Chapter 7
Understanding How Non-experts Collect and Annotate Activity Data

Naomi Johnson, Michael Jones, Kevin Seppi and Lawrence Thatcher

Abstract Inexpensive, low-power sensors and microcontrollers are widely available along with tutorials about how to use them in systems that sense the world around them. Despite this progress, it remains difficult for non-experts to design and implement event recognizers that find events in raw sensor data streams. Such a recognizer might identify specific events, such as gestures, from accelerometer or gyroscope data and be used to build an interactive system. While it is possible to use machine learning to learn event recognizers from labeled examples in sensor data streams, non-experts find it difficult to label events using sensor data alone. We combine sensor data and video recordings of example events to create a better interface for labeling examples. Non-expert users were able to collect video and sensor data and then quickly and accurately label example events using the video and sensor data together. We include 3 example systems based on event recognizers that were trained from examples labeled using this process.

N. Johnson (✉)
University of Virginia, Charlottesville, VA 22903, USA
e-mail: snj3k@virginia.edu

M. Jones · K. Seppi · L. Thatcher
Brigham Young University, Provo, UT 84604, USA
e-mail: jones@cs.byu.edu

K. Seppi
e-mail: k@byu.edu

L. Thatcher
e-mail: lwthatcher@msn.com

N. Kawaguchi et al. (eds.), *Human Activity Sensing*,
Springer Series in Adaptive Environments,
https://doi.org/10.1007/978-3-030-13001-5_7

7.1 Introduction

Small sensors on printed circuit boards (PCBs), such as accelerometers or gyroscopes, are widely available and inexpensive. Many tutorials[1] exist for connecting these small sensors to computing devices like the Arduino[2] or the Raspberry Pi.[3] Using these tutorials, it is not difficult for many people to connect a sensor to a device and to watch the data stream in real time through a console window.

Unfortunately, it remains difficult to write programs that find meaningful events in the data stream. Finding meaningful events requires a person to identify meaningful patterns in the raw data and then to write a program that identifies those patterns. This task is particularly difficult for non-technical people who have little to no experience working with time-varying signals and programming.

Building event detectors for streaming sensor data is important because it can enable the creation of interesting interactive systems. As computing moves from the desktop to the pocket, and now to smaller single purpose devices, event detectors enable devices that respond to events in the world around them. Several physical interactive devices that recognize events in real time from sensor data have recently appeared in the literature (Laput et al. 2015; Zhang and Harrison 2015; Jin et al. 2015).

While many people have ideas for these kinds of physical interactive devices, they may lack the technical skills needed to design and implement the event detector. Reducing the time and effort needed to build an event recognizer will allow faster iterations over ideas for interactive systems.

One way to create an event detector for an interactive system is to use machine learning to train a classifier from example events labeled in data. The gathering and annotation of data for this purpose can be done with tools such as ANVIL, ELAN or ChronoViz; in this chapter we use a "video annotation tool" (VAT) that we created. VAT shows the user video of example events synchronized with the sensor data. In this process, the user records video and sensor data for the example events and then labels the video and sensor data together. The learner takes the labels and the sensor data as input to learn a classifier that operates on the sensor data alone. The video is captured only to help the user during labeling. We also created a data logger on a PCB which we used to collect all sensor data used in this chapter.

Figure 7.1 illustrates the labeling process in the context of a scenario involving an LED turn signal system for cyclists. The system converts bicycle hand signals into LED signals on the cyclist's backpack. First, the designer recruits a friend to make example hand signals while riding a bike. The designer attaches a sensor to the friend's hand and films the friend making example signals. Next, the designer imports the sensor data and the video into our system. The designer defines a set of labels for the events such as "left," "right," and "stop."

[1] See http://learn.adafruit.com/ or https://learn.sparkfun.com/.

[2] http://www.arduino.cc.

[3] http://www.raspberrypi.org.

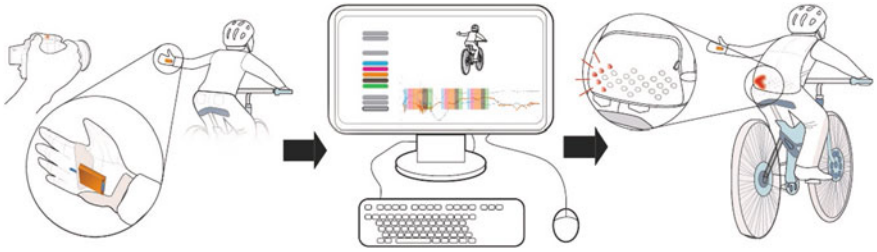

Fig. 7.1 Using VAT to build an event recognizer for bicycle hand signals by labeling example video

Both the video and the sensor data are visible during the labeling process which is shown in the center of Fig. 7.1. The designer can use the video and sensor data to quickly locate and confirm events. The video and data are always kept synchronized so that the current video frame shows what happened for a given sensor value and vice versa. As the video plays, the event stream moves to remain synchronized with the video.

After a while, the designer learns to recognize patterns in the data that correspond to events and can click directly on the next event in the data stream to skip through the video. The video confirms that what the designer thought was an event based on the data is actually an event.

After labeling, the designer passes the label set and the data stream to a machine learning algorithm which learns an event recognizer. The recognizer can then be incorporated into a working prototype of the system as shown on the right in Fig. 7.1.

These steps outlined above were discussed in a previous paper with the same title (Jones et al. 2018). This chapter extends our prior work in three ways: first, by providing more details about how novices can build an event recognizer; second, by providing an in-depth discussion of how to adapt the GMM-HMM approach for event recognition; and third, expanded results that include a study where we have evaluated the use of video and data together for non-experts to label events in the context of 4 studies involving 3 different applications with 74 participants.

7.2 Related Work

7.2.1 Interactive Physical Devices

Building event recognizers fills an important gap in the construction of physical interactive devices. Prior work addresses the physical shape of the object (Savage et al. 2015a; Hudson and Mankoff 2006), and adding functional physical widgets to the object (Savage et al. 2015b; Harrison and Hudson 2009), data collection and event recognition (Jones et al. 2016, 2018). We do not address the design or construction

of the physical interactive devices, but we do address the problem of enabling such devices to respond to events detected by sensor data.

Hartmann et al. (2006, 2007) explore demonstration-based programming for creating input event recognizers from sensors. Designers create an input event recognizer by performing example inputs and then annotating the generated sensor signals directly in the tool. Either thresholding or a pattern matcher based on dynamic time warping (Sakoe and Chiba 1978) recognizes the events based on the labeled example.

Our approach extends this work by presenting video and sensor data to the designer, supporting use outside of the laboratory environment, and by using a different learning algorithm to create an event recognizer. Presenting video and data during labeling decouples example generation from event labeling and allows the system to be used when not connected to a workstation.

7.2.2 Event Recognizers and Interaction

Machine learning is becoming a widely used approach to learning classifiers for sensor input. Several recent papers (see Laput et al. 2015; Zhang and Harrison 2015; Jin et al. 2015) develop specific interactive systems by learning to recognize specific events or conditions in sensor data. The work that we present in this chapter is different in that we envision a general purpose system for developing event recognizers, not one-off creation of recognizers for specific applications.

7.2.3 Hidden Markov Models, ASR and Other Activity Models

HMMs and other sequence models have been applied to activity recognition (Bulling et al. 2014; Patterson et al. 2005) but we employed signal processing for physical computing devices that most closely resembles *automatic speech recognition* (ASR). The translation of acoustic signal into discrete words closely parallels the conversion of motion sensor readings into discrete real-world physical events. One common approach to the ASR problem is to train a *Gaussian-mixture model based hidden Markov model* (GMM-HMM) with the Baum-Welch algorithm and to label recordings using the Viterbi algorithm (Jurafsky and Martin 2009). Fortunately, the strong assumptions undergirding this generative model let it rapidly fit labeled training data (Murphy 2012), making it a good choice for the back end of a fast and focused tool for interactive machine learning. This approach to activity recognition can be seen as a combination of activity recognition based on HMM's (Patterson et al. 2005) and and GMMs (Plötz et al. 2011).

7.3 Building an Event Recognizer with VAT

Building an event recognizer with VAT consists of the following six steps:

1. Define events and split the events into pieces.
2. Attach the data logger to the person or object performing the events.
3. Record video and data of example events.
4. Label the events and pieces of events in the data using the video as a guide.
5. Run the learner on the labels and data.
6. Deploy the learned event recognizer.

We describe the process using a running example in which a designer wants to build a system that recognizes bicycle hand signal events as shown in Fig. 7.1.

The event recognizer discriminates between the different kinds of hand signals so that the correct LED pattern is shown. For example, if the cyclist signals "left turn" by holding their left arm straight out to the left, the system flashes the left turn arrow on the cyclist's back.

7.3.1 Define Event Pieces

Events are decomposed into pieces. A piece of an event is a simple motion that is contained in every instance of an event. The designer might divide the events in the bicycle signal example into two pieces: raise and lower. The "raise" piece represents moving the hand from the bicycle handle bar to the signal position and the "lower" piece represents moving the hand from the signal position back to the handle bar.

The left turn, right turn and stop events all consist of the same pieces (lower and raise) in the same order but the exact motion is different in each case. The learner can distinguish between events represented by the same sequence of pieces as long as the actual motions represented by each event are different.

7.3.2 Attach Data Logger

The data logger is attached to the person or object that will perform the actions so that the data logger can record acceleration and velocity. The data logger should be placed where it will move in ways that are significant for recognizing the event. Figure 7.2 shows the data logger attached to a bicycling glove. Placing the data logger on the back on the hand captures the large motions associated with each signal event. Placing the data logger on the upper arm would not work as well because the upper arm experiences less motion in each of the signals.

Fig. 7.2 The data logger attached to a bike glove. The data logger is secured in the orange housing

The data logger records acceleration and rotation in 3 axes each.[4] The data are either written to an SD card or broadcast over a wireless link to a receiver. The sampling rates for acceleration and rotational velocity can be adjusted independently using a configuration file. In most cases, we sample acceleration and rotation each at 25.6 samples per second. The data logger will support sampling rates upto 102.4 samples per second for the accelerometer when writing to the local SD card. Other sensors could be used with VAT as long as the data sensed results in some visible or audible change that can be captured on video.

7.3.3 Record and Synchronize Video and Data

The user records video and data of the events. We have developed a process to simplify synchronization of video and data upto the video camera framerate.

The synchronize the video and data, the user captures a red flash generated by the data logger ten seconds after the data logger is turned on. The data logger does not need to be visible after the first red flash is captured. The data logger logs the time of the first red flash (in milliseconds elapsed since the processor was turned on). At the beginning of labeling in VAT, the user locates and marks the first video frame that contains the first red flash. VAT can then synchronize the video and data for as long as both continue recording. A red line over the sensor data display indicates the

[4]Using an Invensense MPU-9250.

data that corresponds to the currently displayed video frame. If either is stopped or turned off, synchronization must be repeated.

The accuracy of this approach is limited by the video camera frame capture rate. For the examples used in this chapter, we captured video at 30 frames per second which means that synchronization of the video and data is off by at most 0.033 of a second. Cameras with higher frame rates can be used to synchronize the data with greater accuracy (up to the temporal resolution of the data logger sampling rate). But for human generated events considered in this chapter, 0.033 of a second proved sufficient.

7.3.4 Label Events in VAT

Figure 7.3 shows the use of VAT for labeling. After marking the frame containing the first red flash, the user is presented with the video, the data stream, and a list of buttons. The buttons are used to label events and pieces. Video playback can be controlled using the space bar to toggle play and pause and the arrow keys to advance a frame forward or backward.

The data stream display is synchronized with video playback so that the red vertical line in the data stream display is always over the data recorded during the current video frame. The data stream is also interactive. Clicking in the data stream advances video playback to the corresponding location in the video; scrolling the mouse wheel zooms in and out of the data stream time scale. The numbers beneath

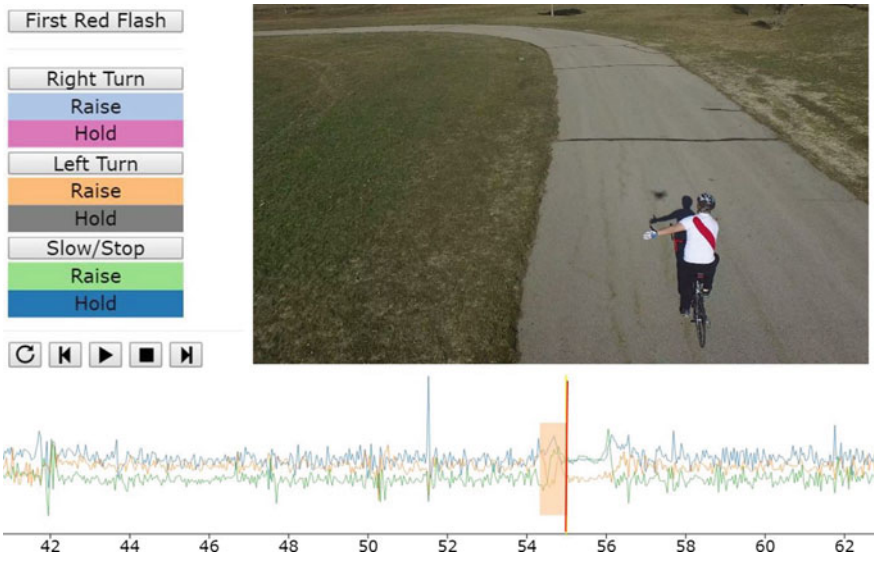

Fig. 7.3 Labeling bicycle hand signals in VAT

the data stream display represent the number of seconds since the data logger was turned on.

In Fig. 7.3, the user is in the process of labeling the "raise" piece of a left turn event. The orange highlight in the data display window marks the region of data currently labeled "raise." The highlight color corresponds to the color of the label shown under the "Left Turn" button.

In the frame of video shown in Fig. 7.3, the rider has extended his arm to the left and will hold it stationary for about one second (from second 55 through second 56) and then put his hand back on the bicycle handlebars. These actions can be seen in the data stream. The acceleration to the left of the red playback line, which is highlighted in orange, corresponds to the rider lifting and straightening his arm. The region to the the right of the red playback line has small acceleration values due to rider holding his arm stationary and extended. After the rider places his hand back on the handlebars, the data logger again begins to record high frequency acceleration due to vibrations transmitted from the bike to the rider's hand.

After assigning labels while viewing both the video and the data, most users begin to recognize patterns in the data associated with events. At that point, the user can quickly click through the data stream to identify likely events and use the video to confirm each event and carefully locate the boundaries between event pieces.

7.3.5 Train Recognizer

The user trains an event recognizer by passing the event definitions, labels and data values to VAT. The event definitions are the events and subevents of interest. The labels are a list of 4-tuples generated as described above and shown in Fig. 7.3. Each 4-tuple contains two timestamps, the event name, and the event piece name. The timestamps mark the beginning and end of the event piece.

7.4 Learning an Interactive Event Recognizer

The learner in VAT learns an event recognizer from the labeled data stream generated by the user; the video was only used to guide the user and is not used as input for the learner. The learner is based on the GMM-HMMs commonly used in speech recognition. The implementation used in VAT is closely related to the mechanisms found in the Hidden Markov Model Toolkit (HTK) (Young et al. 1997).

7.4.1 GMM-HMM Approach for Speech Recognition

In ASR, the recognizer identifies words and word boundaries as they occur in an audio signal. This matches the problem of event recognition using the data logger because events are sensed as a time-varying signal. ASR algorithms are also designed to be robust to variations in the speed and way words are spoken by different people. Similar robustness is needed to recognize events performed by different people.

Speech is often represented using a hidden Markov model, with the hidden nodes representing parts of phonemes and the observed nodes representing segments of the discretized acoustic signal. Typically each phoneme is represented with a sequence of three of hidden nodes with each node representing the beginning, middle, and end of the phoneme (Jurafsky and Martin 2009). The observed nodes can be implemented using the (logarithm of the) of the product of Gaussian mixture models (GMMs). Using a GMM-HMM is convenient because the hidden node transition model can be learned by Baum-Welch and the GMMs can be learned by k-means clustering (Young et al. 1997, p. 17); the resultant model can then be used to decode spoken words by the Viterbi algorithm (Young et al. 1997, p. 9).

7.4.2 Adapting the GMM-HMM Approach for Event Recognition

We adapt the structure and algorithms used in ASR to the problem of recognizing real-world events from sensor signals in VAT. To adapt these ASR algorithms to event recognition, we divide the event to be recognized into pieces in the same manner that spoken words are split into phonemes. A phoneme is a phonetic piece of a word. For example, the word cool has three phonemes with each phoneme corresponding to each spoken sound in the word. For events, each piece of the event is a specific motion that is performed as part of an event.

Figure 7.4 shows a scoop event (as might be found in scooping an ingredient for cooking) split into four pieces: a "load" in which the wrist rotates the spoon to load

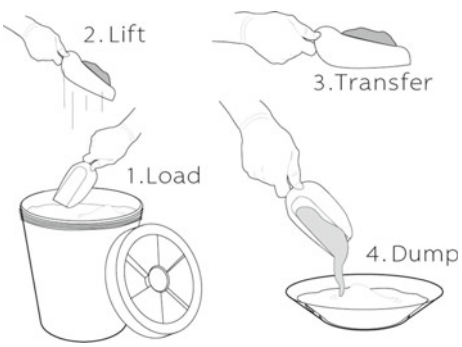

Fig. 7.4 Four pieces of a scoop event: loading, lifting, transferring and dumping

the spoon, the "lift" in which the spoon is raised out of the container, followed by a "transfer" in which the hand moves the spoon, and ending with a "dump" in which the wrist rotates again to the dump the contents from the spoon. Each scoop event contains these four pieces in that order though different people may do each piece differently. We also include a "nothing-happening" event which is applied after the user is done labeling events to mark time periods in which the user did not label any event.

Just as phonemes can occur in different words, motions can also appear in different contexts and with different HMM transition probabilities. In the scooping example, the "transfer" motion might also appear as a piece of the "wash spoon event."

We adapt GMM-HMMs to the problem of event recognition by modifying the process for learning transition probabilities, handling data streams with different sampling rates, and adding an event length threshold. Each adaptation is discussed below.

Learning Transition Probabilities

ASR researchers often represent each phoneme with three hidden nodes; similarly, we represent each piece of an event with three hidden nodes. For the scooping example, this means that each of the four pieces of the scoop event (load, lift, transfer, and dump) are each represented by three hidden nodes; thus, including the hidden node for the "nothing-happening" event, we have a total of $(4 \times 3) + 1 = 13$ hidden nodes. In general, the HMM for an event has $(n \times 3) + 1$ hidden nodes, where n is the number of event pieces.

Recall that a transition probability in an HMM is the probability of moving from one hidden node to another. Because the user does not observe the transitions between hidden nodes, the transition probabilities between the three hidden nodes for a single event piece are learned by uniform segmentation and then Viterbi segmentation as in the HTK (Young et al. 1997, p. 142). The user does observe and mark transitions between event pieces, so these transitions are learned directly from the labels. Combining the transition probabilities within event pieces with the transition probabilities between event pieces gives a full hidden-node transition model without using the Baum-Welch algorithm on the entire sequence.

Handling Different Sampling Rates

Since the data logger records data at different rates, our learning algorithm needs to learn without data from one sensor or the other at some time steps, and the learned model must be able to decode such mixed-sampling-rate input data.

Our approach to handling this mixed-sampling-rate data builds upon the techniques of the HTK. The HTK allows each observation vector to be split into independent streams. Each stream is modeled with a GMM and the weighted product of

these stream GMMs is the observation probability density for a hidden node (Young et al. 1997, p. 6).

Our approach also departs from the HTK approach in the selection of the number of mixture components. Like some ASR researchers (Chen and Gopalakrishnan 1998), we select the number of mixture components in the GMM by optimizing the Bayesian information criterion (BIC), rather than using a fixed user-specified number of mixture components. We use this BIC-optimization approach because our users are likely unfamiliar with GMMs and the ideal number of mixture components varies, with some dependency on the number of training examples. To limit the training time, we cap the number of mixture components at three.

Event Length Threshold

Given the transition probability model and the observation probability models both learned as described above, the Viterbi algorithm can be used to classify events in the application. However, the Viterbi algorithm does not consider the duration or length of the event. This means that the algorithm occasionally incorrectly classifies a very short sequence of data as an event. For example, in the scoop data the algorithm might incorrectly label a sequence of noise in the data with duration 0.3 s as a scoop event. Human generated scoop events do not happen at that time scale.

We set the event length threshold to the duration of the shortest labeled event in the training data minus two standard deviations. We also set the minimum event piece length threshold to 0.05 s because motion at that time scale can not be reliably detected in data captured at less than 20 samples per second. Video captured by a high speed camera, and data captured at a higher sampling rate, may support event pieces with shorter durations.

7.5 Results

We have validated the utility of our system in four ways: first we consider the consistency and accuracy of labels generated by non-experts using VAT second we assess the relative quality of machine learning models, or event recognizers, obtained using labels generated in VAT by non-experts, third we explore the effect of using video during labeling in VAT.

We have also built three functioning systems using event recognizers learned from labels obtained from VAT.

7.5.1 Consistent and Accurate Labeling

To validate the consistency of the labels obtain via VAT we conducted a user study in which participants were shown video of a person walking with a cane and data collected by the data logger. The data logger was attached to the cane. The participants were asked to label a "step" event that was broken into two pieces: (1) the part of the step where the cane tip is in contact with the ground and (2) the part of the step where the cane is not in contact with the ground. The video included 9 steps to be annotated over less than 4 min. The video and data include motion other than walking in addition to the steps to label.

For this data, the annotator agreement is high both when the annotators are compared to each other as a group and when annotators are each compared to a ground truth set of labels.

Ten participants (5 male, 5 female) were recruited from an introductory programming class. All were full-time college students. Although they were recruited from an introductory programming class we selected only students with non-STEM majors. Participants received $30 US for their participation.

We then evaluated these annotations using Krippendorff's alpha (henceforth "α") coefficient, a common statistical measure of annotator agreement (Krippendorff 2012) (We also experimented with the more limited Cohen's Kappa statistic (Cohen 1960), similar results are obtained when it is used). α is typically used in content analysis where textual units are categorized by annotators, which is very similar to the labeling of event pieces we have asked our annotators to do. α is well suited to the evaluation of raw annotations each annotation being a single label for each sample (each time step in our case). The coefficient is a single scalar value which is insensitive to the numbers of annotators, small sample sizes, unequal sample sizes and missing data. The α for our group of annotators is 0.95, which is quite high: values above 0.80 are considered reliable (Krippendorff 2012).

In addition to the study subjects, a team of three researchers annotated the data using the a-b-arbitrate (Hansen et al. 2013) approach to create a gold standard annotation. In the a-b-arbitrate approach, annotators "a" and "b" make sections and the arbitrator breaks ties if needed. In our case there was only one tie which had to be broken by the arbitrator. We then computed the pair-wise α for each study participant and the gold-standard data. The average of the pair-wise α coefficients is 0.935 with standard deviation 0.073.

7.5.2 Quality of Machine Learning Models

In addition to labeling the cane data, we also asked each of the 10 participants to label "scoop" data as shown earlier in Fig. 7.4. The design of the scoop counter is similar to other systems in which accelerometers attached to cooking utensils are used to detect events (Plötz et al. 2011; Pham and Olivier 2009). In this part of the

study, participants were asked to collect the data, including the video, and to label scoop events in the data they had collected.

The purpose of this part of the study was to measure the quality of the event recognizers learned from data and labels generated by non-experts. Because participants each labeled their own video and data, we can not compute α to measure annotator reliability for this group.

We could have measured α for the learned recognizer but we instead use a metric more suited for assessing the recognizer as part of an interactive system. In our intended application involving construction of interactive physical objects, what matters most is identifying events quickly enough to build interactive systems. In that context, a delay of 0.5 s or less may be sufficient.

Our objective is for the trained learner to infer the completion of an event or piece of an event 0.5 s early or 0.5 s late relative to actual occurrence of the event (henceforth "β"). We use an a-b-arbitrate process to identify the actual event boundaries.

We computed an F1 metric using β by counting true positives, false negatives, and false positives as follows: An event that is inferred within 0.5 s of a true event it is a true positive, an event that is inferred by the learner near no true event or near a true event that has already been accounted for by some other, closer, inferred event is a false positive, and a true event that occurs but no inferred event is within 0.5 s of it is a false negative.

In our study of the quality of machine learned models for recognizing scoop events, we collected 12 events and annotations from each participant, used 9 as training data and 3 as test data. Event recognizers learned from the participants' data and labels achieved an average precision of 0.91 (sd = 0.15), average recall of 1 (sd = 0.0) and average F1 of 0.92 (sd = 0.93).

Figure 7.5 illustrates the sensitivity of the learner to the number of samples used as training data. The vertical axis shows the average precision, recall and F1 score for all 10 participants and the horizontal axis shows the number of labeled events used as input. For this data set, learning slows at 6 samples suggesting that the payoff associated with labeling more than 6 samples is minimal.

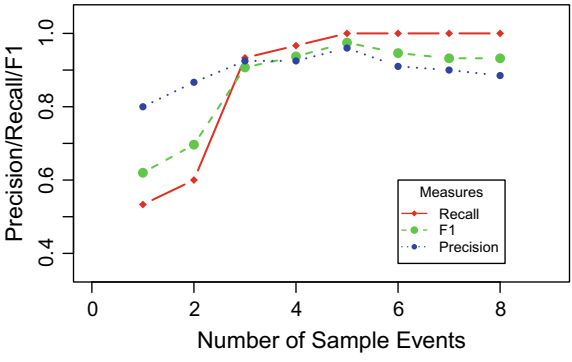

Fig. 7.5 Average learner performance as a function of the number of events labeled. Labeling more than 5 or 6 events produces little improvement in the learned recognizer. The learner was trained and evaluated on data from a single participant at a time

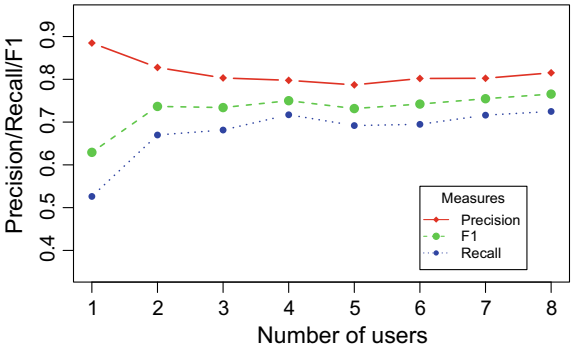

Fig. 7.6 Average learner performance on scoop events generated by three people who's data was not included in the training set as a function of the number of participants' data used in training. Learning a recognizer from data collected by 2 people results in a more general recognizer

Figure 7.5 does not show how the learned recognizer generalizes to other scoops generated by other people—the graph only shows the performance of the learner on data from a single user as labeled by that user.

Figure 7.6 shows how the learner generalizes on this data. The vertical axis in Fig. 7.6 shows the average performance of the learner on scoop data generated by three people who did not participate in the study. As before, ground truth event labels for these three samples were generated using a-b-arbitrate. The horizontal axis shows the number of participants' data used to learn the model. For example, "2" on the horizontal axis means that we combined the scoop data from 2 participants to train a recognizer and then used that recognizer on the amalgamated scoop data.

Collecting data from 2 people lead to a more general event recognizer than collecting data from 1 person, as expected. Collecting data from more than 2 people led to slight improvements in generality on this data set. These results indicate that, for scooping, a person would need to recruit two people to get a reasonably general recognizer but that recruiting more will lead to increasingly general recognizers.

7.5.3 Video Modality

In the study involving 10 participants labeling data related to walking with a cane and scooping a cooking ingredient, we observed that different participants used the video and sensor data differently; most ignored it but some relied on it. We conducted a larger study to explore the role of video and sensor data in event labeling.

In this larger study we asked participants to label data related to taking pills out of a bottle. We attached a data logger to a pill bottle and recorded a person taking the bottle off a shelf, removing a "pill" (we used candy rather than actual medication), taking the "pill" and putting the bottle back on the shelf. We included some events in which the pill bottle was moved but in which no pills were taken. This application is motivated by a health monitoring system that might remind a person to take a daily medication and might notify caregivers that a person has taken their medication.

Sensor data collected by the data logger for this example excludes flat regions in which sensor values are constant over time when the pill bottle is stationary between pill taking events. Most participants could easily identify the beginning and end of the pill taking events without the video. However, identifying pieces of events without the video proved more difficult.

The VAT interface presents two kinds of information to the user for use in labeling as shown earlier in Fig. 7.3. The two kinds of information for labeling are data and video. The data is the output of the data logger plotted against time and the video is the video captured by the camera.

We showed participants different information during labeling in order to understand the impact of the different elements on labeling. Each participant completed two labeling sessions during their visit. In each session they labeled a different collection of 5 pill-taking events and 2 just-move events. In each session they were shown just the data, just the video or both. To minimize the impact of a learning effect, we counterbalanced the order in which participants saw information for labeling. The participants were divided into 4 groups as follows:

1. 15 participants saw only data followed by both data and video,
2. 15 participants saw video and data followed by only data,
3. 15 participants saw only video followed by both data and video, and
3. 15 participants saw both data and video followed by only video.

We did not compare labeling using only video to labeling using only data because we anticipated that labeling using only video would be similar to labeling using both video and data. Our results support that anticipation. We measured the time taken to complete the labeling takes and calculated Kripendorff's α (Krippendorff 2012) for the participants' labels as described above. We also administered a short survey after the labeling session.

We examined the annotation using two-way ANOVA to look for affects involving the interface modality (users using video only, data only or both) and to check for a learning affect. The interface mode has a significant effect ($Pr(>F) = 4.2e{-}06$), but neither the learning effect ($Pr(>F) = 0.28$) nor the interaction of modality and learning had ($Pr(>F) = 0.57$) have a significant affect.

Since the affect of user interface modality was statistically significant in the ANOVA, we also looked at this affect using various two-tailed t-tests as indicated below to further explore this affect as shown in Table 7.1, which shows the value of the t-statistic, the degrees of freedom used in the test and the resultant p-value. Including video in the VAT interface always leads to significant differences in annotator performance compared to including only sensor data in the interface.

When we consider the time taken by the participants there is a significant learning effect, but only for those that used data and both video and data together (in either order) as shown in Table 7.2

Note that the mean value for data only is below the level where even tentative conclusions can be drawn. The inclusion of video ("Both") brings the average α to 0.79 (Table 7.3), which is well into the range where tentative conclusions can

Table 7.1 Including video in the VAT interface has significant impact on annotator accuracy. Tests marked with (p) are paired (unpaired tests are Welch). Bold rows are significant at the Bonferroni corrected level of **0.05/4**

	t	df	p-value
Video only or Both versus Data only	**4.00**	**35.2**	**0.0003**
Data only versus Both (p)	**4.79**	**29**	**4.58e–05**
Video only versus Both (p)	1.03	29	0.310
Video only versus Data only	**3.32**	**46.1**	**0.0017**

Table 7.2 T-tests on time taken to perform labeling involving interface modality. Bold row is significant at the **0.05/3** Bonferroni corrected level

	t	df	p-value
Both first versus Both second	1.44	59	0.16
Data then Both versus Both then Data	**2.56**	**29**	**0.016**
Video then Both versus Both then V.	−0.25	29	0.80

Table 7.3 Interface modality means and standard deviations

	Mean	Std Dev
Video only and Both	0.79	0.12
Data only	0.62	0.22
Video only	0.77	0.13
Both	0.79	0.12

be made. The point where reliable conclusion can be drawn is 0.80 (Krippendorff 2012).

Recall that the purpose of our approach is to obtain reliable learned models for recognizing certain events. In the context of this application, we seek to identify when the person has taken the pill. Finding the end of the event, that is, the point when the bottle goes from being in motion to a rest state on the shelf is easy. However in this application we would like to be able to differentiate taking a pill from merely moving the bottle around. Furthermore, we would like to identify the taking of the pill with minimum latency. Thus the most important point in time for us to identify in this application is the moment when the lid is back on, and the when we believe that the bottle is on its way back to the shelf; this is the first moment that we can believe that the entire event of pill taking is complete. For this reason we also evaluated our β metric obtained from this point in time.

As with the α data, we use two-way anova to looked for affects involving the interface modality and to check for a learning affect. As before, the interface mode has a significant effect (Pr(>F) = 1.27e–06), there is potently a learning effect

Table 7.4 Interface modality t-tests on model accuracy on a held-out test set. Tests marked with (p) are paired (unpaired tests are Welch). Bold rows are significant at the Bonferroni corrected level level of **0.05/4**

	t	df	p-value
Video only or Both versus Data only	**4.76**	**45.75**	**1.97e–05**
Data only versus Both (p)	**5.58**	**29**	**5.10e–06**
Video only versus Both (p)	1.57	29	0.128
Video only versus Data only	**2.62**	**57.57**	**0.01127**

(Pr(>F) = 0.08), but no evidence of interaction effects (Pr(>F) = 0.20). We looked for a learning affect using t-tests structured in various ways but found none.

Since the affect of user interface modality was statistically significant in the ANOVA, we again looked at this affect using various two-tailed t-tests as indicated below to further explore this affect as shown in Table 7.4.

As was the case with the annotation metric α, our results with β were not just statistically significant, they were also of practical significance. With just 5 training examples of this more complex task we obtain an average 0.21 better F1 scores on the test data using the interface with both video and data (relative to just data alone) and 0.15 better training on video alone.

After participants completed both labeling tasks, we asked them if the data or the video were more intuitive to use as part of the labeling task. Of the 30 participants who labeled using the data or both data and video (in either order), 25 said that the video was more intuitive.

7.5.4 Building Interactive Systems

We have implemented three systems as case studies: a cooking scoop counter, a medication monitor and a bicycle turn signal recognizer. In each of these, we use an event recognizer learned from data labeled in VAT along with a program that invokes the recognizer and responds to events. This program also provides the mapping between events and callback functions to the recognizer, as shown in Fig. 7.7. The recognizer uses this mapping to call the appropriate function when an event is detected. The recognizer runs repeatedly on the data it receives from the wirelessly-connected data logger, and spawns a thread for each callback when events are detected.

Cooking Scoop Counter

Using the data recorded by the ten scoop study participants discussed earlier, we trained a recognizer that recognizes when the spoon is used to scoop material out of a canister. On each scoop event, a count is incremented and announced out loud.

```
def left_turn(event):
    leds.left_turn_on(8)

def right_turn(event):
    leds.right_turn_on(8)

event_hooks = {
    'Right Turn': right_turn,
    'Left Turn': left_turn,
}

RealTimeRecognizer(expanduser('~/bike/model.pkl'),
     event_hooks=event_hooks).run()
```

Fig. 7.7 A shortened example of the code for the hand signals

Medication Reminder

This system turns on a sign reminding a patient to take their pills and waits for a smart pill bottle to notice if it has been opened and closed (based on movement data), or only picked up and moved. On detecting the event, the application turns off the sign and sends a text message to the caretaker. To build this system, we attached a data logger to the side of an empty pill bottle, filled the bottle with candy, and collected sample data with three different users.

In order to understand the amount of time required to collect and label data for the medication reminder, we recruited 4 university students with design backgrounds (3 male, 1 female) to collect the data required to train the event recognizer while recording the time required to complete each step. Participants each required between 59 and 80 min, with an average of 71 min, to collect and label the video and data. A little more than half of that time, 39 min, was spent labeling scoop events in the data.

Bicycle Hand Signal

Using VAT, we made a bike riding accessory that interprets a biker's hand signals and displays the corresponding turn signal on the rider's back. Data and video were captured outdoors in a park using a quadcopter to record the video. The quadcopter was flown behind the rider to allow the video camera to capture hand signals made by the rider. A still frame from the video is shown in Fig. 7.3 (the quadcopter shadow is above the rider's left arm). The data logger was attached to a glove worn on the rider's left hand as shown in Fig. 7.2.

The turn signal event recognizer was learned from 20 labeled events including left turns, right turns, and stopping. The LEDs and a laptop are mounted in a backpack

that the rider wears. The rider can make normal biking hand signals while wearing the glove, and the application will automatically detect and display the matching signals on the LED. Figure 7.7 shows an example of the code used to integrate with the recognizer, note that the complexity of the learning algorithm is entirely hidden.

7.6 Conclusion

Adding synchronized video to sensor data for labeling creates a more intuitive labeling experience and supports increased precision in labeling. This approach allows designers to quickly create machine learning models for physical interactive devices which must recognize events in the physical world. We have run several user studies to document behaviour of non-expert users in the context of annotating physical event and to demonstrate the advantages of our approach. More importantly, we demonstrated the value of this approach by building three example systems using event recognizers learned from labeled examples using our process: a cooking scoop counter, a medication monitor and a gesture-based bike turn signal system.

Future work might include other means for supporting the annotation process, for example pre-annotation. In addition the learning algorithm could be replaced or modified to improve the efficiency and performance of the learner.

Acknowledgements This work supported by NSF grant IIS-1406578.

References

Bulling A, Blanke U, Schiele B (2014) A tutorial on human activity recognition using body-worn inertial sensors. ACM Comput Surv (CSUR) 46(3):33

Chen SS, Gopalakrishnan PS (1998) Clustering via the Bayesian information criterion with applications in speech recognition. In: Proceedings of the 1998 IEEE international conference on acoustics, speech and signal processing, 1998, vol 2. IEEE, pp 645–648

Cohen J (1960) A coefficient of agreement for nominal scales. Educ Psychol Meas 20:37–46

Hansen DL, Schone PJ, Corey D, Reid M, Gehring J (2013) Quality control mechanisms for crowdsourcing: peer review, arbitration, & expertise at familysearch indexing. In: Proceedings of the 2013 conference on computer supported cooperative work. ACM, pp 649–660

Harrison C, Hudson SE (2009) Providing dynamically changeable physical buttons on a visual display. In: Proceedings of the SIGCHI conference on human factors in computing systems, CHI '09. ACM, New York, pp 299–308. https://doi.org/10.1145/1518701.1518749

Hartmann B, Klemmer SR, Bernstein M, Abdulla L, Burr B, Robinson-Mosher A, Gee J (2006) Reflective physical prototyping through integrated design, test, and analysis. In: Proceedings of the 19th annual ACM symposium on User interface software and technology, UIST '06. ACM, New York, pp 299–308. https://doi.org/10.1145/1166253.1166300

Hartmann B, Abdulla L, Mittal M, Klemmer SR (2007) Authoring sensor-based interactions by demonstration with direct manipulation and pattern recognition. In: Proceedings of the SIGCHI conference on human factors in computing systems, CHI '07. ACM, New York, pp 145–154 (2007). https://doi.org/10.1145/1240624.1240646

Hudson SE, Mankoff J (2006) Rapid construction of functioning physical interfaces from cardboard, thumbtacks, tin foil and masking tape. In: Proceedings of the 19th annual ACM symposium on

user interface software and technology, UIST '06. ACM, New York, pp 289–298. https://doi.org/10.1145/1166253.1166299

Jin H, Xu C, Lyons K (2015) Corona: positioning adjacent device with asymmetric bluetooth low energy RSSI distributions. In: Proceedings of the 28th annual ACM symposium on user interface software & technology, UIST '15. ACM, New York, pp 175–179

Jones M, Walker C, Anderson Z, Thatcher L (2016) Automatic detection of alpine ski turns in sensor data. In: Proceedings of the 2016 ACM international joint conference on pervasive and ubiquitous computing: adjunct, UbiComp '16. ACM, New York, pp 856–860. https://doi.org/10.1145/2968219.2968535

Jones MD, Johnson N, Seppi K, Thatcher L (2018) Understanding how non-experts collect and annotate activity data. In: Proceedings of the 2018 ACM international joint conference and 2018 international symposium on pervasive and ubiquitous computing and wearable computers, UbiComp '18. ACM, New York, pp 1424–1433. https://doi.org/10.1145/3267305.3267507

Jurafsky D, Martin J (2009) Speech and language processing: an introduction to natural language processing, computational linguistics, and speech recognition. In: Prentice Hall series in artificial intelligence. Pearson Prentice Hall

Krippendorff K (2012) Content analysis: an introduction to its methodology. Sage

Laput G, Yang C, Xiao R, Sample A, Harrison C (2015) EM-Sense: touch recognition of uninstrumented, electrical and electromechanical objects. In: Proceedings of the 28th annual ACM symposium on user interface software & technology, UIST '15. ACM, New York, pp 157–166. https://doi.org/10.1145/2807442.2807481

Murphy KP (2012) Machine learning: a probabilistic perspective. In: Adaptive computation and machine learning series. MIT Press

Patterson DJ, Fox D, Kautz H, Philipose M (2005) Fine-grained activity recognition by aggregating abstract object usage. In: Proceedings of the ninth IEEE international symposium on wearable computers, 2005. IEEE, pp 44–51

Pham C, Olivier P (2009) Ambient intelligence: European conference, AmI 2009, Salzburg, Austria, November 18–21, 2009. Proceedings, chap. Slice&Dice: recognizing food preparation activities using embedded accelerometers. Springer, Berlin, Heidelberg, pp 34–43. https://doi.org/10.1007/978-3-642-05408-2_4

Plötz T, Moynihan P, Pham C, Olivier P (2011) Activity recognition and healthier food preparation. In: Activity recognition in pervasive intelligent environments. Springer, pp 313–329

Sakoe H, Chiba S (1978) Dynamic programming algorithm optimization for spoken word recognition. IEEE Trans Acoust Speech Signal Process 26(1):43–49. https://doi.org/10.1109/TASSP.1978.1163055

Savage V, Follmer S, Li J, Hartmann B (2015a) Makers' marks: physical markup for designing and fabricating functional objects. In: Proceedings of the 28th annual ACM symposium on user interface software & technology. ACM, pp 103–108

Savage V, Head A, Hartmann B, Goldman DB, Mysore G, Li W (2015b) Lamello: passive acoustic sensing for tangible input components. In: Proceedings of the 33rd annual ACM conference on human factors in computing systems, CHI '15. ACM, New York, pp 1277–1280

Young S, Evermann G, Gales M, Hain T, Kershaw D, Liu X, Moore G, Odell J, Ollason D, Povey D et al (1997) The HTK book, vol 2. Entropic Cambridge Research Laboratory Cambridge

Zhang Y, Harrison C (2015) Tomo: wearable, low-cost electrical impedance tomography for hand gesture recognition. In: Proceedings of the 28th annual ACM symposium on user interface software & technology, UIST '15. ACM, New York, pp 167–173. https://doi.org/10.1145/2807442.2807480