Gesture recognition in multimodal dialog systems

Christian Ludwig[1]

[1]Quality and Usability Lab
Department of Software Engineering
Institute of Technology
Berlin, Germany
cludwig@cs.tu-berlin.de

Abstract

This is an introduction to gesture recognition in multimodal dialog systems. The article demands only basic previous knowledge of multimodal dialog systems in general and gesture recognition techniques in special.

This paper is meant to be the right starting point, if you want to get a quick overview of the approach of recognizing gestures with the help of computer systems.

Index Terms: gesture recognition, classification, feature detection, modeling, gesture analysis, overview

1. Introduction

Human-computer interaction (HCI) is becoming more and more popular these days, due to the fact that processing power increases steadily. The goal of usability aspects in HCI is to make interfacing with computers much more natural. One major use is in sign languages. With a good sign language recognition in place, deaf people could greatly benefit from it. Another possible use is to track persons with cameras and be able to provide them with information they need at just the place they are at the moment on a nearby screen ([3]). This scenario could be extended. With gestures it would be possible to send commands to control the computer system.

To successfully recognize gestures we need to know what gestures are and how they integrate into the conceptual structure of multimodal dialog systems (MMDS). We choose the structure outlined by López-Côzar in [1]. After that we will see how to model gestures, which problems show up here, and how to solve them. We will then analyze the model and based on that analysis make a decision, whether we have successfully recognized a gesture or not.

2. Gesture classifications

In MMDS gestures are an input modality to a computer system. They aim to provide easy human-computer interaction.

There are a multitude of different classifications for gestures. For example, we can classify gestures by their granularity, where we divide between *gross-grained*, *medium-grained* and *fine-grained* gestures. Gross-grained gestures describe the movement of the whole body. For medium-grained gestures only parts of the body move, e. g. arms or legs. Fine-grained gestures describe the movement of small parts of the body, e. g. fingers.

Gestures can also be divided between being *communicative*, which means they support the spoken word. They also can be *manipulative*, so that they are the central action in a conversation and the spoken word supports them. In natural language most gestures are communicative. For HCI we mainly focus on manipulative gestures, since we can use them to control a computer system.

Gestures can also be classified by their expression. There we divide between many different groups. With *symbolic gestures*, we support an opinion by nodding or shaking the head. *Deictic gestures* are manipulative and are used to point to items. They can also be used to show routes on an interactive map, by touching on a screen and pointing to the start and end of the route. *Iconic gestures* describe objects and their relationships, *metaphoric gestures* express abstract ideas and *rhythmical gestures* appear synchronously in conjunction with spoken words.

We can see from these classifications, that we perform most gestures with our hands. That means hand gestures are the most important ones to recognize. They allow us to give manipulative gestures as commands to a computer system in a natural way.

There are many other ideas on how to classify gestures. Pavlovic et al. define them in the following way for hand gestures ([2]).

> "A hand gesture is a stochastic process in the gesture model parameter space M_T over a suitable defined time interval I."

This definition is well suitable for gesture recognition in general.

3. Approach to gesture recognition

Technically there are two possible ways to track gestures with computer systems. One is the *intrusive* way, where the users wear a data glove. We mainly get relevant information from such a device, since we directly attach it to the interesting limbs. The other one is the *non-intrusive* way, where we do not force the user to change his behavior while tracking him. Cameras seem to best fulfill this requirement, since they mimic the same behavior as humans do with their eyes. That makes them more natural. The drawback, on the other hand, is that we need to extract our input data from the camera device. We need to distinguish between relevant and non-relevant data. Nevertheless vision-based systems seem to outnumber all others, because algorithms can be shared between other input modalities in a multimodal system as well.

Devices for an intrusive approach include keyboards, mice and data gloves. For some applications, like examining actions performed at a surgery, data gloves give the most detailed and relevant results.

For the approach to gesture recognition, we need to have an idea of the setup, with which we will recognize the gestures. An information system has different needs, than a home entertainment system. So as a first step we need to build an appropriate model for the specific gestures, which we want to recognize in our system. That model greatly depends on your specific task.

Next, we need to analyze the data from the model. There are some parameters in the model, which need to be tuned throughout the process.

After that we need to make a decision, if we have a successfully recognized gesture.

4. Object modeling

There are basically two kinds of models used for gestures, *2D models* and *3D models*. The only difference between both is the vector space for their respective spatial parameters.

It is very hard to model a good approximation for an input interface like that. There are many different ways how to achieve this, but there are two main points, which need to be taken into account:

- time invariance
- spatial invariance

Since the user could perform a gesture very fast or very slow, time invariance of our object model is a very important precondition. For the same reason the space parameters need to be invariant. It should make no difference where the user gestures, as long as our system notices it (e. g. the camera can film it).

The object model needs to combine all the time and space parameters.

4.1. Determine the timing

Gestures are composed of three stages:

- preparation phase
- climax point
- retraction phase

We can identify the preparation phase through the acceleration of the gesturing part in the scene. This is pretty easy measurable with data gloves having an accelerometer built in.

The climax is one single point in time, where most gestures freeze. At that point there is no motion in the scene. Therefore that particular point can be best captured as a frame, taken from a camera.

In the retraction phase, the gesture moves back into its original position. Often, the next gesture quickly follows, so that the one before is not completely finished when the next preparation phase already starts. The old gesture did not reach the home position then. With this knowledge we can find out sequences of gestures.

With respect to the above two parameters, the acceleration and the climax point, we can build a time invariant system by normalizing that time frame.

Most hand gesture systems try to recognize the finger tips of the user. Since these are the most exterior parts, we get a good approximation for the acceleration. They are also good to spot the climax.

4.2. Determine spatial parameters

As described above, spatial parameters fall into two categories, 2D and 3D parameters.

2D parameters result from a 2D object model. A very simple model for simple gestures are *Image Templates*, where we have a reference frame (also called keyframe) every few microseconds. These references can also only be a part of the whole image, which we get by building bounding boxes around the interesting areas. This is useful to avoid unnecessary confusion for the image matching process. For the matching process these Image Templates can be scaled in size, which makes them invariant in space. To achieve an even better result we can transform the image to only show a silhouette or contour of the interesting part in the image. With these *Deformable Templates* we can slightly adjust the contour of the reference frame to achieve a better matching result. So the parameters here could be the image itself. Geometry parameters inside the image, like 2D vectors between some interesting points, could also be used. as well as histograms, or motion parameters computed from previous images. As input devices, we could use touchscreens and pens.

From 3D object models we receive 3D parameters. A 3D model is set up of kinematic chains. When we take a *Skeletal Model* for example, the object is constructed by its bones. These bones have a length and a position in space and can therefore be seen as vectors. These vectors can be compounded to more complex objects. Two bones have a fixed (or semi-fixed) angle between each other. We can also build a model with the corpus of an object, getting a *Volumetric Model*. Here we construct the surface as a grid of vertices and edges. These vertices can also be seen as vectors in space. Since we can scale these models in size, they are space invariant, too. So as resulting parameters we mainly get joints and vectors based on these. The interesting bits are the relation between two adjacent vectors, since we can put some constraints on them, as we will see later. We could create these models using stereo cameras as input devices.

As we can see, 2D models have far less parameters, and thus less degrees of freedom for each of them. 3D models instead, sometimes have hundreds of parameters. Therefore they have a higher computational complexity, but on he other hand they are very cheap on storage usage. Since we save complete images for a 2D model, instead of only vectors for a 3D model, we have an increased demand of storage bandwidth as well.

5. Analysis

Now that we have defined our parameter set for our object model, the challenging part begins.

We need to ask ourselves how to locate the gesture and find out the areas of interest in our image? Then, how do we update the model parameters and which values should they have at the beginning? In this section we will break down these questions and emerging problems thereof.

5.1. Feature detection

For image based models, we need to find the gesture in the picture. Therefore we will *segment* it. The most simple segmentation technique is to mark the interesting items in the scene in front of the camera and then search them in the picture. Another kind of segmentation techniques are *histogram based methods*, where we compute a histogram for each image. Together with other color information or the intensity, we can easily spot the expected objects in the picture. Yet another possible way is to

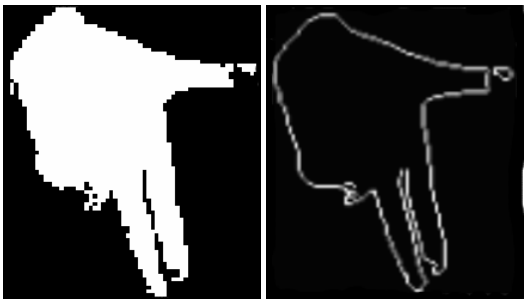

Figure 1: *A silhouette image and a contour image, both reduced from unneeded information (Original source: [6])*

use *edge detection methods*. These embrace that there is a sharp adjustment of intensity at region boundaries.

A special way to deal with the problem of image segmentation is the *model based approach*. Since we mostly know in our systems where to look for the gesture on an image, we can directly cut out that region in a bounding box and analyze it.

There are still many more image segmentation techniques, which are all technically mature. All of them have in common that they are computationally intensive, so that they are hard to implement when there are requirements for realtime processing.

A major problem is the background in our image segments. The detection algorithms have worse recognition rates for complex backgrounds. Especially when the background colors match the same color that we defined as matching parameters, e. g. skin color. For still backgrounds we could take a snapshot of the background and subtract it from each image. For complex backgrounds *Bunch graphs* can be used ([5]). Bunch graphs represent parts of our posture as different graphs, which can be matched to separate foreground and background. This is the same technique used by the police to generate phantom faces.

Image-based models now start to reduce the remaining complexity of information, which is still in the image segments by reducing the color depth to a binary level, leaving us with a contour image or silhouette as shown in figure 5.1.

From stereo-cameras we get two images, which will be segmented and reduced from all unnecessary information. These two images can be used to compute spatial 3D parameters to match a 3D object model.

5.2. Parameter estimation

To reduce the computational complexity of the model in use, we might need to put certain constraints on the parameters. With techniques like the *principal component analysis* or *multidimensional analysis* we correlate each of the parameters with the others to identify and reduce those tuples describing the same characterization. With such a method in place we gain the ability to use a more complicated object model and yet we can compute it in realtime. This gives us the possibility to recognize more fine-grained gestures.

For camera-based 2D parameters we can take keyframes as parameters and try to match these. An even better attempt is to use *motion history images*, where we have a series of images all overlaid with each other, but all in the correct order. Imagine it like taking pictures with a photo camera and exposing the same image all over again. The resulting image has complete information about the whole gesture. This can be used as a parameter in this model and matched against a library template.

For 3D model parameters our set of parameters mostly consists of joints and vectors. The first problem is to give initial values to this parameter set. The easiest way to obtain these is to let the user input them. On the other hand, that means our users need to be familiar with our internal object model. A better way here is to first input the initial values in the lab and then train the model with different people performing our target gestures. After some iterations the parameters will be in between some upper and lower bounds. It could be necessary to try different initial parameter values. After obtaining the most accurate parameter values and their bounds, these can be used for the final product as initial parameter space. The initial parameter set is most crucial for the recognizer. Since the joints are not on a fixed position, we need to find their respective central position for each of them very accurately. If the upper and lower bounds are nearly equal we can use spherical coordinates and get only one additional parameter, the radius, for each joint within which it can move. After we have set up the initial parameters there is an iterative process of updating these throughout the gesture. This is equivalent to computing the inverse kinematics.

The used algorithms in this stage need to cope with some real life difficulties as well. Occlusion of parts of arms and hands for example, so that the gesture is not completely observable to the system. These problems can be mitigated by using more cameras in adequate positions, but cannot be fully alleviated. Because of these issues and to minimize the search operation in the library for each new frame, we need to predict the next parts of the gesture movement. These predictions can be done using *Hidden Markov Models* or *Kalman filters*. Both methods are computational very expensive and thus hard to implement in realtime. On the other hand they are faster than searching the whole library for each captured image. The prediction results can also be used for the segmentation of the next image (cf. section 5.1).

6. Gesture recognition

Our goal now is to use the analyzed parameters to identify one of our pre-defined gestures. These come from a library of training data, which can also be obtained from the field to readjust the system.

The analysis from the previous section gave us an estimation of all parameters and how likely they fit for a given gesture in the library. With the help of *clustering techniques*, like *vector quantization*, we merge our feature parameters with certain criteria to a resulting vector. That resulting vector is compared against stored vectors in a library. It matches the vector with the shortest distance. There are, of course, other clustering techniques, like statistical pattern recognition using *Hidden Markov models* ([4]). Forchhammer et al. extend this idea by introducing *Partially Hidden Markov models* ([7]). Partially Hidden Markov models (PHMMs) are based on ordinary Hidden Markov models (HMMs). These express hidden states of our object model by observable parameters, which results in estimated probability distributions for each hidden state. There is one HMM per gesture. Now PHMMs differ in that the transition probabilities of the hidden states, and the output probabilities, are computed by all previous observations. PHMMs introduce observable "contexts" in the model, which are a weaker idea of states. Under certain circumstances PHMMs are expressible as normal HMMs, but then with a bigger state space. PHMMs are especially useful for modeling image data.

Other *dynamic Bayesian networks* are useful as pattern recognition algorithms, too.

Most of these clustering techniques require a template library against which the observed features can be compared. That library is created by training material of the gestures, which we want to recognize, performed by many different people. So for an optimal set of templates in the library, we need an accurate set of parameters. To compare performance results between different gesture recognition systems, in theory all of these systems should learn from the same template library. In practice each of them should recognize specific gestures, and thus using specific libraries. This makes them hard to compare in general.

As we have found out earlier in section 4.1, we can detect series of gestures. Since there are only certain combinations of gestures possible in most cases, we can not only do a prediction within one gesture, but for the upcoming gestures in general. That greatly reduces our search domain as well. This can be used to implement safety features, so that the system will recognize gesture combinations, which potentially damage the system or result in life threatening operations. All possible series of gestures are stored in the library as a *finite state machine* to save some space ([4]).

In the next step, the gesture recognizer forwards its resulting probabilities for the most likely gestures from the library to the next processing unit. In MMDS this is the multimodal data fusion unit, which takes all probabilities of recognized user input and correlates these against each other. This gives a final global result of recognized and interpreted user input.

7. Conclusion

All in all we can see that gesture recognition as a whole is very expensive in terms of processing power. The performance of such a system greatly depends on the complexity of the gestures, which it needs to recognize. It also depends on how fast the system must be able to identify a performed gesture. This all plays in with the overall number of recognizable gestures.

If we put the focus of our system more on one of the three terms, the others will get worse. We need to find the best balance for each new gesture recognition system. There is no universal system to recognize any given set of gestures ad hoc, yet.

8. References

[1] López-Côzar, R. and Araki, M., "Spoken, Multilingual and Multimodal Dialogue Systems, Development and Assessment", 2005

[2] Pavlovic, V. I. and Scharma, Rajeev and Huang, T. S., "Visual Interpretation of Hand Gestures for Human-Computer Interaction: A Review", IEEE Trans. Pattern Analysis and Machine Intelligence Proc. 19(7): 677–695, 1997

[3] Krumm, J. and Meyers, B. and Brummit, B. and Hale, M. and Shafer, S., "Multi-Camera Multi-Person Tracking for EasyLiving", 3rd IEEE Workshop on Visual Surveillance Proc.: 3–10, 2000

[4] Wu, Ying and Huang, Thomas S., "Vision-Based Gesture Recognition: A Review", International Gesture Workshop Proc.: 103–115, 1999

[5] Triesch, J. and v. d. Malsburg, C., "Robust Classification of Hand Postures against Complex Backgrounds", 2nd Intl. IEEE Conf. on Automatic Face and Gesture Recognition Proc., 1996

[6] Hu, Chao, et al., "Visual Gesture Recognition for Human-Machine Interface of Robot Teleoperation", IEEE Intl. Conference on Intelligent Robots and Systems Proc., 1560–1565, 2003

[7] Forchhammer, S. and Rissanen, J., "Partially Hidden Markov Models", IEEE Conf. on Transactions on Information Theory Proc., 1253–1256, 1996