Real-time Motion-based Gesture Recognition using the GPU

Mark Bayazit, Alex Couture-Beil, Greg Mori
School of Computing Science
Simon Fraser University
Burnaby, BC, Canada*

Abstract

In this paper we describe a real-time system for gesture recognition. Given an input video, we derive a set of motion features based on smoothed optical flow estimates. A user-centric representation of these features is obtained using face detection, and an efficient classifier is learned to discriminate between gestures. We develop a real-time system using GPU programming for implementing the classifier. We provide experimental results demonstrating the speed and efficacy of our system.

1 Introduction

Human gesture recognition in image sequences has many applications including human-computer interaction, surveillance, and video games. In this paper we describe a system for real-time gesture recognition that uses motion cues to discriminate between different gestures. Examples of the set of gestures we use in our experiments are shown in Fig. 1. Consider the video frames shown in Fig. 2, which show examples of motion cues that are used to discern which gesture is being performed.

In this work we focus on recognizing gestures – intentional, choreographed motions performed by a cooperative subject. The main contribution of this paper is developing a real-time system for gesture recognition based on optical flow features. The core algorithm for recognition is based on the work of Fathi and Mori [1], in which discriminative portions of optical flow are learned for recognizing actions. We use a variant of this algorithm, and combine it with a standard face detector [2] to localize the human figure in a video. We use the CUDA GPU programming API to create an efficient implementation of our action recognition algorithm. This results in a real-time system that can be used for applications in human-computer interaction. We demonstrate experimentally that it is efficient and effective for gesture recognition.

There is a vast computer vision literature in the "looking at people" domain. Moeslund et al. [3] and Mitra and Acharya [4] provide surveys of this area, in general and as it pertains to gesture respectively. Closely related pieces of work in the motion-analysis vein include Bobick and Davis [5], who represent global silhouette shape and motion using *temporal templates*. Shechtman and Irani [6] develop a motion-consistency method that avoids aperture effects. Efros et al. [7] recognize the actions of small scale figures using features derived from blurred optical flow estimates. In contrast to these methods, which are based on nearest-neighbour classification, we learn an efficient classifier suitable for real-time applications. Further, our classifier contains the specific parts of motion that are important for discrimination between gestures, and previous work has demonstrated their success on standard action recognition datasets [1].

Another group of methods analyzes the motion trajectories of skin-coloured blobs (typically hands, sometimes with face) [8, 9]. Reliably detecting and tracking hands, especially with fast motions, can be challenging. In addition, our optical flow-based method can incorporate motion cues beyond those in just the tracked hand regions.

Real-time systems include Ike et al. [10], who develop an efficient hand gesture system using multi-core processors. Our approach shares similarities, but is based on motion rather than shape and uses the CUDA API for efficient computation on the GPU.

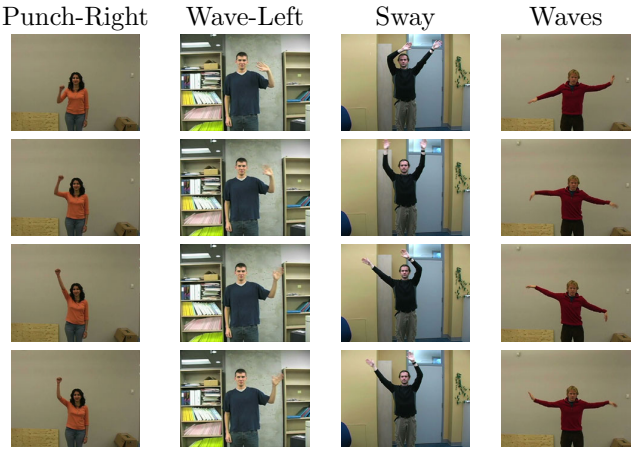


Figure 1: Example frames from a subset of the gestures used in our experiments.

2 Algorithm

The algorithm presented here is based on the work of Fathi and Mori [1]. In that work, which can handle data acquired from a moving camera, a stabilized human figure-centric representation is obtained by running a pedestrian detection algorithm. Human actions

*This research was conducted with the generous support of the Nvidia Professor Partnership and the Canadian Natural Science and Engineering Research Council CRD programs.

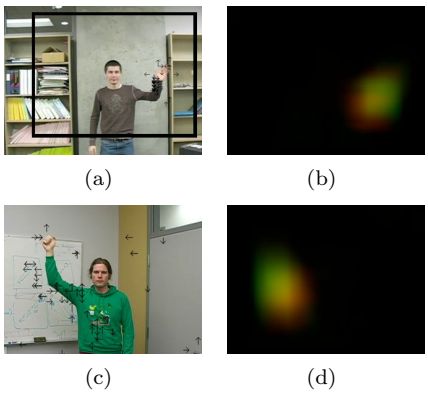


Figure 2: (a,c) Input frames overlaid with an illustration of motion features used to discriminate this gesture from others. The heaviest weighted motion features chosen by our algorithm are shown as arrows noting direction of motion. A face detector identifies a rectangular region (a) which is used to crop a user-centric image (c) before calculating the corresponding optical flow (b,d). Colour is used to denote direction of motion.

are then recognized using a classifier learned from optical flow features.

In our algorithm, motion features are first computed on the input image sequence (stationary camera assumed). A standard face detector is then employed to obtain a user-centric representation, and again a classifier to discriminate between gestures is learned using a variant of AdaBoost. A real-time version of this classifier is deployed using the GPU. In the following sections we provide the details of this algorithm.

2.1 Motion Features

To calculate the motion features, we first compute the optical flow for each frame. The optical flow vector field F is then split into horizontal and vertical components of the flow, F_x and F_y, each of which is then half-wave rectified into four non-negative channels F_{x^+}, F_{x^-}, F_{y^+}, F_{y^-}, similar to Efros et al. [7]. We add another channel corresponding to motion magnitude F_0 which is obtained by computing the L_2 norm of the four basic channels. These five non-negative channels are then normalized to facilitate gesture recognition in soft-real time where frame rates can be variable, and to account for different speed of motion by different users. Given a vector v that represents the optical flow for a given pixel, compute $v' = \frac{v}{||v||+\epsilon}$, where ϵ is used to squash optical flow vectors with very small magnitude, most likely introduced by noise. In experiments we set $\epsilon = 0.5$.

Next, each of the five channels is box-filtered to reduce sensitivity to small translations by the user performing the gesture. This final set of five channels: $\hat{F}_{x^+}$, $\hat{F}_{x^-}$, $\hat{F}_{y^+}$, $\hat{F}_{y^-}$, $\hat{F}_0$ will be used as our motion features for each frame.

We represent a gesture as a collection of movements required to complete a single phase of the gesture, rather than just capture a subset of the gesture phase. Hence, we aggregate the motion features over a temporal history of the last k frames, for some k which is large enough to capture all frames from a gesture phase. In practice, we set k to be the frame rate of our capture data; in other words, we assumed a gesture's phase was less than 1 second. Setting k too high will increase the classification response time when switching from one gesture to another. Setting k too low would increase the sensitivity of our algorithm to noise and variations in users' gesture speeds.

2.2 Face Detection

Face detection is used to create a normalized, user-centric view of the user. The image is scaled based on the radius of the detected face, and is then cropped and centered based on the position of the face (Fig. 2(a)).

The frame is cropped and resized to a 30×40 pixel region centered around the user. All five motion feature channels described above are flattened into a single vector $v \in \Re^{6000}$ ($6000 = 30 \times 40 \times 5$), which will be used to determine the gesture being performed.

2.3 Classification

The aforementioned motion features describe the user's entire motion. While these could be used directly for classification (e.g. nearest neighbour [7]), recent work [1] has suggested that learning a discriminative classifier from these features can highlight the important motions for characterizing particular gestures. Further, learning a classifier that uses a subset of the motion features, and does not have to compare to all training data, will lead to a more efficient system.

Given the labelled training data, we have a multi-class classification problem, separating the data into the different provided gesture classes. In this work we use the multi-class boosting algorithm AdaBoost.MH [11][1] that takes the motion features v as input. The supervised training is based on a set of labelled gestures. In the usual fashion we define a set of weak learners that are based on thresholding a value from a particular component of the motion feature vector v. For a gesture class l, each weak learner $h_t(v, l)$ is of the form:

$$h_t(v,l) = \begin{cases} 1 & \text{if } p_{t,l} v_{\tau(t)} < p_{t,l}\theta_t \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

for a motion feature v, where $\tau(t)$ selects a component of the feature vector, $\theta_t \in (-\infty, \infty)$ is the classification threshold of the classifier, and $p_{t,l} \in \{-1, +1\}$ is a parity for the inequality sign.

The output of the final strong learner on motion feature v for class label l is:

$$H_t(v,l) = \sum_{t=1}^{N} \alpha_t h_t(v,l) \quad (2)$$

where α_t are the weights chosen by AdaBoost.MH. The maximum output value can be found, or, as in our experiments, these values can be used to produce precision-recall curves to analyze the sensitivity of our algorithm.

3 Implementation

Our implementation uses the OpenCV library [12], which provides an implementation of Viola-Jones [2]

[1] We used the implementation available at `http://multiboost.sourceforge.net/`.

face detection, and Horn and Schunck optical flow [13]. We assume a multi-core system and dedicate a thread for face-detection. This non-real-time thread runs at a lower FPS than the main thread; however, since our gestures have minimal torso movement, it is acceptable to create a figure-centric frame based on the last known face position from some previous frame.

3.1 CUDA Classification Optimization

The classifier, which accounts for roughly 10% of CPU time, computes the summation of a set of independent weak-classifiers. Blocks of 512 weak classifiers for a given class l are computed in parallel with the CUDA GPU programming API [14].[2]

The four parameters associated with each weak classifier $h_t(v, l)$ are stored as elements of the arrays:

```
float alpha[ N ], theta[ N ]
int   parity[ N ][ K ], column[ N ]
```

where the $column[t]$ stores the value of $\tau(t)$ used to select the feature vector component, N is the number of weak classifiers, and K is the number of classes. The parity values are encoded as boolean values by mapping $\{-1, 1\}$ to $\{0, 1\}$ respectively. Our implementation used $N = 1024$, and $K = 7$.

The classification is computed with for a given 512 block and class l with the work kernel:

```
__global__ void classify_kernel(
  float* votes, int l, int offset )
{
  extern __shared__ float d[];
  const unsigned int tid = threadIdx.x;
  const unsigned int t = tid + offset;

  d[ tid ] = data[ column[ t ] ];
  d[ tid ] = d[ tid ] > threshold[ t ];
  d[ tid ] = d[ tid ] == parity[ t ][ l ];
  votes[ tid ] = d[ tid ] * alpha[ t ];
}
```

where *votes* is an array of length 512 used to save the votes, and $offset$ is a multiple of 512 used to select the appropriate block to compute. Upon completion, the individual votes stored in *votes* are summed via the reduce algorithm provided as a sample in the CUDA SDK. This process is repeated for each class parity $p \in \{0, 1, ..., K-1\}$ and block offset.

Fig. 3 compares the GPU and single-threaded CPU implementations on an Nvidia 9800GX2 and a dual-core Intel Xeon 3.2GHz. Each classifier was sampled 30 times, and ranges from 512 to 8192 weak classifiers. The CUDA-based implementation provides a speed-up factor of roughly four compared to the CPU method.

4 Results

We have created a dataset consisting of seven gestures (punch-left, punch-right, sway, wave-left, wave-right, waves, and idle) performed by ten different people. The videos were recorded indoors against various backgrounds at 29.97 frames per second (FPS) at 720×480 with a stationary Canon GL2 camera. Each gesture lasted from five to ten seconds, and included a minimum of five continuous cycles of the action. Samples of our dataset are displayed in Fig. 1.

The waves and sway gestures produce motion in the lower and upper sections of the frame respectively, and similar motion around the torso. Two pairs of gestures: punch-left and wave-left; and punch-right and wave-right produce similar motions on their respective sides of the frame. While it is easy to distinguish a left pair from a right pair, determining a wave from a punch may be more challenging.

[2] A similar approach can use SIMD instructions on the CPU.

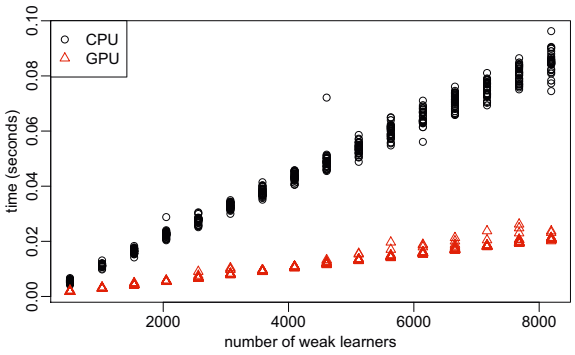


Figure 3: A parallel classifier implementation using the CUDA GPU programming API provides a speed-up factor of four compared to the CPU implementation.

Table 1: Smaller resolution input frames significantly improves efficiency, measured in frames per second (FPS), with minimal loss in accuracy.

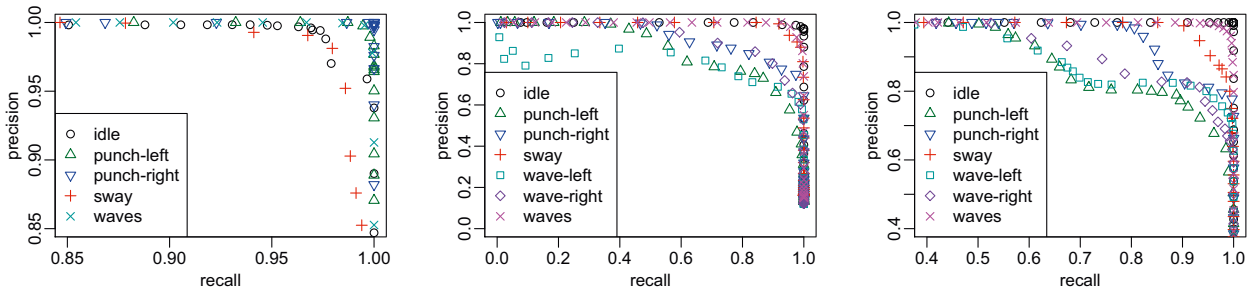
resolution	accuracy	FPS mean	FPS std dev
160×120	0.867	38.0	1.127
320×240	0.873	20.7	0.831

We achieved real-time results by resizing the user-centric frame before optical flow computation. We randomly selected a set of 30 videos from our dataset, and measured the FPS based on the processing time for the complete video. Table 1 show a smaller resolution gains a significant speed-up with minimal loss of accuracy. Where accuracy is defined as the percentage of the time the correct gesture receives the highest classification. The remainder of this section is dedicated to a more detailed analysis of our results, which were obtained using a resolution of 320×240.

We tested the classifier by performing leave-one-out cross-validation on three different configurations of our dataset.

1. The first configuration was a subset of gestures that eliminated the wave-left and wave-right gestures. The punching gestures can easily be identified by motion on either the left or right side of the frame; however, the sway and waves gestures produce motion is overlapping areas. The precision-recall graph in Fig. 4(a) verifies our algorithm is capable of recognizing a this subset of gestures with both precision and recall over 95%.

2. The second configuration we tested contained all gestures, including the two waving and punching pairs. The classification of these pairs proved to be difficult, as shown in Fig. 4(b). The confusion matrix in table 2 shows we were able to distinguish a left pair from a right pair, and only confused the gestures within a given pair.



(a) (b) (c)

Figure 4: Precision-recall graphs for different configurations of the gesture dataset. (a) a set of five gestures from distinct locations of the frame is performed by all ten subjects, (b) all seven gestures are performed by all ten subjects, (c) all seven gestures are performed by seven subjects who have minimal horizontal motion in the punching gestures.

Table 2: Confusion matrix for all seven gestures performed by all ten subjects. Rows represent the true action, and columns represent the percentage of actions returned by the classifier.

	sway	idle	waves	punch-left	wave-left	punch-right	wave-right
sway	1.00	0.00	0.00	0.00	0.00	0.00	0.00
idle	0.00	0.99	0.00	0.00	0.00	0.01	0.00
waves	0.06	0.00	0.94	0.00	0.00	0.00	0.00
punch-left	0.00	0.00	0.00	0.87	0.13	0.00	0.00
wave-left	0.00	0.00	0.00	0.32	0.68	0.00	0.00
punch-right	0.00	0.00	0.00	0.00	0.00	0.93	0.07
wave-right	0.10	0.00	0.00	0.00	0.00	0.14	0.76

3. After reviewing our dataset, we noticed that three of our subjects punched horizontally to the sides rather than vertically. It is not surprising that those three subjects had the poorest results; they were trained against seven vertical punchers and only two horizontal punchers. Fig. 4(c) confirms that limiting our dataset to the seven vertical punchers, and thus employing a stricter control over the gestures, increases our results. However, the disadvantage of this is that we must provide more detailed instructions to our subjects.

5 Conclusion

In this paper we presented a real-time gesture recognition system capable of classifying gestures. We employed a standard face detection algorithm to give a user-centric coordinate frame in which motion features were used to recognize gestures. While this use of face detection assumes a frontal face, i.e. the user is facing the camera, one could also develop a system using tracking in addition to face detection. This could allow the recognition of gestures with larger torso movements. Our system is capable of real-time performance. In particular, we use the CUDA API for GPU programming to obtain an efficient implementation of our classifier. One could also use the GPU for the face detection and optical flow computation, the other major processing needs of our system.

References

[1] A. Fathi and G. Mori, "Action recognition by learning mid-level motion features," in *Proc. IEEE Comput. Soc. Conf. Comput. Vision and Pattern Recogn.*, 2008.

[2] P. Viola and M. Jones, "Robust real-time face detection," *Int. Journal of Computer Vision*, vol. 57, no. 2, pp. 137–154, 2004.

[3] T. B. Moeslund, A. Hilton, and V. Kruger, "A survey of advances in vision-based human motion capture and analysis," *CVIU*, vol. 104, pp. 90–126, 2006.

[4] S. Mitra and T. Acharya, "Gesture recognition: A survey," *IEEE Transactions On Systems, Man, And Cybernetics - Part C: Applications And Reviews*, vol. 37, no. 3, pp. 311–324, May 2007.

[5] A. Bobick and J. Davis, "The recognition of human movement using temporal templates," *IEEE Trans. PAMI*, vol. 23, no. 3, pp. 257–267, 2001.

[6] E. Shechtman and M. Irani, "Space-time behavior based correlation," in *Proc. IEEE Comput. Soc. Conf. Comput. Vision and Pattern Recogn.*, 2005.

[7] A. Efros, A. Berg, G. Mori, and J. Malik, "Recognizing action at a distance," in *Proc. 9th Int. Conf. Computer Vision*, vol. 2, 2003, pp. 726–733.

[8] C. Rao, A. Yilmaz, and M. Shah, "View-invariant representation and recognition of actions," *Int. Journal of Computer Vision*, vol. 50, no. 2, pp. 203–226, 2002.

[9] M.-H. Yang and N. Ahuja, *Face Detection And Gesture Recognition For Human-Computer Interaction*. Kluwer Academic Publishers, 2001.

[10] T. Ike, N. Kishikawa, and B. Stenger, "A real-time hand gesture interface implemented on a multi-core processor," in *IAPR Conference on Machine Vision Applications*, 2007.

[11] R. Schapire and Y. Singer, "Improved boosting algorithms using confidence-rated predictions," *Machine Learning*, vol. 37, no. 3, pp. 297–336, 1999.

[12] G. Bradski and A. Kaehler, *Learning OpenCV: Computer Vision with the OpenCV Library*. Cambridge, MA: O'Reilly, 2008.

[13] B. Horn and B. Schunk, "Determining optical flow," *Artificial Intelligence*, vol. 17, pp. 185–203, 1981.

[14] J. Nickolls, I. Buck, M. Garland, and K. Skadron, "Scalable parallel programming with cuda," *Queue*, vol. 6, no. 2, pp. 40–53, 2008.