A Multi-Media Exchange Format for Time-Series Dataset Curation

Philipp M. Scholl
Albert-Ludwigs-University
Freiburg, 79110, Germany
pscholl@ese.uni-freiburg.de

Kristof Van Laerhoven
Albert-Ludwigs-University
Freiburg, 79110, Germany
kristof@ese.uni-freiburg.de

Ubicomp/ISWC'16 Adjunct , September 12 - 16, 2016, Heidelberg, Germany

ACM 978-1-4503-4462-3/16/09\205 $15.00
DOI: http://dx.doi.org/10.1145/2968219.2968278

Abstract

Exchanging data as comma-separated values (CSV) is slow, cumbersome and error-prone. Especially for time-series data, which is common in Activity Recognition, synchronizing several independently recorded sensors is challenging. Adding second level evidence, like video recordings from multiple angles and time-coded annotations, further complicates the matter of curating such data.

A possible alternative is to make use of standardized multi-media formats. Sensor data can be encoded in audio format, and time-coded information, like annotations, as subtitles. Video data can be added easily. All this media can be merged into a single container file, which makes the issue of synchronization explicit. The incurred performance overhead by this encoding is shown to be negligible and compression can be applied to optimize storage and transmission overhead.

Author Keywords

Data Curation; Activity Recognition; Multi-Media Format; Data Storage; Comma-Separated-Values

ACM Classification Keywords

E.5 [Data]: Files

Examples of Activity Recognition Datasets

HASC Challenge [8]: >100 subjects, time-coded CSV files.

Box/Place-Lab [6]: A sensor-rich home, in which subjects are monitored for long terms. Data is available in time-coded CSV files.

Opportunity [12]: 12 subjects were recorded with 72 on- and off-body sensors in an Activities of Daily Living (ADL) setting. Multiple video cameras were used for post-hoc annotations. Data is published in synchronized CSV files.

Kasteren's Home [7]: 12 sensors in 3 houses. Data is stored in matlab files.

Borazio's Sleep [1]: 1 sensor, 42 subjects. Data is stored in numpy's native format.

Freiburg Longitudinal [9]: 1 sensor, 1 subject, 4 weeks of continuous recording. Data is stored in numpy's native format.

Introduction

At the heart of each Activity Recognition task is a dataset. This dataset might be formed from multiple media streams, like video, audio, motion and other sensor data. Recorded at different rates, sparsely or uniformly sampled and with different numerical ranges, these streams are challenging to process and store. Commonly, datasets are published in multiple comma-separated values (CSV) files, either with a constant rate or time-coded. For small, independent time-series this is a worthwhile approach, mostly due to its simplicity and universality. However, when observing with multiple independent sensors, synchronization quickly becomes a challenge. Different rate recordings have to be resampled, time-coded files have to be merged. Storing such data in several (time-coded) CSV files hides this issue, until the dataset is going to be used. Furthermore parsing CSV files incurs a large performance and storage overhead, compared to a binary format.

An alternative approach is to store time-series in existing multi-media formats. Encoding all multi-media data in one file allows to merge streams, to synchronize them and to store (meta-)data in a standardized format. In the next section, we will first look at the formats commonly used to exchange data in Activity Recognition, afterwards detail a multi-media format and finally evaluate the incurred performance and storage overhead.

Related Work

In the classic activity recognition pipeline [3], the first step is to record and store sensor data. The observed activities, executed by humans, animals or other actors, are recorded with different sensors. Each sensor generates a data *stream*, whether this is a scene camera for annotation purposes, a body-worn motion capturing system or binary sensors like switches. Sampled at different rates, with differing resolutions, ranges, units and formats these streams offer a large variety of recording parameters. These parameters are usually documented in an additional file that resides next to the actual data [1, 6, 7, 8, 12]. The actual data is commonly stored in a CSV file, in a binary format for Matlab or NumPy, or in Machine Learning frameworks specific ones like ARFF [5] or libSVM [4].

Synchronizing such multi-modal data, i.e. converting this data to the same rate and making sure that recorded events happened at the same time presents a major challenge [12, 13, 16]. Possible approaches range from offline recording with post-hoc synchronization on a global clock, to live streaming with a minimum delay assumption - all but the last one require some form of clock synchronization and careful preparation. Storing events with timestamps on a global clock is then one possible way to allow for post-recording synchronization, i.e. each event is stored as a tuple of `<timestamp, event data>`.

The following step of merging such time-coded streams often requires to adapt their respective rates. Imagine, for example, a concurrent recording of GPS at 3Hz and acceleration at 100Hz. To merge both streams: will GPS be upsampled or acceleration downsampled, or both resampled to a common rate? Which strategy is used for this interpolation, is data simply repeated or can we assume some kind of dependency between samples? How is jitter and missing data handled? These question need to be answered whenever *time-coded* sensor data is used. A file format which makes the choice of possible solutions explicit is the goal of this paper.

Multi-Media Container Approach

Sensor data commonly used in Activity Recognition is not different from low-rate audio or video data. Common pa-

rameters are shared, and one-dimensional sensor data can be encoded with a lossless audio codec for compression. Rate, sample format and number of channels need to be specified for an audio track. The number of channels is equivalent to the number of axis an inertial sensor provides, as well as its sample rate. The sample format, i.e. how many bits are used to encode one measurement, is also required for such a sensor. Other typical parameters, like the range settings or conversion factor to SI units (if not encoded as such), can be stored as additional meta-data, as those are usually not required for an audio track.

Lossless compression, like FLAC [17] or WavPack [2], can be applied to such encoded data streams. This allows to trade additional processing for efficient storage. In the evaluation section several lossless schemes are compared. These include the general LZMA2 and ZIP compressors, and the FLAC [17] and WavPack [2] audio compressors. All but the first two can be easily included in multi-media container formats. To use audio streams, data needs to be sampled at a constant rate, i.e. the time between two consecutive samples is constant and only jitter smaller than this span is allowed. Put differently, the time between two consecutive data samples t_i and t_{i+1} at frame i must always be less than or equal to the sampling rate r: $\forall i \in N : t_{i+1} - t_i \leq \frac{1}{r}$. Compared to time-coded storage, the recording system has be designed to satisfy this constraint. Problems with a falsely assumed constant rate recording setup will therefore surface faster. Especially in distributed recording settings, where above mentioned constraints is checked only against local clock which might drift away from a global clock.

Sparsely sampled events can be encoded as subtitles. Here, each sample is recorded independently of its preceding event, i.e. the above mentioned constraint does not hold. Each event needs to be stored with a time-code and the actual event data. Depending on the chosen format, this can also include a position in the frame of an adjacent video stream or other information. For example, this can be used to annotate objects in a video stream. A popular format is the Substation Alpha Subtitle (SSA[10]) encoding, which includes the just mentioned features. Since data is encoded as strings, it is suitable for encoding ground truth labels. To a limited extent, since no compression is available, it can be used for sensor events as well. For example, low rate binary sensors, like RFID readers could be encoded as a subtitle.

Encoded sensor and subtitle data can then be combined with audio and video streams in a multi-media container format. One such standard is the Matroska [11] format, that is also available in a downgraded version called WebM [15] for webbrowsers. Once the data streams are combined into one such file, this data can be ”played” back in a synchronous manner. This means that streams recorded at different rates, and in different formats, need to be converted to a common rate and possibly common format. Meta-data that contains additional information like recording settings, descriptions and identifiers can be stored in addition to the parameters already contained in the stream encoding. For this task off-the-shelf software, like FFMpeg [14] can be used, which also provides functionality like compression, resampling, format conversion and filtering. Annotation tasks can be executing with standard subtitle editing software, discouraging the creation of yet another annotation tool. Furthermore, video streaming servers can be used for transporting live sensor data recordings to remote places.

The use of such a standard format for curating datasets allows for re-using existing software, however not without limitations. Asynchronous, also called sparsely sampled, data

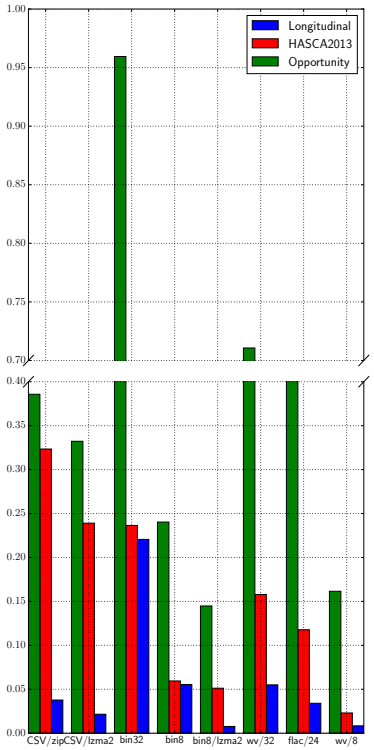

Figure 1: Fraction of storage required for three datasets compared to uncompressed CSV files. Zip and LZMA2 text compression, 32-bit binary, WavPack [2] with 32/8-bits and 24-bit FLAC [17] audio encodings are shown.

recorded at high rates is not supported. This mainly stems from the simplifying assumption that streams are recorded with a constant rate. Satisfying this constraint while recording might be easier than handling asynchronicity later on. For example, breaks, shifts or jitter due to firmware bugs can be detected earlier. Another shortcoming is that structured data can not be stored transparently, each event is assumed to consist of one data type only, e.g. multiple channels of 8-bit integers in contrast to a mix of data types. In general this is hard limitation, however different data types can also be encoded in multiple streams. Also, the en- and decoding overhead might be a limitation, which we will look at in the next section.

Compared to the de-facto standard of using CSV files, encoding sensor data as audio, annotations as subtitle and combining both with video- and audio-based provides several improvements. Important parameters like sampling rate, format and number of axes is included in the file. Adding additional information as meta-data leads to a *self-descriptive* format. *Synchronous* playback of multiple streams, which requires re-sampling, is supported by off-the-shelf software. Related problems, like un-synchronized streams can be caught earlier, since this step is explicit. The container format is *flexible* enough to support different number formats, i.e. values can be encoded as floats or integers of varying bit-size. Optional compression leads to *compact* storage, which allows for efficient storage and transmission. Additionally, when thinking about large datasets, such a container format requires *divisible* storage. This functionality (seeking without reading the whole dataset into memory[1]) is provided.

Evaluation

Compressing sensor data as an audio stream incurs an en- and decoding overhead, and provides optimized storage. In this section both are quantified. By a repetitive measurement of the relative wall clock time for decompression, its overhead is measured. The compression factor is determined by comparing the number of bytes required to store the compressed file to the original, deflated CSV file. Binary and text-based storage is compared. The Zip and LZMA2 algorithms are used for general byte-wise compression, and the lossless FLAC and WavPack compressor for audio-based compression. LZMA2, since it performs better than ZIP, is tested on text and binary files. The approach of compressing binary files with a general compressor is used by Numpy for example. The fraction of required storage after compression is given relative to the deflated, original CSV file. For the runtime overhead, the fraction of reading time relative to reading and *converting* the CSV file into a memory image is reported. The test were run on the Opportunity [12], HASC Challenge [8] and on twenty days of the Freiburg Longitudinal Wrist [9] datasets. A machine with an i7-4600U CPU running at 2.1GHz with 8GB of memory was used for all tests. Figure 2 and Figure 1 show the results of these tests, *CSV/zip* refers to a zip-compressed CSV file, *CSV/lzma2* to an LZMA2 compressed file[2], *bin** refers to signed integers with the respective bit length optionally compressed with LZMA2, *wv** to WavPack compression of varying bit size per value and *FLAC* compressor which only supports 24bits values.

Processing Overhead

It is interesting to check how much overhead is incurred for decompression by each algorithm, as this gives an insight if data needs to be stored in an intermediate format while

[1] which would be required for time-coded storage

[2] the XZ utils package was used

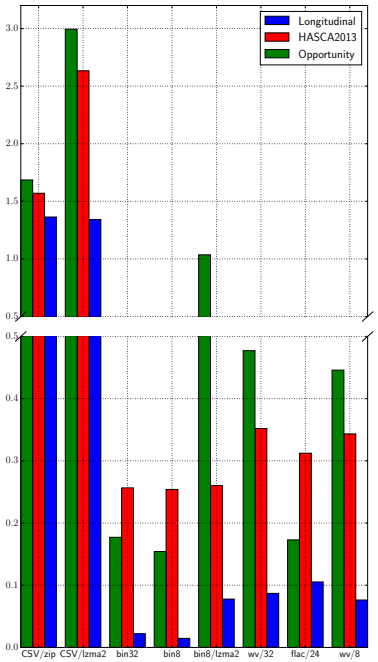

Figure 2: Relative runtime overhead for decoding each format. Shown is the fraction of wall time required to decode the respective scheme relative to the time required to parse a CSV file into memory. The result for each scheme is the (binary) data stored in memory.

evaluating a recognition pipeline. If the overhead is comparatively low no intermediate storage format is required and data can always be loaded from such a file. However, should decoding require more processing time than the actual processing, an intermediate file needs to be used.

Naturally such a format would be binary, at best a memory image which can be mapped into main memory. This is the format that will be decoded to in the following. The baseline is therefore the time required to convert a CSV from disk into a binary format in memory. The fraction of time required to do the same for each compression scheme is reported in Figure 2. Each test is repeated six times, and the first run is discarded, i.e. data is always read from the disk cache.

Just parsing a CSV file incurs an up to hundred-fold overhead (bin8 in Figure 2) compared to reading a binary file. Compressing CSV data [3] can increase the runtime by $1.4 - 3.0$ times. So, looking only at runtime performance a CSV file should hardly be used for large datasets. When comparing compression schemes, it can be seen that WavPack provides the most consistent performance measures over all datasets. It is not slower than the more general LZMA2 compressor, and the FLAC compressor is only faster on two datasets. However, for the decoding task at hand here, an overhead of at least two times is incurred compared to raw binary storage. A trade-off between storage efficiency and performance has to be found.

Storage Efficiency

General compression and audio compression algorithms were tested. Raw binary, WavPack [2] and FLAC [17] compression were taken from the FFmpeg [14] suite with default parameters. Figure 1 shows the amount of compression that was achieved for each dataset per algorithm.

The datasets show different characteristics found in other datasets as well. For example the Longitudinal [9] dataset can be massively compressed with text-based algorithm, almost down to 2% of its original size. This is mainly owed to the fact that the contained acceleration data was recorded with a resolution of only 8-bits, and that a run-length compression was already applied during recording. This run-length compression is deflated for CSV storage first, adding a lot of redundancy. For the same reason, storing data non-compressed in 32-bit binary format is actually larger than the zip-compressed text-format. However, encoding with the original 8-bit resolution in the WavPack compression leads to a slightly better storage efficiency.

The same effect is visible for the Opportunity [12] datasets, where feature vector are stored instead of raw data. Storing in 32-bit binary increases the size again, which means that the average string-length for representing a number in this dataset requires little more than 5byte. Only when limiting the number format to 8bits a stronger compression can be achieved with an audio codec. The maximum dynamic range that can stored with a text-based format is however limited to the (decimal) encoding, (less than 10000 for five digits), while a comparable binary encoding can range up to 2^{5*8}.

The HASCA dataset [8] does not show this effect. Mainly because the CSV data contains floats with at least ten digits. These could be stored with 32 or 64bit, which would be more efficient than their text counterpart. Especially since values are stored with more than eight digits per value.

When optimizing data storage for space efficiency, the encoding of each value is the most critical factor. Limiting the

[3] note that the Figure 2 represent the factor between the compression and simply *reading* and uncompressed CSV file

number of bits per value, in essence assuming a limited dynamic range of the encoded signal, has the strongest influence on the storage efficiency. However, when encoding values in text format and a dynamic range that is limited to four characters is enough, a text compression algorithm is not worse than encoding data in binary format. For the general case and when binary storage can be used, the WavPack compression provides the same storage efficiency as the more general LZMA2 compressor.

Conclusion

Curated time-series data provides the basis for comparing Activity Recognition and other Machine Learning approaches in an objective and repeatable manner. This data usually includes low-rate sensor data, e.g. motion sensors, time-coded annotations, and second-level evidence like video and audio recordings. The state of the art for exchanging this data seems to be a time-coded CSV format. Synchronizing the stored data-streams is usually not done by the dataset provider and the dataset consumer is left with this challenge that usually requires information of the recording setup. This is especially problematic when video or audio data is recorded as well.

Additionally the CSV format incurs a large overhead both in runtime and storage. A possible alternative, with lower overhead, is presented here. Motion and other sensor data, as well as extracted features, can be stored in lossless audio formats. Ground truth labels and other time-coded information can be stored in subtitle format. These streams can then be merged in a common multi-media container (e.g. Matroska), with additional video streams. One recording session is stored in a single file, that can be *self-descriptive*, *synchronized* and with a fitting *storage-runtime trade-off*.

References

[1] Marko Borazio et al. "Towards Benchmarked Sleep Detection with Inertial Wrist-worn Sensing Units". In: *Healthc. Informatics*. 2014.

[2] David Bryant. *The WavPack Codec*. URL: https://www.wavpack.org/.

[3] Andreas Bulling, Ulf Blanke, and Bernt Schiele. "A tutorial on human activity recognition using body-worn inertial sensors". In: *ACM Comput. Surv.* 46.3 (2014), pp. 1–33. ISSN: 03600300. DOI: 10.1145/2499621. URL: http://dl.acm.org/citation.cfm?doid=2578702.2499621.

[4] Chih-Chung Chang and Chih-Jen Lin. *LIBSVM: A Library for Support Vector Machines*. 2011. DOI: 10.1145/1961189.1961199.

[5] Mark Hall et al. "The WEKA data mining software". In: *SIGKDD Explor. Newsl.* 11.1 (2009), p. 10. ISSN: 19310145. DOI: 10.1145/1656274.1656278. URL: http://portal.acm.org/citation.cfm?doid=1656274.1656278.

[6] Stephen S Intille et al. "Using a Live-In Laboratory for Ubiquitous Computing Research". In: *Pervasive Comput.* (2006), pp. 349–365.

[7] T L M Van Kasteren, G Englebienne, and B J A Kr. "Human Activity Recognition from Wireless Sensor Network Data : Benchmark and Software". In: *Act. Recognit. Pervasive Intell. Environ.* 2010.

[8] Nobuo Kawaguchi, Nobuhiro Ogawa, and Yohei Iwasaki. "Hasc challenge: gathering large scale human activity corpus for the real-world activity understandings". In: *Proc. 2nd Augment. Hum. Int. Conf.* (2011), p. 27. DOI: http://doi.acm.org/10.1145/1959826.1959853. URL: http://dl.acm.org/citation.cfm?id=1959853.

[9] Kristof van Laerhoven. *Longitudinal Wrist Motion Dataset*. URL: https://es.informatik.uni-freiburg.de/application/files/hhg%7B%5C_%7Dlogs/0089/index.html (visited on 06/06/2016).

[10] David Lamparter. *Advanced Sub Station Alpha*. URL: http://fileformats.wikia.com/wiki/SubStation%7B%5C_%7DAlpha (visited on 06/06/2016).

[11] Non-Profit Organization Matroska. *Tha Matroska File Format*. URL: https://www.matroska.org/ (visited on 06/06/2016).

[12] Daniel Roggen et al. “Collecting complex activity datasets in highly rich networked sensor environments”. In: *Int. Conf. Networked Sens. Syst.* 2010.

[13] Thomas Stiefmeier et al. “Wearable activity tracking in car manufacturing”. In: *IEEE Pervasive Comput.* 7.2 (2008), pp. 42–50. ISSN: 15361268. DOI: 10.1109/MPRV.2008.40.

[14] Suramya Tomar. “Converting video formats with FFmpeg”. In: *Linux J. 2006* (2006).

[15] Consortium WebM. *The WebM File Format*. URL: http://www.webmproject.org/ (visited on 06/06/2016).

[16] Tracy Westeyn et al. “A naive technique correcting time-series data for recognition applications”. In: *Proc. - Int. Symp. Wearable Comput. ISWC* (2009), pp. 159–160. ISSN: 15504816. DOI: 10.1109/ISWC.2009.19.

[17] Foundation Xiph.org. *The Free Lossless Audio Codec (FLAC)*. URL: https://www.xiph.org/flac/.