Piotr KOPNIAK

Lublin University of Technology

doi:10.15199/48.2015.08.07

Motion capture using multiple Kinect controllers

Abstract. *The following paper describes the process of developing a motion capture system with the use of Microsoft Kinect 360 and Kinect One controllers. The article presents how to use multiple Kinect controllers parallely and how to employ the existing freeware programming frameworks to produce an appropriate motion capture software. The article shows the results of comparison of single and multiple Kinect motion capture systems. The summary of this study will collect the research results and include some suggestions for the future development of this motion capture system.*

Streszczenie. *Artykuł opisuje proces budowy systemu akwizycji ruchu (ang. motion capture) z wykorzystaniem kontrolerów Microsoft Kinect 360 i Kinect One. Przedstawiono w nim sposób użycia kilku kontrolerów równolegle i sposób wykorzystania darmowych szkieletów programistycznych do stworzenia oprogramowania do rejestracji ruchu. Artykuł zawiera wyniki porównania systemów akwizycji ruchu wykorzystujących jeden oraz dwa kontrolery jednocześnie oraz wskazuje ścieżki przyszłego rozwoju systemu. (**Akwizycja ruchu z wykorzystaniem kilku kontrolerów Kinect**)*

Keywords: motion capture, 3D graphics, game controllers, stereovision, skeleton tracking, Kinect.
Słowa kluczowe: akwizycja ruchu, grafika 3D, kontrolery gier, stereowizja, śledzenie szkieletu, Kinect.

Introduction

Motion capture (mocap - for short) is a human body motion recording technique. Motion capture systems allow us to capture three-dimensional data which is a digital representation of the spatio-temporal structure of the motion. This technique is commonly used in sport to optimize training, in medicine for orthopedic analysis of motion impediments or to verify the rehabilitation process. Motion capture data is a source for analysis in different activity recognitions systems. The most known applications of mocap systems are movie and game production. The data achieved by mocap systems allows us to prepare realistic animations of artificial characters [1, 2].

Motion capture systems consist of special hardware and dedicated software. There are a few different types of mocap systems on the market. Video based systems utilize video cameras, electromagnetic systems use emitters and detectors of electromagnetic field, mechanical systems use external skeletons with potentiometers and finally inertial systems use trackers with accelerometers [3, 4, 5]. Almost all of them have one common feature. They are very expensive. This situation results in a necessity to invent cheaper and more available solutions, e.g. a system which uses PlayStation Eye [6] cameras or Microsoft Kinect [7].

This article describes how to build cheap mocap system with use of Microsoft Kinect controllers and open source computer vision frameworks. Prepared solution allows us to use one or more Kinect devices and includes the software developed in Java language on the basis of OpenNI framework and J4K library. Motion capture system application depends on the system's precision, so it was important to determine the accuracy of the new solution. The described system was tested to find its accuracy in joint angle measurements with the use of one and two capture devices. The results of conducted research represent the main part of the article.

Kinect controller

Kinect is a motion sensing input device developed by PrimeSense and Microsoft for the Xbox 360 video game console and next adopted for Windows PCs. The second version, that is Kinect One was developed for Xbox One console. Thanks to available drivers we may use them with different operating systems. Kinect controllers enable users to control and interact with the Xbox consoles remotely. The user may interact with applications through a natural user interface using gestures and spoken commands.

Kinect 360 has built in structured light emitter (number 2 in picture 1) and two CMOS cameras (numbers 3 and 6). One of them is a RGB camera, and the second is an IR monochromatic receiver for depth measurement. Spoken commands and spatial sound is recorded by a group of microphones with noise reduction (marked as number 1) [8].

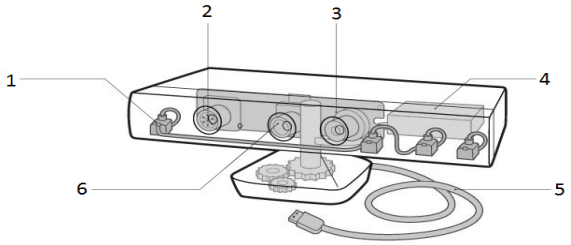

Fig. 1. Diagram of elements of Kinect 360 controller [9]

Kinect 360 controller is based on PrimeSense patented technology [10]. PrimeSense hardware generates a depth map of observed 3D scene. The Kinect combines structured light with two classic computer vision techniques: depth from focus, and depth from stereo for this purpose. Inside structure of Kinect One is not so exactly known yet. What is known it is that Kinect One has got Microsoft X871141-001 SoC which replaces the Prime Sense chip, and three laser diodes for structured light emission.

Laser based structured light emitter projects special pattern of multiple spots having respective positions and shapes. The positions of the spots in the pattern are uncorrelated, while the shapes share a common characteristic. The depth from focus uses the principle that stuff that is more blurry is further away. The astigmatic lens causes a projected circle to become an ellipse whose orientation depends on the depth. Depth from stereo uses parallax. The Kinect analyzes the shift of the speckle pattern by projecting from one location and observing it from another. The image of spots on the object is captured and processed so as to derive a three-dimensional (3D) map of the object [11]. Next the depth map is used for users localisation and determination of their body parts. After that joint localisations are computed and human skeletons are built. Time based differences of human body joint localisation in 3D space represent motion which may be recorded as motion capture data for further processing.

Comparison of Kinect 360 and Kinect One Microsoft datasheets is presented in Table 1.

Available programing libraries

The described mocap solution uses Kinect controllers. To complete the system it was necessary to develop appropriate software, which would communicate with these controllers. Kinect sensor was originally prepared for Xbox game console but many different potential applications and various programing solutions give us the ability to connect it with PC regardless. PC Kinect drivers and frameworks and libraries such as: CL NUI Platform [12], OpenKinect [13], OpenNI [14] and J4K [15] are available for Windows, MacOS X, Linux and ARM systems.

Table 1. Comparison of depth measurement features of Kinect 360 and Kinect One controllers

Feature	Kinect 360	Kinect One
Min - Max Depth Distance	0.8 m (0.4 m in near mode) - 4.5 m	0.5 m – 8 m
Depth Camera Resolution	320 x 240	512 x 424
Framerate	30 Hz	30 Hz
Angular Field of View	57° x 43°	70° x 60°
Skeleton Joints Defined	20	25
Full Skeletons Tracked	2	6

Microsoft has started distributing official Kinect SDK for Windows in June 2011. Since then it is available for programmers for non-commercial use, and for commercial use since February 2012 [16].

There are few Kinect programming frameworks which we may use freely as it was mentioned above. One of the most functional and the best extensible is OpenNI SDK. It was used to develop application for communication with Kinect 360 controller.

The OpenNI framework is an open source SDK used for the development of 3D sensing middleware libraries and applications. In other words it is API for communicating with devices which measure depth and record an image information via underlying drivers. PrimeSense NiTE middleware algorithms for interpreting depth information, e.g. user detection, hand and skeleton tracking or gesture recognitions are now available as a standalone middleware package on top of OpenNI [17]. Previously, NiTE was included into OpenNI via plug-ins. High level diagram of the OpenNI/NiTE 2 architecture is presented in Figure 2 [18].

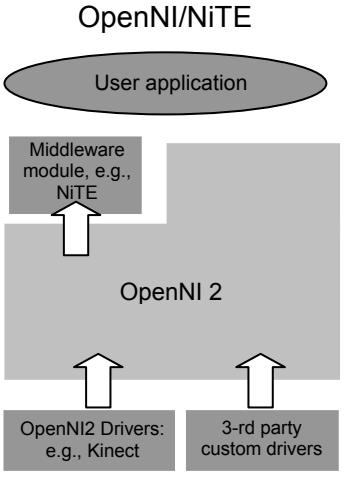

Fig. 2. Architecture of OpenNI/NiTE 2 framework

Irrespective of NiTE any other third party middleware may be simply built to run on top of OpenNI. Links to the most popular of them are available at OpenNI homepage.

To sum up it may be said that it is required to install a few components to develop applications which use the Kinect sensor with OpenNI. They are Microsoft Kinect SDK [19], OpenNI and NiTE middleware. Previous release of OpenNI used a special Kinect driver but the present release is adapted to Microsoft driver from Microsoft Kinect SDK.

Microsoft Kinect SDK is required also to establish communication with Kinect One controller using J4K library. J4K uses Java Native Interface to communicate with Windows library which handles the depth, color, infrared, and skeleton streams. J4K library contains several convenient Java classes that convert these streams into Java objects. Optionally, it is possible to use JogAmp's JOGL Java library to visualize the Kinect data as 3D textured surfaces in openGL.

Multi Kinect solution

The main aim of this research was to evaluate efficiency of angle measurement with multiple Kinect 360 devices. The Kinect 360 controller uses a modified USB port. Connection to PC requires a special cable and additional power supplier which are currently attached to the device bundle. To use multiple capturing devices simultaneously it was necessary to connect them to different USB controllers because of a huge Kinect data bandwidth. If we want to develop applications we must know that not all frameworks support multi-device configuration. Multiple Kinects 360 configuration is supported by J4K, CL NUI Platform or OpenNI 2.x with NiTE 2.x. For instance, the previous releases of OpenNI and NiTE did not support it.

Chosen framework was OpenNI because of its huge functionality and previous experience. J4K was used only for connection with single Kinect One controller. This library will be used in future research with multiple Kinect One controllers.

Motion capture software

OpenNI and NiTE offer C++ standard API, but both of them also have got Java wrappers. It means that it is possible to develop computer vision and motion capture applications in C++ or Java language. Both libraries include runnable examples for both languages with source codes. Mocap solution described in this article was developed in Java programming language.

To prepare Java application for Kinect data processing it is necessary to import two Java libraries included in Redist folders inside OpenNI SDK and NiTE. They are respectively: org.openni.jar and com.primesense.nite.jar. Java wrapper uses JNI technology to invoke native functions, so it is necessary to load native libraries OpenNI and NiTE at the beginning.

The next required step is library initialisation. In this step the needed resources are reserved and the connection to capture devices is prepared. After that it is possible to enumerate the capture devices and open device connections. Sample Java code opening device connections, making UserTracker object for user tracking and opening distinct UserViewer window for each Kinect device is presented below:

```
List<DeviceInfo> devicesInfo =
                        OpenNI.enumerateDevices();
if (devicesInfo.size() == 0) {
   JOptionPane.showMessageDialog(null, "No device
        is connected",
        "Error",JOptionPane.ERROR_MESSAGE);
   return;
}else{
   System.out.println("Number of connected
        devices:"+devicesInfo.size());
}
```

```
int count=0;
Device device;
UserTracker tracker;
for(DeviceInfo devInfo:devicesInfo){
   device = Device.open(devInfo.getUri());
   tracker = UserTracker.create(device);
   new UserViewerApplication(tracker, i).start();
   i++;
}
```

UserViewer class implements NewFrameListener interface and onNewFrame() method. Instance of UserViewer is event listener which reacts when a new frame is received from a Kinect controller. The reaction is reading of the frame and extracting the depth and user data out of it. The next steps are: colouring user body pixels, drawing limbs, computing joint angle and repainting window content. The most important fragments of the code are presented below:

```
public void onNewFrame(UserTracker tracker) {
   …
   mLastFrame = mTracker.readFrame();
   for (UserData user : mLastFrame.getUsers()) {
        if (user.isNew()) {
   mTracker.startSkeletonTracking(user.getId());
        }
   }
   VideoFrameRef depthFrame =
                     mLastFrame.getDepthFrame();
   if (depthFrame != null) {
        ByteBuffer frameData =
               depthFrame.getData().
               order(ByteOrder.LITTLE_ENDIAN);
        ByteBuffer usersFrame =
               mLastFrame.getUserMap().
               getPixels().
               order(ByteOrder.LITTLE_ENDIAN);
        …
        }
        repaint();
   }

//method  invocation  getAngle(user.getSkeleton())

private void getAngle(Skeleton skeleton){
   counter++;
   if(counter%30==0){
               point1 =
        skeleton.getJoint(JointType.LEFT_ELBOW).
                     getPosition();
               point2 =
        skeleton.getJoint(JointType.LEFT_HAND).
                     getPosition();
        angleXY =
        180– Math.round(Math.toDegrees(Math.atan2(
        (Float)point2.getY()-(Float)point1.getY(),
        (Float)point2.getX()-
                     Float)point1.getX())));
        …
   }
}
```

The result of the code is an application which opens as many windows as the number of controller is and each window displays the depth map with coloured users and drawn users' skeletons. Example windows are presented in Figure 3.

A code of an application using J4K library for Kinect One is very similar. A class edu.ufl.digitalworlds.j4k.Skeleton offers a method for joint localisation retrieval:

```
public float[] getJointPositions();
```

As we can see in presented above Java code the angle was computed as a vector angle. The vector was built on the base of two successive joint localisations computed by NiTE library. Because of assumptions that it was necessary to check quality of Kinect device with Kinect SDK software and that the algorithm would use input data computed outside no additional calibration was needed on the beginning. To achieve better results for specific person appropriate mapping of measured and real angle may be done. The problem is that the configuration made by calibration step is very weak, because of the fact that joint localisation computed by SDK and NiTE are unstable and changes quite frequently.

Fig. 3. Depth maps generated by two Kinect 360 controllers with coloured user and drawn skeletons

Conducted research

Our previous research concerning invention of a cheap motion capture system was conducted with use of only one Kinect device [7]. To determine motion capture quality of the new system it was decided to verify the accuracy of joint flexion measurement. Elbow node of a human skeleton was selected to test it. The flection angles of elbow joint measured by the controller were compared to real angles determined by the lines drawn on a testing board. The board had lines marked with angles 0°, 15°, 30°, 45°, 60 °, and 90°. The image of the testing board which was captured through Kinect RGB camera is presented in Figure 4.

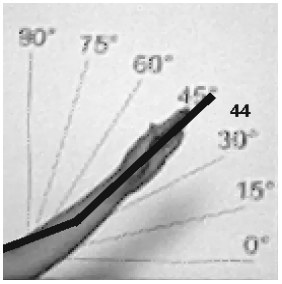

Fig. 4. Testing board, hand and hand limb generated by NiTE middleware

The measurements were performed 30 times for each marked angle in three separate planes determined by axis X and Y, Y and Z and X and Z in left-hand 3D coordinate system. The view axis of the Kinect camera was in line with the Z axis. Average values of measured angles in each plane are presented in Table 2.

Table 2. Joint flexion angle measurement in XY, YZ and XZ planes for Kinect 360 (K.360) and Kinect One (K.One)

XY	**Board**	**0°**	**15°**	**30°**	**45°**	**60°**	**75°**	**90°**
	K.360	2°	16°	29°	43°	57°	73°	86°
	K.One	3°	13°	32°	48°	64°	77°	81°
YZ	**Board**	**0°**	**15°**	**30°**	**45°**	**60°**	**75°**	**90°**
	K.360	-1°	3°	10°	19°	32°	53°	94°
	K.One	16°	23°	25°	36°	44°	49°	65°
XZ	**Board**	**0°**	**15°**	**30°**	**45°**	**60°**	**75°**	**90°**
	K.360	9°	45°	62°	70°	78°	89°	93°
	K.One	5°	16°	29°	40°	50°	66°	78°

The results obtained show that the measured values are close to real values only in a plane perpendicular to the direction of the camera. The applied software has a problem with the correct calculation of joint positions (shown in Figure 4). The second problem is inaccurate determination of bone motion in direction parallel to camera's view axis. These problems appeared during measurements with both Kinect generations. As we can see Kinect One, the successor of Kinect 360 with more advanced depth measurement features, returned data with error level close to error level of Kinect 360.

The next step in the study was angle measurement using two Kinect 360 controllers simultaneously. A diagram of the test system is shown in Figure 5.

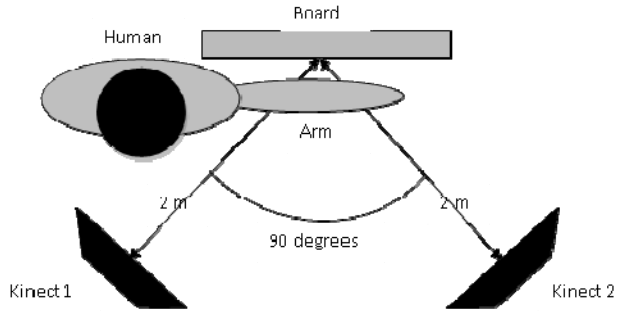

Fig. 5. Diagram of a test system with two Kinect 360 controllers

Distances of both Kinects 360 from the board were determined with a measurement tape. The 90° angle was adjusted with use of two Xsens MTx inertial motion trackers [3]. The maximum error of MTx tracker is equal to 2° [5].

The angle of joint flection was calculated as an average value of measurements made by each device. Tests in YZ plane were omitted due to the fact that in this configuration one of the cameras was still covered up by testing board. Results of the measurements are shown in Table 3.

Table 3. Efficiency of joint flexion angle measurement by two Kinect devices

X Y	**Board**	**0°**	**15°**	**30°**	**45°**	**60°**	**75°**	**90°**
	Kinect 1	0°	16°	28°	44°	55°	68°	84°
	Kinect 2	2°	16°	31°	44°	59°	75°	89°
	Average	1°	16°	29,5°	44°	57,5°	71,5°	87°
X Z	**Board**	**0°**	**15°**	**30°**	**45°**	**60°**	**75°**	**90°**
	Kinect 1	-3°	13°	33°	51°	61°	83°	98°
	Kinect 2	6°	19°	32°	45°	58°	70°	87°
	Average	1,5°	16°	32,5°	48°	59,5°	76,5°	92,5°

The achieved results show that the use of two Kinect controllers at the same time improved significantly the measurement. Results compared to the system with one controller even that it is Kinect One were better. In addition, the system of two Kinects allows partial elimination of errors associated with the correct determination of the position of the joints in the 3D space. Floating positions of joints resulted in large errors in the measurement in case of motion capturing by a single controllers.

Summary

Many motion capture systems are very expensive. This article shows that a cheap motion capture system may be built with the use of Kinect game controllers and available programming frameworks. Such system is able to build depth map of the observed scene, detect users and compute their skeletons' localizations or gesture recognition. The achieved skeletal data may be later used as a source for creating a realistic animation of a human like body or for different motion analysis. The captured data should be saved in a file. We may use one of motion file formats, e.g. C3D or BVH but this element of the system has not been implemented yet. The interface for saving files and sending raw data to application for motion visualisation application developed before and described in article [19] will be the next step of our research.

The accuracy of such a mocap solution is not too high. Maximum errors were over 30°. It is because of the low processing power of such simple devices in comparison to commercial mocap solutions which use more complicated hardware and computation algorithms. Motion capture quality may be leveraged by the application of multiple capturing devices. The achieved results show that the use of two Kinect devices results in better joint localisation and better joint angle measurements and that the future research should utilise multiple Kinect One devices.

The quality of recorded motion may be additionally improved by skeleton smoothing. This may be achieved with the usage of different interpolation algorithms [20].

REFERENCES

[1] Kitagawa M., Windsor B., MoCap for Artists. Workflow and Techniques for Motion Capture. Elsevier, 2008
[2] Menache A., Understanding Motion Capture for Computer Animation and Video Games, Morgan Kaufmann, 2000
[3] Kopniak P., Budowa, zasada, działania i zastosowania systemu rejestracji ruchu firmy Xsens, *Logistyka*, 2014, nr 3, 3049-3058
[4] Kopniak P., Java wrapper for Xsens motion capture system SDK, *Human System Interactions (HSI), 2014 7th International Conference*, Lisbon (Costa da Caparica), 06/2014
[5] Kopniak P., Pomiary kątów ugięcia kończyny w stawie z wykorzystaniem inercyjnego systemu Motion Capture, *Pomiary Automatyka Kontrola*, 60 (2014), nr 8, 590-593
[6] Kopniak P., The use of multiple cameras for motion capture, *Przegląd Elektrotechniczny*, 90 (2014), nr 4, 173-176
[7] Kopniak P., Rejestracja ruchu za pomocą urządzenia Microsoft Kinect, *Pomiary Automatyka Kontrola*, 58 (2012), nr 11, 1016-1018
[8] Kinect for Windows homepage: http://www.microsoft.com/en-us/kinectforwindows/
[9] Schematic diagram of Kinect controller: http://www.wired.com/magazine/wp-content/images/19-07/mf_kinect2_f.jpg
[10] PrimeSense Supplies 3-D-Sensing Technology to "Project Natal" for Xbox 360, http://www.microsoft.com/en-us/news/press/2010/mar10/03-31PrimeSensePR.aspx
[11] PrimeSense Ltd. patents at Patentdocs: http://www.faqs.org/patents/assignee/prime-sense-ltd/
[12] CL NUI Platform: http://codelaboratories.com/kb/nui
[13] OpenKinect project: http://openkinect.org/wiki/Main_Page
[14] OpenNI framework: http://www.openni.org/
[15] Barmpoutis A., Tensor Body: Real-time Reconstruction of the Human Body and Avatar Synthesis from RGB-D, *IEEE Transactions on Cybernetics, Special issue on Computer Vision for RGB-D Sensors: Kinect and Its Applications*, October 2013, Vol. 43(5), 1347-1356
[16] Kinect for Windows homepage: http://www.microsoft.com/en-us/kinectforwindows/
[17] NiTE middleware library: http://www.primesense.com/en/nite
[18] OpenNI/NiTE 2 Migration Guide – Transitioning from OpenNI/NiTE 1.5 to OpenNI/NiTE 2, Document Version 1.1, April 2013: http://www.openni.org/openni-migration-guide/
[19] Kinect for Windows SDK: http://www.microsoft.com/en-us/kinectforwindows/develop/developer-downloads.aspx
[20] Smołka J., Skublewska-Paszkowska M., Analysis of selected interpolation methods with artificial 3D data, *Actual Problems of Computer Science*, 3 (2013), No. 1, 3-18

***Author**: dr inż. Piotr Kopniak, Politechnika Lubelska, Wydział Elektrotechniki i Informatyki, Instytut Informatyki, ul. Nadbystrzycka 36B, 20-618 Lublin, E-mail: p.kopniak@pollub.pl*