HOCHSCHULE ANHALT University of Applied Sciences

DIRECTED RESEARCH STUDIES (DRS)

Prof. Johannes Tuemler

SS 2022

Evaluation of Azure Kinect DK, Demo Application and comparison with Kinect V2

Jerin Puthenveettil Reji (5021347)

Mrika Govori (4070852)

Paul Raju Chemmannoor (4071236)

Submission date: 30.10.2022

Declaration

I, JERIN PUTHENVEETTIL REJI here by declare that this project is the record of bona fide research carried out by us under the supervision of Prof. Dr. Johannes Tuemler, department of Biomedical Engineering Anhalt Univeristy of Applied Sciences Koethen. We further declare that this project has not been previously formed the basis for the award of the credits or other similar title of recognition.

Koethen **Jerin Puthenveettil Reji**

30th October 2022 **5021347**

Declaration

I, PAUL RAJU CHEMMANNOOR here by declare that this project is the record of bona fide research carried out by us under the supervision of Prof. Dr. Johannes Tuemler, department of Biomedical Engineering Anhalt Univeristy of Applied Sciences Koethen. We further declare that this project has not been previously formed the basis for the award of the credits or other similar title of recognition.

Koethen **Paul Raju Chemmannoor**

30th October 2022 **4071236**

Declaration

I, MRIKA GOVORI here by declare that this project is the record of bona fide research carried out by us under the supervision of Prof. Dr. Johannes Tuemler, department of Biomedical Engineering Anhalt Univeristy of Applied Sciences Koethen. We further declare that this project has not been previously formed the basis for the award of the credits or other similar title of recognition.

Koethen **Mrika Govori**

30th October 2022 **4070852**

Table of Contents

List of Figures

List of Tables

Abstract

This project aims in the assessments of the new Azure Kinect depth and colour data sensing by providing a benchmark study with a laser scanner and Kinect v2 camera system for comparison. We use a laser scanner to obtain the ground truth for the absolute location of the target geometry.

This work shows a depth, infrared data and pixel resolution comparison between Azure Kinect and Kinect V2 laser scanners using RGB and infrared camera. The comparison is made using a planar board with square shape. Accuracy and precision tests are done for different ranges between each sensor and the planar board. The evaluation was performed at a range of 1.0 to 5.0 m with 0.5 m intervals. Besides this, we have done two applications to test Azure Kinect in order to navigate with a cursor using hand tracking and estimate the joint angle of the user.

1. Introduction

Image Recognition and Motion Sensing have been the research focus in the field of computer graphics over the years. The origins of the Kinect started around 2005, at a point where technology vendors were starting to develop depth-sensing cameras. Depth sensing is important in various applications, such as augmented and virtual reality, human–computer interaction, robotic manipulation, and navigation.

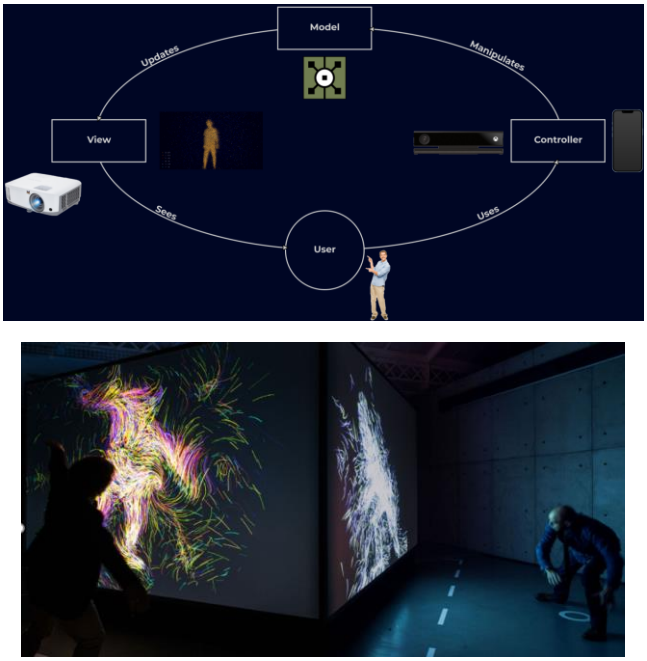


Fig.1: Motion Control View with Kinect **[1]**.

In 2012 and 2013, Microsoft released updated versions of their Kinect sensor as Kinect for Windows and Kinect for Xbox One (a.k.a. Kinect v2), respectively. This device was used mostly for gaming but also for motion tracking! In 2019, Microsoft released their newest version of the sensor as the Azure Kinect DK (Development Kit), which is marketed to enterprise users in areas such as retail, manufacturing, robotics, and healthcare **[2]**.

Fig.2. Kinect V2, Azure Kinect - V4 **[3].**

This project focuses on comparing the depth and color data between the two latest generations of the Kinect sensor. We first review prior work on the accuracy evaluation of the Kinect v2 and Azure Kinect. Next, we describe the methodology for our accuracy assessment using two different methods. We then present the results for the two Kinect systems using laser scanner data as ground truth and a side-by-side comparison across the field of view of two sensors. Finally, we develop 2 application demo's virtual mouse and joint angle detection using Azure Kinect DK.

2. Azure Kinect DK

Azure Kinect DK is a developer kit with advanced AI sensors that provide sophisticated computer vision and speech models. Kinect contains a depth sensor, spatial microphone array with a video camera, and orientation sensor as an all in-one small device with multiple modes, options, and software development kits (SDKs) **[4]**.

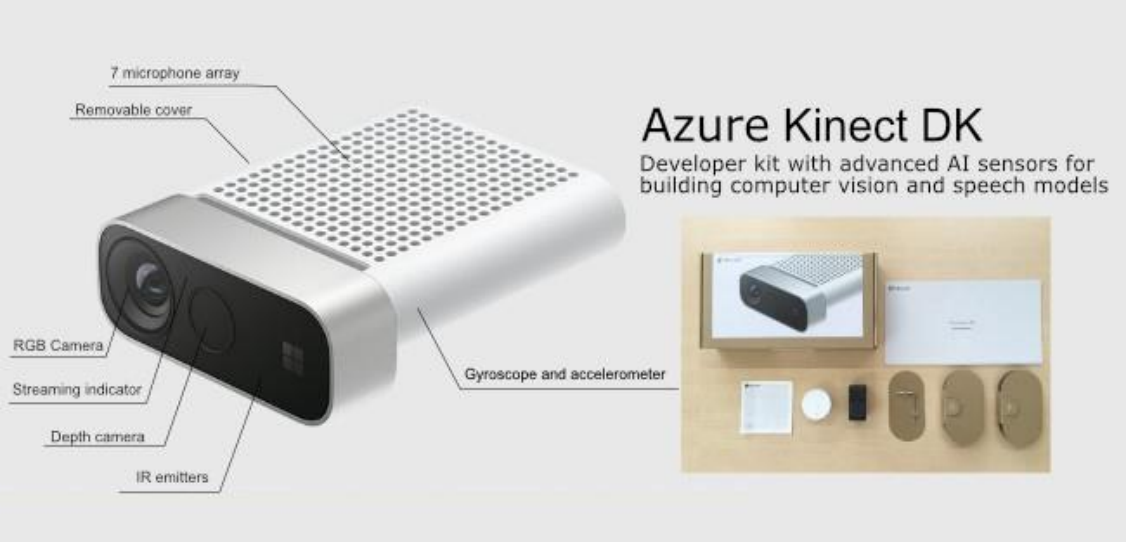


Fig.3: Azure Kinect **[5].**

The service offers a range of APIs and SDKs like Sensor SDK, Body Tracking SDK, Speech SDK, and Computer Vision service APIs to help you innovate faster and leverage machine learning tools for transformative business solutions. Azure Kinect DK is a powerful feature-packed developer kit for small to large enterprises.

Inside the Azure Kinect DK device

- A 7-microphone array designated for sound capture and far-field speech
- 1-MP depth sensor with both narrow and wide FOV options
- Gyroscope (IMU) and accelerometer for spatial tracking and sensor orientation
- 12-MP camera (RGB) for color stream aligned with the depth stream
- Sync pins for easy synchronization of sensor streams

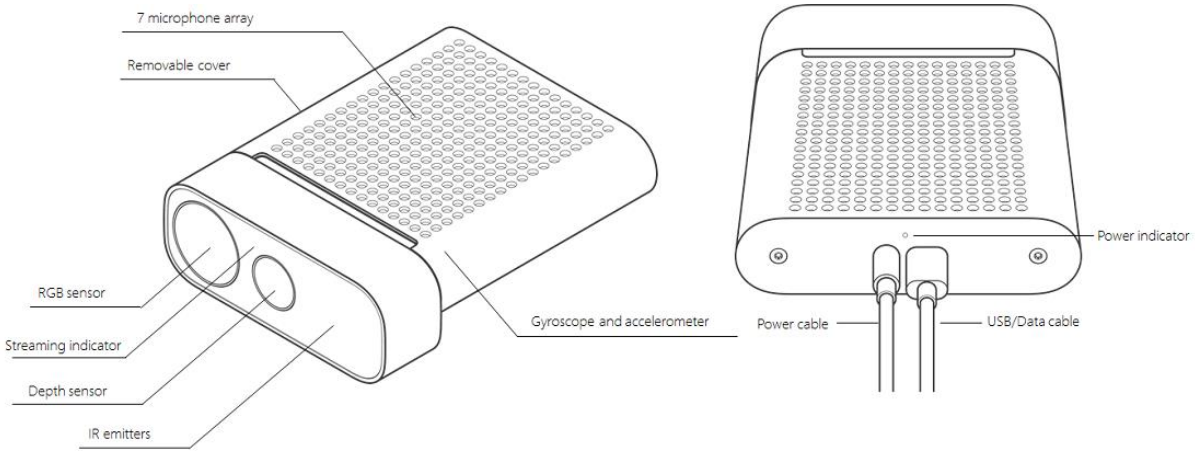


Fig.4: Azure Kinect Device Features **[6].**

2.1 How to set up Azure Kinect?

1. Connect the USB power adapter to the other end of the cable, and then plug the adapter into a power outlet.
2. Connect the USB data cable into your device, and then to a USB 3.0 port on your PC.
3. Verify that the power indicator LED (next to the USB cable) is solid white.
4. Install the sensor and body tracking SDK on the device (https://github.com/microsoft/Azure-Kinect-Sensor-SDK). In our case the the body tracking SDK didn't work, we installed NVIDIA version 472.47 and the problem was solved!

Verify that the device streams data! Once you open the camera, in the Azure Kinect Viewer you will be able to see the Colour (RGB), Infrared Camera (IR) and Depth Camera. The view mode can be 2D or 3D (Point Cloud View). A point cloud is a set of data points in space.

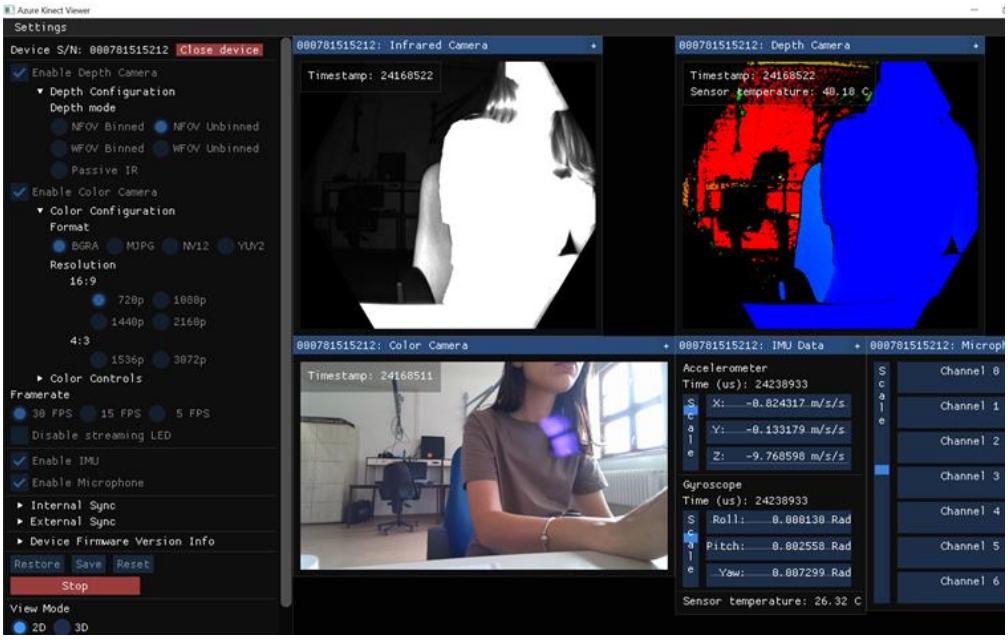


Fig.5: Azure Kinect Camera streaming – 2D.

You can choose different depth modes such as:
1. Narrow Field of View – Binned
2. Narrow Field of View – Unbinned
3. Wide Field of View – Binned
4. Wide Field of View – Unbinned.

Binning is the process of combining pixels together in the camera to make 'super pixels', which can control the sensitivity and resolution of the camera.

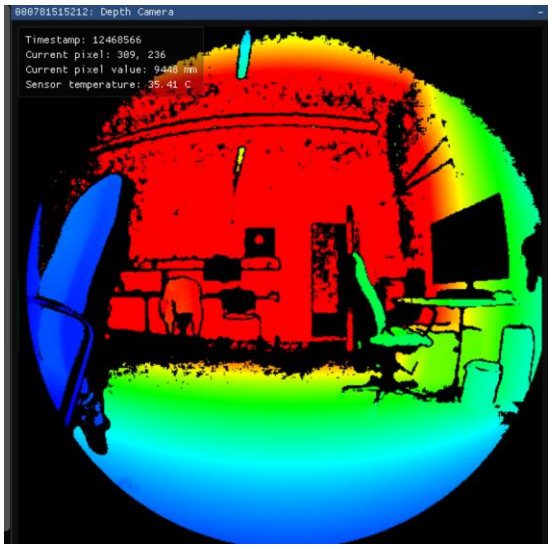


Fig.6: Depth Camera – Wide FOV Binned.

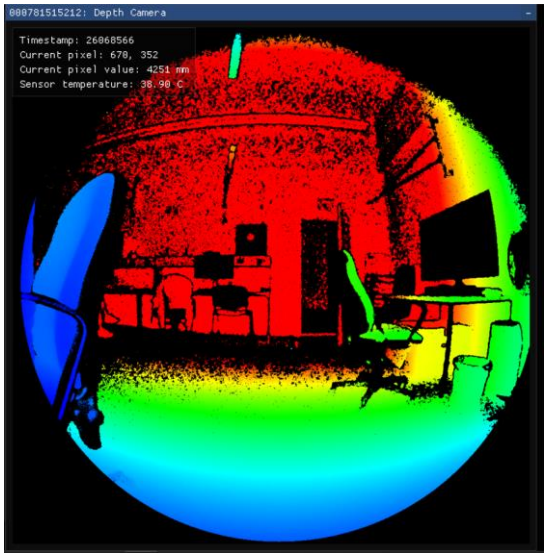


Fig.7: Depth Camera – Wide FOV Unbinned.

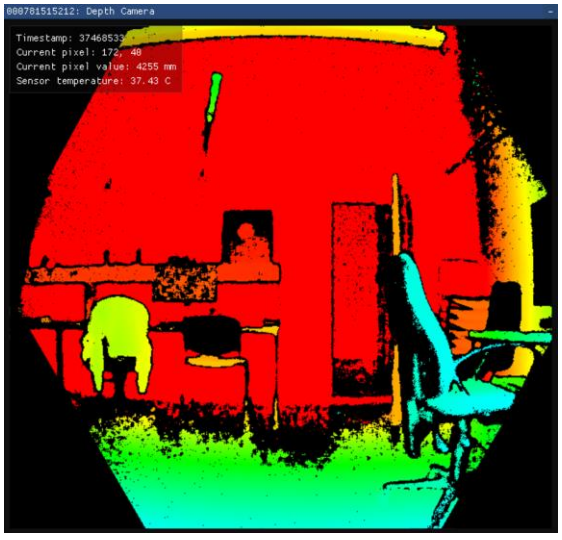


Fig.8: Depth Camera – Narrow FOV Unbinned.

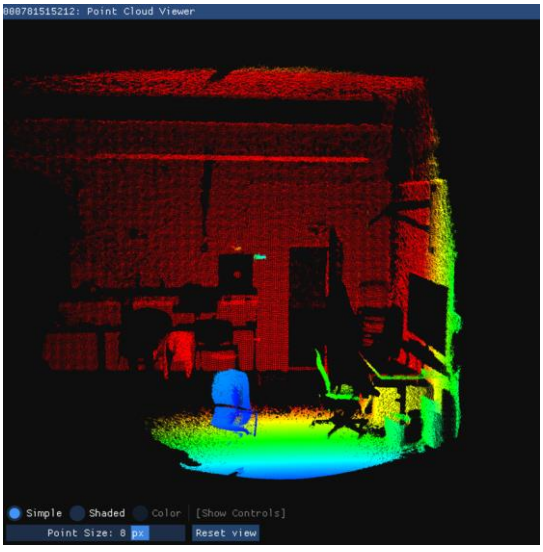


Fig.9: Camera streaming – 3D.

2.2 Time of Flight Methodology

Depth camera measures distance by actively illuminating an object with a modulated light source and a sensor that is sensitive to the source's wavelength for capturing reflected light.

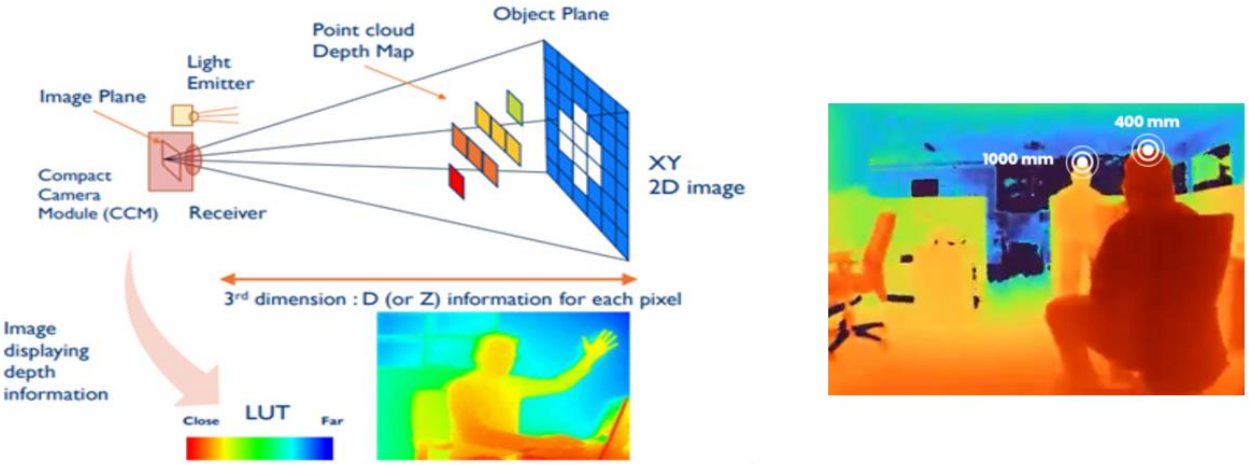


Fig.10: Depth Camera technology **[7]**.

As Azure Kinect uses this technology, we can see by default in the Azure Kinect Viewer the measured distance of the current pixel value in millimeters, the resolution in pixel and also the temperature! We wanted to check the distance from Azure Kinect to the door as in the attached photo below, and the distance was 4251mm.

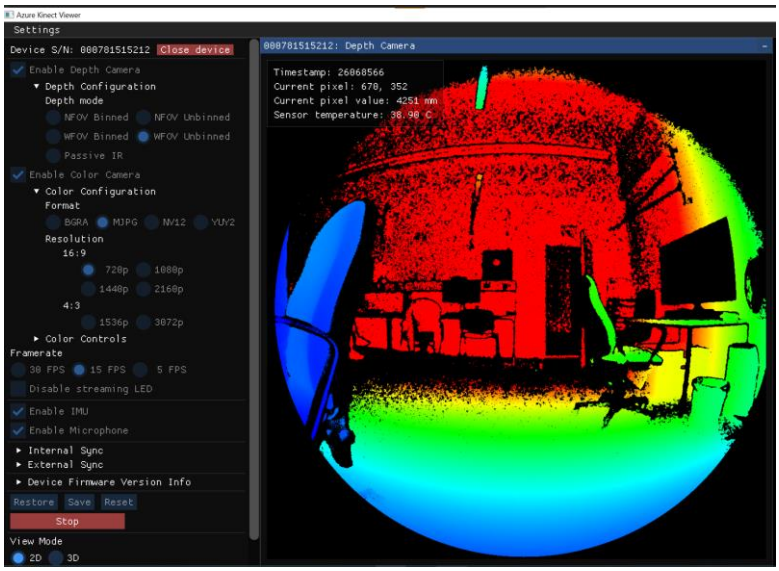


Fig.11: Azure Kinect Depth Camera.

2.3 Azure Kinect Sensor and Body Tracking SDK

Sensor SDK provides cross-platform low-level access for Azure Kinect device configuration and hardware sensors streams.

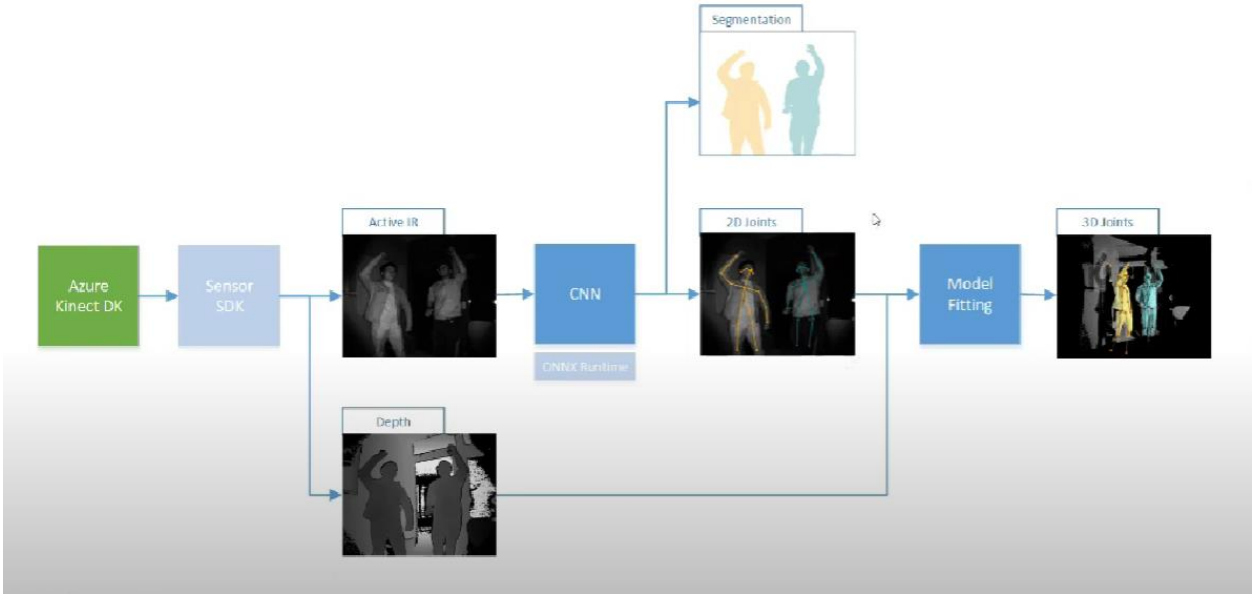


Fig.12: Architecture **[8].**

Body Tracking SDK uses a body tracker object to process Azure Kinect DK captures and generates body tracking results. It also maintains global status of the tracker, processing queues and the output queue.

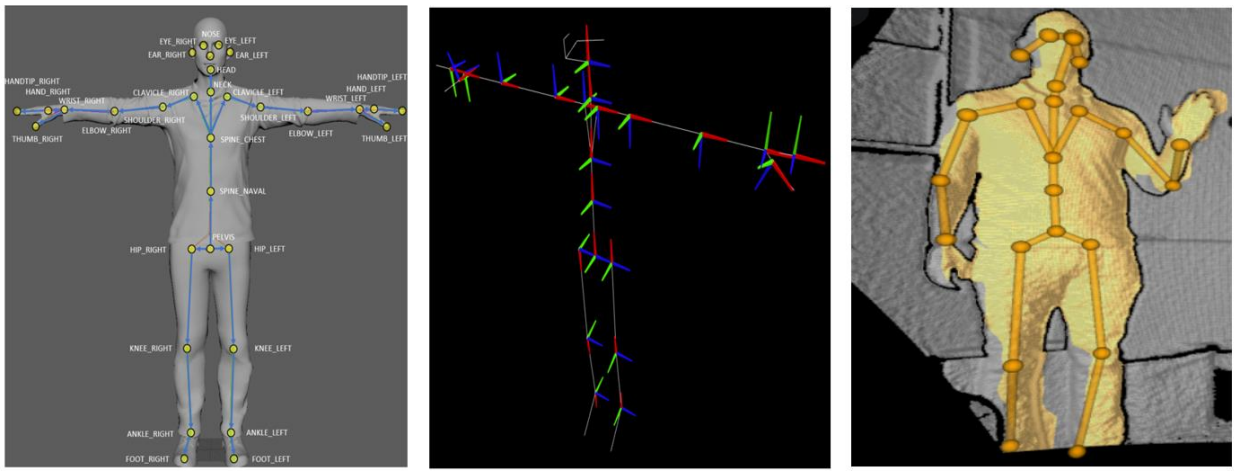


Fig.13: Joint hirearchy and coordinations (x-red,y-green,z-blue).

3. Kinect V2

The Kinect V2 is a 3D sensor produced by Microsoft, it is composed by a RGB camera with resolution of 1920×1080 pixels, an infrared camera with resolution of 512×424 pixels and an infrared emitter. Based on the Time of Flight (ToF) technology and, in particular, on the intensity modulation technique, the on board electronics evaluate the distance between the sensor and each point of the scene viewed by the camera. The accuracy in the determination of such distances is a function of the distance itself and may reach values of about 1.5 mm when the point are close (about 1 m) to the sensor.

Fig.14: Kinect Windows V2 **[9].**

Developed by Microsoft as a game device, since its first appearance on the market, the Kinect sensor has aroused the interest of researchers because of its high potentiality, when used as a measuring instrument, combined with its very low cost. V2 was designed as an XBOX camera for body tracking experiences and it was meant to be stationed in one place.

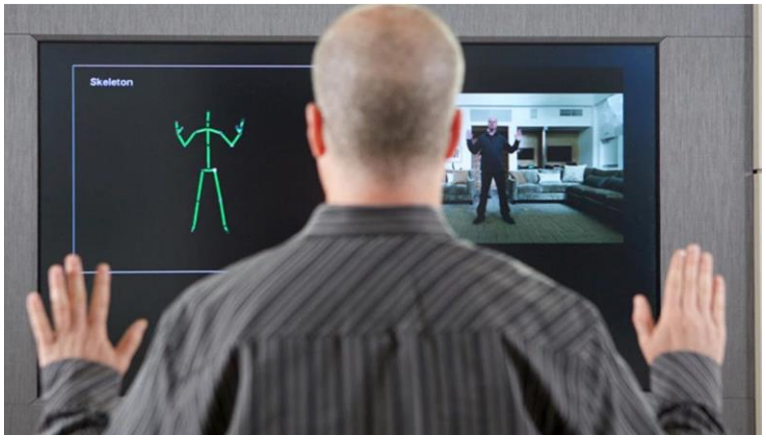


Fig.15: Kinect V2 Application **[10].**

3.1 How to set up Kinect V2?

- Connect the Kinect v2 to the USB 3.0 port of your computer and to the power supply.
- Download and install the **Kinect Configuration Verifier**. Run the tool. It should be able to detect the Kinect v2 and be able to read the color and depth streams. Else please see its error to find what configuration of your computer is causing a problem.
- Download and install the **Kinect for Windows SDK 2.0**.
- Run the **Kinect Studio v2.0** and check if it can display color and depth streams from your device. Note that you need to manually click the **Connect** option in this tool to enable use of the Kinect.
- Open the **SDK Browser v2.0 (Kinect for Windows)** tool. There you can see C++ and C# code examples, install those Visual Studio solutions, compile and run them.

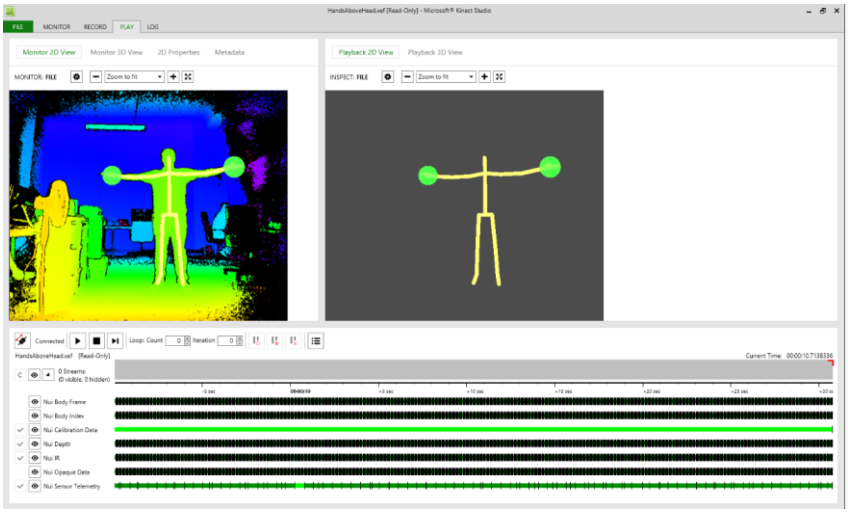


Fig.16: Kinect Studio **[11].**

Once you open the SDK Browser V2.0, you can select and run files in C++/C#:

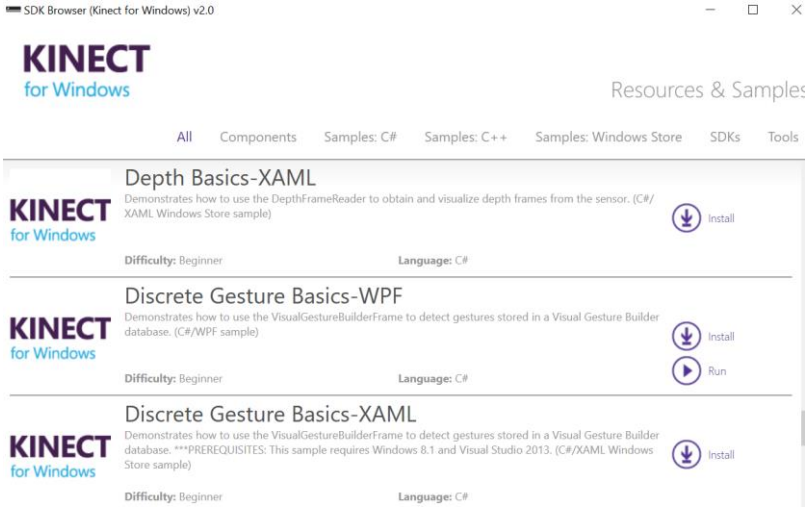


Fig.17: SDK Browser V2.0.

4. Comparison between Azure Kinect and Kinect V2

Azure Kinect, the latest generation of Kinect DKs uses more sophisticated sensors. Microsoft shifted from Kinect known as a gaming peripheral to a stand alone 3D camera leaving a door wide open to the new emerging technology of volumetric capture.

Kinect now has 6 times better resolution, is twice as light and has better audio. It can be easily considered the best off the shelf sensor on the market for volumetric video.

Function	Azure Kinect	Kinect V2
Audio	Array 7 Microphones	Array 4 Microphones
Motion Sensor	3-Axis Accelerometer and Gyrometer	3-Axis Accelerometer
RGB Camera	3840 x 2160px @30fps	1920 x 1080px @30fps
Depth Camera	640 x 576px @30fps 512 x 512px @30fps 1024 x 1024px @15fps	512 x 424px @30fps
Skeleton	28 Joints	25 Joints
Maximum Bodies Tracked	15	6
Programming Languages	C/C++/C#	C++/C#
Operating System	Windows Linux Ubuntu	Windows

5. Experimental Comparison of Azure Kinect and Kinect V2

5.1 Materials used in the Experiment

The depth and pixel accuracy of the Microsoft Azure Kinect and Kinect v2 were tested against ground truth data given by a Bosch PLR15 laser. More information about the sensors and their settings is provided below.

- **Azure Kinect**

The Azure Kinect DK comes with a 12-megapixel color camera (4096 3072 px) and a 1-megapixel ToF depth sensor (1024 1024 px). The depth camera's maximum FOV is 120° 120° in WFOV mode and 75° 65° in NFOV mode. In WFOV mode, the maximum output resolution is 1024 1024 px unbinned and 512 512 px binned. The operational range in WFOV mode is 0.25 to 2.21 m (0.25 to 2.88 m binned), whereas the range in NFOV mode is 0.50 to 3.86 m. (0.50 to 5.46 m binned). In NFOV mode, the maximum output resolution is 640 576 px unbinned and 320 288 px binned. In this experiment, the Azure NFOV unbinned and WFOV binned modes are employed to compare Kinect V2 FOV.

Fig 18: Azure Kinect

- **Kinect V2**

The Kinect V2 hardware utilized in this investigation was the Xbox and Windows versions of the device. The Kinect V2 has a high-definition color camera (1920 x 1080 pixels) and a ToF depth sensor with great dynamic range that transmits data at 30 frames per second (FPS). A field of view (FOV) of 70°x 60° and an operational range of 0.5 to 4.5 m are included in the Kinect V2 depth sensor parameters.

Fig 19: Kinect V2

- **Bosch PLR 15 Laser Rangefinder**

Bosch PLR 15 laser was used to obtain the reference ground truth or benchmark for accuracy evaluation. For quickly measuring lengths and distances, Bosch PLR 15 Laser Rangefinder technology offers high levels of precision every time. Bosch PLR 15 has a range of 0.15 to 15 meters. Regardless of the distance being measured, it delivers precision of +/- 3 millimeters up to ranges of 15 meters. All of our measurements were made indoors, in a range less than 10 meters.

Fig 20: Bosch PLR15 rangefinder

- **Planar board and rectangular white chart**

In this work, a flat or planar surface was assessed (Figure 9). The board, which was made of engineered wood and measured 20 x 250 cm (7.87 x 98.42 in), was flat and covered with a white non-reflective cloth material. A rectangular chart of around 70 x 50 cm (27.55 x 19.68 in) served as the plane's representation. Onto this flat board was adhered the rectangular planar chart. The centre of the board had a square drawn with a black marker that was approximately 17.8 17.8 cm (7x7 in) in size.

Fig 21: Flat planar board with square marked in the centre placed 2m distance

5.2 Experimental Setup and Data Collection

The setting for testing was a 6.5x4.8 m (21.3 x 15.7 ft) indoor area. LED ceiling fixtures lighted the room. The Kinect V2 and Azure Kinect were put vertically on the same surface (Figure 22) and positioned at one end of the room. The midpoint of each Kinect device was situated at an elevation of roughly 1.315 m (51.77 in) above the ground level. The cameras were vertically positioned so that their frontal surfaces were in the same plane.

The cameras were directed towards the surface. When taking each measurement, the Bosch PLR 15 laser was set directly above the Kinect devices, with the laser's height being about 1.35 m above the ground plane.

In this study, a flat surface with a square marked in the centre was analysed (Figure 10). The planar surface was placed on a flat board, with the centre approximately 1.315 m (51.77 in) from the floor level and vertically aligned.

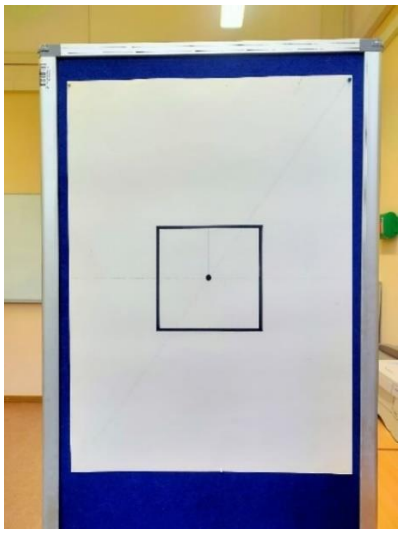


Fig 22: Square and midpoint marked in the centre of planar board used as test object

Data Collection

In this collection of tests, we concentrated particularly on the precision of the sensors under consideration. As indicated in Figure 3, they were installed on a construction surface facing planar board. To avoid interferential noise, we measured depth and pixel data at 9 positions (1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0 m) and changed the sensors at the top of the structure. so that only one sensor faced the planar board throughout measurement. These nine places were chosen in order to reliably record accurate data for each sensor's estimated min, mid, and max ranges.

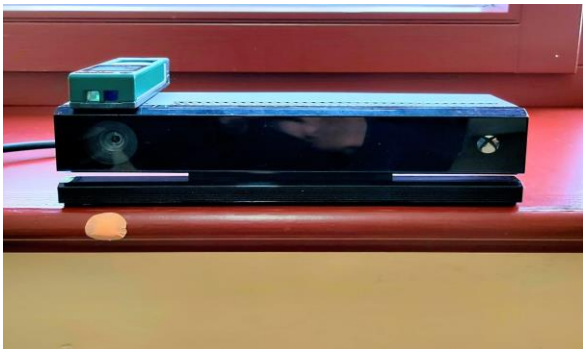


Fig 23: Kinect V2 and Bosch PLR15 laser positioned directly above for experiment

Fig 24: Azure Kinect and Bosch PLR15 laser positioned directly above for experiment

The object was moved throughout the field of vision to test the accuracy of the depth acquisition by the two cameras. As illustrated in Figure 12, the floor in front of the cameras was divided into a grid. The sites were marked with stickers after being measured from the place on the floor right underneath the cameras with measuring tape.

For all scans, the planar object was kept at the same height and properly orientated so that the board midpoint coincided with the midpoint of the Kinect cameras.

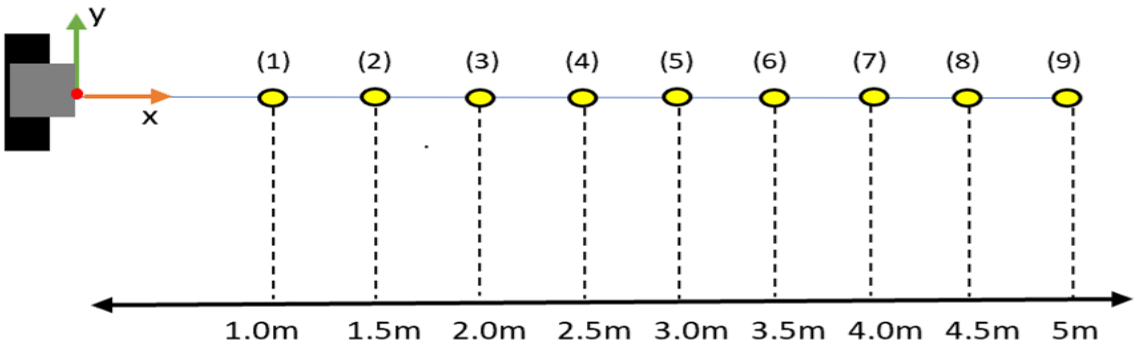


Fig 25: Grid arrangement of the target object locations for testing. Kinect cameras and the Bosch laser rangefinder are located at the origin of the coordinate system.

Data was collected from each Kinect device individually using different software such as the Kinect for Windows SDK (for Kinect V2) and the Azure Kinect viewer v1.4.1 (for Azure Kinect), which allows for the gathering of raw Kinect data (i.e., infrared, colour and depth images). The USB 3 interface was used to connect both devices to the same laptop.

After being switched on, the Kinect cameras do not provide a consistent output value. It must be warmed up for some time before the output can be stabilized. The Kinect cameras were turned on around 30 minutes before the measurement to allow for pre-heating and a more steady accuracy [13].

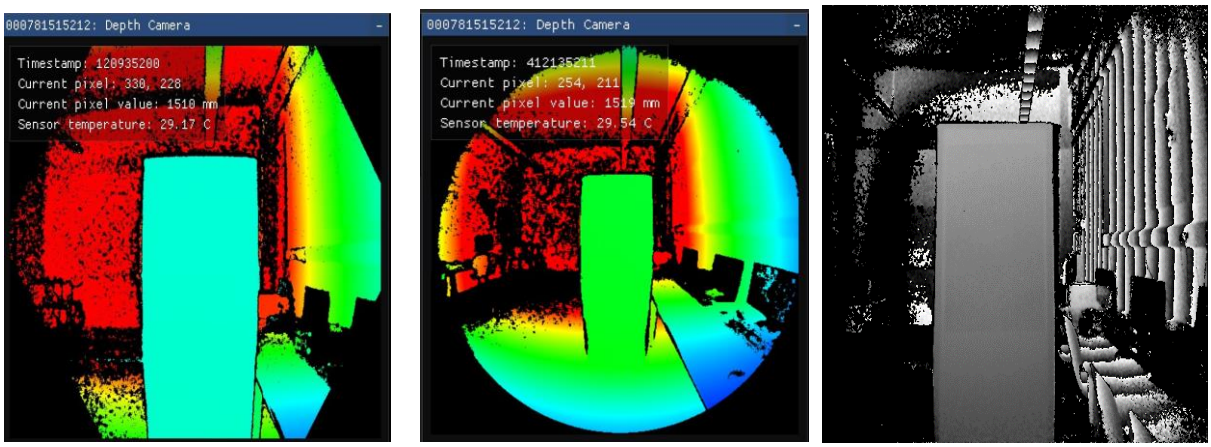
To match the field of view of the Kinect V2, the Azure Kinect was configured to NFOV unbinned mode and WFOV binned mode. Concurrently, the Bosch laser rangefinder performed a measurement to determine the reference ground truth data. With each scan, infrared, depth, and colour data were collected. This procedure was done nine times at key sites ranging from 1.0 m to 5.0 m.

The Kinect V2 infrared data were processed in Matlab for the pixel value and exported in .bip format for post-processing.

5.3 Depth Accuracy Assessment of Azure Kinect

In this study, we evaluate the accuracy of the depth camera included in Azure Kinect. The constraints of obtaining depth data from Kinect V2 limited the depth accuracy rating of the Kinect V2 sensor. Figure 13 depicts the raw image from the Kinect V2 (c)

In the research in the following sections, the planar surface is utilized to acquire depth data to evaluate the performance of the depth camera incorporated in Azure Kinect and the screen is perpendicular to Azure Kinect. The Bosch laser is used to measure nine distances between the Azure Kinect and the planar surface. The depth accuracy can be verified utilizing geometrical relationships using these 9 distances and depth data.



(a) (b) (c)

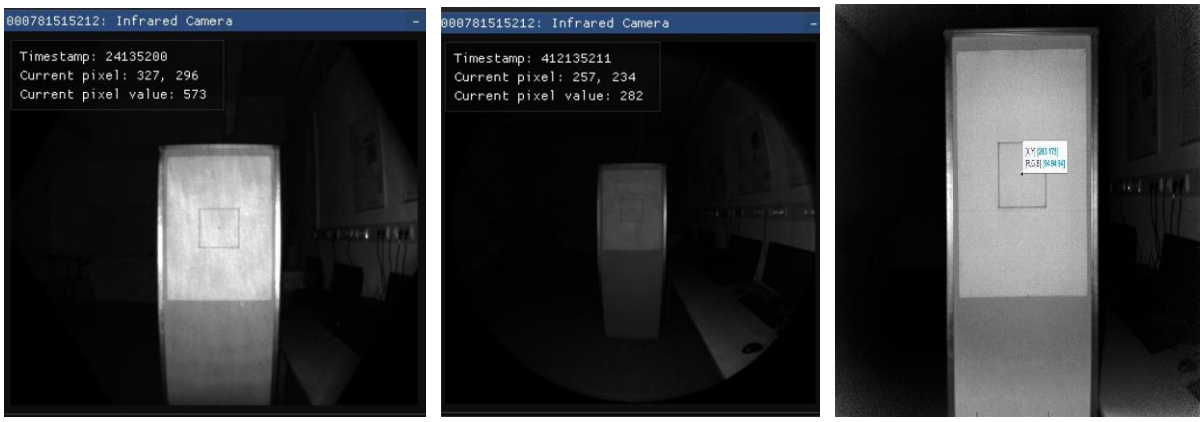
Fig 26: Depth image extracted from experiment at 1.5 m position of (a) Azure NFOV unbinned (b) Azure WFOV binned and (c) Kinect V2

As can be observed in Fig 13, there is a significant difference between the first two modes of the Azure Kinect and the previous Kinect V2. The Kinect V2 uses the whole rectangular area of the pixel matrix, whereas the Azure sensor uses hex for narrow mode (NFOV) and circular for wide mode (WFOV). Furthermore, the Azure Kinect has a broader field of view than the Kinect V2.

5.4 Pixel Assessment of Azure Kinect and Kinect V2

The same planar surface with a square drawn with a point in the center is made to demonstrate the pixel distribution to show how the pixel data differs in space. The pixel distribution from both Kinect V2 and Azure Kinect is studied here. The plane surface is perpendicular to the Kinect cameras. The planar surface is positioned at a distance of 1.0 m to 5.0 m from the Kinect sensor, with 0.5 m intervals.

Azure Kinect SDK (a Software Development Kit) output a matrix of pixel values inside infrared at each point. In the case of Kinect V2, the infrared picture is collected from the Kinect for Windows SDK and fed into Matlab for pixel data processing and display. Then, for a given depth distance, compare the pixel distribution of Azure in NFOV unbinned, WFOV binned, and Kinect V2. The comparison clearly demonstrates the pixel distribution in the various sensors and Azure settings.



(a) (b) (c)

Fig 27: Infrared image with pixel data at 1.5 m position of (a) Azure NFOV unbinned (b) Azure WFOV binned and (c) Kinect V2

5.5 Data Reduction

Depth accuracy or accuracy distribution

Depth accuracy is defined in this project as the difference between the true depth value and the measured depth values corresponding to a planar surface in front of an Azure Kinect.

An Azure Kinect portraying a planar surface captures the depth values inside the square surface's center. Depth accuracy is defined as the difference between the Azure depth value and the true depth value from the laser. The depth accuracy may be calculated by indicating the depth values measured by Azure Kinect as A_d and the true distance between the planar surface and Azure Kinect as D in the following equation [14].

Depth Accuracy = $D_a = D-A_d$

To compute the depth accuracy in the experiment, a planar surface is positioned perpendicular to Azure Kinect at a certain distance, and the depth value representing the planar surface is retrieved using the Azure Kinect viewer SDK. Meanwhile, a Bosch laser distance meter measures the distance between Azure Kinect and the planar surface, which is to be the true value here.

5.6 Results and Discussion

- **Depth Accuracy**

For each board position, the random depth distance from Azure NFOV unbinned and Azure WFOV binned is calculated (Figures 11 and 12). Laser data is used as the basis for comparing depth data.

Tape measurement (mm)	Laser distance (D) (mm)	Azure WFOV binned depth (Ad) (mm)	Depth Accuracy wfovb (Da= D-Ad)	Azure depth NFOV unbinned (Ad) (mm)	Depth Accuracy nfovub (Da= D-Ad)
1000	1010	1018	8	1001	9
1500	1511	1519	8	1510	1
2000	2007	2012	5	2009	2
2500	2503	2506	3	2506	3
3000	3005	3010	5	3012	7
3500	3507	3500	7	3500	7
4000	4006	4010	4	4004	2
4500	4503	4508	5	4501	2
5000	5004	5010	6	5011	7
			Avg Da: 5.67 mm		Avg Da: 4.45 mm

Table 1: Comparison between the Azure WFOV binned depth and Azure NFOV unbinned depth and it's depth accuracy in different laser distances

According to Table 1, there isn't much of a difference in depth between Azure WFOV binned and NFOV unbinned modes at different laser distances. It represents an appropriate depth distribution on several modes of Azure and shows a consistent trend in the graph in figure 14.

By summing the azure Kinect measurement accuracy at all important places in NFOV unbinned and WFOV binned mode, we find that the depth accuracy averages 4.45 mm and 5.6 mm, respectively. The NFOV mode provides a more precise depth value to the reference truth data.

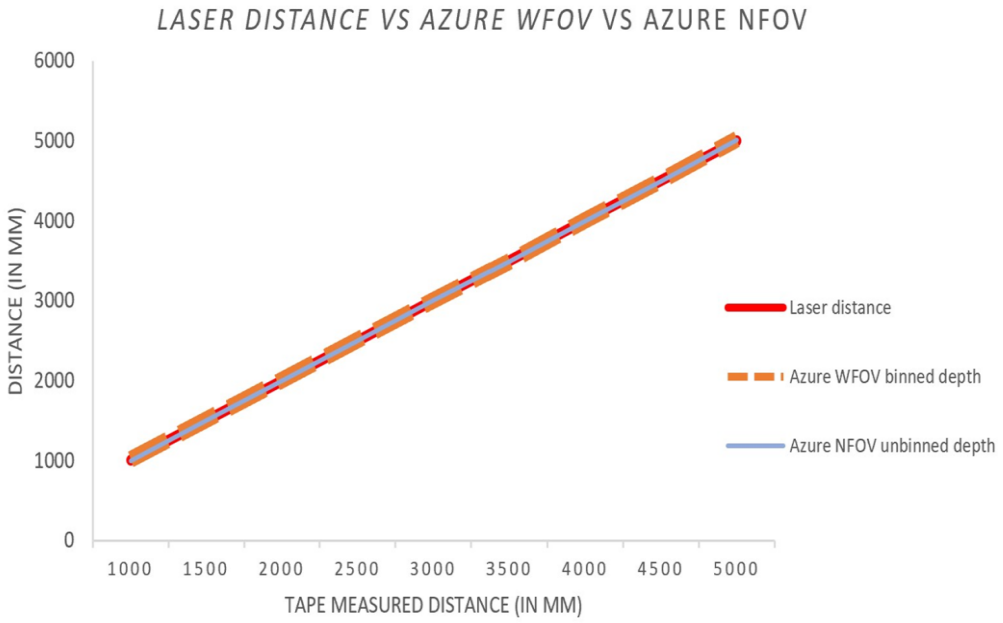


Fig 28: Depth value representation of laser, Azure WFOV and Azure NFOV depth vs tape measured distance

Pixel comparison

A pixel is the smallest unit of a digital image or graphic that can be displayed and represented on a digital display device. A digital image is a 2D array of pixels. Each pixel is characterized by its (x, y) coordinates and its value. Digital images are characterized by matrix size, pixel depth and resolution. The matrix size is determined from the number of the columns (m) and the number of rows (n) of the image matrix (m × n).

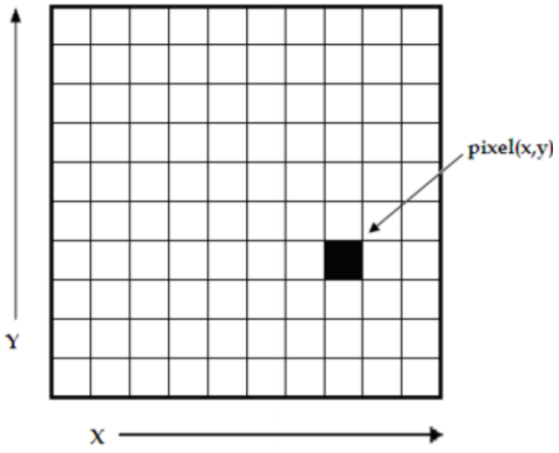


Fig.29: Pixel (x,y) **[12].**

We wanted to test the pixel difference for the same distance between Azure Kinect Narrow and Wide FOV with Kinect V2.

Based on the results of the experiments, Azure Kinect NFOV, WFOV mode, and Kinect V2 pixel data have different pixel values if the object is positioned inside the same depth position. Because they have various depth camera pixel resolutions, they have distinct pixel values in the same point in the image.

	Azure Kinect NFOV		Azure Kinect WFOV		Kinect V2	
Laser Distance (mm)	X	Y	X	Y	X	Y
1010	341	254	237	233	221	199
1511	327	296	257	234	263	175
2007	324	298	257	239	244	191
2503	319	302	257	241	243	192
3005	324	304	254	242	242	192
3507	331	305	260	234	247	193
4006	333	307	262	238	247	196
4503	333	307	274	235	246	211
5004	334	308	268	176	247	218

Table 2: Pixel data of NFOV, WFOV and Kinect V2 with laser distance

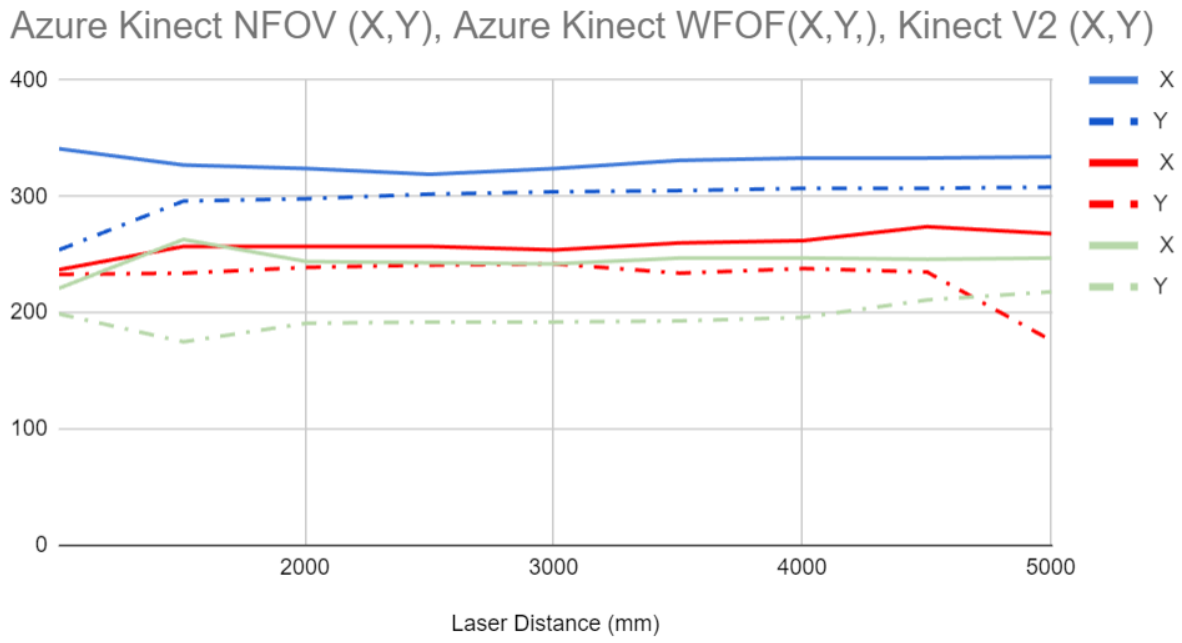


Fig 30: Pixel distribution chart of Azure Kinect NFOV, WFOV and Kinect V2

In accordance with the table above, the wide field of view (WFOV) binned mode of the Azure camera is comparable to the FOV of the Kinect V2. When compared to the other two, NFOV unbinned has higher pixel values in the centre of the square on planar board.

From this experiment and result we got, we can conclude that the Kinect V2 pixel data (pixels for X, Y axis) are similar with Azure Kinect Wide FOV

Discussion

As our data show, the Azure Kinect NFOV unbinned mode has higher average depth accuracy (4.45 mm) than the WFOV binned mode (5.67 mm). However, the depth data for Kinect V2 and two modes of Azure Kinect can be observed to be almost similar at distances ranging from 2.0 to 3.5 m from the camera. For all distances from the planar board, both modes of Azure maintain a same depth value in relation to the laser depth, with a difference of less than 10 mm.

In terms of pixel spatial data, our analysis shows that the Azure Kinect WFOV binned mode has a similar pixel point cloud range to the Kinect V2. At distances of 3 m and more, the Kinect Azure WFOV binned mode overestimated pixel values compared to the distances below 3.0 m.

The Azure Kinect NFOV unbinned mode displays a considerable disparity in pixel values at the closest distance of 1.0 m when compared to distances 1.5 to 5.0. The Azure Kinect NFOV demonstrates pretty consistent change as it moves away from the planar board (3.5 to 5.0)

In the instance of Kinect V2 pixel spatial data, indicating that has a similar range for pixel point cloud to each depth distance of Kinect V2. The Kinect V2 has reduced pixel distribution accuracy at distances of 4 m and more.

6. Demo application by Azure Kinect

6.1 Demo Application 1: Cursor using Hand tracking

The goal behind this application is to track the movement of the hand using Microsoft Azure Kinect for the purpose to navigate cursor on the computer screen. For achieving this goal, we generated a script in using Python programming language. The code was then developed, tested and edited using PyCharm (IDE). The idea behind this application was propelled by recent development in the field of gaming, and sign language detection by accurately tracking the position of user's hand. Technology gave us the ability to transfer any component of the real world into digital form. For the purpose of this motion tracking the device must be able to sense the changes in real time and offers the ability to control device.

Here we use the Azure Kinect's RGB and Depth camera to detect the hand gestures and use this to navigate the cursor on a screen. For achieving this purpose, we are using Mediapipe library in python. Mediapipe is a cross platform framework to develop custom ML solutions for real and streaming media. Mediapipe provides many solutions and some of them include Hand detection, Iris detection, Face mesh, Object detection. The other library we used in this application is open-source computer vision library (OpenCV), it is a library of programming functions for real time computer vision. It consists of C++, C, Python interfaces and supports Windows, Linux, IOS, and Mac OS. And the last library we used in this application is PyAutoGUI, this library allows python script to control cursor function.

As the technology is progressing day by day the need for remote and motion-controlled devices are increasing and this is where we see the future of this application. The application we created here can be used in gaming and controlling of medical devices using hand gestures in future, and enables the user to control and monitor the devices effortlessly in future.

Here in this script, firstly we need to importing necessary packages required for creating this application. In the first few lines of code, we are capturing the video source from Azure Kinect and read this frame. Next, we extract the land marks from Mediapipe and from this here we are using land mark number 8 which is the tip of index finger and next we use landmark number 4 which is the tip of thumb finger. Here we use tip of index finger for moving the cursor on the screen and the tip of thumb finger is used to produce a click function on the screen. To produce a click, function here we first draw a circle around the landmarks number 8 and 4 using lines of code. Then we code to program in such a way that once the distance between circles that around the selected landmarks get closer, we get a click function on the screen.

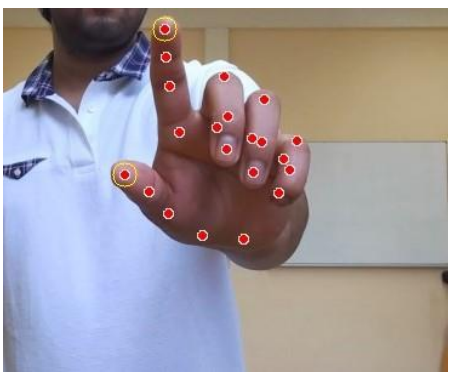


Fig 31: Hand detection using Azure

The image above shows the circle drawn around the index and the thumb finger, in this program we used a circle radius of 10 pixels.

Test setup: The testing of our demo application 1 was done using Microsoft azure. first, we have download and install Azure Kinect viewer v1.4.1 software as a pre requisite, here we used a Windows 10 operating system with 16 GB ram and Nvidia 4GB 3050Ti graphics card for the test. The test was done by keeping Azure Kinect at a height of 1.315 meters from the floor and the test was conducted for various distances from the Azure Kinect (30cm, 60 cm, 1m, 1.6m).

Methodology: First the power supply cable is connected to the Azure Kinect, then the USB type C data cable is connected to Azure and the operating system and made sure all the connections are stable next the power supply is turned ON and the Azure Kinect is kept 30 minutes for preheating. Once the pre heating is complete the program file is opened and run button is pressed, this opens up azure Kinect and we will be able to see our application.

Test Results: After the completion of the test, we were able to find out that the application was working efficiently at a distance of 60cm from the Azure Kinect. At this distance the hand was detected, the user was able to move the cursor on the screen without any difficulties. The cursor was stable (ignoring slight human movements) and the click function also worked perfectly.

(a) (b)

Fig 32: Hand tracking at a distance of (a) 60 cm (b)1.6 m

At a distance of 1.6m from the Azure Kinect the application was showing poor performance, most of the time the hand was not detected and the user was unable to control the cursor on the screen. The cursor was flickering and was unstable. The click function was mostly not detected and sometimes it even clicked at wrong locations.

Script of Cursor using Hand tracking application

```
import cv2
import mediapipe as mp
import pyautogui
cap = cv2.VideoCapture(0)
hand_detector = mp.solutions.hands.Hands()
drawing_utils =mp.solutions.drawing_utils
screen_width, screen_height = pyautogui.size()
index_y = 0
while True:
    _, frame = cap.read()
    frame = cv2.flip(frame, 1)
    frame_height, frame_width, _ = frame.shape
    rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    output = hand_detector.process(rgb_frame)
    hands = output.multi_hand_landmarks
    if hands:
        for hand in hands:
            drawing_utils.draw_landmarks(frame, hand)
            landmarks =hand.landmark
            for id, landmark in enumerate(landmarks):
                x = int(landmark.x*frame_width)
                y = int(landmark.y*frame_height)

                if id == 8:
                    cv2.circle(img=frame, center=(x,y), radius=10, color=(0, 255, 255))
                    index_x = screen_width/frame_width*x
                    index_y = screen_height/frame_height*y
                    pyautogui.moveTo(index_x, index_y)
                if id == 4:
                    cv2.circle(img=frame, center=(x,y), radius=10, color=(0, 255, 255))
                    thumb_x = screen_width/frame_width*x
                    thumb_y = screen_height/frame_height*y
                    print('outside', abs(index_y - thumb_y))
                    if abs(index_y - thumb_y) < 60:
                        pyautogui.click()
                        pyautogui.sleep(1)
                        print('click')


    cv2.imshow('hand detection cursor', frame)
    cv2.waitKey(1)
```

6.2 Demo Application 2: Joint Angle Estimator

The objective of this project was to create a application for Azure Kinect to find angle between joints in our body. The scripting was done in Jupyter notebook (IDE). The idea was to create a application to estimate angle between joints using Azure Kinect. The motivation for this application was from the various bio medical uses that it can provide in future. The application can be developed into the field of physiotherapy for patients and remote consultancy where the doctor can consult the patient from a remote location and assist the patients.

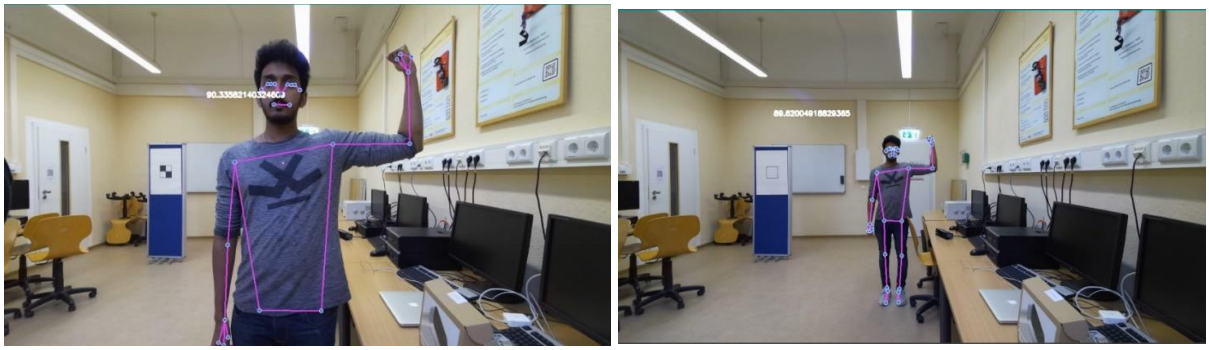
Here we use the Azure Kinect's RGB and Depth camera to detect the angle between the joints. For achieving this purpose, we are using Mediapipe library in python. Mediapipe is a cross platform framework to develop custom ML solutions for real and streaming media. Mediapipe provides many solutions and some of them include Hand detection, motion detection, Face mesh, Object detection. The other library we used in this application is open-source computer vision library (OpenCV), it is a library of programming functions for real time computer vision. It consists of C++, C, Python interfaces and supports Windows, Linux, IOS, and Mac OS.

In this application we are finding the angle between left wrist, left elbow and left shoulder. This application can be modified to find angle between other parts of the body. In the script first we are importing all the required libraries for the program. Then video source from Azure Kinect is taken, once taken this video source is used to process different joints on the body and the required joints are then extracted to find the angle between them, here we are extracting left wrist, left elbow and left shoulder. Then we define a function for calculating angle between these landmarks. Once calculated this angle is then displayed on the screen.

Test setup: The testing of our demo application 2 was done using Microsoft azure. First, we have download and install Azure Kinect viewer v1.4.1 software as a pre requisite, here we used a Windows 10 operating system with 16 GB ram and Nvidia 4GB 3050Ti graphics card for the test. The test was done by keeping Azure Kinect at a height of 1.315 meters from the floor and the test was conducted for various distances from the Azure Kinect ranging from 1m tom 8m at a regular interval of 1m. The person stands in front of the azure Kinect at a distance of 1m in the beginning and the results are taken, later moved to a distance of 2m and this continues up to 8m distance at regular 1m distance from azure and all the readings are taken and evaluated.

Methodology: First the power supply cable is connected to the Azure Kinect, then the USB type C data cable is connected to Azure and the operating system and made sure all the connections are stable next the power supply is turned ON and the Azure Kinect is kept 30 minutes for preheating. Once the pre heating is complete the program file is opened and run button is pressed, this opens up azure Kinect and we will be able to see our application.

Test Results: After the completion of the test, we were able to find out that the application was showing good results till a distance of 5m from azure Kinect ignoring slight variations due to human errors. At a distance of 1m the angle shown in azure was 90.33 degrees and at 3m it showed 89.62 degrees and at 4m it was showing 89.24 degrees.



(a) (b)

Fig 33: Angle estimation at (a) 1m and (b) 3m distance

After a distance of 5m from the azure there was showing large difference between estimated angle and the expected angle. At a distance of 6m it was showing 87.06 degrees, and at a distance of 8m it was showing 83.46 degrees.

Fig 34: Joint Angle estimation at 8m distance

Distance from Azure Kinect.	Angle shown in Azure Kinect in degrees.
1m	90.33
2m	91.36
3m	89.62
4m	89.24
5m	92.49
6m	87.06
7m	85.79
8m	83.46

Table 2: Joint angle estimations from experiment

Script for Joint angle estimation application

```
import cv2
import mediapipe as mp
import numpy as np

mp_drawing = mp.solutions.drawing_utils
mp_pose = mp.solutions.pose

for lndmrk in mp_pose.PoseLandmark:
    print(lndmrk)



def calculate_angle(a, b, c):
    a = np.array(a)
    b = np.array(b)
    c = np.array(c)

    radians = np.arctan2(c[1] - b[1], c[0] - b[0]) - np.arctan2(a[1] - b[1], a[0] - b[0])
    angle = np.abs(radians * 180.0 / np.pi)

    if angle > 180.0:
        angle = 360 - angle
    return angle



cap = cv2.VideoCapture(0)
```

```
with mp_pose.Pose(min_detection_confidence=0.5, min_tracking_confidence=0.5) as pose:
    while cap.isOpened():

        ret, frame = cap.read()

        # Recoloring
        image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        image.flags.writeable = False
        # make detections
        results = pose.process(image)

        image.flags.writeable = True
        image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)

        try:
            landmarks = results.pose_landmarks.landmark

            # get coordinates
            shoulder = [landmarks[mp_pose.PoseLandmark.LEFT_SHOULDER.value].x,
                        landmarks[mp_pose.PoseLandmark.LEFT_SHOULDER.value].y]
            elbow = [landmarks[mp_pose.PoseLandmark.LEFT_ELBOW.value].x,
                     landmarks[mp_pose.PoseLandmark.LEFT_ELBOW.value].y]
            wrist = [landmarks[mp_pose.PoseLandmark.LEFT_WRIST.value].x,
                     landmarks[mp_pose.PoseLandmark.LEFT_WRIST.value].y]

            # calculate angle
```

```
        angle = calculate_angle(shoulder, elbow, wrist)


        # visualize
        cv2.putText(image, str(angle),
                    tuple(np.multiply(elbow, [640, 480]).astype(int)),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 2, cv2.LINE_AA
                    )


        print(landmarks)
    except:
        pass
```

7. Conclusion

In this project we perform brief data analysis of Azure Kinect and its comparison to the Kinect V2 version. Then we thoroughly evaluate the new Azure Kinect with focus on depth accuracy and pixel aspects and comparison to the Kinect V2. Furthermore, we validate both sensors performance in indoor environments. We conclude with a discussion on its improvements in the context of the evolution of the Kinect sensor. It was shown that it is crucial to choose well designed experiments to measure accuracy, since the RGB and depth camera are not aligned.

To conclude our objective of the project was to do a comparison between Azure Kinect and Kinect v2 and to create a demo application for Microsoft Azure Kinect. We compared the depth data of Azure Kinect and with reference Bosch PLR 15 laser scanner (true value), and from the results we found out that the depth accuracy of Azure Kinect is almost similar to the laser scanner. We were able to successfully compare the pixel data of Azure Kinect and Kinect V2. Kinect V2 pixel data (pixels for X, Y axis) are similar with Azure Kinect Wide FOV.

The second part of the project was to create a demo application for Azure Kinect, and we created demo application 1 which uses hand tracking to control the cursor on the screen and this code was then tested and from the evaluation of these results we found out that the application showed best performance at a distance of 60 cm from the Azure Kinect and at a distance of 1.6m the application showed poor performance.

The second demo application was to estimate the angle between the joints using Azure Kinect. Here we choose to estimate the angle between left wrist, left elbow, and left shoulder. The results were taken for 8 readings(1m-8m) and by evaluating those results web found out that till a distance of 5m the difference between estimated angle and expected angle was small, but from 6m the difference in estimated and expected angle was large and least performance was shown at 8m distance from Azure Kinect. we were able to conclude that the application showed good results from 1m to 5m distance.

Reference

[1] - Augmented Reality Software für Unternehmen - Augment IT (augment-it.com)

[2] - https://www.seeedstudio.com/blog/2020/01/08/what-is-a-time-of-flight-sensor-and-how-does-a-tof-sensor-works

[3] - https://en.wikipedia.org/wiki/Kinect

[4, 5, 6] - https://learn.microsoft.com/en-us/azure/kinect-dk/

[7] - https://www.seeedstudio.com/blog/2020/01/08/what-is-a-time-of-flight-sensor-and-how-does-a-tof-sensor-work/

[9, 11] - https://learn.microsoft.com/en-us/windows/apps/design/devices/kinect-for-windows

[10] - http://x-tech.am/xbox-one-for-kinect-2/

[12] Tölgyessy, M.; Dekan, M.; Chovanec, L'.; Hubinský, P. Evaluation of the Azure Kinect and its comparison to Kinect v1 and Kinect v2. Sensors 2021, 21, 413. [CrossRef] [PubMed]

[13] Evaluating and Improving the Depth Accuracy of Kinect for Windows v2 Lin Yang, Longyu Zhang, Haiwei Dong, Abdulhameed Alelaiwi and Abdulmotaleb El Saddik, Fellow, IEEE

[14] - https://www.researchgate.net/figure/A-digital-image-is-a-2D-array-of-pixels-Each-pixel-is-characterised-by-its-x-y_fig1_221918148