International Journal of
Advanced Science Computing and Engineering

Journal Homepage www.ijasce.org/index.php/ijasce

Augmented Reality Applications for Cultural Heritage Using Kinect Sensor

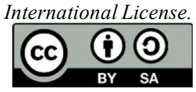
Rita Afyenni [a], Aldo Erianda [a], Putri Wahyuni [a], Ardian Firosha [a], Taufik Gusman [a], Sumema [a]

[a] *Department of Information Technology, Politeknik Negeri Padang, Padang, Indonesia*
Corresponding author: [*] aldo@pnp.ac.id

***Abstract*—There are quite a number of advantages gained from Microsoft Kinect which is a motion sensing device that also includes AR applications to aid in the preservation and promotion of cultural assets. In this work, a batik model was developed with the help of Microsoft Kinect by utilizing its core components consisting of a depth sensor, an RGB camera and a microphone array to allow motion tracking, gesture recognition, and voice command functionalities of the device.The depth sensing features of the device, in particular the structural light employed, transition to Time-of-Flight technology has improved the efficiency and applicability of the device towards AR. Kinect has the ability to create an immersive AR experience by accurately mapping virtual objects to the user's environment, leveraging body joint tracking to create a 3D skeletal model. It is required to take into consideration the kinnect positioning technique which has importance in allowing 3D objects and models to align. Therefore, the conclusion addresses Kinect as it relates to human-computer interaction, in particular, concerning its potential to change body interaction in real time and how it sets the stage for future systems that require an active interaction across a number of fields including culture and heritage.**

***Keywords*— Kinect; sensor; augmented reality; batik.**

Manuscript received 14 Sep. 2024; revised 29 October. 2024; accepted 4 November. 2024; Date of publication 31 Dec. 2024.



I. INTRODUCTION

Microsoft Kinect is a motion-sensing input device that was originally designed for the Xbox gaming console. Launched in 2010, Kinect uses a combination of a camera, depth sensor, and microphone array to capture motion, gestures, and voice commands, allowing users to interact with computers or game consoles in a more natural, immersive way without needing traditional controllers [1]-[11].

The original Kinect was intended as a gaming peripheral, enabling gesture-based gaming and hands-free control of the Xbox interface. However, developers and researchers quickly recognized its potential for a broad range of applications outside of gaming. The Kinect's core technology includes a depth sensor that uses infrared light to gauge distance, a camera for capturing RGB data, and an array microphone for sound localization and speech recognition.

By 2011, Microsoft released the Kinect SDK (Software Development Kit) for Windows, allowing developers to create applications for the device beyond gaming. This opened the door for a wide array of uses, including in fields such as robotics, healthcare, art, and education. The SDK included tools for skeletal tracking, face recognition, speech recognition, and more, making it a versatile tool for computer vision and human-computer interaction research [12]-[15].

II. MATERIALS AND METHOD

The Microsoft Kinect's core technology consists of three primary components: a depth sensor, an RGB camera, and an array microphone. These elements work together to capture rich data that enables motion sensing, gesture recognition, and voice commands.

A. Depth Sensor

The depth sensor is arguably the most innovative and crucial component of the Kinect. It allows the device to "see" in three dimensions, which is fundamental for motion tracking and gesture recognition. The Kinect's depth sensor uses infrared (IR) light to measure the distance between the sensor and objects in the environment. Specifically, the

original Kinect for Xbox 360 utilized a technology developed by PrimeSense, called structured light. The sensor consists of an infrared projector and an infrared camera. The projector emits a pattern of infrared light (often a dot matrix) into the environment. The camera then captures the reflected infrared light from the objects. The depth is calculated based on how the projected pattern is distorted when it bounces back from objects. By analyzing these distortions, the system determines the distance to various points in the scene, creating a depth map — a 3D representation of the environment. In the Kinect for Xbox One and later versions, Microsoft switched to a different depth-sensing technology known as Time-of-Flight (ToF). The ToF sensor sends out pulses of infrared light and measures the time it takes for the light to reflect back to the sensor. By calculating the time, it takes for the light to travel to an object and back, the sensor can accurately gauge distances.

B. RGB Camera

The Kinect also includes a standard RGB camera, similar to the cameras found in webcams or smartphones. The RGB camera captures high-resolution color images (at 640x480 resolution for the original Kinect for Xbox 360 and 1920x1080 resolution for the Kinect for Xbox One). This camera operates like any conventional digital camera, capturing visual data in red, green, and blue color channels.

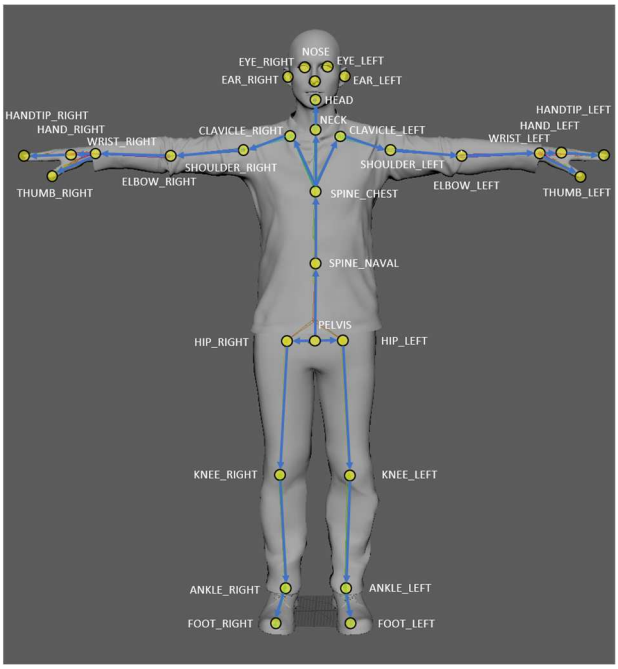


Fig. 1 Human Skeleton Joint

C. Array Microphone

The Kinect includes an array of microphones designed for advanced sound localization and speech recognition. The Kinect has a multi-microphone array (usually consisting of four microphones) that captures audio from the environment. The microphones are spaced apart on the device to allow for beamforming, a signal processing technique that uses the spatial separation between microphones to determine the direction of sound. The microphone array can filter out background noise and focus on sounds coming from a specific direction, such as the user's voice, improving the quality and accuracy of voice commands.

D. Body Joint

Microsoft Kinect is a motion-sensing input device that uses depth-sensing technology to track human body movements in 3D space. The image represents a 3D skeletal model with labeled body joints, which Kinect uses to understand and analyze human poses and movements.

The head and neck joints include several anatomy points. Nose, Right Eye, Left Eye, Right Ear and Left Ear are useful as facial landmarks in establishing the orientation and positioning of the head. For operations tracking head position and movements, the head provides the central anchor. Neck supports the head, which is positioned above the torso and rotates along with head translations.

The head and neck joints have a few important components joints. Nose, Right Eye, Left Eye, Right Ear, and Left Ear functions which serve as clutch connectors of the body, turning the head, jaw, and face upwards and downwards. The Head joint is used as the primary reference point where head position is monitored while the Neck joint can be considered as the root of head, where the head is joined and is adjusted to the upper torso, thereby allowing range of motion around head tilts and neck region.

The joints of upper body are very important in detecting the body movement and gesture. The markers in this region relate to what is known as the clavicle joints on the left and right which respectively correspond to the collarbone that connects the neck and shoulders.(1) With shoulder joints on each side contributing to shoulder position and movement tracking, our arm movements are able to be perceived reliably. The elbow joints (right, left) leverage data to track the forearms, and wrist joints capture the position of the wrist, providing data to track the hands. The hand joints where both of the hands are located also indicate gesture interpretation. Finally, thumb and handtip joints located on every hand record fine details of hand movement.

Since the spine and torso joints represent points of interest for tracking torso orientation and posture. The Spine_Chest joint corresponds to the upper chest or upper spine to help calculate the rotation of the upper torso. The Spine_Naval joint — near the belly button — is used to detect mid-body position and motion. The Pelvis joint represents the center of mass of the body and its position at the base of the spine signifies its position, making it a crucial component in monitoring the movements and keeping the overall posture of the lower body stable.

Lower body joints are used for tracking the leg movement and for balance which is critical. Two joints with Right and Left Hip names that connect the torso to the legs and represent hip positions, allowing for leg movement tracking. Knee positions (Right and Left Knee joints) can be used to observe the movements of the lower legs while walking or bending. Right and Left Ankle joints which denote position of the ankle, help track foot movement and balance. The Right Foot and Left Foot joints, at last, display the location of the feet, shown in turn, they give good information for foot ground contact points.

Kinect uses depth sensing, human detection, and skeletal mapping in order to track body joints in three steps using a

series of one-time steps. In the first instance, it combines the use of infra camera and infrared sensors in order to capture depth data which would in turn assist in making 3D maps of the area. It is through this technique that Kinect is able to detect human bodies through depth information by considering the depth as the shape of contours that fit in human form. After a body is tracked, it overlays a 3D skeletal template on the body, which uses the identifiable joint points on the body for the purpose of tracking the different body parts. Kinect generates 3D coordinates (X, Y, Z) of each joint with respect to the sensor allowing accurate positioning during movement. And this tracking activity would emphasize the pose of the joints until the last one.

III. Result and Discussion

The results of the development of a traditional batik cloth application using the Kinect sensor were obtained as shown in the code and image below.

```
KinectManager manager = KinectManager.Instance;
if(manager && manager.IsInitialized())
{
    Texture2D depthTexture = manager.GetUsersLblTex();
    Texture2D colorTexture = manager.GetUsersClrTex();
    // do something with the textures
}
```

Fig. 2 Get depth- or color-camera textures

This code segment grabs the depth and color textures using a KinectManager instance, which is the one who manages the operations of Kinect. It starts by creating an instance of the KinectManager class (KinectManager manager = KinectManager.Instance;) followed by a testing of Kinect Manager that it has been initialized properly (if (manager && manager.IsInitialized())). This is done in order to ensure that the Kinect sensor is ready to provide data before proceeding. After preparing the sensor, the code goes on to obtain two textures: 'depthTexture' which is about the depth data —Textures are 2D images where each pixel is used to encode information about depth., which is depth data — Textures are 2D images where each pixel is used to encode information about depth. and colorTexture' about the color of the image using manager. GetUsersClrTex()to get a perspective of the scene. In general, this code is designed to fetch and make use of the color and depth textures of the cameras provided in the Kinect, to trigger applications that are based on the depth and color data of the Kinect.

```
KinectInterop.JointType joint = KinectInterop.JointType.HandRight;
KinectManager manager = KinectManager.Instance;

if(manager && manager.IsInitialized())
{
    if(manager.IsUserDetected())
    {
        long userId = manager.GetPrimaryUserID();

        if(manager.IsJointTracked(userId, (int)joint))
        {
            Vector3 jointPos = manager.GetJointPosition(userId, (int)joint);
            // do something with the joint position
        }
    }
}
```

Fig. 2 Get position on join body

The following Code track and obtain the position of right hand (HandRight) joint of user detected by kinect. The first line creates a variable joint of type KinectInterop. KinectInterop. JointType where JointType the right hand.) JointType. HandRight).

Afterwards, it creates a new example instance of KinectManager like so (where KinectManager. Instance;) and checks for manager initialization (if (manager && manager. (KinectSensor.KinectSensors[0]. Is Initialized())`), to make sure that Kinect is ready to use.

After initialization, the code also checks for a user presence in the Kinect view (if (manager. IsUserDetected())). If it is able to detect a user, it uses manager to get the ID of the primary user. GetPrimaryUserID(). Having the user ID, it then checks if the specified joint (right hand) is being tracked (if (manager. JointTracked => Here we have to check if the joint is tracked or not (Joint Tracked => (Int, bool)) (e.g. IsJointTracked(userId, (int)joint)). In the case that the joint is being tracked, the code calls `manager.

GetJointPosition(userId, (int)joint) which returns a Vector3 representing the joint coordinates X, Y, Z that indicates the joint position that was retrieved can be used to drive a control over some object or initiate actions based on hand position activity.

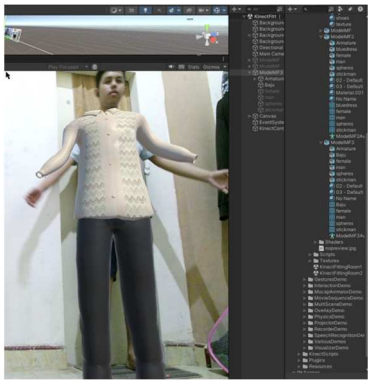


Fig. 3 Join Body

```
KinectManager manager = KinectManager.Instance;
if(manager && manager.IsInitialized())
{
    int usersNow = manager.GetUsersCount();

    if(usersNow > usersSaved)
    {
        // new user detected
    }
    if(usersNow < usersSaved)
    {
        // user lost
    }

    usersSaved = usersNow;
}
```

Fig. 4 Special event handlers for user detection

Using KinectManager, This code snippet checks real-time number of users represented by Kinect Sensors. It first finds if there is an initialized and available instance of KinectManager and assigns it to manager. Instance and check that manager is non-null and initialized. If it meets these requirements, it calls the manager. Below, it calls the method GetUsersCount() and saves the information in the variable usersNow. Code compares usersNow to usersSaved, a

variable that holds the previous number of users that were detected. If usersNow > usersSaved, a new user has entered the sensor field. Otherwise, if usersNow is less than usersSaved, then a user has exited the range of the sensor. The usersSaved variable is then updated with what is currently in usersNow, since it gets used again in the next cycle, this ensures we are only tracking the changes. It also allows for quickly identifying when users are added or removed.

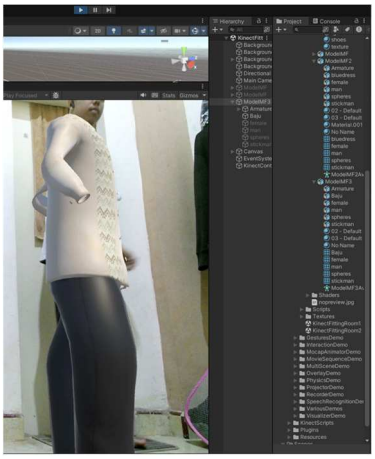


Fig. 5 User Detection

```
void depthFrameReader_FrameArrived(object sender, DepthFrameArrivedEventArgs e)
{
    var depthFrame = e.FrameReference.AcquireFrame();
    if (depthFrame != null)
    {
        using (depthFrame)
        {
            depthFrame.CopyFrameDataToIntPtr(depthImage.DataIntPtr, Kinect2Calibration.depthImageWidth*Kinect2Calibration.depthImageHeight*2);

            // convert depth image coords to color image coords
            int x = 100, y = 100;
            ushort depthImageValue = depthImage[x, y]; // depth image values are in mm

            if (depthImageValue == 0)
            {
                Console.WriteLine("Sorry, depth value input coordinates is zero");
                return;
            }

            float depth = (float)depthImageValue / 1000f; // convert to m, to match our calibration and the rest of the Kinect SDK
            double colorX, colorY;
            calibration.DepthImageToColorImage(x, y, depth, out colorX, out colorY);

            //// when converting many points, it may be faster to precompute pass in the distortion table:
            //var depthFrameToCameraSpaceTable = calibration.ComputeDepthFrameToCameraSpaceTable();
            //calibration.DepthImageToColorImage(x, y, depth, depthFrameToCameraSpaceTable, out colorX, out colorY);

            Console.WriteLine("our color coordinates: {0} {1}", colorX, colorY);

            // compare to Kinect SDK
            var depthSpacePoint = new DepthSpacePoint();
            depthSpacePoint.X = x;
            depthSpacePoint.Y = y;
            var colorSpacePoint = kinectSensor.CoordinateMapper.MapDepthPointToColorSpace(depthSpacePoint, depthImageValue);
            Console.WriteLine("SDK's color coordinates: {0} {1}", colorSpacePoint.X, colorSpacePoint.Y);

            // convert back to depth image
            Matrix depthPoint;
            double depthX, depthY;

            calibration.ColorImageToDepthImage(colorX, colorY, depthImage, out depthPoint, out depthX, out depthY);

            //// when converting many points, it may be faster to precompute and pass in the distortion table:
            //var colorFrameToCameraSapceTable = calibration.ComputeColorFrameToCameraSpaceTable();
            //calibration.ColorImageToDepthImage((int)colorX, (int)colorY, depthImage, colorFrameToCameraSapceTable, out depthPoint, out depthX, out dept

            Console.WriteLine("convert back to depth: {0} {1}", depthX, depthY);
        }
    }
```

Fig. 6 Mapping Coordinate

This code processes depth data coming from a Kinect sensor getting depth image coordinates and converting them from depth coordinates to color coordinates and back from color coordinates to real depth coordinates. It starts by trying to get a depth frame with e.FrameReference. which is to acquire the frame through AcquireFrame() and verify whether it is successful. If so, the frame data is copied into the specified memory location (depthImage. A pointer to a buffer containing the raw image data (IntPtr), formatted according to the spatial resolution of the Kinect's depth image. Next, given the depth image, the code chooses a point (100, 100)` to check its depth value, and determine whether or not it is a valid point (non-zero). It goes on to say if zero is passed then the enter depth and it returns from the code.

If the depth is a valid depth value, it is converted from millimeters to meters by dividing from 1000 to match the expected depth from Kinect SDK output units. Next, the calibration object uses its DepthImageToColorImage method to map the (100, 100) depth image coordinates to (colorx, colory) color image coordinates. An optional distortion lookup table can be passed to accelerate conversions. (The corresponding line is commented out in the code).

Next, a similar transformation is conducted using the Kinect SDK's CoordinateMapper to map the same depth image coordinates to color space, providing a basis for comparison. The depth image coordinates are then converted back to their original values using ColorImageToDepthImage and printed, helping to confirm the accuracy of the round-trip conversion process. This code enables precise calibration and point mapping between Kinect's depth and color image spaces, important for applications requiring synchronized depth and color data alignment.

Fig. 7 User Coordinate

It is necessary to pay attention to the Kinect positioning technique which has an important role so that 3D objects and models can match. Then you can add model gender recognition, so you can determine which traditional clothing appears automatically according to gender. This is done to get a better UI/UX

IV. Conclusion

The Microsoft Kinect's body tracking technology represents a paradigm shift in the way computers perceive and interact with humans. Its advancements in depth sensing, skeletal tracking, and machine learning have broadened the scope of real-time body tracking, making it more accessible, accurate, and versatile. This technology continues to inspire new applications and innovations across a multitude of sectors, fundamentally changing human-computer interaction and setting the stage for the next generation of interactive systems.

References

[1] Nijholt, "Virtual worlds: A new open access journal of virtual reality, augmented and mixed reality technologies, and their uses," *Virtual Worlds*, vol. 1, no. 1, pp. 18-19, Aug. 2022, doi:10.3390/virtualworlds1010002.

[2] J. Manjorin, "How eLearning practitioners can find value in augmented and virtual reality technology," *eLearn*, vol. 2017, no. 8, Aug. 2017, doi: 10.1145/3136555.3133321.

[3] K. Swargiary, "Augmented reality (AR) technology on student engagement: An experimental research study," *Qeios*, Nov. 2023, doi:10.32388/fnwgpu.
[4] J. Y. Yoon et al., "The effect of social media apps on shopping apps," *J. Bus. Res.*, vol. 148, pp. 23-32, Sep. 2022, doi:10.1016/j.jbusres.2022.04.021.
[5] S. Alatrash, S. Arnab, and K. Antlej, "Communicating engineering heritage through immersive technology: A VR framework for enhancing users' interpretation process in virtual immersive environments," *Comput. Educ.: X Reality*, vol. 3, p. 100040, Dec. 2023, doi: 10.1016/j.cexr.2023.100040.
[6] M.-A. Moinnereau, A. A. Oliveira, and T. H. Falk, "Instrumenting a virtual reality headset for at-home gamer experience monitoring and behavioural assessment," *Front. Virtual Reality*, vol. 3, Oct. 2022, doi:10.3389/frvir.2022.971054.
[7] A. Theodoropoulos et al., "Developing an interactive VR CAVE for immersive shared gaming experiences," *Virtual Worlds*, vol. 2, no. 2, pp. 162-181, May 2023, doi: 10.3390/virtualworlds2020010.
[8] R. Kirollos and W. Merchant, "Comparing cybersickness in virtual reality and mixed reality head-mounted displays," *Front. Virtual Reality*, vol. 4, Feb. 2023, doi: 10.3389/frvir.2023.1130864.
[9] P. Sinlapanuntakul, J. Korentsides, and B. S. Chaparro, "Exploring the user experience (UX) of a multi-window augmented reality environment," *Front. Virtual Reality*, vol. 4, Aug. 2023, doi:10.3389/frvir.2023.194019.
[10] M. C. Howard and M. M. Davis, "A meta-analysis of augmented reality programs for education and training," *Virtual Reality*, vol. 27, no. 4, pp. 2871-2894, Aug. 2023, doi: 10.1007/s10055-023-00844-6.
[11] L. Kerawalla et al., "'Making it real': Exploring the potential of augmented reality for teaching primary school science," *Virtual Reality*, vol. 10, no. 3-4, pp. 163-174, Nov. 2006, doi: 10.1007/s10055-006-0036-4.
[12] H. Salmi, H. Thuneberg, and M.-P. Vainikainen, "Making the invisible observable by augmented reality in informal science education context," *Int. J. Sci. Educ., Part B*, vol. 7, no. 3, pp. 253-268, Nov. 2016, doi: 10.1080/21548455.2016.1254358.
[13] Z. Du, J. Liu, and T. Wang, "Augmented reality marketing: A systematic literature review and an agenda for future inquiry," *Front. Psychol.*, vol. 13, Jun. 2022, doi: 10.3389/fpsyg.2022.925963.
[14] W. Handayani, R. Afyenni, and A. Syukri, "Optimalisasi promosi Nagari berbasis website," in *Proc. Appl. Bus. Eng. Conf.*, vol. 10, 2022.
[15] Y. Sonatha et al., "Synergizing AHP and SMART: An integrated decision support system for best employee selection," in *Proc. 11th Int. Appl. Bus. Eng. Conf. (ABEC)*, Bengkalis, Indonesia, Sep. 2023, Feb. 2024.