The Social Signal Interpretation (SSI) Framework

Multimodal Signal Processing and Recognition in Real-Time

Johannes Wagner, Florian Lingenfelser, Tobias Baur,
Ionut Damian, Felix Kistler and Elisabeth André
Lab for Human Centered Multimedia, University of Augsburg
name@hcm-lab.de

ABSTRACT

Automatic detection and interpretation of social signals carried by voice, gestures, mimics, etc. will play a key-role for next-generation interfaces as it paves the way towards a more intuitive and natural human-computer interaction. The paper at hand introduces Social Signal Interpretation (SSI), a framework for real-time recognition of social signals. SSI supports a large range of sensor devices, filter and feature algorithms, as well as, machine learning and pattern recognition tools. It encourages developers to add new components using SSI's C++ API, but also addresses front end users by offering an XML interface to build pipelines with a text editor. SSI is freely available under GPL at `http://openssi.net`.

Categories and Subject Descriptors

I.5 [**PATTERN RECOGNITION**]: Applications—*Computer vision; Waveform analysis; Signal processing*

Keywords

Social Signal Processing, Real-time Pattern Recognition, Multimodal Fusion, Open Source Framework

1. INTRODUCTION

Today's computer interfaces are still based on explicit commands only. This of course differs greatly from natural human communication, which to a large extent is based on implicit interaction. A twinkle in one's eye, the wave of one's hand or the tone of one's voice sometimes tells more about a human's intentions than a hundred words. Moving towards more intuitive interaction is therefore an important aim of research on next-generation human-computer interfaces [7]. However, intuitive interaction requires the computer to perceive implicit user behaviour. Therefore, we have to equip machines with tools that are able to recognize and interpret diverse types of social signals carried by voice, gestures, mimics, etc.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
ACM MM 2013 Oct 21–25, 2013, Barcelona, Spain
Copyright 2013 ACM 978-1-4503-2404-5/13/10 ...$15.00.
http://dx.doi.org/10.1145/2502081.2502223.

Challenges

Developing tools able to detect and react to user behaviour in real-time involves a number of challenges.

- *Synchronization and coherent treatment of signals*

Different kind of sensors have to be used to record the various signals that carry social cues. Microphones replace the human ear and capture the human voice and other sounds. Video and depth cameras replace human vision and allow spotting humans to analyze body and facial expressions. Motion sensors worn by the user measure body posture and motion at very high precision, while physiological sensors monitor physiological signals, such as heart rate or respiration. The signals delivered by these different types of sensors differ greatly in quantisation and sample rate, e. g. a video image consists of several thousand pixel values delivered at rates about 30 frames per second. An audio sample on the other hand can be represented by a single integer or floating point value, but should be sampled at a rate of at least 16kHz. In order to combine information generated by such varying sources raw signal streams need to be synchronized and handled in a coherent way.

- *High amount of variability, uncertainty and ambiguity*

Human communication does not follow the precise mechanisms of a machine, but is tainted with a high amount of variability, uncertainty and ambiguity. Hence, robust recognizers have to be built that use probabilistic models to recognize and interpret observed behaviour. Most often recognition can be broken down in three-steps: (a) *Data segmentation*, which is the process of detecting on- and offsets of actions, which carry relevant information about the user's intention and goals. (b) *Feature extraction*, which relates to the reduction of a raw sensor stream to a set of compact features – keeping only the essential information necessary to classify the observed behaviour. (c) *Classification*, which describes the mapping of observed feature vectors onto a set of discrete states or continuous values. To accomplish these tasks we need to collect large numbers of representative samples and train recognition models.

- *Fusing multimodal data*

To solve ambiguity in human interaction information extracted from diverse channels need to be combined. This can happen at various levels. Already at data level, e. g. when depth information is enhanced with color information. At feature level, when features of two ore more channels are put together to a single feature vector. Or decision level, when probabilities of

different recognizers are combined. In the latter cases fused information should represent the same moment in time. If this is not possible due to temporal offsets (e. g. a gesture followed by a verbal instruction) fusion has to take place at event level. The preferred level depends on the type of information that is fused.

- *Real-time processing*

A natural and fluent interaction requires prompt reactions to observed behaviour. Therefore sensor information needs to be processed in pipelines on the fly, i. e. new sample values are constantly read by the sensors while feature extraction and recognition is applied simultaneously to buffered samples. The window length at which components operate should be variable as it depends on the type of signal and the level of processing. Usually, early processing steps are applied on possibly overlapping small windows of few milliseconds, while later processing takes place over segments of up to several seconds.

Existing Tools

There is a number of free and commercial software related to signal processing and machine learning. Some of them are specialized in a certain task, such as Weka[1], which offers a large collection of machine learning algorithms for data mining. Others are tailored to a certain modality, like Praat[2] for audio processing. Or Matlab[3], which offers a simplified scripting language for applying a large body of signal processing algorithms. Tools with support for live sensor input, on the other hand, are still rare. Examples of architectures for multimodal processing, are Pure Data[4], EyesWeb [1] or OpenInterface [9]. Here, developers can choose from a considerable large set of input, processing and output components, and patch them using a graphical programming environment to construct complex processing pipelines. However, they are not specially tuned for building machine learning pipelines and collecting training corpora. A toolkit developed for real-time affect recognition from speech is the openEAR toolkit with its feature extracting backend openSMILE [3]. It is, however, developed with a strong focus on audio processing.

Social Signal Interpretation (SSI)

The Social Signal Processing framework (SSI) complements existing tools by offering special support for the development of online recognition systems from multiple sensors. Mentioned problems are tackled in a number of ways:

- An architecture is established to handle diverse signals in a coherent way, no matter if it is a waveform, a heart beat signal, or a video image.
- Implementation details related to real-time processing such as buffering, synchronization, and threading are hidden from the developer.
- All tasks to assemble the machine learning pipeline, ranging from live sensor input and real-time signal processing to high-level feature extraction and online classification, are covered.
- Different strategies for fusing information gained from different modalities at various levels are available.
- Support for a large variety of sensors, as well as, filter and feature algorithms to process captured signals.

Two type of users are addressed: developers are provided a C++–API that encourages them to write new components and front end users can define recognition pipelines in XML from available components.

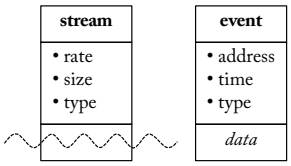
A previous version of SSI (v0.8) has been described in [13]. Apart from additional sensor devices and signal processing algorithms, a completely revised and way more powerful event handling (replacing the concept of a *Trigger*) makes the main contribution of the current version (v0.92).

2. ARCHITECTURE

SSI is implemented in C/C++ and optimized to run on multiple CPU cores. It has been developed with Visual Studio on Microsoft Windows. Since the integration of sensors is a highly platform dependent task other platforms are currently not supported. However, apart from sensor wrappers and the plugin interface large parts of the framework should be platform independent. External libraries, such as OpenCv, have been included to get rid of additional dependencies.

Data structures

In order to keep the framework as generic as possible, data handling is broken down to two basic data structures: streams and events. Basically everything read from a sensor is transformed into a stream and handled as a continuous flow of samples at a fixed sample rate and size. Filter and feature algorithms can be applied to manipulate a stream. During such operations sample rate and size of a stream may change. To keep streams synchronized SSI constantly checks if the number of retrieved samples corresponds to the expected amount of samples according to the sample rate of the stream. If a mismatch is detected a stream is adjusted accordingly, i. e. samples are removed or added via interpolation.



The counterpart of streams are events. Events describe parts of streams that are relevant in some way, e. g. for classification. Events are usually generated from continuous streams by applying some kind of activity detection. Their length is variable and they may contain additional data, e. g. a feature set or a string.

Recognition

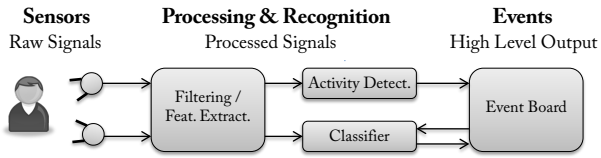


Figure 1: Scheme of a basic recognition pipeline.

A recognition pipeline is set up from several autonomic components. First, transformers can be applied to filter the raw signals and transform it to a compact set of features. Based on the pre-processed streams activity detection components are looking for parts in the signal relevant for recognition. If an on and offset is detected, an event is sent to the event board. Recognition components that have subscribed to the event are now informed. According to the segment they can request stream chunks and

[1] `http://www.cs.waikato.ac.nz/ml/weka/`
[2] `http://www.praat.org`
[3] `http://www.mathworks.com/products/matlab/`
[4] `http://puredata.info/`

feed them to a classifier. Note that the architecture suits both, dynamic classification if streams of variable length are taken (e. g. HMMs), and statistical classification if streams are described by a set of statistical features (e. g. SVM). Finally, a new event containing the recognized class probabilities is generated and possibly received by other components, e. g. fusion algorithms.

Fusion

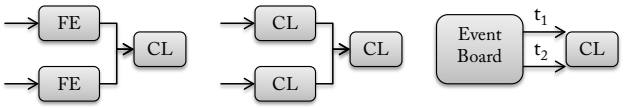


Figure 2: Ways of fusing information in SSI. (FE=feature extraction, CL=classifier)

Figure 2 shows ways of fusing information in SSI. Two or more feature extraction components can be plugged to the same recognition component, which combines features before recognition is applied. Likewise, outcome of two or more classifiers can be combined. Finally, fusion may be applied at event level. This allows combining information at different time scales.

3. AVAILABLE COMPONENTS

Currently, SSI supports streaming from multi-channel audio interfaces, web/dv cameras, Nintendo's Wii remote control, depth sensors such as Microsoft's Kinect, various physiological sensors (Nexus, ProComp, IOM, Alive Heart Monitor), mobile and stationed eye tracker (SMI), the Leap Motion, and a full body motion capture system by XSense. Raw and transformed streams can be stored, visualized and streamed via socket connections possibly using Open Sound Control, Yarp or ActiveMQ.

SSI involves a large set of transformer components, such as moving and sliding average filters, various normalization functions, dsp filters (Butterworth, Chebyshev), derivative, and integral filters, as well as, FFT and statistical features. Several open source libraries have been integrated, including OpenCV, ARToolKit, SHORE, Torch, or OpenSmile. Systems developed at our lab, such as FUBI [5] for Kinect based gesture or EmoVoice [12] for emotional speech recognition, are fully supported, too.

In order to accomplish the machine learning pipeline, classification models such as K-Nearest Neighbours, Naive Bayes, Linear Discriminant Analysis, Support Vector Machines or Hidden Markov Models are part of SSI. Fusion strategies range from weighted and unweighted voting schemes, algebraic combiners over specialist selection algorithms and constructed lookup tables to hybrid and meta level fusion. For feature selection, wrapper approaches such as Sequential Forward/Backward Search, as well as, filter approaches like Correlation-based Feature Selection are available [14].

4. EXAMPLE

Let us demonstrate SSI by means of a pipeline to detect gestures drawn with the mouse. This simple application is chosen to convey the main idea behind SSI, but could be easily extended to a more complex one, e. g. by replacing the mouse with a Kinect sensor to recognize free-hand gestures.

In an XML pipeline each component is represented by an XML element that includes information how it connects to other components. We start from the sensor and open a connection to each stream we would like to access (cursor coordinates and button state).

```
<sensor create="Mouse" mask="1">
  <provider channel="cursor" pin="p_cursor"/>
  <provider channel="button" pin="p_button" />
</sensor>
```

To each channel we assign a pin name that allows us to address the stream later on. By setting the option `mask=1` we tell the sensor to listen to the left mouse button. If it is pressed the stream will now switch from 0 to 1. We detect non-zero segments by plugging an instance of `ThresEventSender` that we connect to the button stream (`pin="p_button"`):

```
<consumer create="ThresEventSender" ...
  mindur="0.2" sname="mouse" ename="pressed">
  <input pin="p_button" frame="0.25s" />
</consumer>
```

As soon as the button is released an event is fired. The name of the event is set to `pressed` and the name of the sender to `mouse`. We also demand that an event is only sent if the segment has a duration of at least $0.2s$. Finally, we set the frame size to `0.25s`, i. e. the component will receive data in chunks of $0.25s$ length.

In order to smooth the cursor signal we include a Butterworth filter of order `5` and a cutoff frequency of $1Hz$:

```
<transformer create="Butfilt" order="5" low="1.0">
       <input pin="p_cursor" frame="0.1s"/>
       <output pin="p_cursor_low"/>
</transformer>
```

Next, the low-passed cursor stream is picked up by a classifier, but only when a button event was detected. Therefore – instead of using a frame size – we set `listen="pressed@mouse"`:

```
<consumer create="Classifier" ...
  trainer="numbers" sname="mouse" ename="gesture">
  <input pin="p_cursor_low" listen="pressed@mouse"/>
</consumer>
```

Via options we set the name of a pre-trained model (`numbers`) containing templates for numbers 0-9. The result of a classification generates a new event (`gesture@mouse`). Finally, we include an instance of `SignalPainter` to visualize current cursor position and an instance of `EventMonitor` to display mouse events during the last $2000ms$ (see Figure 3):

```
<consumer create="SignalPainter" size="10.0">
  <input pin="p_cursor" frame="1" />
</consumer>
<listener create="EventMonitor">
  <input listen="@mouse" span="20000" />
</listener>
```

Note that the frame size of the painter component is set to `1`, which forces the component to draw each sample.

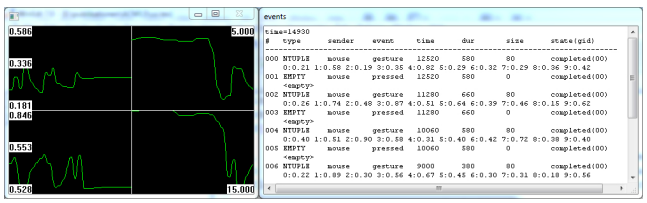


Figure 3: Output of example pipeline.

5. APPLICATION

SSI has been used by researchers around the world. Our EmoVoice component [12] (as part of SSI) was used in a various number of showcases in the European projects CALLAS and IRIS to analyse the expressivity in user speech. An example is the E-Tree [4], an Augmented Reality art installation of a virtual tree that grows, shrinks, changes colours, etc. by interpreting affective input from video, keywords and emotional voice tone.

SSI was also employed to construct the CALLAS expressivity corpus [2]. It consists of synchronized recordings from high-quality cameras and microphones, Nintendo's Wii remote controls and a data glove. Experiments have been conducted in three countries, namely Germany, Greece and Italy. In total, about 15 hours of interaction from more than 50 subjects was collected.

Within the AVLaughterCycle project, which aims at developing an audiovisual laughing machine, SSI has been used for recording and laughter detection in real-time [11]. The work is continued within the EU-funded ILHAIRE project. A first prototype of an interactive system that detects human laughs and responds appropriately is reported in [6].

TARDIS, another project under EU funding, aims to build a simulation platform for young people to improve their social skills. SSI is used for real-time detection of user's emotions and social attitudes through voice, body and facial expression recognition. A visualization tool called NoVA has been developed to visualize detected cues in graphs and heatmaps and give recommendations to the user.

The EU funded CEEDs project exploits the users' unconscious processes to optimize the presentation of large and complex databases. To this end, a sensor platform has been developed on top of SSI [15] and integrated into the eXperience Induction Machine (XIM), a multiuser mixed-reality space equipped with a number of sensors and effectors [10].

A research group at the Institute for Creative Technologies (ICT) from University of Southern California has developed a multimodal sensor fusion framework on top of SSI called MultiSense, which they use to investigate the capabilities of automatic non-verbal behavior descriptors to identify indicators of psychological disorders [8].

Outside an academic context, SSI was used at the Music Hack Day[5] 2013 in Barcelona to process physiological signals captured with the e-Health Sensor Platform for Arduino. During the hacking sessions participants were encouraged to use extracted user states to implement music related projects.

6. CONCLUSION

We introduced the Social Signal Interpretation (SSI) framework for real-time recognition of social signals. SSI supports a large set of sensor devices and allows users to set up recognition pipelines based on synchronized input from multiple modalities. A C++–API encourages developers to develop new components, while a simple XML interface is offered to front end users. SSI is freely available under GPL at `http://openssi.net`.

7. ACKNOWLEDGMENTS

The work described in this paper is funded by the European Union under research grant CEEDs (FP7-ICT-2009-5) and TARDIS (FP7-ICT-2011-7), and ILHAIRE, a Seventh Framework Programme (FP7/2007-2013) under grant agreement $n°270780$.

[5] `http://bcn.musichackday.org/2013/`

8. REFERENCES

[1] A. Camurri, P. Coletta, G. Varni, and S. Ghisio. Developing multimodal interactive systems with eyesweb xmi. In *Proc. NIME*, pages 305–308, New York, USA, 2007. ACM.

[2] G. Caridakis, J. Wagner, A. Raouzaiou, Z. Curto, E. André, and K. Karpouzis. A multimodal corpus for gesture expressivity analysis. In *Proc. LREC*, 2010.

[3] F. Eyben, M. Wöllmer, and B. Schuller. Opensmile: the munich versatile and fast open-source audio feature extractor. In *Proc. MM*, pages 1459–1462, New York, USA, 2010. ACM.

[4] S. W. Gilroy, M. Cavazza, R. Chaignon, S.-M. Mäkelä, M. Niranen, E. André, T. Vogt, J. Urbain, H. Seichter, M. Billinghurst, and M. Benayoun. An affective model of user experience for interactive art. In *Proc. ACE*, pages 107–110, New York, USA, 2008. ACM.

[5] F. Kistler, B. Endrass, I. Damian, C. Dang, and E. André. Natural interaction with culturally adaptive virtual characters. *JMUI*, pages 1–9.

[6] R. Niewiadomski, J. Hofmann, J. Urbain, T. Platt, J. Wagner, B. PIOT, H. Cakmak, S. Pammi, T. Baur, S. Dupont, M. Geist, F. Lingenfelser, G. McKeown, O. Pietquin, and W. Ruch. Laugh-aware virtual agent and its impact on user amusement . In *Proc. AAMAS*, Saint Paul, USA, May 2013.

[7] M. Pantic, A. Nijholt, A. Pentland, and T. S. Huang. Human-centred intelligent human-computer interaction (hci^2): how far are we from attaining it? *IJAACS*, 1(2):168–187, August 2008.

[8] S. Scherer, G. Stratou, M. Mahmoud, J. Boberg, J. Gratch, A. Rizzo, and L.-P. Morency. Automatic behavior descriptors for psychological disorder analysis. In *Proc. FG*, 2013.

[9] M. Serrano, L. Nigay, J.-Y. L. Lawson, A. Ramsay, R. Murray-Smith, and S. Denef. The openinterface framework: a tool for multimodal interaction. In *Proc. CHI*, pages 3501–3506, New York, USA, 2008. ACM.

[10] A. Spagnolli and L. Gamberini, editors. *Validating presence by relying on recollection: Human experience and performance in the mixed reality system XIM*, Padova, Italy, 16/10/2008 2008. CLEUP Cooperativa Libraria Universitaria Padova.

[11] J. Urbain, R. Niewiadomski, E. Bevacqua, T. Dutoit, A. Moinet, C. Pelachaud, B. Picart, J. Tilmanne, and J. Wagner. Avlaughtercycle. *JMUI*, 4:47–58, 2010.

[12] T. Vogt, E. André, and N. Bee. Emovoice - a framework for online recognitionof emotions from voice. In *Proc. PIT*, Kloster Irsee, Germany, June 2008. Springer.

[13] J. Wagner, F. Lingenfelser, and E. André. The social signal interpretation framework (ssi) for real time signal processing and recognition. In *Proc. of INTERSPEECH*, 2011.

[14] J. Wagner, F. Lingenfelser, E. André, and J. Kim. Exploring fusion methods for multimodal emotion recognition with missing data. *IEEE TAC*, 99, 2011.

[15] J. Wagner, F. Lingenfelser, E. André, D. Mazzei, A. Tognetti, A. Lanatà, D. D. Rossi, A. Betella, R. Zucca, P. Omedas, and P. F. Verschure. A sensing architecture for empathetic data systems. In *Proc. AH*, page 96–99, Stuttgart, Germany, 2013. ACM.