Recognition of Fingerspelling in Real-time Video Sequence

Aniket Dhar[1] and Philipp Duernay[2]

***Abstract*— TODO**

I. INTRODUCTION

The accuracy of modern computer vision systems enables a wide range of applications for computers to support society, not only in our daily-life activities, but also in health care or even safety-critical situations. Detection of diseases, cruise control and pedestrian detection are only a few examples to name.

In this project we explore another possible application for computer vision: the recognition of sign language. As a final goal one can image a smartphone app which translates sign language in real-time enabling a normal conversation between deaf and hearing people.

As a first start in that direction we start by recognizing the fingerspelling alphabet as shown in Figure 1. We implement and evaluate a computer vision system that can detect the static gestures of this alphabet[1] in a video sequence in real-time.

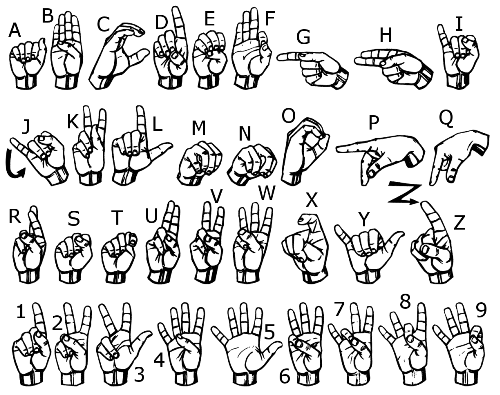

Fig. 1. American Fingerspelling Alphabet

For that we build a classification system and train it on a dataset of static images. As a proof of concept we embed the obtained model in a small application. The application reads a video stream from the webcam, segments the hand of the user and shows the recognized letter.

For computer vision applications one can make use of a lot of tools and libraries that are already implemented. In this project we use *python3* together with *OpenCV* and some other libraries. A full overview is given in section IV

The project has been carried out in context of the Computer Vision course at Delft University of Technology. This report describes the work flow of our project, the applied methods as well as the obtained results.TODO one sentence results

The remaining parts or this document are structured as follows: section II describes our approach and the implemented methods. section III contains an evaluation of the implemented methods and the respective results. section IV concludes the project and gives a short look ahead.

II. APPROACH

The aim of the project is to recognize letters from the sign alphabet in a video sequence. To achieve this we take a classical supervised learning approach: we learn a model from labeled examples off-line, then we apply the trained model on video frames.

We identify three main ingredients that are required for the final application: (1) Object descriptors that represent the pose of the hand unambiguously, (2) a model that we can train on a dataset with labeled descriptors, (3) an application that extracts the hand descriptors from a video sequence and matches it with the trained model.

For all steps we tried different approaches. These are described in further detail in the following sections.

A. Training Data

In total we found two sets that are suitable for our purpose, namely a set from University of Exeter[2] and a set from Thomas Moeslund's[3]. For the rest of the document we refer to the first one as the *ASL-set* and the second one as the *TM-set*.

The TM-set contains between 50-100 black and white images per class for the 24 letters of the alphabet. The subjects wear a black sweater and the pictures are taken in front of a black background. An example can be seen in Figure 2.

The lightning conditions should make the segmentation quite easy, as there is a big contrast between the hand and the background. However, the lightning conditions are also quite artificial and do not resemble a natural use case. Thus it is questionable whether a model trained on that dataset can perform well in a real application.

The ASL-Set contains 2500 coloured images per class for the 24 letters of the alphabet, taken from 5 different subjects. The pictures have been taken from a kinect camera with more or less equal lightning conditions. An example can be seen

[1]Aniket Dhar PUT STUDENTNO
[2]Philipp Duernay - 4622227 - p.durnay@student.tudelft.nl

[1]Letter J and Z involve movement and are not part of this project

[2]`http://empslocal.ex.ac.uk/people/staff/np331/index.php?section=FingerSpellingDataset`

[3]`http://www-prima.inrialpes.fr/FGnet/data/12-MoeslundGesture/database.html`

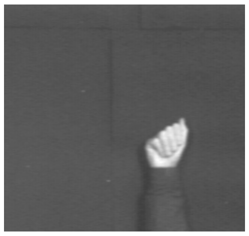
Fig. 2. Example image for the TM-set

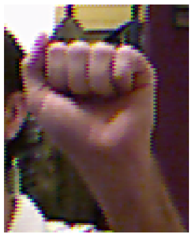
Fig. 3. Example image for the ASL-set

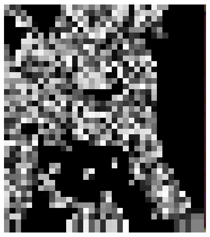
Fig. 4. Likelihood for colour-histogram segmentation on ASL-set

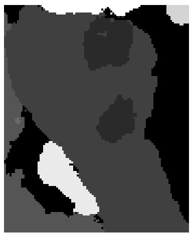
Fig. 5. Depth data int the ASL-set

in Figure 3. Next to the image also depth data is available which can be used for segmentation.

The pictures contain strong shadows and a lot of time the face of the user is also visible on the picture. Although this resembles much more our final use case, it makes colour-based segmentation much harder. The depth data can help in segmentation on the dataset, but we can't rely on it in the final use case, as it is not available for our webcam.

B. Segmentation

For our project we want to separate the hand from the image background. As the shape of the hand can be different in every image we can't use shape-based features for segmentation. Instead we try several texture-based approaches.

Segmentation can also be seen as a two class classification problem. Given a pixel we are looking for the probability that it belongs to the hand or to the background. Based on that probability we can assign it to one of the two. More formally this would be:

$$\textbf{assign } y_1 \textbf{ to x if } p(y_1|x) > p(y_0|x \textbf{ else } y_0 \tag{1}$$

With y_1 as label for the hand and y_0 as label for the background.

C. Colour Threshold

The most straight-forward way is to to use a colour range for segmentation. For example a gray threshold or the a skin colour range. As long as the lightning conditions are good and no other objects with the same color are on the image, it can already work reasonably well.

For this approach we define the likelihood of a pixel being part of the hand:

$$p(y_1|x) = \begin{cases} 1 & \text{if } \gamma_1 < x < \gamma_2 \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

With γ as the threshold(s).

This approach can easily be implemented using conditioned indexing with numpy arrays. All pixels of an image that contain a colour value that is higher than a certain threshold are copied to a new image.

We evaluate a gray value threshold on the TM-set, where everything else but the hand is quite dark. An example can be seen in Equation II-C.2, where we cropped the hand of the part with the highest probability. Here we already get satisfying results.

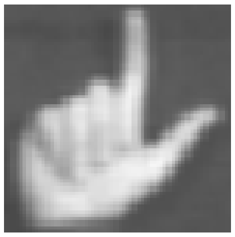
Fig. 6. Colour-threshold segmentation on TM-set

On the ASL-set we try skin-color-based segmentation. We set a range of TODO in HSV colour space to be part of the hand and everything else to be background.

TODO put example

1) Colour Histogram: A colour histogram consists of a histogram for every colour channel. The histogram can be calculated for a range of pixels and therefore capture a colour distribution over several pixels. Thus it is more reliable than a simple colour threshold.

In our approach we first calculate an average histogram for hand pixels, then we define the class conditional probability as the difference to that average histogram. Inspired by the last lab session we define the likelihood as:

$$p(y_1|x) = e^{-\frac{d(\bar{\phi},\phi(x))}{\sigma}} \tag{3}$$

Where ϕ denotes the histogram function, **d** denotes the KL divergence and σ is a scaling factor.

Also this approach can be implemented with numpy arrays. The calculation of the KL divergence we integrate from them *MATLAB* script of the last lab session to python.

On the ASL-set the hand is in almost all cases in the center of the picture. Thus we can obtain the average histogram taking the average histogram for a 6-by-6 window in the center of the picture across the whole dataset. Then we can apply Equation 3 with a 6-by-6 sliding window over every image. An example can be seen in Figure 4, where the likelihood is represented in intensity values.

The example shows the likelihood for Figure 3 one can easily see that the results are not satisfactory. Although big parts of the background are seperated properly, the method fails on image parts where the colour is similar to skin colour. This can be seen in at the face of the subject, as well as the yellow background. Another problem is the big differences even on the hand. All together we don't consider this approach as promising enough.

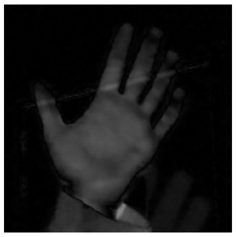

Fig. 7. Dissimilarity image for background subtraction

2) Background Subtraction: Background subtraction [TODO REF] is a way to separate fore- and background in videos. A background frame is calculated and subtracted from the current frame. Thus only objects that changed their appearance remain in the dissimilarity image.

As our application will only run on a stationary webcam, we can assume that the hand is the only moving object and background subtraction is applicable. However, the approach can't be applied during training, where we only have static images available.

For the calculation of the background frame we take an average across several frames. We define a likelihood model for background subtraction as follows:

$$p(y_1|x) = e^{-\frac{||\bar{x}-x||)}{\sigma}} \tag{4}$$

Where $\bar{x}$ is the background, and σ a scaling factor.

We implement this using the OpenCV API. Next some bluring we use `accumulatedSum()` to calculate the average image and `absdiff()` to subtract the background.

An example for a described dissimilarity image can be seen in **??**. The method looks promising as the hand is clearly separable. However, if we move the hand slowly, or the lightning conditions are bad big parts of the hand are also taken as background.

3) Depth Segmentation: For the ASL-set depth data is available. An example can be seen in Figure 5, where the depth data for Figure 3 is displayed in grayscale. The low resolution of the data is apperant as only about 5 different levels are visible. Although it enables us to separate the background partly, for most pictures it is not enough to segment the hand entirely. However, we can use the method in combination with other segmentation methods.

We measure the depth at the center of the image and take it as baseline. Then we calculate the likelihood as follows:

$$p(y_1|x) = e^{-\frac{||x_{center}-x||)}{\sigma}} \tag{5}$$

Where x_{center} is depth value at the center of the image, and σ a scaling factor.

Also here the implementation is quite straight forward. The image is stored in a similar way as a grayscale image. Thus we can subtract the depth value of the center from the whole image and obtain the distance matrix.

4) Nearest Neighbour Classifier: Instead of just measuring values on the image we can also train a classifier on labeled examples. As we want to do the segmentation for the video in real-time, we need a classifier that can be trained quickly. This makes the knn-classifier a good choice.

Labeled examples are unfortunately not available in the given data set, so we use a hands-on method to obtain those. Similar to the histogram method we sample a window of 6-by-6 pixels in the picture center as objects with label y_1. Additionally we can use depth segmentation or background subtraction to obtain background samples with label y_0.

Once the classifier is trained it can be used to label the whole image. As we did not implement the classifier ourselves we don't go into detail about its functionality here and refer to standard pattern recognition literature.

For the implementation we use the 3-nearest-neighbour implementation from *sklearn*. The library allows training a model and obtaining predictions.

5) Markov Random Field: A Markov Random Field (MRF) [TODO REF] enables the incorporation of spatial conditions with label probabilities. We use it to combine these spatial conditions with several of the methods described before.

We apply the depth segmentation and background subtraction model to get soft labels for back and foreground. Additionally we train the knn-classifier as described and obtain another set of soft labels for the image. Then we can define the combined likelihood as a weighted sum of the score of the nearest neighbour classifier and of the background calculation. This can be formulated as follows:

$$p(y_1|x) = \alpha * s_{knn}(_1|x) + \beta * s_{backgr}(x_1|x) \tag{6}$$

With α and β as weights and s_{knn} and s_backgr as the respective soft labels.

For defining spatial structures we take the image gradient. We want the edge value to drop when there was an edge in the initial image. Therefore we assign edges on y and x axis with the respective gradients I_y, I_x multiplied by -1.

In total this gives us the following Markov Random Field:

This way we incorporate information of spatial structures like edges and corners, colour information as well as movement/depth information in our model. The final labelling we obtain with the maxflow graph cut algorithm.

For the implementation we use the `maxflow` package for python. The package allows us to define a graph and to assign weights according to **??**. After that we can execute the maxflow algorithm and obtain the labeled image. The first result still contains some noise. Therefore we apply several bluring/eroding and closing operations, using OpenCV, to obtain a cleaner result.

An example for the method can be seen in Equation II-C.5 and Figure II-C.5. The results look quite promising and in this first evaluation we consider it as our best method so far.

D. Representation

Once the hand is properly segmented from the background we need to define descriptors for the hand shape that enable an accurate recognition. As we are interested in the shape of

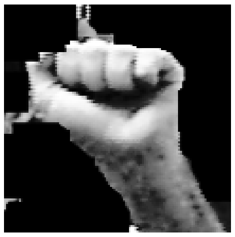

Fig. 8. Segmentation with Markov Random Field on ASL-set

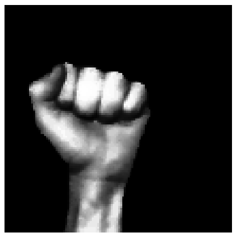

Fig. 9. Video Segmentation with Markov Random Field

the hand, shape based descriptors seem to be a reasonable approach at first. However, also these need to be obtained from the image. In several letters it is particular necessary to also capture the shape of the fingers inside the hand and not only the overall shape. In the ASL-set and on the video there are a lot of shadows in the pictures which make this not an easy task.

Therefore we mainly worked on region-based descriptors. The approaches are described in the following.

1) Grayscale-Pixel: This can be seen as one of the most straight-forward representation. We convert the segmented image to a grayscale image, vectorize it and obtain the feature vector.

The drawback of the method is that less important features not particularly filtered, which results in a huge feature vector. These are generally harder to classify (Curse of dimensionality). Additionally the representation is not rotation/scale invariant.

2) Edge-Pixel: A lot of information about the shape is contained in the edges. If we are able to detect the important edges, we can take the contours of the hand as a shape representation. Therefore we apply the Canny edge detection algorithm [TODO REF] to obtain a binary image with the most important edges. These binary images are vectorized and used as feature vectors.

This method still leads to a high feature vector and is similar to the grayscale pixel approach not rotation/scale invariant.

3) Histogram of Gradients: HoG-features [TODO REF] are commonly used for object detection. The image gradient is calculated for the whole image. Then in a fixed window size a histogram is formed containing the distribution of image gradients.

These features incorporate change in intensity value and should therefore be more informative than the simple Edge-Pixel representation. Especially because the slight changes in shape of the fingers on the hand can be detected.

However, the same drawback as described in the previous paragraphs holds. The representation is not rotation/scale invariant.

In contrast to the first two methods we need our own implementation for HoG-features. We use OpenCV to obtain the image gradients, subsequently we calcualate the angular and magnitude using numpy functions.

4) Bag of HoGs: To compensate for the rotation invariance we apply a "Bag-of-" method. Here we apply a KMeans clustering to obtain the most common histograms of the ASL-dataset. Then we represent Each object by the distance to this codebook. Inspired by MILES [TODO REF] we calculate the distance as follows:

$$\phi(x) = min\{e^{||x - x_{codebook}||}\} \tag{7}$$

Thus the distance of a image to a code word is defined by the distance of its closest instance. The measure should give a high similarity if the HoGs of two objects are similar even if they are at different places in the image.

This method basically builds upon the previous method. After having obtained the HoG-features for all images we use KMeans clustering from the *sklearn* library to get the codebook.

5) Principal Component Analysis: A general way to deal with high dimensionality or less informative features is principal component analysis [TODO REF]. Here we transform the feature space by keeping the maximal variance. We evaluate PCA on top of the previously described methods. The principal components are chosen in a way that 90% of the variance is kept.

Also here we use the *sklearn* library. It provides as with functions to scale the data, as well as to apply PCA.

E. Classification

After having segmented the image and transformed it to a representation we can train a classifier. The focus of the work was on the preceeding steps, which is why we only evaluate several standard classifiers. These are:

- Nearest Neighbour, because it can be trained quickly
- Linear Support Vector Machine, because it can cope with large feature dimensions
- Neural Network, because it generally performs well

In the video the user will not change the shown letter every couple of milliseconds. Thus we have several frames per sample available and can exploit that for classification.

We take a sequence of 5 frames per sample and classify each. The final prediction is based on a majority vote. Additionally we can set a "confidence" threshold to avoid misclassification. If not at least 75 % of the predictions are the same, we reject the classification and a "Letter not recognized" is displayed on the screen.

The majority voting we implement ourselves. The classifiers are taken from *sklearn* library.

III. Evaluation

We select several combinations among the methods described in the previous chapters for evaluation. In a first step we evaluate the methods on the two datasets in cross-evaluation. The methods with the best results, we train on the whole dataset and evaluate in the video.

A. *Performance on Datasets*

We set up a 6-folded cross-evaluation for all methods where we measure the mean performance in accuracy and its standard derivation. An overview of the evaluated methods and their results is given in Table II

TABLE I
EVALUATED METHODS

Dataset	Segmentation	Representation	Classifier	Accuracy
TM-set	Gray-thresh	Edge-Pixel	SVM	
TM-set	Gray-thresh	HoG	SVM	
ASL-set	MRF	HoG	SVM	
ASL-set	MRF	Bag of HoGs	SVM	

TODO blabla results
TODO put confusion matrix

B. *Performance on Video*

For the video evaluation two subjects try every letter and .. TODO

TABLE II
EVALUATED METHODS

Trainingset	Segmentation	Representation	Classifier	Accuracy
TM-set	MRF	HoG	SVM	
ASL-set	MRF	HoG	SVM	

TODO bla bla
TODO put confusion matrix for best one

IV. CONCLUSIONS

TODO

APPENDIX

A. *Implementation*

TODO used libraries TODO what did we implement TODO put link to code

TABLE III
CONFUSION MATRIX

	A	B	C	D	E	F	G	H	I	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y
A																								
B																								
C																								
D																								
E																								
F																								
G																								
H																								
I																								
K																								
L																								
M																								
N																								
O																								
P																								
Q																								
R																								
S																								
T																								
U																								
V																								
W																								
X																								
Y																								