# A Gesture Control Interface for a Wave Field Synthesis System

Wolfgang Fohl
HAW Hamburg
Berliner Tor 7
20099 Hamburg, Germany
fohl@informatik.haw-hamburg.de

Malte Nogalski
HAW Hamburg
Berliner Tor 7
20099 Hamburg, Germany
malte.nogalski@haw-hamburg.de

## ABSTRACT

This paper presents the design and implementation of a gesture control interface for a wave field synthesis system. The user's motion is tracked by an IR-camera-based tracking system. The developed connecting software processes the tracker data to modify the positions of the virtual sound sources of the wave field synthesis system. Due to the modular design of the software, the triggered actions of the gestures may easily be modified. Three elementary gestures were designed and implemented: Select / deselect, circular movement and radial movement. The gestures are easy to execute and allow a robust detection. The guidelines for gesture design and detection are presented, and the user experiences are discussed.

## Keywords

Wave field synthesis, gesture control

## 1. INTRODUCTION

For immersive virtual reality systems, there is a need for direct and natural interaction. One of these interaction types is gesture control. This has been the initial motivation to develop a gesture control interface for the control of movement of virtual sources in the *wave field synthesis* (WFS) system, that the authors are operating. The lab is equipped with a camera-based tracking system (A.R.T.), that permits the tracking of marker positions with a tracking rate of max. 60 Hz and spatial resolution of 0.4mm, which is used for the realization of a gesture control interface [1].

### 1.1 Goals

The primary goal of the work presented in this paper has been to develop a software for interfacing the WFS system with external tracking systems to provide gesture control of the WFS system. To this end, a set of elementary geometrical data transformations between tracking and WFS systems had to be developed. The resulting interface software should be modular and flexible so that it could be easily adapted to other tracking systems and also enables the definition of different gesture sets. Finally, a set of easy-to-apply gestures should be developed to move the positions of virtual sound sources. The gestures trigger the transmission of Open Sound Control (OSC) messages to the WFS system, that finally modifies the source positions.


*NIME'13,* May 27 – 30, 2013, KAIST, Daejeon, Korea.


### 1.2 Background and Related Work

#### *1.2.1 Wave Field Synthesis*

A wave field synthesis system creates the two-dimensional representation of the sound field of a virtual source by exploiting Huygens' Principle which states, that the sound field of a point source can as well be generated by oscillators located along the wave front of the source. A modified version of Huygens' principle applies for linear speaker arrays. Here a delayed version of the source sound signal is fed to the speakers. The delay $\Delta t_i$ of speaker $i$ is the time a sound wave with velocity $c$ would need to pass the distance $d_i$ from the source to speaker $i$:

$$\Delta t_i = \frac{d_i}{c} \tag{1}$$

A very detailed description of wave field synthesis fundamentals, applications, and rendering software is given in the PhD thesis of Marije Baalman [2]. A shorter overview on WFS principles and spatialisation techniques can be found in the article of Corteel and Caulkins [5].

#### *1.2.2 Gesture Control Approaches in Wave Field Synthesis, Music Production and Performance*

Gesture control of WFS systems has gained some attention and is covered in several papers. Gestures are either used for the production of audio material or for live performances in an virtual audio environment. Bredies et al. describe a multitouch table to manipulate sound sources [3], Melchior, Laubach, and de Vries developed a virtual reality authoring tool [10]. In the *Grainstick* project by Leslie et al. [9], a 3D IR tracker is combined with a Wiimote to manipulate sound sources and trigger various sounds during the performance. A similar setup is used by Caramiaux et al. to create an "extended virtual musical instrument" [4].

### 1.3 Paper Outline

In the next section an overview of the WFS system hardware and software components as well as the system architecture is given. Then the repertoire of gestures is presented. After that, the design and implementation details of the gesture control interface is described and the used libraries and frameworks are mentioned. The article closes with a discussion of the results and a preview on future work.

## 2. SYSTEM DESCRIPTION

Our WFS system is designed and developed by FourAudio[1]. It consists of a rectangular assembly of linear speaker arrays. The array is made up of modules, each module containing 8 tweeters and 2 woofers. There is a total of 56 speakers

[1]Four Audio homepage: `http://www.fouraudio.com`

on the long sides of the rectangle, and 48 speakers on the short sides, resulting in 208 channels. Each channel has to be supplied with its individual audio signal. The rendering, i.e. the application of the necessary time delays to the audio signals of the up to 64 virtual sources is performed by a distributed system of 3 Linux PCs and an Mac frontend computer. The computers are connected by two Ethernet networks, one for passing the necessary control and coordination messages by OSC messages, the other network connects the speaker modules and the rendering computers by an proprietary audio network (Dante by Audinate[2]). Details about the network architecture are given below.

A detailed system description can be found in a recently published article [6].

Figure 1 gives an overview over the system structure and the distribution of the software components. The audio processing system consists of one Linux PC as the *WFS control server* and two Linux PCs as *WFS rendering nodes*, and a Mac PC as frontend computer, where the gesture control software is running, and where DAW software like Ardour or Cubase is providing the audio streams for the system. On the Mac, the software audio interconnections between the applications is established by the low-latency audio server JACK[3]. The external audio connections between the frontend computer, the rendering nodes, and the loudspeaker channels are made with the Dante audio network by Audinate, an audio-over-Ethernet solution, that minimizes the necessary wiring of the system and provides flexible signal routing. The overall system engineering and the design of the speaker cabinets was done by Four Audio.

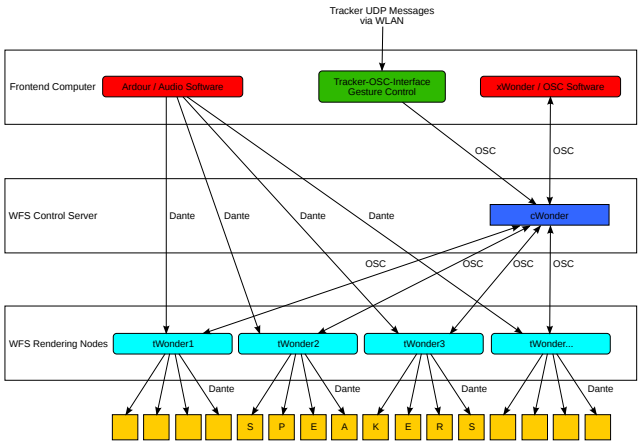


**Figure 1: Structure of the WFS system [7]**

The software for WFS rendering is called *wonder*. It is being developed at the TU Berlin in the department of audio communication[4], and is licensed under the GPL. It is a distributed software system consisting of the components *cwonder* (the controller and coordinator), running on the WFS control server, and the component *twonder*, which applies the proper time delays to the loudspeaker signals, running on the WFS rendering nodes.

The control server coordinates the work of the rendering nodes and keeps track of the current source positions, and provides these data to the GUI component *xwonder* running on the frontend PC. The control server, the rendering nodes, and the frontend PC are communicating via OSC (Open Sound Control) network messages[5]. OSC messages are also the external command interface of the system to manipulate source properties. An OSC message to move a source has the form: */WONDER/source/position(id, x, y, z, t, dur)*. This message moves the source identified by *id* to position (*x,y,z*). Movement starts at time *t* and lasts *dur* seconds.

Figure 2 shows the *xwonder* GUI.

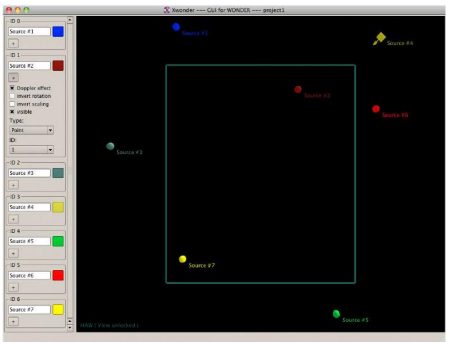

**Figure 2: The *xwonder* GUI to control virtual sources**

The A.R.T tracking system consists of six IR cameras and an evaluation unit that locates the positions of *markers* consisting of an unique assembly of reflecting spheres that can be mounted to the user's hand (see figure 3). As long as the marker is in sight of at least two cameras, the data of the marker position and orientation is provided via WLAN broadcast. The transmitted data contain the *position* of the origin of the hand coordinate system, as well as the *rotation matrix* $\boldsymbol{R}$ of the hand coordinate system relative to the room coordinate system, time stamps and marker ID.

The transformation between hand and room coordinate systems is accomplished by equation 2:

$$\boldsymbol{r_r} = \boldsymbol{R} \cdot \boldsymbol{r_h} + \boldsymbol{p_r} \tag{2}$$

where $\boldsymbol{r_r}$ is a source position in room coordinates, $\boldsymbol{r_h}$ is the same position in hand coordinates, and $\boldsymbol{p_r}$ is the position of the origin of the hand coordinate system in room coordinates.

It is worth noting, that R contains the cosines between the unit vectors of the room and hand coordinate system:

$$\boldsymbol{R} = \begin{pmatrix} \boldsymbol{u_{x,r}} \cdot \boldsymbol{u_{x,h}} & \boldsymbol{u_{x,r}} \cdot \boldsymbol{u_{y,h}} & \boldsymbol{u_{x,r}} \cdot \boldsymbol{u_{z,h}} \\ \boldsymbol{u_{y,r}} \cdot \boldsymbol{u_{x,h}} & \boldsymbol{u_{y,r}} \cdot \boldsymbol{u_{y,h}} & \boldsymbol{u_{y,r}} \cdot \boldsymbol{u_{z,h}} \\ \boldsymbol{u_{z,r}} \cdot \boldsymbol{u_{x,h}} & \boldsymbol{u_{z,r}} \cdot \boldsymbol{u_{y,h}} & \boldsymbol{u_{z,r}} \cdot \boldsymbol{u_{z,h}} \end{pmatrix} \tag{3}$$

Here, e.g. $\boldsymbol{u_{x,r}}$ is the unit vector in $x$-direction of the room coordinate system and $\boldsymbol{u_{z,h}}$ is the unit vector in $z$-direction of the hand coordinate system.

## 3. SOFTWARE ARCHITECTURE

In this section the software architecture of the tracker-OSC interface software is described.

The purpose of this software is to capture the outputs of the tracking system and transform it to source positioning commands for the WFS system.

For the following explanations please refer to figure 4.

From the architectural view, the system presented here is a framework for the interconnection of external position data to the WFS system. The central part of the framework is the *optional components* section in the center of figure 4, where the desired system behavior is implemented.

[2]Audinate homepage: `http://www.audinate.com`
[3]Jack homepage: `http://jackaudio.org`
[4]Homepage of the department of audio communication: `http://www.ak.tu-berlin.de/menue/fachgebiet_audiokommunikation`
[5]OSC Homepage: `http://opensoundcontrol.org/`

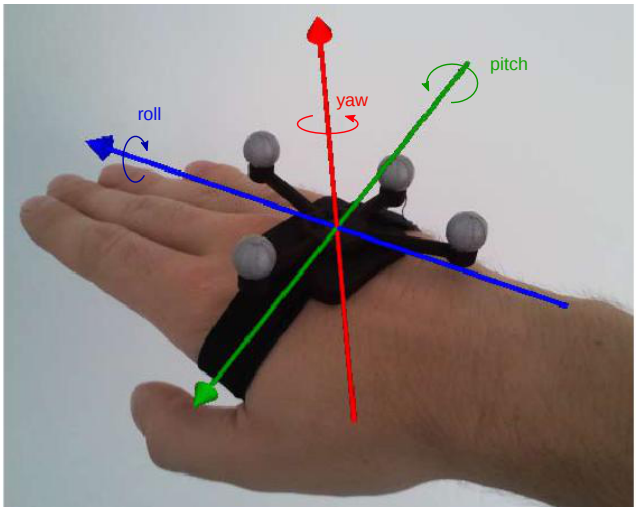


**Figure 3: 3D-marker mounted to user's hand for gesture detection. The hand coordinate system is indicated: x-axis (blue), y-axis(green), z-axis (red). Also indicated are the rotational directions *roll*, *pitch*, and *yaw*.**

The main components of the software are the *TrackerListener*, which captures the UDP packets from the tracking system, the *Matcher*, which updates the *Marker* data objects with the received tracking data and the optional component section . The marker objects internally represent the tracked real-world objects. The *Matcher* also creates a mapping from *Marker* objects to *Source* objects, where *Source* objects are the internal representations of the virtual sources of the WFS system . Thus all desired changes to a *Source* of the WFS system are to be made to the corresponding *Source* within the tracker-OSC-interface software and all changes made to a *Source* within this interface software will result in a change to the corresponding *Source* of the WFS system. Once this mapping is established, all subsequent processing is made with the *Source* objects. Since the *hand* marker (figure 3) is mapped to a *Source* object, it will be visualized in the xwonder GUI. (Usually, no audio stream is connected to the hand source.)

The source data are then passed to a *ConverterBox*, that contains one or more *Converters* to perform the necessary coordinate transformation from the tracker coordinate system to a self-defined coordinate system. It follows an optional processing step, where the application-specific logic is applied, this is where the *GestureComponent* is located which will be explained in the next chapter.

The *GestureComponent* requires the two coordinate systems of the tracker and the WFS system to fit onto each other, since both systems occupy the same space in the real world. For purposes different from gesture control, the architecture would allow users to modify the configuration of Converters during a live performance. To minimize the coupling between the two coordinate systems and to give programmers of additional optional components (as mentitioned briefly in the previous paragraph) the freedom to choose an entirely different coordinate system within their component, a second pass through a *ConverterBox* is applied after the application-specific logic, thus completing the fit of the coordinate systems. Possible *Converters* are: *Mirror*, *Scale*, *Offset*, and *Rotate*.

After this second pass through a *ConverterBox*, the *Source* objects are passed to the *ModificationFactory* module, where the movement history of the sources is stored. To reduce

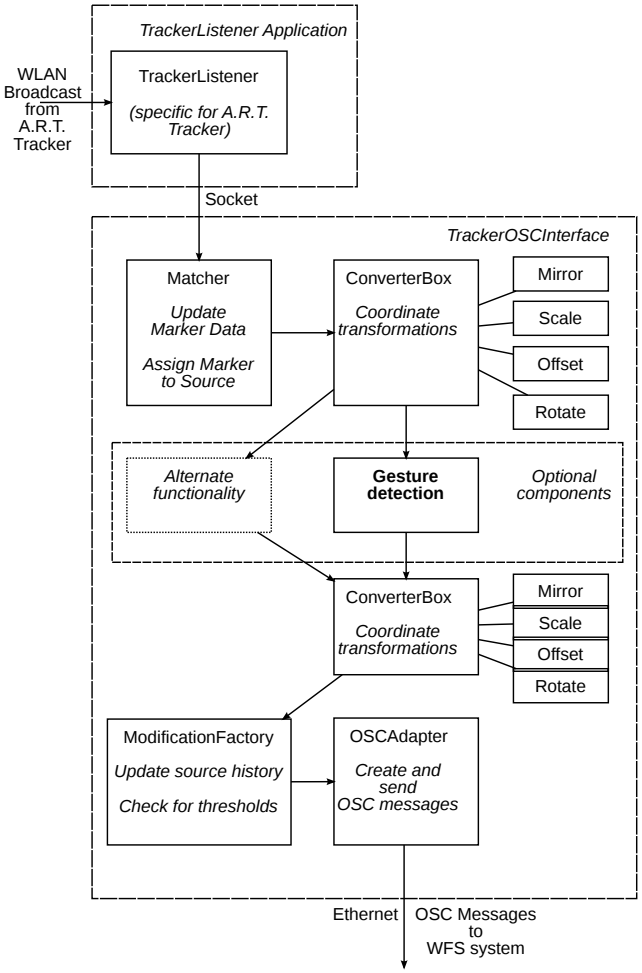


**Figure 4: Processing sequence of the Tracker-OSC interface software**

network and processing load, OSC messages are only created, if the source movement is large enough to trigger a perceivable change in the WFS system. Finally the *OSCAdapter* is creating the OSC messages and tansferring them to the WFS system.

## 4. GESTURES

The purpose of the gestures is the movement of virtual sound sources of the WFS system. A very basic set of gestures is defined: *Selection* and *deselection* of a source, *circular movement* around the user, and *radial movement* towards and away from the user.

A robust recognition of the gestures was required with special attention to ease of operation even for untrained users. The gestures for source movement are to provide a continuous control of the source position. This sort of tasks is best accomplished by an deterministic evaluation algorithm instead of a learning system, because of the wide variety of possible control actions, which in a learning system had to be learned for each desired path of movement and for each user. This would result in an impracticable size of the training data. So in a previous study, several test persons were asked to simulate the tasks of selecting, deselecting and moving sound sources, while explaining the sound sources as realworld objects they could not see but only hear. This was to help to lift the test persons' attention away from the speakers or displays and towards the virtual world around them, since the interface should allow

them to interact with only what they hear. The results of this study led to the present gesture definitions.

An additional goal was to minimize the instrumentation requirements, so only one marker should be used to detect the gestures and calculate the source movements. The marker is mounted at the user's hand as shown in figure 3.

A further goal for the gesture definitions was to avoid the *King Midas problem*: the discrimination between gesture and non-gesture must be intuitive to the user, and must be easy to detect in the data processing.

The procedure of manipulating sources is illustrated in the finite state machine in Fig. 5. The basic sequence is to first *select* a source, then perform one or more *move* operations on the source, then finally *deselect* the source.

The hand is initially in the *FREE* state. To select a source, the user has to take the *selecting pose* as described in the next section. When the selecting pose is detected, the hand proceeds to the *SELECTING* state, and if the pose is held for a sufficiently long time, it reaches the *SELECTED* state. The purpose of the intermediate *SELECTING* state is to suppress the unintended selection of sources. After the source has been selected, one of the two possible movement gestures will be detected which brings the hand to the *CMOVE* state for circular movements around the hand with a fixed distance to the hand, or to the *RMOVE* state for movements towards or away from the user. When the end of a movement is detected, i.e. the hand has not moved for a certain time, the hand goes back to the *SELECTED* state, and finally, if the selecting pose has been left, back to the *FREE* state.

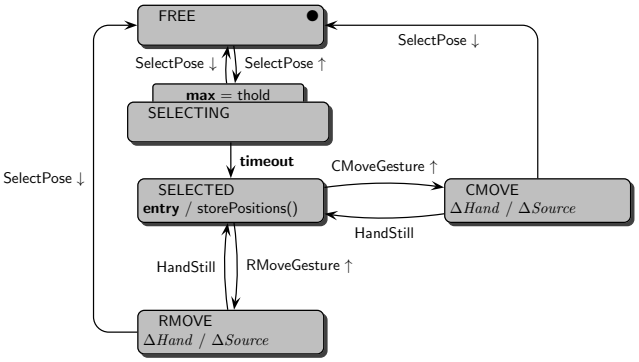


**Figure 5: State diagram of the hand**

The gestures and their detection will be explained in detail in the following sections.

For the discussion of the hand movements, the axes of the hand coordinate system and the rotational directions are defined as shown in figure 3: The *x-axis* is pointing in the directions of the fingers, the *y-axis* is pointing in the direction of the right thumb, and the *z-axis* is pointing upward from the back of the hand. The rotational directions are *roll*: a rotation of the hand around the x-axis, i.e. turning the back of the hand upwards / downwards, *pitch*: rotation around the y-axis, i.e. pointing to the ceiling / to the floor, *yaw*: rotation around the z-axis, i.e. pointing to the left / right. In the following equations, the subscript $h$ and $r$ denote the hand and room coordinate systems respectively.

## 4.1 Select / Deselect Sound Sources

To *select* a source, the *selecting pose* must be taken. The fingers point towards the source, and the hand is raised, i.e. the pitch angle exceeds a threshold value. In addition, the back of the hand has to put upwards, so the roll angle must not exceed a threshold value. This pose must persist a certain time to avoid unintended selection, then the hand is in the *SELECTED* state. The pitch and roll angles can easily be obtained from the rotation matrix $\boldsymbol{R}$:

$$pitch = \arcsin r_{31} \ , \ roll = \arcsin r_{32} \tag{4}$$

When a source has been selected, the *source position at selection*, *hand position at selection*, and the *hand orientation at selection* are stored for the calculations of the source displacement by the subsequent movement gestures. A *tolerance* rectangle of size $2\Delta x \times 2\Delta y$ is constructed in x- and y-direction, and the next hand movement leaving this rectangle decides, which action is taken next: when the yaw angle exceeds a threshold value, a *circular movement* is initiated, and if the tolerance rectangle is left in x-direction (back / front), a *radial movement* is started. The gestures for these movements are described in the following sections.

To *deselect* a source, the selecting pose must be left by lowering the hand. The pitch angle must fall below a lower threshold. There is a hysteresis between the thresholds for selecting and deselecting, otherwise a source just selected will be deselected by the slightest downward shake of the hand. Deselection is executed immediately after the selecting pose has been left. Deselection of a source is possible at any time during a movement and will stop the movement.

## 4.2 Circular Movement

When the hand moves sideways in the *SELECTED* state, the hand proceeds to the *CMOVE* state, and a circular movement is introduced. When entering the state, the desired source positions are calculated from the distance vector $\boldsymbol{s_h}$ from the hand to the source in hand coordinates, which has been stored at the entry into the *SELECTED* state. This vector remains constant throughout the movement, so only a transformation of the source position between the hand and room coordinate systems has to be made to calculate the new source position $\boldsymbol{s_r}$ in room coordinates. For this transformation the hand rotation matrix $\boldsymbol{R}$ and the hand translation in (room coordinates) $\boldsymbol{\Delta p_r}$ must be considered:

$$\boldsymbol{s_{r,circular}} = \underbrace{\boldsymbol{R} \cdot \boldsymbol{s_h}}_{\substack{\text{rotation in} \\ \text{room coordinates}}} + \boldsymbol{\Delta p_r} \tag{5}$$

Since the hand translation $\boldsymbol{\Delta p_r}$ is considered in the calculation of equation 5, no matter how the circular gesture is performed, be it with the elbow as the centre of motion, or by a rotation of the whole body, the result is in both cases as expected: the source is rotating around the centre of the gesture movement. This is even true, if the user is walking across the room. Figure 6 illustrates the geometry of the circular movement.

The circular movement ends, when the hand remains still for a certain amount of time. Then the hand switches to the *SELECTED* state. A slight problem arises, when the circular movement shall be continued, because the hand has to leave the tolerance zone which causes a small jump in the source position, but this jump can be immediately corrected by a subsequent hand movement. As previously stated, deselecting the source also ends the movement.

## 4.3 Radial Movement

When the hand moves in radial direction (towards or away from the user) in the *SELECTED* state, it gets into the *RMOVE* state, and the source begins to move with a speed proportional to the distance between the current hand position and the hand position at selection. The direction of source displacement *in room coordinates* is constant, and

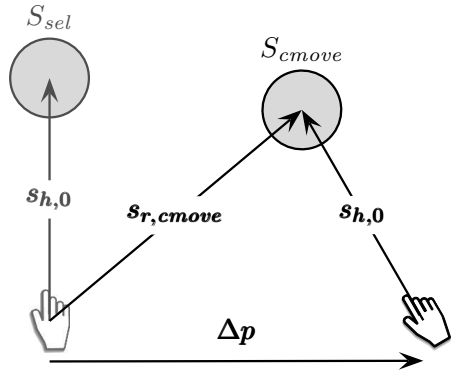


**Figure 6: Calculation of the source position for circular movement**

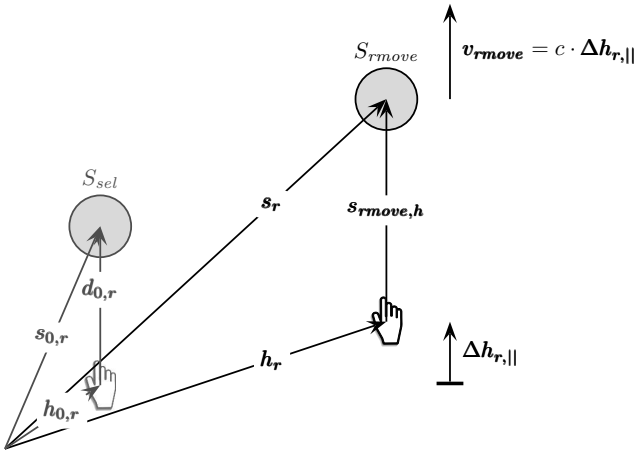


**Figure 7: Calculation of the source position for radial movement. The source moves in the direction of the connecting line between hand and source at the time of selection. The speed is proportional to $\Delta h_{r,||}$, which is the projection of the hand displacement on the direction of movement.**

is along the vector $\boldsymbol{d_{0,r}} = \boldsymbol{s_{0,r}} - \boldsymbol{h_{0,r}}$ (see figure 7). The current source position $\boldsymbol{s_{r,radial}}$ is calculated according.

$$\boldsymbol{s_{r,rmove}} = \boldsymbol{s_{0,r}} + C \cdot \boldsymbol{\Delta h_{r,||}} / \Delta t_{\mathrm{OSC}} \qquad (6)$$

where $\boldsymbol{\Delta h_{r,||}}$ is the hand displacement projected onto the line of movement $\boldsymbol{d_{0,r}}$, $C$ is a constant to adjust the source speed, and $\Delta t_{\mathrm{OSC}}$ is the time since the last OSC message has been sent to the WFS system.

For movements away from the user, the motion only stops, when the *RMOVE* state is left by moving the hand to *hand-position-at-select*, for movements towards the user, the motion stops, when the source reaches *hand-position-at-select*. Of course, the movement also ends, when the source is deselected. It should be noted, that after leaving the *RMOVE* state, the tolerance rectangle is *not* centred around the current hand position, but stays centred around the initially stored position *hand-position-at-select*. Otherwise the fine adjustment of a source position with successive *RMOVE* operations will exhibit a "'jumpy"' behaviour, since each time after the motion has stopped, the hand would have to be moved for the distance $\Delta x_h$.

# 5. IMPLEMENTATION OF THE GESTURE CONTROL SOFTWARE

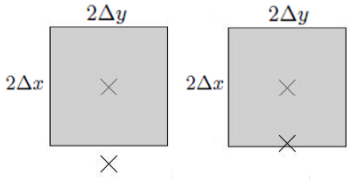


**Figure 8: Tolerance rectangle for radial movement. Left: radial movement active, right: radial movement stopped, because hand is within tolerance rectangle, movement will be continued, when hand leaves tolerance rectangle.**

The software has been implemented in Java 1.6, it consists of two separate applications: The *TrackerListener* and the *TrackerOSCInterface* (see figure 4). The *TrackerListener* collects the messages of the A.R.T. tracking system and forwards them to the *TrackerOSCInterface*. Thus the software can be adapted to other tracking systems by modifying only the *TrackerListener* application, without the necessity to modify the code of the *TrackerOSCInterface*. For capturing the WLAN-messages from the tracker, only the *java.net.DatagramPacket* and *java.net.DatagramSocket* core libraries were used. For the OSC message passing, the *JavaOSC* library by Illposed Software [8] is used.

# 6. RESULTS AND DISCUSSION

The software presented in this article provides a flexible and reliable tool to couple the 3D-tracking system with the WFS system to create a gesture control interface for the latter system.

To this point the gestures do not allow a user to create or modify sounds, so a predefined sound project must be running. The user, instead of being a musician, rather takes on the role of a bandleader or conductor of some sort.

The software is not limited to gesture control, but allows a variety of source manipulations by marker movements. Only the *GestureControl* module in figure 4 must be replaced with a component with different functionality. An example is the *SourceSourceMatcher*, which allows WFS sources to be continuously positioned in a fixed direction and distance to a marker, this creates a "virtual headphones" behaviour, moving two WFS sources to the left and right side when the marker is mounted on the user's head.

A gesture processing module enables users to select a source, move it in circular and radial direction, and finally deselect the source.

The preliminary user study showed great agreement of the test persons with the execution of the select / deselect and the circular motion gestures. For these gestures the test persons also agree in their expectations of the resulting actions. In the implemented system, these gestures were executed by the users without problems.

For the radial motion gestures, the test persons showed greater variance in how to execute the gesture and in the expectation of the resulting action. For the definition of the radial motion gesture, a velocity-based control has been implemented to enable the user to easily displace the source over great distances. In the implemented system, users occasionally experienced problems controlling radial motion, especially when pulling the source nearer. To some extent, the problems are due to the fact, that the acoustical perception of the source position in radial direction is quite difficult. These difficulties even increase, when the source

approaches too near at a too high speed. Also, users occasionally move too far away from the position, where the gesture started, which leads to unintended source motions. Different modifications of the radial movement will have to be implemented and tested, e.g. reducing the speed of approach with decreasing distance of the source to the user.

To actually implement the gesture controlling software, tedious work had to be done to fine-tune the tolerance and threshold values for interpreting hand movements as gestures. Finally, a compromise has been found between accepting different styles of the users to perform the gestures, and the suppression of unwanted actions, due to incident movements that erroneously are being interpreted as gestures by the system.

So with exception of some minor issues with the radial motion gestures, the project goals were reached.

## 6.1 Future Work

After having learned important lessons on defining and detecting gestures and their triggered actions, the next step will be to expand the repertoire of gestures and to develop different alternatives to the current approach to the radial movement of the sources, and to create a musically meaningful project to gather experiences with the gesture control system at real musical performances.

Also the whole concept of non-acoustic feedback has not yet been considered. Next approaches would consider haptic feedbacks onto the hand used for the gesture, with the aim not to increase the abstraction level too much while increasing the intuitiveness when working with the system. Transforming low frequencies of the sound emitted by a selected source to vibrations onto the hand might even help to feel which source is selected, if two sources are very close to each other.

# 7. REFERENCES


[1] A.R.T. ARTTRACK user dokumentation. http://www.ar-tracking.com/products/tracking-systems/arttrack-system/, accessed 2013-02-10.

[2] M. A. Baalman. *On Wave Field Synthesis and electro-acoustic music, with a particular focus on the reproduction of arbitrarily shaped sound sources*. PhD thesis, TU Berlin, 2008.

[3] K. Bredies, N. A. Mann, J. Ahrens, M. Geier, S. Spors, and M. Nischt. The multi-touch SoundScape renderer. In *Proceedings of the working conference on Advanced visual interfaces*, AVI '08, page 466–469, New York, NY, USA, 2008. ACM.

[4] B. Caramiaux, S. F. Alaoui, T. Bouchara, G. Parseihian, and M. Rébillat. Gestural auditory and visual interactive platform. In *Proceedings of the 14th International Conference on Digital Audio Effects (DAFx-11)*, Sept. 2011.

[5] E. Corteel and T. Caulkins. Sound Scene Creation and Manipulation using Wave Field Synthesis. Technical report, IRCAM, Paris, Paris, 2004.

[6] W. Fohl. The wave field synthesis lab at the HAW Hamburg. In R. Bader, editor, *Sound - Perception - Performance*, volume 1 of *Current Research in Systematic Musicology*. Springer, 2013 (in print).

[7] FourAudio. *WFS System Manual*, 2011.

[8] Illposed Software. Java OSC. `http://www.illposed.com/software/javaosc.html`, accessed 2013-02-09.

[9] G. Leslie, B. Zamborlin, P. Jodlowski, and N. Schnell. Grainstick: A collaborative, interactive sound installation. In *Proceedings of the International Computer Music Conference (ICMC)*, 2010.

[10] F. Melchior, T. Laubach, and D. de Vries. Authoring and user interaction for the production of wave field synthesis content in an augmented reality system. In *Mixed and Augmented Reality, 2005. Proceedings. Fourth IEEE and ACM International Symposium on*, page 48–51, 2005.