

K-Multiscope: Combining Multiple Kinect Sensors into a Common 3D Coordinate System

Kyle Stewart
Computer Science Department
College of Charleston
Charleston, SC, USA
stewartkh@g.cofc.edu

Bill Manaris
Computer Science Department
College of Charleston
Charleston, SC, USA
manarisb@cofc.edu

Tobias Kohn
Computer Science and Technology
University of Cambridge
Cambridge, UK
tobias.kohn@cl.cam.ac.uk

ABSTRACT

We present a method for combining data from multiple Kinect motion-capture sensors into a common coordinate system. Kinect sensors offer a cheaper, potentially less accurate alternative for full-body motion tracking. By incorporating multiple sensors into a multiscope system, we address potential accuracy and recognition flaws caused by individual sensor conditions, such as occlusions and space limitations, and increase the overall accuracy of skeletal data tracking. We merge data from multiple Kinects using a custom calibration algorithm, called K-Multiscope. K-Multiscope generates an affine transform for each of the available sensors, and thus combines their data into a common 3D coordinate system. We have incorporated this algorithm into Kuatro, a skeletal data pipeline designed earlier to simplify live motion capture for use in music interaction experiences and installations. In closing, we present *Liminal Space*, a live duet performance for cello and dance, which utilizes the Kuatro system to transform dance movements into music.

CCS CONCEPTS

• **Computing methodologies** → **Motion capture**; *Motion processing*; • **Human-centered computing** → *Interaction design*; • **Applied computing** → *Performing arts*;

KEYWORDS

Motion capture, Kinect sensor, motion mapping, skeletal data, algorithmic music, interaction design

ACM Reference format:

Kyle Stewart, Bill Manaris, and Tobias Kohn. 2019. K-Multiscope: Combining Multiple Kinect Sensors into a Common 3D Coordinate System. In *Proceedings of 6th International Conference on Movement and Computing, Tempe, AZ, USA, October 30, 2019 (MOCO)*, 8 pages. <https://doi.org/10.1145/1234567890>

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.

MOCO '19, October 10–12, 2019, Tempe, AZ, USA

© 2019 Association for Computing Machinery.

ACM ISBN 978-1-4503-7654-9/19/10...\$15.00

<https://doi.org/10.1145/3347122.3347124>



Figure 1: The Veil (2017) is an experiment in musical group dynamics (i.e., spontaneous musical interaction and collaboration among performers), presented at the Music Library of Greece, in Dec. 2017.



Figure 2: Liminal Space (2018) is an aleatoric piece for cello, motion capture, and interactive software motivating this work, discussed in more detail later in this paper. Also see <http://bit.ly/liminalspace2018b>.

1 Introduction

In the development of interfaces for expressive musical interaction and artistic creation, markerless motion capture offers the opportunity for free-form, full-body experiences, sonifying motion without requiring the user to wear a MoCap suit. Users and performers can freely walk in and out of the space, without the need for special equipment, and even serendipitously discover

the experience and decide to engage in it at a moment's notice. This provides wonderful opportunities for artistic creation, installation design, and new movement + technology explorations. Figures 1 and 2 show experiences designed with earlier versions of our system, and motivating the approach presented herein.

The Kinect 2.0 sensor offers a cheap, mobile, and markerless solution, with notable accuracy, which exceeds that of the original v1.0, and will be soon replaced by version 3.0 (announced recently). With the provided Microsoft Kinect for Windows SDK 2.0, a single Kinect sensor can process infrared depth data into skeletons made of 25 unique joints, for up to six users at a time [15]. Individual sensors, however, are prone to producing erroneous data, due to the occlusion of body parts, having multiple users in the same space, and other conditions. Even in a simple scenario, such as a user facing sideways, the sensor may lose tracking for entire limbs.

Our approach is to improve upon the original sensor, by using multiple sensors to capture users from different sides, thus eliminating occlusions and producing redundancy in skeletal data. This can be used to detect and average out noisy data, as well as cover larger areas than that of a single sensor.

Handling data captured by these sensors can be a cumbersome process. Rather than create custom, one-of-a-kind solutions for each motion-controlled instrument or art installation (as we have done before, e.g. [9, 13]), we decided, driven by the desire to generalize, to design our own system to route, organize, store, and otherwise handle motion-tracking data. An early prototype of this system was utilized in Liminal Space (see Fig. 2). Liminal Space is described more fully later in this paper.

Our framework unifies data from multiple sources, which capture skeletal data (full-body or otherwise, such as Kinect, LeapMotion, etc.), by creating a common 3D coordinate space that combines, and updates all data contributed by different sensors in real time. This general solution for $n \in \{1, 2, 3, \dots\}$ sensors offers increased accuracy and area coverage / range over a single sensor, and can be easily adapted to drive a multitude of local and telematic applications, in a way that is modular and allows for rapid prototyping of music interactive installations, and other experiences. In the next section, we go over earlier, similar systems and approaches, and discuss how our approach is different to combining multi-sensor data for creating motion-tracking applications and experiences.

2 Background

Employing multiple depth sensors in pursuit of improving overall accuracy of motion tracking and pose estimation is a subject which has significant research precedent. Due to the ease of access to the Kinect sensor as an entry point into the field, there have been several attempts at solving the occlusion problem and achieving increased accuracy by consolidating the data of multiple Kinect sensors positioned around a user, especially in recent years.

Berger et al. explore a technique for calibrating multiple Kinect v1.0 sensors using their RGB cameras. They find that a

reflective-diffuse checkerboard calibration approach is better in calibrating multiple Kinect v1.0 depth and color sensors, when compared to a point light system [6]. Their solution, using as many as four Kinects, was able to overcome occlusion by viewing the user from multiple angles, resulting in a point cloud which could be mapped to a three-dimensional model.

Asteriadis et al. bring the methods of [6] into motion estimation, applying the checkerboard calibration technique followed by Singular Value Decomposition on the resulting matrix [2]. They apply the final transformation matrix on skeletal data to some success.

Baek and Kim compare live motion to that stored in a database for dance instruction, combining 4 Kinect v1.0s [3]. They use a custom calibration system based on the normal vector derived from the user's hip joints and employ a spline interpolation to coordinate asynchronous data delivery from the sensors.

Gao et al. utilize two Kinect sensors for greater accuracy in full-body pose estimation [8]. Rather than operating on skeletal data, they work with depth data directly and make use of the Iterative Closest Point (ICP) algorithm using the two point clouds gathered from each Kinect. The ICP algorithm attempts to find the transformation matrix which translates a *source* point cloud to the space of a *target* point cloud by matching two nearby points as closely as possible, then iterating until an accuracy threshold is reached.

Kitsikidis et al. work, again, in pose estimation, but make direct use of the Kinect's skeletal data [11]. By applying the ICP algorithm on skeletal data, considering joint confidence as weight, and prescreening to prevent errant sudden motions, they were able to combine data from several Kinects with measurable success. Their solution utilized Kinect v1.0 and had trouble when users failed to directly face a given sensor.

Kim et al. combine 8 Kinects using the ICP algorithm, once more with depth data [10]. They find an 85% accuracy in capturing dynamic movement when compared with the Xsens inertial-based wearable capture system as a control.

Moon et al. pursue a system of multiple Kinects joined by applying Kalman Filtering, where each joint can be evaluated for reliability [17]. Their system performs better than a single sensor, through simple averaging of multiple sensors' data when occlusions are present.

Murray-Browne and Plumbley approach the issue of handling motion-tracking data for use in interactive sound applications [18]. Their system is tailored for capturing musically meaningful features via gesture recognition. They focus on ease-of-use, modularity, and rapid prototyping, among others, utilizing a visual dataflow programming language similar to Max and PureData.

Mokhov and Bardakjian develop the Illimitable Space System (ISS), which captures motion data from a Kinect or similar sensors and facilitates using the data to enhance live performance with visuals [16]. The later ISSv3 brings the system into the augmented and virtual reality landscape [4].

Amin and Burke work in a more general context to create a toolset for interpreting motion data and movement patterns [1].

They distinguish the capture of data from its interpretation and focus on deriving meaning from the data.

Similarly, Dahl and Visi work with marker-based motion capture systems and develop Max abstractions to synthesize meaningful representations and qualities of motion [7].

Our approach differs from the above in that it utilizes skeletal data captured from Kinect 2.0 (or other) sensors, positioned arbitrarily in space (as long as the infrared cameras do not “blind” one another, i.e., by directly facing each other). These sensors are paired, in different possible configurations, with a main sensor, whose coordinate system is used as the main, or reference coordinate system. Our approach works directly with skeletal data for simplicity (i.e., no point cloud data), and utilizes a calibration technique explained in the next section to derive affine transforms to map and merge data into a single coordinate space.

3 System Calibration

As mentioned earlier, our approach, called *K-Multiscope*, converts skeletal data captured from different perspectives (i.e., by different Kinects or other sensors) to a single perspective. For simplicity of presentation, we describe a 2D representation of the problem setup, using a pair of Kinects. Our technique, however, works for with an arbitrary number of chained sensors in 3D data.

Fig. 3 shows a birds-eye view of a 2D setting for two such Kinects, K_L and K_R . In order for this approach to work, we need to translate data from one Kinect’s coordinate system to the corresponding data in the other Kinect’s coordinate system. This translation between coordinate systems is called an *affine transformation* (i.e., a mapping of coordinate points).

Since the Kinects can be placed anywhere in the space to be captured, we cannot anticipate a priori what the affine transformation will be – it needs to be discovered / computed at setup time. We have explored two ways to perform this calibration.

The first approach utilizes *linear regression* to derive an affine transformation between pairs of Kinects, K_L and K_R . The second approach utilizes a *genetic algorithm* to evolve the affine transformation. Both ways are explained below. We then evaluate the results from each approach, against sets of actual data, to determine which performs better.

As mentioned earlier, this *stitching* technique can be generalized by creating a chain of Kinect pairs (e.g., K1 and K2, K2 and K3, K3 and K4, and so on), and converting between coordinate systems, until all data are in the same coordinate system (e.g., the K1 coordinate system).

3.1 Affine Transformation between Kinects

Our approach works in 3D space. This section explains the affine transformation in 2D space, for simplicity. In order to convert data between two Kinects, we need to find:

- the common center, C , that the two Kinects are seeing, and

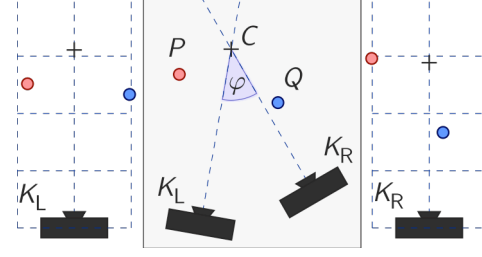


Figure 3: Two Kinects K_L and K_R looking at a scene with two points and a common center C (in the middle). Left and right is the situation as seen from either Kinect alone. The task of calibration is to find the coordinates of the center C as well as the angle of rotation φ , so that the two individual images on both sides combine to the common image in the middle.

- the angle of rotation, φ , relative to this common center (see Fig. 3).

The angle of rotation φ around center C can be written using a 2×2 matrix A_φ as follows (P_L stands for the point P as seen from the left Kinect K_L , and P_R is the same point as seen from K_R):

$$\overrightarrow{CP_L} = \begin{bmatrix} \cos(\varphi) & -\sin(\varphi) \\ \sin(\varphi) & \cos(\varphi) \end{bmatrix} \cdot \overrightarrow{CP_R}$$

The vectors \overrightarrow{CP} can also be written as $\vec{P} - \vec{C}$, yielding:

$$\vec{P_L} = A \cdot \vec{P_R} + (\vec{C} - A \cdot \vec{C})$$

We have thus written the rotation around C using a matrix A and a translation vector $\vec{b} = \vec{C} - A \cdot \vec{C}$ that corrects for the actual center of rotation. The vector \vec{b} also captures the different distances of the two Kinects from the common center C .

Any rotation can thus be expressed as an affine transformation with a matrix A and a translation vector \vec{b} as:

$$\vec{v}' = A \cdot \vec{v} + \vec{b}$$

This works similarly for 3D space. During calibration, we need to calculate estimates for rotation matrix A and translation vector \vec{b} .

Skeletal data from each Kinect may contain errors. In other words, having two Kinects measuring the same sets of movements from different angles reveals inaccuracies in the Kinect internal measuring system. As it turns out, points \vec{v} , as measured by each Kinect, are approximations derived by the learning model inside each Kinect, which converts a 2D depth point cloud to skeletal data. Given the user’s orientation in the space, at different times, we have observed that each \vec{v} and \vec{v}' may be slightly off. We deal with this issue through averaging of data points (once converted to a single coordinate system). This, of course, may affect calibration accuracy. Since we expect that most of the time a Kinect provides reliable data, the more Kinects are added into a scene, the more accurate this averaging approach becomes.

3.2 Linear Regression Algorithm

Our first approach uses linear regression to calculate the affine transformation between two Kinects. At any given time, we take the skeletal joints coming from two Kinects and combine the joints from each skeleton to matrices V and V' , where each column represents the coordinates of one joint, relative to the skeleton's center of gravity. (Centering the points around a center of gravity eliminates the need for a translation \vec{b}). Given that the Kinect data label joints as *head*, *left-hand*, *right-elbow*, etc., we can easily pair respective joints \vec{v} and \vec{v}' from two Kinects.

By multiplying V with its transpose V^T , we average part of the errors out, and get an invertible matrix $V \cdot V^T$. If we assume that the errors in the measurements are completely random and not biased, then the errors are likely to partially cancel each other out in an expression like $\vec{v}_1 + \vec{v}_2 + \vec{v}_3$. Multiplying V by V^T has exactly the effect of replacing each column vector \vec{v} in V by such a sum (each column using a different combination of vectors) and reduces error.

Finally, the estimators \hat{A} and \hat{b} are given by:

$$\hat{A} = (V' \cdot V^T) \cdot (V \cdot V^T)^{-1}, \quad \hat{b} = V' - \hat{A} \cdot V$$

Aggregating all joint measurements from two Kinects as a list of tuples (\vec{v}_i, \vec{v}'_i) , we then get the error ε for an A and \vec{b} as:

$$\varepsilon_{A,\vec{b}} = \frac{1}{n} \sum_{i=1}^n |(A \cdot \vec{v}_i + \vec{b}) - \vec{v}'_i|^2$$

As each measured skeleton yields estimates \hat{A} and \hat{b} , we take the weighted average. Specifically, for each \hat{A}, \hat{b} , we take the corresponding $1 / \varepsilon_{A,\vec{b}}$ as weight, so that those combinations of \hat{A}, \hat{b} that best describe the data contribute slightly more to the overall estimates.

Note that the error $\varepsilon_{A,\vec{b}}$ above is in fact the well-known *sum of squares*. Indeed, finding the coefficients A and \vec{b} in the affine transformation $\vec{v}' = A \cdot \vec{v} + \vec{b}$ is a multivariate *linear regression* problem, which is commonly written with a 4×4 matrix A that includes both the rotation matrix A as well as the translation vector \vec{b} .

As mentioned earlier, during our calibration runs we found that the skeletal data from the Kinects has systematic bias. This introduced an additional *scaling factor* into the matrix A (see above). To counteract this, we introduced a rescaling step into our calculations, making sure that $\det(A) = 1$, and eliminated outliers where such distances differ by more than 10 centimeters.

3.3 Genetic Algorithm

Our second approach uses a genetic algorithm (GA) to calculate the affine transformation (i.e., rotation matrix and translation vector, A and \vec{b} , respectively). In practice, this approach is very successful as it quickly converges to a solution, given a set of simultaneous data from two Kinects, K1 and K2.

We have experimented with various GA configurations. We utilize a genotype of 3 elements, the first element being the

rotation angle φ and the other two elements comprise the translation vector. Since the two Kinects are already set up on the same plane, for efficiency, we solve for 2D space (although solving for 3D space is similar).

Currently, we use a *population* of 200 individuals, which we evolve for 250 *epochs* (genesis, crossover, mutation, and fitness calculation), as follows:

- **Mutation:** the mutation operation used adds or subtracts a small random displacement to each element in the gene, while ensuring that values remain within appropriate limits.
- **Crossover:** the crossover operation takes two parents and swaps the rotation angle φ between them (without changing the translation vector). Thus, new possibilities emerge.
- **Fitness Function:** to calculate fitness, we convert each genotype to the corresponding rotation matrix and translation vector. Then, we use this affine transformation to convert data from Kinect K2 to Kinect K1's coordinate space. We calculate fitness via the *mean squared error* between the two data sets. The closer the two data sets are, the higher the fitness.

This approach can derive a highly accurate estimate of the affine transformation, for a pair of Kinects, within a few seconds on a regular laptop (i.e., MacBook Pro, 2.8GHz i7). In particular, Figure 4 shows how quickly the genetic algorithm converges over a series of runs. In most cases, the evolved solution's fitness is very high (i.e., > 0.9) within the first few generations.

In the next section, we compare the quality of affine transformations derived by the two approaches.

3.4 Evaluation

This section compares the two techniques presented above, which produce an affine transformation to best convert data between the coordinate spaces of two Kinects.

Our experiment has two Kinect sensors placed facing a common space (as in Fig. 3) and containing a single user. We

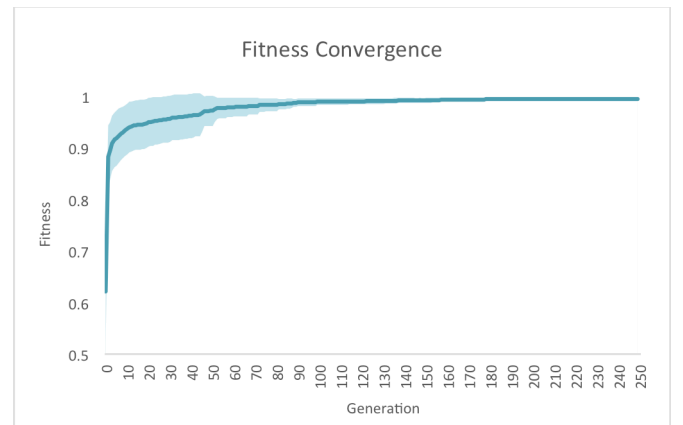


Figure 4: Fitness convergence of the genetic algorithm approach over 20 runs. The line chart indicates the average convergence across all runs. The area chart indicates the 95% confidence interval.

arbitrarily select one sensor as the *primary*, and the other as the *secondary*. Both sensors face parallel to the floor at the same height. We recorded live Kinect skeletal data of the user's movement producing several data sets.

From the first data set, we derive an affine transformation using each of the above methods.

In order to remove timing discrepancies between the two Kinects, we search each data set and select pairs of skeletons (one from each Kinect sensor) that arrived almost simultaneously (within a fraction of a second). Hundreds of such pairs were selected.

For each of these pairs, we apply each affine transformation to the secondary Kinect's skeleton, creating two versions in the primary space, the original, unaltered primary data, and the transformed secondary data. We calculate the *mean squared error* between the two data sets (which is the same as the fitness function used in the GA approach). We plot the resulting fitness scores (see Fig. 5).

Data sets *A* and *B* contain movement of the user across the floor throughout the space shared by both sensors. In data set *C*, the user remains in place, and only moves his arms. Data set *D* combines all above movements, i.e., the user moves throughout the floor space while simultaneously moving his arms.

Our evaluation results show that the fitness of the genetic algorithm solution remains close to 1 throughout all data sets. On the other hand, the fitness of the affine transformation created via linear regression varies significantly (between 0.84 and 1) within any given data set. This suggests that the generic algorithm approach is much better / far more accurate.

Finally, not only the solution derived by the genetic algorithm is pretty accurate, but also, as seen in the previous section (Fig. 4), it gets discovered very quickly. This means we could be running the error-checking algorithm continuously in the background, and, if at some point the error drastically increases (i.e., one of the sensors was moved, accidentally or otherwise), we could re-run the calibration algorithm automatically to update the affine transformation, as needed. This further increases the robustness of the presented approach.

3.5 Scaling up to More Sensors

This section describes how our approach scales up to more than two sensors. Adding more sensors is straightforward. It involves utilizing the calibration method to pair each new sensor, say K3, with an existing one (K1 or K2).

3.5.1 Capturing Single Area from Many Viewing Angles

One possibility is to have *all sensors oriented so that they observe part of the same scene (or area) with the main sensor*. In order for this to work, all additional sensors, K2 to Kn, have to be paired directly to the main one, K1, i.e., to directly transform data from these sensors to K1's coordinate system. This way, all sensors can be calibrated as described above.

The advantage of this approach is that such a setup can handle multiple degrees of occlusion, i.e., where 3 or more people line up partially and obstruct each other, from the point of view of the

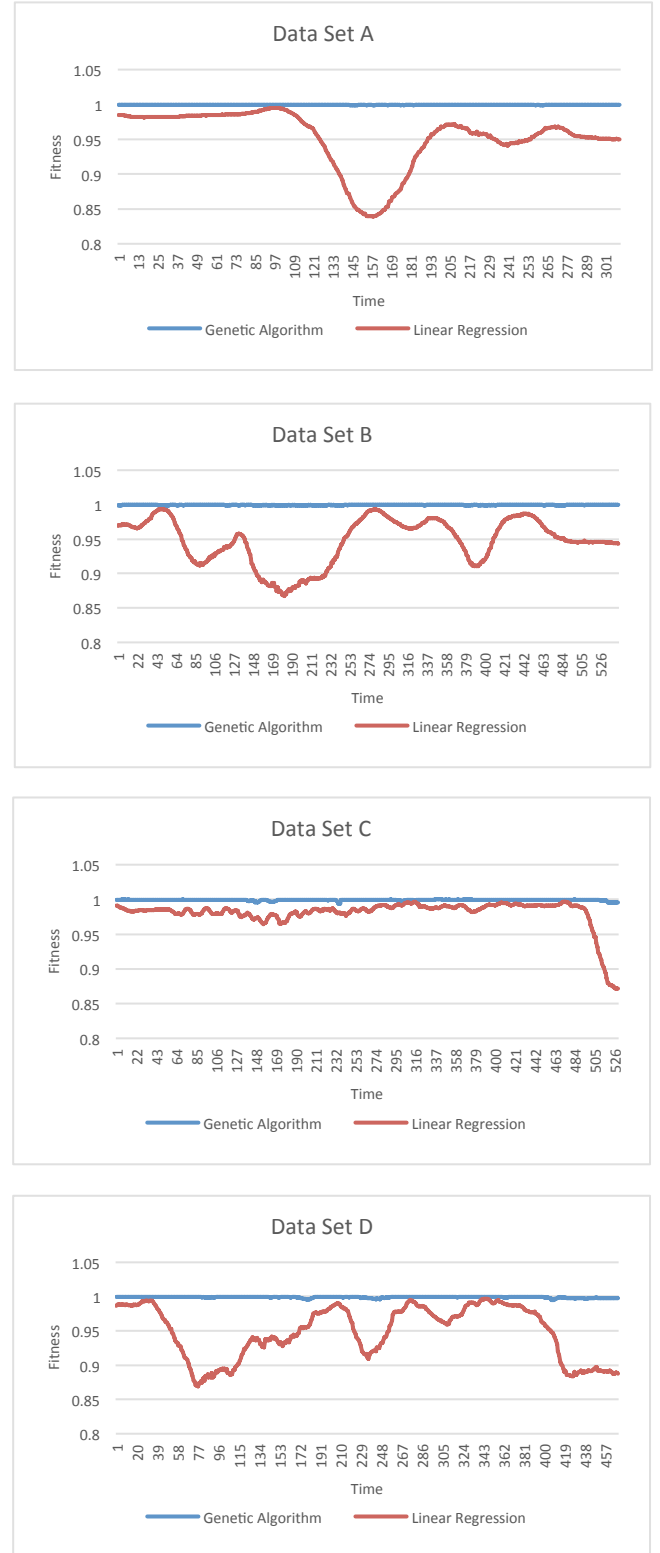


Figure 5: The results of evaluation, testing *fitness* over *time*, for the genetic algorithm (top) and linear regression (bottom) solutions, respectively, on data sets *A* through *D*. Time (x-axis) is represented in data frame units (1/30 sec).

main (or other) sensor. Another advantage is that scaling up does not add any new computational costs, as the calibration and transformation operations can run in parallel, utilizing each Kinect Client's separate processing power (see section 4 below).

3.5.2 Capturing Extended Area from Single Viewing Angles

Another possibility is to *spread sensors apart to create a larger sensing area*. In order for this to work, each new sensor, K_{n+1} , has to be oriented so that it observes part of the same scene (or area) with at least one other sensor, K_n (where n ranges from 1 to as many sensors desired). Again, this allows each new sensor to be calibrated with the method described above, so that all data ends up in the same coordinate system (i.e., K_1 's coordinate system, but extended beyond what K_1 can individually sense).

The advantage of this approach is that it can handle arbitrarily large spaces. Of course, as a tradeoff, obstruction situations can be handled to a lesser degree.

In this case, scaling up does increase upfront computation, as the calibration operations cannot be run in parallel anymore – they have to be run in a series: first, we need to calibrate (i.e., pair) K_1 with K_2 ; then, K_2 with K_3 ; and so on. At first, it would also appear that data transformations have to be done in a series (i.e., to map data from sensor K_3 to K_1 's coordinate system, first they need to be mapped to K_2 's system, and then to K_1 's). However, any two consecutive transformations can be combined into a single, new transformation. Therefore, for each sensor K_2 to K_n , we can pre-compute (i.e., at calibration phase) a single transformation to convert this sensor's data to K_1 's space. Thus, the cost of this approach, during regular sensing, is equivalent to the previous one.

3.5.3 A Mixed Approach

Finally, a combination of the above is possible. In other words, we may combine several sensors in a group all observing the same scene (or area), to better handle occlusions and increase data accuracy in one part of the sensing space. This can be repeated for different parts of the sensing space (i.e., introduce additional such groups of sensors all paired together). In-between portions of the space could then be handled with single sensors paired sequentially. A careful design could provide for greater accuracy in certain parts of the space, where such is needed, and lesser accuracy for less important parts of the space.

In summary, the presented system is flexible enough to handle different designs / space sensing requirements, for a variety of installations, with minimal extra computational costs.

4 System Architecture

Our goal is to facilitate rapid prototyping of new experiences by creating a pipeline for motion tracking data, based on multiple Kinect sensors. Our system is an extension of the Kuatro architecture [9]. It gathers data from a variable number of Kinect sensors, converts all data to a common coordinate system, averages data for accuracy, and delivers data to other systems to generate musical, visual, or other experiences.

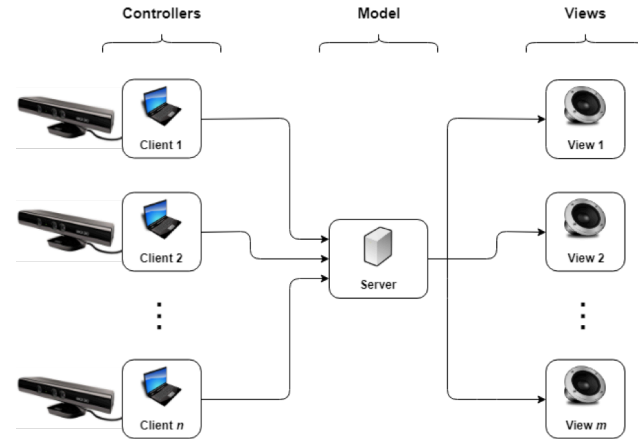


Figure 6: The Kuatro pipeline involves n Clients, which gather data into packets, and send them to a Server. Server processes data, and, for each user, consolidates multiple perspectives into a single / common coordinate system. Server then repacks and sends common coordinate values to m Views. Each View interprets data as desired. Directed connections represent OSC communication.

To accomplish this, we utilize a Model-View-Controller (MVC) architecture consisting of Clients, Views, and a Server (see Fig. 6). Communication between each module is done via an API based on the Open Sound Control (OSC) protocol. The advantage of this approach is that each component may be developed independently from the others, so artists and developers can focus on different parts of the system / experience.

Our current implementation is written in CPython for Clients (due to Kinect API limitations), and Jython for the Server and Views (to make use of the JythonMusic library, which supports rapid prototyping and deployment of musical and other experiences [14]).

In our system, each Client gathers data directly from the Kinect sensor via the Kinect for Windows SDK 2.0. The Client accesses skeletal data at a rate of 30 frames-per-second, and compresses each skeleton into a single OSC message, which is sent to the Server.

The Server receives data from the Clients, processes it via the K-Multiscope algorithm, thus converting to a common coordinate system. As this data model is updated, the Server broadcasts the updated information to the Views. The Server also includes a simple GUI, which shows the data flow of all connected Clients, as well as an interface to control calibration.

Each View receives data from the Server, and acts as the interface to the user. The Server can communicate with several Views at once, and, due to the MVC architecture, each View can approach rendering of data in various ways. Views are specific to the experience desired and could be acoustic, visual, or other.

4.1 OSC Messaging API

Communication between the various modules of the Kuatro is facilitated by an OSC messaging API. Modules are distributed

over several machines, so messages are sent over the network and received at registered port addresses.

The following sections describe the message API between the Clients, Server, and Views. For each message, specific details are provided about its *raison d'être*, and the conditions under which it is generated. This better explains the system architecture and its processes.

4.1.1 Client-to-Server API

Kuatro Clients may send the following OSC messages to the Kuatro Server:

- **/registerDevice** – Registers a new Client to the Server. The arguments include the IP of the Client to identify it (hereafter referred to as the Client's ID) and the type of sensor connected.
- **/newUser** – Notifies the Server that a Client has identified a new user. Because each sensor has limited information, this notification may mean a user who is already visible to one Kinect, has been seen by another Kinect in a space of overlapping perspectives. The arguments include the Client ID, a unique identifier from 0 to 5 representing the new user (as each Kinect can track up to 6 unique users), and the XYZ coordinates of the user's center of mass.
- **/lostUser** – Notifies the Server that a Client has lost tracking on a user. The arguments include the Client ID, and the user's identifier.
- **/jointCoordinates** – A single user's skeleton, including up to 25 joints' XYZ coordinates. The arguments include the Client ID, user identifier, and a joint name, x, y, z, and tracking confidence for each joint in the skeleton. Individual joints can be enabled or disabled in the Client.
- **/handState** – A user's hands can each be (a) open, (b) closed, or (c) pointing (called "lasso"), provided they are being tracked. The arguments include the Client ID, user identifier, hand (left or right), and the hand's state.

4.1.2 View-to-Server API

Each View must register with the Kuatro Server using the following OSC message:

- **/registerView** – Registers the View with the Kuatro Server so the Server will begin sending data to it. Arguments include the IP address of the View and the port at which the View receives OSC messages.

4.1.3 Server-to-View API

The Kuatro Server may send the following OSC messages to the Views:

- **/newUser** – Notifies the View that a new user has been identified. The Server will only report unique users to the Views and will not report a new perspective being established. The arguments include a unique identifier ≥ 0 representing the new user (these are distinct from those reported by the Clients), and the XYZ coordinates of the user's center of mass.

- **/lostUser** – Notifies the View that a user has been removed, as there are no Clients tracking it. The only argument is the user's identifier.
- **/jointCoordinates** – A single user's skeleton, including up to 25 joints' XYZ coordinates. These coordinates take into account each sensor's view of the user, which have been combined through the K-Multiscope algorithm. The arguments include the user identifier and a joint name, x, y, z, and tracking confidence for each joint in the skeleton. This list will include each joint which is reported by at least one of the Kinects.
- **/handState** – The hand state seen by the majority of perspectives, or the higher state in case of a tie, is reported to the View. The arguments include the user identifier, hand (left or right), and the hand's state.

5 Case Study: Liminal Space

Liminal Space is a piece for cello, motion capture, and interactive software. It was developed in parallel with this work and demonstrates our approach to utilizing Kinect sensors via the Kuatro architecture [9] and JythonMusic [14] for developing interactive music installations and experiences.

"Liminal Space" was first performed in Limassol, Cyprus, July 2018 (see Fig. 7), as part of the juried musical program of the Sound and Music Computing conference. The design, architecture, and technical approach behind this piece are presented herein for the first time.

The piece's narrative design explores what happens when the past – J.S. Bach's Sarabande from Cello Suite No. 1 in G major (BWV1007) – meets the present, i.e., movement computing, stochastic music, and interaction. Through the use of Kinect sensors, sound spatialization, and algorithmic composition, a new musical interface is created, between a cellist and a dancer.

The piece has five sections. Each section has predetermined musical phrases (different sets for different sections). In different



Figure 7: Performance of "Liminal Space" in Limassol, Cyprus with Erin Leigh Butcheck (dance) and Leslie Jones (cello). Also, see <http://bit.ly/liminalspace2018b>.

sections, different movements of the dancer (sometimes the left hand, sometimes the head, sometimes the right foot) trigger sounds to play. The dancer knows well what section we are in, and what movements cause what sounds. She also drives the transition between sections, through her movements.

Compositionally and choreographically, we find that there is a fine balance between free expression (i.e., movement) of the dancer, and her ability to “play” music with her movements, in the context of this piece. While with the cellist, all her movements are geared towards fine control of her instrument, with the dancer, if all movements make or control sound, then she cannot dance anymore - she a prisoner of her ability (i.e., Midas touch). Creatively, this introduces an interesting dimension to explore. How much, or how little (and when)? The decisions we made can be seen in this video – <http://bit.ly/liminalspace2018b>.

6 Conclusion

We presented our latest results in the development of a multiscope motion tracking system utilizing Kinect 2.0 sensors. The system incorporates a live calibration component, which allows to transform skeletal data from Kinects observing the same or different scenes from various perspectives, into a uniform / common 3D coordinate system. This allows developing motion experiences, which incorporate many performers / users in the same space or different spaces, while reducing obstruction problems and data errors caused by system confusion (i.e., inaccurate tracking).

Future plans include extending the system to incorporate data from different types of motion and depth sensors, such as LeapMotion and RealSense cameras. This approach can expand the design and complexity of future interactive experiences to include multiple perspectives, sensing strategies, as well as local and telematic performers, as originally explored in [5].

ACKNOWLEDGMENTS

Funding for this work has been provided in part by the US National Science Foundation (DUE-1044861 and DUE-1323605), IBM, and the Niarchos Foundation. David Johnson and Seth Stoudenmier contributed to the original development of the Kuatro framework. Leslie Jones (cello), Erin Leigh Butcheck (dance), Paul Helling (audio processing code), and Timothy Ward (recording support) contributed to the development and performance of “Liminal Space”. Timothy Ward contributed to the design and performance of “Veil”.

REFERENCES

- [1] Sam Amin and Je Burke. 2018. OpenMoves: A System for Interpreting Person-Tracking Data. In *Proceedings of 5th International Conference on Movement and Computing, Genoa, Italy, June 28–30, 2018 (MOCO)*, 4 pages. <https://doi.org/10.1145/3212721.3212846>

- [2] Stylianos Asteriadis, Anargyros Chatzitofis, Dimitrios Zarpalas, Dimitrios S. Alexiadis, and Petros Daras. 2013. Estimating human motion from multiple Kinect sensors. In *Proceedings of the 6th International Conference on Computer Vision / Computer Graphics Collaboration Techniques and Applications (MIRAGE '13)*. ACM, New York, NY, USA, Article 3, 6 pages. DOI: <https://doi.org/10.1145/2466715.2466727>
- [3] Seongmin Baek and Myunggyu Kim. 2015. Dance Experience System Using Multiple Kinects. In *International Journal of Future Computer and Communication* vol. 4, no. 1, pp. 45-49. <https://doi.org/10.7763/IJFCC.2015.V4.353>
- [4] Sebouh-Steve Bardakjian, Serguei A. Mokhov, Miao Song, and Sudhir P. Mudur. 2016. ISSv3: from human motion in the real to the interactive documentary film in AR/VR. In *SIGGRAPH ASIA 2016 Virtual Reality meets Physical Reality: Modelling and Simulating Virtual Humans and Environments (SA '16)*. ACM, New York, NY, USA, Article 1, 5 pages. DOI: <https://doi.org/10.1145/2992138.2992139>
- [5] Christopher Benson, Bill Manaris, Seth Stoudenmier, and Timothy Ward. 2016. SoundMorpheus: A Myoelectric-Sensor Based Interface for Sound Spatialization and Shaping. In *Proceedings of the 16th International Conference on New Interfaces for Musical Expression (NIME 2016)*, Brisbane, Australia, Jul. 2016.
- [6] Kai Berger, Kai Ruhl, Yannic Schroeder, Christian Bruemmer, Alexander Scholz, and Marcus Magnor. 2011. Markerless Motion Capture using multiple Color-Depth Sensors. *Vision, Modeling, and Visualization*. The Eurographics Association. <http://dx.doi.org/10.2312/PE/VMV/VMV11/317-324>
- [7] Luke Dahl and Federico Visi. 2018. Modosc: A Library of Real-Time Movement Descriptors for Marker-Based Motion Capture. In *Proceedings of ACM MOCO conference (MOCO'18)*. ACM, New York, NY, USA, 4 pages. https://doi.org/10.475/123_4
- [8] Zhiqian Gao, Yao Yu, Yu Zhou, and Sidan Du. 2015. Leveraging Two Kinect Sensors for Accurate Full-Body Motion Capture. In *Sensors*. 15, 9 (Sep. 2015), 24297–24317. <https://doi.org/10.3390/s150924297>
- [9] David Johnson, Bill Manaris, Yiorgos Vassilandonakis, and Seth Stoudenmier. Kuatro: A Motion-Based Framework for Interactive Music Installations. In *Proceedings of the 2014 International Computer Music Conference*, Athens, Greece, Sep. 2014.
- [10] Yejin Kim, Seongmin Baek, and Byung-Chull Bae. 2017. Motion Capture of the Human Body Using Multiple Depth Sensors. *ETRI Journal*, 39: 181-190. <https://doi.org/10.4218/etrij.17.2816.0045>
- [11] Alexandros Kitsikidis, Kosmas Dimitropoulos, Stella Douka and Nikos Grammalidis, Dance Analysis using Multiple Kinect Sensors. In *VISAPP2014, Lisbon, Portugal, 5-8 January 2014*.
- [12] Bill Manaris and Andrew R. Brown, Making Music with Computers: Creative Programming in Python, Chapman & Hall/CRC Textbooks in Computing, pp. 502, May 2014.
- [13] Bill Manaris, David Johnson, and Malory Rourk, “Diving into Infinity: A Motion-Based, Immersive Interface for M.C. Escher’s Works”, 21st International Symposium on Electronic Art (ISEA 2015), Vancouver, Canada, Aug. 2015.
- [14] Bill Manaris, Pangur Brougham-Cook, Dana Hughes, and Andrew R. Brown, JythonMusic: An Environment for Developing Interactive Music Systems. In *Proceedings of the 18th International Conference on New Interfaces for Musical Expression (NIME 2018)*, Blacksburg, VA, Jun. 2018.
- [15] Microsoft. 2014. Kinect for Windows SDK 2.0: Features. Retrieved March 30, 2019 from [https://docs.microsoft.com/en-us/previous-versions/windows/kinect/dn782025\(v%3dweb.10\)](https://docs.microsoft.com/en-us/previous-versions/windows/kinect/dn782025(v%3dweb.10)).
- [16] Serguei A. Mokhov, Amandeep Kaur, Mehak Talwar, Keerthana Gudavalli, Miao Song, and Sudhir P. Mudur. 2018. Real-time Motion Capture for Performing Arts and Stage. In *Proceedings of SIGGRAPH '18 Educator's Forum*. ACM, New York, NY, USA, 2 pages. <https://doi.org/10.1145/3215641.3215642>
- [17] Sungphill Moon, Youngbin Park, Dong Wook Ko, and Il Hong Suh. Multiple Kinect Sensor Fusion for Human Skeleton Tracking Using Kalman Filtering. In *International Journal of Advanced Robotic Systems*, (March 2016). <https://doi.org/10.5772/62415>
- [18] Tim Murray-Browne and Mark Plumbley. 2014. Harmonic Motion: A Toolkit for Processing Gestural Data for Interactive Sound. In *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME), 2014-06-30*. Goldsmiths, University of London, UK.