INTERACTIVE FULL-BODY MOTION CAPTURE USING INFRARED SENSOR NETWORK

Son Duong[1] and Min-Hyung Choi[2]

[1,2] Department of Computer Science and Engineering, University of Colorado Denver, CO USA

ABSTRACT

Traditional motion capture (mocap) has been well-studied in visual science for the last decades. However the field is mostly about capturing precise animation to be used in specific applications after intensive post processing such as studying biomechanics or rigging models in movies. These data sets are normally captured in complex laboratory environments with sophisticated equipment thus making motion capture a field that is mostly exclusive to professional animators. In addition, obtrusive sensors must be attached to actors and calibrated within the capturing system, resulting in limited and unnatural motion. In recent year the rise of computer vision and interactive entertainment opened the gate for a different type of motion capture which focuses on producing optical markerless or mechanical sensorless motion capture. Furthermore a wide array of low-cost device are released that are easy to use for less mission critical applications. This paper describes a new technique of using multiple infrared devices to process data from multiple infrared sensors to enhance the flexibility and accuracy of the markerless mocap using commodity devices such as Kinect. The method involves analyzing each individual sensor data, decompose and rebuild them into a uniformed skeleton across all sensors. We then assign criteria to define the confidence level of captured signal from sensor. Each sensor operates on its own process and communicates through MPI. Our method emphasizes on the need of minimum calculation overhead for better real time performance while being able to maintain good scalability.

KEYWORDS

Motion Capture, Infrared Sensor

1. INTRODUCTION

Technologies and methods that produce high quality motion capture (mocap) data have been available for quite some time. However they are usually out of reach of the mainstream users due to cost and complexity. The traditional mocap setups like optical or magnetic systems require a cumbersome configuration of markers and/or sensors on the actor. There have also been research about markerless motion capture but they also require complex pre-processing and calibration in a laboratory environment. Another drawback is the high cost of these systems, and the data itself needs a lot of post processing. Because of those conditions, while traditional methods can yield very accurate data, they can only used in particular application segments. In recent years the entertainment industry has started introducing cheap and simple devices to stimulate some basic motion capture technology to the mainstream market. The Wiimote and Playstation Move have incorporated tracking sensors to record hand movements to estimate motion (or pose) and apply the motion into target applications. Based on computer vision technology, Playstation Eye and Microsoft Kinect have drawn a lot of interests for casual applications. These devices employ infrared sensors to track an actor's position and allow some basic visual interaction with on screen objects. While they do not measure up to the traditional motion capture system in term of

DOI : 10.5121/ijcga.2013.3404

accuracy and detail, all of these devices are meant for interactive use, fast response, efficient data processing at very low cost.

The Kinect Device is one of the first widely accepted motion sensors released to mainstream users. It comes with two sensors, an infrared and a RGB sensor, but this paper only focuses on the use of the infrared sensor for mocap, although we might explore the potential of the RGB sensor in the future. However the techniques presented here is based on receiving and processing data from any infrared device, and should work with other devices, such as the Playstation Eye with iPisoft's Markerless Motion Capture suit.

Since its inception, there have been various projects in Kinect communities to promote intuitive user interactions, from patience monitoring [41] to robotics and surgical operations [42]. However Kinect has many draw back such as inconsistent data, noise, very limited actor orientation and compromise in accuracy for responsiveness. This paper addresses some of the disadvantages while remains simple and inexpensive enough that it can still be used interactively. Specifically these are main contributions that this paper seeks to address and correct or improve:

- Limited actor orientation: sensor can only detect an actor facing it directly.
- Inaccurate joint tracking: due to either occlusion or orientation.
- Conflicting and corrupted data: due to signal interference.
- Scalable method that can be used to track up to 360 viewpoints for full body.

2. RELATED WORK

2.1. Markerless Motion Capture

The idea of using markerless motion capture has been active for quite some time. While cumbersome, a traditional motion cap model using markers can reliably relays the joint position on active model thanked to the attached markers which can be easily identified by some parameters. A markerless motion capture process does not have this data and as thus, has to construct the joint information base on the visual cue. A comparison about the two methods was given by Lanshammar and Persson based on video image [40]. Some popular approaches to constructing the data include using depth images [1,7,19] and RGB image segmentation [5,8], or both[11]. Depth image method first involves generating an intensity map base on the depth info of each pixel. Data calibration such as near and far plane can be used to remove environment background so that only the actor's information is constructed. The next part is to define a region for each key body part. This can be done by either texture map targeting or establishing local planars [44]. Alternatively, a visual hull of the actor is reconstructed into a workable model [4,17,36]. A visual hull can be described as a 3D approximation of the object's shapes, in this case a human actor. The silhouette of the actor for each camera is calculated from a background subtraction using the depth image generated from an infrared sensor. The hull of a human model is then constructed to represent the actor in 3D. There are several methods on how to construct one such as constructing it from segmented body part. One approach is to use super-quadric shape with a joint hierarchy to hold them together. Our main interest in this paper is on the accuracy of the joint hierarchy itself instead of the model presentation. Once we have the correct joint data we can then use it to rig any general human mesh for demonstrating purpose. In case of an absolute model presentation is needed, a full body scan of the actor will be needed. S. Corazza et. al provided another method by constructing a visual hull base on a point cloud system with embedded kinematic constraint, this method while produce a very accurate model it is computing intensive [2,8].

2.2. Motion Analysis

Upon reconstructing the model, the next step in markerless motion capture is how to recognize the actor's motion based on the cue. Depending on the need of the application the methods are varied. In the case of single sensor the interest lies more within detecting a specific set of poses for interaction. This case usually treats the depth image as a 2D space image[13,26,28]. To be able to analyze real 3D motion the process usually involved a set of training data.[21,24,29] This is usually done by training the data through a variety of human pose[30,33,34]. In general, markerless motion cap is about understanding a set of poses and interpreting the motion in between. The quality of the final animation depends on the original training set (synthesis image) and the decision tree.[1,38,39]. Synthesis image is created from a random set of parameter including a depth image and 3D texture meshes.

2.3. Occlusions and Collision

One of the most significant disadvantages of markerless motion capture is occlusion. Since the data come from a view of a sensor instead of being generated on the actor, it is quite common that certain joints will not be seen by a sensor at certain pose. A common method to deal with this issue on single sensor by using an inferred or prediction system: as in using the position of other visible joint to make a guess for where the occluded joint is. This can be quite confusing not only for the occluded joint but also for the joint in front since the algorithm might be confused in separating the 2 joints. Vladimir Dochev, Tzvetomir Vassilev, and Bernhard Spanlang proposed a method of using multiple layers of depth map to detect occlusion [3]. It divides the depth map into several layers and separated the body parts base on their depth level. Two depth maps of the actor are taken from the front and back then decomposed and labeled into difference body parts. Each body part is then given a bounding box (BB) volume. The BB volume is then used to detect collision between body-parts. This method can be used by any infrared sensor. In our prototype process however the collision approach is generally too expensive for interactive application. Another approach using Image Space interference test was introduced [26] but it is also too expensive for our purpose.

2.4. Sensor Calibration

Since most markerless motion capture system will require quite a few sensors to be able to record the full body motion, they have to be calibrated so data from each device can be combined together. A typical problem of markerless systems is noise. Noise is an inherited issue with the model even among high end system. They can come from air and lighting condition, surface property of the tracked object among other thing. Professional grade sensor however does have benefit to address other source of noise. For example in the same system most likely each sensor will have a unique identifier and operate on a different infrared frequency so they don't have to worry about signal interference between sensors. This is a problem with consumer grade sensor (especially for mass produce sensor like Kinect): when working with multiple devices all the infrared signals on each sensor will operate on the same frequency. This means when more than one sensor is pointed at the same direction, they will create some noise and interference. An illustration of this problem is showed in the figure below.

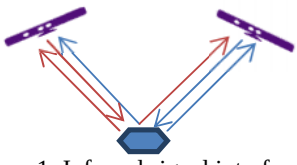

Figure 1: Infrared signal interference

An infrared sensor shoots a beam at the object and waits for a reflective signal, based on this wait time the depth (distance from the sensor) of a particular point is constructed. Due to the scattering rate when a beam hits the surface, it is quite common for this signal to be reflect in different direction and not just toward the original sensor and it's quite possible for another sensor to pick this signal up. In the case of professional systems this is not an issue since each sensor will only pick up the signal of its corresponding frequency while ignoring signal from other sensor. But since all of our sensors operate on the same frequency they can cause interference. Our experience showed that the level of interference depends on the density and orientation of the sensors. For example two sensors with a parallel orientation will suffer a lot more interference comparing to two sensors that are perpendicular to each other. We find that with enough spread (about 4m apart) and wide enough angles (around 60-120 degree) the interference should not have serious effect when working on the skeleton data, and there are suggestions to correct and improve the quality of the Depth Image. One way to improve the depth image is to correct the noise issue. A typical method is Kalman filters to reduce the fluctuation by adaptive hole-filling with spatial-temporal information [9,10,11]. While the depth data is the base where all of our calculation is derived from, we would want a smooth and stable data set to work with, but each filter will come with an overhead cost that needs to be considered for speed. Kai Berger et al. provided a comprehensive method of setting up and calibrate four infrared sensors [6] using the Bougueti's checkerboard tool under different lighting condition.[22] This method is used to calibrate and align the depth sensors using an RGB camera. A checker pattern is printed on an aluminum foil and used as the point of reference. The purpose is to provide a unique predictable IR pattern to be used in the calibration. We found that the matching silhouette approach using Depth Map while might provide a more accurate motion cap (because the Skeleton data is extracted from it) it also has some limitation. It requires an elaborated calibration process to align each sensor, and this is only at a specific point in space. The displacement of an actor can result in the data from each sensor becomes out of phase.

2.5. Unsynchronized Camera Tracking

Camera tracking, a common approach for a single moving camera [4], is mainly used for scene reconstruction and not for motion since it cannot cover all angle at the same time. Unsynchronized multi-camera animation tracking like the one proposed by N.Hasler et al[20] uses an approach that treat the multi-camera case as an extension of the single-moving-camera for the calibration process. In both cases, the main difficulty lies with identifying the sensor position in a 3D space in relative to the scene or tracking object. A common technique is using the Structure-from-Motion (SfM) approach. This technique employs the use of continuous 2D images taking from a moving sensor and identify common feature between images to reconstruct the scene in 3D. First each camera is calibrated using a feature based method against a static background. In the case of infrared sensor this is the distance trajectory from the feature point P to the sensor. In each frame the 2D image information of P is recorded and match with the same point in 3D space. As the camera moves around and a set of consecutive frames are compared and the 3D presentation of P is created usually by triangulation. In the case of multiple sensors, the authors of the method proposed in [20] extend the case of one sensor. Instead of moving one sensor and take different frames continuously, the process is replaced with multiple sensors taking several frames at the same time. Each sensor calculates its own set of SfM data like described above based on their own projection matrix. The theory of SfM reconstruction in this case is: given consistent structure in each image I_n, the point $P_{x,y}$ in each image can be calculated by the projection matrix P_n. The method used in this paper is similar to the pairwise matching described in [45], we further enhance this technique by constructing matching planes between each coordinate space.

3. OVERALL CONCEPT AND DESIGN

Our approach to markerless motion tracking can be summarized as follow:

- Process the data from each sensor and reconstructed more accurate-kinematic base data set.
- Construct a uniform skeleton model on each sensor.
- Use a smart calibration system to reduce complexity.
- Establish a ranking system to merge the data from all sensors as well as providing tools for further optimization.
- Good flexibility and scalability.

While it's possible for a computer to be able to connect and run several Kinects at the same time, the skeleton tracking can only be activated on the first devices on the same computer. We have tried to use file I/O to transfer data between separate process, but this is proven to be very slow even only between two processes. Our design principles are

- Each sensor should run on its own process in parallel
- We assign each sensor to a separate USB hub to secure sufficient bandwidth
- We need a thread-safe mechanism to ensure all sensors are at the same frame

These considerations lead us to MPI in order to handle the data exchange between our processes. We found that the overhead of messaging between process is acceptable, and can be optimized by reduce the complexity of the message to bare minimum. Also since MPI can handle communication between processes on the same computer or communication between nodes in a network without many differences, the use of MPI thus can imply greater scaling flexibility in the case we want to use more devices. Another useful feature of MPI is its Barrier Synchronization routine, in which we use to synchronize our sensor. Assuming we have a powerful enough computer or network, each sensor should be running at the same steady frame rate. However some data lag might appear during the gathering process. One final advantage that makes us choose MPI over other mean of communication is due to its scalability. There are very little differences between handling processes in one computer or processes across a network, this will allow us to add more sensors and computing nodes as needed by our accuracy and computer's processing power without modifying the base code too much.

It is worthwhile to mention the balance between accuracy and speed. A single infrared/RBG is neither a speedy nor accurate by the average animation capture standard. Especially most consumer grade device even when working at optimal condition its sensor is normally capped at 30 frames per second (Kinect, Playstation Eyes). Since we cannot make the device any faster, the goal of our research is to produce a better result from the given raw data. But another goal is to maintain its interactive element. While we intend to maintain fair balance between efficiency and accuracy, but in case of conflict our priority goes to interactivity.

The data structure was designed to handle following tasks effectively: capture the raw data for process, prepare the data to be sent and received via MPI, reconstruct each skeleton at the main process and merging into the final skeleton. For those purpose we have developed following classes:

- jointArray: It is the place holder for the raw data from the Kinect. It also exists at the end of the main process where it is used to hold the final skeleton.

- angleSet: this class is where we hold all the data we processed at each child node, they don't contain the skeleton data itself, but one can be constructed from them. The original spine coordinate is carried over to serve as the starting point when the new skeleton is constructed.
- processSkeleton: this class is created using data from angleSet, it contains a copy of a kinematic built jointArray, and the confident angles. This is what we send to the main process. Once arrived, the data will be used to construct the variable synchAngle at the main process.

Synchronized Sensor Space: each sensor needs to be able to understand their relative position to one another in order for the merging process. And while there are methods to pre-calibrate the sensors base on their setup, we feel it too restrictive for our purpose since the process is complicate for end users and it requires third party tools. In addition, each time the sensors are set up differently or the actor moves out of the designate spot the system may need to recalibrate. We have designed a smart calibration method that will be able to self-calibre as long as the sensors are in reasonable position using the info from each skeleton. It will also allow the actors to move relatively freely without worrying about de-synch the data from each sensor. In another word, calibration and synchronization is done at run time. Input time set up information can help, but not critical (we have it as a safety measure).

Ranking system: one of the biggest drawbacks of infrared device is it only returns optimal data when the actor face the sensor directly, and the quality degrades as the actor turn away from it. We propose a method to address this problem by distributing a priority ranking system based on how far away an actor turned away from sensor.

4. METHOD AND IMPLEMENTATION

4.1. Joint Info Processing

We start by extracting the raw data to fill our jointArray for each device. These includes the position of each joint and a status flag; true means the joint will not be considered for future calculation, false means the data point is good and can be used. For every joint Jn(x,y,z, status):

- Tracked: indicate the joint is actually seen by the sensor and properly tracked. We set the status flag to true.
- Inferred: the joint is not actually seen, but its position is extrapolated using other joint. (For example, based on the wrist and shoulder position the sensor can guess our elbow position). This is mainly how occluded joints are constructed. For our purpose this data is not good enough but we may fall back on it in case a joint cannot be seen by any sensor. We set the status flag to false.
- Unknown: there is no information about this joint from sensor. (Either it doesn't see it, or the joint is outside the FOV or min/max distance). We won't use them for calculation, thus the status flag is set to zero.

4.2. Smooth Filter

To alleviate the noise problem in order to get more accurate data we need some filters. There are many types of filter that can be employed for processing signal noise such as the Savitzky-Golay algorithm that is based on a Least-Squares Polynomial smoothing [42]. The balance between accuracy and speed is an important consideration for choosing the right type of filter. Exponential or double Exponential Smoothing Filter can give good result but also very expensive for interactive applications. Most devices' API and library already comes with a few filters for the

noise generated on a particular sensor but often that's not enough. We focus on the difference between data generated by different sensors that even in the case of perfect synchronization will still not guarantee to be same given the nature of markerless motion capture. In this paper we tackle the issue using several methods, while they will not be independent filters they are used directly as part of our data construction. We construct a uniform model on our sensor based on kinematic data which is balance between speed and accuracy. The reason why we decomposed and reconstruct our skeleton data instead of using the raw joint position data from each sensor is to ensure the uniformity between our presentations. It may seems redundant due to our new data is calculated based on the raw data, but by reconstructing them using only the angle and rotation of each bone, we can impose the same offset for the bone sections between the joint across all sensors. This has the benefit of our new skeleton will have the same orientation value as the raw data, but the noise of the specific joint location is reduced. We also use a joint average filter which is cheap, but reliable if we are able to feed it with good data which are provided by our uniform models and a ranking system.

4.3. Building Angleset

Using the information from jointArray, we break down the 20 joints into 19 set of angles Ax(Ja, Jb, Jc) where x is the angle from one particular joint to another and Ja, Jb, Jc are 3 connected joints from jointArray. We will then use these 19 angles to calculate the new 20 joint location for our uniformed skeleton. The method to construct Ax is as followed: we check the status of the 3 joints component that makes up the angle. For our purpose (Ja.Status && Jb.Status && Jc.Status) has to be true for the set to be taken into calculation. This method is meant to enhance the accuracy and authenticity of the later reconstructed joint, instead of just relying only on the status of each individual joint. For example in the pose where the actor points the hand directly at the sensor, only the hand joint is visible and tracked correctly while the wrist and elbow joints are occluded (and thus inferred). While for simple tracking application the correct position of the hand might be sufficient, this is not a reliable construct in term of hierarchy kinematic.

Once all joints in Ax(Ja, Jb, Jc) is verified, we calculate the angle between them by first calculating and normalize two vector a,b and the angle θ between them, as well as their axis of rotation n:

a = (Ja – Jb),Normalize();
b = (Jc – Jb).Normalize();
θ = DotProduct(a,b).Normalize();
n = CrossProcut(a,b).Normalize();

Also for each Ax that is verified, we set the associated angle's status to true indicating it can be used in reconstruction.

4.4. Skeleton Reconstruction

Once the data of the angleSet A is completed, we begin our first reconstruction. Starting from the spine joint we apply a forward kinematic process basing on the angular information and pre-defined offset to map out the skeleton. The most critical part in this reconstruction process is the rotation matrix that allow us to calculate a new joint by rotate a vector about a certain angle around an arbitrary axis using a single Rotation Matrix R(n, θ). To rotate a unit vector b to the new position b' around an arbitrary axis n by an angle θ, using the rotation matrix R, the derivation is descried as follow:

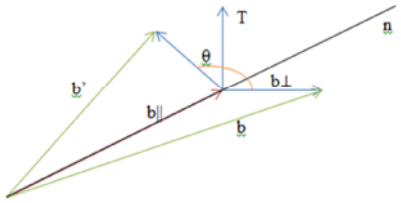

Figure 2: the rotation model

$R(n, \theta) =$

$$\begin{pmatrix} o' \\ p' \\ q' \end{pmatrix} = \begin{vmatrix} \cos\theta(1 - n_x^2) + n_x^2 & n_x n_y(1 - \cos\theta) + n_z \sin\theta & n_x n_z(1 - \cos\theta) - n_y \sin\theta \\ n_x n_y(1 - \cos\theta) - n_z \sin\theta & n_y^2(1 - \cos\theta) + \cos\theta & n_y n_z(1 - \cos\theta) + n_x \sin\theta \\ n_x n_z(1 - \cos\theta) + n_y \sin\theta & n_y n_z(1 - \cos\theta) - n_x \sin\theta & n_z^2(1 - \cos\theta) + \cos\theta \end{vmatrix}$$

Let's call Rn is the normalize-homogenous rotation matrix for joint n, we then multiply a scalar Bn (which varied for each specific joint) to get the new joint location.

$$Jn_{new} = \begin{pmatrix} * & \cdots & 0 \\ \vdots & \ddots & \square \\ 0 & \cdots & 1 \end{pmatrix} * Bn$$

For joints that are properly calculated due to availability of the associate angle Ax, we set the status of the joint to true. For joints that the associated angle data is not available, we temporary fill it with the original sensor data, but set the joint status to false. Structure wise this new joinArray is the same as the initial jointArray with the difference is that it is constructed using hierarchy kinematic as opposed to raw data. There are two purposes to why we construct this new jointArray:

- As discussed above, this helps us to build a more reliable model based on kinematic structure and thus minimize the error that can be produced from the raw data (noise).
- By using the angle information with the same pre-define offset on all sensors, we are able to produce a uniform skeleton model across all process and eliminate the differences result from distance and scaling of each sensor. The uniform skeleton model is a critical key in our calibration and synchronization.

We then embedded this new jointArray into our processSkeleton class with the additional of the two confidentAngle. Then we send this information from each child process to the main process.

4.5. Data Calibration

All the tasks carried so far are on an individual basic by each process working with their respective sensor. This is where our method is different than the traditional calibration discussed in previous works. Usually the data about the setup of the sensors is given at input time to ensure each sensor can understand and interpreted each other, in other methods the set up data is used in after-processing when the data is merged. The method we are describing is doing the opposite. Each sensor works independently without needing the calibration data of other device, but they produce a uniform skeleton model in their respective world space. When these uniformed skeletons are sent to the same space of the reference sensor, they will be synchronized with that space in run time.

When each skeleton arrive at the main process, they are put into the same world space of the reference sensor. This is the sensor defined by user whose coordinate space provides the

orientation and displacement for the final model of which all other sensors are synched to. We can call this our active work space, or global space. At first the skeletons from other sensor will appear to be at arbitrary location and orientation within the reference space because their original local spaces were all different. But thanked to the uniformed structure of all skeleton it's possible to synchronize them to the same point through a series of translation and rotation, and the info needed can be calculated from each skeleton itself.

Translation: the translation offset is simple to calculate. Similar to the approach in [20] the offset between two data sets we can get this information by calculating the difference between the same feature points on each frame of reference. However we don't have to rely on a static cue, instead due to our previous preparation we already have the reference point. Ideally, if each sensor is able to return a perfect set of data then each of our 20 joints can be used as a reference point but this will be most likely not the case. All we need is the distance between each child skeletons and the reference skeleton. For this calculation to be accurate we need to pick a center point on each skeleton. The spine joint is chosen for this given its position at the middle of the skeleton. Also it is the most reliably tracked joint at all time since it is not contested by occlusion in most poses. This is also the root joint for our skeleton construction so it's made a natural selection. For each skeleton T we calculate a displacement vector m with the reference skeleton R. To further enhance the accuracy of the translation, the difference between the hip centers and average the two values are calculated.

After the translation a rotation is needed in order to put all skeletons in the same orientation, this process is more complicate than the translation process above. We establish the idea of a plane that each skeleton is based on. This plane is constructed using a set of 3 joints on the skeleton, which 3 joints is used and under what circumstance need to have these considerations.

- They have to be well representative of the body orientation.
- They need to be the least affected by movement.

Picking the correct 3 joints to construct this plane is a difficult task because the human body can twist. The spine joint is an obvious candidate for the first joint of any set given its location. The set of (spine, right shoulder, left shoulder) joints can construct a good plane to represent the upper body but not necessary the entire body if the actor is twisting the hip, the same applies for the set of (hip center, right hip, left hip). The safest method to construct the plane is using a calibration pose for each sensor when the actor is not twisted. A more dynamic way of doing it will be to average two or more planes from the same skeleton. We then using the cross product to figure the normal vector of each plane, then calculate the angle ϱ between each plane and the reference plane. Since all skeletons are already at the same root from the previous translation, we create a rotation axis around Y-axis using the spine joint. Next we rotate all the joints of each skeleton around this axis with their respective ϱ. The result is now we have all skeleton from each sensor in the same reference frame.

We believe there is a more reliable way to calculate the ϱ angle through the use of an augmented sensor. This sensor can be set to track only the body orientation instead of the full body tracking and thus can be set up and calibrated specifically for this purpose. For example if a sensor can reliably track the two shoulder joints we can use the data to calculate the body angle base on the Z-coordinate of the two shoulders by:

$$\varrho = \mathrm{Tan}^{-1}(\mathrm{shoulderRight.Z} - \mathrm{shoulderLeft.Z}, \mathrm{shoulderLeft.X} - \mathrm{shoulderRight.X})$$

4.6. Merging the Final Skeleton

When all skeleton data is synchronized, they are used to reconstruct the final skeleton. Note that the data from each individual skeleton might not be complete or accurate due to:

- Joints are occluded, and thus their positions are inferred and not reconstructed due to the lack of kinematic data.
- Joint quality's degradation due to the actor is turning away from the sensor.

Our reconstruction process basically selects the best possible data from each skeleton for each joint. To help with this selection progress we propose a ranking and token system.
The processSkeleton class has a confidentAngle with two angles α and β. These angles are used to track how far away the actor is turning away from the camera. Their concept and construction is similar to the construction of the planes as in the planes are constructed using a set of 3 joints on the skeleton, but with a few key differences:

- The planes are compared against the sensor's XY plane and not the reference plane. The angles are calculated based on the normal vector of the skeleton's plane and the Z axis of the sensor.
- The reason that there are two angles is because α represents the orientation of the upper (above the spine joint) body while β represent orientation of the lower body (below the spine joint). Unlike the angle for the sensor calibration where the angle has to uniquely reflect the sensor's orientation, these angles can be varied on the same skeleton. For example when an actor twist in the same pose, a certain sensor might return good data for the facing half of the body while it's better to take the data of the upper from another sensor. To improve accuracy both the vector Vα and Vβ are calculated using the half body average:

Vα = Average(Cross(spine, left shoulder, shoulder center).normalize(), Cross(spine, right shoulder, shoulder center).normalize())
Vβ = Average(Cross(spine, left hip, hip center).normalize(), Cross(spine, right hip, hip center).normalize()).

Table 1: Joint Ranking System

Degree of turning	Rank	Description
$-60^o > x > 60^o$	4	Data degradation can be very bad. Ignore this joint if better data available from other skeletons.
$-30^o > x > 30^o$	3	If this is α, ignore if better data is available. If this is beta, recommend average the value with other skeleton of the same rank.
$-10^o > x > 10^o$	2	Data can generally considered stable enough, recommend average with other skeleton of the same rank if possible.
$-10^o < x < 10^o$	1	The best possible data, it can be used as a standalone although one might want to consider average with skeleton in rank 2. Also considering giving this skeleton set the lead token.

One thing to note is the sensor with the lead token is not necessarily the same sensor being used as the reference, although it can be and technically is ideal for the role. However repeatedly changing the reference sensor is not desirable in term of speed since it will need more synchronization pass (more rotation), and it can cause inconsistency in term of the actor's location and orientation. Our main purpose of introducing this token aside from pointing out which sensor has the best data, it also provides a heuristic locality for adjacent sensors. For

example, the two sensors immediately next to the lead sensors will most likely hold better data than those that are farther away; this can be used for data optimization without paying for expensive arithmetic. It also provides a good indicator for the case when we have no choice but to change the reference camera. This usually happens when the actor turns so far (> |60 degree|) away from the reference sensor that the data it receives cannot be used to reliably calculate the reference angle. In this case we have to change our reference sensor to one that is closer to the direction the actor is facing (and then synch them back to the original orientation). Every sensor can hold the lead token when the actor is facing them, but arbitrary making every sensor into a potential reference has the disadvantages described above. It is better (and cheaper) to specify a specific set of potential sensors to be used as reference, and switching between them depending on their proximity to the sensor that's holding the lead token.

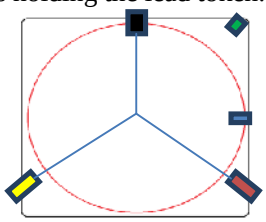

Figure 3: Let Black, Yellow, Red serve as our potential reference sensor. If Green is hold the lead token, then Black will serve as our reference, if blue is holding the lead token than we pick Red as our reference sensor.

Another advantage of this set up is that the user or application can add or remove sensors between the reference sensors without needing to recalibrate the whole system each time it is done. Finally after all necessary preparation, we merge the data and construct our final skeleton with a simple process. We consider all the candidates for each joint from each skeleton. Starting from the sensor with the lead token, we check all adjacent sensors for their joint status and rank. User can set up some constraint on how many sensors away from the lead sensor to reduce the amount of calculation. The joint selection is described in the table below:

Table 2: Joint selection depending on its status and rank.

	Rank 1	Rank 2	Rank 3	Rank 4
Status True	Take.	Take. Average with others of the same condition.	Take or ignore depending on which joint. Ignore if there are better joint available.	Ignore if any better joint is available. Only take if all other skeletons are in the same condition.
Status False	Ignore. If the joint is reported false on all sensor, then take base on confident value	Ignore	Ignore	Ignore.

Note that according to the table the status of a joint take precedent over the ranking system. The reason for this is because even if the actor is facing the camera, a false status means the joint is being occluded, and thus it's better to use joint from another sensor that can track properly even though from a less than optimal angle. In the case of a joint cannot be tracked we will use the inferred data from the top rank sensor, assuming it is in the best position to guess the missing joint.

5. EXPERIMENTS AND RESULT

We run a setup with 3 Kinect Sensors with various test scenarios to verify the concepts we introduced so far.

5.1. Calibration Test

The first test is to check the accuracy of our smart calibration and synchronization method. To perform this test we prepare three sets of data:

- The skeleton data from the reference sensor. (skeleton A)
- The skeleton from a child sensor with an input time (calibrated with physical measurement). (skeleton B)
- The skeleton data from a child sensor synchronized with data calculated in run time. (Skeleton C).

Our metric is as followed: for each joint we calculate the distance of the vector between each joint and the root position. For each n joint we have DAx, DBx, DCx for skeleton A, B, and C. Then we calculate the percent error E as:

$$E_{Bx} = \frac{|D_{Ax} - D_{Bx}|}{|D_{Ax}|} \times 100 \quad ; \quad E_{Cx} = \frac{|D_{Ax} - D_{Cx}|}{|D_{Ax}|} \times 100$$

The graphs below compares the difference between the two types of calibration, the data is taken over 600 frames (about 20 seconds) where the actor performs a variety of poses. The line is an error average between all joints.

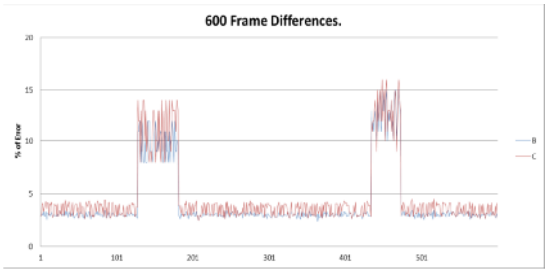

Figure 4 : the percentage of error between two different types of calibration. B(blue) line represents the tradition input time calibration while C(red) line represents the runtime calibration.

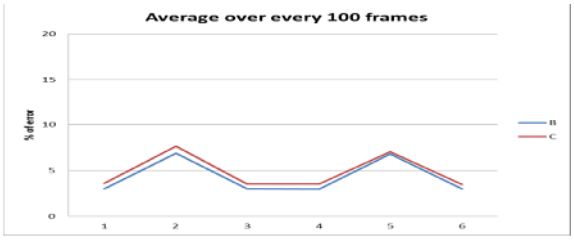

Figure 5: the same data set of Figure 4, but averaged out the value of every 100 frames to better illustrate the differences between two calibration methods.

Note that we're ignoring the spike value showed in the graph since they are represented frames that a particular sensor cannot come up with a good data and those data will likely be provided by another sensor on those frame. Below we include a few screenshots from our prototype to demonstrate the visual result. The first set of screenshots demonstrates the flexibility and accuracy of our calibration method.

For the 3 screenshots in Figure 6: we arbitrary drop two sensor a few meters away from each other and eyeball them toward the same general direction.

- The first screen (a): shows the skeletons from each sensor after the child skeleton (blue) is sent and put in the space of the reference skeleton (red).
- The second screen (b) shows the result of our synchronization, as showed we achieve a pretty good overall overlap between two skeleton, this result was achieved by using no input time calibration data (we don't actually have it anyway since we're setting up this example casually on purpose), as described in our method the calibration data came from our main process itself in run time.
- The third screen (c); we left the programming running, and had a person pick up the child (blue) sensor and arbitrarily moved it around. Basically it was displaced for around a meter in several directions with some small turn left and right, the screen was captured during this process and it shows our method was able to keep up with the changes in real time with the two skeleton are still closely matched.

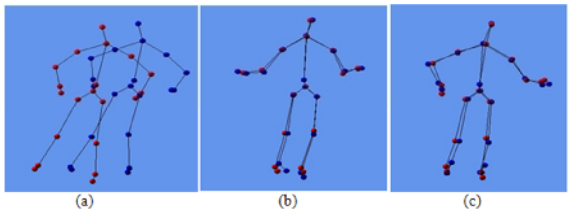

Figure 6: Calibration test: a)The data from 2 sensors placing casually before a synchronization. b) This figure shows the skeleton synced into the same coordinate using our new method, the sensor still at the original position. c) one sensor is displaced (during run time), we can see that the action does not affect our synchronization by much.

5.2. Construction Test

The second set of screenshots demonstrates our merging process and how a new skeleton can be constructed using partial data from multiple skeletons.

For the screenshots in Figure 7, we again set up two sensors but their locations are more deliberated and closer to an actual setup of a camera setup for a traditional marker less motion capture system. The child sensor (green) is put at about 2 meter away from the reference (red) sensor, their direct line of sight from an angle of 30 degrees at their intersection. A reminder here is that we don't make this information available to our system at input time.

- The first screen (a): two skeletons are pre-synched from two sensors. We can notice a few differences here comparing to the first screen of the previous set. First is the angle between the two skeletons. Another difference here is that the two skeletons are rendered using raw data, so they're not uniform. We take this screen to emphasize the differences between them due to scaling caused by distance and camera orientation.
- The second screen (b): show two skeleton synched, basically the same concept of the second screen from the first set.
- The third screen (c): we see another skeleton (blue), this is the newly merge skeleton. We also rendered the original two skeletons (pre-synched) to server as reference. We purposely stretch our right arm completely outside the red sensor FOV and cause its data to be unavailable. But the green sensor being at a different angle, is still able to track that arm as showed on the green skeleton, and thus they were using to construct the missing arm on the blue skeleton (after synched and merged).

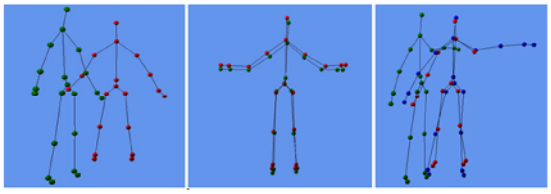

Figure 7 Construction test : a) Two skeleton raw data. b) Two skeletons are synched. c) Merged data, demonstrate using different skeleton to construct missing joints.

5.3. Modeling Test

Visually, this test allows us to render poses that would be otherwise impossible to render with one camera, specifically poses with the actor turning at a great angle and/or with occlusion. Figure 8 shows the skeleton data rendered with rigged and skinned model to demonstrate what is possible before and after the application of our proposal method.

Here we can see the result of the same pose from the same actor:

- The first screen (a): With one sensor: the actor facing the sensor directly. No occlusion and the pose is rendered correctly.
- The second screen (b): the same pose, but the actor turns about 70 degree away from the sensor. Due to occlusion the pose is not modeled and rendered correctly. The overall orientation is mixed up and the occluded arm has a strange rendering. This is not a pose specific problem but in general what would happen to the model when an actor turns too far away from the sensor for reason we have mentioned earlier.
- The second screen (c): with combination of multiple sensors, the pose is now rendered in good quality. The orientation is correct and the occluded part is also rendered satisfactorily.

We match our three sensors setup against the existing XNA Avatar demo from Microsoft. In the Avatar demo, the tracker cannot properly track the actor at a body orientation of more than 30 degree away from the sensor, and occlusion results in a lot of strange animation on the avatar. Our set up on the other hand shows significant improvement with reliable tracking of up to 80 degree of body orientation away from the reference sensor.

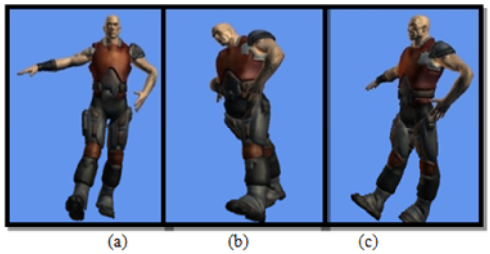

Figure 8: Modelling test demonstrating a correct pose in (c)

6. CONCLUSION

This paper has introduced a new method of merging data from multiple infrared devices, and we have shown that this method can be applied to an accurate and efficient full-body motion capturing system using consumer grade devices on the market. We introduce a novel calibration process which will help lessen constraints in the set up process. Our prototype showed the method

work reasonably well even without extensive calibration steps. This was possible due to the construction of a uniform skeleton and selection of data that enhance the capacity of the sensor and provide a more accurate tracking ability. This in turn gives more flexibility to the action that can be performed by the actor, including greater turn angle, a bigger field of movement, and more reliable occlusion report for a confident pose estimation. The combination of using MPI for direct communication and the lead token/reference sensors allow us to achieve a good level of scalability since the calculation complexity does not scale with data volume. Depending on the need of the application and degree of accuracy/complexity of the application and the desire angle freedom more sensor can be added without making a significant changes in the base method. The entire process is completed with limited overhead ensuring efficiency and scalability. All of these are done using a commodity device available in the market, and thus it may have a wide range of possibility for future applications.

REFERENCES

[1] J. Shotton, A. Fizgibbon, M. Cook et al. Real-Time Human Pose Recognition in Parts from Single Depth Image. *Microsoft Research Cambridge & Xbox incubation.*

[2] S. Corazza, L. Mundermann, A.M. Chaudhari et al. A Markerless Motion Capture System to Study Musculoskeletal Biomechanics: Visual Hull and Simulated Annealing Approach. *Annals of Biomedical Engineering*, Vol 34, No. 6, June 2006.

[3] V. Dhochev, T. Vassilev, and B. Spanlang. Image-space Based Collision Detection in Cloth Simulation on Walking Humans. *International Conference on Computer Systems and Technologies – CompSysTech'2004*.

[4] S. Izadi, D. Kin, O. Hilliges et al. KinectFusion: Real-time 3D reconstruction and interaction. Using a Moving Depth Sensor. *Microsoft Research Cambridge, UK.*

[5] Y. Liu, C.Stoll, J.Gall etal. Markerless Motion Capture of Interacting Character using Multi-View Image Segmentation. *Automation Deparment, TNlist, Tsinghua University.*

[6] K. Berger, K. Ruht, Y. Schroeder et al. Markerless Motion Capture using multiple Color-Depth Sensor. *The Eurographics Association 2011*

[7] L. Ballan and G.M. Cortelazzo. Marker-less motion capture of skinned models in a four sensor set-up using optical flow and silhouettes. *Proceedings of 3DPVT'08 – the Fourth International Symposium on 3D Data Processing, Visualization and Transmission*, 2008.

[8] L. Mundermann, S. Corazza and T.P. Andriacchi. The evolution of methods for the capture of human movement leading to markerless motion capture for biomechanical applications. *For submission to Journal of NeuroEngineering and Rehabilitation.*

[9] M. Camplani and L. Salgado. Adaptive Spatio-Temporal filter for low-cost sensor depth maps. *University of Politecnica de Madrid, Spain*, 2012

[10] M. Camplani and L. Salgado. Efficient Spatio-Temporal Hole Filling Strategy for Kinect Depth Maps. *University of Politecnica de Madrid, Spain*, 2012.

[11] S. Matyunin, D. Vatolin, Y. Berdnikov, and M. Smirnov, Temporal filtering for depth maps generated by Kinect depth sensor,. *3DTV Conference: The True Vision – Capture, Transmission and Display of 3D Video*, May 2011.

[12] K. Lai., L. Bo, X. Ren, and D. Fox, A large scale hierarchical multi-view rgb-d object dataset," in *Robotics and Automation, IEEE International Conference*, May 2011.

[13] M. Siddiqui and G. Medioni. Human pose estimation from a single view point, real time range sensor. In *CVGG at CVPR*, 2010.

[14] Y. Zhu and K. Fujimura. Constrained optimization for human pose estimation from depth sequences. *In Proc ACCV*, 2007.

[15] S. Knoop, S. Vacek and R. Dillmann. Sensor fusion for 3D human body tracking with an articulated 3D body model. *In Proc ICRA*, 2006.

[16] Cappozzo, F. Catani, U. Della Croce and A. Leardini. Position and orientation in space of bones during movement: anatomical frame definition and orientation. *Clin. Biomech.* 10:171-178, 1995

[17] Laurentini. The visual hull concept for silhouette based image understanding. *IEEE PAMI* 16: 150-162, 1994.

[18] A. Kakadiaris and D. Metaxis. Model-based estimation of 3D human motion with occlusion base on active multi-viewpoint selection. In *Prc IEEE CVPR* 81-87, 1996.
[19] W. Matusik, C. Buehler, R. Raskar, S. Gortler and L. McMillan. Imaged-based visual hull. *Proc ACM SIGGRAPH* 369-374, 2000.
[20] N. Hasler, B. Rosenhahn, T.THormahlen et al: Markerless motion capture with unsynchronized moving cameras. *Computer Vision and Pattern Recognition, 2009. CVPR IEEE 2009.* pp 224-231.
[21] Y. Kim, D. Chan, C. Theobalt, S. Thrun. Design and calibration of multi-view of sensor fusion system. *Computer Vision and Pattern Recognition Workshops*, 2008.
[22] J. Bouguet. Sensor calibration toolbox. *http://www. vision. caltech. edu/bouguetj/calib_doc*, 2010.
[23] E. De Aguiar, C. Stoll, N. Ahmed et al. Performance capture from sparse multi-view video. In *ACM Transaction on Graphic* vol 27, p. 98, 2008.
[24] L. Guan, J. Franco, M. Pollefeys: 3d object re-construction with heterogeneous sensor data. *4th International Symposium on 3D Data Processing, Visualization and Transmission.* Atlanta, GA, 2008.
[25] G. Baciu, W.S. Wong, H. Sun. An Image-Based Collision Detection Algorithm. *The journal of Visualization and Computer aniation, pp 181-192.* 1999
[26] Balan, L. Sigal, M. Black, J. Davis and H. Haussecker. Detailed human shape and pose from images. In *CVPR,* 2007
[27] Sundaresan and R. Chellappa. Multi-sensor tracking of articulated human motion using motion and shape. In *Asian Conference on Computer, Hyderabad,* 2006.
[28] T.B Moeslund and E. Granum. A Survey of computer vision-based human motion capture. In *International Conference on Face and Gesture Recognition,* 2000.
[29] M. Yamamoto, A. Sata, S. Kawada, T. Kondo and Y. Osaki. Incremental tracking of human actions from multiple views. In *CVPR, p2-7,* 1998.
[30] D.M Gavrila and L.S Davis. 3-D model-based tracking of humans in action: A multi-view approach. In *Computer Vision and Pattern Recognition, p73-80,* 1996.
[31] Kakadiaris and D. Metaxas Model-Based Estimation of 3D Human Motion. *IEEE Transactions on Pattern Analysis and Machine Intelligence, vol22 no12,* December 2000.
[32] T. Cham and J.M. Rehg. A multiple hypothesis approach to figure tracking. In *Computer Vision and Pattern Recognition,* v.2, June 1999.
[33] D. M Gavrila. The visual analysis of human movement: A survey. *Computer Vision and image understanding*: *CVIU,* 73pp 82-98, 1999.
[34] T. Tan, L. Wang, W. Hu. Recent developments in human motion analysis. *Pattern Recognition,* 36 pp 585-601, March 2003.
[35] Menier, E. Boyer, and B. Raffin. 3D skeleton-based body recovery. In *Proc of the 3rd International Symposium on 3D Data Processing, Visualaization and Transmission, Chapel Hill,* June 2006.
[36] R. Kehl and L.V Gool. Markerless tracking of complex human motion from multiple views. *Computer Vision and Image Understanding, 104(2)pp 190-209,* 2006.
[37] P. Besl and H. McKay. A method for registration of 3d shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence,* 14:pp239-256, 1992.
[38] Cedras and M. Shah: Motion-based recognition: a survey. *Image and Vision Computing 1995,* 13(2) pp 129-155.
[39] J. Aggarwal, Q. Cai: Human motion analysis: a review. *Computer Vision and Image Understanding* 73(3): pp82-98, 1999.
[40] H. Lanshammar, T Persson, V. Medved. Comparison between a marker-based and amarker-free method to estimate centre of rotation using video image analysis. In *Second World Congress of Biomechanics,* 1994.
[41] E. Stone and M. Skubic. Evaluation of an Inexpensive Depth Sensor for Passive In-Home Fall Risk Assessment. *University of Missouri,* 2011.
[42] R. W Scahfer. What is a Savitzky-Golay Filter? *IEEE Signal Processing Magazin,* pp111-116, July 2011.
[43] M. Azimi. Skeletal Joint Smoothing White Paper. *MSDN digital library*, 2012.
[44] V. Lepetit, P. Lagger and P. Fua. Randomized tree for real time keypoint recognition. *In Proc, CVPR* pp775-781, 2005.
[45] T. Thormahlen, N. Hasler, M.Wand, and H.-P. Seidel. Merging of unconnected feature tracks for robustcamera motion estimation from video. *In CVMP,* Nov. 2008