Sign Language Recognition Using Kinect

Conference Paper · April 2012

DOI: 10.1007/978-3-642-29347-4_46

3 authors, including:

Simon Lang
University of Bonn
1 PUBLICATION **38** CITATIONS
SEE PROFILE

Marco Block-Berlitz
Hochschule für Technik und Wirtschaft Dresden
37 PUBLICATIONS **86** CITATIONS
SEE PROFILE

Sign Language Recognition Using Kinect

Simon Lang, Marco Block, and Raúl Rojas

Freie Universität Berlin
Institut für Informatik und Mathematik
Takustr. 9, 14195 Berlin, Germany
{slang,block,rojas}@inf.fu-berlin.de

Abstract. An open source framework for general gesture recognition is presented and tested with isolated signs of sign language. Other than common systems for sign language recognition, this framework makes use of Kinect, a depth camera which makes real-time 3D-reconstruction easily applicable. Recognition is done using hidden Markov models with a continuous observation density. The framework also offers an easy way of initializing and training new gestures or signs by performing them several times in front of the camera. First results with a recognition rate of 97% show that depth cameras are well-suited for sign language recognition.

1 Motivation and Introduction

Using gestures as a natural communication interface between human beings and machines becomes more and more important. This involves controlling computers, as well as processing and translating sign language.

When Microsoft released Kinect in November 2010, it was mainly targeted at owners of a Microsoft Xbox 360 console, being advertised as a controller-free gaming experience. The device itself features an RGB camera, a depth sensor and a multiarray microphone, and is capable of tracking users' body movement [9,10]. The interest in the device has been high among developers, and thus, shortly after its release an unofficial open source driver was introduced, followed by many Kinect-based projects and technical demos. Even though Microsoft stated that "Kinect that is shipping [2010's] holiday will not support sign language", several demos show how it technically is capable of recognizing signs [11,12,13].

In sign languages, manual features are generally used along with facial expressions and different body postures to express words and grammatical features. The manual components can be split into four parameters: handshape, palm orientation, location, and movement. There are similar signs that differ in one of these components only, and thus without considering context, signs can only be recognized precisely when all of these components are known. Nevertheless, a great number of signs can be distinguished by only considering hand location and movement [6].

After the related work part in section 2, we present Dragonfly, an open source C++ framework for general gesture recognition that can be used to recognize signs of sign language, utilizing the two above-mentioned manual components.

This is achieved by using hidden Markov models that allow training and recognition of isolated signs. In section 4, the framework is tested in several experiments, and an evaluation shows how well it performs when using optimal parameters. A conclusion in section 5 summarizes the achievements of this work and what future work may follow in order to improve it for better sign language recognition.

2 Related Work

This section summarizes the basics of hidden Markov models as well as their application on sign language recognition. Alternate methods of feature extraction are also presented.

2.1 Hidden Markov Models

Hidden Markov models (HMMs) are a type of stochastic model related to finite state machines. An HMM features a number N of states $S_1, S_2, \cdots, S_N$, being in exactly one of theses states at any time $t = 1, 2, 3, \cdots$, where the state at time t is referred to as q_t.

The initial probability distribution π describes the probability of starting in a specific state. Every state S_i features a probability of transitioning to a state S_j, stored in the transition probability matrix $A_{N\times N}$. A set of output probability distributions $B = \{b_j(k)\}$ describes the probability of observing the k^{th} out of M observation symbols in state j. Observation sequences are denoted as $O = O_1 O_2 \dots O_T$, where T is the number of observations in the sequence and each O_t represents one such observation.

If M is infinite (e. g. when observations are real numbers), the HMM features a continuous observation density. Each state then includes a mean vector μ and a covariance matrix Σ for use with a logarithmically concave function, e. g. a D-dimensional multivariate Gaussian distribution $\mathcal{N}$,

$$\mathcal{N}(O_t; \mu, \Sigma) = \frac{1}{(2\pi)^{\frac{D}{2}} |\Sigma|^{\frac{1}{2}}} \cdot \exp\left(-\frac{1}{2}(O_t - \mu)^{\mathrm{T}} \Sigma^{-1} (O_t - \mu)\right).$$

Given the triple $\lambda = (A, B, \pi)$ as a compact notation for HMMs, the three basic problems that come along can be summarized as follows [7,8,3]:

1. Given an observation sequence O, determine the probability of O being generated by λ, i. e. efficiently calculate $P(O|\lambda)$. This is done using a scaled version of the forward-backward procedure.
2. Determine the state sequence $Q = q_1 q_2 \dots q_T$ that is most likely to be traversed, given an observation sequence O and a model λ. The scaled Viterbi algorithm takes care of this calculation.
3. Determine how to adjust the parameters of λ in order to maximize $P(O|\lambda)$. This is achieved by the Baum-Welch algorithm, modified to accept multiple observation sequences at once.

A method of initializating HMMs is proposed by Kelly et al. [1], where the initial parameters are calculated automatically with an optimal number of states.

2.2 Recognizing Sign Language

Recognizing sign language involves two major processes, namely extracting features and interpreting them. While the former is usually done using a 2D camera [2,5] and detecting the positions of hands and head, Vogler and Metaxas [4] use a set of three orthogonally placed 2D cameras to extract 3D data of the signers body parts. The results show that this method is more accurate than using 2D data.

In order to conveniently recognize signs and to handle the statistical variations when performing them, both intra- and interpersonal, HMMs are introduced and each sign is represented by a separate HMM. An observation sequence can be seen as a performance of one such sign, and each single observation represents a vector of body part information, e. g. a hand's position, movement speed, and the distance between both hands.

When a sign is performed, the probability of that performance given each HMM is calculated. The HMM with the highest probability is most likely to have produced that sign. This information is essential for actually building a sign language recognition framework.

3 Dragonfly Framework

The framework presented in this work is called Dragonfly (**Dra**w **g**estures **on** the **fly**) and is capable of learning and recognizing gestures and signs. It is written in C++ and makes use of the free cross-platform Kinect driver OpenNI released by PrimeSense, including NITE skeletal tracking which automatically extracts users' body parts such as their hands and elbows.

The main classes to be included in other software are called `DepthCamera` and `Dragonfly`. The former acts as an interface to OpenNI and can easily be replaced to use a different Kinect driver with skeletal tracking. The latter is the actual interface to the framework which processes the skeleton data for gesture recognition.

Dragonfly features an own implementation of continuous density HMMs, offering automatic initialization, Baum-Welch re-estimation with multiple observation sequences, and serialization, among others. For vector and matrix calculations, the Vision Numerics Libraries are used. Boost provides several other helpful features such as an implementation of the observer pattern.

3.1 Gesture Recognition

Observations are recorded for every user separately, and consist of an N-dimensional feature vector. This can be data such as velocity or absolute position of each hand, or distance between hands.

By default, observations are recorded when the dominant (e. g. right) hand moves above a given threshold, such as the torso's y-position. Each of these observations is saved in a matrix which represents the entire observation sequence. Several of these matrices are stored in a list that can be seen as a set of observation sequences.

Observation recording can be turned on and off for each user separately. Every time the user's hand moves below the threshold, the probability of the newly recorded sequence given each existing HMM is calculated, and results are compared. This is done in order to determine the model that best matches the sequence.

To make use of this information, a system for callback functions has been implemented. Since HMMs must have distinct names, these can be uniquely associated with a signal using a hashmap. A signal can be linked to and unlinked from an HMM by calling appropriate methods in `Dragonfly`.

3.2 Learning New Gestures

Creating new gestures and training them is done successively in one method, by providing a maximum number of states, a set of observation sequences – used for initialization and re-estimating – and a set of negative test sequences that do not contain the actual gesture to be trained. An example of one out of several similar training sequences is shown in figure 1.

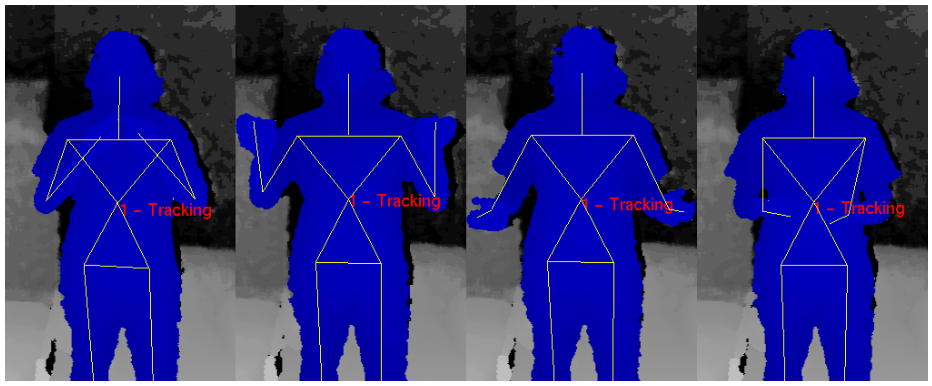

Fig. 1. Example training sequence of the sign PAKET (German for packet)

Cross validation splits observation sequences into sequences actually used for training (two third), and positive test sequences the model should recognize correctly without having them used for training (one third). Additionally, a set of existing HMMs that represent different gestures can be provided.

The algorithm then initializes and trains several HMMs using the training sequences, and determines the optimal HMM given the rest of the data, according to an optimality criterion that must be defined as well.

In detail, the algorithm works as follows:

1. Split provided observation sequences into training and positive test sequences. Two third are used for training and one third for positive testing.
2. Initialize with $N = 1$ states. Set the best HMM to `NULL` and the best value for each optimality criterion to the worst possible value.
3. Create $Q = 5$ HMMs, each with N states, from the given set of training sequences using automatic initialization, which works as follows:
   - Create a point cloud with every single observation of every sequence as a point.
   - Perform k-means clustering on the point cloud, initial prototypes are chosen randomly among its points. Empty clusters are avoided by determining a new prototype from the biggest cluster.
   - Sort the resulting clusters and assign each of them to a separate state.
   - Calculate the transition probability distribution A using data from all observation sequences, where

     $$a_{ij} = \frac{\#\text{transitions from } S_i \text{ to } S_j}{\#\text{transitions from } S_i} \qquad , 0 \leqslant i, j < N.$$

   - Compute the initial probability distribution π, where

     $$\pi_i = \frac{\#\text{observation sequences starting in } S_i}{\#\text{observation sequences}} \qquad , 0 \leqslant i < N.$$

   - Set each state's μ to the mean vector of the corresponding cluster.
   - Determine Σ for each state by calculating the covariance matrix for each corresponding cluster.

   Due to the randomness in k-means clustering, results may vary. Hence, several HMMs are created using the same algorithm.
4. Re-estimate these HMMs by the Baum-Welch algorithm, using the same training sequences as input.
5. Set $q = 1$.
6. Among the Q created HMMs, choose the one at position q.
7. If at least half of the observation sequences could not be processed due to underflow, discard this HMM and go to step 10.
8. Determine the values for all optimality criteria given the newly created model and all provided data, such as positive and negative test sequences, and HMMs of other gestures.
9. Update the best value for each optimality criterion. If this HMM is better than the stored best HMM according to the chosen criterion, define it as the new best HMM.
10. Increment q. If $q \leqslant Q$, go to step 6.
11. Increment N. If $N \leqslant S$ (where S is the maximum number of states), go to step 3.
12. If any of the three split combinations is left, split observation sequences accordingly and go to step 2.

13. Return the best HMM (which is `NULL` in case the procedure failed to create any HMM at all).

This procedure guarantees to deliver a re-estimated HMM that best matches the given data for the chosen criterion, depending on how well k-means clustering performs. Possible optimality criteria are σ_r (recognition rate), σ_v (variance), and σ_{np} (lowest negative above worst positive rate).

The first criterion σ_r uses positive test sequences only and tests them with the newly created HMM and all other HMMs. Negative test sequences are neglected. Each positive test sequence is tested with all HMMs and the number of correct results is saved and then summed up. A test result is correct when the sequence given the new HMM has a higher probability than the sequence given any other HMM. The summed number is divided by the total number of tests, and the resulting recognition rate is to be maximized.

The second criterion σ_v calculates the average logarithmic probability of all negative test sequences given the new model, and subtracts it from the average logarithmic probability of all positive test sequences given that model. Gestures of the new model can be distinguished from other gesture more clearly the higher this value is.

Determining σ_{np} is done by saving the lowest probability of any positive test sequence given the new model – i.e. saving the worst positive test sequence. Then, probabilities of all negative test sequences are calculated for the model. The number of negative test sequences with a probability higher than that of the worst positive test sequence, is divided by the total number of negative test sequences. The lower this value gets, the better the success rate of the new HMM is.

For equal values of the third criterion, the second criterion is used to determine which HMM is better. HMMs can be saved to and loaded from a file at any time during recognition.

4 Experiments and Results

First experiments were made with a vocabulary of 25 signs of German Sign Language that were trained with the help of fluent speakers. Best results could be achieved with a nine-dimensional feature vector, composed of $(RH_x, RH_x, RH_z, LH_x, LH_y, LH_z, RH_v, LH_v, RE_x)$, where RH and LH correspond to the right and left hand relative to the neck, respectively, v means velocity, and RE corresponds to the right elbow.

Every sign was tested 40 to 60 times, results show an overall recognition rate of $97,0\%$. Detailed results are shown in figure 2, where the recognition rate of each sign is shown next to a boxplot.

Each boxplot shows the distance between the actually performed sign and the best recognized sign that is unequal. This illustrates for positive values how well the recognized sign could be distinguished from others, and for negative values how much other signs were preferred. Since the logarithmic probabilities

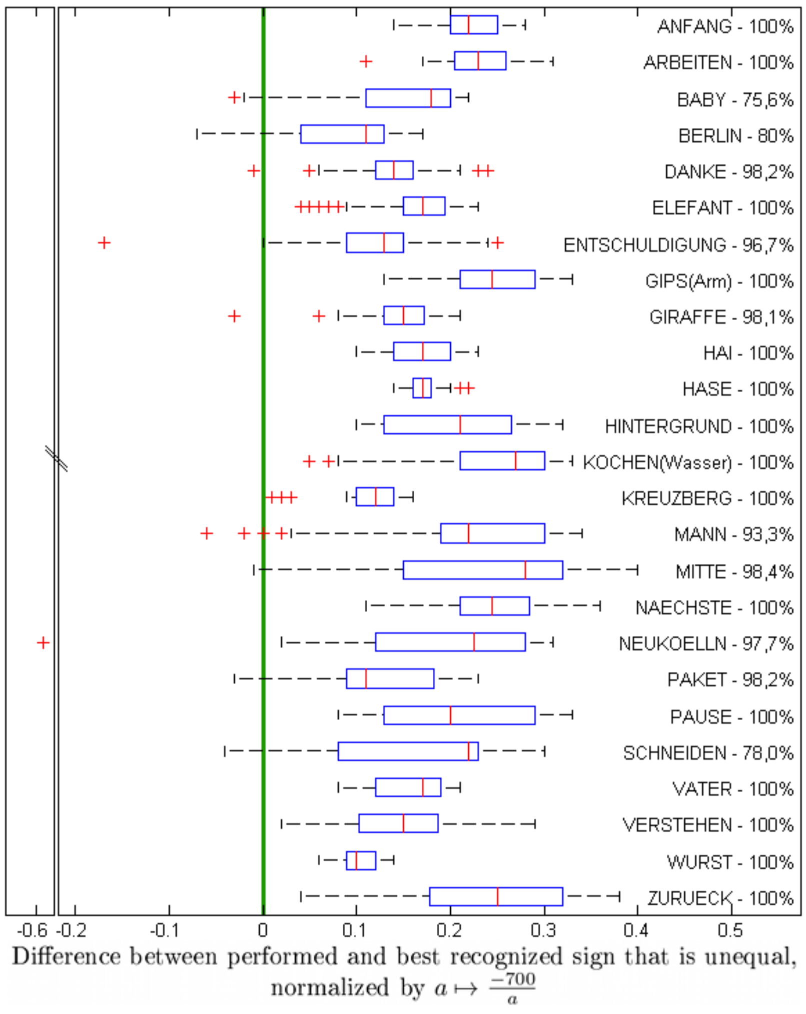

Fig. 2. Test results of the first experiment, the signs trained are ANFANG (beginning), ARBEITEN (to work), BABY, BERLIN, DANKE (thank you), ELEFANT (elephant), ENTSCHULDIGUNG (sorry), GIPS (gypsum), GIRAFFE, HAI (shark), HASE (rabbit), HINTERGRUND (background), KOCHEN (to boil), KREUZBERG (a district in Berlin), MANN (man), MITTE (a district in Berlin), NAECHSTE (next), NEUKOELLN (a district in Berlin), PAKET (packet), PAUSE, SCHNEIDEN (to cut), VATER (father), VERSTEHEN (to understand), WURST (sausage), and ZURUECK (back)

are negative, values have been normalized by $a \mapsto \frac{-700}{a}$ (higher values are better). The sign NEUKOELLN (a district in Berlin), for example, features a high recognition rate and can usually be distinguished well from other signs, however one performance was not recognized correctly and is far off the sign that was recognized instead.

Signs that mainly contain movement towards the camera, such as DANKE (German for “thank you”), have a significantly worse recognition rate with a feature vector that does not contain the hands’ z-positions. The right elbow’s x-position (if the user is right-handed) helps distinguish other signs where the hand is near the face and the arm is either held away from or close to the body.

Since the depth of each body part is known, the recognition rate does not change when standing closer to the camera or further away from it, as long a minimum distance is kept in order for skeletal tracking to work. Especially the use of z-values shows an advantage of depth cameras over ordinary RGB cameras, where extracting depth information is generally harder and less reliable.

On a 2 GHz dual-core machine with 4 GB memory, the probabilities of a sequence given all 25 signs were calculated in less than 300 milliseconds.

5 Conclusion and Future Work

Kinect and other depth cameras offer 3D data without a complicated camera setup and efficiently extract the users’ body parts, allowing for easier recognition of not just hands and head, but also other parts such as elbows that can further help distinguish similar signs. Another advantage is the independency of lighting conditions due to the use of infrared light, however thus, the cameras are limited to in-door use.

The presented framework offers recognition and learning of isolated signs, using NITE skeletal tracking and an own HMM implementation. This implementation includes a new way of initialization and several optimality criteria for HMM comparison. First experiments were made with a vocabulary of 25 signs of German Sign Language, and show a high recognition rate of $97,0\%$ when using depth-camera-specific features. Future experiments will show how well the presented methods perform when using a larger vocabulary of more than 100 signs.

Accurately recognizing sign language, however, not only involves tracking hands. There are signs that only differ in mouthshape or handshape and are similar otherwise. Facial expression, body posture and head movement are often used to express grammatical features.

When detection of these essential components is supported by the backend, Dragonfly can be extended to support continuous sign language recognition. This also involves detecting a sign’s start and end position, as well as movement epenthesis as described by Kelly et al. [1] in order to distinguish hand movement within a sign from movement between two signs.

In conclusion, this work shows that depth cameras are well-suited for sign language recognition. The approach is worth further consideration and features its own advantages, while leaving room for improvement of both the underlying technology as well as the framework itself.

The source code of Dragonfly is available under the terms of the GNU Lesser General Public License, version 3, at *https://bitbucket.org/Slang/dragonfly/*.

References

1. Kelly, D., McDonald, J., Markham, C.: Recognizing Spatiotemporal Gestures and Movement Epenthesis in Sign Language. In: 13th International Machine Vision and Image Processing Conference (IMVIP '09). IEEE Computer Society Washington, DC, USA (2009)
2. Dreuw, P., Rybach, D., Deselaers, T., Zahedi, M., Ney, H.: Speech Recognition Techniques for a Sign Language Recognition System. In: INTERSPEECH 2007, 8th Annual Conference of the International Speech Communication Association (ISCA 2007), pp. 2513-2516 (2007)
3. Li, X., Parizeau, M., Plamondon, R.: Training Hidden Markov Models with Multiple Observations – A combinatorial Method. In: IEEE Transactions on PAMI, Vol. PAMI-22, No. 4, pp. 371-377 (2000)
4. Vogler, C., Metaxas, D.: ASL Recognition Based on a Coupling Between HMMs and 3D Motion Analysis. In: Proceedings of the Sixth International Conference on Computer Vision, ISBN: 978-8-17319-221-0, pp. 363-369. Narosa Publishing House (1998)
5. Starner, T., Pentland, A.: Real-Time American Sign Language Recognition from Video Using Hidden Markov Models. In: ISCV '95 Proceedings of the International Symposium on Computer Vision, ISBN: 978-0-81867-190-6, pp. 265-270. IEEE Publications, U. S. (1995)
6. Boyes Braem, P.: Einführung in die Gebärdensprache und ihre Erforschung. ISBN: 978-3-92773-110-3, 1st edition, SIGNUM-Verlag. In: Internationale Arbeiten zur Gebärdensprache und Kommunikation Gehörloser, No. 11 (1990)
7. Rabiner, L. R.: A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. In: Proceedings of the IEEE, Vol. 77, No. 2, pp. 257-286 (1989)
8. Rahimi, A.: An Erratum for "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition", website of Ali Rahimi at MIT Media Laboratory, `http://xenia.media.mit.edu/~rahimi/rabiner/rabiner-errata/rabiner-errata.html`
9. Official Microsoft Xbox website, introduction of Kinect, `http://www.xbox.com/en-US/kinect`
10. Countdown to Kinect: 17 Controller-Free Games Launch in November, Microsoft News Center, `https://www.microsoft.com/presspass/press/2010/oct10/10-18mskinectuspr.mspx`
11. Kinect Downgraded To Save Money, Can't Read Sign Language, News at Kotaku, `http://kotaku.com/5609840/kinect-dumbed-down-to-save-money-cant-read-sign-language`
12. CopyCat and Kinect, overview of the CopyCat Kinect demo on the website of the Center for Accessible Technology in Sign (CATS), `http://cats.gatech.edu/content/copycat-and-kinect`
13. Integrating Speech and Hearing Challenge Individuals, YouTube channel of Dr. Natheer Khasawneh, `http://www.youtube.com/user/knatheer#p/a/u/1/vVL398dUU5Q`