

reactIVision: A Computer-Vision Framework for Table-Based Tangible Interaction

Martin Kaltenbrunner
 Music Technology Group
 Universitat Pompeu Fabra
 Barcelona, Spain
 mkalten@ia.upf.es

Ross Bencina
 Sonic Fritter Pty Ltd
 Melbourne, Australia
 rossb@audiomulch.com

ABSTRACT

This article provides an introductory overview to first-time users of the reactIVision framework – an open-source cross-platform computer-vision framework primarily designed for the construction of table-based tangible user interfaces. The central component of the framework is a standalone application for fast and robust tracking of fiducial markers in a real-time video stream. The framework also defines a transport protocol for efficient and reliable transmission of object states via a local or wide area network. In addition, the distribution includes a collection of client example projects for various programming environments that allow the rapid development of unique tangible user interfaces. This article also provides a discussion of key points relevant to the construction of the necessary table hardware and surveys some projects that have been based on this technology.

Author Keywords

Tangible User Interface, Computer Vision, Application Development Framework.

ACM Classification Keywords

H.5.2. User Interfaces, I.4.9. Image Processing and Computer Vision Applications, D.2.6. Programming Environment

INTRODUCTION

The reactIVision framework has been developed as the primary sensor component for the reactTable [5], a tangible electro-acoustic musical instrument. It uses specially designed visual markers (fiducial symbols) [fig. 2] that can be attached to physical objects. The markers are recognized and tracked by a computer vision algorithm optimized for the specific marker design [1] improving the overall speed

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. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

TEI'07, February 15-17, 2007, Baton Rouge, Louisiana, USA.

Copyright 2007 ACM ISBN 978-1-59593-619-6/07/02...\$5.00.

and robustness of the recognition process. These fiducial marker symbols allow hundreds of unique marker identities to be distinguished as well as supporting the precise calculation of marker position and angle of rotation on a 2D plane.

reactIVision and its components have been made available under a combination of open source software licenses (GPL, LGPL, BSD) and can be obtained both as ready to use executable binaries and as source code from a public SourceForge site¹. This document describes the features of reactIVision 1.3 which has been released in conjunction with the publication of this article. The reactTable software website² provides further information about the project.

ARCHITECTURE

reactIVision has been designed as a distributed application framework rather than an object code library. Each component of the system is implemented as a separate executable process. Communication between components is achieved using a published protocol. This design simplifies use for novice programmers and facilitates integration with popular programming environments such as Processing and Pure Data. The architecture also allows the execution of framework components on different machines, which can be useful in certain installation contexts.

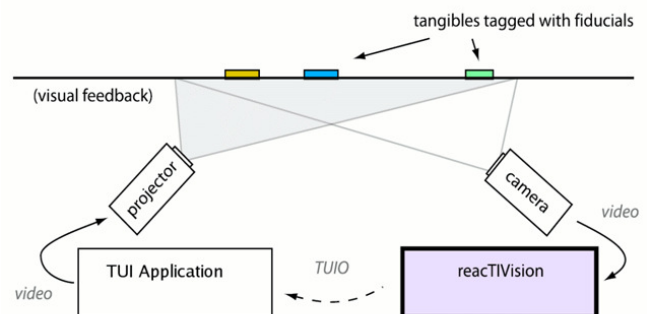


Fig. 1: reactIVision diagram

¹ <http://www.sourceforge.net/projects/reactivision>

² <http://mtg.upf.edu/reactable?reactivision>

Recognition Component

The reactIVision application acquires images from the camera, searches the video stream frame by frame for fiducial symbols and sends data about all identified symbols via a network socket to a listening application. The reactIVision application has been designed in a modular way, making it easy to add new image recognition and frame processing components.

The code base is cross-platform with builds for all three major operating systems, Windows, Mac OS X and Linux. It has been written in portable C++ code, combined with platform-dependent frame acquisition components. The video acquisition framework is also available separately as open source software under the name PortVideo.³

Communication Component

ReactIVision defines its own communication protocol TUIO [6] that was specifically designed for the needs of tabletop tangible user interfaces: encoding and transmitting the attributes of tangible artifacts that are found on a table surface. In order to provide fast and reliable communication with local and remote client applications the protocol layers a redundant messaging structure over UDP transport. TUIO defines a set of Open Sound Control [10] protocol messages. These messages constantly transmit the presence, position and angle of all found symbols along with further derived parameters. On the client side these redundant messages are then decoded to generic add, update and remove events corresponding to the physical actions that have been applied to each tangible object.

In order to achieve maximum compatibility with existing musical application environments reactIVision can alternatively send MIDI [8] control messages that can be individually configured for each fiducial symbol. However, due to the various limitations of MIDI, such as bandwidth and data resolution, TUIO is the recommended and default transport layer.

Client Components

In order to facilitate the development of tangible interface applications the reactIVision framework provides a large collection of example clients for a variety of programming languages including C++, C#, Java, Processing and Pure Data. Example clients provide a full TUIO client implementation that decodes the messages to generic interface events and draws the results into a graphical window or simply prints them to the console. Additional unsupported example projects are available for SuperCollider, Max/MSP and Flash. The TUIO simulator written in platform independent Java can be used to simulate a table environment during the initial development phase.

³ <http://www.sourceforge.net/projects/portvideo/>

FIDUCIAL ENGINES

This section gives some background regarding the history, design, evolution, and capabilities of the marker tracking implementations employed by reactIVision.

After some initial experiments with publicly available marker systems such as ARToolkit [7], the first reactTable prototype made use of E. Costanza's original D-touch [3] code, which was kindly provided by its author. Further development of the reactTable generated requirements for more compact symbol sizes as well as improved processing speed for real time musical interaction. This first lead to a reimplementing of the d-touch tracking algorithm with significant performance gains. Subsequently the fiducial marker geometry was redesigned to take advantage of a genetic algorithm, which minimized marker size and facilitated a more efficient tracking algorithm. All three fiducial recognition engines (d-touch, classic and amoeba) are available within the reactIVision application, with the most recent and reliable amoeba engine as the default.

In all three fiducial engines the source image frame is first converted to a black & white image with an adaptive threshold algorithm. This image is then segmented into a region adjacency graph reflecting the containment structure of alternating black and white regions. This graph is searched for unique tree structures, which are encoded into the fiducial symbols. Finally the identified trees are matched to a dictionary to retrieve unique marker ID numbers.

Amoeba Engine

The highly compact geometry of the amoeba fiducials was obtained by a genetic algorithm. This GA optimized the fiducial appearance using a set of fitness functions targeting shape, footprint size, center point and rotation angle accuracy. The current set distributed with reactIVision contains 90 different symbols that have been chosen from a pool of 128 with certain tree structure constraints. In this case all symbols have 19 leaf nodes and a maximum tree depth of 2. The limitation to specific tree structure constraints allows the exclusion of other structures found in noise, providing higher robustness of the algorithm by avoiding the detection of false positives.

The position of an amoeba symbol is calculated as the centroid of all found leaf nodes (small circular monochrome blobs), which provides sub-pixel accuracy. The orientation of the marker is calculated as the vector from the marker centroid to the centroid of all black leaves which are distributed in the upper part of the symbol.

A second fiducial set used internally for reactTable installations provides roughly 300 extra symbols that are usually printed onto business cards and handed out to the public. Just as with the standard symbol set, unique fiducial IDs are derived by comparing the detected tree structure to a dictionary of known trees.

Finger Tracking

As an initial solution for the tracking of fingertips in a multi-touch surface the simplest amoeba fiducial, with a single tree branch [fig. 2d] can be used as a small finger sticker. While this method is not as elegant as other optical finger tracking methods [4] it has proved to be simple and robust without any additional computational overhead since it can be detected using the existing fiducial tracking algorithm. Due to the minimal nature of the symbol, no rotation angle can be calculated from its structure, although in the case of tracking the finger as a simple pointer the position information alone is sufficient. One drawback of this symbol's simple tree structure is the possibility of finding false positives in noise. In most cases false-positives can be filtered by taking into account the presence and trajectory of potential finger markers in past frames and neglecting the appearance of false positives in isolated frames.

Recent reactIVision development builds contain an improved plain finger tracking component, without the need of the described finger symbol sticker. This layer is fully taking advantage of information already provided by the segmenter in order to identify and track fingertips that are touching the table surface, at no significant additional computational cost. This additional plain object tracking also provided a method of double-checking objects that have been lost by the fiducial tracking core, which significantly improved the overall tracking robustness. Due to the relatively recent addition to the code base, this feature along with formal comparative results on its performance will be made available together with a future reactIVision release.

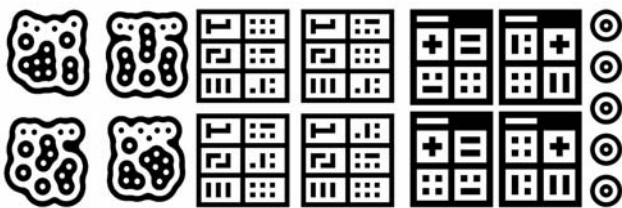


Fig. 2: symbols a) amoeba b) classic c) d-touch d) finger

Classic Engine

The "classic" fiducial tracking engine uses the original d-touch fiducial set (figure 2b) and geometry evaluation scheme while its code-base has been re-implemented from scratch. The dice shaped fiducial symbols can represent 120 different identities that are obtained by permutations of the positions of regions with two to six sub-regions. The primary region with a single sub region is used for the determination of the rotation angle and is therefore always placed in the upper left corner of the symbol. As already mentioned above, the dice symbols do not optimally use the available space and the calculation of the fiducial center point and rotation angle is not as accurate as with the amoeba set.

D-Touch Engine

The original d-touch code was eventually released publicly under the GPL license and has since been integrated into reactIVision. Although D-Touch can use a variety of different topology based symbol sets, including the original dice set used by the classic fiducial tracking engine, the implementation embedded in reactIVision uses a reduced subset of the dice style symbols with 24 permutations of regions with one to four sub-regions. The extra region needed for angle calculation is a single empty box on top of the symbol, which occupies less space than the main code regions.

HOW TO BUILD A TABLE INTERFACE

Table Construction

The design of a table depends on both general application requirements and the installation environment. For a musical instrument the table needs to be mobile and easy to assemble and disassemble, for public installation the table needs to be robust and accessible. In many cases a standard glass table might be sufficient for building a first prototype. Apart from the general structure, the table's most important component is its surface.

Whether used with or without projection, it is recommended that the table's surface be semitransparent, such as sanded glass or Plexiglas with a blurring coating. One simple way to achieve a blurring surface is to place a sheet of ordinary tracing paper on the table. The reason a blurring surface is desirable is that on transparent surfaces objects can be tracked above the table until the image loses focus, sometimes leading to unpredictable detection results. It is usually desirable that the objects are detected only when they are in contact with the table's surface, such that they disappear from the camera's view immediately when lifted. In addition to improving sensor behavior a semitransparent surface serves as an ideal projection screen for projected visual feedback, which in many cases is needed for table-based tangible user interfaces.

Camera & Lens

reactIVision in general will work with any type of camera and lens. Most of the better quality USB or FireWire webcams with a resolution of 640x480 at 30fps will be sufficient. For larger tables, industrial grade USB2 or FireWire cameras provide higher resolutions and frame rates. If DV or video cameras are to be used, they need to support full frame mode, since an interlaced video signal completely destroys the structure of fiducial symbols in motion.

When working with any computer vision system, the overall recognition performance is strongly dependent on the source image quality. Image quality results from a combination of various factors, which include the camera sensor, the lens quality, the illumination and other important camera and lens settings. In general we have

found that cameras with CCD sensors provide much better overall image quality than CMOS sensors. Cameras with an exchangeable lens mount are to be preferred. To decrease the minimum distance to a sufficiently large surface the system needs to use wide-angle lenses. The necessary focal length of the lens can be calculated as a function of the sensor size, the distance to the surface and the diameter of the viewable area of the surface. Be aware that some consumer grade “wide-angle” lenses may not focus consistently across the full viewing area which can have detrimental effects on tracking performance.

To set up the best image quality obviously requires that the lens is focused. A simple focusing procedure is to fully open the iris and then try to achieve the best possible focus. After that the iris can be slowly closed until a perfectly sharp image is achieved. In addition to focus, the camera's shutter speed needs to be fast enough to avoid motion blur, since long exposure times will cause blurry images of moving fiducial symbols, making them more difficult or impossible to recognize. Both narrower iris and faster shutter speeds result in less light reaching the sensor, which needs to be compensated by stronger illumination. Low lighting levels can also be corrected slightly by increasing the sensor gain, although too much gain will decrease the image quality by introducing grainy noise.

Illumination

In a camera-projector system the two visual components need to operate in different spectral bands so they do not interfere with each other. Since the projector obviously needs to operate in the visible range, the camera has to work in the infrared (IR) spectrum only. CCD camera sensors are perfectly sensitive to infrared light, but most of the time are protected with an IR filter which needs to be removed from the sensor housing or lens. At the same time the table setup needs to be illuminated with strong and diffuse IR light, which is completely invisible to the eye and therefore does not interfere with the table projection. Suitable light sources are IR LED arrays which are available in different intensities, alternatively one could use halogen lights, which produce a lot of IR but need to be equipped with IR pass filters which can be purchased in any photography shop. These IR pass filters also need to be applied to the camera in order to filter all visible light, most importantly from the projection, since the projected image would otherwise overlay and interfere with the fiducial symbols. In the case where no projection is required, the setup can operate in the visible spectrum, significantly simplifying the illumination process.

Mirrors and lens distortion

If a camera or projector does not have a sufficiently wide-angle lens, placing a mirror into the table helps to achieve a larger active surface while maintaining a relatively low table height. Unfortunately mirrors as well as wide-angle

lenses produce distorted images both for the projection and the camera image.⁴ reactIVision comes with a built-in calibration component which offers a simple mechanism to correct these distortion errors. In the case of projection the image needs to be pre-distorted by applying the image as a texture onto a virtual surface in order to again appear straight on the table surface. The TUIO distortion example code provides a simple graphical feedback component with built-in distortion engine. Both distortion components, the reactIVision sensor application as well as the application providing the visual feedback, need to be calibrated in order to match the physical object position with the virtual projection position. See the usage section below for more details on the calibration process.

Computer Hardware

The rest of the hardware can be built from standard off-the-shelf components. In many cases a modern dual-core computer will be more than sufficient to handle both the computer vision component along with the actual tangible interface application. For self-contained table setups a laptop or small shuttle PC might be the right choice if everything needs to fit inside the table. The projector usually resides underneath the projection surface pointing at a mirror on the table's bottom edge, therefore a small form-factor combined with a strong lamp and an appropriate wide-angle lens are its most important features. Since projectors can produce a considerable amount of heat, appropriate ventilation must be assured to avoid overheating within the table.

Tangibles

Almost any object, including simple wooden or plastic geometric shapes, everyday objects or artifacts, and even food or vegetables can be turned into an easily trackable tangible interface component by attaching a fiducial marker to it. Ideally the symbol has to be attached on the bottom side of the object in order to hide it from the user's attention and also to avoid possible hand occlusion problems. The fiducial symbol set can be printed with a laser printer onto ordinary white office paper. Gray recycled paper is less desirable as it tends to degrade symbol contrast. Some ink-jet inks are invisible in the infrared domain and therefore unusable for IR illuminated setups, although such ink can be used to add additional user-readable color codes to the symbols that stay invisible to the computer vision component. In order to protect the symbols from scratches, and color loss, the printed paper surface can be coated with transparent adhesive foil, which also simplifies cleaning of the symbol's surface from dirty spots that can degrade recognition.

⁴ Note that here we are referring to spatial warping rather than inconsistent focus across the image, which cannot be corrected.

FRAMEWORK USAGE

reactIVision application handling

The main reactIVision application only provides a very simple GUI showing the actual camera image and some visual feedback on the fiducial detection performance. It is generally configured by calling the application with various command line options at startup and can be controlled with some key commands during execution.

Startup options include the configuration of the following features. See the documentation that comes with the application for more details.

- Distortion mode and calibration file
- Fiducial engine alternatives
- TUIO host name and port number
- Optional MIDI transport and configuration file
- Parameter inversion (when using mirrors)

During runtime the following features of the reactIVision application can be controlled using simple key commands.

- Switch to calibration mode
- Change the display type
- Verbose on screen and console feedback
- Application control: pause and exit

Distortion calibration procedure

This section briefly explains the calibration procedure using the reactIVision sensor application in conjunction with the TUIO distortion example, made available by Marcos Alonso as part of the reactIVision framework. This example application can be extended to take advantage of its distortion correction functionality.

In the calibration folder that comes with the application package there are two calibration sheet examples for rectangular and square table setups, which can also be used for round tables. Print the desired document scaled to match the size of your visible table surface and place the sheet onto the table with the calibration grid facing downwards.

Start the TUIO_Distort application and switch to calibration mode by hitting the 'c' key. Using the keys 'a,w,d,x' adjust each vertex on the projected grid to match the vertices on the sheet. You can navigate between vertices using the cursor keys. After finishing this first calibration step you can switch the TUIO_Distort application to normal mode by hitting the 'c' key again while leaving the calibration sheet untouched in its positions on the table. Now start the reactIVision application in distortion mode by providing a grid file with the '-g' option. Once started, switch into calibration mode by hitting the 'c'. In the same ways as for the calibration procedure of the projected graphics, you now need to adjust each vertex to match the grid on the sheet by using the keys mentioned above. After finishing this second calibration step and exiting the calibration mode by hitting the 'c' key, both applications will be

synchronized and the projected visual object feedback should exactly match the physical object positions. You can see a preview the resulting image distortion within reactIVision by hitting the 'g' key.

Application programming

All of the TUIO client examples for standard object oriented programming languages, such as C++, C#, Java and Processing implement an event based callback mechanism that notifies registered classes when objects are added, moved or removed from the table. The same events are generated for (finger tracking) cursor operations.

In general, application logic has to implement the *TuioListener* interface, which defines various callback methods such as *addTuioObj()*, *updateTuioObj()* and *removeTuioObj()*. These methods are called by the framework-supplied *TuioClient* class, which derives events from the continuous stream of status information received from the sensor application. The *TuioClient* class has to be instantiated and started using the *connect()* method at the beginning of the session. It is also necessary to register all *TuioListener* classes that need to be notified by the *TuioClient* using the *addTuioListener()* method. The *TuioClient* operates in its own thread in the background until it is terminated using the *disconnect()* method.

For environments such as PureData or Max/MSP, TUIO client objects are provided that decode events from the TUIO protocol stream and provide them to the environment via appropriate messages.

EXAMPLE PROJECTS BASED ON REACTIVISION

reactTable

This table-based instrument has been the driving force for the development of the reactIVision framework, since the reactTable's real-time music interaction and expressivity demand very high performance and recognition robustness from the sensor component. The physical artifacts on the reactTable surface allow the construction of different audio topologies in a kind of tangible modular synthesizer or graspable flow-controlled programming language. Several users can simultaneously manipulate various objects and control additional features with finger gestures. The reactTable web documents the various instrument features in greater detail.⁵

recipe-table

This project, which was shown during the Ars Electronica Festival 2005, has been developed by a group of students within the Interface Culture Group at the University of Art and Industrial design in Linz. The recipe table is a fully working prototype of a future kitchen environment, where food and food products placed onto an interactive surface

⁵ <http://mtg.upf.edu/reactable/>

are detected and the system suggests a series of possible recipes that can be cooked with those ingredients. Changing the ingredients position in relation to each other allows the user to navigate within the possible recipes according to his or her personal preferences. reactIVision has been used to identify and track the labeled products, simulating a barcode tracking system. In the near future such an environment could identify and track RFID labels that will soon be incorporated into standard consumer products. Further information about this intelligent environment and its creators can be found on the project web page.⁶

Blinks & Buttons

Blinks is a table-top interactive installation by the German artist Sascha Pohflepp, where projected photos are distributed on an interactive surface. Moving a glass prism over a photo causes it to refract the light to the sides of the table. This light contains projections of other photos taken at exactly the same moment in other locations. The user can browse the image collection over time. reactIVision has been used to track the prism controller in conjunction with the Processing application. You can find more information about this installation at the project's web site.⁷

FUTURE WORK

The reactIVision framework is still being actively developed and the existing code-base will be improved and new features added. An important improvement in the next release will be the inclusion of the plain finger tracking layer, which doesn't require fiducial stickers on the fingertips. We are also planning to include additional fiducial engines such as ARToolkit, Barcodes and Semacode decoding into reactIVision. Video acquisition under Linux needs to be extended to support a wider range of cameras, eventually by incorporating the promising unicap library⁸, which provides a uniform camera access method for Linux operating systems.

ACKNOWLEDGMENTS

The authors would like to thank the Music Technology Group at the Universitat Pompeu Fabra for supporting the development of this publicly available software, as well as the rest of the reactTable team, Sergi Jorda, Günter Geiger and especially Marcos Alonso who support and contribute to this framework in various ways. We would also like to thank the numerous reactIVision users for their suggestions and encouragement. Last but not least we are grateful for the initial support of Enrico Costanza by making the development of this framework possible with his earlier D-Touch contribution.

⁶ <http://www.recipetable.net/>

⁷ <http://blinksandbuttons.net/>

⁸ <http://unicap-imaging.org/>



Fig. 3: examples a) reactTable b) recipe-table c) blinks

REFERENCES

1. Bencina, R. & Kaltenbrunner, M. "The Design and Evolution of Fiducials for the reactIVision System", Proceedings of the 3rd International Conference on Generative Systems in the Electronic Arts (3rd Iteration 2005), Melbourne (Australia)
2. Bencina, R. & Kaltenbrunner, M. & Jordà, S. "Improved Topological Fiducial Tracking in the reactIVision System", Proceedings of the IEEE International Workshop on Projector-Camera Systems (Procams 2005), San Diego (USA)
3. Costanza, E. & Shelley, S. B. & Robinson, J. "D-touch: A Consumer-Grade Tangible Interface Module and Musical Applications". Proceedings of Conference on Human- Computer Interaction (HCI03), Bath (UK)
4. Han, J. Y. "Low-cost multi-touch sensing through frustrated total internal reflection", Proceedings of the 18th annual ACM symposium on User interface software and technology 2005, Seattle (USA)
5. Jordà, S. & Kaltenbrunner, M. & Geiger, G. & Bencina, R. "The reactTable*", Proceedings of the International Computer Music Conference (ICMC2005), Barcelona (Spain)
6. Kaltenbrunner, M. & Bovermann, T. & Bencina, R. & Costanza, E. "TUIO - A Protocol for Table Based Tangible User Interfaces", Proceedings of the 6th International Workshop on Gesture in Human-Computer Interaction and Simulation (GW 2005), Vannes (France)
7. Kato, H. & Billingham, M. & Poupyrev, I. & Imamoto, K. & Tachibana, K. "Virtual Object Manipulation on a Table-Top AR Environment", Proceedings of the International Symposium on Augmented Reality (ISAR 2000), Munich (Germany)
8. MIDI Manufacturers Association, <http://www.midi.org/>
9. Ullmer, B. & Ishii, H. "Emerging Frameworks for Tangible User Interfaces", In J. M. Carroll, editor, Human-Computer Interaction in the New Millennium, pp. 579-601, 2001
10. Wright M. & Freed, A. & Momeni, A. "OpenSound Control: State of the Art 2003", Proceedings of the 3rd Conference on New Interfaces for Musical Expression (NIME03), Montreal (Canada)