Fusion of Multiple Motion Capture Systems for Musculoskeletal Analysis

Vincent Samy, Ko Ayusawa, Yusuke Yoshiyasu, Ryusuke Sagawa, Eiichi Yoshida

Fusion of Multiple Motion Capture Systems for Musculoskeletal Analysis

Vincent Samy, Ko Ayusawa, Yusuke Yoshiyasu, Ryusuke Sagawa and Eiichi Yoshida[1]

***Abstract*— An accurate and convenient method of measuring human movements is essential for human motion analysis. In recent days, several types of motion capture system became available. Since each measurement technology has both merits and demerits, their suitable choice depends on each application or varying situations. Therefore, it is important that the motion analysis software should flexibly select and be connected to several measurement systems simultaneously according to target applications. This paper presents a software designed to manage different motion capture systems and to perform the musculoskeletal analysis. It allow us not only to get data from different systems but also give us the capability of synchronizing/merging their data.**

I. Introduction

With the rise of AI, human motion analysis has recorded an increasing interest among researchers. Thanks to modern technologies, it can be performed in real-time and crossed many modern domains. It is used in sociology, where an emotion can be extracted from a gait analysis [1] [2]. It also became a first-hand tool for rehabilitation where tracking human motion is vital to correct a patient movement [3]. Comparing to video only, it provides a more precise insight in sport performances [4] like athletics [5] or swimming [6]. And of course, it appears in several biomechanics and biomedical analysis studies [7] [8]. The software that provides all those information has become the keypoint of all these analysis techniques. More recently, Ohashi *et al.* [9] presented their new motion capture system using multiple camera along a spatiotemporal filter. They compared it with an optical system and presented the efficiency of their system.

In the past year [10] we worked on a software to monitor factory worker in order to detect risks of physical health by utilizing the human musculoskeletal model [11]. Mainly, it supported real-time force estimation using either a motion capture system or an IMU-based system. From our motion analysis software we can detect potential risk of lumbar pain using IMU sensors. The main drawback of our system is the number of sensors. To reduce this number we have considered adding a vision-based system and coupled it with IMUs [12]. During these experiments and the data analysis, we suffered from the coupling and the synchronization of the different systems.

*This work was partly supported by JSPS KAKENHI Grant No. 17H00768, No. 18H03315, and No.17K18420.

[1]V. Samy, K. Ayusawa, Y. Yoshiyasu, R. Sagawa and R. Sagawa are with CNRS-AIST JRL (Joint Robotics Laboratory), UMI3218/RL, Tsukuba, Ibaraki, Japan. Corresponding Author: V. Samy vincent.samy@aist.go.jp

Futhermore, along our research in the human motion analysis, we had the need of using several very different motion capture systems. Either to test new sensors or to compare musculoskeletal computation results, we had an increased need of managing that many motion capture systems. We thus decided to have a more dedicated software to this purpose. Ultimately, it provides the following:

1) It facilitates and factorizes communication between multi-process. Therefore, it will be easier for future developers to implement new systems or to review their past implementation.
2) It allows the use of several sensor system at the same time, thus giving the possibility to merge/combine data together. This can be useful to increase accuracy of acquired data or to make software synchronized recording.
3) It gives a user (or a program) the capacity to directly accept or stop the data flow in real-time. It means that we can record data of several systems at once and replay the record while excluding some data. Assuming a failure system exists, this also could be used to discard data of a whole system, while keeping the others alive, if the data are detected to be corrupted or not to exist (*e.g.* out-of-range of an IMU sensor, occlusion of a camera, etc).

This paper is presenting our motion capture systems management software. It is currently compiled as a form of a plugin of the ergonomic assessment support software platform Dhaiba [13]. The software is divided in three main parts. i) communication with motion capture systems, ii) communication with the musculoskeletal library, and iii) user interface.

II. General software process

Our software has been designed so that it can use different types of motion capture systems. To be more specific, we currently have tested it with three kind of systems;

- Optical motion capture system (Motion Analysis)[1],
- IMU-based system (XSens)[2],
- Vision-based system (SkeletonNet) [14].

This allowed us to be able to record a human motion with all motion capture systems at once. Such various set of systems requires different kind of communication process. We then designed a system manager (see Fig. 1) as a Dhaiba [13]

[1]Motion Analysis Corp: https://motionanalysis.com/

[2]XSens awinda series: https://www.xsens.com/products/mtw-awinda/

plugin that handles motion capture systems' output, pack the data and send them to the musculoskeletal library. Then the musculoskeletal library computes the kinematics (generalized coordinates, velocity, acceleration and wire length) and dynamics (wire force, joint reaction force, joint torque) and send the results to the core of Dhaiba. As explained in our previous paper [10], these information are then dispatched across Dhaiba to render a 3D model, to plot, to record or to stream them elsewhere. In parallel, Dhaiba's user interface is used to communicate directly with the user. It enables the switch on/off a system, change global weights, etc. In short, the software allows multi-combination of any of the motion capture systems which means that it can combine two (or all) of them. It also provides on-the-fly (de)activation of a system as well as on-the-fly weight tuning. On-the-fly (de)activation may seem useless at first sight, but it will come to be handy when two motion capture systems run simultaneously and one fails to provide data. Such failure can be caused for several reasons like a system breakdown, an out-of-range sensor, occlusions, etc.

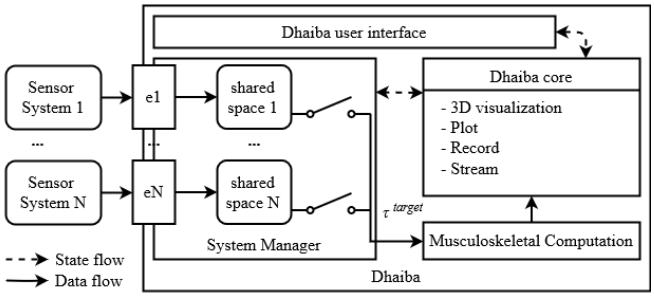

Fig. 1. System manager general process. Data are first received from external motion capture systems and then stored in a shared memory space. Depending on whether or not the user has switch off a system, the system manager fetches the data from the shared space and sends them to the musculoskeletal library. The results are then sent to the Dhaiba core for visualization, recording and streaming. State flow represents the flow of user-defined parameter transmitted through Dhaiba's user interface. Data flow represents the flow of data relative to the musculoskeletal computation.

A. Receiving data from motion capture systems

Inter-process communication and inter-system communication can generally be narrowed down to two main possibility: internet and shared memory. Dhaiba also provides a third way that grants inter-plugin communication. A great advantage of this system is that it can process any kind of data but has two main drawback. One is to implement a dedicated Dhaiba plugin in order to connect to it. Formally, it is mandatory to write a plugin that handles a sensor system output and re-inject the data to the communication system. This system will handle data transmission between plugins and can eventually record them. Another problem is that the current system keeps all information during the whole experiment. At the end, it then dumps all the information into a file. It means that for very long experiment, the computer might use virtual memory or worst, reach an out-of-memory state.

We also have a streaming system to receive data from an internet connection which is the main output form of the optical motion capture system. This system is essential for long distance communication. Depending on the communication mode (TCP/UDP) it may get slower or loose packets. For local use, these considerations are off context.

Lastly we use a shared memory system that provides data sharing between system processes. This is fast, reliable and it can shared any type of data. Although, it requires the processes to run under the same machine.

In all of the three communication protocol, data transmission can be parallelized. So, more than one sensor system can use the same communication protocol at the same time. In our tests, we use a TCP communication along the motion capture system, the Dhaiba protocol along XSens and a shared memory protocol for the vision system. Reasons for this choice are quite simple: i) The motion capture already provides a network connection, ii) we already have an XSens plugin for Dhaiba, and ii) our vision-based system is working under Matlab which provides a good opportunity to use inter-process shared memory.

All data are temporary stored in a dedicated shared space that is refresh at the sensor system frequency. We then need to fetch and manage these data.

B. Data management

We mean by data management the way that data are merged and send to the musculoskeletal library. One of the main problem is to synchronize data that come from various sources. First, note that synchronizing different systems is not trivial and, at the hardware level, it is just impossible to synchronize. Thus, the synchronization is done at the software level.

Since each motion capture system has its own clock and frequency, and hardware synchronization is not an option, we make a time approximation of the data. To ensure that data are the closest in time, we use for each sensor system a shared space that is updated as soon as new data are available. Then, at a given time, we loop over all shared spaces to recover the latest data. Although, each latest data are fetched, it is done recursively so timing may vary a bit between the first fetched data and last one. One way to overcome this problem would be to use multiple threads to fetch all data at once. Even though we fetch data at the (almost) same time, each sensor system having its specific frequency, the shared space won't be updated at the same rate. Moreover, highly dynamic motion increases data coherency loss between motion capture systems. If the frequency is high enough, the time difference between each data fetch is then small and we can neglect it. If not, one could try to implement a prediction algorithm based on previous data. Such estimator is not trivial to build and requires to solve two major problems. One is to have clock synchronization between all motion capture systems. This could be made at the software level by sliding each system clocks to a software reference clock. One is to compute the

prediction itself where the difficulty resides in keeping some kinematic consistency with the model.

As a first version, our current system does not yet implement the multi-threading concepts and prediction. We then make the assumption that the frequency of each sensor system is high enough so that time difference between motion capture systems can be neglected. Once data are fetched, we may combine them. To understand how data are merged, it is necessary to understand how they act on the Inverse Kinematics (IK). Currently, our IK looks like the following:

$$\min_{\mathbf{q},\mathbf{l}} \sum_{\tau} W_\tau \|S(\boldsymbol{\tau}^{target} - f(\mathbf{q}))\|^2 + W_l \|\mathbf{l}_d - \mathbf{l}\|^2 \quad (1)$$

where W_τ and W_l are user-defined weights that should depend on the system reliability, $\mathbf{l}_d$ is a vector of desired wire length or the vector of wire length at rest, $\mathbf{l}$ is the vector of current wire length, $\mathbf{q}$ is the generalized coordinate vector, S is a selection matrix, $\boldsymbol{\tau}^{target}$ is the target of a task τ, $f(\mathbf{q})$ is a non-linear function that can take the following forms:

- $f(\mathbf{q}) = \mathbf{q}$,
- $f(\mathbf{q}) = {}^{p}r_0(\mathbf{q})$,
- $f(\mathbf{q}) = {}^{p}E_0(\mathbf{q})$,
- $f(\mathbf{q}) = {}^{p}X_0(\mathbf{q})$,

where ${}^{p}r_0$ is the absolute position of a point p, ${}^{p}E_0$ is the absolute orientation of a point p and ${}^{p}X_0$ is the absolute transformation (position + orientation) of a point p. Finally, p is chosen so that it corresponds to the targeted point which means that the IK solution converge toward $p = \boldsymbol{\tau}^{target}$. These values are generally computed with a forward kinematics algorithm.

Tasks' target $\boldsymbol{\tau}^{target}$ generally correspond to sensors' outputs, so a way to combine data is throughout the IK itself. Each system provides a set of tasks that should be compatible together (considering that data are received at the same time). Often, sensors measure position and/or orientation of a human body and thus provide directly a task target. So, one last thing for the IK to work properly is to ensure cohesion between data of each motion capture system. In our case, vision and optical systems provide absolute position targets and IMU system, orientation targets. Ensuring cohesion of data between system requires that data are measured in the same reference frame. Since each system has its own reference frame, we decided to transform all data in our model reference frame which we will call world frame. In [12], we have explained how these transformations are done to combine IMU and vision. However, it does not deal with the optical system. We first have added the possibility to merge the optical system with IMU in the same manner of vision and IMU. Since IMU does not require an absolute position of the reference frame, we only needed to deal with matching the reference frame orientation. For optical and vision combination is a bit more tricky. Indeed, both deal with absolute position of the markers and both have their own reference frame. So, to resolve this problem, we have added an translation offset of vision data to match the reference frame of the optical system. This is simply done with

$${}^{M}r_V = {}^{M}r_0 - {}^{V}r_0 \quad (2)$$

where ${}^{M}r_0$ is the translation from the world (the model) frame to the optical system reference frame, ${}^{V}r_0$ is the translation from the world frame to the vision reference frame, and ${}^{M}r_V$ is the transformation to apply to each vision data. For the orientation part, data are transform directly in the world frame. Note that here, we have decided that the world frame origin is placed at the optical system reference frame, this is an arbitrary choice and does not change any results.

Finally, we need to give access to some IK intrinsic variables (like weights) to the user. This is done through Dhaiba's user interface.

C. User interface

The software provides three distinct states. i) Online, ii) Record, and iii) Load. The online state corresponds to the real-time computation of musculoskeletal parameters along with all the Dhaiba tools such as plot and 3D visualization. By default, the record state switches all system sensors and deactivate musculoskeletal computation to minimize the impact of computational tools over recording. Nevertheless, it is still possible to have the Inverse Kinematics (IK) and Inverse Dynamics (ID) running so 3D visualization and plots can output data. It records synchronized data in a CSV file. Lastly, the load state allows to replay data saved in a CSV file. In all states, the user can enable/disable a sensor system, tune weight, etc. An example of the settings is given in Fig. 2. This relatively simple GUI provides some tweaking tools such as system-wide weight and filter weight. It allows easy (de)activation of system for the IK. In the given case, XSens (IMU sensors) and Matlab (vision-based sensor) are switched on while Cortex (optical system) is off. The loaded file is called 'fulltrial' which is a synchronized recording of all three systems during an experiment. From this interface, it is clear that the user can make any combination of any system. Finally, in this example, the output is redirected to another plugin and not sent to an armature.

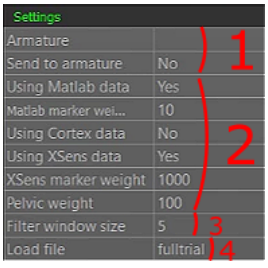

Fig. 2. Different available settings for the load state. In order: 1. Armature relative information for direct display. 2. (De)activation of each system with their respective system-wide weight. 3. Filter for absolute position estimation for vision-based system. 4. File of data to load (Only to load previously recorded file).

III. Implementation design

We currently manage 3 different systems, it could also be possible to have 2 systems of the same kind (*e.g.* 2 vision systems). So we decided to design our library so it may handle several system and adding/removing a whole system can be simplified. Fig. 3 represents how sensor system are managed by the program.

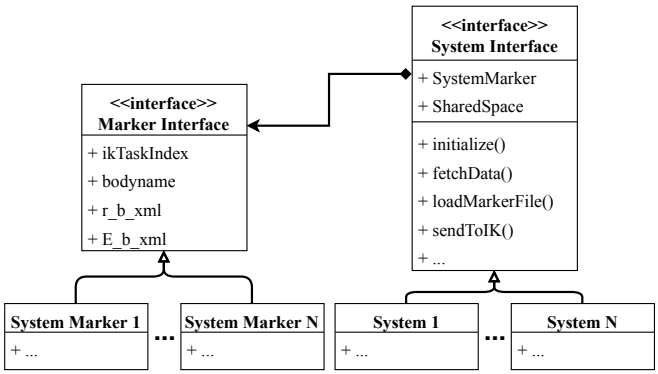

Fig. 3. Simplified model of the sensor system interface. Note that methods' arguments have not been written. A system marker shall inherit from the marker interface that provides system-dependent mandatory variables. A system shall inherit from the system interface which ensure that mandatory functions and variables exists for each system.

Each sensor system should embed information in a form of a marker. Here, we define a marker as a point on a body that represent a given sensor (*e.g.* an IMU, a motion capture marker,...). For vision-like system, this marker is an output of the underlying deep learning system. Markers have predefined sensor relative information such as the body it is attached to and its relative position/orientation. After initialization, it also gets attributed a unique index that corresponds to its task in the IK. This index stay unchanged as long as no task is remove from the IK otherwise it needs to be updated. A system might also embed specific information relative to its sensors. As an example, vision-based system uses its own simplified model, thus, extra information is added to map results of the vision-based model to the IK model. In the IMU case, we have added correction matrix to improve imprecision raising from the difference between IMU placement and predefined orientation.

Then, we have the system interface that will give a basis to all system. First of all, each system comes along with an xml file that is load by *loadMarkerFile()*. This file is used to set the predefine values and fill its corresponding system marker as seen above. The *initialize()* method will provide a task inside the IK for each sensor and associate a sensor to a task thus filling the last marker interface. *fetchData()* pulls data from the shared space. This must be done in thread-safe way, and finally *sendToIK()* update the IK's tasks. This is the bare bone of the *SystemInterface* class but it also embedded a system-wide weight factor, activation/deactivation functions, etc. Also, a sensor system might have system-specific attributes/methods.

Finally, on top of that, a system manager class is added and is given an handle to each system so it can perform a recursive call to each base functions. It can also give access to handle if a system-specific function is needed. Although, a c++ programmer could generated this using dynamic polymorphism and inheritance, since a sensor system is not a runtime object but is created once and for all, we use static polymorphism and Curiously Recurring Template Pattern (CRTP) for better performance. In the case of multi-threading, the manager does not recursively call each base functions but rather recursively dispatch each base function to a thread. The manager class is governed by a user-defined frequency which tells the manager class at what rate it shall fetch data and update the IK.

IV. Experiments

This section presents the examples of experimental results by using the developed software. We conducted the measurement of a human subject with the three measurement systems: the optical motion capture system, the IMU sensors, and a single RGB video camera. The experimental setup can be seen in Fig. 4

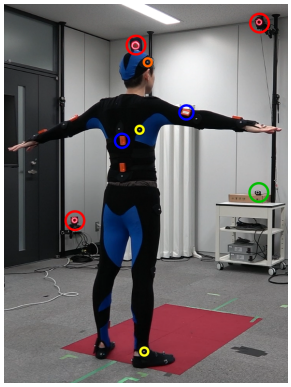

Fig. 4. Experimental setup. Red circles are cameras of the optical system and yellow circles are its markers. Blue circles are IMU sensors. Green circle is the camera used for vision.

Though our software can provide the online analysis, this section shows the results of offline musculoskeletal analysis against the same recorded motion in order to be compared. A more detailed result about the combination of the IMU sensors and a video camera can be found in [12]; the number of attached IMU sensors should be reduced for practical usage, which could be archived by the combination with a vision system.

The muscle tension estimation was made on the right arm biceps brachii caput breve which is mainly responsible for the flexion of the forearm in the elbow joint. It has been computed from a gymnastic motion Fig. 5 and the results are presented in Fig. 6.

As can be seen from this figure, the developed software could record the data streamed from several motion system at the same time during a unique sequence of motion. From the recorded data, the software could compute the inverse dynamics of muscle tensions from the different types of data resource (i.e, the position of optical markers, the orientation of IMU sensors, video images). The software could also

Fig. 5. Snapshots of the experiments. Motion on top is done 3 times and at the bottom, one time with the right arm and one time with the left arm.

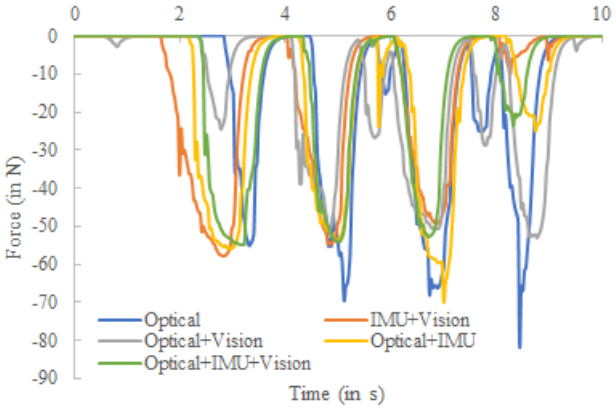

Fig. 6. Plot of the right arm biceps brachii caput breve force of different sensor system combination for the same motion. Data have been recorded during a unique sequence and replay with different settings such as optical, IMU and Vision systems, and their combinations. To not overload the figure we have omitted Vision and IMU alone results. Instead, optical system alone and all 4 possible combinations have been computed.

perform the musculoskeletal analysis not only from single data resource but also their combinations, which indicates the applicability of the software toward various applications.

V. Conclusion and Future Work

We have presented a new software to perform musculoskeletal analysis while integrating the different data resources of several motion capture systems. To realize the integration, the software provides components about communication, data management and user interface. We also made an experiment where we simultaneously recorded the data from IMUs, from optical motion capture and from vision, by using the software. We computed the muscle tension by using the data obtained from the three measurement systems. The software could perform the estimation from the different data resources as well as their combination, and is expected to be used for various types of applications.

This paper mainly focuses on the integration and management of different motion capture. On the other hand, the force sensors (like force plates, pressure sensors, EMG, ...) also provide important information about human dynamic analysis. In our future work, it will be investigated to manage different force sensors and to develop the inverse dynamics that handles different types of force information.

References

[1] J. Peterman, A. Christensen, M. Giese, and S. Park, "Extraction of social information from gait in schizophrenia," *Psychological medicine*, vol. 44, no. 5, pp. 987–996, 2014.

[2] W. Tao, T. Liu, R. Zheng, and H. Feng, "Gait analysis using wearable sensors," *Sensors (Basel, Switzerland)*, vol. 12, pp. 2255–83, 12 2012.

[3] H. Zhou and H. Hu, "Human motion tracking for rehabilitation—a survey," *Biomedical signal processing and control*, vol. 3, no. 1, pp. 1–18, 2008.

[4] S. Barris and C. Button, "A review of vision-based motion analysis in sport," *Sports Medicine*, vol. 38, no. 12, pp. 1025–1043, 2008.

[5] J. P. V. Azcueta, N. C. Libatique, and G. L. Tangonan, "In situ sports performance analysis system using inertial measurement units, high-fps video camera, and the android platform," in *2014 International Conference on Humanoid, Nanotechnology, Information Technology, Communication and Control, Environment and Management (HNICEM)*, Nov 2014, pp. 1–6.

[6] D. D. Rowlands, D. A. James, and J. B. Lee, "Visualization of wearable sensor data during swimming for performance analysis," 2013.

[7] W. Hahn and T. Marr, "Accurate human motion estimation using inertia measurement units for use in biomechanical analysis," 2017.

[8] A. Sano, A. J. Phillips, A. Z. Yu, A. W. McHill, S. Taylor, N. Jaques, C. A. Czeisler, E. B. Klerman, and R. W. Picard, "Recognizing academic performance, sleep quality, stress level, and mental health using personality traits, wearable sensors and mobile phones," in *2015 IEEE 12th International Conference on Wearable and Implantable Body Sensor Networks (BSN)*, June 2015, pp. 1–6.

[9] T. Ohashi, Y. Ikegami, K. Yamamoto, W. Takano, and Y. Nakamura, "Video motion capture from the part confidence maps of multi-camera images by spatiotemporal filtering using the human skeletal model," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct 2018, pp. 4226–4231.

[10] V. Samy, K. Ayusawa, and E. Yoshida, "Real-time musculoskeletal visualization of muscle tension and joint reaction forces," in *2019 IEEE/SICE International Symposium on System Integration (SII)*, Jan 2019, pp. 396–400.

[11] Y. Nakamura, K. Yamane, Y. Fujita, and I. Suzuki, "Somatosensory computation for man-machine interface from motion-capture data and musculoskeletal human model," *IEEE Transactions on Robotics*, vol. 21, no. 1, pp. 58–66, 2005.

[12] V. Samy, K. Ayusawa, Y. Yoshiyasu, R. Sagawa, and E. Yoshida, "Musculoskeletal Estimation Using Inertial Measurement Units and Single Video Image," in *15th IEEE International Conference on Advanced Robotics and Its SOcial Impacts (ARSO 2019)*, Beijing, China, Oct. 2019. [Online]. Available: https://hal.archives-ouvertes.fr/hal-02311483

[13] Y. Endo, M. Tada, and M. Mochimaru, "Dhaiba: development of virtual ergonomic assessment system with human models," in *Proceedings of The 3rd International Digital Human Symposium*, 2014.

[14] Y. Yoshiyasu, R. Sagawa, K. Ayusawa, and A. Murai, "Skeleton transformer networks: 3d human pose and skinned mesh from single rgb image," *Lecture Notes in Computer Science*, p. 485–500, 2019. [Online]. Available: http://dx.doi.org/10.1007/978-3-030-20870-7_30