A Segmentation-based Approach for Realtime Gesture Recognition

Yifeng Huang <fyhuang>, Ivan Zhang <zhifanz>

March 16, 2011

Contents

1 Introduction

With the introduction of the Microsoft Kinect™and the subsequent release of open-source drivers for Windows, we have seen entirely new avenues of motion sensing techniques open which had previously been prohibitively expensive for individual experimention.

The first stage – that of converting raw RGB and depth images into a usable representation of the body – is provided by a motion tracking program. Our project takes motion tracking data from this program and tries to produce a more holistic representation of the user's semantic intentions. We accomplish this through a gesture recognition system where, for a given target application, a set of gestures is defined, and the system recognizes those gestures when performed by the user.

The gestures we have selected to recognize as a demonstration of the system are as follows: "clap", "flick_left", "flick_right", "high_kick", "jump", "low_kick", "punch", "throw", "wave". To simplify collection of training data, we recognized only "right-handed" gestures – that is, only punches made with the right hand, kicks made with the right foot, etc. are recognized when the gesture has a handedness. Most of these gestures are self-evident; a "clap" can be multiple or just a single clap, a "throw" resembles throwing a ball overhand, and the distinction between "low_kick" and "high_kick" is mainly in whether the foot comes to chest level or not. Samples of some of these gestures can be found in figures 3 and 4.

1.1 Related work

We also considered approaches such as dynamic time-warping and HMMs[1], as well as temporal Bayesian networks[2, 3], but ultimately implemented a two-stage algorithm using a simpler segmentation model and a learned recognition model.

2 Overview of model

2.1 Definitions and data formats

We will begin by introducing some terms to be used in the rest of this report. A "frame" is a single time slice. Each frame contains the locations and joint angles of all 20 joints. Each joint is represented as a 3-dimensional point; in addition, each joint has a parent joint, which is intuitively defined. (For a precise listing, see JointState.cs:43.) Using this, we define a "joint angle" to be the angle between the two vectors: $Pa(Joint) \rightarrow Joint$ and $Pa(Pa(Joint)) \rightarrow Pa(Joint)$. A "gesture" or "input gesture" is a non-empty sequence of frames. We call the output of the recognition engine a "result". Each result encapsulates three possible gesture "classes"/"types"/"names" that the engine thinks are most likely with their confidence scores.

2.2 Recognizing a gesture

The process of recognizing a gesture involves several steps. First, we train the various models used in the pro-

gram with an appropriate set of training data. There are two trained models: one, the "neutral stance" model, recognizes whether single frames represent a "neutral stance". The second, the "recognizer", takes input gestures (sequences of frames as defined above) and produces a result. During real-time operation, the engine receives 30 raw frames per second from the motion tracker. Raw frames are pre-processed by storing the absolute position of the neck joint, and then recomputing the positions of the other joints relative to the neck joint (joint angles are also computed at this step). This is necessary, as the Kinect is able to track backward- and forward motion, and our method needs to be translation-independent. The processed frames are sent to the "segmenter", which uses the "neutral stance" model to separate the input stream into logical segments. Some of these segments are then sent to the "recognizer" as input gestures. Results from the recognizer are then printed/acted upon as they are received.

This segmentation-based approach is the core of our ability to recognize gestures in real-time. By using a simpler method to first separate the continuous input stream into discrete segments, heuristically culling those segments, and only then running the full recognition model on those segments, we achieve high recognition speed while still maintaining good recognition accuracy. (See the results section for a more detailed description of our performance results.) Needless to say, this is important in real-time applications like video games.

2.3 The "neutral stance" model

The neutral stance model represents a stance where the user is idle and not in the middle of performing a gesture. The neutral stance model is fairly simplistic: we model the mean and variance of the normalized positions of all the joints from a customized set of parents. The selections were made in this model to fit closely with expectations about the human body (all the right arm joints, for example, were parented to the right shoulder joint), as well as to minimize model error from differing body shapes, height, etc.

2.4 Model for single gesture recognition

To formalize our model, we introduce the following notations: Let C be the overall gesture class node that is hidden and we would like to know its value, ranging from 1 to k, where k is the total number of gesture class labels. Let $F_1, \ldots, F_m$ be feature nodes that are parents of the input gesture sequence node D, and further in our most basic model, C is the parent of all feature nodes. The basic graphical model we used is shown in TODO. To model the cumulative distribution function (CDF) of the class label node, which takes on discrete values, given inputs from all feature nodes, which produce continuous values, we considered several different approaches (see the section on Development and Implementation for the full story).

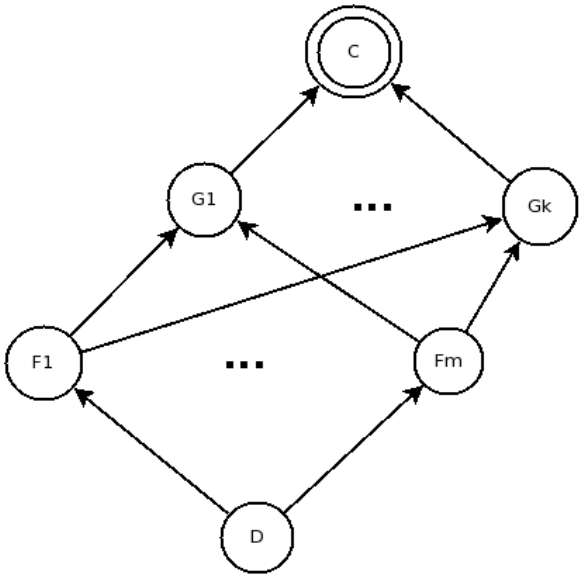

Figure 1: Logistic regression model

2.4.1 Logistic regression

One approach is that instead of considering the class label node by itself, we break it into k subnodes, $G_1, \ldots, G_k$, and let node C be the parent of all k gesture class subnodes. Each subnode G_i takes on binary values such that 1 means the input gesture sequence belongs to this particular gesture class i, and it is natural to model the relationship between each G_i and $F_1, \ldots, F_m$ as a bernoulli distribution, i.e. $P(G_i \mid F_1, \ldots, F_m) \sim Bernoulli(\theta)$ where θ is depended upon the values of F_1 through F_n. We can also view each $P(G_i = 1 \mid F_1, \ldots, F_m)$ as the confidence score that the given input gesture sequence belongs to this particular gesture class i. Then, the overall class label node, which takes on values 1 through k, is simply a selector variable that selects the gesture class that has the highest confidence score for this particular input gesture sequence. To determine the CDF of each subnode G_i, which follows a bernoulli distribution, we naturally use a logistic regression during learning.

2.4.2 Softmax regression

Another approach we considered was to model the distribution of C given feature values $F_1, \ldots, F_m$ according to a multinomial distribution, i.e. $P(C \mid F_1, \ldots, F_m) \sim Multinomial(\theta_1, \ldots, \theta_m)$ for some $\theta_1, \ldots, \theta_m$ such that their sum is 1. After we obtain the specific parameters

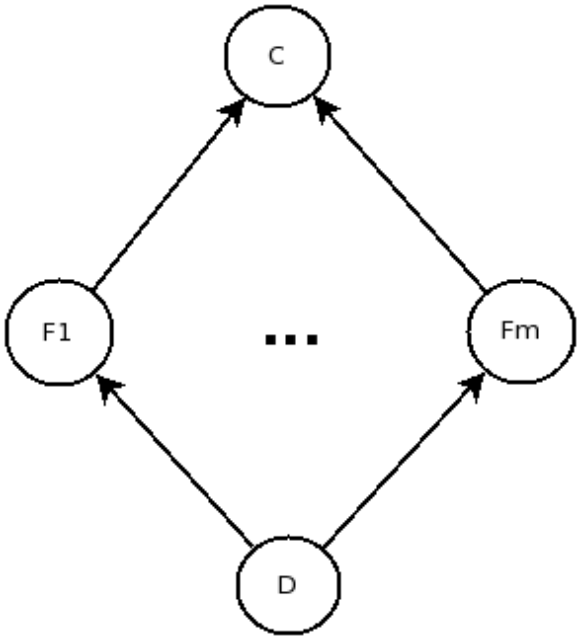

Figure 2: Softmax regression model

of this distribution given any inputs $F_1, \ldots, F_m$, we can treat each $P(C = k \mid F_1, \ldots, F_m)$ as a confidence score that this input gesture sequence belongs to a specific gesture class k. The final output of our "recognizer" is then simply the most probable gesture class under the computed set of feature values. To determine the CDF of class node C, which follows a multinomial distribution discussed above, we naturally use a softmax regression during learning.

3 Development and implementation

Early during development, we decided to implement cross-validation in order to take advantage of our small training data size. We use a roughly 75%/25% split for cross-validation. We first concentrated on the recognizer component (single gesture recognition), as we predicted that this part would be the most difficult to get good results in. The first model we implemented was a naïve Bayes model. Since the naïve Bayes model, in its simplest implementation, uses only binary features, we started with a small set of binary features. These were features, for example, that measured whether the difference between the minimum and the maximum values of a component (i.e. the X-position of a joint) over the entire input gesture was greater than a certain threshold. This produced significantly-better-than-random results, but the highest we could coax it to go was about 60% accuracy.

We quickly realized, however, that binary features were not going to be enough to encode the vast amount of variance and sublety present in the problem domain. At this point, taking inspiration from some comments made during the interactive sessions, we decided to implement a logistic regression model with continuous features. In both of these models, the features are functions which map an entire input gesture to a real number. In making the decision between choosing a more temporally-cogniscent model and modeling these temporal relationships with features, we chose the latter due to the much lower cost of learning and inference over the simpler model, in addition to the ability to easily encode certain features as entire-gesture functions rather than as graph structures. (For example, the difference between the minimum and the maximum of a component is significantly more compactly represented as a feature function.)

Upon implementing logistic regression and obtaining some promising results, we started to add more features to expand the power of our model. Surprisingly, the training data contained enough information to support at least 16 different features (which is what we ended up with). None of our features were learned with near-zero weight across all gesture classes. We did, however, begin to encounter non-convergence problems with our logistic regression. After speaking to Andrew and doing some research, we determined that some of our features were causing complete- or quasi-complete separation within our dataset. On realizing this, we added L_2-regularization to the logistic regression, as well as a check for completely-separating features. Upon the discovery of such a feature, the program warns the user and turns the feature off (effectively preventing it from having influence on either the training or the recognition process).

At this point, we were fairly satisfied with our results, but wanted to see if we could more naturally represent the relationship between the resulting gesture class and the feature set than a set of distinct logistic regressions. We decided to try a softmax regression with the same feature set. However, after running both the logistic regression and the softmax regression through a series of cross-validated tests, we found that the logistic regression still produced slightly better results, as well as being easier to work with in the code. When we were running experiments with the softmax regression, we discovered some interesting phenomena. Since we had not implemented separation-testing for the softmax code, we ran it using several features which were known to completely separate the training data. As a result, while the regression did not fail to converge, the model was completely unable to recognize test inputs, often producing results near random. We are still not completely certain whether the separation or simply a particular quirk of the feature causes this. Thus we decided to use the logistic regression as the final recognition technique.

3.1 Feature selection

We began with a fairly spartan set of features, measuring global (across a single input gesture) phenomena, such as (as mentioned) the difference between the peaks. This worked for fairly large movements, such as the punch and high kick gestures. We found that the data for the hand joint angle was quite noisy (by which we mean oscillating from 0 to 90 degrees at a time), so for the features that used joint angles we measured the wrist joint angle instead. This gives much less noise, but also means we cannot see the angle of the hand. The gestures we started having problems with were initially the two flick gestures, the wave, and the clap. We tried several feature types to account for the large errors in recognizing the flicks. First, we hypothesized that adding a directionality to the peak measurement (whether max or min comes first) would help us to distinguish between the left and right flicks. Unfortunately, we discovered a particular quirk of the way the training data was recorded: in many of the flick left training instances, the TA returned his hand beyond the initial point, causing the min and max measurements to go to unexpected values (and switch direction). Thus we had to use more sophisticated features to capture this difference. These and other quirks (the flicks were particularly inconsistent in their execution in the training data), along with the relative paucity of training instances, made the flicks among the hardest of the gestures to accurately identify.

Next, we tried to measure the overall "skew" of the gesture from the starting point: more positive or negative (say, on the X axis). This particular feature, however, caused such bad separation behavior that we were unable to actually use it. Next we incorporated a conditional measurement (say, to only apply the feature in frames where the hand is beyond a certain plane), in order to remove the influence of the starting and ending positions from the measurements. We also started adding multi-joint features: for example, a feature which measures the minimum distance between the hands at any point along the input gesture. This helped our performance greatly with the flick left gesture (and the corresponding maximum distance feature with the flick right), as well as the clap gesture (as might be expected).

To capture the jump motion, we had to re-incorporate some absolute position data, in this case using the absolute Y-position of the neck joint. Differentiating wave from the flicks proved to be a small problem: while we intuited that a feature which captured the number of critical points would be able to distinguish them (wave would have more critical points), implementing such a feature reliably proved to be difficult. Since the data is quite noisy, many "fake" critical points appear in the input, especially at regions where the joint does not move (i.e. the real critical points and the beginning and end of sequences). We had originally thought to implement smooting in the raw data converter, but decided it was easier to implement some crude smooting in the feature instead. Despite the inclusion of significants amount of noise in this feature, we found that it still had high effectiveness in distinguishing between wave and the other gestures.

3.2 Segmentation algorithm

The segmentation algorithm uses a more fixed heuristic, and learns only the neutral stance feature. Initially, we had thought about using a more fully-integrated system instead of our current, more isolated design. We met with Hendrik and discussed several more integrated gesture segmentation solutions, such as HMMs over the entire input sequence and dynamic time warping.[1] However, we determined that this method might be too expensive to implement for a real-time recognizer (especially one that would be competing for CPU time with the motion tracker). While we did some basic tests, we did not pursue more advanced optimizations such as dynamic programming methods. Instead, we decided to use a simpler model for the segmentation phase, reasoning that a simple "performing gesture" and "not performing gesture" distinction would be sufficient to achieve reasonable performance. This intuition proved to be somewhat correct: our original implementation of this idea was fairly successful. Basically, we look for long strings of non-neutral-stance frames and call that a gesture.

However, because no training data was provided for the neutral stance feature upon which this method depends, we culled our own set of training data from the provided sequences. Since the sequences involved only Hendrik, we inadvertently trained the model to match his body shape. Thus, when we tested the segmenter on a sequence recorded by me, it failed to segment even a single sequence. In order to improve the generalization performance of the segmenter, we tried a number of different approaches to learning the neutral stance.

At first, we tried to use the joint angles instead of the joint positions, reasoning that this would be more independent of body shape. While this was true, even with extreme weighting and damping, we were unable to effectively combat the noise from the angle measurements. We then tried multiple modifications of combining the joint angles and positions. Our current model calculates modified joint angles using different joint parents. By using more "distant" parents, we cut down significantly on sampling noise (for example, in the hand joints); and by choosing parents carefully, we can maintain sufficient body-shape independence. Currently, our model parents all the joints on the limbs to their respective "connecting" joints on the main body (i.e. right

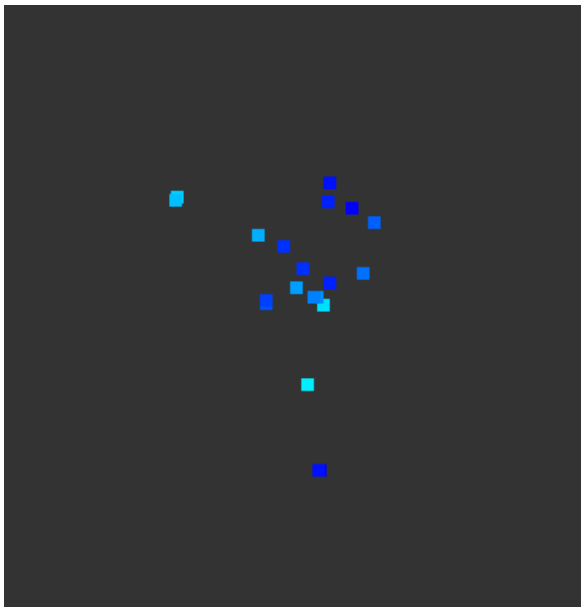

Figure 3: High kick gesture

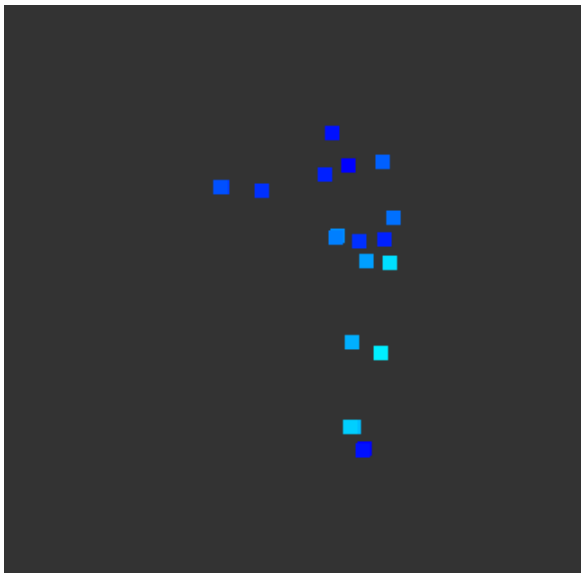

Figure 4: Punch gesture

elbow, right wrist, and right palm are parented to the right shoulder). This approach segments fairly effectively - we were able to achieve 60-70% accuracy (subjectively) on our testing sequences.

Finally, we noticed that most of the errors in our segmentation occurred when the user made gestures too quickly and did not return fully to the neutral stance. In these cases, our algorithm wouldn't break the gesture in the middle, and instead would attempt to recognize the entire sequence as a single gesture. To deal with this error, we introduced a damping factor to the confidence threshold. The damping factor increases as the length of the gesture increases. That is, as the gesture currently being segmented gets longer, the threshold of confidence that is required to call a frame a "neutral stance" gets smaller. This change has helped considerably in faster-moving parts of sequences.

4 Results

4.1 Visualizations of gestures

We created visualizations of some gestures to help the reader visualize the input data. These visualizations also helped us during the development process. One graph of the joint components over a particular gesture is also included (in figure 5).

4.2 Single-gesture recognition

Using the L_2-normalized logistic regression model, we achieved extremely good results in single-gesture recognition tests. Typically, cross-validation tests reported

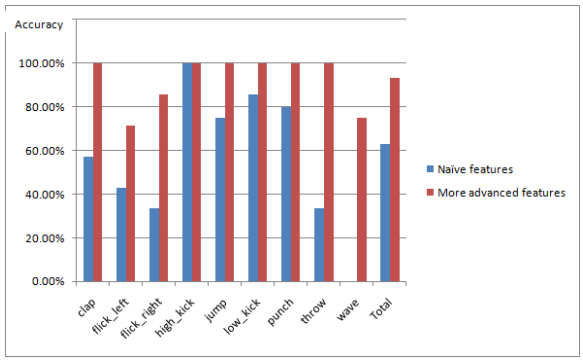

Figure 5: Performance comparison, initial features and final feature set

near or more than 90% accuracy. The output of a sample testing run can be seen in table 1.

We can immediately see the potential for this kind of model. We also include a plot which compares the results we obtain using our original set of naïve features and our final results using more holistic features. This comparison is in figure 6.

4.3 Real-time recognition

As mentioned above, we were able to achieve over 70% (subjective) accuracy in real-time recognition. (The results were measured by a human watching the animation and comparing his observations to the recognized gestures.) Some gestures are missed and some single gestures are "split", but for the most part, gestures can be performed in real-time with fairly high recognition rates. We have created a demonstration video which can be viewed at:

Gesture name	# correct	# total	% correct
clap	8	8	100%
flick_left	5	7	71.4%
flick_right	6	7	85.7%
high_kick	10	10	100%
jump	5	5	100%
low_kick	8	8	100%
punch	6	6	100%
throw	3	3	100%
wave	3	4	75%
Total	54	58	93.1%

Table 1: Sample run, cross-validation index = 0

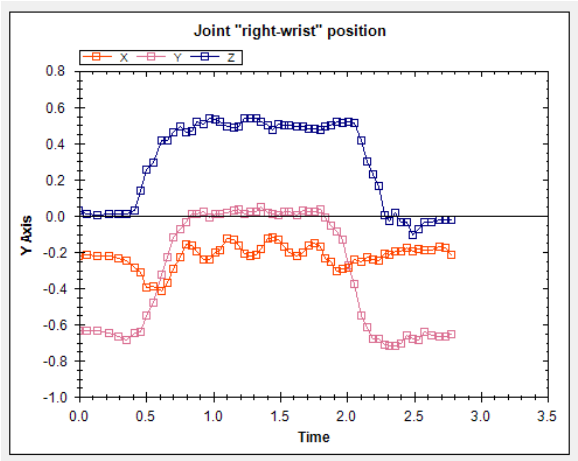

Figure 6: Plot of joint components in right-wrist during wave

`http://www.youtube.com/watch?v=9gNDSujA3DQ`

5 Conclusions and applications

We have prototyped one possible method of performing real-time gesture recognition. We think that this method presents a useful blend of speed and accuracy in real-time recognition, as well as fairly good robustness to noisy training data and features. Through doing this project, we have learned the importance of iterative development and of effective management of a large number of features and small data set in logistic regression. While it's certainly not optimal for every application, we could definitely see it or a similar approach being used in a fast-paced video game. After all, with so many punches and kicks in the training data, it would be a shame not to put them to use!

References

[1] Chunsheng Fang. From dynamic time warping (dtw) to hidden markov model (hmm). 2009.

[2] Ying Luo, Tzong-Der Wu, and Jenq-Neng Hwang. Object-based analysis and interpretation of human motion in sports video sequences by dynamic bayesian networks. *Computer Vision and Image Understanding*, 92(2-3):196–216, 2003.

[3] Vladimir Pavlovic, James M. Rehg, Tat-Jen Cham, and Kevin P. Murphy. A dynamic bayesian network approach to figure tracking using learned dynamic models. *Computer Vision, the Proceedings of the Seventh IEEE International Conference on*, 1:94–101, 1999.