Multimed Tools Appl (2015) 74:211–226
DOI 10.1007/s11042-013-1757-5

Automatic registration of a virtual experience space with Kinect

Jaemin Soh · ByungOk Han · Yeongjae Choi · Youngmin Park · Yong-Ho Seo · Hyun S. Yang

Published online: 12 November 2013

Abstract The popularization of virtual experience spaces involves several difficulties, and this paper focuses on part of them—additional equipment removal and cumbersome initialization process lightening. In this paper, a new automated initialization method to replace the traditional initialization process is proposed using the Kinect sensor, an affordable three-dimensional video capture device. Although the accuracy of this method is lower than that of other methods owing to the limitations of the Kinect sensor, it shows comparable performance under certain assumptions and offers users the opportunity to experience virtual space in a simple, low-cost manner.

Keywords Automatic registration · Camera calibration · Virtual spaces · Kinect

1 Introduction

Virtual experience space provides a user with experiences and services in a virtual environment. For example, a virtual reality arcade or a three-dimensional (3D) military training simulator offers unique and interesting experiences by simulating various situations that might be impossible in the physical world. However, complex, high-cost equipment is required for such simulations.

The CAVE [1] suggests a new method that uses multiple image projectors instead of special equipment. The projection of virtual space images onto multiple screens around a user enables the user to easily experience a virtual environment. However, the method limits any direct interaction between the user and the virtual space. A virtual experience space for an e-learning system, which consists of one screen and one camera, was suggested in [6]. Even though this system has some limitations, the devices are easily available; therefore, the system can be adopted easily in the school and home environments.

One of the difficulties in popularizing a virtual experience space is the initialization process. A virtual experience space inherently needs a calibration step because it uses a coordinate

J. Soh · B. Han · Y. Choi · Y. Park · H. S. Yang
Department of Computer Science, KAIST, Daejeon, Republic of Korea

H. S. Yang
e-mail: hsyang@kaist.ac.kr

Y.-H. Seo (✉)
Department of Intelligent Robot Engineering, Mokwon University, Daejeon, Republic of Korea
e-mail: yhseo@mokwon.ac.kr

system that is different from the physical environment. Lee [6] solves this problem by using prevalent camera calibration methods such as those defined by Zhang [9] and Tsai [8]; however, special markers such as a checkerboard must be positioned in the physical environment to match the reference points in the virtual space. This requirement might be cumbersome for a user who is not familiar with these technologies.

The Kinect sensor, which is used to capture 3D space information, is a popular product owing to its affordability. In this paper, a new automatic registration method using 3D space information captured by Kinect to replace the traditional initialization process of virtual systems is suggested. Some research studies focused on calibration methods using Kinect [3, 10]. However, they still require additional equipment and markers such as a checkerboard. In addition, the automatic 3D space information retrieval methods such as the ones described in [4] and [7] are not computationally effective. By contrast, our method does not require the use of markers, and it performs the registration in real-time.

We begin in Section 2 with an introduction about virtual experience space and its registration method. We present our approach in Section 3. The results of experiments are presented in Section 4. A discussion of future work is presented in Section 5.

2 Virtual experience space and registration

2.1 Virtual experience space

This paper aims to develop a technology for spreading a virtual experience space with a simple setup as shown in Fig. 1. This system consists of three parts; the display (e.g., a monitor or a combination of a projector and a screen), the audio (e.g., speakers), and image sensors (e.g., a Kinect). These components are positioned around the action zone where users play. The display and the audio are required for the user experience, and the image sensor acquires data related to the user's behavior and position.

This system provides users with the combination of sounds and images that are part of the virtual space. It also simulates the users' silhouettes based on the data acquired from Kinect, and it provides functionality for manipulating objects in the virtual space. Thus, the users can feel like they are really inside the virtual space and can be more involved in the prepared situation.

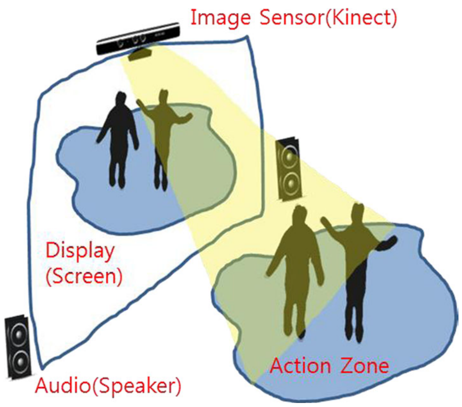

Fig. 1 Virtual experience space setup

2.2 Registration

Registration is the process of overlaying the real space with the virtual space in order to create a correspondence between points in real space and points in virtual space. This can be resolved by making the coordinate systems of the real camera and the virtual camera consistent with each other. The intrinsic and extrinsic parameters of the cameras can be calculated and then used to augment real objects into the virtual space and vice versa.

Using a common camera model, the coordinates of a point in the real world can be converted into camera-centered 3-dimensional coordinates using the extrinsic parameter and the converted point can be translated into a point with 2-dimensional coordinates using the intrinsic parameter of the camera.

2.3 Coordinate

In order for users to have a new experience through interaction with virtual objects, the virtual space needs to be authored in advance to encourage the users to perform tasks, such as moving virtual objects. When authoring the virtual space, a standard coordinate system is necessary to deploy objects in the virtual space, and this coordinate system needs to reflect the real-world coordinate system where the authors or users stand.

In a camera-centered coordinate system, if a camera is positioned parallel to the floor, actions performed in the real world can be simulated in the virtual space simply by translating the movement. However, if the camera is not parallel to the floor, authors will have difficulty in translating the movement. In fact, if the camera does not capture the entire body of the user, the camera is tilted diagonally by moving it up a little. The more tilted the camera, the greater the difference between the camera-centered coordinate system and the coordinate system of the user.

Thus, a coordinate system that is orthogonal to the floor in the real world is ideal for both authoring and registration.

2.4 Camera assumption

This paper assumes that the sensor is located at the front center of the action zone and is tilted downward to capture the entire body of a user, as in Fig. 2b. The action zone is the area where a user stands and moves his whole body to interact with virtual environment. The world coordinates and camera coordinates are displayed in Fig. 2b, c, d.

The origin of the real-world coordinates is assumed to correspond to the point in the 2D image that is the center of the x-axis and the mass center of the valid distance pixels along the y-axis (Fig. 2a). Section 3 will discuss the process of determining the 3D position of this point in the real world. The parameter could be calculated using the normal direction of the basis plane and the origin position of the coordinates according to the applications.

The other assumption is that the color data image and depth data image are pre-aligned using other methods, such as NuiImageGetColorPixelCoordinateFrame-FromDepthPixelFrame() from the MS Kinect SDK (http://www.microsoft.com/en-us/kinectforwindows/).

3 Automatic registration

3.1 Intrinsic parameter

In traditional methods, the acquisition of the positions of pairs that correspond between the real world and the image is required to calculate a camera's intrinsic parameter. To obtain the

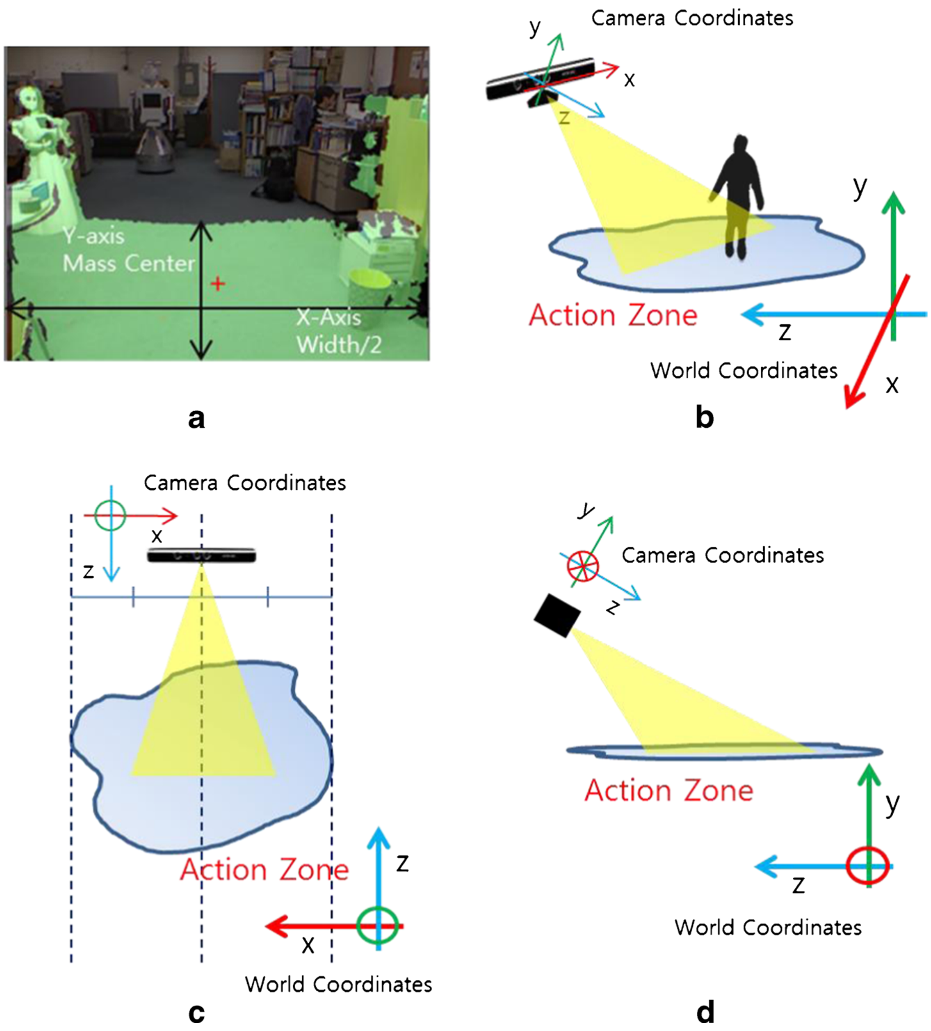

Fig. 2 **a** Example of coordinate origin (*red cross*), expected action range (*green*) **b** Camera and Action Zone coordinates **c** Camera and Action Zone (x-z plane) **d** Camera and Action Zone (y-z plane)

positions in the camera image, a prepared marker or a pattern panel is used to automatically or manually determine the corresponding locations in the real world. Each camera has its own parameters, even cameras of same model, although the difference is not significant. For automation, we use a simple camera model and parameters from the manufacturer.

Ignoring the radial distortion, the intrinsic parameter model is as follows:

$$\begin{bmatrix} F_x & S & C_x \\ 0 & F_y & C_y \\ 0 & 0 & 1 \end{bmatrix} \tag{1}$$

Where the skew parameter S is assumed as 0 and (C_x, C_y) is the center of the image.
The manufacturer of Kinect provides the camera parameters as shown in Table 1.

Table 1 Intrinsic related parameters from Kinect SDK

NUI_CAMERA_COLOR_NOMINAL_HORIZONTAL_FOV	62.0f
NUI_CAMERA_COLOR_NOMINAL_VERTICAL_FOV	48.6f

These values are based on a 640×480 image resolution, and they represent the FOV in the horizontal and vertical directions.

Horizontal FOV ($\mathrm{FOV_h}$) and Vertical FOV ($\mathrm{FOV_v}$) are defined as follows:

$$\mathrm{FOV_h} = 2\mathrm{atan}\left(\frac{Width}{2F_x}\right) \tag{2}$$

$$\mathrm{FOV_v} = 2\mathrm{atan}\left(\frac{Height}{2F_y}\right) \tag{3}$$

From (2) to (3),

$$\mathrm{F_x} = \frac{Width}{2\tan\left(\frac{FOV_x}{2}\right)} \tag{4}$$

$$\mathrm{F_y} = \frac{Height}{2\tan\left(\frac{FOV_y}{2}\right)} \tag{5}$$

Using the values in Fig. 3 and the Eqs. (4) and (5), $\mathrm{F_x}$=532.569434 and $\mathrm{F_y}$=531.541079 are calculated.

Therefore, the Kinect intrinsic parameter is as follows:

$$\begin{bmatrix} 532.569434 & 000.000000 & 320.000000 \\ 000.000000 & 531.541079 & 240.000000 \\ 000.000000 & 000.000000 & 001.000000 \end{bmatrix} \tag{6}$$

3.2 Extrinsic parameter

The extrinsic parameter represents the relation between the camera and the real world. This consists of 3DOF translation and 3DOF rotation which can be obtained by calculating the position of the world coordinate's origin and the angle of the camera's rotation from the floor plane.

3.2.1 Rotation calculation

Using Kinect, the distance from the camera and the camera-centered coordinates of each pixel in the 2D image can be achieved. The rotation part of the extrinsic parameter is obtained by calculating the angle between the up-vector of the camera and the normal vector of the floor with 1° of freedom along the normal direction. Compensation of this degree of freedom can be handled by multiplying the inverse angle along the normal direction with the assumption that the camera looks towards the center of the action zone

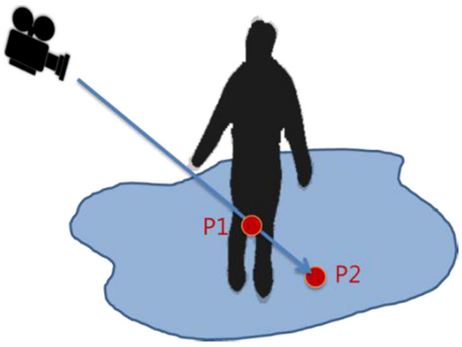

Fig. 3 Origin of the floor plane

(mentioned in Section 2.4). The up-vector means the up direction of the camera and holds the proper camera direction resolving the rolling effect. In this paper, up-vector is used due to the resemblance to the normal direction of the floor. With the coordinates in Fig. 2b-d, this vector has the value of (0, 1, 0).

With the camera angle of looking downward, the entire body of the user is captured by the camera and the floor plane is assumed as the dominant plane. Therefore, the normal direction of the floor plane is the dominant direction of the pixels of the image. The Algorithm 1, 2 and 3 calculate the rotation part of the extrinsic parameter:

Algorithm 1 Rotation part calculation

```
pn ← Calculate_PlainNormal()                    ▷ Plain normal vector from Algorithm 2

up ← ( 0, 1, 0 )                                ▷ up direction vector, Y-up case

dot ← pn · up                                   ▷ Calculate Rotation between 2 vectors

angle ← arc_cos( dot )
axis ← pn × up

mat ← Convert_AxisAngle_To_Matrix( axis, angle )

euclidean_val ← Convert_Matrix_To_Euclidean( mat )
mat_compen ← Convert_AxisAngle_To_Matrix( up, -euclidean_val.y )
                          ▷ For Compensation of 1DOF Y-axis(up direction) rotation

return mat × mat_compen                         ▷ Rotation Matrix 3x3
```

Algorithm 2 Plain normal calculation, Calculate_PlainNormal()

```
image_step: image sampling step                                                    (7)
outlier_threshold: threshold value for determining normal direction outlier        (8)
```

```
count ← 0
sum ← (0, 0, 0)

j ← image_step
while j < image_height do
  i ← image_step
  while i < image_width do
    c ← depth( i, j )                                    ▷ depth value of pixel (i, j)
    if c is valid then                         ▷ check the distance is too close or too far
      n ← Calculate_Normal( i, j )
                        ▷ normal vector of (i, j) in camera centered coordinate, Algorithm 3
      push back n to normal_list
      sum ← sum + n
      count ← count + 1
    endif
    i ← i + image_step
  endwhile
  j ← j + image_step
endwhile

temp_normal ← sum / count
normalize( temp_normal )

sum ← (0, 0, 0)
count ← 0
y_mass_center ← 0
for each e in normal_list do
                        ▷ remove the outliers and check whether this point is in the plane
  if temp_normal · e > outlier_threshold then
    sum ← sum + e
    count ← count + 1
    y_mass_center ← y_mass_center + y value of e in 2D image
  endif
endfor
y_mass_center ← y_mass_center / count
                        ▷ y direction mass center in 2D image for selecting origin    (9)

return sum / count                                                ▷ Plain normal vector
```

Algorithm 3 Normal of (i, j) calculation, Calculate_Normal(i, j)

```
NeighborStep: step value for selecting 8-neighbor(←1) or 4-neighbor(←2)          (10)
NDist: distance of neighborhood                                                    (11)
----------------------------------------------------------------------------------------
Bias_x[8] = { 1, 1, 0, -1, -1, -1, 0, 1 }
Bias_y[8] = { 0, 1, 1, 1, 0, -1, -1, -1 }

sum ← (0, 0, 0)
count ← 0

for k ← 0 to 7 do with increase step as NeighborStep
    p0 ← Get_Camera_Centered_Position( i + Bias_x[k] * NDist, j + Bias_y[k] * NDist )   (12)
    p1 ← Get_Camera_Centered_Position( i + Bias_x[(k + 6) % 8] * NDist,
                                          j + Bias_y[(k+6) % 8] * NDist )
    if p0, p1 are valid then
        pc ← Get_Camera_Centered_Position( i, j )
        sum ← sum + ( p0 – pc ) × ( p1 – pc )                                            (13)
        count ← count + 1
    endif
endfor

return sum / count
```

Get_Camera_Centered_Position(i, j) is from the Kinect SDK; Convert_AxisAngle_To_Matrix and Convert_Matrix_To_Euclidean are well-known methods that convert between rotation representations. These methods are discussed in detail in (http://www.euclideanspace.com/maths/geometry/rotations/conversions/index.htm).

The normal directions of pixels are calculated by Algorithm 2 and Algorithm 3. The goal is to get the overall direction of the whole image; therefore, only a representative set of the pixels need to be concerned. For uniform sampling, this algorithm skips pixels by a predefined step which will be tested in Section 4.

The normal direction of one pixel is defined as the average of the cross-products of the vectors of each vertically positioned pair from near pixels (see (12) and (13)). This result is less accurate; but faster than other elaborate methods, and it is sufficient because this algorithm aims to obtain only one representative value.

3.2.2 Translation calculation

The point of origin, which is needed for the translation part, is based on the assumptions described in Section 2.4. The x-axis value is fixed and the y-axis value is calculated in Algorithm 2 at (9).

This origin point from the 2D image is denoted as $\mathbf{p}_{2D}$ and its 3D position from the Kinect sensor as $\mathbf{p}_{3D}$. If a user or other objects are located at this position, then $\mathbf{p}_{3D}$ is not the origin of the floor plane (P1 in Fig. 3). Therefore, the intersection of the floor plane and the line between the camera center and $\mathbf{p}_{3D}$ has to be calculated (P2 in Fig. 3).

A plane is represented by a 4D-vector $\boldsymbol{\pi}$(a, b, c, d). The normal direction, which is obtained in the previous algorithm, represents (a, b, c). Because 3 values are fixed, the plane that includes $\mathbf{p}$(x,y,z) is defined as (a,b,c,−ax−by−cz). If $\mathbf{q}(x',y',z')$ is on this plane, then $ax'+by'+cz'+d=0$. For all pixels that have the same direction as the representative normal direction, each "d" needs to be calculated and tested how many other points are on this plane. The plane with the most pixels represents the floor plane.

In homogeneous expression, the line between the camera center $\mathbf{C}(0,0,0,1)$ and $(\mathbf{p}_{3D} \mid 1)$ is represented by $\mathbf{L}=\mathbf{C}\cdot\mathbf{p}_{3D}^{T}+\mathbf{p}_{3D}\cdot\mathbf{C}^{T}$, and $\mathbf{L}\boldsymbol{\pi}$ is the point where $\mathbf{L}$ intersects with the plane $\boldsymbol{\pi}$ [2]. Using the above values, the purposed point is $(\mathbf{p}_{3D}.x\times d, \mathbf{p}_{3D}.y\times d, \mathbf{p}_{3D}.z\times d, -\pi\cdot\mathbf{p}_{3D})$. Therefore, the origin $\mathbf{O}$ is defined as:

$$O\left(\frac{\mathbf{p}_{3D}.x \times d}{-\pi\cdot\mathbf{p}_{3D}}, \frac{\mathbf{p}_{3D}.y \times d}{-\pi\cdot\mathbf{p}_{3D}}, \frac{\mathbf{p}_{3D}.z \times d}{-\pi\cdot\mathbf{p}_{3D}}\right) \tag{14}$$

3.2.3 Extrinsic parameter form

Generally, the extrinsic parameter is represented as a 3×4 matrix, which is a combination of a 3×3 rotation matrix and a 3×1 translation matrix. This form uses the 3×3 rotation matrix defined in Section 3.2.1 and the 3×1 translation matrix from the vector (14) in 3.2.2.

4 Experiment and analysis

A point in world coordinates is converted to the camera-centered position using an extrinsic parameter, and the result is converted to a position on the 2D image using an intrinsic parameter. Users can see the synthesized scene with this process, and gesture recognition technologies help the users sense virtual the reality by giving the user a task where he controls virtual objects.

In Section 3, four parameters are presented; 1) the threshold value (7) for outlier removal in Algorithm 2; 2) the step size (8) for sampling image pixels; 3) neighbor model (10) for Algorithm 3 (i.e., 4-Neighbor or 8-Neighbor); and 4) the distance to the near pixel for calculating the normal direction (11). The machine used for the experiments has an Intel® Core™ i5-2500 @ 3.30GHz and 4GB memory, and the experiment was conducted at the place shown in Fig. 4. The overall configuration and the equipment setup are described in Figs. 1 and 2. Figure 4 shows the registration result (green grid) and normal directions of sampling points (red lines). The green circles have are included points in the normal plane calculation and blue circles are outliers.

4.1 Outlier threshold to calculate normal vector

To clearly illustrate the performance changes, the sampling step is set to 5 pixels. The graph in Fig. 5 shows the effect of the outlier threshold value (x-axis) on the rotation part of the extrinsic parameter and on the time and accuracy performance. The blue line indicates the execution time(in ms; time units indicated on the left side of the graph), while the red and the green lines show the rotation differences Axis-Diff and Angle-Diff (both in degrees; units indicated on the right side of the graph). Axis-Diff and Angle-Diff are from the values of axis-angle represent forms of the result rotation matrix, both Tsai result and proposed method.

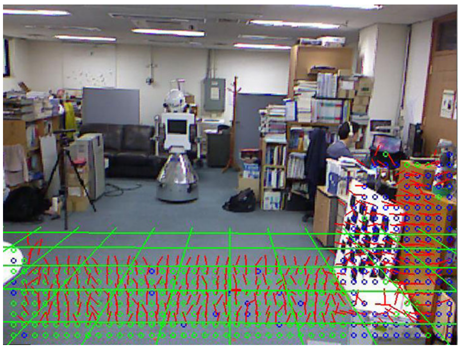

Fig. 4 Experiment location and registration result

Execution time ranges from 57 ms to 30 ms. With smaller outlier values, more noises are produced when the dominant plane is calculated; therefore, execution takes more time and errors increase. Based on this result, the outlier threshold of 0.8 is used in other experiments.

4.2 Step value

The sampling step size that is used to calculate the normal direction of the floor plane also affects the execution time and the comparative difference from the Tsai results, as shown in Fig. 6.

The sampling step size is the most influential parameter. The graph in Fig. 6 shows that the rotation accuracy is not affected by the step; nevertheless, the execution time drops dramatically. Considering the standard deviation of the accuracy, a sampling step size of 10 pixels is acceptable, which gives an average execution time of 6.27 ms.

The specifications of the machine used, the configuration of the installation, and the content and other process needed to run the scenario all affect the performance; therefore, tuning these parameters could produce better results.

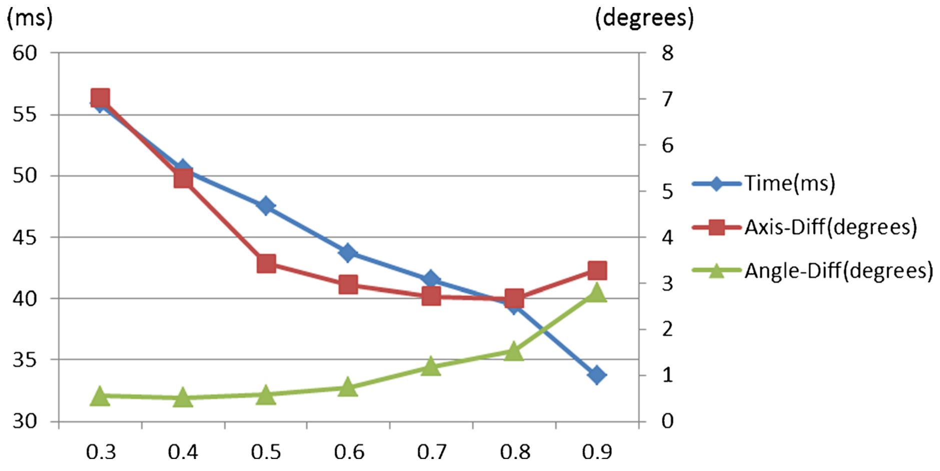

Fig. 5 Performance according to outlier threshold value

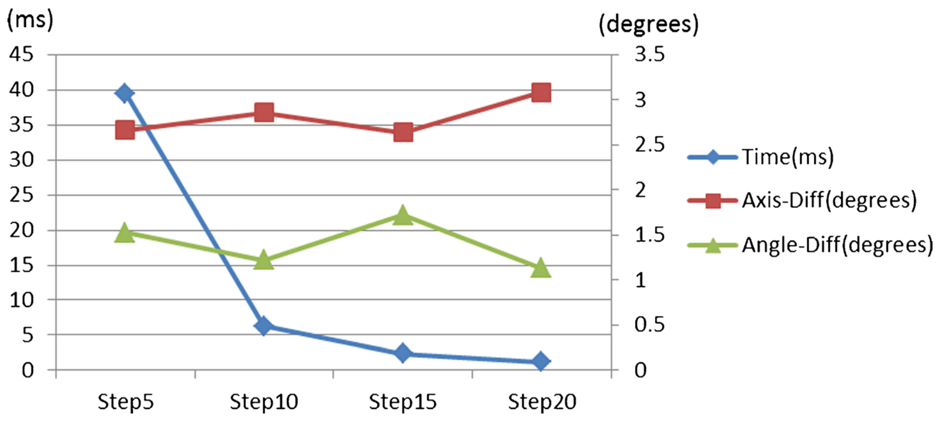

Fig. 6 Performance according to step value

4.3 4-neighbor vs. 8-neighbor

Table 2 represents the changes in time and accuracy (sampling step size is set to 5 pixels) using the 4-neighbor and 8-neighbor models for calculating the rotation in Algorithm 3, as described in 3.2.1. The result shows that this variable doesn't significantly affect the performance.

4.4 Distance of near pixels for normal calculation

Setting the outlier threshold as 0.8, the sampling step size as 5 pixels, and the neighbor model as 8-Neighbor, the performance result based on the near pixel distance is as shown in Table 3. Like the neighbor model, this variable doesn't significantly affect the performance.

4.5 Time

The average time for 100 executions is 2 ms~40 ms, depending on the step value that is used for sampling. If the step is set to 10 pixels, the average time of executions is 6 ms. This result shows that the registration process could be executed for each frame within a reasonable time.

4.6 Comparison of the camera parameter analysis with the Tsai method

As described in Section 3.1, this paper uses a simple camera model, similar to the one used with Tsai [8]. Some points on the floor are set down and their positions in a simulated scene are manually entered. The focal length difference is about 40 mm and rotation difference is about 2°.

Table 2 Performance according to 4-neighbor and 8-neighbor

Neighbor model	4-neighbor	8-neighbor
Time (ms)	36.25	39.47
Axis-difference(degrees)	2.14563	2.66234
Angle-difference(degrees)	2.10386	1.52542

Table 3 Performance according to the near pixel distance for calculating the normal direction

Distance (pixels)	Dist 3	Dist 5	Dist 7
Time (ms)	39.47	40.96	41.62
Axis-difference(degrees)	2.66234	2.73233	2.89421
Angle-difference(degrees)	1.52542	3.13892	3.01121

4.7 Application

Registration makes it possible for a user to interact with the objects inside the virtual world. For showing this, an application is developed (Fig. 7). This application can allow a user to manipulate virtual objects by selection, translation and rotation; therefore, it is suited for educational or entertainment scenarios.

To implement this application, the manipulation gestures are designed. Touching an object for 1 s with one hand means selection and release. Object follows the hand after selection and system enters from the translation mode to the rotation mode with two hands push gesture for 1 s. User can rotate the selected object with two hands in the rotation mode.

4.8 Additional analysis and future work

This paper made some assumptions, which could cause some problems. First, according to the assumption that the floor plane is the most dominant plane, when two or more planes have similar portions of an image or when a wall is more dominant than the floor, the result of the

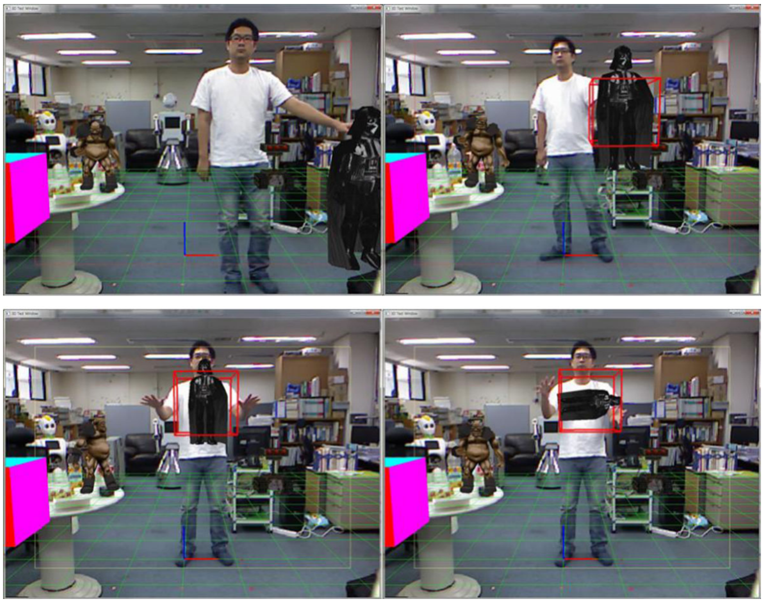

Fig. 7 A manipulation application; object selection, translation and rotation

registration could be wrong. To offset this problem, the camera could be repositioned and adjusted to point at the front center of the action zone, while ensuring that the floor covers more than half of the captured image. In addition, other technologies will be developed using the assumption of the possible camera rotation angle or other information.

The reliability of a 3D point cloud from Kinect needs to be analyzed and applied to improve the accuracy of this registration. The materials of objects could also influence the result of the Kinect sensor [5].

5 Conclusion

This paper aims to create a virtual experience space with less cost and effort in order for a user to experience a new environment and situation. To achieve this goal, the fully automated registration method using a Kinect sensor is developed. Even though it has slightly less accuracy, it doesn't need any other equipment, such as a calibration panel, and it is fully automated and fast. This algorithm takes 6 ms of execution time with a rotation error of approximately 2°.

With this algorithm, scenarios that control virtual objects in a real world background are possible, like the application shown in Section 4.7. Furthermore, some scenarios that use virtual world backgrounds and segmented user images would be improved with advanced technologies, such as a user image boundary enhancement using aligned 3D and 2D information.

Acknowledgments This work was supported by the IT R&D program of MKE & KEIT [10041610, The development of the recognition technology for user identity, behavior and location that has a performance approaching recognition rates of 99 % on 30 people by using perception sensor network in the real environment]. This research was supported by the KUSTAR-KAIST Institute, Korea, under the R&D program supervised by the KAIST.

References

1. Cruz-Neira C, Sandin DJ, DeFanti TA, Kenyon RV, Hart JC (1992) The CAVE: audio visual experience automatic virtual environment. Commun ACM 35(6):64–72. doi:10.1145/129888.129892
2. Hartley R, Zisserman A (2004) Multiple view geometry in computer vision, 2nd edn. Cambridge University Press, March, pp. 65–72
3. Herrera CD, Kannala J, Heikkilä J (2012) Joint depth and color camera calibration with distortion correction. IEEE Trans Pattern Anal Mach Intell 34(10):2058–2064
4. Khoshelham K (2010) Automated localization of a laser scanner in indoor environments using planar objects. IEEE Int Conf Indoor Positioning Indoor Navig, pp. 1–7. doi:10.1109/IPIN.2010.5647576
5. Khoshelham K (2011) Accuracy analysis of kinect depth data. ISPRS Workshop Laser Scanning 38(5):W12
6. Lee S, Ko J, Kang S, Lee J (2010) An immersive elearning system providing virtual experience. IEEE Int Symp Mixed Augmented Real, pp. 249–250
7. Scaramuzza D, Harati A, Siegwart R (2007) Extrinsic self calibration of a camera and a 3D laser range finder from natural scenes. Int Conf Intell Robots Syst, pp. 4164–4169
8. Tsai RY (1987) A versatile camera calibration technique for high-accuracy 3D machine vision metrology using off-the-shelf TV cameras and lenses. IEEE J Robot Autom 3(4):323–344
9. Zhang Z (2000) A flexible new technique for camera calibration. IEEE Trans Pattern Anal Mach Intell 22(11):1330–1334
10. Zhang C, Zhang Z (2011) Calibration between depth and color sensors for commodity depth cameras. IEEE Int Conf Multimed Expo, pp. 1–6

Jaemin Soh received his B.S. degree and M.S. degree in Computer Science from Korea Advanced Institute of Science and Technology (KAIST), South Korea, in 2006 and 2008. He is currently pursuing his Ph.D. at the Artificial Intelligence and Media Laboratory (AIM Lab.) in KAIST. His research interests are in the field of computer vision, intelligent agent, augmented reality and human computer interface.

ByungOk Han received his B.S. degree in the Department of Computer Engineering from Chung-Ang University and M.S. degree in the Robotics Program from Korea Advanced Institute of Science and Technology (KAIST) in 2008 and 2010. He is currently pursuing a Ph.D. at the Artificial Intelligence and Media Laboratory, KAIST. His research interests include computer vision, image understanding, and face recognition.

Yeongjae Choi received his B.S. degrees from the Department of Computer Engineering, Chonbuk National University, in 2007. He is currently pursuing a Ph.D. at the Artificial Intelligence and Media Laboratory, KAIST. His research interests include computer vision, image understanding, and human computer interaction.

Youngmin Park received his B.S. degree in Computer Science and Engineering from Dongguk University, South Korea, in 2011. He is currently pursuing his M.S at the Artificial Intelligence and Media Laboratory (AIM Lab.) in Advanced Institute of Science and Technology (KAIST). His research interests are in the field of pattern recognition and computer vision, image segmentation, and deformable models.

Yong-Ho Seo received his B.S., M.S., and Ph.D. degrees in the Department of Electrical Engineering and Computer Science from KAIST (Korea Advanced Institute of Science and Technology) in 1999, 2001, and 2007 respectively. He was an Intern Researcher at the Robotics Group, Microsoft Research, Redmond, WA, USA in 2007. He was a consultant at Qualcomm CDMA Technologies, San Diego, CA, USA from February 2008 to August 2009. Since March 2010, he has been currently a Professor and a Chairperson of the Department of Intelligent Robot Engineering, Mokwon University. He served Program Committee Members of many international conferences on robotics, computer vision, virtual reality and multimedia. His research interests include humanoid robots, human robot interaction, robot vision and wearable computing.

Hyun S. Yang is a full Professor of Dept. of Computer Science and Director of AIM (AI & Media) Lab at KAIST. He got his BS degree from Dept. of Electronics Engineering. Seoul National University, Korea and both MS and Ph.D. Degrees from School of Electrical and Computer Engineering at Purdue University, West Lafayette, Indiana, USA. He was a Governing Board Member of IAPR (International Association of Pattern Recognition). He is a Fellow of the National Academy of Engineering of Korea (NAEK), Vice President of Int'l VSMM (Virtual Systems and Multimedia) Society and Vice Chair of IFIP TC14. He was General Chairs of VSMM2010 (14th Int'l Conf. on Virtual Systems and Multimedia), VRCAI2010 (ACM SIGGRAPH 9th Int'l Conf. on VR Continuum and Its Applications for Industry) and ICEC2010 (IFIP 9th int'l Conf. on Entertainment Computing). His research activities include Computer Vision, VR/AR, Human Robotics, and Ubiquitous/ Wearable Computing.