reacTIVision and TUIO: A Tangible Tabletop Toolkit

Martin Kaltenbrunner
Universitat Pompeu Fabra
08018 Barcelona, Spain
martin.kaltenbrunner@upf.edu

ABSTRACT
This article presents the recent updates and an evaluation of reacTIVision, a computer vision toolkit for fiducial marker tracking and multi-touch interaction. It also discusses the current and future development of the TUIO protocol and framework, which has been primarily designed as an abstraction layer for the description and transmission of pointers and tangible object states in the context of interactive tabletop surfaces. The initial protocol definition proved to be rather robust due to the simple and straightforward implementation approach, which also supported its widespread adoption within the open source community. This article also discusses the current limitations of this simplistic approach and provides an outlook towards a next generation protocol definition, which will address the need for additional descriptors and the protocol's general extensibility.

Author Keywords
Human Computer Interaction, Tangible User Interfaces, Interactive Surfaces, Computer Vision, Protocols.

ACM Classification Keywords
H.5.2. User Interfaces: Interaction styles, Input devices and strategies, Theory and methods.

INTRODUCTION
The TUIO protocol and reacTIVision framework comprise a toolkit for the rapid development of tabletop tangible user interfaces and multi-touch surfaces. Both components have been initially developed for musical applications in the context of the reacTable [1] project, a tangible modular synthesizer based on an interactive table surface. After the presentation of this instrument and also Jeff Han's multi-touch demos based on FTIR [2] had created considerable public interest in gesture-controlled surfaces, the TUIO protocol was eventually adopted by several open source initiatives with the goal to reverse engineer large multi-touch surfaces. Access to such a variety of freely available tools based on a shared protocol supported the democratization of the emerging tangible and multi-touch user interface technology. Since their initial publication and release in 2005, TUIO [3] and the reacTIVision toolkit have been successfully used for the design and implementation of numerous research, commercial and hobbyist projects, supporting the widespread adoption of the tangible interaction paradigm.

TANGIBLE SURFACE ABSTRACTION
The initial goal of the TUIO protocol definition was to provide a simple description of pointer and token states in the context of a two dimensional table surface, where pointers are defined as untagged points with normalized Cartesian coordinates, while tangible tokens provide an additional identification tag and rotation angle. Although this is a very simplified view of an interactive surface context, this description provides a basic solution for the implementation of multi-touch surfaces and the tracking of tagged physical objects. Such a basic model has of course its limitations, which became even more evident with its adoption within other application areas as well as with the further development of the reacTIVision engine itself. We will discuss these limitations and the consequent future extensions to this model further below.
After evaluating existing alternatives for the controller context [4], the TUIO protocol was based on Open Sound Control (OSC) [5], which has been widely adopted for the encoding of control data from musical instruments and general-purpose interactive devices. OSC successfully intends to overcome the performance limitations of the musical standard MIDI protocol, specifically regarding its bandwidth and data resolution, hence allowing for a more fine-grained control of advanced musical instrument designs. On the other hand the open approach of OSC compared to MIDI, makes it more difficult to interconnect arbitrary controller systems, therefore OSC based protocols such as TUIO need to define a clear semantics of the specific usage scenario within a separate message name space. TUIO in this case defines a range of profiles for the description of token and pointer state changes. Although OSC itself does not specify a default transport layer, most implementations including TUIO, are currently based on the delivery of UDP packets, which allow the necessary low latency delivery over commonly available local, wired or wireless IP networking infrastructure.

The initial application scenario for the TUIO protocol was defined by the interchange of control data between two or more table interfaces, which had been constructed for the first series of reacTable concerts, where four players were performing on two instruments located in different cities. Therefore the protocol design needed to be fast and robust enough for a musical performance over a standard Internet connection. Since the transmission of natural events - such as adding, moving and removing objects – could cause inconsistencies when certain events, most importantly the remove messages, are lost during transport, the protocol structure was specifically designed to stay consistent even when used on a fast but error prone UDP channel. Hence TUIO implements a state model instead of transmitting events, all currently active token and pointer identifiers are transmitted within each message bundle, which allows the continuous reconstruction of add and remove events on the receiving side by comparing the local and received identifiers.

The specification of such a descriptive network based protocol suggests the design of a distributed architecture, separating the tracking sensor component from the actual user application. This distributed approach enables the interoperability of various sensor technologies, platforms and programming environments. Apart from earlier considerations regarding the limited processing power of a single CPU system, which nowadays have become less important with the advent of powerful multi-core processors, another motivation for choosing this architecture was the use of the framework for teaching purposes. Dealing with students from different backgrounds and with varying technical skills, ranging from engineers to artists, providing a collection of TUIO client implementations for programming languages such as C++. Java and C# and more importantly multimedia authoring tools such as Processing, Pure Data, Max/MSP, Quartz Composer and Flash, allowed the involved students to concentrate on the actual interface design task using the most appropriate tool.

THE REACTIVISION ENGINE

Since its last open source software release[1] and the previous publication of its general functionality [6], the reacTIVision engine has undergone major feature and performance improvements. In addition to the significant improvement of the overall symbol tracking robustness, the recently published public version 1.4 also supports basic multi-touch finger tracking. While the initial versions of reacTIVision only performed the direct tracking of amoeba style fiducial symbols, which have been specifically developed in conjunction with the fiducial tracking core *libfidtrack*, the latest release introduces various tracking layers, which significantly enhance the symbol tracking performance. This is especially important in conditions with fast moving objects due to expressive gestures in musical performance.

[1] http://reactivision.sourceforge.net/

Fiducial Tracking

The principal fiducial tracking method used within reacTIVision is based on the analysis of region adjacency graphs, originally derived from Costanza's d-touch concept [7]. After applying a local adaptive threshold to the original camera image, the resulting binary image is then segmented into a graph of adjacent black and white regions. Hence the identification of the amoeba symbols is based on a dictionary search of previously defined tree structures that are encoded into the marker topology, and the actual symbol layout carries additional information, which allows the precise calculation of the symbol centre point and its rotation angle [8].

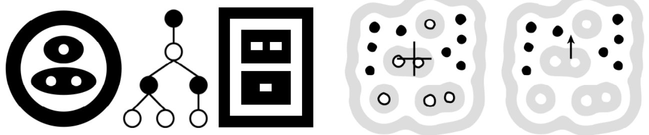

Figure 1: a) fiducial tree and two possible representations, b) encoded center point and angle information

Since the symbol structure allows an almost arbitrary representation of the actual geometry, we used a genetic algorithm [9] for the automatic generation of optimally shaped fiducial symbols, which eventually resulted in the organic amoeba appearance of the presently used fiducial marker collection. This genetic algorithm is driven by a fitness function that selects the generated symbols based on their size, symmetry and position and angle precision. The symbol position is calculated from the average of all leaf nodes, while the orientation vector points from the center point to the average of all black leaf nodes only. The current default set is for example defined by 18 nodes within a tree with maximum depth of two layers, which results in a possible range of 128 sequences, from which only 108 symbols have been selected to meet the minimum size and precision requirements.

The limitation to dedicated tree spaces, with a clearly defined node count and tree depth ensures the overall tracking robustness, since it is rather improbable to find these complex tree structures within arbitrary image noise, which limits the probability of finding false positives. We also separate the currently used alternative symbol collections by at least three nodes in order to avoid wrong symbol identification due to erroneous image data.

On the other hand this strict analysis is prone to minor changes of the symbol structure, such as addition and loss of individual leaf nodes, which can often appear in noisy or blurred images. While in these cases the algorithm is still capable of identifying the presence of a fiducial symbol in general, the identification of the individual symbol has become impossible, since the actual tree structure has been broken. Nevertheless we use the presence of unknown fiducial symbols within a *secondary fuzzy fiducial tracking* layer, where we simply assign unidentified erroneous symbol structures to the ones previously tracked nearby, which helps to improve the total symbol recognition rate.

Fast expressive movements, which are very common within musical performance, unveil the limitations of optical tracking methods. Problems such as motion blur can only be partially resolved with shorter camera exposure times and stronger illumination. Since these parameters are limited, very fast object movements yield a blurry fiducial image and hence result in a complete destruction of the fiducial structure, making it impossible for both the standard and fuzzy tracking method to identify an actual symbol. Therefore a third layer is tracking the position of the root node region, the usually white fiducial background. With the knowledge of the previous fiducial position and the displacement of the region centre from the actual symbol centre, the position of fast moving fiducial markers can be updated very accurately using just the *root node tracking method*. To summarize, the trajectory of fast moving objects, can be tracked accurately with a combination of the three methods outlined above, where the symbol can be tracked in all individual frames without additional filtering methods. Currently we are allowing a single frame without tracking result, which we are using to calculate the correct speed and acceleration updates before the object is finally removed from the list if not found in the following frame. Since the actual position during this single frame is not updated, we are planning to introduce an additional Kalman filter [10] in order to estimate the position of the lost symbol, which then also can be reassigned more easily to a nearby root region.

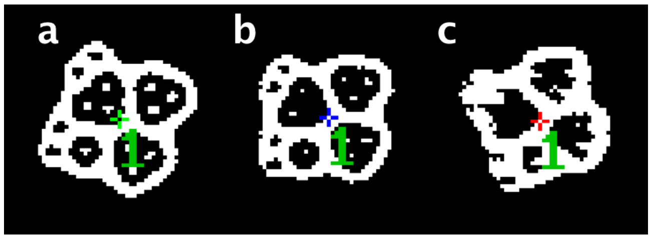

Figure 2: The three tracking modes a) full b) fuzzy and c) root region, depending on the retrieved symbol quality

Finger Tracking

The complementary multi-touch tracking layer introduced with the latest reacTIVision release takes advantage of the existing image processing infrastructure, without introducing a significant CPU overhead for this additional task. We are simply retrieving all white region candidates with a given size from the available image segmentation data and calculating the error comparing the candidate region to a round region prototype. The average finger size and maximum error can be adjusted within the application, yielding good tracking results in well-adjusted conditions. Compared to sole multi-touch trackers, reacTIVision is required to maintain the full fiducial structure intact, and therefore cannot afford the application of destructive image filters such as Gaussian blur in order to smooth the finger blob contour. Since this approach does not introduce any additional or parallel image filtering in order to enhance the source image, the initial configuration task of the camera settings and illumination environment has to be done more carefully than with comparable multi-touch only solutions. On the other hand this combined method ensures a low latency performance for musical applications, while providing simultaneous fiducial and finger tracking within the same image processing thread. The currently used tiled local adaptive threshold [11] method yields good results, and further improves the performance by neglecting tiles with a gradient below a configurable value. Unfortunately this method introduces square artifacts around low contrast regions, which can degrade the finger tracking accuracy. In order to improve the initial image quality we are currently evaluating alternative local adaptive threshold algorithms though, which should equally meet the requirements of the marker and blob tracking tasks.

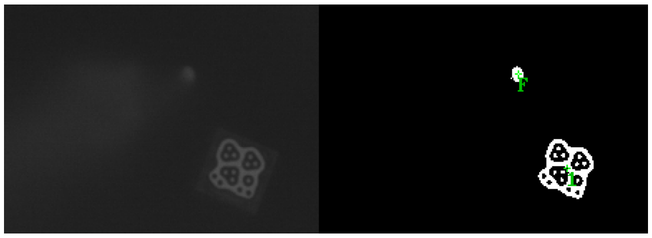

Figure 3: Original camera image and binary threshold image with finger and fiducial tracking feedback.

Blob Tracking

With the following release, we introduce an additional generic blob tracking layer, which is also taking advantage of the existing computational infrastructure, by selecting white regions within a given, configurable size range from the available segmentation data structures, while previously detected finger and fiducial root regions are excluded. In order to avoid additional image processing tasks, these regions are already encoded into a linked list of line spans during the segmentation process, which also annotates the final pixel area of each region. This data representation allows the reconstruction of the region contour and area without additional analysis of the actual source image itself, which again avoids additional processing overhead for this complementary tracking task. The span list implicitly encodes the full blob contour information in a compact format for further processing. The derived list of contour points can be efficiently reduced to the outer (and inner) blob contour, and consequently to a simplified list of contour points, which describes the overall blob geometry in sufficient detail. Finally for each of these retrieved regions, the oriented bounding box is calculated, which is providing an approximate description of its position, size and orientation. Current reacTIVision development builds already implement these basic geometry descriptors for untagged objects, which as a consequence have been also included within a third additional blob profile in an updated revision of the TUIO protocol, which we will describe in more detail below. The additional and more detailed geometry descriptors will be included in a future TUIO 2.0 specification though.

Amoeba Symbols

In addition to the updates to the core tracking software described above, some significant improvements to the fiducial symbol layout and rendering have been implemented, which enhance the overall tracking performance in boundary conditions such as low camera resolutions, reduced symbol sizes or increased surface distance, all of which result in a smaller size of the symbol in the actual camera image.

The number of symbols provided with the default set has been increased from the original 90 amoeba symbols to a total of 108 usable symbols out of the possible range of 128 within the described tree space. An improved fiducial generation algorithm, which introduces - already during the generation process - the final selection rules based on the symbol size and orientation vector length, yielded 20% more usable symbols that met the imposed criteria. By applying a high penalty to symbols that did not meet the initial size and vector boundaries, the genetic algorithm also converged much faster towards a usable symbol, thus reducing the overall generation time.

The newly created symbol set had its minimum orientation vector length improved by almost 5%, which generally supports the more robust calculation of the symbols' rotation angle. Also the maximum symbol size has been reduced by more than 10%, while as well showing a more uniform and narrow size distribution.

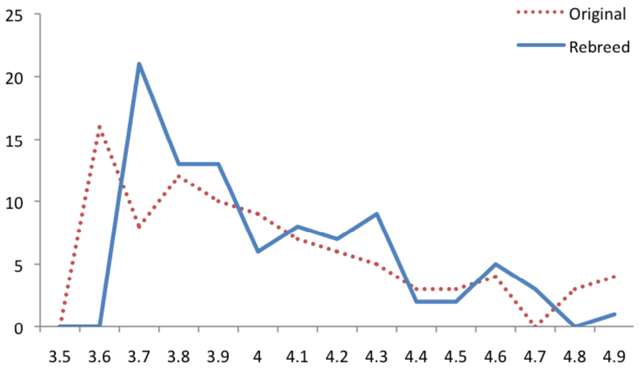

Figure 4a: Distribution of the orientation vector length

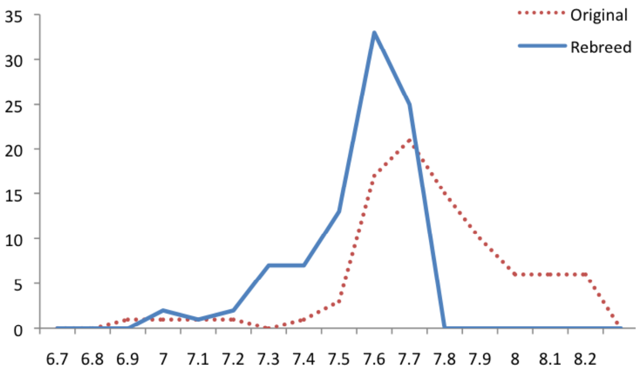

Figure 4b: Distribution of the symbol footprint size

We also introduced an alternative set of smaller symbols from a different tree space with 12 nodes only. While this tree space contains 15 possible variations, we selected 12 symbols that met the size and vector length criteria. This additional set can be used for applications that only require a limited set of different symbols IDs, but has the advantage of a reduced maximum symbol size by another 10% compared to the default amoeba set. In general, reacTIVision allows the usage of any arbitrary set of tree sequences, although we currently only provide these two subsets for the moment. Additional symbol collections can be added with a simple configuration file, but we recommend separating the selected tree spaces by at least three tree nodes, in order to avoid wrong symbol identifications. It should be also considered, that smaller symbol sets will not provide sufficiently precise position and rotation information, while these simpler tree configurations are also more likely to be found in arbitrary noise, and are therefore prone to yield false positives compared to the carefully selected standard symbols. Alternatively there are already third party fiducial generators available, which allow the generation of alternative symbols of any desired tree size.

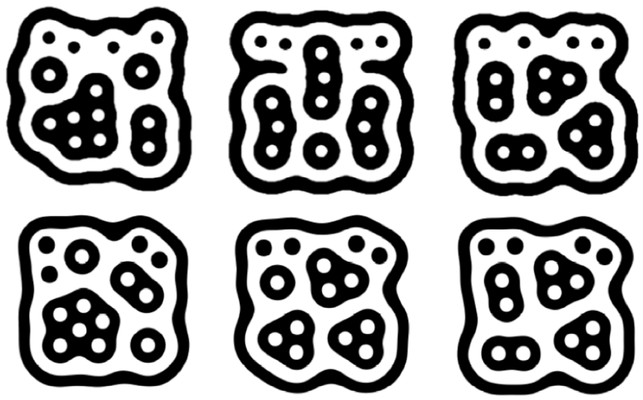

Figure 5: comparing the original symbol rendering (top row) to the improved rendering method (bottom row)

An improved graphical rendering method for the amoeba symbols introduces relatively enlarged leaf node proportions, which also support a better tracking performance of small scale or distant markers. Finally this new symbol set has been released in a vector based PDF format, which allows a high quality printing of the marker collection in any arbitrary size.

In order to increase the number of available marker IDs, the collection of 108 plus 12 standard symbols has been doubled though the addition of the inverted symbols with an initial black root node, resulting in a total number of 216 plus 24 standard symbols that are delivered with the current public release. This total of 240 distinguishable default markers should be sufficient for most application cases, considering that the individual marker IDs can also be repeated within the same context.

Performance Evaluation

The combination of the three marker tracking methods yield a satisfactory tracking performance even with fast moving objects, provided the camera and illumination settings are optimally configured with an appropriate short exposure time, which also guarantees a low latency image acquisition. The following chart illustrates the improvements in these boundary conditions, where the symbol structure is partially destroyed due to motion blur caused by expressive object handling as shown above.

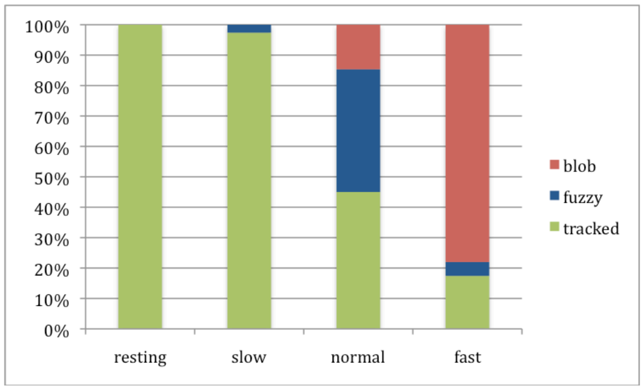

Figure 6: Tracking modes at varying speeds.

An evaluation of the tracking accuracy of resting symbols showed a standard deviation of around 0.05 pixels for the symbol centre point and a standard deviation of around 0.5 degrees for the rotation angle. These results allow for single pixel position accuracy and a rotation angle accuracy of three degrees in normal conditions, without significant jitter problems. We expect to improve the tracking accuracy for the rotation angle with the introduction of an additional Kalman filter component.
Performance measures of the latency of the current image processing chain depend on various factors, such as the camera resolution and resulting buffer size, the platform, compiler and CPU speed of the test system. Evaluating the frame latency on a 2GHz Core Duo Macbook on Linux, the median processing, analysis and delivery time for a VGA sized frame ranges around 5ms, which results in an acceptable system latency considering the complexity of the task. The following table illustrates the evolution of the processing latency for fiducial tracking adding the additional finger and blob tracking layers.

	ICC 11.0	**GCC 4.4**
Fiducial tracking	4.8 ms	5.6 ms
Fiducial & touch tracking	5.0 ms	6.0 ms
Fiducial & touch & contour	5.4 ms	6.5 ms

A more detailed analysis and review of the tracking performance for various conditions such as fiducial size, and comparison with other marker systems would unfortunately exceed the limits of this article and will be therefore addressed in a subsequent publication.

THE TUIO PROTOCOL

The original TUIO protocol specification was concentrating on the specific needs of the reacTable project, mainly focusing on tagged object and finger tracking in the context of a remote collaboration scenario, while ensuring the overall robustness of the networked distributed system. During the development and feature enhancements of our own tracking application, as well as with the integration of the TUIO protocol into further projects, several issues regarding missing features within the present profiles and the need for additional protocol extensions emerged. The extension of the existing message structure needs to be planned carefully though, considering the stability of all current implementations that rely on solid shared protocol definition.

Original TUIO Specification

A TUIO profile defines two central messages: *set* messages and *alive* messages. Set messages are used to communicate information about a token's state such as position, orientation, velocity and acceleration. Alive messages indicate the current set of tokens present on the surface using a list of unique session IDs. Additional *fseq* messages are defined to tag each frame update with a unique sequence ID. At typical TUIO bundle is therefore typically comprised of at least three messages, while the *set* messages can be accumulated in order to fully use the available space of a UDP packet. TUIO messages are commonly delivered to UDP port 3333 in the default configuration, although alternative transport methods are equally allowed. Please note that the following clear text message representations are only for demonstration purposes, the actual OSC message is transmitted in a compact binary format.

```
/tuio/[profile] alive [active session_IDs]

/tuio/[profile] set [session_ID attributes]

/tuio/[profile] fseq [int32]
```

There are two basic profiles for the description of pointers and tokens (here cursors and objects), which are commonly used within the context of a 2D surface. There exist additional profiles for 2.5D environments, which include the distance to the surface, as well as 3D environments, which also provide 3D rotation information for the object profiles. All profile types are generally designed to describe the surface or the space above an interactive table environment. Most currently available TUIO implementations are concentrating on the 2D profiles though. The specific set message syntax for the cursor and object profiles include attributes such as position, velocity and acceleration and is structured as following:

```
/tuio/2Dobj set sid id xpos ypos angle xvel yvel
rvel maccel raccel

/tuio/2Dcur set sid xpos ypos xvel yvel maccel
```

Updated TUIO 1.1 Specification

In order to provide a smooth transition path we are introducing an intermediate and backwards-compatible TUIO 1.1 specification, which adds two new features to the existing protocol specification, without breaking the existing client implementations: A third profile for the description of untagged objects and the possibility of multiplexing multiple tracker sources.

The complementary blob profile allows the further distinction between identified tagged symbols and unidentified plain blob objects, which are also providing basic additional geometric information.

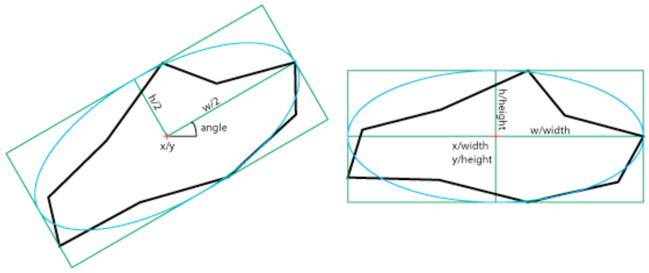

Figure 7: Simple description of a blob enclosure

```
/tuio/2Dblb set sid xpos ypos angle width height
area xvel yvel rvel macc racc
```

The profile's set message format describes the inner ellipse of an oriented bounding box, with its center point, the angle of the longer axis, its width and height as well as the blob area. Hence this compact format describes the approximate elliptical blob enclosure, which also allows the reconstruction of the oriented bounding box. The blob area is normalized by `pixels/width*height`, providing quick access to the overall blob size. The blob dimensions are defined as normalized values after performing an inverse rotation by -angle.

```
/tuio/[profile] source [name@address]
```

In order to allow the multiplexing of several TUIO trackers on the client side, an optional source message can be transmitted within each TUIO bundle, which enables the identification of the bundle's origin. The name@address argument is a single string that specifies the application name and any unique source address.

An issue, which currently cannot be addressed within a backward compatible TUIO extension, is the lack of reliable timing information. Assuming that OSC does already provide a sufficient time tag within the actual bundle header, we decided to not include a redundant time tag into the TUIO message structure. Unfortunately OSC implementations interpret the bundle time as a delivery time, which in some cases could cause the OSC layer to drop bundles with an earlier time stamp. The current TUIO implementations partially intend to compensate this with the inclusion of velocity and acceleration attributes within the set message structure.

Since several TUIO developers are working with Actionscript, the need for an alternative communication model for Flash, which currently does not support UDP sockets has emerged. The presently used workaround, which expands TUIO/OSC messages to an XML format that can be interpreted by Flash, is not comparable to the good performance results delivered by the common UDP transport method. As an alternative, TUIO can support an additional TUIO/TCP mode or a Flash local connection, which can be used for connecting to this kind of closed environments. The transparency of alternative transport methods is an advantage of the chosen OSC encoding.

Future TUIO 2.0 Specification

It has become clear that even with the intermediate protocol extensions the current simplistic approach is by far not sufficient for the description of a generalised tangible interaction environment. While TUIO 1.1 already addresses the basic needs for an additional descriptor for the object geometry, the strict separation of *cursor*, *object* and *blob* profiles is one of the mayor limitations for the future protocol extensions. Also, the existing object profile is lacking the possibility of transmitting marker content data while the cursor profile is missing important attributes such as cursor ID and pressure data. Finally TUIO is also missing detailed timing information, which unfortunately cannot be retrieved from the OSC bundle time tag as originally intended. The number of potential enhancements and changes to the current protocol structure justifies the introduction of a new TUIO 2.0 protocol specification, which eventually will resolve the shortcomings and design limitations of the current protocol generation.

As a consequence TUIO 2.0 will allow a more substantial update for the existing TUIO infrastructure. A flat and more extensible profile structure, which also integrates better into the overall OSC bundle and message formatting, will allow future incremental message updates which can add additional descriptors and functionality. *Token*, *Pointer* and *Geometry* messages can now be handled in parallel and can share the same session ID if these are actually referring to the very same object. Therefore for example *Pointer* messages can be extended with a bundled *Bounds* message, which transmits the actual geometry in addition to the generic pointer information. The basic geometry descriptors, which are similar to the format introduced in TUIO 1.1, can be incrementally extended with additional messages describing the *Contour*, *Skeleton* or full *Area* of the described region. It depends on the capabilities of the tracker implementation or the actual application setup.

Tokens will carry an additional type ID, which allows the multiplexing of various symbol types within a session. A *Token* can be extended with more detail by an optional *Symbol* message that encodes the information about the actual marker type and content, which will allow the introduction of alternative marker types, such as data matrix (or QR) codes or RFID tags. *Pointers* will include various additional attributes, such as pointer ID and type ID as well as pressure and region of influence, and can be also extended with optional *Control* messages, which allow the encoding of additional control dimensions from buttons, wheels, knobs or sliders. Finally the new TUIO protocol generation will also allow the description of object associations, such as container relationships or mechanical connections between individual objects. This for example allows the encoding of token-constraint systems as well as constructive object assemblies, which will extend the overall encoding possibilities from the purely spatial approach of the original specification and will therefore extend the scope of the protocol to support a broader range of tangible user interfaces.

Although there are recent developments towards the clarification of the OSC bundle timing with the introduction of a revised OSC 1.1 specification [12], TUIO 2.0 will add a redundant time tag to the *fseq* message of each frame, providing fine-grained timing information, which is necessary for correct gesture analysis. Since TUIO is based on OSC, any implementation can already choose to define its private message space in order to transmit custom controller data. TUIO augments this possibility with the definition of a *Custom* message syntax, which allows associating these custom attributes to the existing TUIO objects. There have been suggestions to introduce a back channel for the configuration of the tracker environment, but there are currently no plans to abandon the simplicity of the current unidirectional protocol approach.

The TUIO 2.0 specification draft is already close to its finalization, although we will wait until the consolidation of the intermediate TUIO 1.1, before we will start with the implementation of this next generation protocol. In order to support the migration towards the new version, the according client implementations will support both protocol generations.

THIRD PARTY TUIO IMPLEMENTATIONS

During the early stages of the TUIO development, the available tracker and client implementations were limited to the reacTable and similar environments, for which we initially designed this protocol. The first external project, which picked up the TUIO protocol as an abstraction for multi-touch interaction, was the touchlib library by David Wallin. This was also one of the first publicly available multi-touch tracking applications, since reacTIVision only implemented the touch functionality at a later point. Since then, a growing number of multi-touch and tangible interaction platforms have implemented the TUIO protocol, which lead to its more widespread adoption. A recently established community website provides detailed information about the current and future TUIO specifications, implementation notes for the development of TUIO enabled software as well as a growing list of client and tracker applications that support our protocol.[2] Please also refer to this website for further information about the projects mentioned in the following software selection.

[2] http://www.tuio.org/

TUIO Trackers

The currently available tracker implementations mostly include multi-touch software based on computer vision, such as touché, BBTouch and Community Core Vision (formerly tBeta). Further TUIO tracker implementations are based on controller hardware such as the Wiimote controller device, where WiimoteTUIO for example allows the rapid development of Whiteboard applications using only the IR tracking capabilities of a Wiimote controller and a suitable TUIO client application. In addition to that, there exist TUIO bridges for dedicated multi-touch hardware, such as the devices from N-trig, which are presently used for most available multi-touch tablet PCs. Similar integration initiatives have been started for Windows 7 and the Microsoft Surface, which have been extended to provide TUIO support at the system level [13]. Finally there also exist a variety of iPhone applications, which allow the usage of this hand-held device as a remote multi-touch controller that can send the TUIO over its wireless network connection. It has been shown that especially for this application case, the TUIO state model proved to be very robust on this error prone channel.

TUIO Clients

Apart from the primary TUIO client implementations, which are available for most mainstream programming languages and multimedia environments, the community contributed a large collection of additional TUIO implementations for several other environment that were not directly support by ourselves. This includes programming languages such as Objective C, Python, Smalltalk, Ruby and Actionscript as well as sound and media environments such as VVVV, SuperCollider, Chuck or Open Frameworks, and there are also several higher-level programming environments for gesture recognition and tangible interface development for Java, C# or C++ available, that are using TUIO as the common input layer.

Based on the TUIO client reference implementations, which basically decode the touch and object events from the protocol, there is also a growing number of end user applications available, taking advantage if these input events as an alternative controller interface. Applications such as NASA WorldWind, Google Earth, Second Life or the Blender game engine have been enhanced with multi-touch control with the help of the TUIO protocol.

Most recent versions of mainstream operating systems such as Windows 7 and Mac OS X 10.6 already include system level support for multi-touch input. Within the X-Window system, which is commonly used on Linux operating systems, the multi-pointer X-Server MPX [14] has recently been included into the main branch and will therefore soon become a standard component of all major Linux distributions. We are currently involved in the integration of the TUIO framework into MPX through the development of the xf86-input-tuio driver component, which will allow the seamless integration of the existing tracker software and libraries into the operating system infrastructure.

CONCLUSIONS AND FUTURE WORK

We have shown and documented the improvements and current functionality of the reacTIVision framework and provided an outlook to the future blob geometry extension that will be included within the next public version, which is also reflected within an intermediate update to the TUIO 1.1 protocol. In parallel there is ongoing work towards the definition and implementation of a future TUIO 2.0 protocol, which will hopefully provide a solid base for the realization of more versatile interactive surface environments. The future work on the tangible interaction framework will also shift the focus to a more generalized view of the overall TUIO platform, where reacTIVision will serve as a common reference implementation for the newly defined protocol features, which intends to open the further development to third party implementations based on alternative technologies.

Robust tracking performance regarding speed, latency and reliability are even more important in the context of expressive musical performance. Although our improvements have shown to be suitable for live performance conditions, optical tracking systems have also clear limitations regarding their temporal resolution. State of the art industrial cameras can deliver images at frame rates between 60-200 Hz, hence while we are adding further features to our tracking engine, we will ensure that frame rates up to 200 Hz can be processed in real time and at a reasonable latency. Apart from the common image analysis, we are currently evaluating the incorporation of dedicated GPU programming methods, which promise to deliver improved image processing performance. This approach will provide a responsive controller for musical performance and will allow the implementation of more fine-grained musical control gestures, such as performing a vibrato or the tapping of a rhythm. Additionally we are also looking into alternative sensor methods, which can augment and improve the overall input performance.

ACKNOWLEDGEMENTS

This research has been conducted in the context of the reacTable project, which is a team effort and has been developed in collaboration with my colleagues Günter Geiger, Marcos Alonso and Sergi Jordà within the Music Technology Group at the Pompeu Fabra University in Barcelona, Spain. The reacTIVision fiducial tracking engine and amoeba symbols have been developed in collaboration with Ross Bencina, and I'd also like to thank Enrico Costanza for his initial contribution to its development, as well as Till Bovermann for his participation in the TUIO initiative, which had been partially supported by the European Commission Cost287-ConGAS action. Many thanks also to my thesis supervisors Sile O'Modhrain and Xavier Serra for their valuable advice during the course of the project.

REFERENCES

1. S. Jordà, M. Kaltenbrunner, G. Geiger, and R. Bencina, "The reacTable," *Proceedings of the International Computer Music Conference,* 2005.
2. J. Y. Han, "Multi-touch sensing through frustrated total internal reflection," *SIGGRAPH 2005 Sketches,* 2005.
3. M. Kaltenbrunner, T. Bovermann, R. Bencina, and E. Costanza, "TUIO - A Protocol for Table Based Tangible User Interfaces," in *GW '05: Proceedings of the 6th International Workshop on Gesture in Human-Computer Interaction and Simulation*, 2005.
4. G. Reitmayr and D. Schmalstieg, "An open software architecture for virtual reality interaction," *VRST '01: Proceedings of the ACM symposium on Virtual reality software and technology,* 2001.
5. M. Wright, A. Freed, and A. Momeni, "OpenSound Control: state of the art 2003," *NIME '03: Proceedings of the 3rd conference on New interfaces for Musical Expression,* 2003.
6. M. Kaltenbrunner and R. Bencina, "reacTIVision: a computer-vision framework for table-based tangible interaction," *TEI '07: Proceedings of the 1st international conference on Tangible and embedded interaction,* 2007.
7. E. Costanza and J. A. Robinson, "A region adjacency tree approach to the detection and design of fiducials," *Vision, Video and Graphics (VVG),* pp. 63–70, 2003.
8. R. Bencina, M. Kaltenbrunner, and S. Jorda, "Improved Topological Fiducial Tracking in the reacTIVision System," *Proceedings of the IEEE International Workshop on Projector-Camera Systems*, 2005.
9. R. Bencina and M. Kaltenbrunner, "The Design and Evolution of Fiducials for the reacTIVision System," in *Proceedings of the 3rd International Conference on Generative Systems in the Electronic Arts*, 2005.
10. R. E. Kalman, "A New Approach to Linear Filtering and Prediction Problems," *Transactions of the ASME--Journal of Basic Engineering,* pp. 35-45, 1960.
11. J. Bernsen, "Dynamic thresholding of grey-level images," *Proceedings of the 8th International Conference on Pattern Recognition,* pp. 1251–1255, 1986.
12. A. Freed and A. Schneider, "Features and Future of Open Sound Control version 1.1 for NIME," *NIME '09: Proceedings of the 9th Conference on New Interfaces for Musical Expression,* 2009.
13. W. A. König, R. Rädle, and H. Reiterer, "Squidy: a zoomable design environment for natural user interfaces," *CHI EA '09: Proceedings of the 27th international conference on Human factors in computing systems,* 2009.
14. P. Hutterer and B. Thomas, "Groupware support in the windowing system," *AUIC '07: Proceedings of the eight Australasian conference on User interface*, 2007.