The Journal of The Institute of Internet, Broadcasting and Communication (IIBC)
Vol. 22, No. 1, pp.111-120, Feb. 28, 2022. pISSN 2289-0238, eISSN 2289-0246

https://doi.org/10.7236/JIIBC.2022.22.1.111
JIIBC 2022-1-17

AR 플랫폼을 사용하지 않는 실제 방 기반 인테리어 시뮬레이션 연구

A Study on Interior Simulation based on Real-Room without using AR Platforms

최규석[*], 김준건[**], 임창묵[***]

Gyoo-Seok Choi[*], Joon-Geon Kim[**], Chang-Muk Lim[***]

요 약 가구는 실내에서 다른 구조물과 잘 어울리는지 확인하는 것이 구매결정에 있어서 필수적이며 COVID-19 사태로 인한 언택트 마케팅 상황에서는 더욱 중요한 요소가 되고 있다. 이에 따라 가구 배치 인테리어 시뮬레이션을 위해 ARCore, ARKit 등 AR(Augmented Reality) 오픈 소스의 등장으로 AR을 이용한 길이 측정법이 등장하고 있는 추세이다. AR를 이용하는 이러한 기존 방식은 평면 카메라 이미지를 토대로 깊이 맵(Depth Map)을 생성하고 복잡한 입체화 연산이 수반되는 방식이라 스마트폰을 사용하여 정확한 실내 크기 정보가 요구되는 작업에서는 한계가 드러난다. 본 논문에서는 ARCore, ARKit 사용 없이 스마트폰에 내장된 가속도 센서, 자이로스코프 센서만으로 정확한 실내 크기를 측정하는 방법을 제안한다. 또한 제시된 기법을 이용한 활용 예제로 미리 디자인된 방 인테리어를 각방에 맞게 적용하는 방법을 제시하였다.

Abstract It is essential to make a purchase decision to make sure that the furniture matches well with other structures in the room. Moreover, in the Untact Marketing situation caused by the COVID-19 crisis, this is becoming an even more impact factor. Accordingly, methods of measuring length using AR(Augmented Reality) are emerging with the advent of AR open sources such as ARCore and ARKit for furniture arrangement interior simulation. Since this existing method using AR generates a Depth Map based on a flat camera image and it also involves complex three-dimensional calculations, limitations are revealed in work that requires the information of accurate room size using a smartphone. In this paper, we propose a method to accurately measure the size of a room using only the accelerometer and gyroscope sensors built in smartphones without using ARCore or ARKit. In addition, as an example of application using the presented technique, a method for applying a pre-designed room interior to each room is presented.

Key Words : Virtual Reality, Augmented Reality, Interior Design, Trigonometric Function,

[*]종신회원, 청운대학교 컴퓨터공학과
[**]준회원, 청운대학교 컴퓨터공학과(교신저자)
[***]준회원, 청운대학교 컴퓨터공학과
접수일자 2021년 11월 29일, 수정완료 2022년 1월 3일
게재확정일자 2022년 2월 4일

Received: 29 November, 2021 / Revised: 3 January, 2022 / Accepted: 4 February, 2022
[*]Corresponding Author: cmake@kakao.com
Department of Computer Engineering, Chungwoon University, Korea

I. 서 론

2020년 11월, 통계청의 발표에 따르면 온라인 거래액은 전년 동월 대비 17.2% 증가, 그중에서도 모바일쇼핑 거래액은 21.9% 증가로 나타났고, 이는 COVID-19 사태의 장기화로 인하여 온라인 쇼핑의 비중이 계속 커지고 있음을 잘 보여주고 있다[1]. 그중에서도 가구의 거래액은 전년 동월 대비 무려 42.5%인 1375억 원이 증가하였다. 가구는 다른 제품들보다 크기가 중요하고, 또한 실내에서 다른 구조물과 잘 어울리는지 확인하는 것이 구매결정에 있어서 필수적으로 중요한 사항이다. 하지만 가구는 오프라인 매장을 방문해도 구매 후 배치할 방에서 공간을 얼마나 차지하는지, 또 다른 소품들과 얼마나 어울리는지 등은 직접 배치해보지 않고는 알기가 힘들다. 게다가 온라인 쇼핑에서는 오프라인에서처럼 실물을 보는 것이 불가능하고 반품으로 인해 발생하는 비용이 다른 제품보다 더욱 높기 때문에 소비자는 온라인 상의 가구 선택에 있어서 다른 제품 소비보다 신중해질 수밖에 없다.

이러한 문제점에 대해 IKEA는 IKEA Place라는 애플리케이션을 출시하여 소비자가 가구를 미리 배치해보는 체험을 할 수 있게 했다[2]. 하지만 IKEA Place 앱은 IKEA 오프라인 매장에 전시된 것과 같은 주변과 조화로운 디자인 없이 각각의 가구를 배치하는 정도의 체험만을 구현해 놓아서 사용자 맞춤형의 인테리어 디자인을 체험해볼 수 없다. 게다가 Google의 ARCore, Apple의 ARKit등 현존하는 AR 애플리케이션의 대부분은 카메라로 얻은 평면 이미지를 입체화 연산하는 방식이기에, 이 과정에서 복잡한 많은 연산을 필요로 하게 된다[3][4]. 이러한 연산이 저성능 스마트폰에서 원활하게 이루어지기엔 여러 가지 실질적 제약이 따르고, 따라서 사용자 공간을 정확히 인식하는 데 한계가 있을 수 있다. 예컨대, 배치된 가구와 벽지를 포함하는 완성된 인테리어 디자인을 적용해볼 수 없다. 이에 따라 본 논문에서는 입체화 연산 없이 보다 간편한 방법으로 이러한 한계점을 해결할 방안을 제시하고자 한다.

II. 본 문

그림 1의 1, 2, 3번째 그림은 카메라를 이용하여 바닥의 각 꼭짓점을 가운데 조준점에 맞추어 찍는 과정을 캡처한 스크린샷이고, 4번째 그림은 한 천장 꼭짓점을 조준점에 맞추어 찍는 과정을 캡처한 스크린샷이다. 이러한 과정을 통해 얻은 정보를 기반으로 계산하여 아래 그림 2와 같은 방 정보를 얻게 된다. 본 논문에서는 제안된 방법의 구현을 위해 Unity 게임 엔진을 이용하여 가구배치 인테리어 시뮬레이션을 위한 애플리케이션을 제작하였다.

그림 1. 꼭지점을 위해 캡쳐된 스크린샷(예)
Fig. 1. Capturing screenshots for vertices(example)

그림 2. 애플리케이션 스크린샷
Fig. 2. Screenshots of application for interior simulation

그림 2는 그림 1에서 찍었을 때 얻은 센서의 값을 기반으로 인테리어 시뮬레이션을 실행한 결과 화면이다. 여기서 3번째 화면은 가상의 가구와 벽지를 적용한 모습이다. 본 논문에서는 기존 AR 플랫폼(AR Core, AR Kit 등)을 사용함 없이 다음과 같은 과정을 통해서 보다 간편한 방법으로 사용자 맞춤형의 인테리어 시뮬레이션을 구현한다.

사용자의 방을 정확하게 측정하기 위해서는 사용자의 스마트폰 조작을 통해 방 정보를 불러오는 과정이 필요하다. 먼저 사용자로 부터 스마트폰을 들고 있는 높이를 수동으로 입력받는다. 그리고 사용자가 무작위의 위치에서 방바닥을 이루는 정점(Vertex) 네 곳 중 세 곳을 찍고, 천장을 이루는 네 정점 중 한 정점을 마지막으로 찍으면 렌더링에 필요한 모든 방 정보를 얻게 된다. 이 과정에서 각 정점마다 사용자가 얼마나 좌우로 회전하였는지, 그리고 상하로 기기를 얼마나 기울였는지를 측정한

값과 삼각함수를 이용하여 지점의 위치를 계산하게 된다.

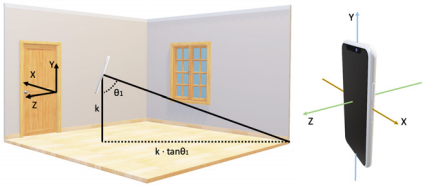


그림 3. 거리 측정을 위해 필요한 각도 (왼쪽)와 가속도 센서의 가속도 좌표 방향(오른쪽)
Fig. 3. The angle required for distance measurement (left) and the direction of the acceleration coordinates of the accelerometer (right)

정점과 사용자의 바닥의 거리를 구하려면 기기를 들고 있는 높이, 그리고 측정하려는 스마트폰 사용자와 정점까지의 거리를 알아야 한다. 이때 바닥에서부터 사용자가 기기를 들고 있는 지점까지의 높이를 k라고 가정했을 때, k는 앞서 언급했듯이 사용자에게서 직접 값을 입력받는다. 그리고 기기의 카메라 방향과 바닥이 이루는 각을 θ_1라고 가정했을 때, θ_1의 값을 얻기 위해서 가속도 센서(Accelerometer)를 이용한다. 가속도 센서는 오른쪽 그림과 같이 각 축의 방향에 작용하는 가속도의 크기를 m/gs^2 단위로 반환한다. (단, g는 중력가속도) 예컨대, 기기가 눕힌 상태로 평평한 바닥 위에 정지하여 온전한 중력의 힘을 받는다면 이 힘은 -Z 방향으로 작용하고, 벡터의 크기는 1이 되며, 반환되는 Z 값은 -1이 된다.

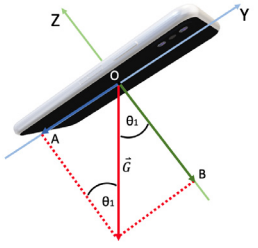


그림 4. 센서의 방향에 따라 중력가속도 $\vec{G}$ 의 성분 분해
Fig. 4. Decomposion of gravitational acceleration $\vec{G}$ depending on the direction of the sensor

거리 측정을 위해 필요한 θ_1을 구하려면 가속도 센서의 Y값과 Z값을 이용하여야 한다. 왼쪽 그림과 같이 θ_1 (단, $0 \leqq \theta_1 < \pi$)을 구하기 위한 -Y, -Z 방향 벡터(Vector)인 $\overrightarrow{OA}$, $\overrightarrow{OB}$의 크기를 구하려면 중력 벡터인 $\vec{G}$를 각각 핸드폰 좌표의 Y, Z 성분 벡터로 나누어서 표현하면 식 (1)과 같다.

$$\vec{G} = \overrightarrow{OA} + \overrightarrow{OB} \quad (1)$$

가속도 센서 값이 m/s^2 단위로 반환된다면 $|\overrightarrow{OA}|$, $|\overrightarrow{OB}|$는 각각 $g \cdot \sin(\theta_1)$, $g \cdot \cos(\theta_1)$이 되겠지만 m/gs^2 단위로 반환된다면 각각 $\sin(\theta_1)$, $\cos(\theta_1)$이 된다. 이 상황에서 가속도 센서를 이용하여 $|\overrightarrow{OA}|$, $|\overrightarrow{OB}|$를 알 수 있으므로 역삼각함수를 이용하여 θ_1을 구할 수 있다. 가속도 센서의 Y값을 A_y, Z값을 A_z라고 가정할 때, 식(2), (3)는 다음과 같다.

$$A_y = \begin{cases} |\overrightarrow{OA}| & \text{if } (\overrightarrow{OA}\text{방향이 } \overrightarrow{OY} \text{ 방향과 일치}) \\ -|\overrightarrow{OA}| & \text{if } (\overrightarrow{OA}\text{방향이 } \overrightarrow{OY} \text{ 방향과 반대}) \end{cases} \quad (2)$$

$$A_z = \begin{cases} |\overrightarrow{OB}| & \text{if } \overrightarrow{OB} \text{ 방향이 } \overrightarrow{OZ} \text{ 방향과 일치}) \\ -|\overrightarrow{OB}| & \text{if } \overrightarrow{OB} \text{ 방향이 } \overrightarrow{OZ} \text{ 방향과 반대}) \end{cases} \quad (3)$$

이때, 가속도 센서에서 반환되는 A_y, A_z를 이용하여 θ_1을 계산하는 알고리즘을 구현하면 다음과 같다.

get θ_1 Algorithm

A_y : *Y-value of accelerometer*
A_z : *Z-value of accelerometer*

if $A_z < \alpha$ **then**
    $\theta_1 \leftarrow \arcsin(-A_y)$
else if $A_z \geq \alpha$ **then**
    $\theta_1 \leftarrow \arccos(-A_z)$
end if
α=Threshold for decision to use A_y or A_z

그림 5. θ_1을 구하는 알고리즘
Fig. 5. Algorithm to get θ_1

이렇게 구해진 θ_1과 k를 삼각함수를 이용하여 거리를 구하는 방법은 식 (4)과 같다.

$$a_n = k * \tan\theta_1 \quad (n = (1, 2, 3)) \quad (4)$$

다음 과정에서 이 방법으로 각각 3개의 거리(a_1, a_2, a_3)를 구할 것이다.

그림 6의 왼쪽 그림은 방 생성에 사용될 길이, 각도를 나타낸 것이다. 그림과 같이 사용자가 서있는 바닥의 위치를 P_0으로 잡고, 그 좌표는 (0, 0, 0)으로 잡는다. 그리고 a_1과 사용자의 시야 방향이 이루는 각을 θ_2, 바닥의 네 정점을 각각 P_1, P_2, P_3, P_4라 하자. 방을 생성하기

위해서는 이 정점들의 좌표를 알아내야 하며, 이를 위해서는 그림의 a_1, a_2, a_3의 길이, P_2를 바라보는 순간의 θ_2 값, P_3를 바라보는 순간의 θ_2값이 필요하다.

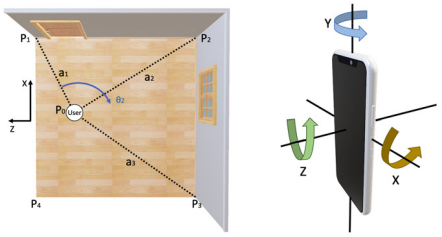


그림 6. 방의 평면도(왼쪽)와 자이로스코프 센서의 회전축 방향(오른쪽)
Fig. 6. The plane figure of the room (left) and the direction of the rotation axis of the gyroscope sensor (right)

사용자의 카메라 화면 중앙에 띄운 포인트를 P_1과 일치시켜 차례로 찍는다. 사용자가 점 P_1를 찍었을 때, 각 순간의 θ_1값을 가져와서 위의 정사영을 이용한 방법으로 a_1의 길이를 구하며, a_2, a_3도 같은 방법으로 구한다.

사용자가 위의 과정대로 P_1을 찍었다면, P_1을 찍은 시점부터 시야에 따라 변하는 θ_2값을 구하기 위해 자이로스코프 센서(Gyroscope Sensor)를 이용한다. 이 센서는 오른쪽 그림과 같이 세 개의 축을 기준으로 회전하는 각각의 각속도를 rad/s 값으로 반환한다. 기기를 좌우로 회전할 때 좌우로 시야가 이동한 각을 구하기 위해 Y, Z 방향 회전 각속도를 이용한다.

자이로스코프 센서는 특정 시점으로부터 총 회전한 각도를 반환하지는 않는다. 그렇기 때문에 자이로스코프 센서에서 반환된 값을 매 순간 더해 누적하여 사용하여야 한다. 사용자가 다음 정점을 찍기 위해서 기기를 좌우로 회전시킬 때 회전하는 축은 자이로스코프 센서의 Y, Z 축이다. 사용자 스마트폰의 기울기에 따라 Y 축과 Z 축이 회전하는 정도가 달라지며, 이 둘은 반비례한다.

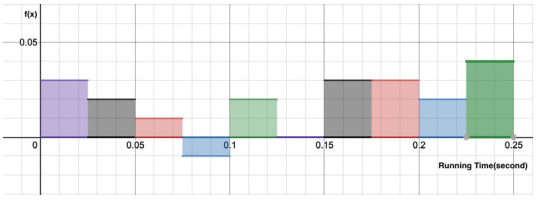


그림 7. 실행시간에 따른 $f(x)$의 그래프
Fig. 7. $f(x)$ graph over execution time

위의 그림 7은 자이로스코프 센서의 주기를 임의로 40Hz로 설정하고 그림 6의 P_1을 찍은 시점부터 0.25초간 기록한 자이로스코프 센서의 시간, 각속도 그래프 $f(x)$의 예시이다. θ_{2-y}는 자이로스코프 센서의 Y방향으로 회전한 값, θ_{2-z}는 자이로스코프 센서의 Z방향으로 회전한 값을 0.25초 마다 각각 누적한 값이다. 0.075초에서 0.1초 사이에는 반시계 방향으로 시야를 움직였고, 0.125초에서 0.15초 사이에는 정지하였으며, 나머지 구간에서는 시계방향으로 시야를 움직였다. 기기를 들고 오른쪽으로 시야를 이동할 때에는 +방향으로, 왼쪽으로 시야를 이동할 때에는 -방향으로 속도가 변하는 것을 볼 수 있다. 총 회전한 각(rad)은 다음과 같이 각속도 그래프를 적분하여 도출할 수 있다. 예컨대 위의 그래프에서 0.25초일 때 그림 6의 θ_2 값은 0.00475rad 가 된다. 총 실행시간을 T초, 자이로스코프 센서의 값을 받아오는 주기를 t초, 그래프에 그려진 함수를 f(x)라 가정했을 때 T초간 사용자가 회전한 각은 식 (5)와 같으며, 이를 알고리즘(Pseudo Code)으로 표현하면 그림 8과 같다.

$$\theta_2 = \sum_{n=0}^{\lfloor \frac{T}{t} \rfloor} \int_{nt}^{(n+1)t} f(x)\,dx \quad (rad) \tag{5}$$

get θ_2 Loop Algorithm

t : *loop cycle*
G_y : *Y-value of gyroscope sensor*
G_z : *Z-value of gyroscope sensor*
$\leftarrow$: *symbol of substitution*

$tmpG_y \leftarrow G_y$
$tmpG_z \leftarrow G_z$
$tmpA_y \leftarrow A_y$
$tmpA_z \leftarrow A_z$
if $tmpA_z < -0.5$ **then**
  $\theta_{2-y} += tmpG_y \cdot \sin(-tmpA_y)$
  $\theta_{2-z} += tmpG_z \cdot \cos(-tmpA_y)$
else if $tmpA_z \geq -0.5$ **then**
  $\theta_{2-y} += tmpG_y \cdot \cos(-tmpA_z)$
  $\theta_{2-z} += tmpG_z \cdot \sin(-tmpA_z)$
end if
$\theta_2 \leftarrow -(\theta_{2-y} + \theta_{2-z}) \cdot t$

그림 8. θ_2를 구하는 알고리즘(t초를 주기로 반복).
Fig. 8. Algorithm to get θ_2(repeat every t seconds)

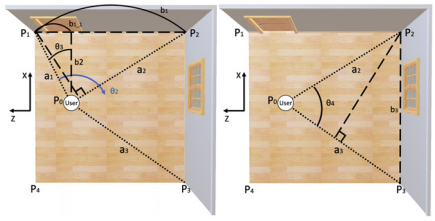


그림 9. 좌표 도출을 위한 값 표현도
Fig. 9. Expression drawing of values for deducing coordinates

사용자가 P_2를 찍게 되면 a_2의 길이를 구할 수 있다. 가구의 배치를 쉽게 하려면 바닥을 이루는 모든 선분이 좌표공간의 X 축, 또는 Z 축에 평행할 필요가 있다. 선분 $P_1P_2(b_1)$이 Z 축에 평행하기 위해서는 P_1과 P_2의 X 좌표를 일치시켜야 한다. X 좌표를 일치시키게 좌표를 구하기 위해서는 그림 9에 표시된 b_1, b_{1-1}, b_2의 값을 알아야 하고, b_1, b_{1-1}, b_2의 값을 알기 위해선 θ_3값을 알아야 한다. 필요한 값을 구하기 위한 모든 식을 나열하면 아래와 같이 나열할 수 있다.

Variables Set, Vertexes Positioning When Pointed P_2

$$b_1 = \sqrt{(a_1 \cdot \sin\theta_2)^2 + (a_2 - (a_1 \cdot \cos\theta_2))^2}$$
$$\theta_{3-1} = \arccos\left(\frac{a_2 - a_1 \cdot \cos\theta_2}{b_1}\right)$$
$$b_2 = a_2 \cdot \sin\theta_{3-1}$$
$$\theta_3 = \arccos\frac{b_2}{a_1}$$
$$b_{1-1} = a_1 \cdot \sin\theta_3$$
$$\theta_{2-1} = \theta_2$$

$$\therefore P_1 = (\ b_2,\ 0,\ b_{1-1}\ )$$
$$P_2 = (\ b_2,\ 0,\ -(b_1 - b_{1-1})\ )$$

그림 10. P_2를 찍었을 때의 변수 설정
Fig. 10. Variables set when pointed P_2

이 순간의 θ_2는 P_3의 좌표를 구하는 과정에서도 사용한다. 때문에 값을 따로 변수에 저장할 필요가 있으며 이를 위해서 변수 θ_{2-1}를 설정하였다.

사용자가 세 번째 점 P_3를 찍었을 때 b_3의 길이만 알면 P_3의 좌표를 알 수 있다. θ_4는 현재의 θ_2에서 두 번째 점을 찍었을 때의 θ_2값을 저장한 θ_{2-1}를 빼서 구할 수 있다. b_3는 삼각함수와 피타고라스 정리를 이용하여 아래와 같이 구할 수 있다. P_4의 좌표는 P_1의 Z 좌표, P_3의 X 좌표와 각각 같기 때문에 사용자가 찍지 않아도 된다.

Variables Set, Vertexes Positioning When Pointed P_3

$$\theta_4 = \theta_2 - \theta_{2-1}$$
$$b_3 = \sqrt{(a_2 \cdot \sin\theta_4)^2 + (a_3 - (a_2 \cdot \cos\theta_4))^2}$$

$$\therefore P_3 = (\ b_2 - b_3,\ 0,\ -(b_1 - b_{1-1})\ )$$
$$P_4 = (\ b_2 - b_3,\ 0,\ b_{1-1}\ )$$

그림 11. P_3를 찍었을 때의 변수 설정
Fig. 11. Variables set when points P_3

천장의 높이를 구하는 과정은 다음과 같다. 사용자에게 그림 12의 P_7을 찍게 하여 θ_5를 알아낸다. 이 때 천장의 높이는 k+h이다.

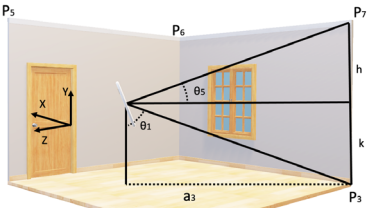


그림 12. 방의 높이를 계산하기 위한 값 표현도
Fig. 12. Expression drawing of values for estimating room's height

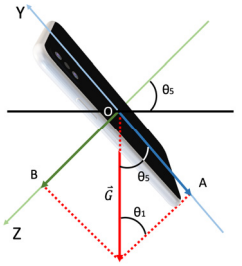


그림 13. θ_5를 구하기 위한 설명도
Fig. 13. Diagram to get θ_5

바닥과 평행한 평면과 기기가 바라보는 방향 사이의 각 θ_5는 식 (6)과 같이 구할 수 있으며, 방의 높이 k+h는 식 (7)과 같이 구할 수 있다.

$$\theta_5 = \theta_1 - \frac{\pi}{2} \quad (6)$$

$$k+h = k + a_3 \cdot \tan\theta_5 \quad (7)$$

각각의 P_1, P_2, P_3, P_4 위에 존재하는 P_5, P_6, P_7, P_8은 천장을 나타내며 이들은 각각 좌표공간의 X좌표, Z좌표가 동일하다. 정점들의 Y좌표는 k+h이다.

이제 렌더링에 필요한 모든 좌표를 알아냈다. 후술할 과정을 위해 좌표를 다음과 같이 평행이동 시킨 후 P_0을 제외한 모든 정점들을 꼭짓점으로 직육면체를 생성한다.

Creating Coordinate Model

$P_0 = (\ b_3 - b_2,\ 0,\ b_1 - b_{1-1}\ )$	$P_5 = (\ b_3,\ k+h,\ b_1\ )$
$P_1 = (\ b_3,\ 0,\ b_1\ )$	$P_6 = (\ b_3,\ k+h,\ 0\ )$
$P_2 = (\ b_3,\ 0,\ 0\ )$	$P_7 = (\ 0,\ k+h,\ 0\ )$
$P_3 = (\ 0,\ 0,\ 0\ )$	$P_8 = (\ 0,\ k+h,\ b_1\ )$
$P_4 = (\ 0,\ 0,\ b_1\ )$	

그림 14. P_3를 원점으로한 각 정점의 좌표값
Fig. 14. The coordinate of vertices when P_3 is the origin.

위 그림에 나타난 바와 같이 실제 방 크기에 맞춘 가상의 방을 생성했다면 사용자마다 방의 크기 및 비율이 다를 것이다. 실내에 가구 오브젝트를 적절한 위치에 배치하기 위해서는 방 크기에 맞게 조정된 좌표가 필요하다. 아래 그림 15의 과정을 통해 각자 방에 맞게 좌표를 조정할 수 있다.

Furnitures Positioning

가구의 세로, 높이, 가로를 각각 lengthX, lengthY, lengthZ, 양방향 여백의 비율 중 좌표값이 더 작은 곳의 백분율을 각각 blankX, blankY, blankZ, 그리고 X, Y, Z 좌표를 각각 locationX, locationY, locationZ라 할 때, 비율 좌표로 변환하는 식은 다음과 같다.

$$blankX = \frac{locationX - \frac{lengthX}{2}}{b3 - lengthX} \cdot 100$$

$$blankY = \frac{locationY - \frac{lengthY}{2}}{(k+h) - lengthY} \cdot 100$$

$$blankZ = \frac{locationZ - \frac{lengthZ}{2}}{b1 - lengthZ} \cdot 100$$

실제 좌표로 변환하는 식은 다음과 같다.

$$locationX = \frac{(b3 - lengthX) \cdot blankX}{100} + \frac{lengthX}{2}$$

$$locationY = \frac{(k+h-lengthY) \cdot blankY}{100} + \frac{lengthY}{2}$$

$$locationZ = \frac{(b1 - lengthZ) \cdot blankZ}{100} + \frac{lengthZ}{2}$$

그림 15. 방의 비율에 따라 가구의 좌표를 생성하는 방법
Fig. 15. The method for generating furniture coordinates based on room proportions

디자이너가 실제 좌표로 인테리어를 디자인 한 경우 비율 좌표로 변환하는 식을, 반대로 사용자가 디자인을 적용했을 때 사용자의 방에 맞추어야 하므로 실제 좌표로 변환하는 식을 사용한다.

그다음 화면에 출력할 때, VR 보기를 구현하려면 기기의 회전에 따라 생성된 방 모델도 회전에 맞추어 회전을 할 필요가 있다. 방 모델 생성 후 θ_1, $\theta_6(\theta_6 = \frac{\pi}{2} - \theta_3)$을 이용하여 현재 시야에 맞게 실제 방과 생성된 방을 일치시킨다. 이후 자이로스코프 센서를 이용해서 시야의 회전을 구현한다.

P_0에서 Y 방향으로 k 만큼 이동한 ($-b_2$ + b_3, k, $-b_{1-1}$) 좌표에 카메라를 배치한다. Unity에서는 카메라의 로테이션이 (0, 0, 0)이면 +Z 방향을 바라보게 된다.

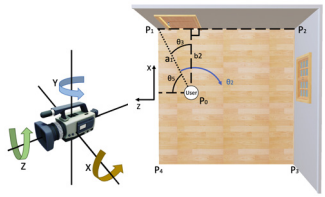


그림 16. 카메라의 회전축 구성과 좌우 회전값을 구하기 위한 설명도
Fig. 16. Configuration of the camera's rotation axis and diagram to obtain the left and right rotation values

카메라의 좌우 회전인 Y 값은 시계방향 회전이 양수 방향이므로 $(\theta_6 + \theta_2)$ rad로 설정하며, 반복 함수에 포함하여 좌우로 계속 회전할 수 있게 한다. 본 논문에서는 호도법을 사용하여 모든 각을 나타냈지만 Unity의 경우 오브젝트의 회전에는 도(°) 단위를 쓰는 육십분법을 사용하므로 위에서 구한 값에 $\frac{180}{\pi}$ 값을 곱하여 육십분법으로의 변환이 필요하다.

Convert and apply camera's rotation

$$revX \leftarrow -\theta_5 \cdot \frac{180}{\pi}$$

$$revY \leftarrow -(\theta_6 + \theta_2) \cdot \frac{180}{\pi}$$

Loop Function

$$rotX -= G_x$$

$$arCamRot \leftarrow \left((rotX + revX) \cdot \frac{180}{\pi}, (\theta_2 + revY) \cdot \frac{180}{\pi}, 0 \right)$$

그림 17. 시야 방향 보정 후 자이로스코프 센서를 이용하여 시야의 회전을 구현하는 방법
Fig. 17. The metod for implementing the rotation of the field of view using a gyroscope sensor after correcting the direction of the field of view

시야 이동을 구현하기 위해 자이로스코프 센서의 X, Y, Z 좌표를 사용하여 VR 미리보기를 실행하기 전에 가속도 센서를 이용하여 위 그림과 같이 각도 값을 보정하여야 한다.

카메라의 상하 회전인 X 값은 고개를 드는 방향으로의 회전이 음수 방향이다. 다음 코드는 회전X값을 보정하는 코드이다. 정면을 바라보는 상태에서 위로 시야를 이동하든 아래로 이동하든 θ_1은 감소하지만 Z방향의 힘은 방향이 역전되기 때문에 고개를 드는 방향으로 회전할 경우 +방향, 숙이는 방향으로 회전하면 -방향이다. 이

점을 이용하여 Z값으로 기기가 어느 방향을 바라보고 있는지 판단하여 보정을 할 필요가 있다.

III. 모의 실험

그림 18. 거치대에 회전 각도기를 붙여 실제 각도와 θ_1의 비교.
Fig. 18. Attach a rotating protractor to the cradle and compare the actual angle and θ_1.

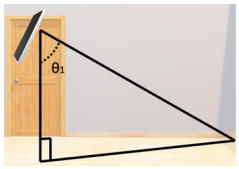
모델 측정을 위한 센서 사용 식이 정확한지 알아보기 위해 간단한 모의 실험을 진행하였다. 실험에 사용한 기종은 LG V10, Samsung Galaxy S10+이다. 모의 실험을 위한 프로그램 작성에 Unity를 사용하였으며, 가속도 센서는 스크립팅 API Input.acceleration을, 자이로스코프 센서는 스크립팅 API Input.gyro를 통해 사용하였다.[5][6] 이 실험은 정확한 측정 장비를 이용하지 않았으므로 오차가 있을 수 있다.

```
public float getTheta1()
{
    float x= Input.acceleration.y;
    float tempZ = Input.acceleration.z;
    if (float.IsNaN(x)) x = Mathf.PI /2;
    return x;
}
```

그림 19. 가속도 센서의 y좌표만을 이용하여 작성한 getθ_1 함수
Fig. 19. The getθ_1 function created using only the y-coordinate of the accelerometer

첫 번째 실험은 A_y만을 이용해서 θ_1을 측정하였다. x변수를 선언하는 줄의 식에서 Input.acceleration.y는 본 논문의 A_y이고, $\arcsin(-A_y)$ 식을 적용했다. $\{\theta_1 \mid 0 \leq \theta_1 \leq \pi\}$일 때 A_y의 값은 음수의 범위 내에서 결정되기 때문에 -1을 곱해주었다. $\theta_1 = \frac{\pi}{2}$일 때 $-A_y$의 값은 1이 된다. 하지만 기기의 흔들림으로 인해 1을 초과하는 경우가 생기는데, arcsin 함수의 정의역의 구간은 [-1, 1]이다. 이 구간을 벗어난 값을 Mathf.Asin()의 매개변수로 전달하면 NaN 값이 반환되기 때문에 Mathf.Asin()에서 NaN이 반환되었을 경우 getTheta1()에서 $\frac{\pi}{2}$값인 Mathf.PI / 2를 반환하도록 해야 한다.

표 1. 각도기로 잰 각도$\theta_1(P)$와 기기별 $\theta_1(Result)$값.
Table 1. Angles $\theta_1(P)$ measured with a protractor and $\theta_1(Result)$ values for each device.



$\theta_1(P)$	$\theta_1(Result)$	
	V10	S10+
0	0	0
37	34	33
45	42	41
60	56	55
90	82	81

표 1에서 단위는 도이며, 첫 번째 열은 각도기로 잰 θ_1(P) 값이다. 2, 3번째 열은 각각 V10, S10+ 기종으로 getTheta1() 함수를 이용하여 θ_1(Result)을 구한 것이다.

표 1에서 주목해야 할 것은 각도기로 잰 θ_1(P) 값과 getTheta1()로 얻어낸 θ_1(Result) 값의 차이이다. 0도~45도 사이에서는 측정 장비의 오차범위 내에서 값이 측정되었으나 90도에 근접하였을 때는 측정 장비의 오차범위를 넘어 오차가 커지는 것을 볼 수 있었다.

이어서 두 번째 실험은 θ_2 값 도출의 정확성을 알아보는 실험이다. 가속도 센서의 Y값과 자이로스코프 센서만을 사용하여 θ_2를 측정할 경우 역시 θ_1(Deg)의 값이 커질수록 오차가 커지는 문제가 생길 수 있음을 확인할 수 있었다.

```
public float getTheta2()
{
    return -(theta2_y + theta2_z)*gyroFixrate;
}

void FixedUpdate()
{
    float tempY = Input.acceleration.y;
    theta2_y += Input.gyro.rotationRateUnbiased.y * Mathf.Sin(-tempY);
    theta2_z += Input.gyro.rotationRateUnbiased.z * Mathf.Cos(-tempY);
    /*이하 코드 생략*/
}
```

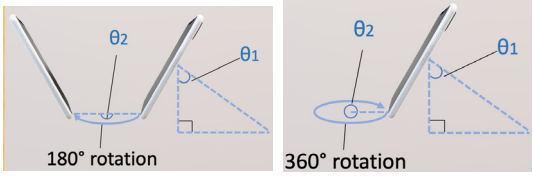
그림 20. 가속도 센서의 Y좌표와 자이로스코프 센서를 이용하여 작성한 getθ_2 함수
Fig. 20. The get θ_2 function using the Y-coordinate of the accelerometer and the gyroscope sensor

θ_2를 계산할 때 기기의 기울기 값인 θ_1을 사용한다. 따라서 기울기가 각각 다른 상황에 따른 오차범위 차이를 알아볼 필요가 있다. 즉, 두 번째 실험은 각도기를 이

용하여 기기의 기울기를 각각 0도, 37도, 45도, 60도, 90도로 설정하고, 기기를 바닥에 수직한 축을 중심으로 180도, 360도로 회전하여 얻은 θ_2값이 정확한지 알아보는 실험이다. getTheta2() 함수는 θ_2를 리턴하는 함수이고 FixedUpdate()는 설정된 주기마다 실행되는 함수이다. theta2_y, theta2_z는 함수 외부 클래스에서 선언된 float형 변수이다. theta2_y는 매 주기마다 $G_y \cdot \sin(-A_y)$을 누적하며 더한 값을 저장한다. theta2_z는 매 주기마다 $G_z \cdot \cos(-A_y)$을 누적하며 더한 값을 저장한다.

표 2. 각 θ_1 값에 따라 기기를 180도, 360도 회전 시킨 후의 θ_2의 값

Table 2. θ_2 value after rotating the device 180 degrees and 360 degrees according to each θ_1 value



$\theta_1(P)$	$\theta_2(Result)$	
	V10	S10+
0	180	180
37	178	181
45	178	184
60	178	180
90	155	156

$\theta_1(P)$	$\theta_2(Result)$	
	V10	S10+
0	361	359
37	357	363
45	358	364
60	350	356
90	307	310

표 2는 사용자가 휴대폰을 들고 180도 돌았을 때와 360도 돌았을 때 각각 getTheta2() 함수의 반환 값을 측정하여 정리한 표이다. θ_1이 0일 때에는 비교적 정확한 θ_2 값이 측정되었으나 θ_1의 값을 크게 하여 측정할수록 정확도가 떨어지는 현상을 볼 수 있었다. 첫 번째 실험의 θ_1 정확도 문제가 θ_2 산출에도 영향을 준 것으로 볼 수 있다.

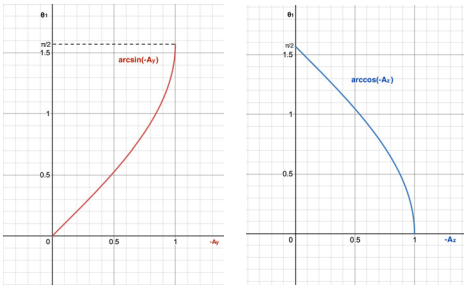


그림 21. 가속도센서의 값에 따른 θ_1 함수 그래프.

Fig. 21. θ_1 graph according to the value of the accelerometer.

그림 21에서 왼쪽 그래프는 $\theta_1 = \arcsin(-A_y)$이며, 오른쪽 그래프는 $\theta_1 = \arccos(-A_z)$를 의미한다. 왼쪽의 $\theta_1 = \arcsin(-A_y)$ 함수는 x가 증가할수록, 오른쪽의 $\theta_1 = \arccos(-A_z)$ 함수는 x가 감소할수록 기울기 절대값이 무한대로 발산하는 것을 볼 수 있다. 기울기 절댓값이 커질수록 가속도 센서 값의 변화에 따른 θ_1 값의 변화가 커지며, 손떨림 등의 진동에 기울기 절댓값이 작은 구간보다 더 큰 오차를 발생시킬 수 있다. 실제로 첫 번째, 두 번째 실험에서 $\arcsin(-A_y)$ 그래프의 기울기의 증가량이 커질수록 정확성이 떨어지는 양상을 보였다. 두 그래프는 기울기 절댓값이 커지는 구간과 방향이 각각 다르지만 똑같이 θ_1을 구하는 함수이므로, 기울기 절댓값에 따라 arcsin 함수를 사용하는 A_y, arccos 함수를 사용하는 A_z를 사용할 필요가 있다.

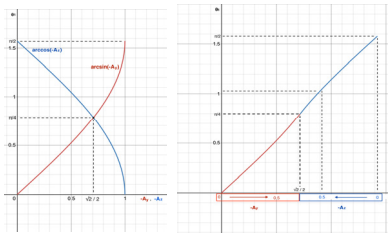


그림 22. 비교를 위해 두 개의 그래프를 한 좌표평면에 표현한 그림.

Fig. 22. A picture of two graphs represented on one coordinate plane for comparison.

그림 22에서 왼쪽 그림은 $\theta_1 = \arcsin(-A_y)$, $\theta_1 = \arccos(-A_z)$ 그래프를 한 좌표평면에 그린 것이고 오른쪽 그림은 $\theta_1 = \arccos(-A_z)$를 좌우반전시켜 $\theta_1 = \arcsin(-A_y)$에 일부를 이어그린 그래프를 나타낸 것이다. 그림 22의 왼쪽에 그려진 두 그래프를 보면 점 $(\sqrt{2}/2,\ \pi/4)$에서 만나는 것을 볼 수 있다. 그림 22의 오른쪽에 그려진 그래프와 같이 이 점을 임계점으로 지정하여, $-A_y$값이 $\sqrt{2}/2$보다 커지면 $-A_z$를 사용하여 최대 기울기 절댓값이 $\sqrt{2}$를 넘지 않도록 한다.

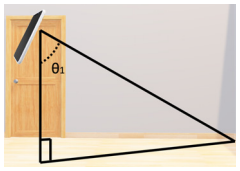
```
public float getTheta1()
{
    float x= Input.acceleration.y;
    float tempZ = Input.acceleration.z;

    if (tempY>-0.707f&&tempZ<0) x = Mathf.Asin(-tempY);

    else x = Mathf.Acos(-tempZ);

    return x;
}
```

그림 23. 수정된 getθ_1 함수

Fig. 23. The modified getθ_1 function

세 번째 실험은 getTheta1() 함수를 위와 같이 수정한 후에 첫 번째 실험을 다시 하는 방식으로 진행하였다. 수정 전의 첫 번째 실험과 달리 본문에 작성된 대로 일정 기울기에 따라 A_y과 A_z를 선택하여 θ_1을 계산하는 코드를 작성하였다. Z의 값이 -0.5 아래로 내려가면 A_y, 그렇지 않으면 A_z를 사용하도록 하였다.

표 3. 각도기로 잰 각도$\theta_1(P)$와 기기별 $\theta_1(Result)$값.
Table 3. Angles $\theta_1(P)$ measured with a protractor and $\theta_1(Result)$ values for each device.



θ_1(P)	θ_1(Result)	
	V10	S10+
0	0	0
37	35	36
45	42	43
60	58	58
90	90	90

앞서 A_y만을 이용한 코드로 실험한 결과와 달리 기울기의 정확도가 개선되었음을 알 수 있다.

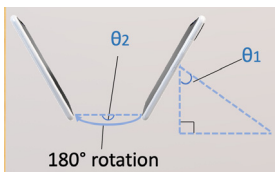
```
void FixedUpdate()
{
    float tempY = Input.acceleration.y;
    float tempZ = Input.acceleration.z;
    if (tempY>-0.707f)
    {
        theta2_y += Input.gyro.rotationRateUnbiased.y * Mathf.Sin(-tempY);
        theta2_z += Input.gyro.rotationRateUnbiased.z * Mathf.Cos(-tempY);
    }
    else
    {
        theta2_y += Input.gyro.rotationRateUnbiased.y * Mathf.Cos(-tempZ);
        theta2_z += Input.gyro.rotationRateUnbiased.z * Mathf.Sin(-tempZ);
    }
    /*이하 코드 생략*/
}
```

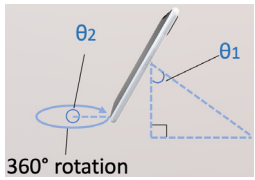
그림 24. 가속도 센서의 Z값 크기에 따라 가속도 센서의 Y값 또는 Z값을 선택하여 θ_2계산에 반영하는 FixedUpdate() 함수.
Fig. 24. FixedUpdate() function that selects the accelerometer's Y value or Z value according to the accelerometer's Z value size and reflects it in the θ_2 calculation.

어서 네 번째 실험에서는 두 번째 실험과 같이 θ_2를 계산하는 실험을 하되 기울기를 세 번째 실험과 같이 계산하여 반영하는 실험을 진행하였다. 이때, θ_2를 계산할 때 사용하는 누적변수 theta2_y, theta2_z를 계산하는 FixedUpdate() 함수를 그림 24와 같이 수정하였다.

θ_2를 반환하는 두 번째 실험에서는 θ_1(Deg)의 값이 90에 가까울수록 오차가 커지는 결과를 얻었던 반면, 위 실험에서는 θ_1(Deg)의 값에 관계 없이 오차범위가 일정한 것으로 정확도가 개선되었음을 알 수 있었다.

표 4. 각 θ_1 값에 따라 기기를 180도, 360도 회전 시킨 후의 θ_2의 값
Table 4. θ_2 value after rotating the device 180 degrees and 360 degrees according to each θ_1 value





θ_1(가정)	θ_2(Result)	
	V10	S10+
0	180	180
37	180	181
45	182	180
60	180	178
90	179	181

θ_1(가정)	θ_2(Result)	
	V10	S10+
0	360	360
37	361	361
45	359	361
60	358	360
90	360	360

그림 25. 시뮬레이션 결과를 찍은 스크린샷(예)
Fig. 25. Screenshots of the simulation result(example)

가로 길이 294cm, 세로 길이 405cm, 높이 230cm인 실제 방을 대상으로 본 논문에서 구현한 가구배치 인테리어 시뮬레이션 위한 애플리케이션 테스트를 진행했다(그림 25 참조). 테스트로 측정된 방의 크기는 가로 길이 300cm, 세로 길이 400cm, 높이 253cm로 가로와 세로의 길이는 실제 방의 길이와 오차범위 10cm 이내의 결과를 얻었지만, 높이의 경우 23cm의 오차를 내었다. 이러한 결과를 얻은 이유에는 측정자마다 각기 서로 다른 측정 자세로부터 기인한다. 즉 측정기기의 높이를 애플리케이션에 입력된 높이보다 높게 위치하는 등 자세의 문제로 오차가 발생하는 문제로서 앞으로 해결해 나가야 할 과제이다.

IV. 결 론

장기간의 COVID19 감염 사태로 인하여 온라인 쇼핑

수요가 계속적으로 증가하고 있는 상황이며 실물을 직접 체험하지 않고, 소비자의 구매결정에 실질적 도움을 줄 수 있는 정보를 제공하는 것이 중요해졌다. 이와 관련하여 세계적 가구회사로 유명한 이케아(IKEA)사에서 개발된 IKEA Place 앱이 존재하나, 이것은 완성된 인테리어 배치까지 사용자에게 보여줄 수 없으며, 처리해야하는 연산의 양이 많기 때문에 저성능 스마트폰에서는 원활하게 이용할 수 없는 문제점이 존재한다. 이에 따라 본 논문에서는 Google의 ARCore, Apple의 ARKit를 보다 간편하게 대체할 수 있는 실용적 기술을 제시하였다. 즉, 카메라로 얻은 평면 이미지를 입체화하는 기존 방법과 차별화하여 이러한 복잡한 입체화 연산 없이 간편하게 사용자 스마트폰을 통해서 얻은 방 정보와 삼각함수 및 피타고라스 정리를 이용하여 사용자 맞춤형의 인테리어 디자인을 적용하고자 하는 것이 본 논문의 핵심적 제안사항이다. 또한 그 활용방안으로서 미리 디자인된 인테리어 템플릿을 적용하는 방법을 제시하였다. 언택트 마케팅이 중심이 된 시장에서 복잡하고 한계가 큰 방법을 사용하지 않아 소비자의 제품 선택에 더 도움을 줄 것으로 기대된다. 더 나아가서 인테리어 디자인 말고도 시뮬레이션, 거리 측량, 게임 등 더 다양한 분야에서 이 설계를 응용할 수 있을 것으로 기대된다.

References

[1] KOSTAT, 『Online Shopping Trends in November 2020』, 2021.01.05., p3-4.

[2] Say hej to IKEA Place, https://www.ikea.com/au/en/customer-service/mobile-apps/say-hej-to-ikea-place-pub1f8af050

[3] Overview of ARCore and supported development environments, https://developer.google.com/ar/develop/fundamentals

[4] ARKit Overview - Augmented Reality, https://developer.apple.com/augmented-reality/arkit/

[5] Unity - Scripting API : Input.acceleration, https://docs.unity3d.com/ScriptReference/Input-acceleration.html

[6] Unity - Scripting API : Gyroscope, https://docs.unity3d.com/ScriptReference/Gyroscope.html

[7] Kwonhee Lee, Kwanghyun Kim, and Jongtaek Oh, "A Study on step number detection using smartphone sensors for position tracking", The Journal of The Institute of Internet, Broadcasting and Communication(JIIBC), Vol. 18, No. 3, pp 119 ~ 125, 2018.

[8] Dong-Min Lee, Seung-Jung Shin, "Real-Time Visual Production using Unity 3D", International Journal of Advanced Smart Convergence(IJASC) , Vol. 7, No. 4, pp 138 ~ 146, 2018.

[9] Sun Uk Kim, A Comparative Evaluation of User Experience Design on Virtual Reality Indoor Bikes, Journal of the Korea Academia-Industrial cooperation Society(JKAIS) , Vol. 19, No. 1, pp 48 ~ 55, 2018.

저 자 소 개

최 규 석(종신회원)

- 제 9권 6호 참조
- 1991 ~ 1996 : ㈜SK텔레콤 중앙연구원 책임연구원
- 1997 ~ 현재 : 청운대학교 컴퓨터공학과 교수
- 주관심분야 : 인공지능, 이동통신, ITS, 빅테이터, 가상현실, 증강현실

김 준 건(준회원)

- 2019 ~ 현재 : 청운대학교 컴퓨터공학과 학사과정
- 주관심분야 : 가상현실, 증강현실, 인공지능

임 창 묵(준회원)

- 2019 ~ 현재 : 청운대학교 컴퓨터공학과 학사과정
- 주관심분야 : 인공지능, 딥러닝, 가상현실, 게임개발