GestureSeg: Developing a Gesture Segmentation System using Gesture Execution Phase Labeling by Crowd Workers

Sven Kratz
FX Palo Alto Laboratory
3174 Porter Drive
Palo Alto, CA, 94304, USA
kratz@fxpal.com

Jason Wiese
FX Palo Alto Laboratory
3174 Porter Drive
Palo Alto, CA, 94304, USA
wiese@fxpal.com

ABSTRACT

Most current mobile and wearable devices are equipped with inertial measurement units (IMU) that allow the detection of motion gestures, which can be used for interactive applications. A difficult problem to solve, however, is how to separate ambient motion from an actual motion gesture input. In this work, we explore the use of motion gesture data labeled with gesture execution phases for training supervised learning classifiers for gesture segmentation. We believe that using gesture execution phase data can significantly improve the accuracy of gesture segmentation algorithms. We define gesture execution phases as the start, middle and end of each gesture. Since labeling motion gesture data with gesture execution phase information is work intensive, we used crowd workers to perform the labeling. Using this labeled data set, we trained SVM-based classifiers to segment motion gestures from ambient movement of the device. We describe initial results that indicate that gesture execution phase can be accurately recognized by SVM classifiers. Our main results show that training gesture segmentation classifiers with phase-labeled data substantially increases the accuracy of gesture segmentation: we achieved a gesture segmentation accuracy of 0.89 for simulated online segmentation using a sliding window approach.

ACM Classification Keywords

H.5.m. Information Interfaces and Presentation (e.g. HCI): Miscellaneous

INTRODUCTION

Most current smartphones and wearable devices such as smartwatches and fitness trackers are equipped with highly-sensitive inertial measurement units (IMUs). These sensors usually incorporate an accelerometer, a gyroscope and a magnetometer. This triplet of sensors provides information about the device's current acceleration, rate of rotation and absolute orientation. Motion information from a device's IMU is typically used for simple mode switching (changing the device's screen orientation from landscape to portrait) and for context-aware applications (e.g. inferring the user's activity or mode of transport), but it can also be used to recognize complex motion gestures [14, 20, 29].

ACM 978-1-4503-4322-0/16/06. . .$15.00
DOI: **http://dx.doi.org/10.1145/2933242.2933261**

Gesture-based user interfaces offer a valuable alternative to other modalities for capturing user input. For instance, smartwatch touchscreens are incredibly small, which limits their fidelity as an input device. Even on a smartphone using the touchscreen typically requires two hands: one to hold the device and one to interact. Speech interfaces require speaking out loud, which can be embarrassing (i.e., when walking on the sidewalk), or even inappropriate (i.e., during a meeting). Hardware buttons offer a limited input vocabulary (typically binary), and take up valuable physical space on a device. While gestures are not always appropriate, they offer a large vocabulary as input space, and in the right context they contribute to an enjoyable user experience (as evidenced by the success of the Wii game console).

While the broad vocabulary of gestural interactions is a major part of its value, one consequence of this broad vocabulary is that it can be difficult for a developer who is deploying a gesture-based interface to design a new gesture and deploy a system that recognizes that gesture with high precision and accuracy. For example, a developer has to segment a gesture (i.e., the segmentation of the sensor's data stream into non-gesture and gesture segments), choose useful features for identifying a gesture, and handle uncertainty in the gesture interface.

Gesture segmentation in particular is a significant issue for designers of user interfaces with motion gesture recognition. To best make use of the fluidity that gestures enable for interactive use, automatic segmentation of the gesture from non-gesture movements is highly desirable. This is a challenging problem to solve, which is evident by the many previous works [14, 20, 17] that have mostly relied on manual gesture segmentation using a *push-to-gesture* mechanism, i.e., requiring some sort of user action to delimit the start and end of an input gesture. Another approach [27] used a predefined *delimiter gesture* to mark the start of a subsequent input gesture. These methods all require an extra step that interrupts the fluidity of interacting with a gestural interface and thus detracts from the user experience of using a gestural interface. This paper introduces a new approach for automatically segmenting gestures in an effort to make automatic segmentation easier and more accurate.

Specifically, we present three main contributions to the area of automatic gesture segmentation for IMU-based gesture data:

First, we demonstrate the feasibility of training supervised learning algorithms to classify specific gesture execution phases, i.e., the *start*, *middle* and *end* of a gesture, as well as the actual gesture class of the classified execution phase. Apart from applying gesture phase detection to the problem of automatic gesture segmentation, gesture phase detection can also be used to increase the responsiveness of gesture-based interfaces, as, e.g., it leaves developers the option of providing guiding feedback during gesture execution, or already start executing commands tied to a gesture (i.e., when the start of a gesture has been detected.) while the user is finishing up the gesture movement.

Second, we describe the collection of a comprehensive gesture data set with manually-labeled gesture execution phases. We describe the technical set-up we used and the lessons learned when using crowd workers to accomplish the time-intensive task of labeling motion data in the way we propose.

Third, we show how we used this data set to train a SVM-based classifier for automatic gesture segmentation. Our results indicate that using gesture phase information significantly improves the accuracy of segmenting motion gesture data from ambient motion in comparison to using labeled data where gesture phase information is not used.

RELATED WORK

In the following, we provide an overview of related work in three fields that are closely tied to the topic of this paper: *IMU-based motion gesture recognition*, *gesture segmentation* and *crowd annotation of machine learning data sets*:

IMU-Based Motion Gesture Recognition

A significant amount of work has been accomplished in the area of IMU-based gesture recognition, with the application of several different machine learning algorithms. Schloemer et al. [29] used Hidden Markov Models to detect gestures using data from a game controller's IMU. Hoffman et al. [14] applied AdaBoost [23] to an IMU-based gesture data set in order to improve the recognition accuracy. Several previous works [2, 19, 20, 24] used Dynamic Time Warping [28] for gesture recognition. Lastly, Kratz et al. studied methods for achieving rotation-invariant template-based gesture recognition [18] and the effects combining accelerometer and gyroscope data for different approaches to gesture recognition [19].

Gesture Segmentation

There has been much prior work in the domain of gesture segmentation. Older work has focused mainly on gesture segmentation from video [3, 4, 10]. A specific sub-domain of gesture recognition is hand gesture segmentation [7, 21, 34], mainly to support the recognition of hand sign language. Further work has focused on body segmenting body movement gestures, e.g., for dance [15]. More recently, Wu et al. [35] used depth information in addition to color imagery and deep learning for gesture segmentation.

In contrast to video, IMUs provide data that is far sparser than video. IMU data can also be noisy, increasing the difficulty of segmentation and recognition. Thus, many of the contribution in this vast body of work cannot be applied directly to IMU-base gesture recognition or segmentation.

There have been some previous works on gesture segmentation for IMU-based data. Ashbrook [6] suggests using a threshold on the variance of the last N samples as decision criterium for gesture segmentation. We believe that this approach has two drawbacks: (1) although we might be able to segment gesture data from inactivity, we have no further information about the gesture the user is inputing (we believe that the gesture ID can be detected with some certainty from the start phase of a gesture), and (2), this method might not perform well when deployed in the field and used in "noisy" environments where the user is subjected to motion, e.g., when using public transport.

Xu et al. [36] describe a classifier for IMU-based hand gesture that also contains a module for gesture segmentation. They basically extend Ashbrook's approach by (heuristically) checking for additional parameters. The authors also propose a method of reconciling the heuristic application on multiple data axes. However, they do not provide any information on the accuracy of their segmentation approach.

Ruiz et al. [27] proposed using a classifier to detect a "gatekeeper" gesture that would then enable the entry of a subsequent command gesture. The results which we presented in this paper suggest that such a design is not necessary and that the intended gesture can be recognized directly.

Crowd Annotation of Gesture data

A large body of work has previously described the use of crowd workers for the annotation of machine learning data sets in general as well as gesture recognition in particular. For instance, Spiro et al. [31] used crowd workers to annotate hand movements in video. The *Glance* project used crowd workers to rapidly code behavioral events in behavioral videos [22]. Ouyang et al. [26] used crowd sourcing to create a gesture vocabulary for touchscreen-based gestures for launching mobile applications. The *CrowdLearner* project [5] used a crowd-based approach to train classifiers for motion an on-screen gestures.

We are, at present, not aware of any previous publication that has tasked crowd workers with labeling gesture execution phases of such data using video and sensor visualization. However, Grijincu et al. [12] used video and crowd workers to annotate video of gestures on tabletop interface.

FEASIBILITY OF GESTURE PHASE CLASSIFICATION

To test the initial assumption that gesture execution phases (i.e., start, middle and end of a gesture) can be recognized by a classifier, we analyzed a motion gesture data set that was used in a previous work [13]. When gathering this data set, 25 test users were asked to perform at least 20 repetitions of

6 different motion gestures. The data (temporal sequences of 3-axis acceleration, rotation rate and absolute orientation) was gathered on an iPhone 4 at a frequency of 100 Hz. The motion gestures were manually segmented using a *push-to-gesture* button, delimiting the start (pushing the button and holding it) and end of a gesture (releasing the button). In this way, a total of 3507 gesture entries were recorded.

Extracting Gesture Phase Labels

As we only recorded motion data when the *push-to-gesture* button was pushed and held, we did not know the exact length of the proposed gesture *start*, *middle* and *end* phases. For this reason, we conducted an exploratory statistical analysis on the number of data samples per gesture entry in our data set (Table 1):

	average	median	st. dev.	max	min
Nr. of Samples	286	266	109	1766	24

Table 1. A set of exploratory statistics on the number of data samples per gesture entry. N=3507 gestures in total.

Using this empirical data on gesture sample lengths we computed the phase lengths for segmenting the gesture into the three proposed phases using a value L_{cutoff}, defined as:

$$L_{\text{cutoff}} = \mu - 2\sigma \tag{1}$$

Figure 2 shows how we used L_{cutoff} to segment the data for a complete gesture into three phases. We also used L_{cutoff} to filter out gestures that are unusually short, i.e., we considered only gestures with a length of L_{cutoff} or greater to build a segmentation model.

Segmenting the gesture data like this, we obtain three labelled data classes, *start*, *middle* and *end*. To obtain a visual indication that the data would be separable, we conducted a Principle Component Analysis (PCA) on the labeled data. Figure 1 shows 2D and 3D plots of the PCA with data dimensionality reduced to two and three dimensions, respectively. Visually, Figure 1 indicates the possibility of separating *start* and *end* from *middle*, as those points are spatially well separated. At this projection dimensionality, there appears, however, to be a substantial intermixing of *start* and *end* (red and blue dots), which may pose a problem for a classifier.

To obtain an initial classification result, we trained a Support Vector Machine (SVM) classifier with a Radial Basis Function (RBF) kernel with the segmented and labeled data. Using 6-fold cross validation, the average accuracy for classifying the gesture phases was 88.1%.

Discussion

Our preliminary results show that it is indeed possible to train a classifier to recognize distinct phases of gesture entry. However, our initial approach has the following shortcomings:

1. The data set used in the preliminary study only includes motion samples between pressing and releasing the *push-to-gesture* button. It may, however, be of interest to also consider motion samples shortly before and after gesture execution, as these could contain important motion information that could be improve the classification accuracy of the gesture phases.
2. Since we also want to delimit gestures from other motion, we will need to train a classifier for a further label, *noise*. This will allow us to delimit a legitimate gesture entry from uninteresting motions of the mobile device.

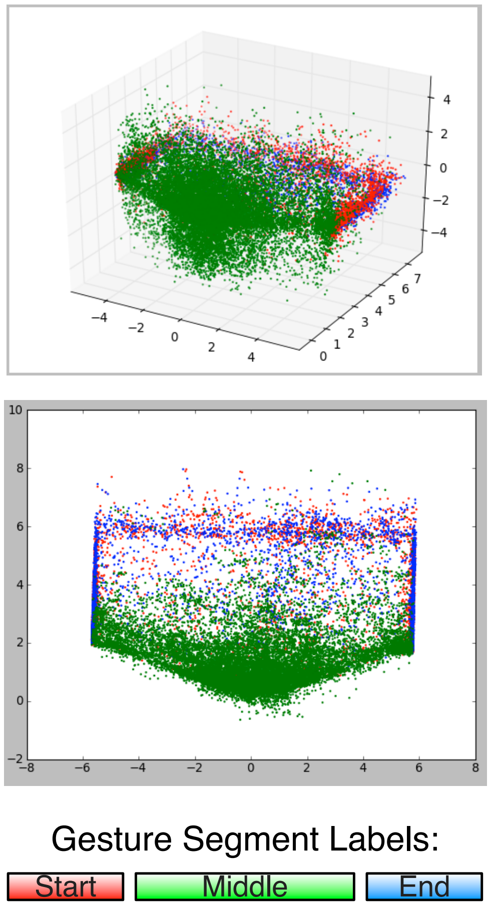


Figure 1. 2D and 3D PCA plots of the segmented gesture data. The plots provide a visual indication that the *start* and *end* gesture segment classes can be separated by a classifier from *middle*.

RECORDING AN IMPROVED GESTURE DATA SET

To address the issues with the gesture data set used in the previous section, we conducted a new round of gesture recordings to obtain an improved data set. Our goal was to cap-

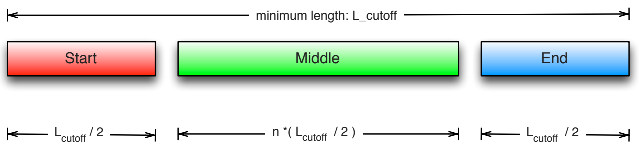


Figure 2. The proposed segmentation strategy for our existing data set is to build a classifier for the start, middle and end phases of a motion gesture. We used empirical measurements of gesture lengths to determine L_{cutoff} in order to set the correct number of sample points for each of the gesture phases.

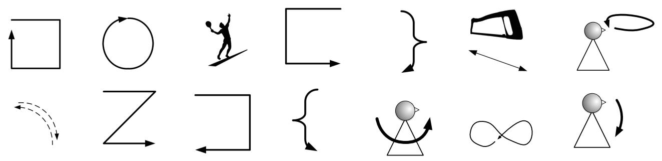


Figure 3. The 14 gestures of the newly recorded gesture data set.

ture the complete data stream from each user session, so that "noise" samples would also be included. In addition to gestures manually delimited with a *push-to-gesture* button, we also captured free-form gestures from the users without any active delimitation.

For the gesture recording we recruited a total of 10 participants (3 female, 1 left-handed), all staff members of an industrial research lab. The majority of the participants were between 30 and 45 years old. On average, the participants reported that they moderate experience (2.7 on a 5-point Likert Scale, 5=*very experienced*) with motion gestures.

For the data set, we captured a set of 14 different gestures with a minimum of 15 repetitions per gesture for each user, so about 2100 total gesture entries. We chose the gesture set from gestures used previously in the literature [13, 14, 17, 29], so that it is possible to make comparisons (e.g., classification accuracy) with previous work using a subset of gesture types used in our research. Figure 3 shows an overview of the gestures we used. During the recording sessions, users were shown these images as a guide for performing each gesture type.

We used a Bluetooth YEI 3-Space IMU [37] to record the gesture data. We recorded the following 3-axis motion information at a rate of 120 Hz: absolute orientation, rotation rate, acceleration and "linearized acceleration".[1]

To support manual segmentation of the motion data into distinct gesture phases, we also captured all user gesture entries on video from the front and from the side of the user using a pair of cameras. We implemented a short script in Python using OpenCV to juxtapose the two video views and record them to a single video file for each user and gesture type.

The test users found it easy to translate the graphical gesture representations of the gestures to motions—after data sampling was completed, they rated the ease of this process with an average of 4.4 on a 5-point Likert Scale (5=*very easy*).

GESTURE PHASE LABELING VIA CROWD WORKERS

We initially implemented a software tool [16] that can be used to graphically annotate the gesture data in order to define the different phases (start, middle and end) of a gesture that we are interested in. However, initial attempts at annotating the gesture data manually this way revealed to be a very time consuming process: about 10 minutes are needed to annotate the 15 gesture entries per gesture type per each user. Thus, we estimate that to completely annotate the entire 10 user data set a proficient annotator will need about 23 h.

[1]This is the orientation-compensated acceleration of the device, with the gravity acceleration component removed [37].

Web-Based Labeling Tool

To alleviate the time demands of manually labeling the gesture data, we developed a web-based labeling tool for use with Amazon Mechanical Turk crowd workers. Figure 4 (a) shows a screenshot of the labeling tool. The tool shows a time plot of the 12 sensor values contained in the data together with the video of a given gesture entry. A time cursor on the gesture plot is synchronized with the video, so that the annotator can correlate the video and sensor data in order to delineate the individual gesture phases. Further controls allowed the annotator to select the video playback speed (full, 1/2, 1/4 and pause) and define the annotation type. The interface provides buttons to switch between label types (start, mid and end). Workers can label the data by dragging along the data plot, with the desired label type selected. A link to a help page [32] with a tutorial video was provided on the labeling tool's webpage.

Implementation Details

The web-based labeling tool was implemented using the D3.js framework [8] for to generate the plot. We implemented a custom server backend in Python that, once a task is submitted by a worker, stores the task metadata and data labels in a MongoDB database. The webpage is displayed directly on Amazon Turk's worker interface through an iframe being served by our server.

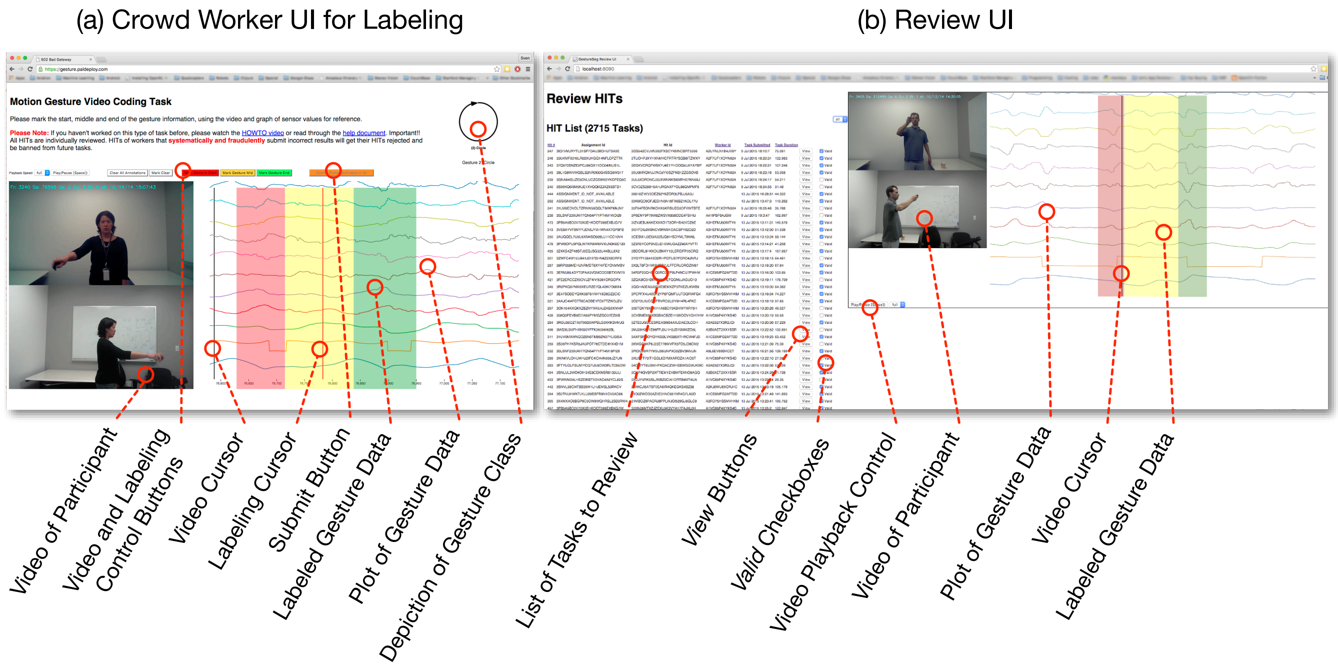
UI Design Considerations for Use on Crowd Working Platforms

Initial test-postings of gesture labeling tasks, or "Human Input Tasks" (HITs) in Amazon Mechanical Turk jargon, showed some design weaknesses of the labeling interface that needed to be corrected:

- **Validation:** not all crowd workers completed the task correctly. We therefore added validation code that ensured gesture labels were entered *completely*, i.e., all phases of a gesture input are labeled, and *correctly*, i.e., start, mid and end phases of the gesture input are labeled in this order.

- **Task Fatigue and Abuse:** we noticed that some particular workers would start out working on the tasks normally, but would gradually transition to inputting low-quality annotations. At the same time, the task execution time would go down significantly. We took this as a sign that the workers were abusing the task and rapidly entering valid but non-meaningful labelings. To correct this issue, we implemented a timeout for the submit button, that would allow task submission at a random time of 40–70 s after the task had started (before implementing this the average task time for non-abusing workers was around 2 minutes).

- **Task Window Size:** for another set of initial crowd workers, we observed consistently incorrect annotations, although they had not submitted tasks with unusual rapidity. After further investigation, we noticed that labels had only been added on one side of the the gesture plot. This led us to the conclusion that part of the task window must have been obscured by a low screen resolution or shrunken window size. We thus added JavaScript code to the web


Motion Gesture Video Coding Task

Please mark the start, middle and end of the gesture information, using the video and graph of sensor values for reference.

Please Note: If you haven't worked on this type of task before, please watch the HOWTO video or read through the help document. Important!! All HITs are individually reviewed. HITs of workers that systematically and fraudulently submit incorrect results will get their HITs rejected and be banned from future tasks.

Review HITs

HIT List (2715 Tasks)

Figure 4. (a) Shows the web-based data labeling tool for use by crowd workers. (b) Shows the web-based review tool used to validate the labeling inputs.

interface to disable the task interface until a the worker adjusted the task window to a certain minimum size.

Data Preprocessing for Amazon Turk

To permit data labeling via crowd workers, we had to shorten the original gesture recordings as they contained on average 15 gesture entries and thus took up to 10 minutes to label manually—a timespan we deemed was too long for an individual crowd worker to work on a single data set. To divide up the data into smaller subtasks, we therefore used the button-delimited subset of the data we recorded (where individual gestures had been delimited using the IMU's button).

We used the existing button presses as markers to delimit the gesture. However, we sampled the data such that the 200 ms preceding and trailing the button-based delimitation were also visible to the crowd worker. This was done based on our assumption important information may be contained in the data preceding and trailing the delimiting button press, e.g., cases where the participant initiated or terminated the gesture entry before pressing or after releasing the button, respectively. While 200 ms may appear short, it is actually a quite generous addition of data for each gesture. Adding 200ms at the start and the end of the gesture constitutes roughly 19% increase in the available data, given the average total gesture length of 248.5 samples and a sampling rate of 120 Hz.

As the video data had also been recorded matching the original data sets of 15 gesture entries, we implemented shell scripts using FFMPEG [1] to process the video data as previously described.

In this way, we generated a total of 2424 pairs of data and corresponding video samples.

Review Tool

In initial task deployments, we recognized that a number of labeling tasks were not completed correctly by the crowd workers. We therefore implemented a web-based tool to review the crowd workers' results. Figure 4 (b) shows a screenshot of the review tool. It features two main panes. The left pane shows a list with each row containing information (loaded from a database) on a submitted worker tasks. On each worker task row, there is a *view* button. Clicking this button shows the video snippet, plot of gesture data and annotation results (superimposed on the plot) for the associated task. If the task is valid, a *valid* checkbox can be clicked, and the task's status in the database will be updated to be valid.

Implementation Details

We adapted the D3.js data viewing components of the worker interface to develop the review tool. We made use of the React.js framework [11] to display a dynamic list view containing the worker task metadata. The backend reused most of the original server script and accessed the MongoDB database containing the worker task metadata.

Results of Crowd Labeling

After finishing the implementation of worker interface, reviewer interface and the backends, we deployed the labeling tasks on Amazon mechanical turk. A total of 2715 tasks were deployed on Amazon Mechanical turk. This number takes into account invalid tasks that were completed incorrectly and subsequent re-submitted. At the end of our one-week deployment on Amazon mechanical turk, we obtained a total of 2192 correctly labeled data sets.

A total of 89 distinct crowd workers worked on the data. The most productive worker completed 730 correct task assignments, the least one task assignment. The median number of tasks completed correctly per worker was 5.

INITIAL RESULTS WITH NEW DATASETS

Our initial results indicate that it is possible to train a classifier to distinguish between the start, middle and end phases of gesture entries. This previous classifier was trained generally over all different gesture types. For our manually-annotated gesture data, we trained classifiers for each phase of each gesture. Thus, we intended to find out if it would be possible to classify the gesture type, for instance, just by looking at the start phase.

We trained the classifiers using the WEKA toolkit [33], and therefore preprocessed the gesture data as follows:

All gesture data was mean-shifted and normalized to a $[-1, 1]$ interval. Then, to obtain feature vectors of homogenous length we subsampled each marked gesture phase segment to contain 10 samples of the 12 data points provided by the sensor. Thus each feature vector has a length of 120. We used linear interpolation to subsample the data evenly from the source annotations.

We trained a multi-class SVM classifier (with a radial basis function kernel, with parameters $C = 4.0$ and $\gamma = 0.5$, obtained via a manual grid search for best accuracy) on training data labeled with the gesture ID and gesture phase (i.e., a total of 42 different classes). Our results using 10-fold cross validation show an average precision of 0.93 (and F1 score of 0.921) with a minimum precision of 0.78 and a maximum accuracy of 1.0. These results not only indicate that we can classify the *type* of gesture being performed but also the current *phase* of gesture performance.

For comparison with our first data set, we also trained a multi-class SVM classifier (again with a Radial Basis Function kernel, $C = 4.0$ and $\gamma = 0.5$) on the training data labeled just with the gesture phase for each gesture entry (i.e., three different classes). Using 10-fold cross-validation we obtained an average accuracy of 0.95 (and F1 score of 0.95). This result provides us with a verification that our new data set is on par or better than our initial data set. We also get an indication that manual segmentation, taking part of the lead-in and lead out of a gesture into consideration, yields improved recognition accuracy for gesture-phase classification.

However, these results only show the detectability of gesture phases, not the possibility of segmenting them (or entire gestures) from noise data. In the following, we will more closely explore gesture segmentation, using the crowd-labeled data set and an SVM-based approach for segmentation.

SVM-BASED GESTURE SEGMENTATION ALGORITHM

In the following we will describe an SVM-based method to gesture segmentation. We present a comparison it to a method [6] previously described in the literature and also study the effects of using gesture phase information on the segmentation accuracy.

Data Type	**Noise**	**Start**	**Mid**	**End**
Sample Count	923,502	184,507	319,134	185,936
Equiv. time in min	128	25	44	25

Table 2. Number and type of samples in our data set. The minute equivalent time is based on the IMU's 120 Hz sampling rate.

Data Characteristics

Data captured from the IMU is represented as temporal sequences of 12 dimensional samples representing the IMU's sensor readings at a given point in time. Starting out, we split the data into four subsets based on the data label classes: *Noise*, *Start*, *Mid* and *End* data. Consequently, Noise data was all data that was not labeled with any class, Start data is data labeled as belonging to the start of a gesture, Mid data was labeled as belonging to the middle of a gesture and End data was labeled as belonging to the end of a gesture. The following table provides an overview of the number and type of samples obtained through our capturing sessions and crowd-based labeling:

To obtain a better understanding of the labeled data set, we conducted an initial statistical analysis on the lengths of all labeled gesture segments, which is shown in Table 3.

Gesture Phase	**Start**	**Mid**	**End**
Number of Instances	2708	2708	2708
Average Length	66.0	114.8	67.7
Median Length	56	101	56
Length Std. Dev.	32.6	37.2	37.2

Table 3. Descriptive statistics on the length of the labeled gesture segments. Label count denotes the total number of data intervals of a given gesture phase label.

Data Classes

We hypothesized that the labeling of gesture phase data (i.e., the start, middle and end of a gesture) would improve gesture segmentation accuracy. Thus we extracted the following classes of data from the dataset.

- *noise*: all data that was not labeled as belonging to a gesture is labeled as *noise*. 2714 instances.
- *start*: all data that was labeled by crowd workers as the start of a gesture is labeled as *start*. 2708 instances.
- *mid*: all data that was labeled by crowd workers as the middle of a gesture is labeled as *mid*. 2708 instances.
- *end*: all data that was labeled by crowd workers as the end of a gesture is labeled as *end*. 2708 instances.
- *gesture*: all data from a contiguous start-mid-end sequence was labeled as *gesture*. This class is used to compare the accuracy of classifiers trained on a gesture segment to that trained on entire gestures. 2655 instances.

Data Preprocessing

We preprocessed data sequences representing gestures, gesture phases or noise sequences for building SVM-based models. To begin, we mean-shifted and normalized the data magnitude to lie within a $[-1, 1]$ interval. Because gesture entries are sequences of IMU measurements of variable length, we needed to apply subsampling to equalize the sequence length for the purpose of representing these subsampled sequences as feature vectors for training SVM classifiers. Thus, we subsampled each labeled data segment to a standard length of 30 vectors. This resulted in the feature vector for each data segment to consist of 30 * 12 = 360 elements. We implemented subsampling using linear interpolation, using Code 1 (see Appendix). In a deployed version of this model, similar preprocessing would take place on the stream of incoming data, using the parameters calculated during the training of the classifier.

SVM-Based Segmentation Classifier C_{SVM}

We used a SVM-based classifier for gesture segmentation. For successful segmentation, the goal of the classifier was set up to distinguish between noise and gesture data. We used the *libsvm* [9] SVM library. We configured libsvm to SVM models using a radial basis function kernel with the following parameters $C = 1.5$, $\gamma = 0.125$. The parameters C and γ were obtained using a manual grid search for best accuracy. In the following we will refer to the SVM-based classifier as C_{SVM}.

Heuristic Segmentation Classifier of C_{HEUR}

As a comparison to C_{SVM}, we implemented a heuristic segmentation classifier, C_{HEUR}, based on an approach previously described by Ashbrook [6]. This approach is based on calculating the standard deviation of noise data and gesture data. We calculated the standard deviation for all sequences belonging to the noise, gesture, start, mid and end classes, respectively. We then aggregated this data, for the sequences of all the aforementioned classes, respectively, by calculating average standard deviation across all sequence samples. This results in a 12 dimensional vector representing the average standard deviation for each of the classes.

For classification, we then use the average standard deviation of the class (i.e., gesture, start, mid and end) we want to segment as the threshold for the input sequences. We do this by checking which components of the input sequence's standard deviation lie above or below the threshold, obtaining a boolean vector of comparison results. We then use the mode of the comparison result to determine the type of data the input sequence belongs to. Code 2 (see Appendix) shows the implementation of the heuristic function we used.

Results of Data Class and Classifier Comparison

Using our labeled data set, we ran a comparison of the SVM-based classifier C_{SVM} with the heuristic approach C_{HEUR}. Since we hypothesize that using gesture phase information can improve gesture segmentation accuracy, we compared the accuracy of training C_{SVM} and C_{HEUR} using data labeled with gesture phase information vs. gesture-only labeling.

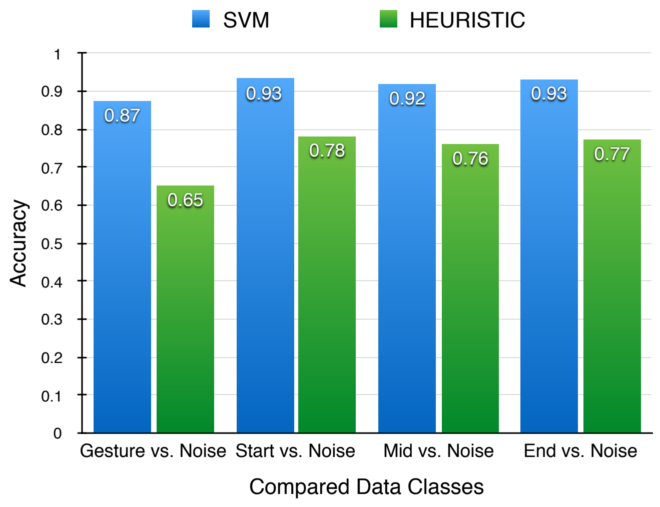


Figure 5. This figure shows a comparison of the accuracy of the SVM-based (C_{SVM}) and heuristic (C_{HEUR}) classifiers for classification of each the *gesture*, *start*, *mid*, and *data* data classes vs. *noise*.

We thus compared the classification accuracy of classifying each of the *gesture*, *start*, *mid*, and *data* classes vs. *noise*.

To ensure generalizability of the results, we used 10-fold cross-validation for evaluating C_{SVM}. Since there is less concern with overfitting due to the simplicity of C_{HEUR}, we did not use cross-validation with that technique.

The results of the comparison are shown in Figure 5. As can be seen, C_{SVM} performs better than C_{HEUR} for every data class. More importantly, the results also show that using data labeled with gesture phase clearly improves segmentation accuracy: the accuracies for *gesture* vs. *noise* (no phase information) are 0.874 and 0.653 for C_{SVM} and C_{HEUR}, respectively, and the best accuracy was achieved when using *start* vs. *noise* (with phase information) with 0.934 and 0.78 for C_{SVM} and C_{HEUR}, respectively.

ONLINE SEGMENTATION ALGORITHM

The results of the previous section show that classifiers can distinguish between noise data and gesture data with a relatively high accuracy. Using gesture phase information to label the gesture data yields the most accurate result. However, the previous comparison was run in-place using statically-labelled data. To obtain better insights into how C_{SVM} or C_{HEUR} would perform on system with live streaming data, we implemented a further test harness to train and classify the algorithm under a simulated live data capture scenario.

Data Chunking and Relabeling

To simulate live data streaming, we partitioned our data into chunks and attempt to classify each chunk individually as noise or gesture data. This reflects the way we envision a gesture segmentation algorithm operating on live data could be implemented: collect a time series of measurements from an IMU up to a certain "chunk" length, and then classify the time series of the chunk to determine if a gesture is being executed or not, then repeat the process.

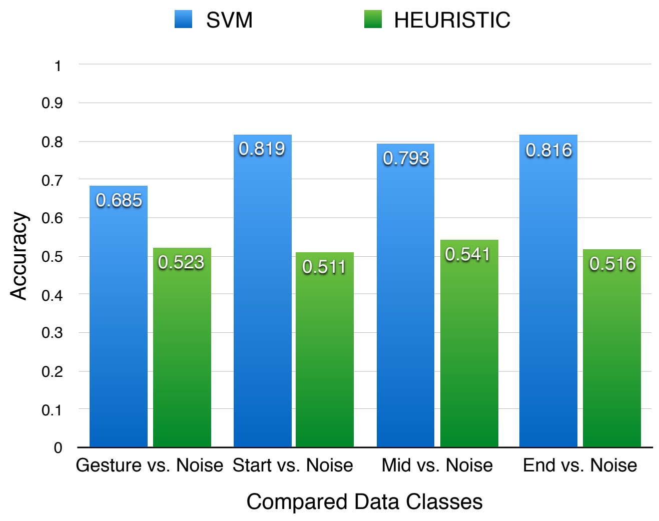


Figure 6. This figure shows the accuracy results for the online segmentation algorithm using SVM-based (C_{SVM}) and heuristic (C_{HEUR}) classifiers for classification of each the *gesture*, *start*, *mid*, and *data* data classes vs. *noise*.

We chose 83 samples as the chunk size. This is the rounded up average (82.83) of gesture phase segment length as shown in Table 3. To obtain labels for the extracted chunks for supervised learning, we used the gesture segment labels from the original data set to assign a sublabel to each item in a chunk, corresponding to its position and label in the original data set. In this case, chunks can initially have sublabels of multiple types. To find a unified label for the entire chunk, we calculated the mode of all labels and use this as the final label for each chunk. Code 3 in the Appendix shows how we implemented the chunking and labeling.

Chunking the data in this way resulted in the following amounts of labeled chunks:

Label	**Noise**	**Start**	**Mid**	**End**
Count	9753	2306	3886	2135

Table 4. Number of chunks per data class after chunking.

Classifier Training

We preprocessed the labeled chunks as described previously to train C_{SVM}. Again, we were interested in the effect of the labeled gesture phases on training. Thus, we compared the accuracy of classifying noise vs. the following data classes: *start*, *mid*, *end* and *gesture*, where gesture are start, mid and end combined into a single label. We used 10-fold cross validation when evaluating C_{SVM}. Again as a comparison, we also evaluated C_{HEUR} on the same data classes.

Results of Data Class and Classifier Comparison

Figure 6 shows the comparison results for the online algorithm. Again, C_{SVM} had a higher accuracy than C_{HEUR} with a significant margin. The accuracy results of C_{SVM} using gesture phase labeled data vs. plain gesture data are similar to those of the previous study (see Figure 5): using the whole gesture for training yields significantly lower segmentation accuracy than using either of the gesture phases, 0.685 vs. 0.819 for gesture vs. start, respectively. Of the gesture phase labeled data, *start* data yielded the highest accuracy, with *end* second and *mid* third, with minimal difference between start and end classes.

These evaluation results of online gesture classification show that our method is robust under (simulated) live streaming conditions. The fact that these results effectively mirror the previous evaluation using the original static-labeled data further strengthens our hypothesis that using gesture phase information for gesture segmentation classifiers improves segmentation accuracy.

SLIDING WINDOW ONLINE SEGMENTATION ALGORITHM

The evaluation segmentation algorithm described in the previous section is potentially limited by the fixed data segments that are used in training and validating the machine learning approach. To gain further insights into the effects of gesture phase data on automatic segmentation we also explored a sliding-window approach to online segmentation.

Window Generation and Classifier Training

Instead of using fixed chunks as in the previous section, we used a sliding window on the gesture data to train and test a supervised learning algorithm, which decides if the window contains gesture or noise data. As the size of the sliding window, we again chose the average gesture phase segment length of 83 data points. We generated the sliding window data by shifting a window of 83 data points across our data sets in 1 data point increments. This resulted in a total of 1,612,996 windows on the data. To generate labels for supervised learning we used the gesture segment labels from the original data set to a assign a sub label to each item in a window, corresponding to its position and label in the original data set. We then calculated the mode of all sublabels in the window to determine the final label for the window.

Because of the large number of potential training samples, we used only every 100th window for training classifiers. Validation was conducted using the entire windowed data set. To evaluate the effect of gesture phase data on automatic classification, we trained multiclass SVM classifiers on the following combinations of data class sets:

$S_0 = \{noise, start, mid, end\}$

$S_1 = \{start,\ noise \cup mid \cup end\}$

$S_2 = \{mid,\ noise \cup start \cup end\ \}$

$S_3 = \{end,\ noise \cup start \cup end\ \}$

$S_4 = \{noise,\ start \cup mid \cup end\ \}$

Using *libmSVM*, we trained a multiclass SVM for S_1 and binary SVMs for S_2–S_4. We used radial basis function kernels with the same parameters as in the previous section. As a comparison, we also evaluated C_{HEUR} on the same data class sets, however with the loss of multiclass classification.

Results of Sliding Window Segmentation

Figure7 shows the accuracy results of our analysis of the online segmentation algorithm using the sliding window ap-

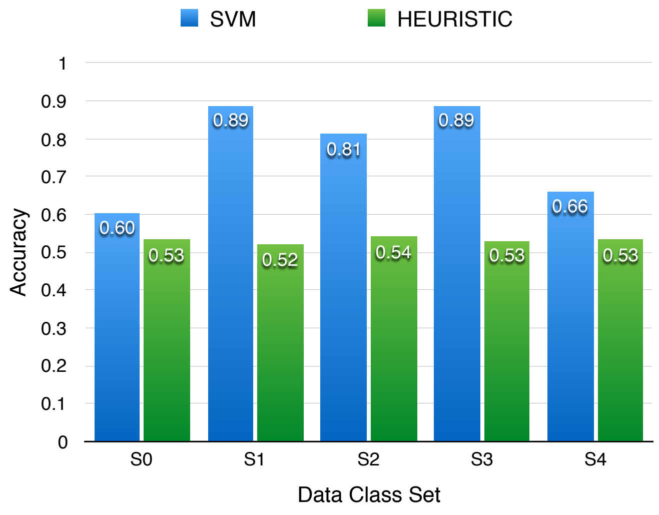


Figure 7. This figure shows the accuracy results for the online segmentation algorithm using C_{SVM} and C_{HEUR} using a sliding window approach.

proach. Similar to the previous evaluations, training the algorithms with data sets using gesture segment data (S_1–S_3) to differentiate between a specific gesture phase and noise plus other gesture phases resulted in the highest accuracy scores, with S_1 using C_{SVM} achieving an accuracy of 0.819. Also similar to the previous analyses, C_{HEUR} resulted in the lowest accuracy accross all data class sets.

Analysis of Offset Effect

We synthetically generated a label for each offset window based on the ground truth of our labeled data set. Thus, each window is offset from the start of the nearest ground truth label in the labeled data set by a specific value. Because the data set is fully labeled, the range of possible offsets from the start of nearest ground truth label is constrained. Our analysis indicates that the offset ranges between (-)446 samples before and (+)445 samples after the start of the nearest ground truth label (which is defined to be at 0 offset. Figure 8, *bottom*, shows a density plot of the offsets of the sliding windows. We can see that the offset of the majority of windows lies within ± 100 of the start of a ground truth label.

Figure 8, *top*, shows the effect of offset on segmentation accuracy. Although the average accuracy is acceptable, there is a noticeable dip for approximately the first 50 samples after offset 0. We believe this is not a severe issue as the accuracy quickly recovers after the dip. We furthermore believe that the dip is due to labels changing in the original data set (e.g., from *noise* to *start*) at offset 0, and this is probably the part of the data which appears most ambiguous to classifiers. We think that the observed dip and recovery of the segmentation accuracy could cause a low amount of lag or unpredictability for the onset point of detecting the start of a gesture when our approach is used in a live system. We will need to be examine this phenomenon more closely in future work.

DISCUSSION

In this paper, we explored the hypothesis that using motion gesture data labeled with gesture phase information increases

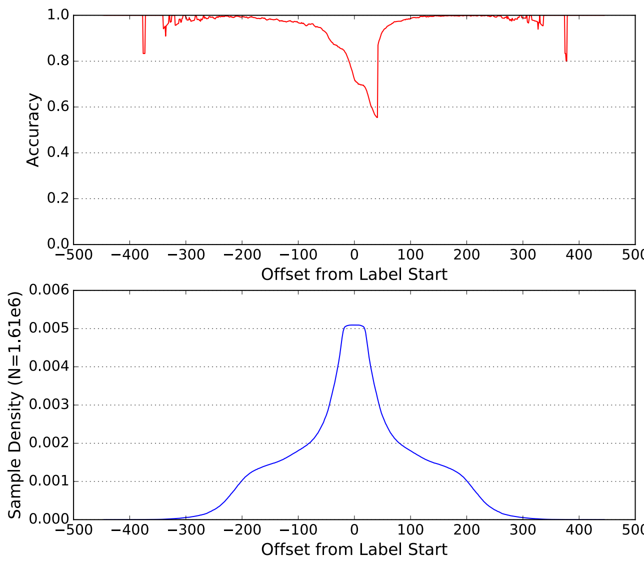


Figure 8. Analysis of the effect of the offset of the window with respect to the original labeled data sample. The top plot shows the segmentation accuracy given the offset point from the start of a labeled segment for C_{SVM} on data set S_1. The bottom plot shows the density of samples with a given offset.

the accuracy of automatic gesture segmentation when compared with using gesture data without additional phase information. We described our process for collecting a new motion dataset and for using crowd workers to label the data with gesture phase information. This dataset allowed us to test our hypothesis by comparing the accuracy of SVM-based classification and a heuristic technique on the different types of labeled data. We conclude from our results that using gesture phase information improves automatic segmentation accuracy.

Use of Crowd Workers for Labeling Motion Gesture Data

One insight we gained in this paper is that, given the proper instructions and validation routines, crowd workers can be used effectively to label motion gesture data. The quality of the data labeling obtained from the crowd workers has resulted to acceptable machine learning accuracy for supervised learning, and, although the development of the web-based user interface for use by the crowd workers took a substantial amount of time, the execution speed of manual labeling using crowd workers was impressive: all 2715 tasks were completed within 3–4 hours of an afternoon weekday.

Feedback in User Interfaces

One of the particularly interesting results of these experiments was the high accuracy of the 42-class C_{SVM} classifier. In general, the more classes that a classifier has, the worse its accuracy is. The high accuracy of this many-class classifier demonstrates the potential for identifying a gesture just from its start, while the user is still in the process of completing that gesture. This kind of subdivision of recognizing a gesture is particularly valuable in providing a fluid user interface. For example, rather than a developer receiving an `onGestureCompleted` callback in her code, she could instead receive `onGestureStart`, `onGestureMiddle`, and `onGestureEnd` for detecting a particular gesture, similar to the event callback for a mouse (e.g. `onMouseHover`, `onMouseDown`, `onMouseUp`, etc.). The interface can then provide more immediate feedback to the user (i.e. that the gesture is being recognized, and updating the UI accordingly).

Handling Uncertainty

This approach of segmenting a gesture into its start, middle, and end also has the potential to improve the handling of uncertainty in detecting gestures. A whole-gesture classifier has one chance to classify a gesture correctly. However, classifying the start, middle, and end of a gesture offers the possibility of three different classifications, which could be combined to improve the classification accuracy. One way this could be used is as a tie-breaker (e.g. first use a whole-gesture classifier and then follow up with gesture phase classifiers if the confidence is too low). They could also be used as votes (e.g. only two of the three gesture phases have to classify the correct gesture, the minority vote is thrown out). Furthermore, this approach maps nicely to the concepts introduced by Schwarz et al. [30]

LIMITATIONS AND FUTURE WORK

While the results of these experiments demonstrate the potential value of using gesture phase classification to improve the accuracy of segmenting gestures from noise and classifying those gestures, there are some limitations to the existing experiment design that highlight important opportunities for future work.

Dataset and Crowd Labeling

Although the crowd workers generally performed well on their labeling tasks, the dataset may still be imperfect. Due to the sheer number of workers and task items, there may still be errors in the labeling. Another issue is that the labeling may not be optimally consistent between crowd workers. Although we specified how the data should be labeled in a series of tutorial videos shown to the crowd workers, there may still be individual variations between crowd workers that we could not control.

Using data initially delimited with a ”push-to-gesture” button may have potentially injected some hysteresis into the data set, favoring the recognition of gesture start and stop points. However, we believe that the button push does not have a significant effect: The button on the IMU were micro switches with very little travel. Since it was a physical button, the users usually already had their finger placed on it. So the total movement required to press the button was much smaller than, for instance, tapping a touch screen button on a smartphone. More important, the section “Initial Results” presents gesture phase recognition results using a subset of the non-button data that was manually annotated, which clearly shows that gesture phases can be detected independent of any button press. Our main results reflect this.

Additionally, all data was recorded under laboratory conditions, with the users mostly stationary. Future evaluations will need to test whether the approach proposed in this paper fares well under higher-intensity user motion (e.g., traveling on public transport). However, the present results should be suitable for enabling auto-segmenting gestural interaction for scenarios where the user is using the device in a stationary setting. Furthermore, we achieved good segmentation accuracy results without needing to model individual users.

Testing Method

The tests we conducted demonstrate the potential value of using gesture phase data for training classifiers for automatic gesture segmentation. However, all tests were conducted on the existing data set. Even the simulated live streaming evaluation may be overly optimistic. Thus a follow-up experiment with users entering live data is desirable.

In the case of the online segmentation algorithm using a sliding window approach, segmenting the data set into sliding windows resulted in an overabundance of training data. We had to subsample to every 500th window to obtain acceptable model generation times with using SVMs, yet still used 16130 labeled feature vectors for training. Other machine learning techniques, e.g., deep learning, that are better suited to large training data sets might result improved accuracy in this case.

We believe, however, that the current paper contributes the foundations needed to implement live gesture segmentation system in the near future: the engineering effort that we put into generating the labeled data set and the results of tests we conducted on the use of gesture phase information for automatic gesture segmentation will be of great utility when developing such a system.

Future Work

With the insights gained from this paper, our immediate intent is to develop and evaluate a live prototype gesture segmentation application on a wearable device such as a smartwatch. Furthermore, we aim to explore the use of further machine learning approaches with the aim of increasing our segmentation accuracy. Lastly, we would like to extend our approach to gesture segmentation to other sensor techniques such as skeletal tracking by depth cameras and use it to enable gesture segmentation in augmented workspaces, where, for instance, a users’s desk is tracked via a depth camera.

APPENDIX

This appendix contains listings of several Python methods which we referenced in the paper and were used in preprocessing and conducting evaluations on our data sets. All code is written in Python using the NumPy [25] scientific computing library.

Code 1. Data subsampling.

```python
def subsample(data,  num_items):
    rows = data.shape[0]
    subsampled_data = []
    for k in xrange(num_items):
        idx = float(k) * float(rows-1) \
            / float(num_items)
        low = math.floor(idx)
        high = math.ceil(idx)
        fac = idx - low
        # linear interpolation between
        # low and high
        value = data[low, :] * (1.0-fac) \
                + data[high, :] * fac
        subsampled_data.append(value)
    return subsampled_data
```

Code 2. The comparison function for C_{HEUR}. The function returns either true or false.

```
def heuristic(sequence):
        std = np.std(sequence, axis = 0)
        comp = std > NOISE_THRESH_STD
        r =  mode(comp)[0][0]
        return r
```

Code 3. Chunking/labeling algorithm for online segmentation test.

```
def chunk_and_label(in_data,
                    in_labels,
                    chunk_len = 83):
    chunks = []
    labels = []
    datalen = len(in_data)
    idx = 0
    while idx + chunk_len < datalen:
        fr = idx
        if idx + chunk_len > datalen:
            to = datalen
        else:
            to = idx+chunk_len
        majority = mode(in_labels[fr:to])
        labels.append(majority)
        chunks.append(in_data[fr:to])
        idx = to
    return chunks, labels
```

REFERENCES

1. FFMPEG: a complete, cross-platform solution to record, convert and stream audio and video. **https://www.ffmpeg.org/**, January 2016.
2. Akl, A., and Valaee, S. Accelerometer-based gesture recognition via dynamic-time warping, affinity propagation, & compressive sensing. In *Acoustics Speech and Signal Processing (ICASSP), 2010 IEEE International Conference on*, IEEE (2010), 2270–2273.
3. Alon, J. *Spatiotemporal gesture segmentation*. PhD thesis, Ph. D. Thesis, Boston University, 2006.
4. Alon, J., Athitsos, V., Yuan, Q., and Sclaroff, S. A unified framework for gesture recognition and spatiotemporal gesture segmentation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on 31*, 9 (2009), 1685–1699.
5. Amini, S., and Li, Y. Crowdlearner: rapidly creating mobile recognizers using crowdsourcing. In *Proceedings of the 26th annual ACM symposium on User interface software and technology*, ACM (2013), 163–172.
6. Ashbrook, D. *Enabling Mobile Microinteractions*. PhD thesis, Georgia Institute of Technology, 2010.
7. Bhuyan, M., Ghosh, D., and Bora, P. Continuous hand gesture segmentation and co-articulation detection. In *Computer Vision, Graphics and Image Processing*. Springer, 2006, 564–575.
8. Bostock, M. D3.js: Data-Driven Documents. **http://d3js.org/**, January 2016.
9. Chang, C.-C., and Lin, C.-J. Libsvm: A library for support vector machines. *ACM Trans. Intell. Syst. Technol. 2*, 3 (May 2011), 27:1–27:27.
10. Dong, Q., Wu, Y., and Hu, Z. Gesture segmentation from a video sequence using greedy similarity measure. In *Pattern Recognition, 2006. ICPR 2006. 18th International Conference on*, vol. 1, IEEE (2006), 331–334.
11. Facebook, inc. React: a JavaScript Library for Building User Interfaces. **https://facebook.github.io/react/**, January 2016.
12. Grijincu, D., Nacenta, M. A., and Kristensson, P. O. User-defined interface gestures: Dataset and analysis. In *Proceedings of the Ninth ACM International Conference on Interactive Tabletops and Surfaces*, ACM (2014), 25–34.
13. Guse, D. Gesture-Based User Authentication on Mobile Devices using Accelerometer and Gyroscope (Master's Thesis). *Quality and Usability Group, Deutsche Telekom Laboratories, TU Berlin* (2011).
14. Hoffman, M., and Varcholik, P. Breaking the status quo: Improving 3d gesture recognition with spatially convenient input devices. In *IEEE Virtual Reality Conference (VR)*, IEEE (Waltham, Massachusetts, USA, 2010), 59–66.
15. Kahol, K., Tripathi, P., and Panchanathan, S. Automated gesture segmentation from dance sequences. In *Automatic Face and Gesture Recognition, 2004. Proceedings. Sixth IEEE International Conference on*, IEEE (2004), 883–888.
16. Kratz, S., and Back, M. Towards accurate automatic segmentation of imu-tracked motion gestures. In *CHI 2015 Extended Abstracts* (April 2015).
17. Kratz, S., and Rohs, M. A $3 Gesture Recognizer - Simple Gesture Recognition for Devices Equipped with 3D Acceleration Sensors. In *Proceedings of the 14th international conference on Intelligent user interfaces.* (Hong Kong, China, Feb. 2010).
18. Kratz, S., and Rohs, M. Protractor3d: A closed-form solution to rotation-invariant 3d gestures. In *Proceedings of the International Conference on Intelligent User Interfaces (IUI 2011), Palo Alto, CA, USA, February 13-16, 2011. Short paper*, IUI '11, ACM (New York, NY, USA, feb 2011).
19. Kratz, S., Rohs, M., and Essl, G. Combining acceleration and gyroscope data for motion gesture recognition using classifiers with dimensionality constraints. IUI '13, ACM (march 2013).
20. Kuehnel, C., Westermann, T., Hemmert, F., Kratz, S., Müller, A., and Moeller, S. I'm home: defining and evaluating a gesture set for smar-home control. *International Journal of Human-Computer Studies 69*, 11 (2011), 1071–5819.

21. Kulkarni, V. S., and Lokhande, S. Appearance based recognition of american sign language using gesture segmentation. *International Journal on Computer Science and Engineering 2*, 03 (2010), 560–565.

22. Lasecki, W. S., Gordon, M., Koutra, D., Jung, M. F., Dow, S. P., and Bigham, J. P. Glance: Rapidly coding behavioral video with the crowd. In *Proceedings of the 27th annual ACM symposium on User interface software and technology*, ACM (2014), 551–562.

23. LaViola Jr, J. J., and Zeleznik, R. C. A practical approach for writer-dependent symbol recognition using a writer-independent symbol recognizer. *Pattern Analysis and Machine Intelligence, IEEE Transactions on 29*, 11 (2007), 1917–1926.

24. Liu, J., Zhong, L., and Wickramasuriya, J. User evaluation of lightweight user authentication with a single tri-axis accelerometer. *Proceedings of the 11th International Conference on Human-Computer Interaction with Mobile Devices and Services* (2009), 1–10.

25. Numpy Developers. NumPy: the fundamental package for scientific computing with Python. `http://www.numpy.org/`, January 2016.

26. Ouyang, T., and Li, Y. Bootstrapping personal gesture shortcuts with the wisdom of the crowd and handwriting recognition. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ACM (2012), 2895–2904.

27. Ruiz, J., and Li, Y. DoubleFlip: a motion gesture delimiter for mobile interaction. In *Proceedings of the 2011 annual conference on Human factors in computing systems*, ACM (2011), 2717–2720.

28. Sakoe, H., and Chiba, S. Dynamic programming algorithm optimization for spoken word recognition. *Acoustics, Speech and Signal Processing, IEEE Transactions on 26*, 1 (1978), 43–49.

29. Schloemer, T., Poppinga, B., Henze, N., and Boll, S. Gesture recognition with a Wii controller. In *Proc. TEI '08*, ACM (New York, NY, USA, 2008), 11–14.

30. Schwarz, J., Mankoff, J., and Hudson, S. E. An architecture for generating interactive feedback in probabilistic user interfaces. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, ACM (2015), 2545–2554.

31. Spiro, I., Taylor, G., Williams, G., and Bregler, C. Hands by hand: Crowd-sourced motion tracking for gesture annotation. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2010 IEEE Computer Society Conference on*, IEEE (2010), 17–24.

32. Sven Kratz and Jason Wiese. Help page for GestureSeg's crowd worker user interface. `http://bit.ly/24woVLV`, May 2016.

33. The University of Waikato. Weka 3: Data Mining Software in Java. `http://www.cs.waikato.ac.nz/ml/weka/`, January 2016.

34. Werapan, W., and Chotikakamthorn, N. Improved dynamic gesture segmentation for thai sign language translation. In *Signal Processing, 2004. Proceedings. ICSP'04. 2004 7th International Conference on*, vol. 2, IEEE (2004), 1463–1466.

35. Wu, D., and Shao, L. Deep dynamic neural networks for gesture segmentation and recognition. In *Computer Vision-ECCV 2014 Workshops*, Springer (2014), 552–571.

36. Xu, R., Zhou, S., and Li, W. J. Mems accelerometer based nonspecific-user hand gesture recognition. *Sensors Journal, IEEE 12*, 5 (2012), 1166–1173.

37. YEI Technology. *3-Space Sensor User's Manual*. `http://www.yeitechnology.com/sites/default/files/YEI_TSS_Users_Manual_3.0_r1_4Nov2014.pdf`, 2014.