# MotionHub: Middleware for Unification of Multiple Body Tracking Systems


Philipp Ladwig
Mixed Reality and Visualization Group (MIREVI)
University of Applied Sciences
Düsseldorf, Germany
philipp.ladwig@hs-duesseldorf.de

Kester Evers
Mixed Reality and Visualization Group (MIREVI)
University of Applied Sciences
Düsseldorf, Germany
kester.evers@study.hs-duesseldorf.de

Eric J. Jansen
Mixed Reality and Visualization Group (MIREVI)
University of Applied Sciences
Düsseldorf, Germany
eric.jansen@study.hs-duesseldorf.de

Ben Fischer
Mixed Reality and Visualization Group (MIREVI)
University of Applied Sciences
Düsseldorf, Germany
ben.fischer@hs-duesseldorf.de

David Nowottnik
Mixed Reality and Visualization Group (MIREVI)
University of Applied Sciences
Düsseldorf, Germany
david.nowottnik@hs-duesseldorf.de

Christian Geiger
Mixed Reality and Visualization Group (MIREVI)
University of Applied Sciences
Düsseldorf, Germany
geiger@hs-duesseldorf.de


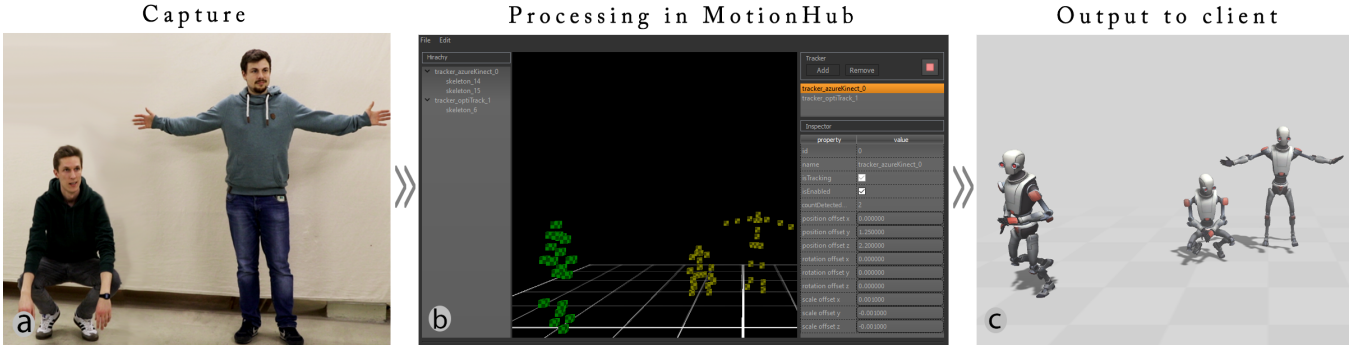


**Figure 1: MotionHub is an open-source middleware that offers interfaces to multiple body tracking systems. a) Two users were captured and tracked by a Microsoft *Azure Kinect*. b) The graphical user interface of MotionHub. The green cubes represent an OptiTrack recording while the yellow ones represent an Azure Kinect live capture. c) MotionHub streams unified skeletal representation in real time to clients such as the Unity game engine via a plug-in.**


## ABSTRACT

There is a substantial number of body tracking systems (BTS), which cover a wide variety of different technology, quality and price range for character animation, dancing or gaming. To the disadvantage of developers and artists, almost every BTS streams out different protocols and tracking data. Not only do they vary in terms of scale and offset, but also their skeletal data differs in rotational offsets between joints and in the overall number of bones. Due to this circumstance, BTSs are not effortlessly interchangeable. Usually, software that makes use of a BTS is rigidly bound to it, and a change to another system can be a complex procedure. In this paper, we present our middleware solution MotionHub, which can receive and process data of different BTS technologies. It converts the spatial as well as the skeletal tracking data into a standardized format in real time and streams it to a client (e.g. a game engine). That way, MotionHub ensures that a client always receives the same skeletal-data structure, irrespective of the used BTS. As a simple interface enabling the user to easily change, set up, calibrate, operate and benchmark different tracking systems, the software targets artists and technicians. MotionHub is open source, and other developers are welcome to contribute to this project.




*MOCO '20, July 15–17, 2020, Jersey City/ Virtual, NJ, USA*



ACM ISBN 978-1-4503-7505-4/20/07…$15.00
https://doi.org/10.1145/3401956.3404185



## CCS CONCEPTS

• **Computing methodologies** → **Motion processing**; • **Information systems** → *Multimedia streaming*;

## KEYWORDS

Body tracking, middleware, skeletal data, motion capture, Azure Kinect, OptiTrack

**ACM Reference Format:**
Philipp Ladwig, Kester Evers, Eric J. Jansen, Ben Fischer, David Nowottnik, and Christian Geiger. 2020. MotionHub: Middleware for Unification of Multiple Body Tracking Systems. In *7th International Conference on Movement and Computing (MOCO '20), July 15–17, 2020, Jersey City/ Virtual, NJ, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3401956.3404185

# 1 INTRODUCTION

Real-time body tracking is an indispensable part of many interactive art performances, mixed-reality applications and games. Due to the progress in research, higher computing power and advances in sensor technology, a large number of BTSs that are suitable for such applications have been developed in recent years. Developers are spoilt for choice: They have to select a system and must consider various advantages and disadvantages concerning price, accuracy and size of the tracking area. However, in many cases not all requirements are known at the beginning of a project and will only be developed over time. This can pose a challenge as a subsequent change to another tracking system can be costly. In such a scenario, a middleware that allows for an effortless switch between different BTSs would be useful.

The term 'middleware' was probably first used in 1968 [35]. Since then, various sub types have been described, but the basic meaning has never changed: A middleware receives, processes and transmits information between at least two peers. Middlewares are often able to understand multiple information representations and to translate them into a standardized or unified one.

The contribution of this paper is the open-source middleware 'MotionHub' and an open-source game engine plugin, which can be seen in Fig. 1b and c. The intention behind our work is to unify the output data of the wide variety of available BTSs. We believe that a unification will not only lead to an easier switch between BTSs during production but that it will also reduce the preparation time for developments that involve BTSs. Integrating a BTS into an existing application requires a manual effort. Then, if developers want to integrate another BTS or exchange the previous one with it, an additional effort must be considered. In some cases, an update to a newer version of a BTS is desired, which produces an even higher manual effort. Our intention is to integrate a generic BTS protocol into applications once in order to reduce the overall time spent for maintenance, the setup and the switch between different BTSs.

Beyond this, more benefits can be mentioned. If a middleware understands different BTSs, it is possible to prioritize and to switch between them automatically to exploit the advantages of each system. A possible scenario could be the use of a BTS that is accurate but limited in tracking space and a different one that has a larger tracking space but is less accurate. A middleware could prioritize the more accurate BTS as long as the tracked person stays in the 'accurate' tracking space and automatically switches to the 'less accurate' space whenever he or she leaves it.

Despite the advantages, the concept of the MotionHub also has a major drawback: By nature, a middleware induces a delay because it receives, converts and resends the data between two ends, which requires a certain processing time. Therefore, this aspect receives special attention in this work and we evaluate it at the end of this paper.

A demo video with a summary of MotionHub's capabilities can be watched here: https://youtu.be/7caURXod-ag.

The Git repository can be found here: https://github.com/Mirevi/MotionHub.

# 2 RELATED WORK

Body tracking is a wide field with many years of research and development, and it has produced a large number of different software and hardware approaches as well as standardizations and file formats. In this section, we only mention the most common and important ones for MotionHub.

## 2.1 Standards and File Formats

The probably best known and oldest de facto standard for data exchange in virtual reality is the *Virtual-Reality Peripheral Network (VRPN)* by Taylor et al. [42]. It offers simple and unified interfaces to a broad range of devices of different manufacturers. Many of these devices share common functionalities such as 6-DOF tracking or button input while the way of accessing these functions differs between manufacturers. VRPN unifies functions across different devices as generic classes such as *vrpn_Tracker* or *vrpn_Button*. Therefore, it can be seen as both a standard and a middleware. The approach of MotionHub is similar, but it focuses on body tracking.

The first official international standard for humanoid animation is the *H-Anim*, which was created within the scope of the *Extensible 3D (X3D)* standard and is a successor of the *Virtual Reality Modeling Language (VRML)* [13, 14]. H-Anim was published in 2006 [15], updated in 2019 [16, 17] and is one of the only efforts yet to create an official open standard for humanoid avatar motion and data exchange.

*COLLADA* and *FBX* are interchange file formats for 3D applications and are widely used today. While humanoid animation is not the focus of COLLADA, its open and versatile structure enables developers to save body tracking data. Compared to COLLADA, the proprietary FBX format mainly focuses on motion data but lacks a clear documentation, which has led to incompatible versions.

While COLLADA and FBX can also be used for writing and reading 3D geometry, the *Biovision Hierarchy (BVH)* file format was developed exclusively for handling skeletal motion data and is therefore simpler in structure. It is supported by many body tracking applications and, because of its simplicity and less overhead compared to other file formats, it is often used for real-time transmission of humanoid motion data. A deeper and more comprehensive overview of further tracking file formats is given by Meredith and Maddock [30].

## 2.2 Software and Hardware

Microsoft started shipping the Kinect in 2010 and thereby allowed the body tracking community to grow substantially since the Kinect was an affordable and capable sensor. During this time, *PrimeSense* made *OpenNI* [39] and *NITE* [39, p.15] publicly available. OpenNI offers low-level access to the Microsoft Kinect and other PrimeSense sensors, while NITE was a middleware that enables the user to perform higher-level operations such as body tracking and gesture recognition. PrimeSense stopped the distribution of OpenNI and NITE after having been acquired by Apple Inc. Although OpenNi

is still available on other web pages [39], there is no active development anymore.

Based on the success of the Kinect and other PrimeSense sensors, many proprietary BTSs such as *iPi Mocap Studio* [27], *Brekel Body* [2] or *Nuitrack* [21], which are based on RGB-D streams, have been developed. Examples for BTSs that use IMU-based tracking are *Xsens MVN* [47], *Perception Neuron* [37] or *Rokoko Smartsuit* [12]. Optical solutions that are based on passive or active (infrared) markers include *OptiTrack Motive:Body* [24], *Qualisys Track Manager* [1], *ART-Human* [18], *Vicon Tracker* [28] and *Motion Analysis Cortex* [23].

The concept of *OpenVR* [29], which is implemented in *SteamVR* [6], and *OpenXR* [26] is similar to the idea behind MotionHub. It unifies the interfaces of a large number of different types of mixed-reality input and output devices to a single library. The fact that it is open source is probably an important reason for its success. While OpenVR and OpenXR do not focus on body tracking, there are efforts to extend their functionalities. For example, *IKenima* [20] has developed an application named *Orion*, which is based on inverse kinematics and utilizes HTC Vive trackers, which are attached to the hip and the feet. IKinema was acquired by Apple Inc. in 2019, and they stopped the distribution of Orion.

*MiddleVR* [33] is a proprietary middleware that supports various commercial input and output devices and offers a generic interface to the Unity game engine [43]. Body tracking is supported, but it is not the focus of the software. The commercial software most closely related to MotionHub is *Reallusion iClone* with its plug-in *Motion LIVE* [25]. It focuses on real-time body tracking and supports several different tracking systems for face, hand and body capture.

## 2.3 Research

The research that is most closely related to our work is *OpenTracker* [40] and *Ubiquitous Tracking* (UbiTrack) [36, 45]. Both systems are generic data flow network libraries for different tracking systems. Unlike MotionHub, they do not focus solely on body tracking. They offer a generic interface to object tracking systems for mixed-reality applications similar to the idea behind VRPN. Although OpenTracker and UbiTrack have a different focus than MotionHub and the research was conducted more than 16 years ago, the concept of unification is similar and reusable for our work.

Suma et al. [41] have developed the *FAAST* (Flexible Action and Articulated Skeleton Toolkit), which is based on OpenNI and NITE. It provides a VRPN server for streaming the user's skeleton joints over a network. However, the development is discontinued. Damasceno et al. [8, 9] present a middleware for multiple low-cost motion capture devices that is applied to virtual rehabilitation. A similar system is suggested by Eckert et al. [10] for the use of playing exergames.

*OpenPTrack* [34] is one of the most recent systems. It is not described as a middleware itself but rather as a BTS. However, since OpenPTrack supports the processing of multiple RGB-D camera streams of various manufacturers and uses different tracking algorithms (such as its own or OpenPose [4]), it rather acts as a middleware. Similar to MotionHub, OpenPTrack is open source and focuses mainly on body tracking. The difference between the two systems is that OpenPTrack solely concentrates on working with RGB-D streams while MotionHub is based on the intention of including different tracking technologies such as optical, IMU-based or magnetic tracking in order to exploit the specific advantages of each individual technology. Therefore, our approach requires a more generic approach to be able to fuse, calibrate and merge the heterogeneous data of different BTSs.

To the authors' knowledge, there is no middleware available today that supports high-cost and low-cost BTSs as well as recent systems and different technologies (not only RGB-D streams) and is open source. Although the concept of a body tracking middleware is not new, MotionHub meets the mentioned aspects and is therefore, to our knowledge, a unique system with a valuable contribution to the community.

# 3 SYSTEM

MotionHub receives raw skeletal data from different BTSs and processes it to create a unified skeleton in real time. The raw data is generated by the respective software development kits (SDKs) of each BTS. Each system uses different transmission methods and protocols, hierarchy structures, units, coordinate systems and different numbers of joints as well as rotation offsets between joints. To perform a correct transformation to the unified skeleton of the MotionHub, each input type requires a specific procedure for receiving, transforming and mapping data. As soon as the data is available in its proper format, it is transmitted to the MotionHub client via the *Open Sound Control (OSC)* [46] protocol.

## 3.1 Unified Skeleton

In order to unify the output data of different BTSs, we transform the heterogeneous skeletal-data structures into a generic skeleton representation consisting of 21 joints, as shown in Fig. 2. The structure of the joints is based on the Unity humanoid avatar skeleton [44], is similar to the *H-Anim LAO-1* standard [3, p.20] and is also used by

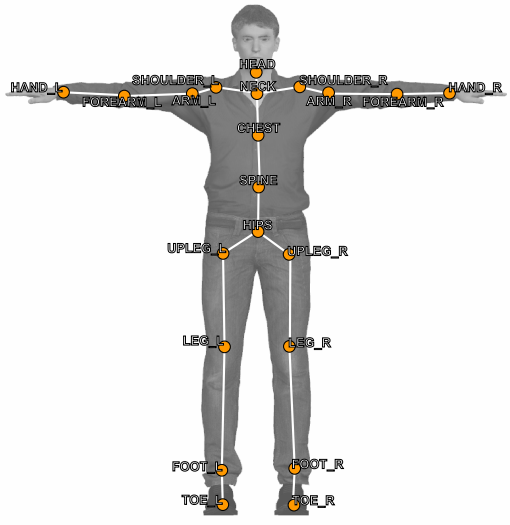


**Figure 2: The unified skeleton structure streamed by MotionHub.**

BTSs such as Azure Kinect. This standard of joint names and joint indices allows for a structured implementation of the transformed skeletal data for avatar animation in external applications. Each joint is represented by a global position (Vector3), a global rotation (Quaternion) and a confidence value (Enum [none, low, medium and high]) in a right-handed coordinate system. All raw skeletal data is processed based on this unification, which will be discussed in detail later on.

Some BTSs like Azure Kinect provide joint confidence values based on distance and visibility. For joints of BTSs that do not provide confidence values, for example OptiTrack, we use 'high' as default value.

## 3.2 Subsystem Architecture

Each BTS has its own native capture frequency and update rate. In order to receive and process incoming data as fast as possible, the data processing code needs to be executed independently from the main program loop. As a solution to this problem, we incorporated autonomous threads for each BTS within the MotionHub. A tracking thread receives raw skeletal data from the respective SDK, processes it and immediately sends it to the (game engine) client. While using threads, it is important to protect memory areas from simultaneous access with appropriate 'atomic' structures. Many consumers such as tracker threads, the UI thread, the render thread or the network sender thread access protected areas. We experienced a better performance and a lower latency of processing and sending data when protected memory was copied to pre-allocated areas first, with a subsequent processing on the copy instead of locking critical sections during the time of processing.

To stream skeletal data to the game engine client, we use the OSC protocol [46] because of its simplicity. The structure of our protocol is similar to the Biovision Hierarchy (BVH) file format but is extended with further MotionHub-specific control messages, which are necessary for the communication between MotionHub and the client side. Furthermore, we created a skeletal-data representation that is more compact than BVH or VRPN-based data streams in order to further reduce the network latency. As our lower-level protocol we prioritize UDP over TCP because we prefer a fast connection and low latency over packet loss recovery. Each OSC package consists of translation (three float values) and rotation (four float values) data for each joint as well as the skeleton ID (integer). Furthermore, a render window is integrated into the UI module to visualize incoming as well as transformed skeleton representations. Skeleton joints are rendered in different colors depending on their confidence values, as shown in Fig. 3.

## 3.3 Dependencies and Build System

MotionHub is written in C++ because of two reasons: First, C++ is considered as one of the fastest programming languages [19], which is an important aspect for a real-time body tracking middleware. The second reason is that building an interface to numerous SDKs is only possible in C++, because a significant number of BTSs only provide a SDK that is based on this programming language.

Because Microsoft's body tracking SDK [31, 32] is based on neural networks, it requires the NVIDIA CUDA® Deep Neural Network library (cuDNN) [38]. For handling matrix, vector and quaternion calculations, the MotionHub uses Eigen [11]. The user interface is created with the Qt 5 framework [5].

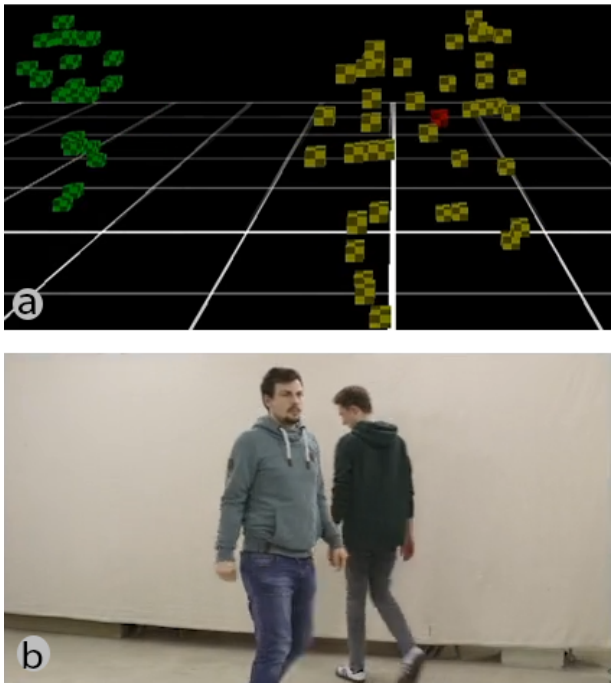


**Figure 3: The OpenGL render window in a) indicates the current tracking confidence of joints by the aid of color. Some BTSs, such as Azure Kinect, provide such values. Yellow joints have a 'medium' confidence value while red joints have 'low' confidence, because they are occluded, as can be seen in b). For BTSs that do not deliver a confidence value, a default value can be selected manually. In this figure, the recording of OptiTrack's value is 'high' by default and rendered in green.**

MotionHub aims to be as open as possible, and it automatically downloads and configures several software dependencies when building with CMake [22]. This fact significantly reduces the required amount of work for developers and allows for an easier development in the future. Binaries are also available. We have chosen Windows as our target operating system, because many SDKs are only available for this system.

## 3.4 User Interface

MotionHub's main UI can be seen in teaser Fig. 1b. A detailed view of the right side of the UI is shown in Fig. 4. The white numbers 1 and 2 indicate the buttons for adding and removing BTSs. Number 3 is a toggle button, which starts or stops the global tracking loop. All BTSs can be started or stopped together or individually.

As shown in Fig. 4 below number 4, selecting a tracker in the list (orange box) will display its properties in the *Tracker Property Inspector*. Different tracking systems have different coordinate systems. For example, OptiTrack has its center in the middle of the tracking area on the floor while the origin of the Azure Kinect lies inside of the camera. When multiple BTSs are combined in one

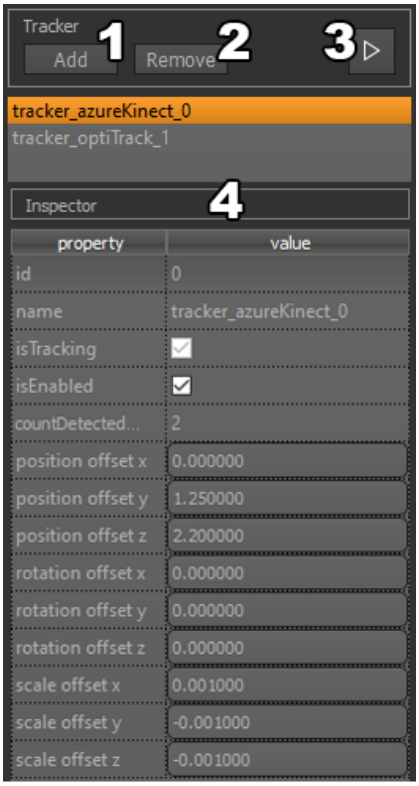


**Figure 4: MotionHub user interface: *Tracker Control Panel* (1–3) and *Tracker Property Inspector* window (4).**

tracking area, their origins must be spatially merged. Therefore, the user can offset the coordinate system origin of different BTSs in the Tracker Property Inspector.

### 3.5 Conversation Matrices

Processing joint position data to the MotionHub's unified coordinate space is done in multiple steps: applying translation, rotation and scale offsets to merge tracking spaces and, if necessary, mirroring the correct axes. Rotations of joints, however, are most complicated and differ between BTSs. A list of specific rotations of the implemented BTSs can be found online in MotionHub's documentation: https://github.com/Mirevi/MotionHub. In addition to the coordinate system, we had to consider the skeletal structure. In some BTSs the skeleton is structured in a hierarchy so that the joint rotations are in local coordinate spaces. These local values are transformed to global rotations by the MotionHub before they are transmitted to the receiver side. For example, each joint rotation of the Azure Kinect system is offset by different values.

$$R_i = I_i O_i^{-1} T$$

The output rotation quaternion $R$ for all joints $i$ of a tracker is the product of the tracker-specific global coordinate system transformation $T$, the inverse global offset orientation $O$, and the raw input rotation $I$.

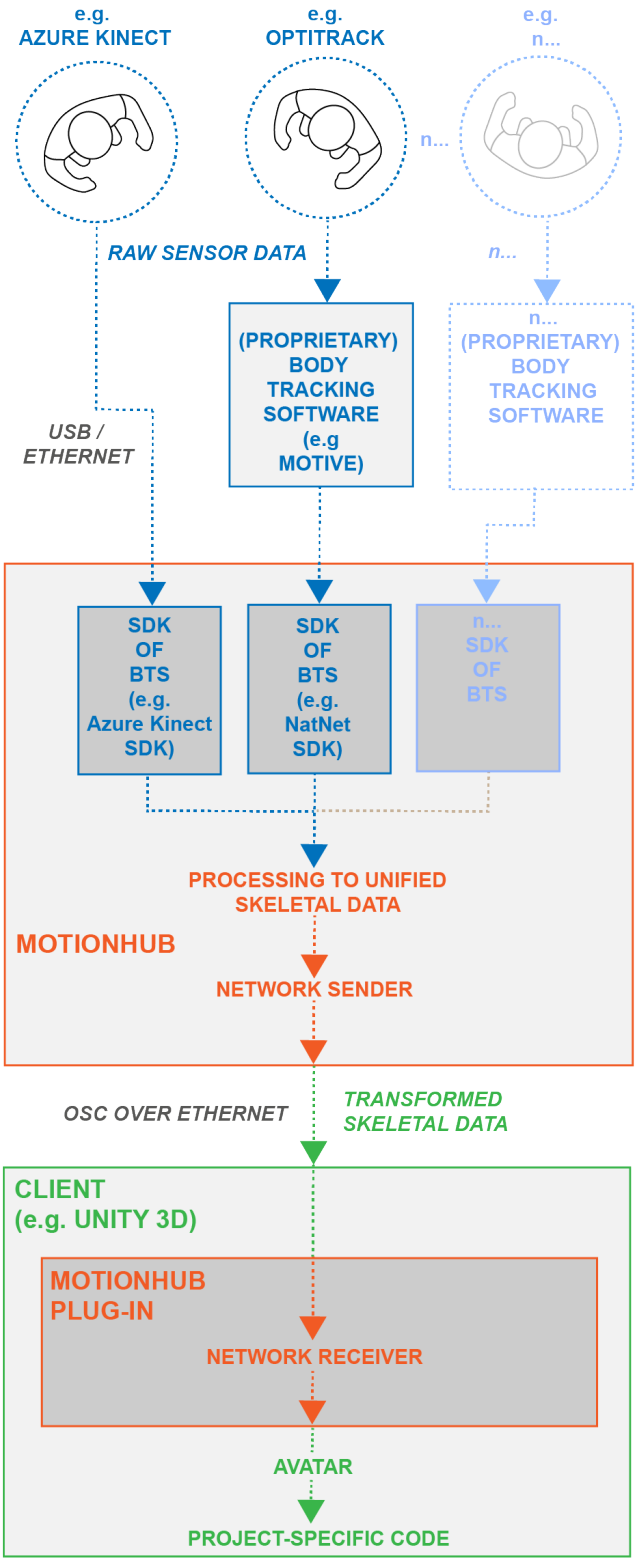


**Figure 5: The data flow (image template kindly provided by Daemen et al. [7]).**

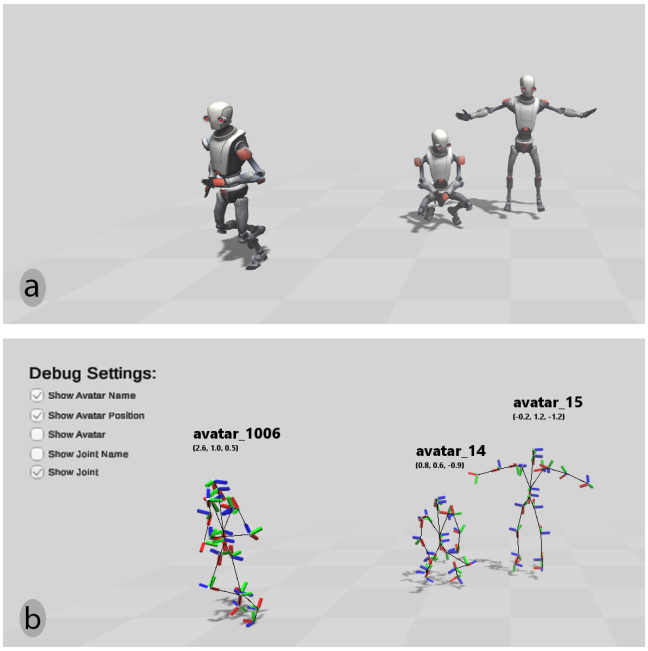


**Figure 6: The Unity game engine plug-in: a) shows the avatar view in the Unity Renderer. b) shows the debug view displaying the position and rotation of all joint axes.**

## 3.6 Game engine client

To be able to receive skeletal data in a game engine, we developed a receiver package for the Unity engine. It contains code that creates a character for each received skeleton and animates it with given rotation values shown in Fig. 6. Character animation is solved in the plug-in code by the multiplication of all inverse joint-rotation values of the character in a hierarchy order and the joint rotation in T-pose.

$$R_{i\,(client)} = I_i T_i \prod_{k=j(i)+1}^{n} r^{-1}_{f(i,k)}$$

For all joints $i$, the transmitted rotation quaternion $I$ is multiplied by the character's joint rotation in T-pose $T$ and the product of all inverse joint rotations $r$ in the skeleton hierarchy above the current joint. While $f(i,k)$ returns the joint which is $k$ nodes above $i$ in the hierarchy, $j(i)$ shows on which hierarchy level the joint is located. The process iterates through the joint hierarchy upwardly, starting with the parent of the joints and ending with the root joint, with $n$ representing the number of iterations. Afterwards, the product quaternion R is applied to the character's local rotation.

During the development of a plug-in and the integration of a BTS into MotionHub, it is critical to preview the processed data in order to identify and debug rotation offsets on different axes. To facilitate this process in Unity, our plug-in is able to visualize debug options, as can be seen in Fig. 6b. These options include the toggle for the display of the skeleton ID, avatar position, joint axes, joint names and avatar meshes. The skeleton ID is equal to the internal one of the MotionHub and is transferred to the client by OSC data packages. Moreover, the plug-in is designed to be avatar mesh independent. This means that a switch between different skeletons is possible without code changes as long as Unity recognizes them as humanoid.


**Figure 7: Playing 'Human Tetris' with the MotionHub. a) shows the physical set-up with OptiTrack and Azure Kinect. b) shows our evaluation game created in Unity 3D.**

# 4 EVALUATION

In order to verify the practical benefit of the MotionHub during the development and production of an application, we created an interactive game called 'Human Tetris'. It incorporates the movements of two users. The procedure is as follows: Player #1 starts with a random body pose. The shadow of this pose is orthogonally projected onto a wall and is freezed when Player #1 verbally confirms that he or she is ready. Next, player #2 needs to prepare and try to copy the body posture and shadow of player #1 while the wall visually approaches the players. As soon as the wall reaches player #2, the game calculates and displays a score depending on how well the posture was imitated. The game is shown in Fig. 7, and a video of the game can be watched here: https://youtu.be/O_5hiweZQhE

## 4.1 Procedures

We conducted three tests: 1.) playing Human Tetris, 2.) measuring the time for switching between BTSs and 3.) measuring the delay induced by MotionHub. For the first test, we played two rounds of Human Tetris and incorporated two Azure Kinects as well as an OptiTrack motion tracking system with 24 cameras (12 x Prime 13 and 12 x Prime 17W) and a suit with 50 passive markers. We ran one instance of each the game, OptiTrack Motive and the MotionHub,

**Table 1: Delays between physical motion and recognized motion by two specific BTSs (MH stands for MotionHub).**

| System @30 Hz | With MH | Without MH | Induced delay |
|---|---|---|---|
| OptiTrack | 127 ms | 114 ms | 13 ms |
| Azure Kinect | 222 ms | 151 ms | 71 ms |

which receives the data of three BTSs simultaneously. Both Azure Kinects and OptiTrack Motive ran on the same PC (Intel i7 6700K, Nvidia GTX 1080). The communication between the BTSs and MotionHub was conducted via *localhost*. First, we played a round with only one Azure Kinect and OptiTrack while we covered the second Kinect in order to prove and illustrate for the demo video that this sensor is deactivated. At the beginning of the second round, the OptiTrack player left the tracking area, and another player uncovered the Kinect and joined the game. After the first test, we gathered qualitative data from the players in a non-structured interview.

The second test encompassed measuring the time required for switching between two BTSs in MotionHub. We utilized our Human Tetris game to conduct the test and switched between OptiTrack and Kinect.

In the last test, we recorded the visual response times with a high-speed camera to be able to measure the induced delay. Therefore, we recorded the time between a physical movement and the recognition of this movement for each BTS. The test was conducted without MotionHub (visual responses in Motive and Kinect Body Tracking Viewer) and with MotionHub (visual response in the MotionHub render window and in the Unity game engine renderer). MotionHub and the Unity game engine were executed on the same PC (Intel i7 6700K, Nvidia RTX 2080) and were connected by a localhost connection via UDP. We used a computer monitor with a refresh rate of 144 Hz and the camera captured with 1000 frames per second (Sony DSC-RX10M4). OptiTrack and Kinect worked with an update rate of 30 Hz (33.3 ms each). Any higher update rates of the OptiTrack system results in a significant decrease of the Kinect's tracking quality due to infrared light interference with the OptiTrack system. We used Motive:Body v2.0.2 and Azure Kinect Body Tracking SDK v1.0.1.

## 4.2 Results

The first test showed that the concept of MotionHub is applicable. The players were familiar with both BTSs and reported no unexpected behavior of the tracking systems except for a slightly higher delay of the Azure Kinect.

In the second test, the measured time for switching between two different BTSs (from OptiTrack to Azure Kinect) was 8 seconds.

For the third test, the results of the measurements of the induced delay, which we conducted with the high-speed camera, can be seen in Table 1. Previous tests had showed significantly higher delays due to coupling the frequency of sending out data packets with the main application loop of the MotionHub. Shorter delays were achieved by independent tracker threads that send out new data as soon as it is available (without waiting for the main application loop, which also draws the user interface and processes user input). This solution is described in more detail in section 3.2.

We assume that the difference of the induced delay by MotionHub between OptiTrack (13 ms) and Azure Kinect (71 ms) is caused by different refresh rates of the underlying SDKs. OptiTrack Motive and the thread of NatNetSDK within MotionHub exchange data with more than 240 Hz (although the cameras only capture with 30 Hz) while the Azure Kinect Body Tracking SDKs exchange data with 30 Hz.

We did not mention the delay between the renderers of MotionHub and Unity in Table 1 because we were not able to visually identify a difference on the camera images. Both renderers showed synchronized output and therefore the same delay. However, in order to make a statement about the network delay between MotionHub and the game engine, we used the timestamps of the network packets for measuring the time required for sending and receiving an UDP packet on the same PC (localhost), and it constantly stayed below 1 ms even when MotionHub was sending data.

# 5 POSSIBILITIES AND FUTURE WORK

The current status and the concept of MotionHub can be widely extended. For example, a node-based network structure would be possible, in which multiple MotionHub instances with different BTSs could work together simultaneously. Each MotionHub node could stream tracking data to a master node that merges the tracking data into a common coordinate space and performs sensor fusion. Possible applications of such a structure could be improving the tracking quality when tracking a single person or realizing the tracking of a large number of people at the same time. In order to accomplish this, a reliable method of calibrating and spatially merging the coordinate systems of different BTSs must be found. While merging different RGB(-D) cameras is a well-studied problem, it remains an open question how to successfully merge BTSs with different technologies such as IMU-based tracking and optical-marker tracking. Future research is required in this regard.

Another feature of the concept of MotionHub could be to add further tracking modules such as face or hand tracking. Many manufacturers and research publications focus on either body, face or hand tracking. With some minor alterations, MotionHub could be extended to include the tracking of these body areas.
Furthermore, the acquisition of data for machine learning could also be realized because (unified) data of different sensors could be acquired simultaneously within one application. Moreover, this functionality could also be used for benchmarking different systems simultaneously.

# 6 CONCLUSIONS

We have presented and evaluated our open-source system MotionHub. It allows for merging tracking data of different body tracking systems into a standardized skeleton, which can be streamed to clients such as a game engine. The MotionHub induces an additional delay of 13 ms for a marker-based optical tracking system (OptiTrack) and 71 ms for a markerless optical tracking system (Azure Kinect). Our system enables its users to change the tracking system on the fly and without further configurations within the receiver side. MotionHub has pointed out the feasibility of promising features to extend and combine the possibilities of available BTSs. In the future, we intend to further develop those features. We

plan to add a semi-automatic calibration procedure for matching different coordinate systems between BTSs and extend MotionHub with the support of more BTSs such as OpenPTrack [34], OpenPose [4] and others. We hope to contribute a valuable solution for the community.

## ACKNOWLEDGMENTS

We thank the MIREVI Group and the 'Promotionszentrum Angewandte Informatik' (PZAI) in Hessen, especially Ralf Dörner. This project is sponsored by: German Federal Ministry of Education and Research (BMBF) under the project numbers 16SV8182 and 13FH022IX6. Project names: HIVE-Lab (Health Immersive Virtual Environment Lab) and Interactive body-near production technology 4.0 (german: 'Interaktive körpernahe Produktionstechnik 4.0' (iKPT4.0)).

## REFERENCES

[1] Qualisys AB. 2020. Qualisys Track Manager (QTM). (2020). Retrieved February 24, 2020 from https://www.qualisys.com/software/qualisys-track-manager/
[2] Brekel. 2020. Affordable tools for Motion Capture & Volumetric Video. (2020). Retrieved February 24, 2020 from https://brekel.com/
[3] Don Brutzman. 2006. Humanoid Animation (H-Anim). (2006). Retrieved February 29, 2020 from https://x3dgraphics.com/slidesets/X3dForAdvancedModeling/HumanoidAnimation.pdf
[4] Zhe Cao, Gines Hidalgo, Tomas Simon, Shih-En Wei, and Yaser Sheikh. 2018. OpenPose: Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields. *CoRR* abs/1812.08008 (2018). arXiv:1812.08008 http://arxiv.org/abs/1812.08008
[5] The QT Company. 2020. Qt 5. (2020). Retrieved February 12, 2020 from https://www.qt.io/
[6] Valve Corporation. 2020. SteamVR on Steam. (2020). Retrieved February 26, 2020 from https://store.steampowered.com/app/250820/SteamVR
[7] Jeff Daemen, Jens Herder, Cornelius Koch, Philipp Ladwig, Roman Wiche, and Kai Wilgen. 2016. Semi-Automatic Camera and Switcher Control for Live Broadcast. In *Proceedings of the ACM International Conference on Interactive Experiences for TV and Online Video (TVX '16)*. Association for Computing Machinery, New York, NY, USA, 129–134. https://doi.org/10.1145/2932206.2933559
[8] Eduardo Damasceno. 2014. A Motion Capture Middleware for Exergames Increse the precision with Neural Nets. (06 2014).
[9] E. F. Damasceno, A. Cardoso, and E. A. Lamounier. 2013. An middleware for motion capture devices applied to virtual rehab. In *2013 IEEE Virtual Reality (VR)*. 171–172. https://doi.org/10.1109/VR.2013.6549417
[10] M. Eckert, I. Gómez-Martinho, J. Meneses, and J. F. M. Ortega. 2016. A modular middleware approach for exergaming. In *2016 IEEE 6th International Conference on Consumer Electronics - Berlin (ICCE-Berlin)*. 169–173. https://doi.org/10.1109/ICCE-Berlin.2016.7684747
[11] Eigen. 2020. C++ template library for linear algebra. (2020). Retrieved February 21, 2020 from http://eigen.tuxfamily.org
[12] Rokoko Electronics. 2020. Smartsuit Pro - Motion Capture Suit. (2020). Retrieved February 24, 2020 from https://www.rokoko.com/en/products/smartsuit-pro
[13] International Organization for Standardization. 1997. ISO/IEC 14772-1:1997. https://www.iso.org/standard/25508.html. (1997).
[14] International Organization for Standardization. 2003. ISO/IEC 14772-1:1997/AMD 1:2003. https://www.iso.org/standard/30998.html. (2003).
[15] International Organization for Standardization. 2006. ISO/IEC 19774:2006. https://www.iso.org/standard/33912.html. (2006).
[16] International Organization for Standardization. 2019. ISO/IEC 19774-1:2019. https://www.iso.org/standard/64788.html. (2019).
[17] International Organization for Standardization. 2019. ISO/IEC 19774-2:2019. https://www.iso.org/standard/64791.html. (2019).
[18] Advanced Realtime Tracking GmbH. 2020. ART Human. (2020). Retrieved February 24, 2020 from https://ar-tracking.com/products/software/art-human/
[19] Isaac Gouy. 2020. The Computer Language Benchmarks Game. (2020). Retrieved February 27, 2020 from http://benchmarksgame.wildervanck.eu/which-programs-are-fastest.html
[20] IKinema. 2020. IKinema Orion. (2020). Retrieved February 25, 2020 from https://youtu.be/Khoer5DpQkE
[21] 3DiVi Inc. 2020. Nuitrack Full Body Skeletal Tracking Software. (2020). Retrieved February 24, 2020 from https://nuitrack.com/
[22] Kitware Inc. 2020. CMake. (2020). Retrieved February 27, 2020 from https://cmake.org/
[23] Motion Analysis Inc. 2020. Cortex Software. (2020). Retrieved February 24, 2020 from https://www.motionanalysis.com/software/cortex-software/
[24] NaturalPoint Inc. 2020. OptiTrack Motive:Body. (2020). Retrieved February 24, 2020 from https://optitrack.com/products/motive/body/
[25] Reallusion Inc. 2020. iClone Motion LIVE. (2020). Retrieved June 07, 2020 from https://mocap.reallusion.com/iclone-motion-live-mocap/default.html
[26] The Khronos Group Inc. 2020. OpenXR. (2020). Retrieved February 25, 2020 from https://www.khronos.org/openxr/
[27] iPi Soft LLC. 2020. iPi Soft- Markerless Motion Capture. (2020). Retrieved February 24, 2020 from http://ipisoft.com/
[28] Vicon Motion Systems Ltd. 2020. Tracker - Delivery Precise Real-World Data | Motion Capture Software. (2020). Retrieved February 24, 2020 from https://www.vicon.com/software/tracker/
[29] Joe Ludwig. 2020. OpenVR. (2020). Retrieved February 25, 2020 from https://github.com/ValveSoftware/openvr
[30] Michael Meredith and Steve Maddock. 2001. Motion Capture File Formats Explained. *Production* (01 2001).
[31] Microsoft. 2020. Azure Kinect Body Tracking SDK. (2020). Retrieved February 12, 2020 from https://docs.microsoft.com/en-us/azure/kinect-dk/body-sdk-download
[32] Microsoft. 2020. Azure Kinect Sensor SDK. (2020). Retrieved February 12, 2020 from https://github.com/microsoft/Azure-Kinect-Sensor-SDK
[33] MiddleVR. 2020. MiddleVR SDK. (2020). Retrieved February 10, 2020 from https://www.middlevr.com/middlevr-sdk/
[34] Matteo Munaro, Filippo Basso, and Emanuele Menegatti. 2016. OpenPTrack. *Robot. Auton. Syst.* 75, PB (Jan. 2016), 525–538. https://doi.org/10.1016/j.robot.2015.10.004
[35] Peter Naur and Brian Randell. 1969. *Software Engineering: Report of a Conference Sponsored by the NATO Science Committee, Garmisch, Germany, 7-11 Oct. 1968, Brussels, Scientific Affairs Division, NATO.*
[36] J. Newman, M. Wagner, M. Bauer, A. MacWilliams, T. Pintaric, D. Beyer, D. Pustka, F. Strasser, D. Schmalstieg, and G. Klinker. 2004. Ubiquitous tracking for augmented reality. In *Third IEEE and ACM International Symposium on Mixed and Augmented Reality*. 192–201. https://doi.org/10.1109/ISMAR.2004.62
[37] Noitom. 2020. Perception Neuron Motion Capture. (2020). Retrieved February 24, 2020 from https://neuronmocap.com/
[38] NVIDIA. 2019. NVIDIA CUDA® Deep Neural Network library. (2019). Retrieved February 12, 2020 from https://developer.nvidia.com/cudnn
[39] Inc. Occipital. 2020. OpenNI 2 SDK. (2020). Retrieved February 10, 2020 from https://structure.io/openni
[40] Gerhard Reitmayr and Dieter Schmalstieg. 2005. OpenTracker: A Flexible Software Design for Three-Dimensional Interaction. *Virtual Real.* 9, 1 (Dec. 2005), 79–92.
[41] E. A. Suma, B. Lange, A. S. Rizzo, D. M. Krum, and M. Bolas. 2011. FAAST: The Flexible Action and Articulated Skeleton Toolkit. In *2011 IEEE Virtual Reality Conference*. 247–248. https://doi.org/10.1109/VR.2011.5759491
[42] Russell M. Taylor, Thomas C. Hudson, Adam Seeger, Hans Weber, Jeffrey Juliano, and Aron T. Helser. 2001. VRPN: A Device-Independent, Network-Transparent VR Peripheral System. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology (VRST '01)*. Association for Computing Machinery, New York, NY, USA, 55–61. https://doi.org/10.1145/505008.505019
[43] Unity Technologies. 2019. Preparing Humanoid Assets for export. (2019). Retrieved February 7, 2020 from https://docs.unity3d.com/Manual/UsingHumanoidChars.html
[44] Unity Technologies. 2020. Unity - Manual: Humanoid Avatars. (2020). Retrieved February 29, 2020 from https://docs.unity3d.com/Manual/AvatarCreationandSetup.html
[45] M. Wagner, A. MacWilliams, M. Bauer, G. Klinker, J. Newman, T. Pintaric, and D. Schmalstieg. 2004. Fundamentals of Ubiquitous Tracking. In *IN ADVANCES IN PERVASIVE COMPUTING*. Austrian Computer Society, 285–290.
[46] Matthew Wright. 2005. Open Sound Control: An Enabling Technology for Musical Networking. *Org. Sound* 10, 3 (Dec. 2005), 193–200. https://doi.org/10.1017/S1355771805000932
[47] XSens. 2020. MVN Animate. (2020). Retrieved February 24, 2020 from https://www.xsens.com/products/mvn-animate