VNU Journal of Science: Comp. Science & Com. Eng., Vol. 32, No. 2 (2016) 23–30

Kinect Based Character Navigation in VR Game☆

Thi Chau Ma*, Minh Duong Hoang

VNU University of Engineering and Technology,
E3 Building, 144 Xuan Thuy Street, Cau Giay, Hanoi, Vietnam

Abstract

VR game is a hot field because it provides practical experience for the player. However, in the game, players are usually required attaching special equipments. Those equipments are often very expensive. Moreover, many devices make players less flexible because they are attached to players. VR game using Kinect is a good choice because Kinect is affordable and not attached directly to the player. In this paper, we propose a technique of Kinect based character navigation in VR Game. As environment is supported by Kinect, we implement character navigation by player's hand actions. Besides, we use markers to increase the effectiveness of hand action recognition. In our experiments, character's rotations reaches 70% in accuracy. Straight movement and turning reach 98% in accuracy.

Received 30 November 2015, revised 09 January 2016, accepted 14 January 2016

Keywords: Kinect application, hand recognition, VR games.

1. Introduction

In the last two years, in Vietnam, the game attracted a lot of investments and researches. According to [1], Vietnam is the largest market in Southeast Asia Game. However, restrictions still exist, obviously, the solutions are still awaited. An important factor that make the game to be attractive is the ability of character movements. To control the movement of the characters, players have to touch and navigate specialized devices to control the movements of the characters. These devices were typically gamepad, keyboardist, Wiimote. Besides, with technology advances, in virtual reality games, players no need to touch game devices. In these games, body actions or speech were used to control the character movement through the special devices, such as Kinect, Virtuix Omni.

We are interested in games using Kinect because the type of these games gives players the flexibility and comfort while playing without exposure Kinect. Moreover, Kinect relatively low cost comparing with other devices such as, head mounted devices or data gloves. Kinect-supported games can be classified into two main categories. In the first type, entire bodies of players are identified to contribute and play games like audition games [2], or collision ball games [3]. In the second type, hand gestures are recognized to control games [4]. In our research, we want to make use of hand actions obtained by Kinect to navigate character in order to give players the freedom and comfort in playing games. However, the accuracy of the hand action recognition and the number of hand actions based on Kinect were limited because of identical when recognizing different bone joints. In this paper, we propose a solution of Kinect based character navigation in VR Games. We recognize hand action to navigate characters in

☆ This work is dedicated to the 20th Anniversary of the IT Faculty of VNU-UET.

* Corresponding author. Email: chaumt@vnu.edu.vn

VR Games. Marker is glued to the hands of players to obtain more number of hand actions and increase accuracy in hand action recognition. The use of glued makers does not reduce the degree of flexibility of players.

2. Related Works

With the available and affordable Kinect, Kinect and its applications appear a lot in the life. First of all, we mention the applications in the medical. Kinect provide a Software Development Kit (SDK) which gives access to skeletal tracking data and can be used directly in rehabilitation game developments. So, there are lots of physical therapy and rehabilitation applications [5][6][7] using Kinect. These applications are classified into 3 groups. Fall detection and Fall risk reduction, Stroke Rehabilitation and exercise games. In [8], authors presented a low-cost system to surgical training for computer assisted surgeries by utilizing dual sensors and marker based tracking components. Markers were patterns in the form of checkboard. By using markers, they implemented calibration of dual infrared sensors and point correspondence establishment in a stereo configuration. Then they accomplished 3D localization by multiple back projection lines. The system was effectively the combination of inexpensive electronic devices and traditional tracking algorithm.

Secondly, in the VR environment, a hot and exciting field, Kinect attracted much attention. Kinect is one of easy ways of 3D reconstruction. In the environment, data recognized by Kinect is divided in two types: Human skeleton and hand recognitions. RGBDemo [9] is an open source software providing a simple toolkit to manipulate Kinect data. In this software, 3D model of an object would be rebuilt when the object are taken with markers. 3D builder app [10] and Kinect sensor are used to create a 3D reconstruction for a 3D printer. Ultra Seven [11] is a game used Kinect in AR environment. In the game, the player is turned into a character and shot his power. The character was modeled game player by body recognition with Kinect. Easy to see the above Kinect applications are based on recognition of human skeleton.

Another type of Kinect applications is based on hand actions [12][13][14]. In [12], authors accurately reconstructed complex hand poses across a variety of subjects using Kinect. The system allowed continually recovering from tracking failures. Matthew Tang [13] showed a novel method for recognizing hand gestures using Kinect sensor. In the approach, they recognize 'grasp' and 'drop' gestures with over 90% accuracy. In her thesis [14], Yi li used K-means clustering and Graham Scan algorithm to recognize nine popular gestures and counting numbers. Kinect data range was between 0.5m to 0.8m. Hands were distinguished from the background by depth information.

In Kinect applications in particular and recognitions in general, Marker is an effective tool. Marker is a very simple sticker, usually in the form of checker board pattern. Marker is popular in modeling by its simple manipulation and efficiency. Checker board pattern has been using in camera calibration [15][16][17] to reconstruct 3D model from 2D images. Checker board pattern is also use in Kinect calibration [18] to align depth camera and RGB camera. Marker help robot to detect path to navigate.

In this paper, we want use hand actions from Kinect to control the movement of character in VR game. We utilize marker to detect hand actions easily and accurately.

3. Proposed method

In this paper, we propose a Kinect-based navigation technique in VR game. In the game, we use a Kinect sensors to obtain player's hand actions. The players navigates characters using actions only on one hand. The character's actions include straight movement, turning (left/right) and rotation. We chose these action in implementation because hand gestures are meaningful to people. So, learning and using hand gestures are easy. Moreover, in the game, most free parts of body but hand makes player more comfortable. In technique term,

the problem is more focus. Meanings of these actions are described clearly as follow. Straight movement is character goes forward/backward. Turning is character turns left/ right. Rotation is character rotates his body left/right or up/down to observe the scene. This action changes eye view to make decision of other actions. The player's thumb finger is worn marker for determining the exact location, and recognizing of complex hand gestures as well. In addition to character navigation, the player can also use the hand to perform other actions in the game such as, shooting, and selection.

Firstly, we define some parameters later used in the process of navigation. Root is player's neck joint (Fig. 1) on player's. $Root_{depth}$ is the distance from Kinect sensor to root. $Hand_{depth}$ is the distance from Kinect sensor to the wrist joint of the hand. $Hand_X$ is the wrist joint in horizontal direction in acquired image. $Root_X$ is root in horizontal direction. **StraightVector** is a vector showing straight direction of the character. **OrthVector** is a vector orthogonal to **straightVector** on the left. **StraightSpeed** and **OrthSpeed** are speed vectors in movements.

We defined three types of navigation: straight movement, turning (left/right) and rotation corresponding to the following actions:

- Straight movement: move the wrist towards or away from the root. The change of the distance from the wrist to the root decides the direction (forward/backward) and speed of movement.
- Turning: change the wrist to left or right while distance between the wrist and root does not change.
- Rotation: rotate the wrist. The changed angle decides the speed of rotation.

These actions are determined by comparing the position of wrist with root and rotation of marker glued on a thumb finger. In detail, navigation technique includes three main steps (Fig. 2): parameter setup, hand marker tracking, and navigation estimation. In parameter setup, players are asked to stand or sit in front of

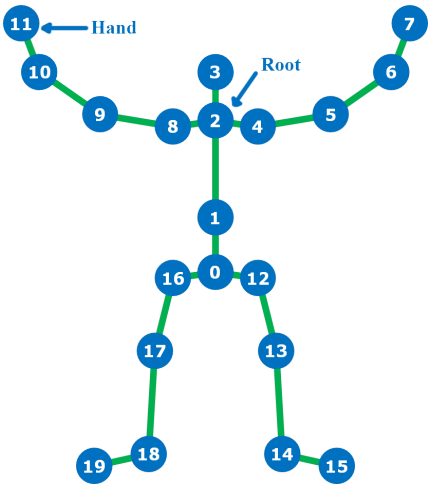

Fig. 1: Kinect skeleton.

the Kinect sensor for a while to obtain some measurements such as, root position and hand depth. These measurements are utilized to calculate character's speed and acceleration. In hand marker tracking, we use emguCV [19] and POSIT algorithm [20] to determine marker and its' rotation. Then, above defined actions are recognized. Finally, we perform navigation estimation by recognized actions.

4. Implementation

4.1. Parameter setup

At the beginning process, player is required to stand in front of Kinect to collect initial data about root and wrist joint position. Player simply need to be in the pose such that Kinect can see his root and wrist joint position. Player wears a marker in front of fist. In order to track rotation easily, we should use marker in the form of checker board. We define some stand distances by the initial depth data as follows:

- $InitDist_{Straight} = Hand_{Depth} - Root_{Depth}$
- $InitDist_{Orth} = Hand_X - Root_X$
- $HandRange_{Straight} = Max(Hand_{Depth}) - Min(Hand_{Depth})$
- $HandRange_{Orth} = Max(Hand_X) - Min(Hand_X)$

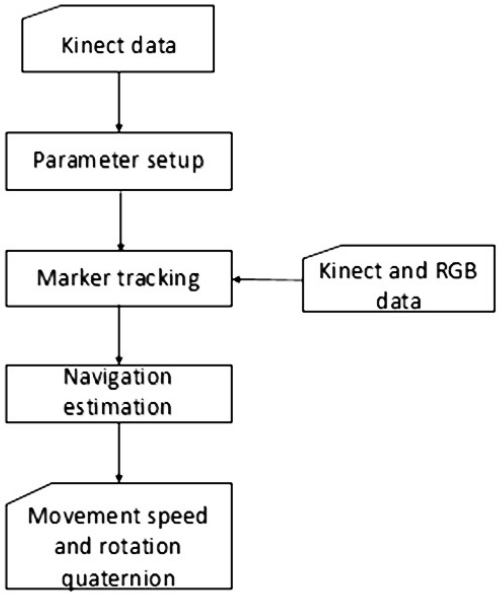

Fig. 2: Overview method process.

4.2. Hand and marker tracking

Hand tracking is processed by using Kinect SDK. To detect marker, we process RGB image from Kinect to get contours fitting for 4 vertices. After that, camera is calibrated in stardard form, and data of marker is extracted in the form of a 2D matrix.

4.2.1. Skeleton tracking

The skeleton tracking feature of Kinect is used to get hand position. This process has two stage: first computes a depth map (using structed light), then infers body position (using machine learning). At first stage, the depth images are all computed by the PrimeSense hardware built into Kinect. Pixels in a depth image indicated calibrated depth in the scene, rather than a measure of intensity of color. Depth images offer several benefits, such as working in low light levels, giving a calibrated scale estimate, being color and texture invariant and resolving silhouette ambiguities in pose.

The depth image received from previous stage is transformed to body part image by using randomized decision forest. This tree is learned from 100,000 depth images with known skeleton and theirs derivations. After classifying image into body parts, this information is pooled to generate the position of 3D skeletal joints. The mean shift algorithm is used to compute modes of probability distributions. By using this algorithm, the densest region of body part is extracted. For more detail information, please read [21].

4.2.2. Marker tracking

The marker detection has three main steps. First, the contour detection algorithm[22] is used to detect all quadrangle. To normalize the marker image, the input image must be unwarp by using perspective transformation. The estimated transformation will transform the marker the square form to extract data from it. In the implementation, we use simple marker 5x5 bit. For each normalized quadrangle, the image is divided into cells. The data of each cell is decided by the number of white pixels; if the number of white pixels is greater than the number of black pixels then the cell's value is 1, otherwise, the value is 0. After comparing the candidate marker's value with the real marker's value, the marker is recognized.

4.3. Navigation estimation

4.3.1. Forward, backward

When playing, player do hand actions. While distance between wrist joint changes, the character moves straight away. If the distance increases, character moves forward. Otherwise, character moves backward. The estimataion of straight speed is shown in Algorithm 1.

At each frame, the depth distance between Hand and Root $depthDistance$ is calculated and the difference between current pose and initial pose $rawVertical$ is obtained. If $rawVertical$ is greater than minimum distance $minDeltaStraight$, $rawVertical$ is converted into range $[0, 1]$ and assigned to $scaledVertical$. Otherwise, $scaledVertical$ is 0. After that, the speed vector of forward movement is calculated.

4.3.2. Left, right

We use horizontal distance between root and wrist joint to control character turning (Algorithm 2).

Algorithm 1: Estimate forward/backward

$depthDistance \leftarrow Root_{Depth}$ - $Hand_{Depth}$
$rawVertical \leftarrow (depthDistance - InitDist_{Straight})$
if $rawVertical > minDeltaStraight$ **then**
    $scaledVertical \leftarrow rawVertical$ / $HandRange_{Straight}$
else
    $scaledVertical \leftarrow 0$
StraightSpeed $\leftarrow$ **StraightVector** $\times$ $playerMaxSpeed \times scaledVertical$

Algorithm 2: Estimate Left/right movement

$orthDistance \leftarrow Hand_X$ - $Root_X$
$rawHorizontal \leftarrow orthDistance - InitDist_{Orth}$
if $rawHorizontal > minDeltaOrth$ **then**
    $scaledHorizontal \leftarrow rawHorizontal$ / $HandRange_{Orth}$
else
    $scaledHorizontal \leftarrow 0$
OrthoSpeed $\leftarrow$ **OrthoVector** $\times$ $playerMaxSpeed \times scaledHorizontal$

The horizontal distance between Hand and Root *orthDistance* is obtained and the difference between current pose and initial pose *rawHorizontal* is calculated by subtract current *orthDistance* to $InitDist_{Orth}$. If *rawHorizontal* is greater than minimum distance *minDeltaOrth*, *rawHorizontal* is converted into range $[0, 1]$ and assigned to *scaledHorizontal*. Otherwise, *scaledHorizontal* is 0. After that, the speed vector of left movement is calculated.

4.3.3. Rotation

Player rotates the wrist to control character rotation. The marker is rotated follow the hand. By using Coplanar POSIT algorithm [20], the estimated angle can be inferred into 3 axis angles (ie. yaw, pitch, roll). The estimated roll angle is used to rotate left, right, and the pitch angle decide to rotate player up or down. After that, each frame add amount of angle to user's quaternion to change the rotation angle of player (Algorithm 3).

Firstly, the left/right rotation is measured. By tracking marker, the marker rotation state *markerState* is estimated. The estimated rotation and translation matrix *HomoMat* is calculated by coplanar POSIT algorithm. After extract *yaw*, *pitch*, *roll* angle, the *markerState* is checked. If marker is in *FRONT* state and *roll* angle is greater than minimum angle to rotate $minAngle_X$, $axis_X$ is obtained by normalize *roll* angle into range $[0, 1]$. If marker is in *LEFT*, *RIGHT* state, the $axis_X$ is respectively 1 and -1. The up/down rotation is measured by using *pitch* angle instead of *yaw* angle because of the instability of *yaw* angle estimation. If *pitch* angle is too small then player does not rotate. Because the value of *pitch* is the same when marker rotate up and down, the value of *roll* and *yaw* is checked to infer whether player want to rotate up or down. If *roll* and *yaw* less than 90, player wants to rotate up; and otherwise, player rotates down. After normalized $axis_X$ and $axis_Y$, the rotation speed of each frame Δ_{angleX} and Δ_{angleY} are calculated.

5. Evaluation

To prove the effectiveness of the proposed method, we experimented on several routes (Fig. 3). These routes are in the shapes of lines,

Algorithm 3: Estimate rotation

$axis_X \leftarrow 0, axis_Y \leftarrow 0$
$markerState \leftarrow Extract(markerData)$ % detect marker on thumb figure %
$HomoMat \leftarrow CoPOSIT(source, des)$ % extract homography matrix %
$(yaw, pitch, roll) \leftarrow Extract(HomoMat)$ % determine yaw, pitch or roll %
if $markerState == FRONT$ **then**
    if $Abs(roll) > minAngle_X$ **then**
        $axis_X \leftarrow roll/maxAngle_X$
    else
        $axis_X \leftarrow 0$
else
    if $markerState == LEFT$ **then**
        $axis_X \leftarrow 1$
    else
        $axis_X \leftarrow -1$
$pitch \leftarrow pitch + 90$
if $pitch < minAngle_Y$ **then**
    $axis_Y \leftarrow 0$
else
    if $roll < 90$ *and* $yaw < 90$ **then**
        $axis_Y \leftarrow Abs(pitch)/maxAngle_Y$
    else
        $axis_Y \leftarrow -Abs(pitch)/maxAngle_Y$
$\Delta_{angleX} \leftarrow axis_X \times dampling \times TimeFrame$
$\Delta_{angleY} \leftarrow axis_Y \times dampling \times TimeFrame$

triangulations, rectangles, hexagons. Each route estimated minimum number of movement actions (ie. forward, backward, left, right) and rotation (ie. left/right, up/down) actions to complete each test (Fig. 4). We compare the time to complete test of this method to keyboard/mouse method (Table 1). Though, our proposal's runtime has not reached those of using keyboard/mouse. In small games including mainly character's movement, our runtime is acceptable. The proposed method works well with simple movement action, such as forward, backward, left, right; however, the accuracy of movement is decrease a bit with compound movement actions like forward-left,

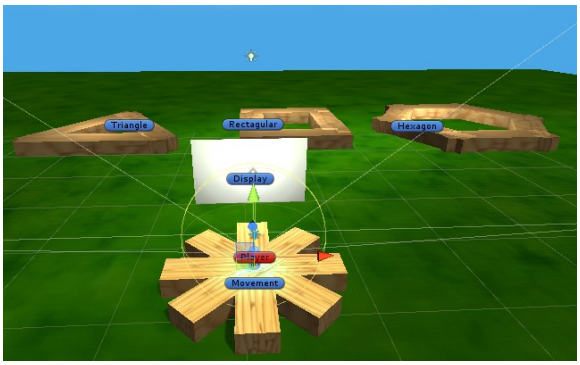

Fig. 3: A test map.

Fig. 4: A test.

backward-left, etc. According to table 2, the accuaracy of up/down rotation is lowest because of shadow and errors of estimation algorithm.

Each test is defined by a sequence of movements and rotations. An error is defined as follow: "when user sends control a signal, the responding result is not corrected or the responding time is over a time threshold, then an error is occurred". In order to calculate the accuracy, we experimented each test several (around 20) times. The experimenters do the sequence of actions as definition of the test in order to reach the final destination. For each sending signal, if the responding movement/rotation is incorrect or the responding time is too long (the time threshold is 1 second), then it is counted as an error. The accuracy is calculated by the percentage of the number of corrected actions overall the number of all actions, where number of corrected actions equals to the subtraction between the number of actions and the number of errors.

We applied Kinect based character navigation in Maze Game and Haunted House Game. In

Maze, we navigate character finding his way to get destination. In Haunted House Game, we navigate character escaping from a ghost's chase.

Table 1: Comparison of detection time reductions

Route name	Movement / Rotation	Time (keyboard/ our method)
Movement Test	12 / 0	$55.3s/38.2s$
Rotate Left/Rightt	0 / 2	$6.2s/4.6s$
Rotate Up/Down	0 / 2	$20.1s/4.7s$
Triangle	2 / 2	$44.8s/21.1s$
Rectangular	4 / 4	$25.6s/16.5s$
Hexagons	8 / 8	$34.0s/20.1s$

Table 2: Accuracy of character navigation

Route name	Movement	Rotation	Error
Movement Test	12	0	3%
Rotate Left/Rightt	0	2	2%
Rotate Up/Down	0	2	30%
Triangle	2	2	4%
Rectangular	4	4	2%
Hexagons	8	8	3%

6. Conclusions

In the paper, we proposed a technique to navigate characters by hand actions in VR game. A Kinect was used to obtain the hand actions. In this technique, we also suggested using the marker attached to the player's hand to increase the accuracy of hand action recognition. So, we not only take advantage of low-cost device but also enhance flexibility for players. accuracy of straight movement is up to 98%. Rotate left / right test also gives high accuracy, but rotate up/down have low accuracy. We deploy our solution in some realtime games such as finding short route in Mazes, finding way to escape the haunted house.

In the near future, we study deeply recognition techniques to enhance the accuracy of recognizing rotation, especially the accuracy of rotate up/down. We also think about how to improve running time to apply our method in many kinds of VR games. Beside that, more gestures can be defined to control not only the navigation but also game actions (ex. jumping, shooting, interacting, etc.) by using hand finger. Moreover, by using Kinect, the action in game can be defined by body action; this makes the games more interactive and interesting.

Acknowledgments

This work has been supported by VNU University of Engineering and Technology, under Project No. CN.15.02

References

[1] http://www.vnu-itp.edu.vn/en/news/market/178-viet-nam-hien-nay-la-thi-truong-game-lon-nhat-khu-vuc-dong-nam-a.html.
[2] D. C. (series), Mtv games, Microsoft Game Studios, 2011.
[3] Kinect adventures, . Microsoft Game Studios, 2010.
[4] http://www.metacritic.com/game/xbox-360/angry-birds-trilogy.
[5] A. M. L. D. C. V. L. M. Khademi, H. Mousavi Hondori, S. C. Cramer, Comparing direct and indirect interaction in stroke rehabilitation, .in Proceedings of the Extended Abstracts of the 32nd Annual ACM Conference on Human Factors in Computing Systems 2014 1639–1644.
[6] L. D. S. C. M. Khademi, H. M. Hondori, C. V. Lopes, Comparing 'pick and place' task in spatial augmented reality versus non-immersive virtual reality for rehabilitation setting, .inAnnual International Conference of the IEEE Engineering in Medicine and Biology Society 2013 4613–4616.
[7] H. M. Hondori, M. Khademi., A review on technical and clinical impact of microsoft kinect on physical therapy and rehabilitation, Journal of Medical Engineering 2014.
[8] W. L. Hongliang Ren, A. Lim, Marker-based surgical instrument tracking using dual kinect sensors, IEEE transactions on automation science and engineering. 2014.
[9] http://rgbdemo.org/index.php.

[10] http://windows.microsoft.com/en-us/windows-8/apps-windows-store-tutorial.
[11] https://code.google.com/p/kinect-ultra/.
[12] T. Sharp, et. al., Accurate, robust, and flexible real-time hand tracking, Best of CHI Honorable Mention Award 2012.
[13] M. Tang, Recognizing hand gestures with microsoft's kinect, CS228 2011.
[14] Y. Li, Hand gesture recogniton using kinect, Master thesis 2012.
[15] Z. Zhang, A flexible new technique for camera calibration, IEEE Transactions on Pattern Analysis and Machine Intelligence 22 (2000) 1330–1334.
[16] P. F. Sturm, S. J. Maybank, On plane-based camera calibration: A general algorithm, singularities, applications, International Conference on Computer Vision 1999.
[17] J. Heikkila, O. Silven, A four-step camera calibration procedure with implicit image correction, CVPR, IEEE Computer Society 1997 1106–1112.
[18] U. N. Carolina Raposo, Joao Pedro Barreto, Fast and accurate calibration of a kinect sensor, 3DV-Conference (2013) 342 – 349.
[19] http://www.emgu.com/.
[20] D. F. D. Denis Oberkampf, L. S. Davis., Iterative pose estimation using coplanar feature points, Comput. Vis. Image Underst. 1996 495–511.
[21] J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, A. Blake, Real-time human pose recognition in parts from single depth images, in: Proceedings of the 2011 IEEE Conference on Computer Vision and Pattern Recognition, CVPR '11, IEEE Computer Society, Washington, DC, USA, 2011, pp. 1297–1304.
[22] S. Suzuki, K. Abe, Topological structural analysis of digitized binary images by border following 30 (1) (1985) 32–46.