Piotr KOPNIAK

Institute of Computer Science, Lublin University of Technology, Lublin, Poland

The Use of Multiple Cameras for Motion Capture

Streszczenie. *Artykuł dotyczy procesu budowy taniego systemu rejestracji ruchu. Rozwiązanie to wykorzystuje kamery PlayStation 3 Eye. Artykuł ten pokazuje, w jaki sposób wykonać synchronizacje wielu kamer i jak stworzyć oprogramowanie do rejestracji i przetwarzania danych wideo w czasie rzeczywistym. Niniejszy artykuł prezentuje także algorytm i rezultaty wyszukiwania oraz śledzenia ruchu jednokolorowych obiektów na podstawie obrazów z dwóch zsynchronizowanych kamer. (**Wykorzystanie wielu kamer do rejestracji ruchu**).*

Abstract. *This article concerns the creation process of a cheap optical motion capture system. The solution uses PlayStation 3 Eye cameras The paper shows how to synchronise multiple cameras and how to develop software for capturing and processing real-time video data. The article presents an algorithm and the results of findings and tracking of mono-colour objects based on images from two synchronised cameras, too.*

Słowa kluczowe: rejestracja ruchu, widzenie komputerowe, przetwarzanie obrazu, synchronizacja kamer.
Keywords: motion capture, computer vision, image processing, camera synchronization.

doi:10.12915/pe.2014.04.40

Introduction

Motion capture techniques allow to record the motion of a human body for further processing or electronic devices steering thanks to gesture recognition. The techniques are now used in medicine, movies, different types of entertainment and computer games production [1-4]. Implementation of this new technology involves a number of problems. One of them is the cost of equipment. Professional optical motion capture systems like Vicon products are costly because they use heavy duty cameras. The cameras may take up to 200 frames per second and each frame has 5 Mpixel resolution [5-7]. Many solutions in real life do not require such precision and may be build cheaper. Economical solutions may be constructed on the base of common devices like web cameras, PlayStation 3 (PS3) Eye cameras [8] or Microsoft Kinect sensors [9].

Proposed motion capture solution is based on a set of 6 or 8 PS3 Eye cameras located around 3D scene which observes the scene from different angles. Various points of view allow us to compute 3D object location from their 2D images and compute object motion in 3D space on the base of sequences of 2D camera frames. Stereovision methods [10, 11] may be used for this step of image data processing.

The construction of the system is possible because of a special multi-camera software driver. PS3 camera uses USB port to communicate with a game console or a PC. Windows system standard drivers can find only one PS3 camera despite a few cameras connected in parallel. To use multiple cameras simultaneously we must use a special driver. We can download such driver for two cameras freely from Code Laboratories. If we want to capture video with more cameras we must buy commercial version of the driver [12].

Processing video streams from multiple cameras causes a very important problem of time synchronisation. When we develop a motion capture system we must have pictures taken by all cameras at the same time as the accuracy of computing the spatial position of objects in 3D space depends on this synchronisation. Preparing the synchronization is a part of our research and it is described in the next section of the paper.

After properly made synchronization, the images taken at the same time by two or more calibrated cameras and located in fixed positions allows us to compute a 3D object location. In our case the objects will be colour LED markers mounted on a human body. A 3D location of markers will indicate the location of a human body parts and allows to calculate the angles of joints. Determined marker locations function of time on a base of sequences of taken frames represent captured motion of the human body.

Synchronisation of cameras

Based on the analysis of PlayStation Eye camera construction [13, 14] the hardware synchronization of two Eye cameras was prepared during our research.

PlayStation Eye camera uses OV7720 series CMOS VGA sensor made by OmniVision Technologies. A top view of pinout and a fragment of a block diagram which describes the input and output of a video timing generator of the chip is presented in Figure 1.

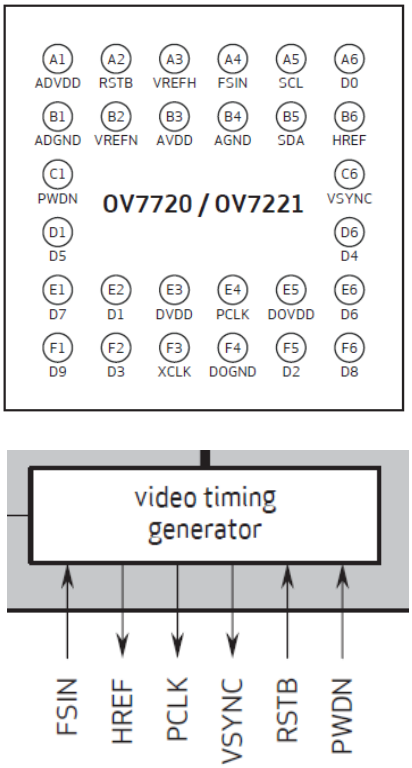

Fig.1. Pinout and fragment of OV7720 CMOS VGA chip block diagram

In synchronization process we used two pins: FSIN and VSYNC of the sensor. The FSIN pin is the frame synchronization input to the CMOS and the VSYNC pin is the vertical sync output. VSYNC signal is the signal which releases the camera shutter and causes frame recording. FSIN signal is a clock input to produce appropriate VSYNC signal.

We may synchronize two cameras connecting VSYNC pin of a master camera and FSIN of a slave camera. The signal passed into FSIN input enforces generation VSYNC of the slave camera synchronized with VSYNC of the master camera. Figure 2 shows the positions of VSYNC and FSIN connectors on the camera board.

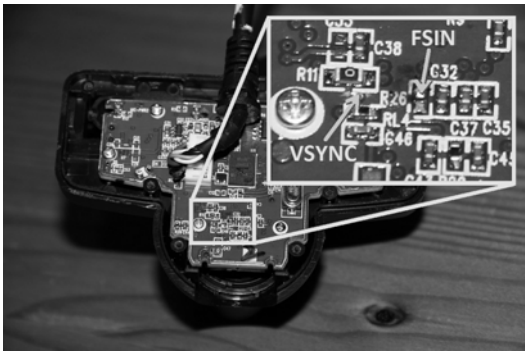

Fig.2. Location of VSYNC and FSIN connectors on PS3 camera board

VSYNC signal is rectangular signal with 3.3 V peaks lasting for 280μs. A frequency of the signal is 30 Hz in VGA mode and 60 Hz in QVGA mode. It means that if we want to synchronize multiple cameras, we should make an electronic module which reproduces and amplifies a VSYNC signal and next connect it into FSIN pin of each slave camera.

Verification of cameras' synchronisation

The next step was a verification of made camera synchronization. We used a two-way oscilloscope to measure VSYNC signals of two unsynchronized and synchronized cameras. Figure 3 shows VSYNC signals before and after synchronization. As we can see, in Figure 3 unsynchronized signals have peeks in different points of time and synchronized signals have peeks at the same moments. It means that synchronization was made appropriately and images are taken by different cameras in the same time.

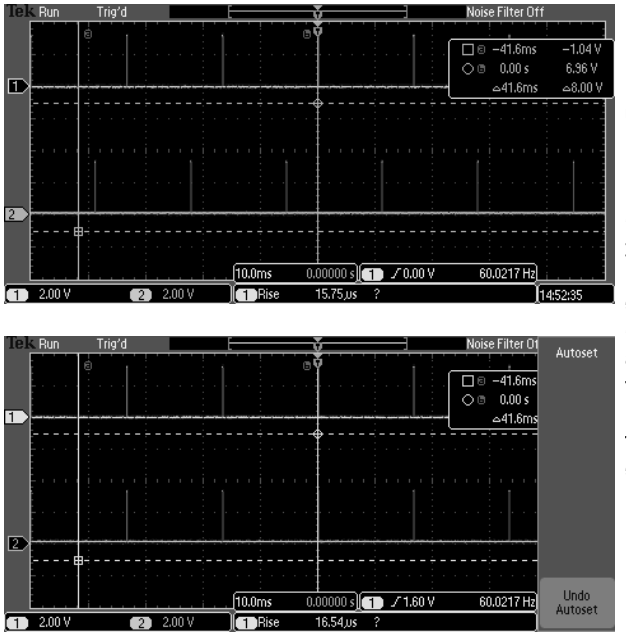

Fig.3. VSYNC signals of unsynchronized and synchronized PS3 Eye cameras

Multi-camera video capturing

As it was mentioned above if we want to design application which uses multiple cameras we may use CL-EyeMulticam driver. Code Laboratories provide CL Platform Software Developer Kit, too. The SDK gives developers a full access and control over the camera usage and manipulation. The APIs provided are exposed as standard C based functions for portability and ease of use, also the full source code of all sample applications is included that acts as a baseline testing framework.

It has been decided to make a motion capture application in Java language. It is possible because the Platform's SDK consists of wrappers for numerous programming languages and many application examples for different languages, too. One of them is Java/Processing wrapper. The wrapper provides classes and functions for Processing script language based on Java. The interface can be used to develop a pure Java application, too.

To develop a multi-camera Java application we have to use functionality packaged into `CL-EyeMulticam.dll` native library. The C library may be used through Java Native Interface (JNA) mechanism. We must prepare own Java class which will load the library with use of the standard Java `System.load(dll_path)` method. The class can use a native code through special Java methods which must be developed. The methods have to be native, static, empty and their names and argument lists must be the same as the names of methods implemented in the dll library. For instance, the method returning the number of connected PS3Eye cameras should be defined as:

```
native static int CLEyeGetCameraCount();
```

For capturing process we must create camera instance using the method:

```
int CLEyeCreateCamera(int cameraIndex,
          int mode, int resolution,
          int framerate);
```

next, we must start capturing:

```
boolean CLEyeCameraStart(
          int cameraInstance);
```

and capture the camera frame:

```
boolean CLEyeCameraGetFrame(
          int cameraInstance,
          int[] imgData,
          int waitTimeout);
```

Captured image data in RGBA color space will be saved into `imgData[]`array. Next the data may be processed or displayed. In our case the next processing step was finding single-colour objects and computing their centers-of-gravity (COG) with the use of OpenCV library.

Detecting objects and calculating their COG's with the use of OpenCV

OpenCV is a real-time computer vision library with very rich functionality. It has got over 500 high-level and low-level functions [15]. OpenCV is an open source and has API interfaces for many programming languages and platforms, including Android. Since 2.4.4 version it has had a full functional Java wrapper. However it have been used a different Java wrapper – JavaCV.

Our tracking algorithm is based on finding mono-colour image areas which represent the tracked objects and

determination of their center points location. To find the location of the mono-color object in a grabbed frame we use color thresholding and center-of-gravity computation. PS3 Eye camera produces images in RGBA color space. We do not need the alpha channel for color detection. OpenCV `cvCvtColor(srcImage, dstImage, CV_RGBA2RGB)` function converts the captured four-channel image into a three-channel RGB space. The color thresholding in RGB color space is not a good solution, either, because of the natural irregularity of object brightness which causes huge differences in RGB channel data and huge thresholding errors. The thresholding should be done in HSV color space where the color's hue is represented by one channel only. An appropriate color space conversion may be done with `cvCvtColor()` function whose the third parameter equals `CV_RGB2HSV`. To eliminate the camera noise and to achieve better result the source image is additionally filtered by `cvSmooth()` OpenCV's function with a median filter.

The found object is represented as a group of image pixels with a regular or irregular shape. We may use moments to obtain mathematical expression for the center of the shape [8]. In physics the moment M expresses how force F operates at distance d along a rigid bar from a fixed fulcrum. The point where the force operates may be replaced by the point with a mass. In case of a computer vision, pixels replace points and the mass component can be replaced by a pixel function, e.g. pixel intensity.

Let us assume that each pixel p_i has intensity I_i and (x_i, y_i) coordinate in two dimensional image space. The sum of the pixels' moments around *y-axis* is as follows:

$$M_y = I_1 x_1 + I_2 x_2 + ... + I_n x_n \tag{1}$$

The sum of pixels' moments around *x-axis* may be written as:

$$M_x = I_1 y_1 + I_2 y_2 + ... + I_n y_n \tag{2}$$

and summary of whole system intensity as:

$$I_{sys} = I_1 + I_2 + ... + I_n . \tag{3}$$

The distances of shape from the x and *y-axis* are described by following equations:

$$\bar{x} = \frac{M_y}{I_{sys}}, \quad \bar{y} = \frac{M_x}{I_{sys}} . \tag{4}$$

The center-of-gravity point, $(\bar{x}, \bar{y})$ is the shape's center.

With OpenCV we may calculate different types of moments using a parameterized equation:

$$m(p,q) = \sum_{i=1}^{n} I_i(x,y) x^p y^q \tag{5}$$

where function arguments p and q are the powers for x and y, and $I(x,y)$ is a pixel intensity function. Moments may be expressed as function $m()$ with different parameters. The moments' equations and summary of system intensity may be written as:

$$M_x = m(0,1), \quad M_y = m(1,0), \quad I_{sys} = m(0,0) . \tag{6}$$

It means that center-of-gravity $(\bar{x}, \bar{y})$ can be expressed as:

$$\bar{x} = \frac{m(1,0)}{m(0,0)}, \quad \bar{y} = \frac{m(0,1)}{m(0,0)} . \tag{7}$$

All moments are calculated at once thanks to one `cvMoments()` function of OpenCV library. It produces `CvMoments` object with all needed moments' data. A piece of Java code for moment and COG (`posX`, `posY`) calculation is presented below:

```
CvMoments moments = new CvMoments();
cvMoments(detectThrs, moments, 1);
double mom10 =
    cvGetSpatialMoment(moments, 1, 0);
double mom01 =
    cvGetSpatialMoment(moments, 0, 1);
double area =
    cvGetCentralMoment(moments, 0, 0);
posX = (int) (mom10 / area);
posY = (int) (mom01 / area);
```

Results of our COG position determination and tracking paths are presented in Figure 4. As we can see there are two images captured by two PS3 Eye cameras. The images have drawn white paths which represent motion of the same red LED marker recorded by two synchronized cameras.

Fig.4. Output video frames of two synchronized Eye cameras with red LED marker tracing paths

Discussion

OpenCV The calculated centers-of-gravity of an object, recorded by multiple cameras allow us to determine the object location in 3D space with the use of stereovision methods. The review, comparison, and implementation of the methods in our processing application will be the next step of our research.

Documentation of Eye camera CMOS describes different work modes of the sensor. As it was mentioned above in default it can capture 30 fps in VGA mode and 60 fps in QVGA mode. This frequency value is enough to save a standard movie but is too low when we want to capture fast human moves. The camera sensor may be configured to achieve higher frequencies through `FrameGrabber` OpenCV's class instance, e.g. we use VGA mode with 60 fps frequency. It gives higher motion capture accuracy. We did not find the highest frequencies but it will be verified in the future.

The developed software does complex computation. We tested our application with two cameras only and the output videos after processing were not fluent enough. It was because the computation was done by CPU only. OpenCV has an interface to use nVidia CUDA technology for supporting complex computation with the support of a graphic card. It requires CUDA graphic drivers and CUDA toolkit installed. We plan to use this technology to accelerate our processing.

Conclusion

There are different motion capture systems available on the market. The systems for professional use are very expensive. It is possible to build an easier and much

cheaper solution. We may use PS3 Eye cameras for this purpose. This article describes how to synchronize and use multiple cameras in parallel and how to develop appropriate processing software. We proved that it is possible to make hardware synchronization of PS3 Eye cameras and that it works correctly. This article proposes methods for detection of color markers and their motion tracking, as well. The methods are still developed, optimized and will be supplemented with stereovision methods to achieve the full functionality of a less expensive motion capture system in the future.

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., Motion Capture Data Visualization Developed with the Utilization of jMonkeyEngine, *Computer Graphics. Selected Issues*, University of Szczecin, (2010)

[4] Skublewska-Paszkowska M., Łukasik E., Smołka J., Analysis on motion interpolation methods, *Actual Problems of Economics, National Academy of Management, Kyiv,* 137 (2012), No 11, 448-455

[5] Vicon Motion Capture System: http://www.vicon.com/, (2013)

[6] Liberty Motion Capture System: http://www.polhemus.com/, (2013)

[7] Shape Wrap II Motion Capture System: http://www.measurand.com/, (2013)

[8] Killer Game Programming in Java Homepage: http://fivedots.coe.psu.ac.th/~ad/jg/index.html, (2013)

[9] Kopniak P., Rejestracja ruchu za pomocą urządzenia Microsoft Kinect*, Pomiary Automatyka Kontrola*, 58 (2012), 1477-1479

[10] Kęsik J., Evaluation of the Stereo-Counterpart Search Area for the Stereovision Algorithms Using the Surveillance Type Cameras Set, *Polish Journal of Environmental Studies*, 18 (2009), No. 3b, 154-159

[11] Kęsik J., Projekt systemu skanowania obiektów 3D, z wykorzystaniem platformy obrotowej i układu stereowizyjnego, *Pomiary Automatyka Kontrola*, 57 (2011), 1483-1485

[12] Code Laboratories Products: http://codelaboratories.com/products/, (2013)

[13] OV7720/OV7221 CMOS VGA (640x480) Camera Chip Sensor with OmniPixel2 Technology, http://www.zhopper.narod.ru/mobile/ov7720_ov7221_full.pdf, (2013)

[14] OV7720 VGA product brief, http://www.ovt.com/download_document.php?type=sensor&sensorid=79, (2013)

[15] Bradski G., Kaehler A., Learning OpenCV, Computer Vision with the OpenCV Library, O'Reilly, (2008)

Author:
dr inż. Piotr Kopniak, Lublin University of Technology, Computer Science Institute, ul. Nadbystrzycka 36 B, 20-618 Lublin, E-mail: p.kopniak@pollub.pl.