Sign Language Translation Using Kinect And Dynamic Time Warping

Author: Jinhua Xu

Abstract

This project explores the capability of a simple gesture recognition pipeline on a mobile device using Microsoft Kinect. The results show that given a limited number of gestures (sign language vocabulary of size 5) and a few algorithmic optimizations (dynamic time warping with locality constraint, Sakoe-Chiba band), a real-time continuous gesture recognition system can be implemented on a mobile device with reasonable accuracy.

1 Introduction

Sign language translation is an important area of research that could help bridge the communication gap between audio speakers and hearing impaired people. It is the most natural and fluent way to communicate compared to writing or typing, which can hinder conversation. A sign language translator must be able to recognize continuous sign gestures in real-time, and this is made much more accessible if the procedure can be carried out on a mobile device. There has been little work done in supporting gesture recognition on a mobile platform.

This project approached the problem using a Microsoft Kinect to capture object depth data, and from that calculate the skeleton joint data. Next, the Dynamic Time Warping algorithm is used to calculate the optimal matching gesture from a database of gesture sequences. The dataset used consists of five gestures in the American Sign Language vocabulary, namely “I”, “you”, “love”, “no”, and “please”.

2.1 Previous Work

In the area of gesture recognition, the most widely used algorithm studied is Dynamic Time Warping [1, 2], first used by the speech recognition community. This will be the primary approach of this project. Other methods include Hidden Markov Models [3], as well as Neural Networks [4]. Comparatively, DTW performs more accurately on a small vocabulary of gesture sequences, however it is limited by its computational demands as a larger vocabulary is used. Meanwhile, HMM and NN models require a more involved training phase, but the recognition pipeline has lighter computational demands.

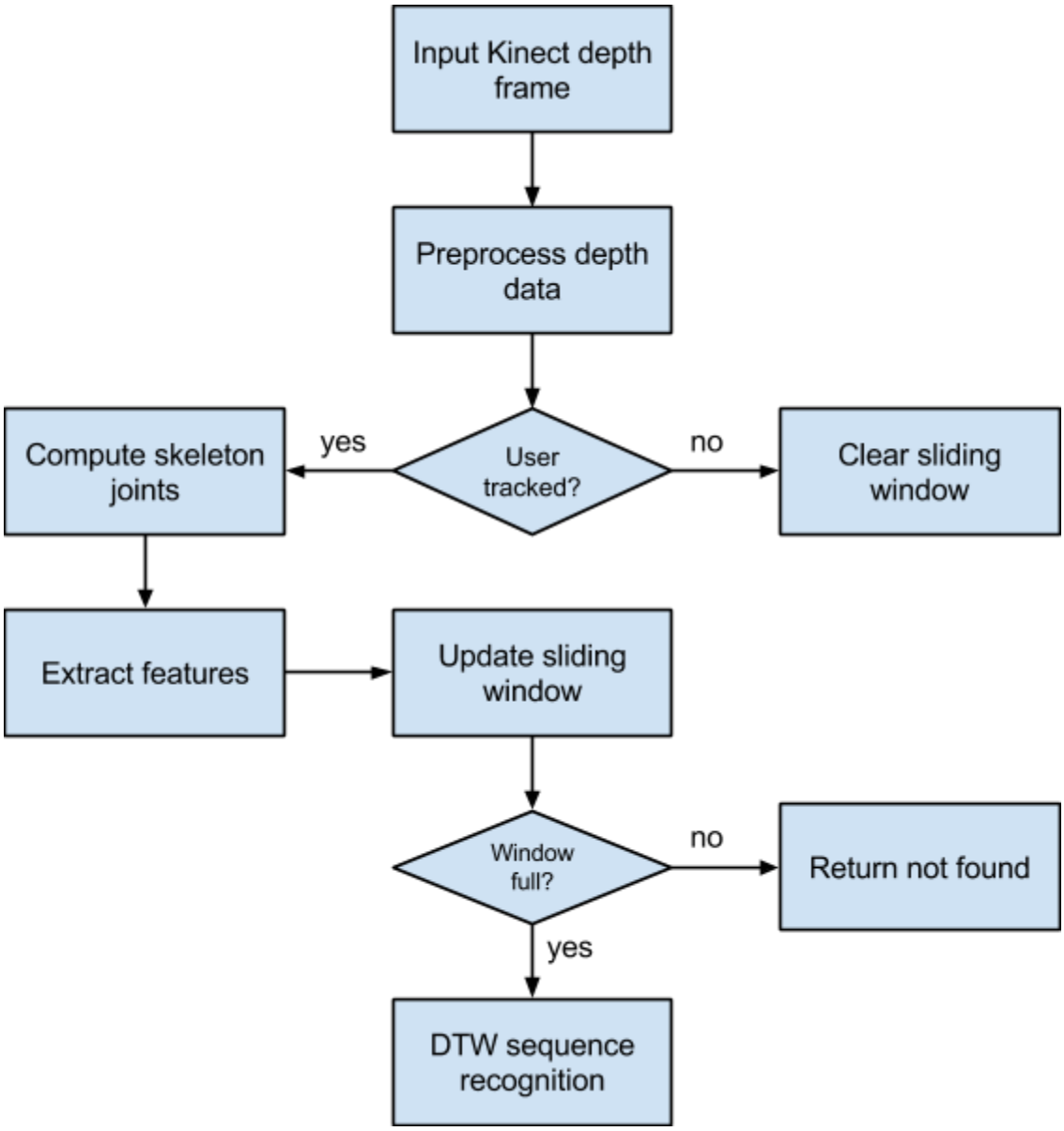
2.2 Key Contributions

The key contribution of this project is to apply previous gesture recognition techniques and make it feasible under the resource constraints of a mobile device. The author recognizes that

a Microsoft Kinect needs to be plugged into a power outlet and is rather clumsy to mobilize, but the focus of this paper is on the implementation and feasibility of the underlying computation, regardless of the sensor. Furthermore, this technology can be replaced by something inherently mobile, such as the Structure sensor [5].

3.1 Summary of the Technical Solution

The project uses an Android implementation of libfreenect and OpenNI for the depth sensor input processing and skeleton joint computation [6], and uses OpenCV for feature extraction and implementation of a locality-constrained version of Dynamic Time Warping. Recognition is done using a fixed-size sliding window of the last N features. The following block diagram summarizes our approach.



3.2 Preprocessing Depth Sensor Data

The OpenNI library provides skeleton joint tracking utility, similar in function to the official Microsoft Kinect API. It achieves this by first obtaining the depth data from the Kinect sensor, then map the depth data to the corresponding skeleton joints via a randomized decision forest trained from over a million training samples [7]. The returned skeleton joints include the centroid locations of the head, neck, shoulders, elbows, hands, torso, hips, knees, and feet in 3D world coordinates. For compatibility on the mobile device, this project used an Android implementation of the OpenNI library. The resulting mobile skeleton joint tracking performs at 30 fps for a 640x480 image.

3.3 Feature Extraction

After skeleton joint coordinates are computed, they are extracted into a feature vector $\vec{p}$. Since sign language usually only involves upper body movements, only the head, neck, shoulders, elbows, and hands coordinates are retained, and all other joint data are discarded. Formally, the considered set of coordinates is $S \in \Re^3$ where $|S| = 8$ .Then, a centroid 3D coordinate, $\vec{C} = \frac{\vec{shoulder_l} + \vec{shoulder_r}}{2}$, is computed and all remaining coordinates are normalized by subtracting the centroid: $\forall \vec{v} \in S,\ \vec{v'} = \vec{v} - \vec{C},\ S' = \{\vec{v'}\}$. This allows the gesture recognition to be translation-invariant. Then, the distance D between the shoulders are calculated, and all coordinates are further normalized by dividing by the shoulder distance: $\forall \vec{v'} \in S',\ \vec{v''} = \frac{\vec{v'}}{D},\ S'' = \{\vec{v''}\}$. This further makes the algorithm scale-invariant. The final feature vector $\vec{p}$ consists of concatenating the elements of S'' into a 24-dimensional vector.

3.4 Feature Sequence Recognition Using Dynamic Time Warping

To compare the similarity between two feature sequences, one should account for differences in the sequences that might vary in time or speed. The Dynamic Time Warping algorithm addresses these problems by computing an optimal match between two sequences with certain restrictions.

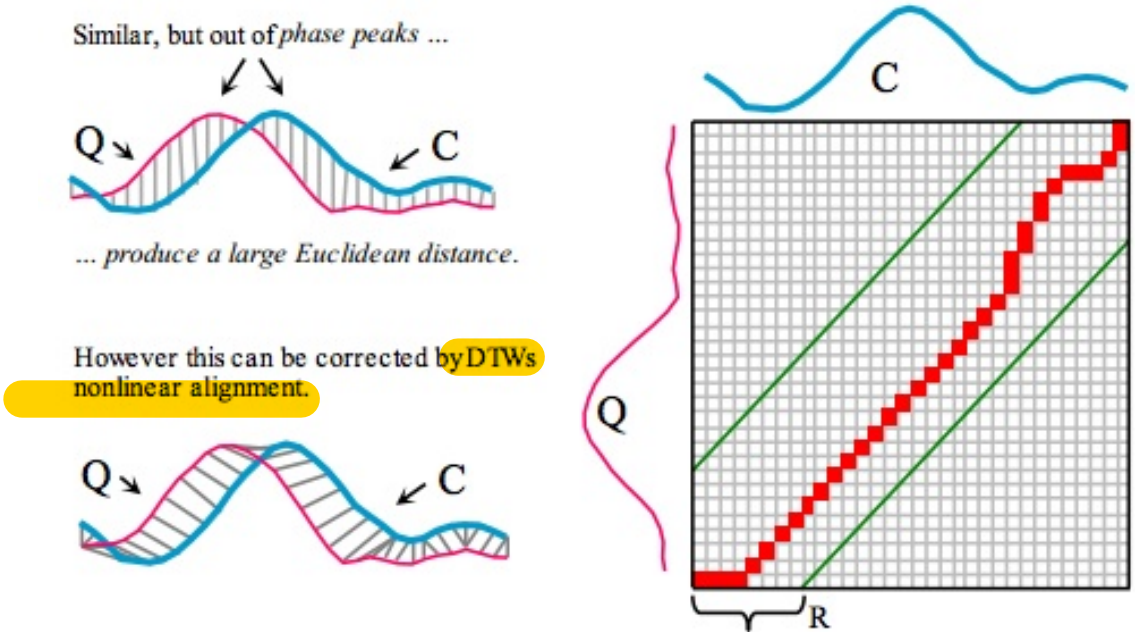
Formally, let the two sequences be $Q = \{Q_1,\ Q_2,\ \ldots\ Q_m\}$ and $C = \{C_1,\ C_2,\ \ldots\ C_n\}$ where each sequence is composed of features extracted from consecutive frames in time. Let the distance between features Q_i and C_j be the Euclidian distance denoted by $dist(i,\ j)$. The DTW algorithm aims to compute an optimal match cost matrix T of size $m\ x\ n$ where the element $T(i,j)$ denotes the optimal cost between the two sequences compared up to elements Q_i and C_j respectively. This is given by the recurrence:

$$T(i,j) = dist(i,j) + min[T(i,j-1), T(i-1,j), T(i-1,j-1)]$$

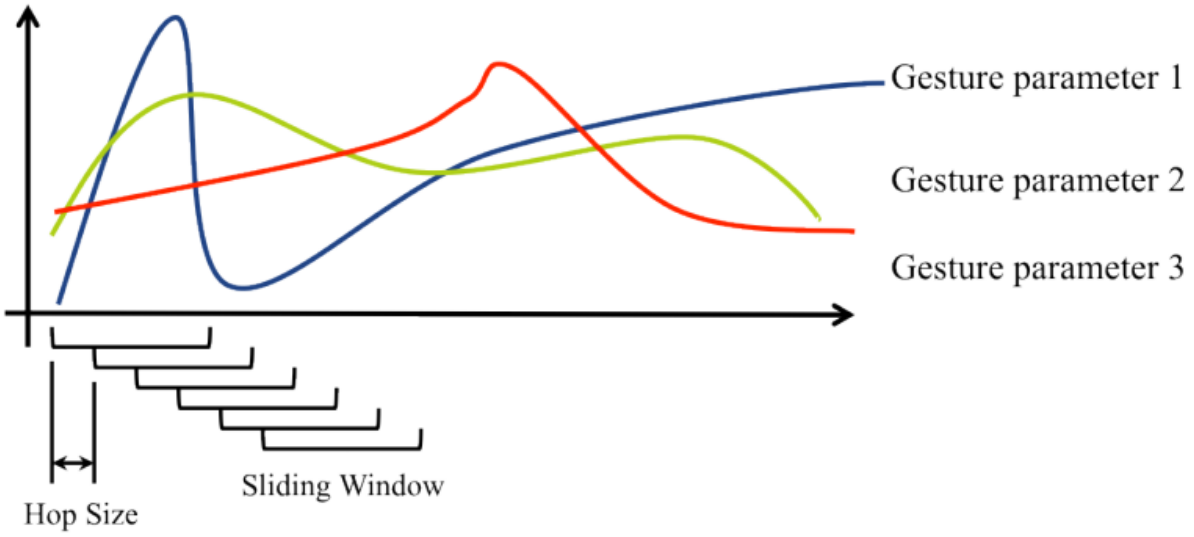
Intuitively explained, if $T(i,j-1)$ happens to be the minimum argument, then sequence Q is in a compressed subsequence, and if $T(i-1,j)$ is the minimum, then sequence Q is in an elongated subsequence. Only when $T(i-1,j-1)$ is chosen do both sequences realign. After the entire matrix is computed, we can obtain the optimal match cost between the two entire sequences by querying $T(m,n)$.

The original DTW algorithm in $O(mn)$. We can optimize this by applying the Sakoe-Chiba band [1], essentially pruning the search space by skipping any cell that lies outside a band width R from the diagonal, formally $\forall T(i,j)$ s.t. $|i-j| > R$. This reduces the search runtime to $O(mR)$.

The DTW approach is summarized by the following figure. The green lines represent the Sakoe-Chiba band, while the red highlighted path represents the optimal path that minimizes the pointwise Euclidean distance between the two sequences. For the mobile environment, we used $m = n = 60$ which corresponds to two second sequences, and a band width $R = 10$.



Finally, to be able to recognize gestures in a continuous fashion, we implement a sliding window that contains the last n features from the latest frames. When the window is full, the earliest feature is popped from the front and the new feature is pushed in the back. This is implemented using a `std::deque` data structure in C++. The sliding window is then taken as a feature sequence and compared against all sequences in the database, and the lowest cost optimal matched database sequence is taken as the winner -- as long as it is above a certain threshold. This is illustrated below:



Since the recognition pipeline must compare the current window against all gesture sequences in the database, the running time increases linearly with the number of gestures in the database. This could be mitigated by adding a hop size parameter h, limiting the invocation of DTW to once every h frames. In practice, because our data set is small enough, this was not necessary.

4 Experiments

Five different ASL gestures were tested: “I”, “you”, “love”, “no”, “please”. Using a sliding window width of 60 frames, a Sakoe-Chiba band of 10 frames, we were able to achieve 30 fps for 640x480 frames. Below are some captured images of the recognition pipeline correctly labeling the gesture in real-time.

To test the accuracy of the recognizer, each gesture was performed at two separate speed (slow and fast), and three different locations (left, middle, and right), and at two separate depths (near and far), resulting in 12 samples for each. A gesture is defined to be successfully recognized if the pipeline outputs the correct label for 90% of the duration of the gesture. Here are the results for each gesture:

	Near left	Near middle	Near right	Far left	Far middle	Far right	Total
I	2	2	2	2	1	2	91.67%
You	2	2	2	2	2	1	91.67%
Love	2	2	2	2	2	2	100%
No	2	1	2	1	1	1	66.67%
Please	2	2	2	2	2	2	100%

Most of the distinct gestures are correctly recognized accurately. The few problems arise when two gestures are very similar to each other position-wise, such as “I” and “no”. Because the current pipeline does not take into account hand contour or posture, and instead rely on 3D position coordinates, it is expected to get similar gestures wrong.

Performance wise, the pipeline spends around 20-25 ms per frame, which translates to a framerate of 30 fps. Because our sequences were limited in both window size and quantity to compare, memory footprint was not a concern. However, this could become a limiting factor if we were required to handle thousands of gestures.

5 Conclusion

In this project we explored the capability of a simple gesture recognition pipeline on a mobile device. We were able to show that given a limited number of gestures (sign language vocabulary of size 5) and a few algorithmic optimizations (dynamic time warping with locality constraint, Sakoe-Chiba band), we can achieve a real-time continuous gesture recognition system on a mobile device. However, we realize the limitations of the system -- the computational demand increases linearly with the number of gestures to compare against in the database, so that a complete sign language translator that can translate thousands of distinct ASL vocabulary words is infeasible with the current approach. To satisfy such demands, a system implemented using Hidden Markov Model or Deep Neural Networks would be more appropriate.

6 References

[1] Sakoe, H. & chiba, S. (1978). Dynamic programming algorithm optimization for spoken word recognition. IEEE Trans. Acoustics, Speech, and Signal Proc., Vol. ASSP-26. pp. 43-49.
[2] A. Corradini, “Dynamic time warping for off-line recognition of a small gesture vocabulary,” in RATFG-RTS ’01: Proceedings of the IEEE ICCV Workshop on Recognition, Analysis, and

Tracking of Faces and Gestures in Real-Time Systems (RATFG-RTS'01). Washington, DC, USA: IEEE Computer Society, 2001.
[3] H. Lee and J. Kim, "An HMM-Based Threshold Model Approach for Gesture Recognition," IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 21, No. 10, pp. 961-973, 1999
[4] S. Fels and G. Hinton, "Glove-talk: A neural network interface between a dataglove and a speech synthesizer," Neural Networks, IEEE Transactions on, vol. 4, no. 1, pp. 2–8, 1993.
[5] Structure Sensor. (2015, June 7). Retrieved from http://www.structure.io/
[6] Niisato, Hirotaka. (2014, April 17). "openFrameworks 8.1 and OpenNI 2.2 on Android tutorial". Retrieved from http://www.hirotakaster.com/weblog/openframeworks-8-1-and-openni-2-2-on-android-tutorial/
[7] J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake. 2011. Real-time human pose recognition in parts from single depth images. In *Proceedings of the 2011 IEEE Conference on Computer Vision and Pattern Recognition* (CVPR '11). IEEE Computer Society, Washington, DC, USA, 1297-1304. DOI=10.1109/CVPR.2011.5995316 http://dx.doi.org/10.1109/CVPR.2011.5995316