DT2215 ADVANCED INDIVIDUAL COURSE IN MUSICAL COMMUNICATION

KTH, Royal Institute of Technology

Detection of musical cues in conducting hand movements for control of music reproduction

Nino Prekratić

ninop@kth.se

1. ABSTRACT

The aim of this project was to develop a motion detection system which can be used for real time control of sound synthesis or sound reproduction parameters, inspired by the traditional role of the conductor in a choir or an orchestra. The system should enable the user to integrate an emotional expression with the reproduced music with ease. By tracking the user's hand movements, the system detects and calculates the amount of the movement, its velocity and the change in acceleration. These calculated cues are mapped to the dynamics, tempo and articulation of the sound-generating part of the system. Several different methods for calculating the three mentioned sound parameters were utilized and tested. It was shown that it is not trivial to detect hand movements and interpret them in a musical way, as humans can easily do. Different tested approaches work within different sets of limitations and the development of a more robust system requires further development and investigations.

2. INTRODUCTION

This project is motivated for the most part by two research projects. The first one was shown in Vienna's *Interactive Sound Museum* with the *Personal Orchestra* [1] where a user can control a video projection and audio reproduction of the Vienna Philharmonic Orchestra in concert. In this installation, users can set the tempo of the audio/video reproduction and interactively control volume and instrumentation of the reproduced music using only movements of the handheld infrared baton sensor.

The second work that served a major role as a guideline for this project was an investigation by Dahl and Friberg [2] where they performed two experiments in order to determine the extent to which emotional intention could be conveyed through musicians' movements. Through their work, authors concluded that intentions of happiness, sadness and anger are well communicated, while fear is not. They also rated movement cues which could be considered to correlate to different emotional intentions. For example, Anger is primary associated by jerky movements, Happiness with large and smooth movements, Sadness with slow and smooth movements. Lastly, and

most importantly for our project, the authors pointed out that it is reasonable to assume that body movements cues could correspond to the cues contained in the audio signal. Taking this into consideration, the amount of movement is related to dynamics, the velocity of movement is related to tempo and the jerkiness of movement is related to articulation.

In order to combine the two mentioned projects, our goal was to develop a hands-free system that does not require a physical handheld sensor and is capable of manipulating three different musical cues in order to insert emotional intention and interpretation into music reproduction. These musical cues are, as previously mentioned, dynamics, tempo and articulation. This interaction should be achievable in a natural and intuitive manner, and every user should be able to understand and start using the system quickly.

To satisfy the need of hands-free operation of the system, we used the *Microsoft Kinect* [3] motion detection sensor. The data from *Kinect* were first gathered and processed in a *Processing 3.0* [4] programming environment. This processed information was then transferred to *Pure Data* [5] visual programing environment, where it was scaled and routed to *pDM* [6] sequencer with implemented KTH music performance rules [7]. This sequencer, which is also realized in *Pure Data* environment, is thus controlled by the gestures, and the manipulated parameters are the sound level, tempo and overall articulation performance parameters.

Besides the *Personal Orchestra* in Vienna *Interactive Sound Museum,* there are a few other examples of systems where users manipulate sound reproduction using physical movement. One of the first and oldest attempts in doing a "conducting" interface is *The Radio Baton* [8] interface. With this system a user is able to control the reproduction of electronically stored and played back musical score by using a mechanical baton in their hand. Parameters that can be manipulated are tempo, volume and balance of certain voices in the score. Another example is the *Home conducting* [9] project developed by using a simple web camera and *EyesWeb* [10] software. In this project, the author developed a system with several interaction levels which makes it suitable for different user profiles, from novice and children to experts. User's expressive movements are mapped into semantic expressive descriptions which are then mapped to performance rule parameters in *pDM* where the electronic score of a certain musical piece is reproduced.

There is also an example of a system [11] which uses *Kinect* motion sensor and is able to, based on user's gestures, manipulate the tempo of the music playback and the dynamics of separate instruments in the musical piece. Here, tempo manipulation is done in a similar way to the *Personal Orchestra* project, by time stretching technique. Lastly, an interesting project called *Virtual Orchestra* [12] including *Kinect* sensor was developed. It is a video game that was meant to be played by users who do not have any experience with conducting and should be easy to learn. It enables the player to manipulate the tempo, volume, pausing and starting of a song represented as a MIDI file. The game also gives visual feedback by informing the player about the instruments that are currently playing, the overall volume and the next two measures of notes.

3. TOOLS AND IMPLEMENTATION

To be able to establish a hands free operation in the project, we have used second generation *Microsoft Kinect* [3] motion input sensing device.

In order to store and manipulate the data stream from *Kinect*, the programing environment *Processing 3* [4] was used. To establish communication between *Kinect* and *Processing,* a library developed for *Processing* environment called *KinectPV2* [12] was used. This library was developed in a way that takes advantage of built-in functions of an official *Microsoft Kinect SDK* [13], which also has to be installed on the system. An additional library called *Signal Filter* [14] was used for filtering of input signal. The filtering applies a first order low-pass filter with an adaptive cutoff frequency. In order to transfer data from *Processing* to the *Pure Data,* OSC communication protocol was used. This communication was supported by *OscP5* [15] library developed by Andreas Schlegel.

After filtering the signal and calculating the values which represent wanted musical cues, the data were sent to *Pure Data* [5] visual programming environment developed by Miller Puckette. *Pure Data* is an open source dataflow programming language where the functions or "objects" are connected or "patched" together in a graphical environment which is based on the layout control and audio signals modular synthesis systems.

The last element in the chain of this project's implementation, which enables the real time manipulation of reproduced sound, is *pDM* [6] sequencer with implemented ability to manipulate *KTH Music Performance Rules* [7] in real time. *pDM* is developed by Anders Friberg in the *Pure Data* (*pd*) visual programming environment. In order to get *pDM* running under the newest version of *pd*, it was necessary to use some *pd* libraries from an older and unsupported version of *pd* named *PD-Extended.* The reason for this is that *pDM* was built on that version of *pd*.

The project was running on a *MacBook Pro* machine with *Windows 8.1* installed as a secondary boot option for the computer.

4. METHOD

4.1 General overview

It was decided that the system is going to be controlled by information gathered from movement detection of the user's right hand. One of the reasons for this was to simplify and speed up the iterative nature of prototyping process for the project, with a goal of applying the same processing and calculation to the other hand at a future time and combining the influences of both hands. We also wanted to accommodate for the users who try to control the systems with only one hand.

The first step in the process is the extraction of skeleton data from *Kinect* sensor. This data is calculated from the RGB image provided by the *Kinect*, and is mapped to a coordinate system defined as can be seen in *Figure 1*.

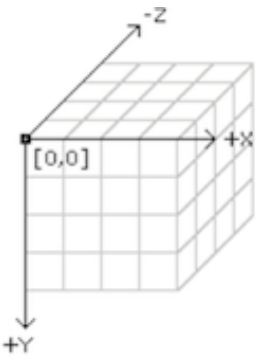

Figure 1. – Coordinate system of Kinect RGB image

By doing this, we acquired a visual representation of the location of each joint in the detected skeleton on the user's body image. The joint locations are represented with *x, y* and *z* pixel location values in a 3D space (*Figure 1.*), but we have taken into account only *x* and *y* values. Location data used in our calculations are *Hand_right joint*, *Elbow_right* joint, *Shoulder_right* joint and *Spine_middle* joint locations, which are represented with green circles in *Figure 2*. These values were stored in a *PVector* class of *Processing*, which is a convenient way of storing the *x* and *y* values of a certain location on screen.

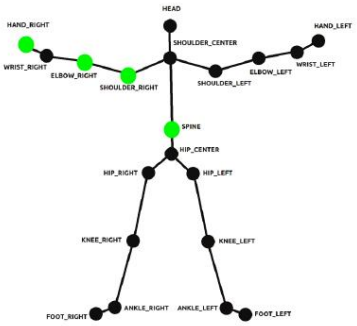

Figure 2. – Kinect skeleton information and used joint locations

The reason for using these values is to make the system adaptable to every single person who is using it. Human beings can, as we know, vary greatly in their proportions, so we wanted to enable the system to detect certain user dimensions and use them to scale its calculations. As it can be seen in *Figure 3,* the main dimensions we are interested in are the right hand size (blue line – *rightHandSize*), the distance between the right hand joint and the spine middle joint (red line - *centerHand*), and the distance between the right shoulder joint and the spine middle joint (green line – *centerShoulder*).

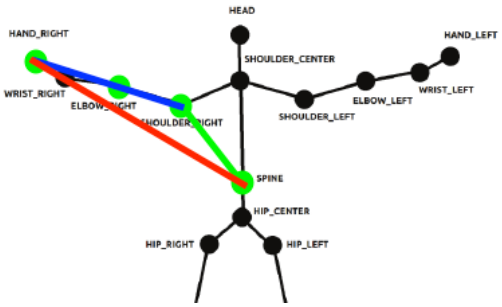

Figure 3. – Joints used for calculations and distances between them

By knowing the dimensions of the *rightHandSize* and *centerHand*, we know the maximal possible value of *centerShoulder*. This information is important for system calibration and calculation of the dynamic cue value, which will be explained in more details in the following sections. The velocity and jerkiness parameters do not need to be calibrated and scaled based on the user dimensions.

In order for the system to start functioning, the user right hand joint has to be detected. After that, the data representing the aforementioned joints is filtered and used to calculate many different values, such as maximum hand magnitude, current hand magnitude and spine shoulder joints distance. Also, the values of right hand velocity and jerkiness are calculated.

After these calculations are performed, the resulting data is used in different functionalities that enable the system to work and manipulate the selected audio cues. In order to achieve proper system operation, besides the current values of this data, it is crucial to know the previous ones, so their values have to be stored. For example, the difference in joint positions between two subsequent frames is needed to calculate velocity.

The resulting values of all of the implemented functionalities are routed to

their desired location in the *pDM* sequencer and are used for manipulation of targeted cues, i.e., dynamics, tempo and articulation. This routing is realized in a flexible way, so that the user can easily test all implemented methods. Also, many of the raw and calculated data are visualized so the process of development and troubleshooting is simplified. Details on the implementation of all of the mentioned functionalities are going to be explained in the rest of the text and can also be explored in the project code attached to this report. The *Processing* code is contained in file *Kinect_calculations.pde* and the *pd* patch is contained in the *Kinect_routing.pd* file. The code is commented in order to additionally describe all of the functionalities.

4.2 Calibration

In order to enable the scalability of the system with a goal of retaining proper functionalities regardless of user's dimensions, the system has to be calibrated. The calibration process is performed constantly while system is working. This is necessary in order to achieve proper functionality even for a single user, since the dimensions of interest change with user's distance or rotation in relation to the *Kinect* sensor. Also, some measures are taken in order to avoid unrealistic information calculated from *Kinect* data which is generated by errors in joint location detection that happen from time to time. For example, this is evident when user's hand is located in front of his/her body.

As mentioned before, the key measures used for the calibration process are the right hand size and the distance between the right hand shoulder joint and the spine middle joint. These dimensions are calculated from the joint location values gathered from the *Kinect* image. Since these joint locations are represented as *x* and *y* values in a two-dimensional coordinate system, the distances between them can be calculated as Euclidean distances between two points in two dimensions:

$a = (x_1, y_1)$, $b = (x_2, y_2)$

$$dist(a,b) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \quad (1)$$

This is easily implemented in the *Processing* environment as all the joint location values are stored as a *PVector* class objects. If we think of them as vectors whose origin is at location (0,0) and end points at the value stored in *PVector,* when we subtract two vectors, we will get the vector between the two joint locations. This is done with *sub()* function implemented in *PVector* class. The magnitude of this vector, calculated with *mag()* function of *PVector* class, will give us the wanted distance between two points (as defined with equation (1)).

Scalability is important only for calculation of dynamic cue of the system. It is achieved by comparing the current distance between *Hand_right* and *Spine_center* joints (red line in *Figure 3*) with the reference value of total right hand size, where total right hand size represents maximal sound level in *pDM.* This reference value is defined as a sum of magnitudes of the right hand size vector (the blue line in *Figure 3*) and the vector between *Shoulder_right* and *Spine_center* joints (the green line in *Figure 3*).

The distance between *Hand_right* and *Spine_center* joints, representing the current sound level is calculated as:

- centerHand = dist (Hand_right, Spine_center)

In order to calculate the right hand size, distances between different joints have to be calculated and summed. These distances are:

- rightForearm = dist (Hand_right, Elbow_right)
- rightUpperArm = dist (Shoulder_right, Elbow_Right)

These values are needed to calculate the right hand size:

- rightHandSize = rightForearm + rightUpperArm

Next, to calculate the total right hand size, we have to calculate the distance between Shoulder_right and Spine_center joints:

- centerShoulder = dist (Shoulder_right, Spine_center)

Finally, the total hand size (main reference value) is calculated as:

- totalRightHandSize = rightHandSize + centerShoulder

One more calibration principle is implemented in this project, very similar to the process explained here, with the difference of using alternative joint location data. This will be explained in the next section.

4.3 Dynamics

The parameter used to manipulate the dynamic cue (sound level) in the *pDM* is calculated in three different ways. All of these methods are based on the current hand distance scaled according to the reference hand size value in order to determine current volume.

The first two methods (denoted as Vol_1 and Vol_2 in *pd* patch) use a different calibration method compared to the one described in the previous section. Unlike the method from Section 4.2, calibration in the Vol_1 and Vol_2 methods uses only the *rightHandSize* value for the reference.

The measure for calculating the current volume, that is scaled according to the reference *rightHandSize*, is the distance between *Hand_right* and *Shoulder_right* joints (*magRight* in *Processing* code). This value is stored and used every time the current hand position reaches the maximal position on the *x* axis in the current hand move cycle. A hand move cycle can be divided into two phases looking at the horizontal, *x* axis of the system. One phase encompasses the user movement of the hand away from the body (movement towards OUT, increasing *x* values) and the other phase encompasses the movement of the hand towards the body (movement towards IN, decreasing *x* values). That gives us two extreme positions on the *x* axis for the hand movement trajectory, one minimum (MIN) and one maximum (MAX) in every cycle. In the first approach for dynamic cue calculation (Vol_1), we are concerned only with the MAX position. This approach works well within certain limitations, but suffers from several issues. One drawback is that the reference position from where the current hand distance is calculated is located on the right shoulder. Therefore, it is difficult to detect small values of the current hand size and, consequently, produce small volume values. Also, the calculations are triggered in the MAX position of the hand only on the *x* axis, which makes it impossible for the user to control the volume by moving his/her hand up and down instead of left to right.

The second calculation method (Vol_2 in *pd* patch) uses a different logic, whose goal is to eliminate the problem of having only one axis. Here, we do not take into account the extremes of hand movement cycle on the *x* axis, but we monitor the current hand size value. We again measure it as the distance between *Hand_right* joint and *Shoulder_right* joint (*magRight*). After this value has been increasing for a certain number of frames (minimum 15) and when it has been detected that the value started to decrease, we conclude that a maximum position of the current hand size in a given movement direction has been reached. The value that has been detected in the frame immediately before the detection of decline is then sent for calculation of the dynamic cue. This method eliminates the restriction of using only one axis, but does not provide a reliable mechanism. It produces a lot of unexpected output which does not correspond to user input or intention.

The third and final method (Vol_3) for calculating the dynamic cue is based on the calibration method and reference value explained in Section 4.2. It requires a more complex reference measurement, but this reference has less variations in its value than the one in previous methods, because the distance between the shoulder and the spine joint is more constant than others. Also, we calculate the reference only when the hand position on *x* axis is 50 pixels away from the *Spine_center* joint *x* position and the current hand move cycle is in the expanding phase. This helps to avoid calculations in positions for which the data from *Kinect* is not reliable (when the hand is overlapping with the body). A maximal value of the current hand size (*centerHand*), which is scaled in relation to the reference value (*totalRightHandSize*) in order to calculate the dynamic cue, is being detected at a given hand cycle movement. In this approach, the hand cycle does not depend on the *x* axis position, but only on the current hand size (the maximal *centerHand* values are detected). This method is the most reliable one because of its many implemented security measures where we avoid to do calculations when the data is not useful. Also, the choice of taking the *Spine_middle* joint as a reference for calculating the current hand size makes this method more intuitive for users, and achieves a wider dynamic range more easily.

All of these approaches for calculation of the dynamic cue have another level of filtering implemented before routing the data to the *pDM* sequencer in order to achieve a smoother control of the sound level parameter. This filter is implemented in *pd* environment by using the *line* object. Also, if there is no new information about the volume (no new data is calculated), the last value that has arrived to *pDM* is kept, but is decreased by 0.1 every 100 ms.

4.4 Tempo

The tempo cue is mapped to the velocity value that was calculated from *Kinect* data for the right hand joint in four different ways. In a two dimensional space we were working in, the velocity of a certain joint is calculated as:

$$v_i = \sqrt{(\dot{x}_i)^2 + (\dot{y}_i)^2} \qquad (2)$$

where $\dot{x}_i$ and $\dot{y}_i$ are the first derivatives of the position coordinates. Considering that we work in a discrete domain, we can look at the first derivation as the distance between the current and the previous position of the hand. This is easily calculated in *Processing* using two objects of *PVector* class with stored current and previous location data and calculating the distance between them.

The first method (VelMax in *pd* patch) is based on hand movement phases in a movement cycle observed only on *x* axis of the coordinate system. As we have already mentioned, there are two movement phases for every single cycle, i.e., the movements towards IN and towards OUT, with their respected extreme positions, MIN and MAX. In both IN and OUT phases, the maximal velocity value in a phase is detected and used. Theoretically, one value per phase should be routed to *pDM*, but because of the way the code was written, few different values per phase are sent. Each sent value represents the maximum element in each cluster of velocity values. This is a solid approach which gives good result when a user is aware of its functioning, but that does not make it very intuitive. Also, it does not work very well when the path between the MIN and the MAX position of the hand movement cycle is short and the frequency of hand movement between these extremes is high. For high frequency hand movements with short path, the idea of looking at the velocity calculated with equation (2) as a representation of the tempo is false. This could be explained with the hand not being able to achieve high enough velocity over such a short path.

To fix the problem of high frequency movements, we have developed a second approach of calculating tempo information (VelMinMax in *pd* patch). This approach is based on detection of MIN and MAX extremes observed on *x* axis of the hand movement cycle. The time needed for the hand to go from one extreme to another is calculated and used as the velocity value that is later scaled. This approach works well when the detection of the extreme position is detected correctly. The problem is that MIN and MAX values cannot be detected with a 100% accuracy and it often happens that this method does not give any output because extreme positions were not sensed. Also, when user makes bigger hand movements, with larger distances between MIN and MAX, the subjective feedback of the tempo parameter is not as good as in the VelMax approach.

The third approach for calculating the velocity (VelAvg in *pd* patch) is again based on movement phase detection dependent on a *x* axis. In every phase, all of the velocity values that are calculated based on equation (2) are summed and the number of samples is counted. When the hand reaches one of the extreme positions, the average value is calculated and used for control of the tempo parameter in *pDM*. This approach works similarly to the VelMax approach, but with a slightly different subjective feedback. Updating the tempo information only once per cycle does not feel natural. Also, the problem with high frequency movements and unprecise detection of the extreme positions is still present.

The fourth and last approach (VelRaw in *pd* patch) for calculating the tempo parameter is the simplest and most straightforward one. The raw data that has been calculated based on equation (2) is being filtered once more with a first order Butterworth filter in order to further reduce the signal fluctuations, and routed to *pDM* if it is greater than 0 and below a certain extreme value. This method gives the best subjective feedback when using the system and it is not limited to the hand movement cycle constrains. However, the problem of high frequency hand movements is still present.

All four methods for calculating the tempo parameter have similar filtering implemented in *pd* as the dynamic

parameters before they are mapped to *pDM* (by using *line* object in *pd*). The only difference in the filtering is that when the tempo parameter values go from a lower to a higher level, the smoothening is not as strong as when the values decrease. Also, there is no value reductions if no new information has been received.

4.5 Articulation

The articulation cue can be controlled in four ways according to the input signal of the right hand joint. All methods are based on calculation of jerkiness of the input signal (right hand joint position). Jerkiness can be defined as a change of the acceleration of the signal. From mathematical point of view, it can be presented as a third derivation of a hand joint position. This calculation is again easily achieved in the *Processing* environment by using the *PVector* object class and by calculating velocity, acceleration and then jerkiness from hand position data. Overall, it is questionable whether this way of calculation really represents the articulation cue as required for our system. As with the velocity parameter calculated in a similar way, it is true only under certain conditions (mainly with longer hand movement cycles).

The first approach for calculating jerkiness (Jerk_1 in *pd* patch) is similar to the first approach for velocity calculations (VelMax). In every phase of the hand movement (towards OUT or IN), the maximal value is selected from raw calculated jerkiness data. These values can vary greatly, so we presumed it needed some initial scaling. This scaling was implemented in a function (*jerkinessFunc* of *Processing* code), which forms categories for input signal and returns the value of the category to which the input corresponds. This approach, similar to the VelMax approach, does not give a very intuitive feedback to the user. Even though, similar to VelMax calculations, we do not get only one maximal value per movement phase, infrequent refreshment of data is not suitable for real time usage of the system.

In the second approach (Jerk_2 in *pd* patch), raw calculated jerkiness data is routed to *pDM*. Constant update of calculated values that are sent to *pDM* are beneficial to the subjective feedback when using the system. But some unpredicted values come up frequently, interrupt the system and misrepresent the user's intent. These values occur when the user moves his hand in a way that values calculated in this manner do not represent the articulation cue.

The third method (Jerk_3 in *pd* patch) is very similar to the first one, with a difference of not performing categorization before routing the information to *pDM*. This method has not shown beneficial and it does not represent any improvement compared to Jerk_1 approach.

The final method (Jerk_4 in *pd* patch) is an alternative way of implementing the Jerk_3 approach. The dependence on the hand movement cycle is avoided and peak values of jerkiness are detected. This method constantly looks for the largest value in the signal flow (calculated jerkiness). It replaces the maximal value of the calculated signal with the current one while the signal level is rising, but when the current value falls below 30% of the current maximal value, this maximal value is then sent to PD. There are also some limitations that the raw signal has to satisfy in order to avoid very low level noisy signal and extreme peaks that are calculated every now and then. This approach gives a

small upgrade to the third approach, but it still does not represent user's articulation intents. The data refresh rate is larger than in the Jerk_1 and Jerk_3 methods, but of course smaller than the Jerk_2. It can be considered as the best of all considered methods because the constant update can be undesirable when calculating jerkiness in order to avoid using the undesirable data from the raw information.

We have also tested an alternative approach for calculating the third derivation of a signal in a discrete domain. This method is called *Finite Differences Method* [16]. The calculations were performed according to the following equation:

$$\dddot{f}(x) \approx \frac{f(x)-3*f(x-1)+3*f(x-2)-f(x-3)}{h^3} \quad (3)$$

where *f(x)* is the current sample value, *f(x-1)* is the previous sample values, *fs(x-2)* is the one before that and so on. *h* is the distance between the samples and we can presume it is 1 in our case. This calculation has given us the same results as calculation done with the built-in class functions of the *Processing* environment.

Articulation calculations are subject to the same filtering in *pd* as the tempo calculations before they are mapped to the parameters in *pDM*.

5. DISCUSSION

This project has demonstrated that it is not trivial to develop a system which can be controlled only by movement detection. Not only that there are a lot of technical challenges and limitations of the equipment that was used, but there are also challenges of conceptualization and understanding of how to correctly interpret user movement and map it to musical cues of interest.

As we can see in the example of dynamic control, no matter how we defined the calculation principle, there are always some limitations inherent to the solution. For example, when we calculated the current hand size by measuring the distance between the hand and shoulder joints, we have limited ourselves in an unnatural way. If the user does not know exactly how the system works, it is not easy to achieve the desired results, and sometimes the task is even frustrating. In this case, the system is improved by switching to middle spine joint as the reference from where the current hand distance is calculated. This position enables more intuitive operation, but the limitation of having a fixed reference point is still present and reduces the flexibility of a system. For example, if the user makes circular movements around this point, the distance value does not change and the volume of reproduced music stays the same even though the user's intention may be different.

Similarly, we needed to make two different approaches for calculation of the tempo cue in order to make the system work properly and provide useful data for two different operation modes. Still, these two methods do not work side by side and cannot provide one uniform solution for the tempo cue calculation problem. This is a severe limitation.

The articulation cue has shown to be the most problematic one to understand, detect and calculate. Looking at the mathematical idea behind it, it seems straightforward, but our implementation has shown otherwise. There could be many reasons for it, including noise that is enhanced by doing the third derivation,

taking a wrong approach in interpretation of the data, dealing with extreme values resulting from bad detection and so on.

Overall, more work needs to be done, both on the side of conceptualization and on the side of implementation, in order to develop a more robust system. The first and crucial thing is how to understand our movements and how to interpret them by using a computer. It has been shown in our experience that we cannot rely solely on the mathematical interpretations and definitions.

6. FUTURE WORK

As already concluded, a lot more work on different aspects of the project is needed in order to develop a more robust and precise system. Besides the big conceptual work, many small steps can be made in order to improve the current implementation.

First of all, the code should be rewritten in a way that is more readable, modular, reusable, with a concise naming conventions and so on. For example, the method of implementing OSC communication should be done in a more elegant way, without so much repetition in the code. The whole program was written in a manner of developing new ideas and testing which work and which don't, so it comes out as unorganized.

Second, start and stop functionality of the system has to be improved. User should be able to start and stop the system with his/her movements. At the moment, this functionality is mapped to hand detection, which doesn't work perfectly.

Third, a function should be implemented to make the system aware of which exact user is controlling the system. The reason for this is when another person comes in front of the *Kinect* sensor, the system should be aware of who the user which controls it is and how to switch between these users if wanted. Now the system just crashes when this happens.
Forth, the second hand functionality should be implemented and a method for combining the contributions of both hands should be developed.

Next, the two different methods for assessing the tempo cue should be merged into one. When the hand is in the high frequency movement mode, the system should detect that and work in this mode (VelMinMax). When user is doing a more typical movement, the last developed method (VelRaw) should be used.

Finally, the mapping of calculated values to musical cues should be done in a more complex and interesting manner, and not directly mapped as it is now. The type of mapping could be an implementation of *Fuzzy analyzer of emotional expression in music performance and body motion* [17]. Its functioning depends on the sound level, tempo and articulation parameters which makes it convenient for our project. The output it provides are the cue values for emotional expressions of happiness, sadness and anger.

7. REFERENCES

[1] Borchers, J., Samminger, W. and Mühlhäuser, M. (2002). *Personal Orchestra: Conducting Audio/Video Music Recordings.* WEDELMUSIC'02 Proceedings of the Second international conference on Web delivering of music, Pages 93 – 100.

[2] Dahl, S., Friberg, A. (2007). *Visual Perception of Expressiveness in Musicians' Body Movements.* Music Perception, Volume 24, Issue 5, PP. 433 – 454.

[3] Microsoft, (2013). *Kinect for Xbox One.* Online at: http://www.xbox.com/en-US/xbox-one/accessories/kinect

[4] Processing foundation, (2017). *Welcome to Processing 3.* Online at: https://processing.org

[5] Pure Data (2016). *Pure data.* Online at: https://puredata.info

[6] Friberg, A. (2006). *pDM: An Expressive Sequencer with Real-Time Control of the KTH Music-Performance Rules.* Computer Music Journal. Spring 2006. Vol 30. No. 1. Pages: 37-48

[7] Friberg, A., Bresin, R., Sundberg, J. (2006). *Overview of the KTH Rule System for Musical Performance.* Advances in Cognitive Psychology (2006). Volume 3. Issue 2-3. Pages 145-161.

[8] Mathews, M. V. *The Conductor Program and the Mechanical Baton.* In M. Mathews & J. Pierce, eds. Current Directions in Computer Music Research. Cambridge, Mass: The MIT Press, (pp. 263-282), 1989.

[9] Friberg, A. (2005). *Home Conducting – Control the overall musical expression with gesture.* Proceedings of the 2005 International Computer Music Conference, San Francisco: International Computer Music Association. (pp. 479-482).

[10] Casa Paganini – INFOMUS. *The EyesWeb project. Online at: http://www.infomus.org/eyesweb_ita.php*

[11] Rosa-Pujazo´n, A., Barbancho, I., Tardo´n, L. J., Barbancho, A.M.(2013) *Conducting a Virtual Ensamble With a Kinect Device.* Proceedings of the Sound and Music Computing Conference 2013, SMC 2013, Stockholm, Sweden

[12] Lengeling, S. T. (2016). *Kinect V2 library for Processing.* Online at: https://github.com/ThomasLengeling/KinectPV2

[13] Microsoft, (2014). *Kinect for Windows SDK 2.0.* Online at: https://www.microsoft.com/en-us/download/details.aspx?id=44561

[14] de Courville, R. (2015). *Signal Filter (beta).* Online at: https://github.com/SableRaf/signalfilter

[15] Schlegel, A. (2015). *OscP5.* Online at: https://github.com/sojamo/oscp5

[16] Wikipedia*, (2016). Final differences method.* Online at: https://en.wikipedia.org/wiki/Finite_difference_method

[17] Friberg, A. (2004). *A Fuzzy analyzer of emotional expression in music performance and body motion.* Proceedings of Music and Music Science, Stockholm, October 28- 30, 2004