1}^{K} wB(x)\delta(C(j), c(x))}{\sum_{y=1}^{K} wB(y)} \tag{14}$$

Then we can simply choose the class with the highest probability.

$$c(Inv) = \arg\max_{j=1..3} \tilde{p}(C(j) \mid x) \tag{15}$$

The classification algorithm can be summarized as follows:

Classification Algorithm

1. Find the k-nearest neighbors with weighted distance from the training pool
2. Find the naïve Bayes probability of each, while weighted disproportional to their distance

3. Choose the class of the vector with the highest probability

Figure 6: Localization results for gestures

5.3 Localization of gesture

Once the class of the gesture has been selected as described above, the next step is to identify the location of the important position of the gesture that is, the part of the gesture that is most important for determining the meaning of the gesture in the context of the system. Figure 6 shows the relevant points for the three gestures

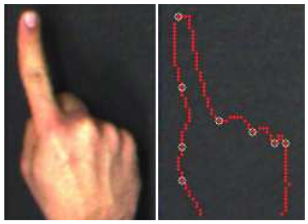

Figure 7: A pointing gesture with its gradient points

considered in this system.

For the holding-out gesture, all that is needed is the center of the hand, which is very easy to obtain. However, for pointing and grasping gestures, it is more complicated: for pointing gestures, what the system needs is the position of the finger-tip plus the angle of pointing-finger; for grasping gestures, it is the position between two grasping fingers. In the following sections we describe how we locate these positions in these types of gestures.

5.3.1 Pointing Gesture

A pointing gesture with its gradient points (see section 4.2) is depicted in Figure 7. As it can be seen, the furthest gradient point from the starting or ending of the contour is in fact the finger-tip we are looking for. Thus, if we have G gradient points, the gradients vector will be

$$P(g)=\{p_1, p_2,..., p_G\} \quad (16)$$

Considering *bp* as base-point to be the point in the middle of starting and ending points of the outer-contour and d as the Euclidean distance between two points, the finger-tip, *tip*, will be

$$tip(p_j) = \arg\max_{j=1..G} dist(bp, p_j) \qquad (17)$$

In the old version, the finger-tip was assumed to be the "highest" point of the outer contour. While this assumption is true for upward pointing gestures, it does not hold when the point is to the right or the left. We therefore improved the algorithm by using gradient points.

5.3.2 Grasping Gesture

In the grasping gesture, finding interesting location (between the grasping fingers) is not as simple as finding the finger tip in the pointing gesture. We can observe that the

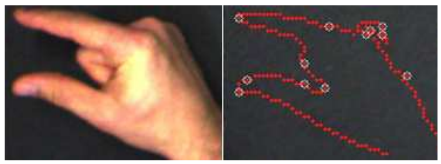

Figure 8: A grasping gesture with its gradient points

space between two grasping fingers is actually the area where most of the gradient points occur. Therefore, the goal is to find the center of the smallest area which has the maximum number of gradient points.

To find the coordinate of this center, we do the following steps. First we weight each point according to its distance from the other points. The value of each point will be the sum of the reciprocal of the normalized distance of it to the other points. Based on (9), (12) and (16) it will be

$$weight(p) = \sum_{g=1}^{G} (1 - \delta(p, p_g)) * \frac{1}{dist(p, p_j)} \qquad (18)$$

To avoid noise, we ignore points which are very close to the base-point.

Then the center of the area between grasping fingers, *gc*, can be estimated by calculating the center of the gravity of all these weighted points, which is

$$gc_x = \frac{\sum_{g=1}^{G} weight(g)\, p_x(g)}{\sum_{g=1}^{G} weight(g)}$$
$$gc_y = \frac{\sum_{g=1}^{G} weight(g)\, p_y(g)}{\sum_{g=1}^{G} weight(g)} \qquad (19)$$

This method works well and can estimate the location of grasping center with an acceptable precision.

6 Experimental Results

To testing the recognition algorithm we constructed a training pool with less than 200 samples for each of the gestures, for a total of 580 samples in the training pool. These samples were made by three persons in different lighting conditions. Then we created a testing pool with about 40 samples for each gesture by a person other than those three whose gestures were represented in the training pool.

The performance without weighting the invariants for identification of each of the gesture types, along with the overall performance, are shown Figure 9.

The x axis of this graph represents the value of K for the K-nearest neighbor

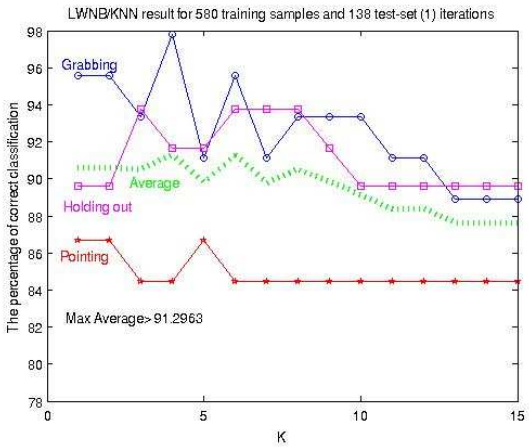

Figure 9: LWNB and KNN results without distance weighting

selection, while the y axis shows the percentage of gesture instances of each type that were correctly classified. The highest overall performance 91.3% correct classification at K=4.

After trying many combinations of weights on the members of invariants, we found the best weighting vector (*wDist*) to be $W_{dist}(m) = \{0.6, 0.6, 1, 1\}$. That is, the gradient changes are both weighted at 1.0, while the number of points and contour length are weighted at 0.6. This result is intuitively acceptable: the range of object-points and length of contour is much wider than the number of changes in gradients.

Testing the recognition system with distance weighting, we achieved the results shown in Figure 10. It is quite obvious that after applying distance weighting the performance increases and the fluctuation decreases. We observed that when K is bigger than 7-8 is the performance starts to sink.

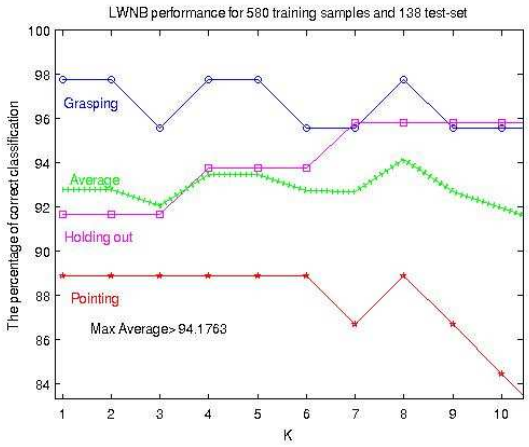

Figure 10: Classification result with distance weighting

The best performance seemed to be around K between 4 and 7. Table 1 shows detailed results classification results for values of K between 4 and 6.

Table 1: Performance results of classification for different K

	K=4	K=5	K=6
Average No weighting	91.29	89.81	91.25
Average (DW)	93.47	93.47	92.73
Pointing	88.88	88.88	88.88
Grasping	97.77	97.77	95.55
Holding out	93.75	93.75	93.75

Running the full gesture-recognition process on a frame takes less than 50 msec on average. Of this time, segmentation takes 20-30 msec, while the recognition process takes 10-20 msec.

7 Conclusions and Future Work

We have described a static gesture recognition method to distinguish between three gestures types: pointing, grasping and holding out. The process is based on classifying invariants of image blocks using a locally weighted naïve Bayes and K-nearest neighbors classifier.

The segmentation (pre-processing) is done by an adaptive method of extracting the background, which is considered to be static (the color of the surface of the table), and segmenting the remaining pixels into regions of interest afterwards.

Four invariants of each ROI are then extracted. These invariants are: Number of pixels, length of the outer-contour and changes in x and y gradients.

The extracted invariants are then compared against the invariants in a pool of labeled examples created during a training phase for each type of the gestures. The best suitable type of gesture is then given by using a locally weighted naïve Bayes classifier which is fed by the K-nearest neighbors of each invariant in the invariants pool.

After classification, an appropriate algorithm is applied according to the obtained type of the gesture to get the important information of that certain gesture. This required information is the finger-tip and its angle for pointing gestures, area of grasping for grasping gestures and the center of hand-pit for holding out type.

In an experiment, the whole process takes less than 50 msec in total and has an overall performance of about 93% at identifying the correct gesture type.

There are several improvements we intend to work on in the future. First, we will make the K-nearest neighbor algorithm adaptive. This means that we will modify the value of K in cases where the probabilities of two gesture types are very close in order to produce better classification in these difficult cases.

Another improvement will be add and test more invariants, for example like those used in [9]. The method to extract change in gradient by following the contour should also be improved to be more precise and robust under noisy conditions.

We are also contemplating exploring the performance of other methods such as genetic algorithms to help in determining the best weighting vector.

The performance of the full JAST system will shortly be tested through a user evaluation. The results of this study will provide a useful indication of how the gesture-recognition component performs in practice. In future versions of the system, we may incorporate information from other input-processing components of JAST—for example, face and gaze tracking—to help make a better decision under uncertain conditions.

Acknowledgements

We thank the members of the Cognitive Robotics Lab at TUM for useful discussions and suggestions, particularly Thomas Rückstieß, and Christian Osendorfer.

This research was supported by the EU project JAST (FP6-003747-IP), http://www.euprojects-jast.net/.

References

1. E. Frank, M. Hall, and B. Pfahringer: Locally Weighted Naïve Bayes. In: Nineteenth Conference on Uncertainty in Artificial Intelligence (UAI2003), 2003
2. Z. Zheng and G. I. Webb: Lazy learning of Bayesian rules. Machine Learning, 41(1), 53-84 (2000)
3. H. Zhou, D. J. Lin and T. S. Huang: Static Hand Gesture Recognition based on Local Orientation Histogram Feature Distribution Model. In: Proc. IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshop, pp. 161 (2004)
4. H. Hongo, M. Ohya, M. Yasumoto, K. Yamamoto: Face and hand gesture recognition for human-computer interaction. In: Proc. IEEE 15th Int. Conf. Pattern Recognition, vol 2, pp. 921-924 (2000)
5. H. Wu, T. Shioyama and H. Kobayashi: Spotting Recognition of Head Gestures from Color Image Series. In: Proceedings of the 14th International Conference on Pattern Recognition, vol. 1, pp.83-85 (1998)
6. H. Boehme, et al.: User Localization for Visually-based Human-Machine-Interaction. In: Proceedings International Conference on Automatic Face- and Gesture Recognition , pp.486-491, IEEE Computer Society (1998)
7. L. Jiang, H. Zhang and J. Su: Learning K-Nearest Neighbor Naïve Bayes for Ranking. In: Lecture Notes In Computer Science, vol. 3584, pp. 175-185, Springer (2005)
8. R. L. Morin, D. E. Raeside: Reappraisal of Distance-Weighted k-Nearest Neighbor Classification for Pattern Recognition With Missing Data. In: IEEE TRANS. SYS , MAN, AND CYBER. Vol. SMC-11, no. 3, pp. 241-243 (1981)
9. K. Kuno, Y. Shirai: Manipulative hand gesture recognition using task knowledge for human computer interaction. In: Proc. Third IEEE International Conference on Automatic Face and Gesture Recognition, pp. 468-473 (1998)
10. A. McIvor: Background subtraction techniques. In: Proc. Of Image and Vision Computing, Auckland, New Zealand (2000)
11. D. W. Marquardt: An Algorithm for Least-Squares Estimation of Nonlinear Parameters. In: SIAM Journal on Applied Mathematics. SIAP, vol. 11(2), pp. 431-441(1963)
12. J.A. Nelder, and R. Mead: A Simplex Method for Function Minimization. In: Computer Journal **7** (4), pp. 308-313, (1965)
13. J. C. Lagarias, J. A. Reeds, M. H. Wright, and P. E. Wright: Convergence Properties of the Nelder-Mead Simplex Method in Low Dimensions. In: SIAM Journal on Optimization, vol.9 (1), p112-147 (1998)
14. M. E. Foster, T. By, M. Rickert, and A. Knoll: Human-robot dialogue for joint construction tasks. In: Proceedings of the 8th international conference on Multimodal interfaces, pp. 68-71 (2006)
15. M. Rickert, M. E. Foster, M. Giuliani, T. By, G. Panin, and A. Knoll: Integrating language, vision, and action for human robot dialog systems. In: Proc. HCI International, vol. 4555, pp. 987-995, Springer (2007)
16. I. Weiss: Geometric invariants and object recognition. In: International Journal of Computer Vision 10(3), pp. 207-231 (1993)
17. M. Giuliani and A. Knoll: Integrating multimodal cues using grammar based models. In: Lecture Notes in Computer Science: Universal Access in Human-Computer Interaction. Ambient Interaction, vol. 4555, pp. 858-867 (2007)