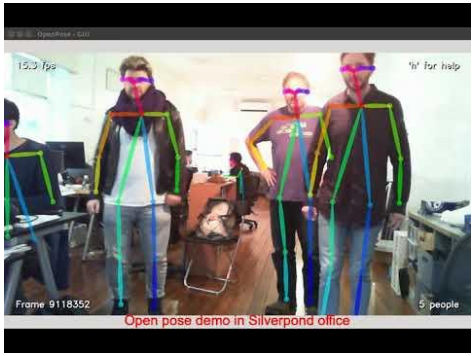
3D Cameras

Laboratorio di Realtà Virtuale

Manuel Pezzera – manuel.pezzera@unimi.it
Alessandro Tironi – alessandro.tironi@unimi.it
Jacopo Essenziale – jacopo.essenziale@unimi.it
Renato Mainetti – renato.mainetti@unimi.it

Overview

- 3D Cameras
  - Microsoft Kinect, Orbbec Astra, Intel RealSense
- 2D Cameras and human pose estimation
  - OpenPose, PoseNet, wrnch.ai
- Kinect 2 and Unity
  - Microsoft Kinect SDK
  - Kinect Unity Integration
  - Avateering in Unity
  - Kinect as NUI
  - Kinect & VR



Overview

- 3D cameras, also called “Depth Cameras”, are able to obtain a “3D photo” of the environment.
- It means that can calculate the distance between the camera and a subject.
- See the following image:

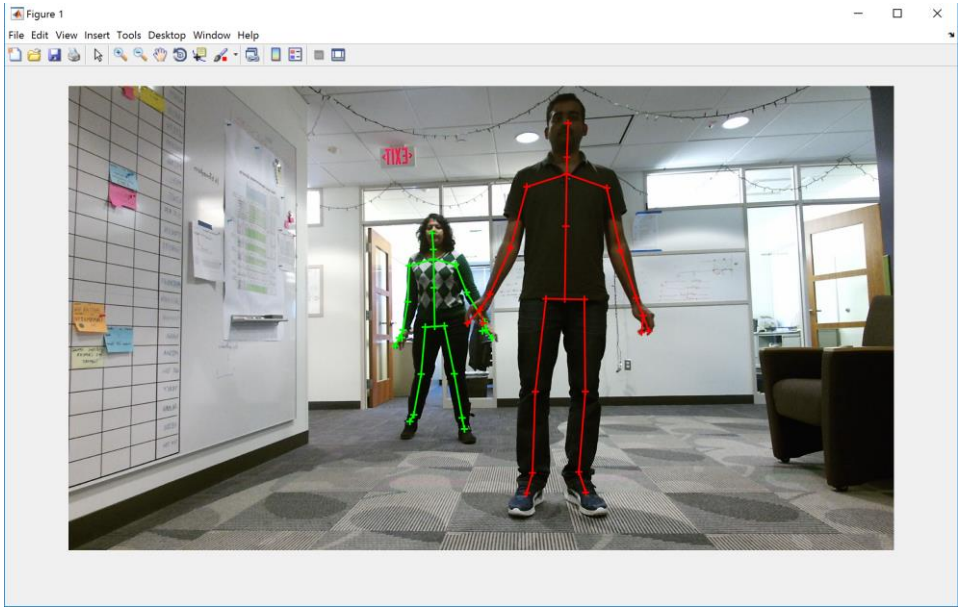
- The left image is recorded using a classical 2D camera and for an algorithm could be difficult to find the silhouette of the person. While, on the other hand, the picture on the right is captured using Kinect. The background is farther, so it has a different color.

3D Cameras: overview

- Depth image is used to detect human posture
  - Kinect 1 recognizes 20 different body joints.
  - Kinect 2 has been improved and can recognize 26 joints.

- Born as a gaming device/controller it has also been widely used in the research field.

- Lot of projects used it to record people movement trying to diagnose diseases or helping with home-rehabilitation.



3D Cameras: overview

- Most common depth cameras are:
  - Kinect 1
  - Kinect 2
  - Orbbec
  - Intel RealSense
  - HoloLens

Kinect 1

Kinect 2

Orbbec Astra

Intel Realsense D435

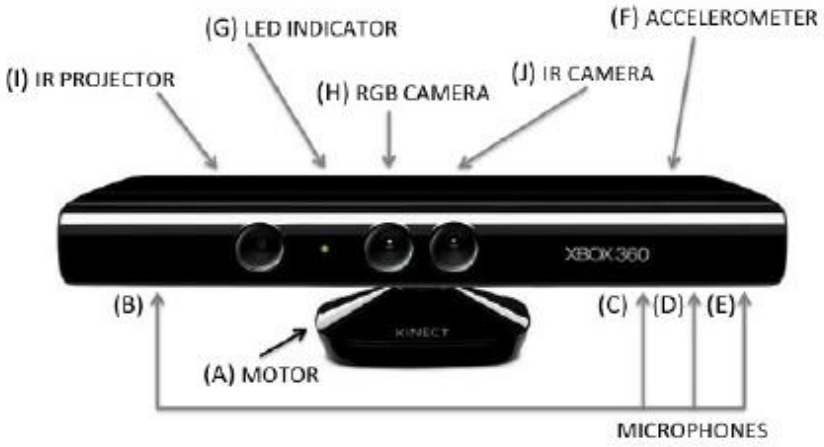
- Coming soon:
  - Azure Kinect

Azure Kinect

HoloLens

Kinect 1

- Launched in 2010 (for Xbox 360).



- Windows version came in 2012
  - Near mode added

- Kinect 1 has four main components:
  - An infrared projector
  - A RBG camera
    - 640x480 @ 30fps, 1280x960 @ 12fps
  - An infrared camera
    - 640x480 @ 30fps
  - Microphones array
    - 4 microphones

Kinect 1: how does it work?

1. The projector shoots an irregular pattern of dots invisible to humans.

2. CMOS sensors in the IR camera can detect the infrared light bounced off our subjects

3. Depth is calculated for every pixel in the scene: Kinect compares the data captured by the IR camera with the irregular pattern that has been projected. Triangulation methods are used to calculate the final distance.

This technology is called “structured light”.

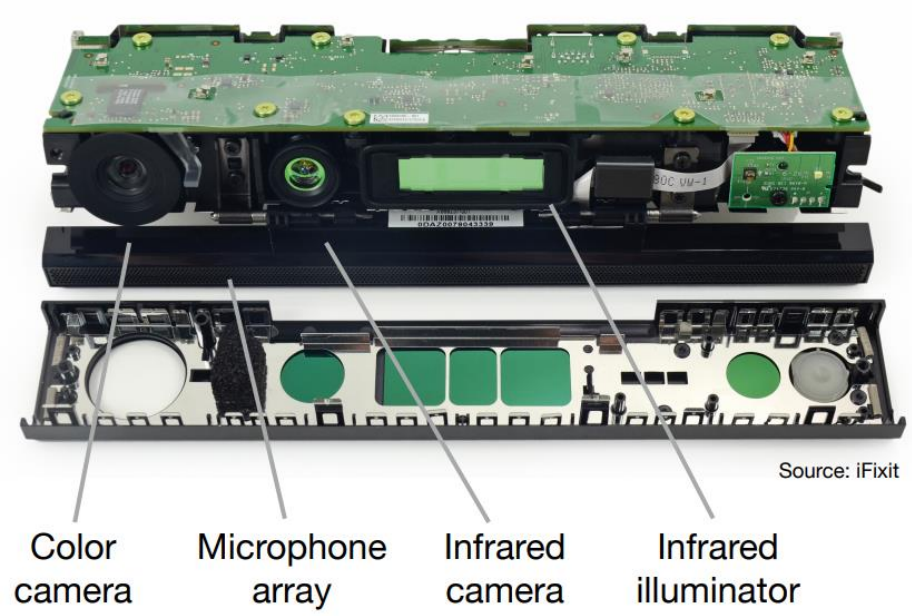
Useful videos:

- https://www.youtube.com/watch?v=uq9SEJxZiUg
- https://www.youtube.com/watch?v=dTKlNGSH9Po

Kinect 2

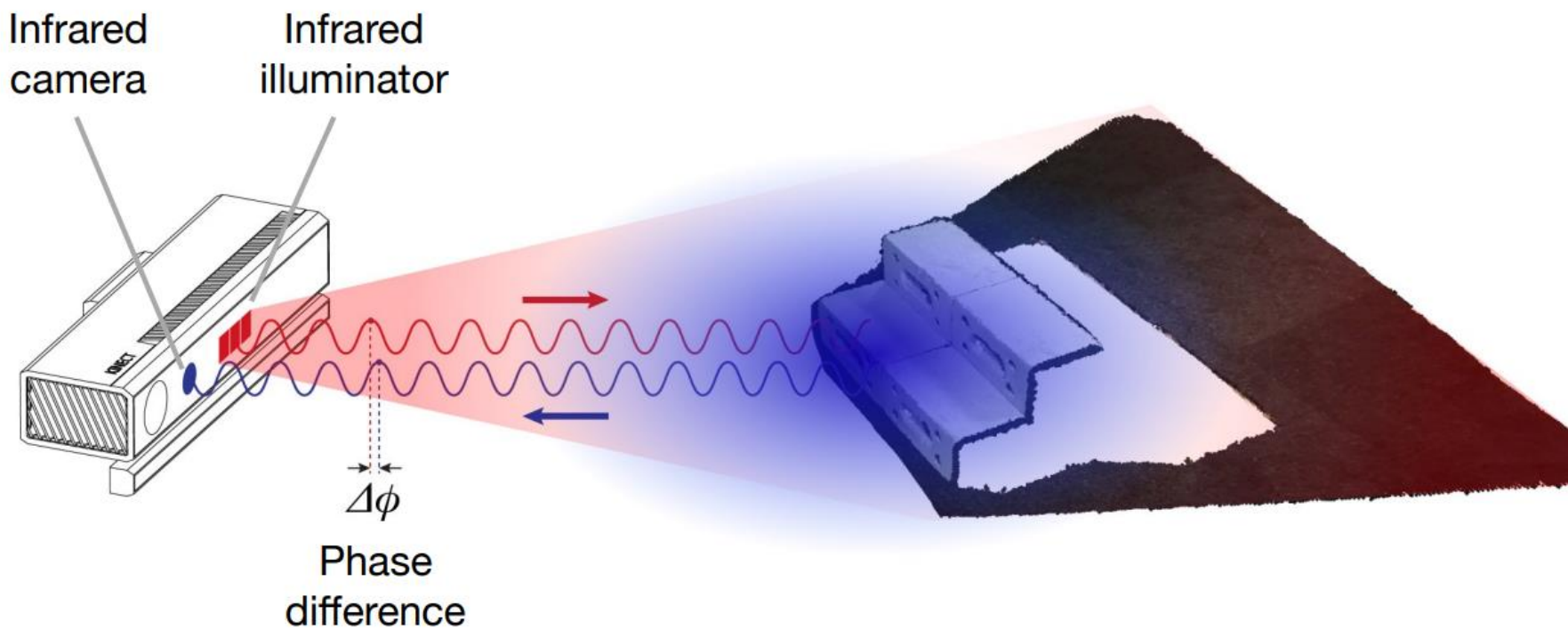
- Launched in 2013 for Xbox One and PCs.

- Main components:
  - An RBG camera
    - 1920x1080 @ 30fps
  - An infrared camera
    - 512x424 @ 30fps
  - Microphones array
    - 4 microphones



- Kinect 2 has the “new” Time of Flight (ToF) technology to calculate the depth image.
  - No dots are used here.

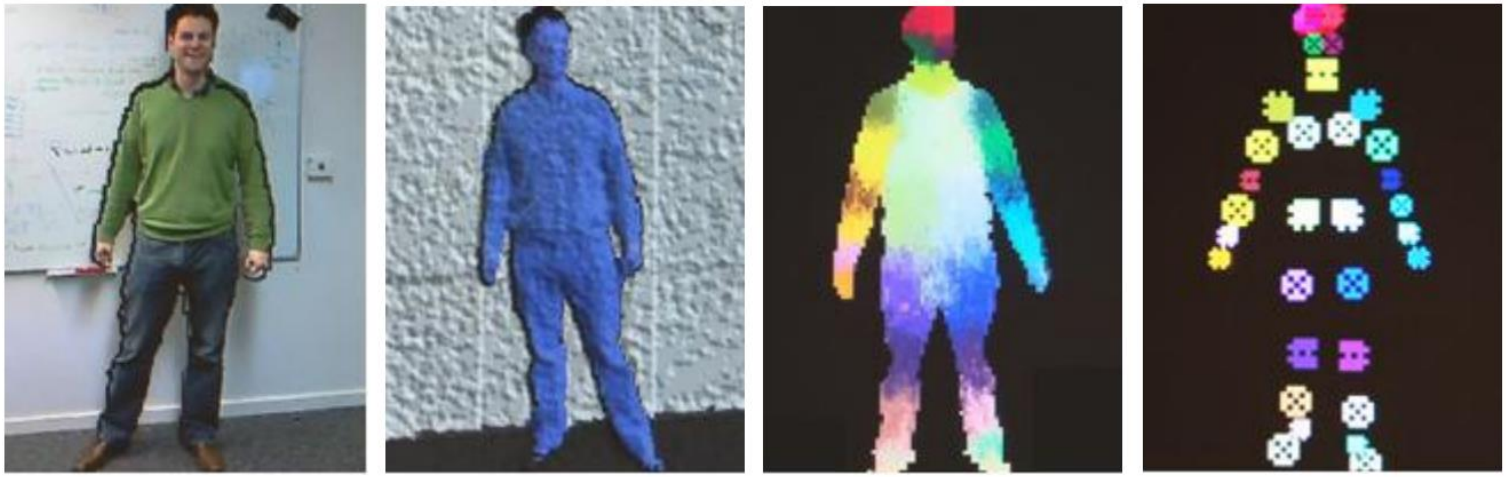
Time of Flight technology



- Useful link:
  - https://blogs.technet.microsoft.com/microsoft_blog/2013/10/02/collaboration-expertise-produce-enhanced-sensing-in-xbox-one/

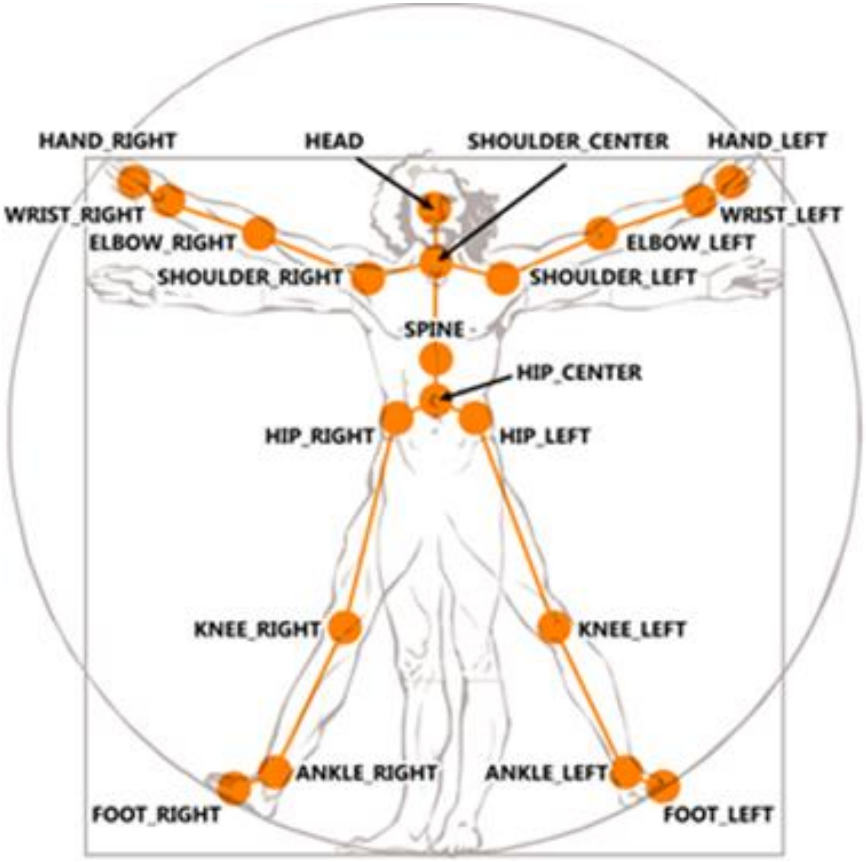
Kinect: human recognition

- We understood how Kinect 1 and Kinect 2 obtain the depth image.
- But how do they use this image to recognize human posture?
  - Usually random decision forest classifiers are used

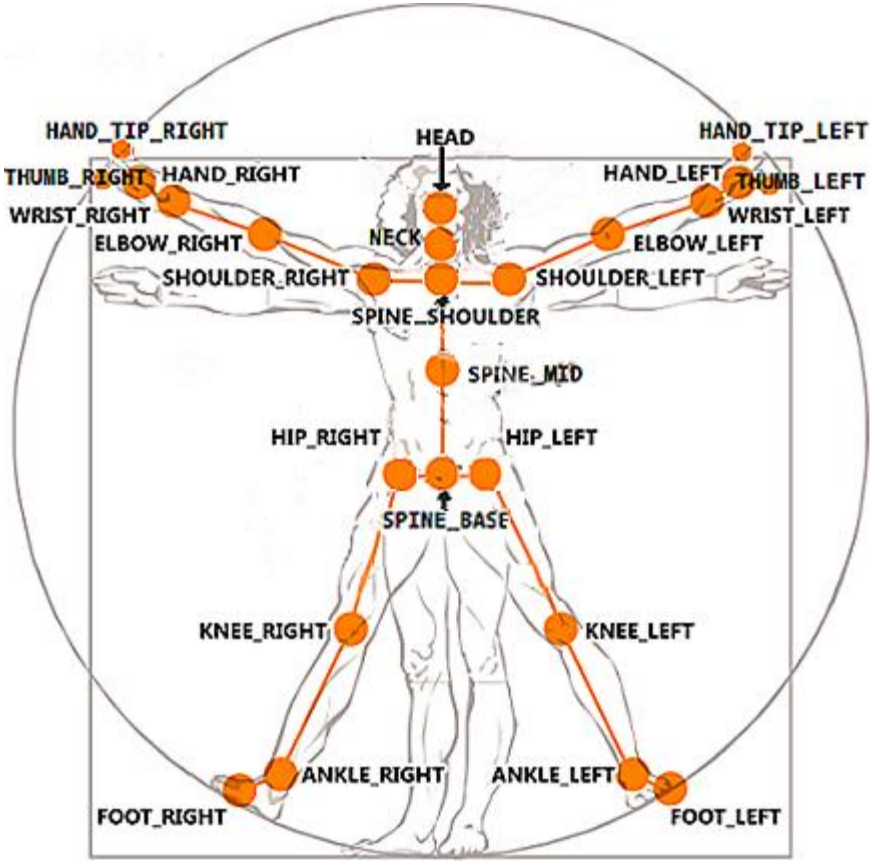


Detailed information: *Shotton, Jamie, et al. "Real-time human pose recognition in parts from single depth images." Cvpr. Vol. 2. 2011.*

Kinect 1 and Kinect 2 skeletons

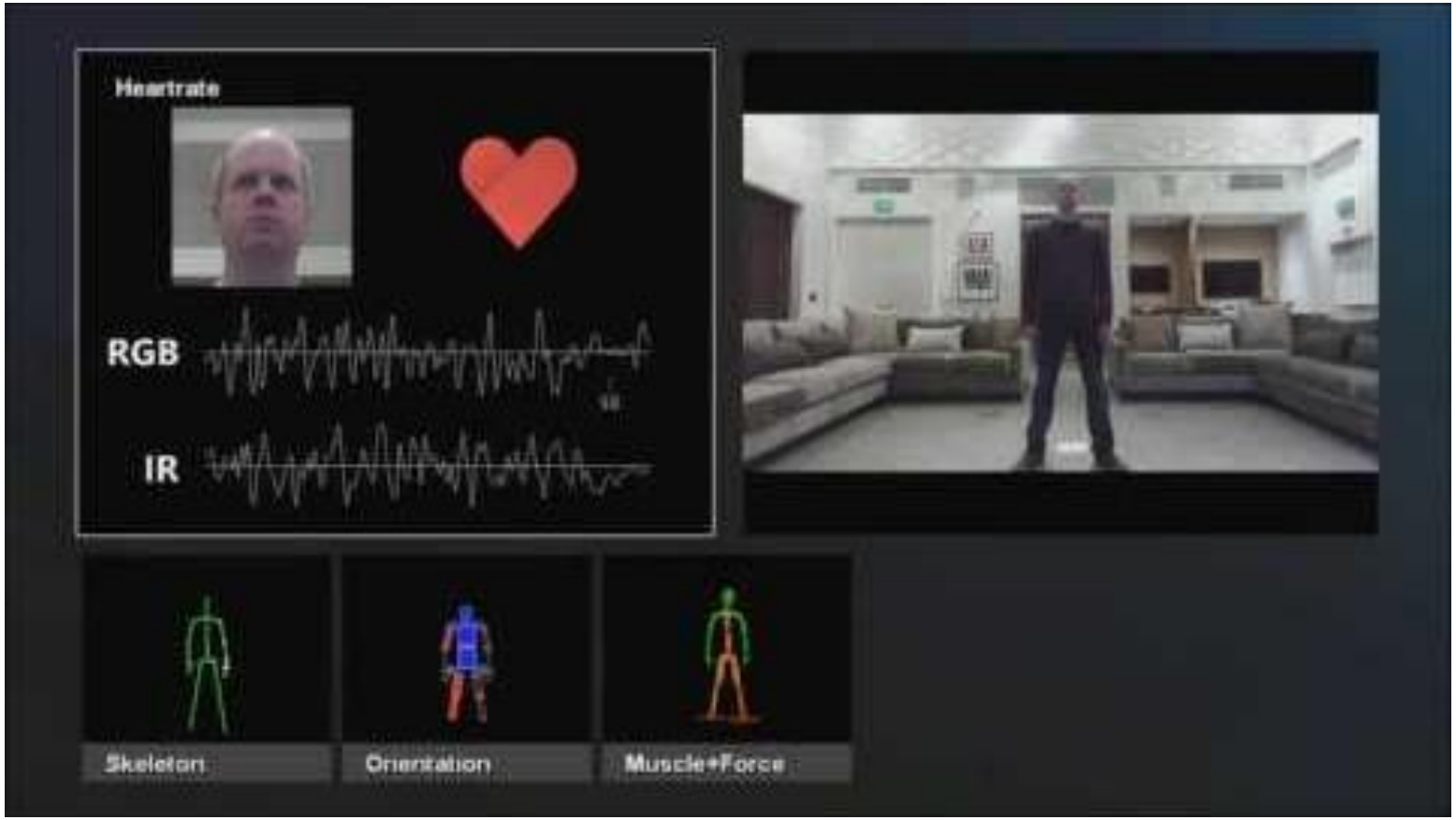


Kinect 1



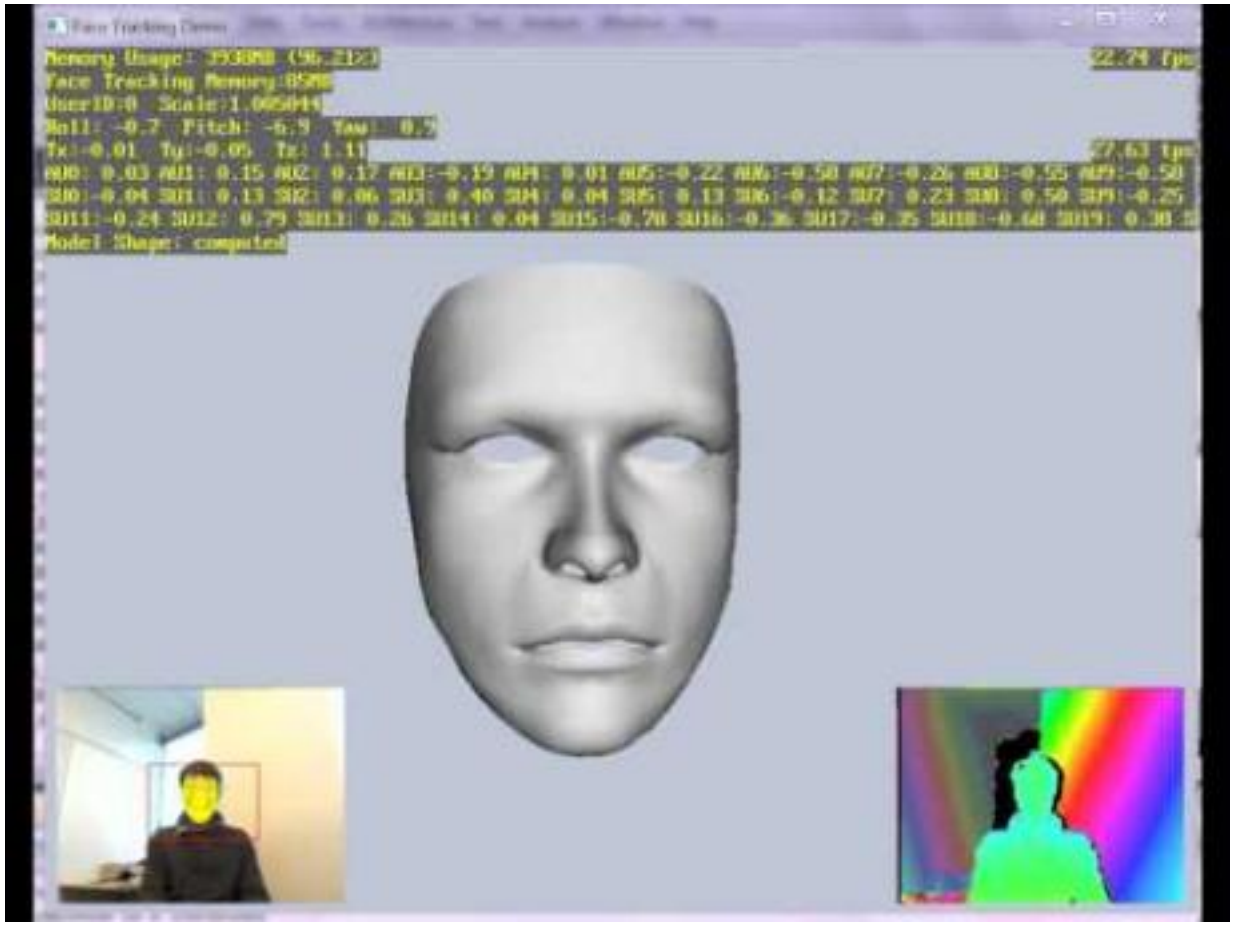
Kinect 2

Kinect 2 Demo



https://www.youtube.com/watch?v=OWzjn656kb4

Kinect 2 Demo



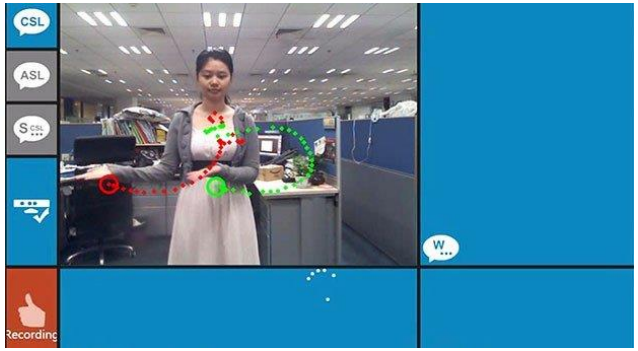
Video: https://www.youtube.com/watch?v=8_yMPLSZSEs

Kinect 2 HD Face: https://github.com/Vangos/kinect-2-face-hd

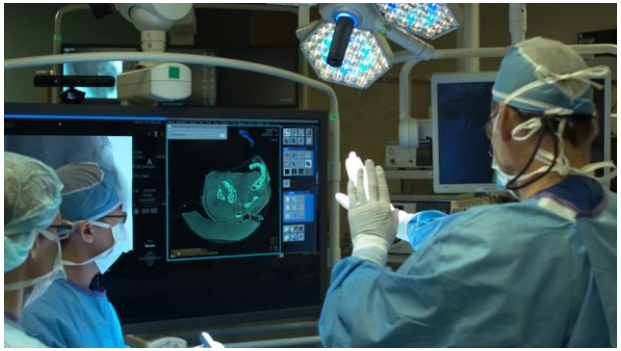
Kinect Application - Unity

- Few innovative uses of Microsoft Kinect
  - Produce high-quality 3D scans
    - E.g. Kinect Fusion
  - Help with stroke (or other diseases) recovery
  - Translate sign language
  - Retrieve data via gestures
  - Control robots with body movement

Kinect Fusion



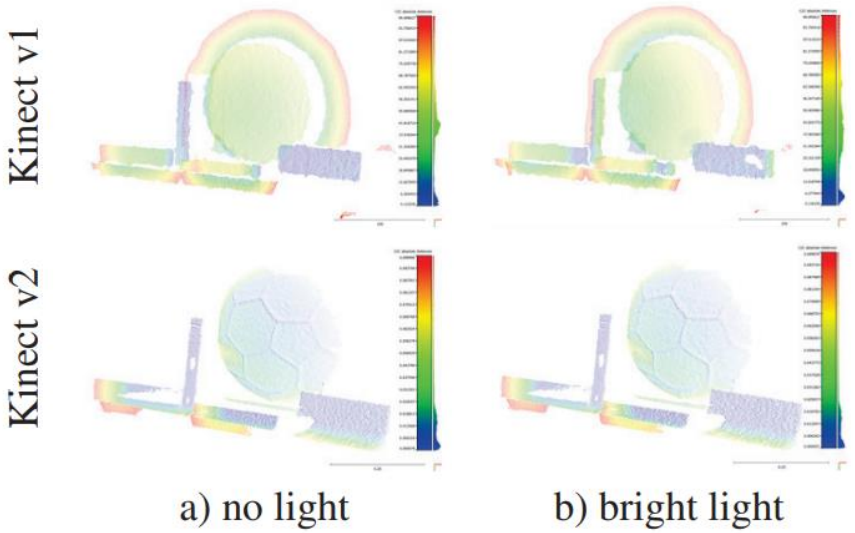
Translate sign language



Retrieve data via gestures

Kinect 1 and Kinect 2: depth image comparison

- Kinect 1 records reliable images just after starting, Kinect 2 needs to be pre-heat for at least 25 minutes in order to achieve reliable results.
- Accuracy decreases exponentially if we increase distance for Kinect 1, Kinect 2 does not have this problem.
- Artificial light can affect Kinect v1. Kinect 1 is total blindness in present of sunlight while Kinect 2 works quite good.



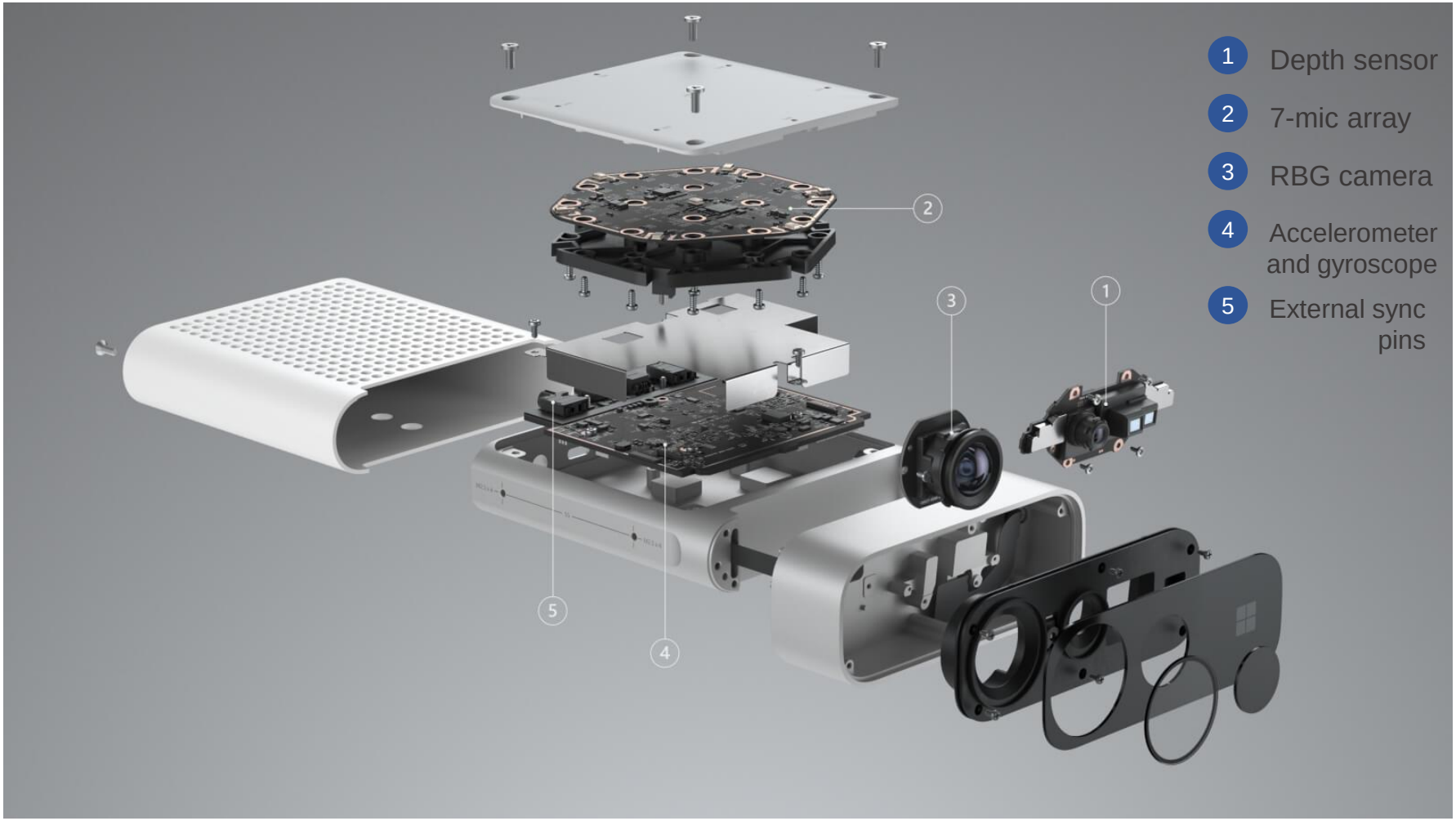
Source:

- Wasenmüller, Oliver, and Didier Stricker. "Comparison of kinect v1 and v2 depth images in terms of accuracy and precision."
- Zennaro, S., et al. "Performance evaluation of the 1st and 2nd generation Kinect for multimedia applications."

Azure Kinect

- Preorder started February 2019
- 12 MP RBG Camera
  - Up to 3840x2160 @ 30 FPS
- 1 MP Depth camera
  - 640x576 @ 30 FPS
  - 512x512 @ 30 FPS
  - 1024x1024 @ 15 FPS
  - Time-Of-Flight camera
- Motion sensor
  - Accelerometer and gyroscope
- Microphone array
  - 7 microphone circular array
- Vision API & Speech Service Sdk (from Azure)

Azure Kinect

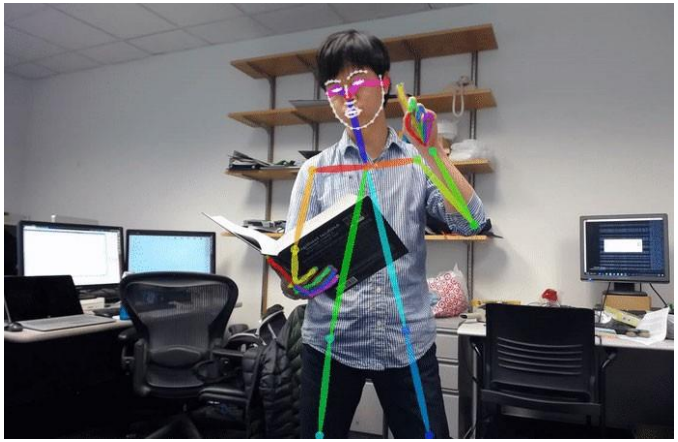


Azure API

https://www.youtube.com/watch?v=2URblck97y8

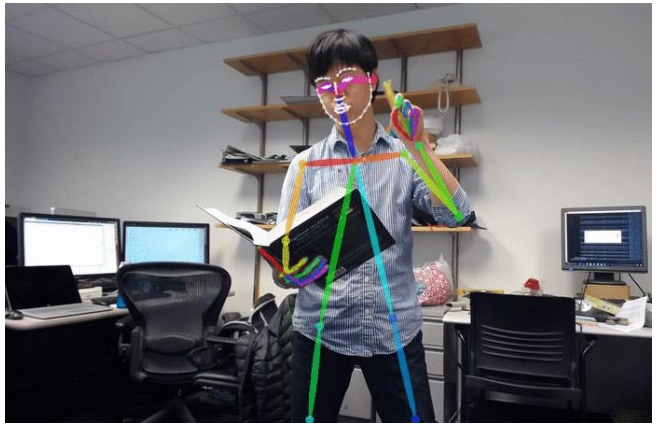
Human Pose Estimation with 2D Cameras

- In the last years few algorithms to estimate human pose from 2D cameras have been developed
  - E.g.: OpenPose, PoseNet, wrnch.ai
- These algorithms extract joints positions from a classic 2D image.
- Both offline and real-time!



OpenPose

- Source code available:
  - https://github.com/CMU-Perceptual-Computing-Lab/openpose
- Real time
- Multi-person
- Able to detect human body, hand, facial and foot, for a total of 135 keypoints
  - Face has 70 keypoints!
- Unity plugin available:
  - https://github.com/CMU-Perceptual-Computing-Lab/openpose_unity_plugin
- Demo: https://www.youtube.com/watch?v=Cu7g2Ldm-WM



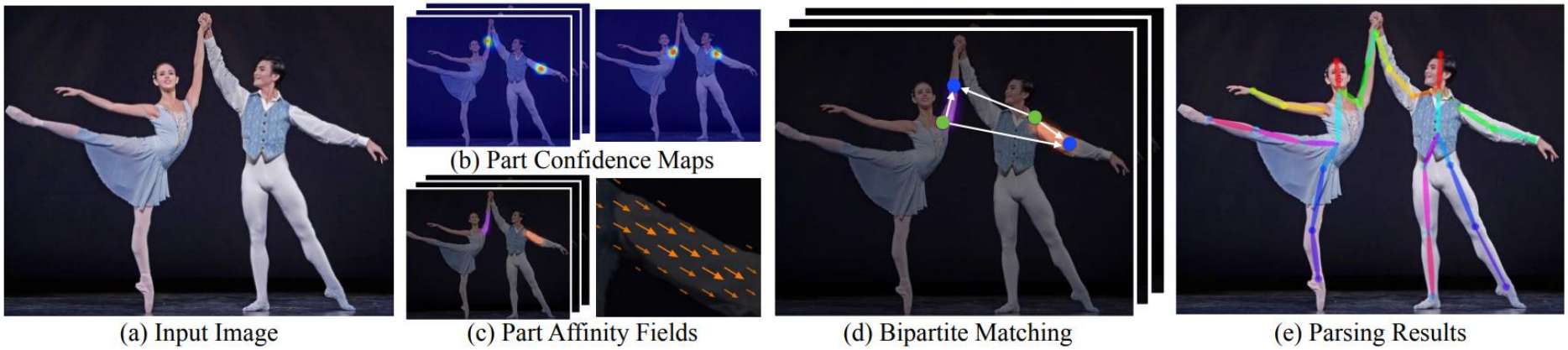
OpenPose

- Unbelievable!
- But…cons?
  - 9.75 FPS on a GTX 1080 Ti (body only)
    - All keypoints (body + hands + face): 3 FPS (4 if you are lucky)
  - With two GTX 1080 Ti we can reach 18 FPS

Graphics card model	#GPUs	Millisec / frame body-only+	FPS body-only+	GPU memory body (MiB)*+	Fps all (body,hands,face)+
GTX 1080 Ti (11GB)**	1		~9.75		~1/2 or 1/3 of body-only
Titan X Pascal (12GB)	1	115.7	8.6	1489	~1/2 or 1/3 of body-only
GTX 1080 (8GB)	1	119.4	8.4	1443	~1/2 or 1/3 of body-only
GTX 1070 (8GB)**	1		~8.75		~1/2 or 1/3 of body-only
Quadro M6000 (24GB)**	1		~6.6	~1500	~1/2 or 1/3 of body-only
GTX 980 (8GB)**	1		~6.5	~1500	~1/2 or 1/3 of body-only

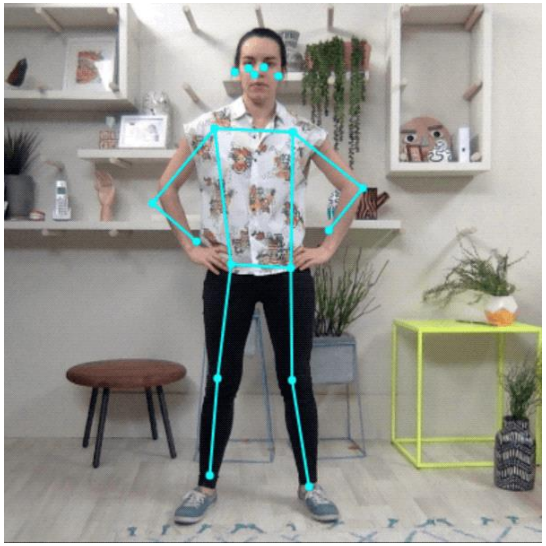
OpenPose: how does it work?

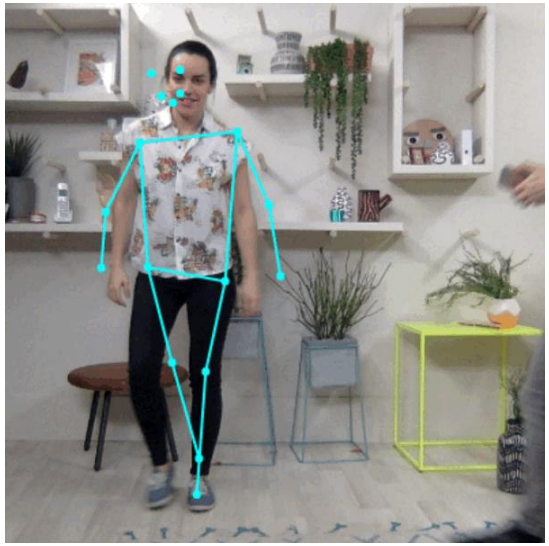
- A convolutional network predicts a set of 2D confidence map of body parts locations and a set of 2D vector fields of part affinities, which encode the degree of association between parts.
- Non Maximum Suppression used to extract body parts locations.
- Bipartite matchings are performed to associate body part candidates.
- Finally merge all the limbs together.
- Source: *Cao, Zhe, et al. "OpenPose: realtime multi-person 2D pose estimation using Part Affinity Fields." arXiv preprint arXiv:1812.08008 (2018).*

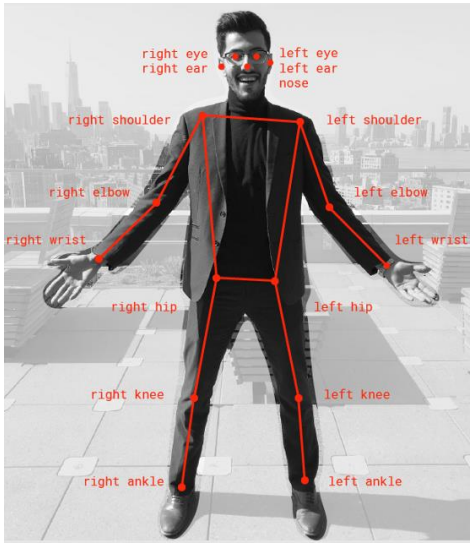


PoseNet

- Real-time multi-person human pose detection (like OpenPose).
- Demo online available:
  - https://storage.googleapis.com/tfjs-models/demos/posenet/camera.html
  - If you have a webcam you can try it
- Simpler than OpenPose, it has only 17 keypoints.
- Unity plugin available: https://github.com/infocom-tpo/PoseNet-Unity

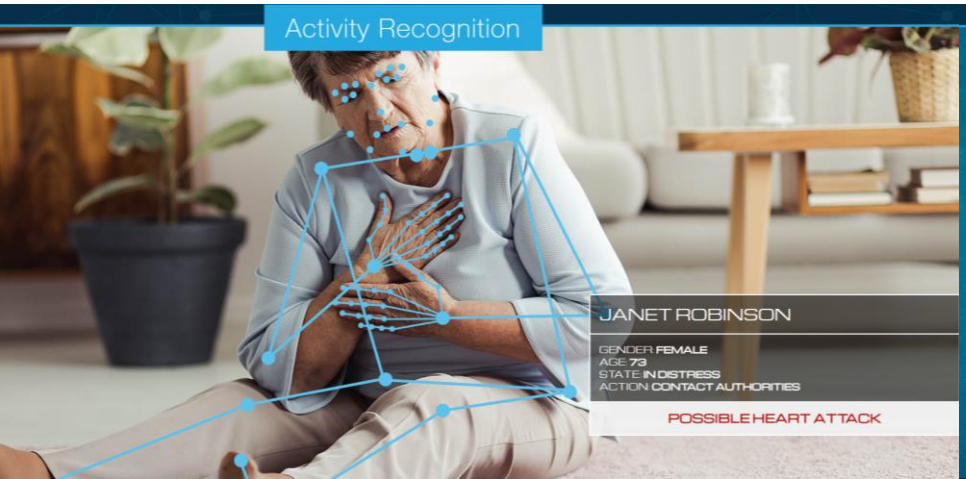






wrnch.ai

- Human pose detection
- Activity recognition
- Gesture recognition
- It works on mobile devices
- No price, no demo, unable to try, few information available…

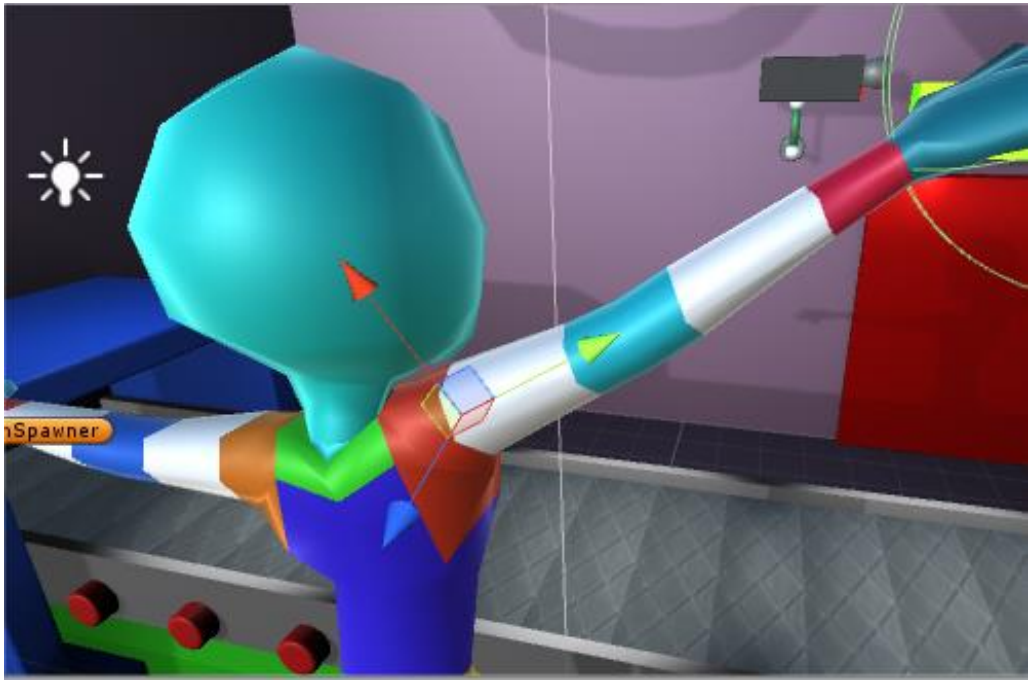


Kinect & Unity

- Let’s try now Kinect with Unity.

Kinect’s rotations

- For each frame, Kinect 2 returns the orientation quaternion of each joint (leaf joints excluded).
- Each quaternion represents the absolute rotation of the parent bone (e.g: the ElbowRight quaternion represents the right arm rotation).
- The Y axis is always parallel to the bone



Microsoft Kinect SDK 2.0

- The SDK gives you the ability to access all the Kinect's features (frame by frame):
- Color sensor (RGB image)
- Depth sensor (Float depth image)
- Audio samples
- Body tracking (Joint Pos (X,Y,Z), Bone Quaternion (X,Y,Z,W))

- Other features:
- Face tracking
- Gesture Recognition

Sensor initialization and joint evaluation

- Let’s try showing the movements of the joints in a unity scene.
- Remember to deallocate each frame after using it. (in c# use the using structure)
  - It is possible to retrieve the 3D coordinates using the `Body.Joints` dictionary:

```
// Retrieves the absolute position of the spine base (root).
CameraSpacePoint spineBasePosition = body.Joints[JointType.SpineBase].Position;
```

  - Pay attention to the reference axis system! (Kinect is a “right-handed”)

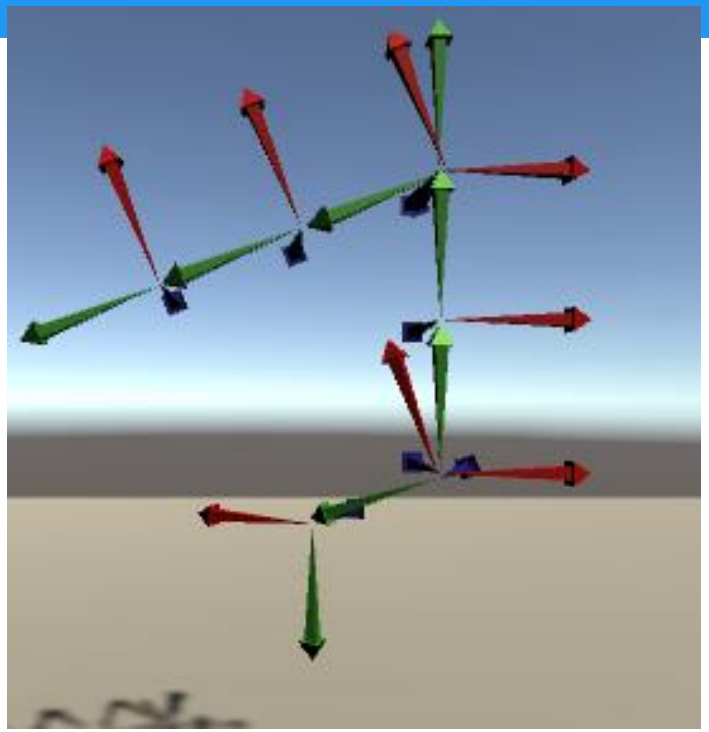
Sensor initialization and shutdown

```
private void InitializeKinectSensor()
{
    // Gets and opens default sensor.
    KinectSensor sensor = KinectSensor.GetDefault();
    if (!sensor.IsOpen)
        sensor.Open();

    // Opens depth stream.
    BodyFrameReader bodyFrameReader = sensor.BodyFrameSource.OpenReader();

    // Initializes array of tracked bodies.
    Body[] bodies = new Body[bodyFrameReader.BodyFrameSource.BodyCount];
}
```

```
private void ShutdownKinectSensor()
{
    Sensor.Close();
}
```

From positions to rotations

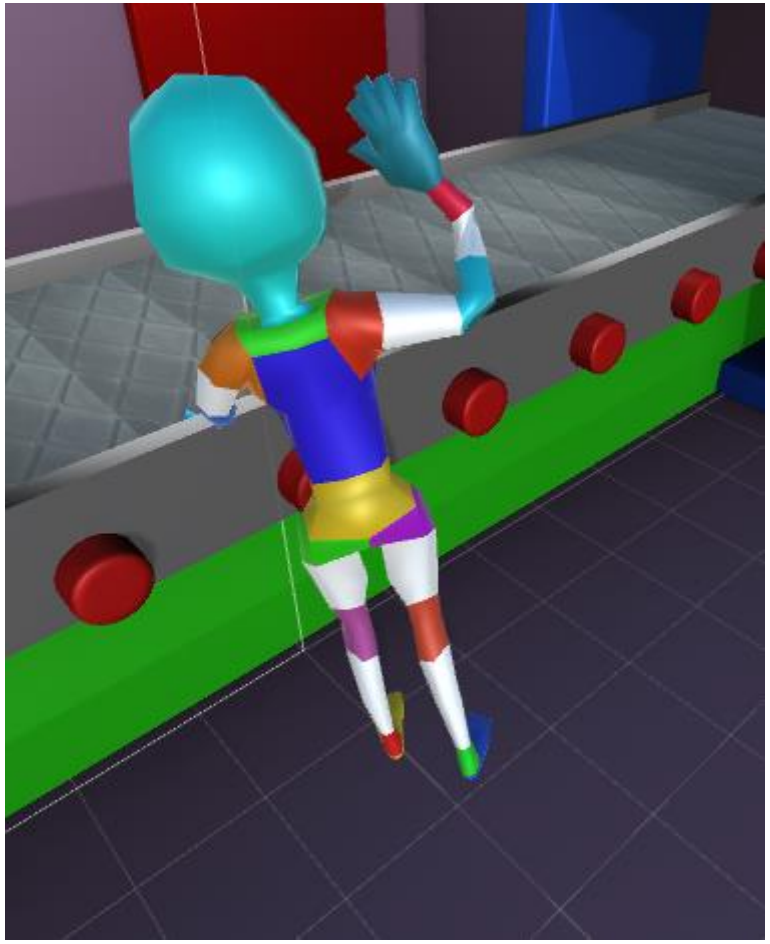
- We show in another scene how Kinect rotations change in real time.
- We switch from Kinect's reference system to Unity's system by reverting Z and W values for each quaternion.
- If we want to apply additional rotations, we multiply the retrieved quaternion by another.



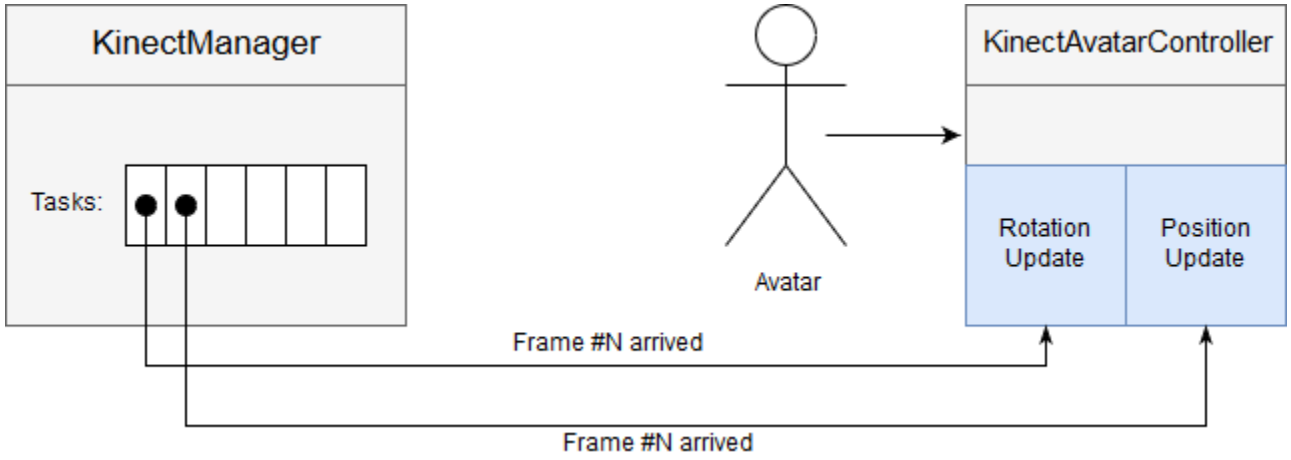
```
// Gets the quaternion acquired from Kinect.
// Also reverses the Z axis and the angle in order to convert the reference frame.
var currentJointOrientation = avatarBody.JointOrientations[currentJoint].Orientation;
Quaternion newOrientation = new Quaternion(currentJointOrientation.X,
        currentJointOrientation.Y, -currentJointOrientation.Z,
        -currentJointOrientation.W);
```

Avateering

- Let’s import an already skinned avatar from Blender and apply rotations to joints.
- If we correctly polish rotation offsets then we will see the mesh move without any deformation.



How to use the Kinect package inside our scene



- There is a singleton *KinectManager* object which constantly retrieves a new frame and sends it to a set of handles (named *KinectServices*). Each *KinectService* will elaborate the new frame independently.
- The avateering-related *KinectService* will apply received rotations to the avatar, after the required elaborations.

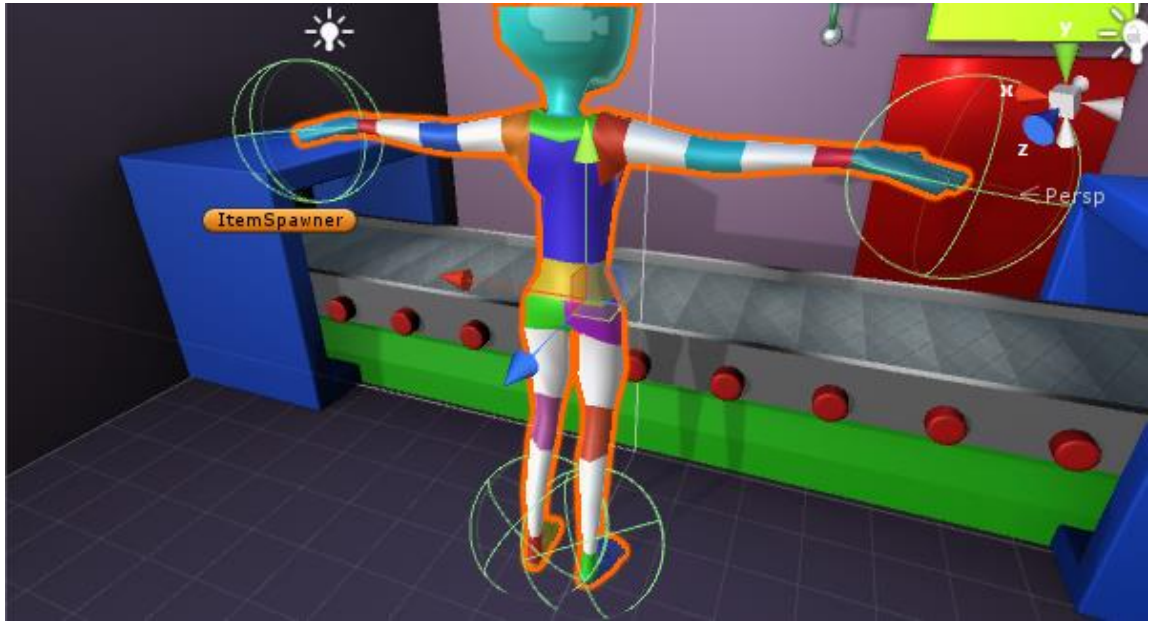
```
// Executes each of the elaboration tasks in the given order.
int nTasks = Tasks.Count;
for (int i = 0; i < nTasks; i++)
        Tasks[i].Elaborate(bodies);
```

Avateering algorithm (simplified)

```
7 riferimenti
public override void Elaborate(Body[] bodies)
{
    foreach (JointType joint in Enum.GetValues(typeof(JointType)))
    {
        // Gets the quaternion acquired from Kinect.
        var jointOrientation = bodies[avatarId].JointOrientations[joint].Orientation;
        Quaternion orientationQuaternion = new Quaternion(
            jointOrientation.X, jointOrientation.Y,
            -jointOrientation.Z, -jointOrientation.W
        );

        // Applies corrective rotation from the corrective rotations dictionary.
        Quaternion adjustment;
        if (correctiveRotations.TryGetValue(joint, out adjustment))
            orientationQuaternion *= adjustment;

        // Applies the rotation.
        avatar.JointObjects[joint].transform.rotation = orientationQuaternion;
    }
}
```

Hitting cubes

- We want the avatar to prevent the red cubes from arriving to the end of the belt by punching or kicking them, so we apply a rigidbody and a sphere collider to hands and feet.



Hitting cubes

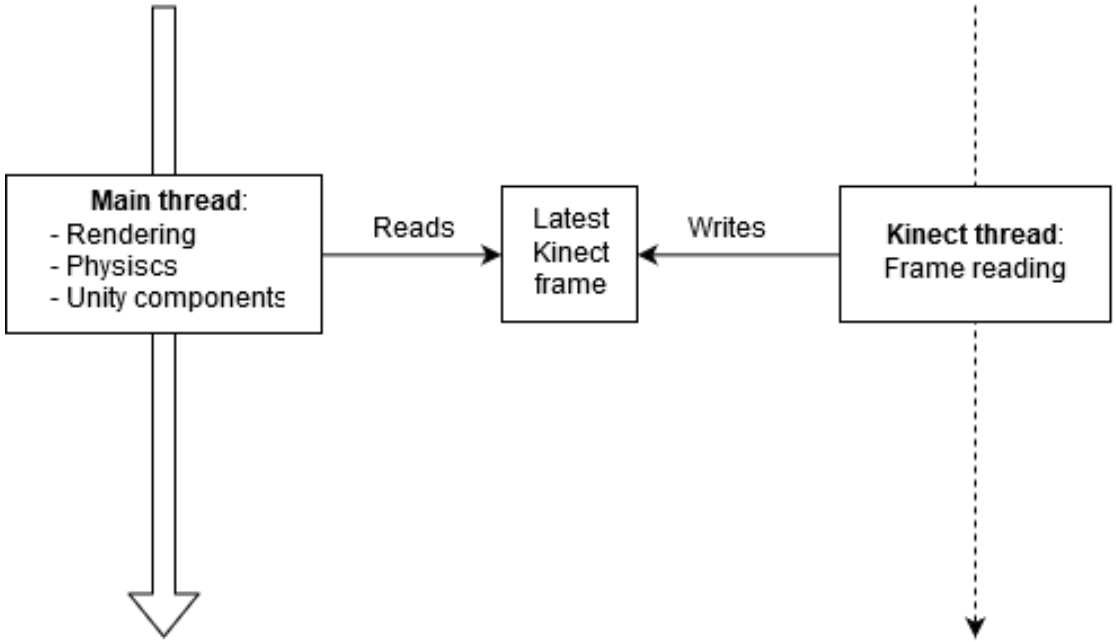
- In order to compute the punch/kick direction we buffer the hands and feet positions from the previous frame. Whenever a collision occours, we compute the direction vector with the current position and the buffered one.

```
public override void Elaborate(Body[] bodies)
{
    Body playerBody = bodies[player.Id];

    foreach (JointType jointType in bufferedJoints)
    {
        // Retrieves the joint position.
        CameraSpacePoint acquiredPosition = playerBody.Joints[jointType].Position;
        Vector3 newPosition = new Vector3(acquiredPosition.X, acquiredPosition.Y, acquiredPosition.Z);

        // Updates buffers.
        bufferOld[jointType] = bufferNew[jointType];
        bufferNew[jointType] = newPosition;
    }
}
```

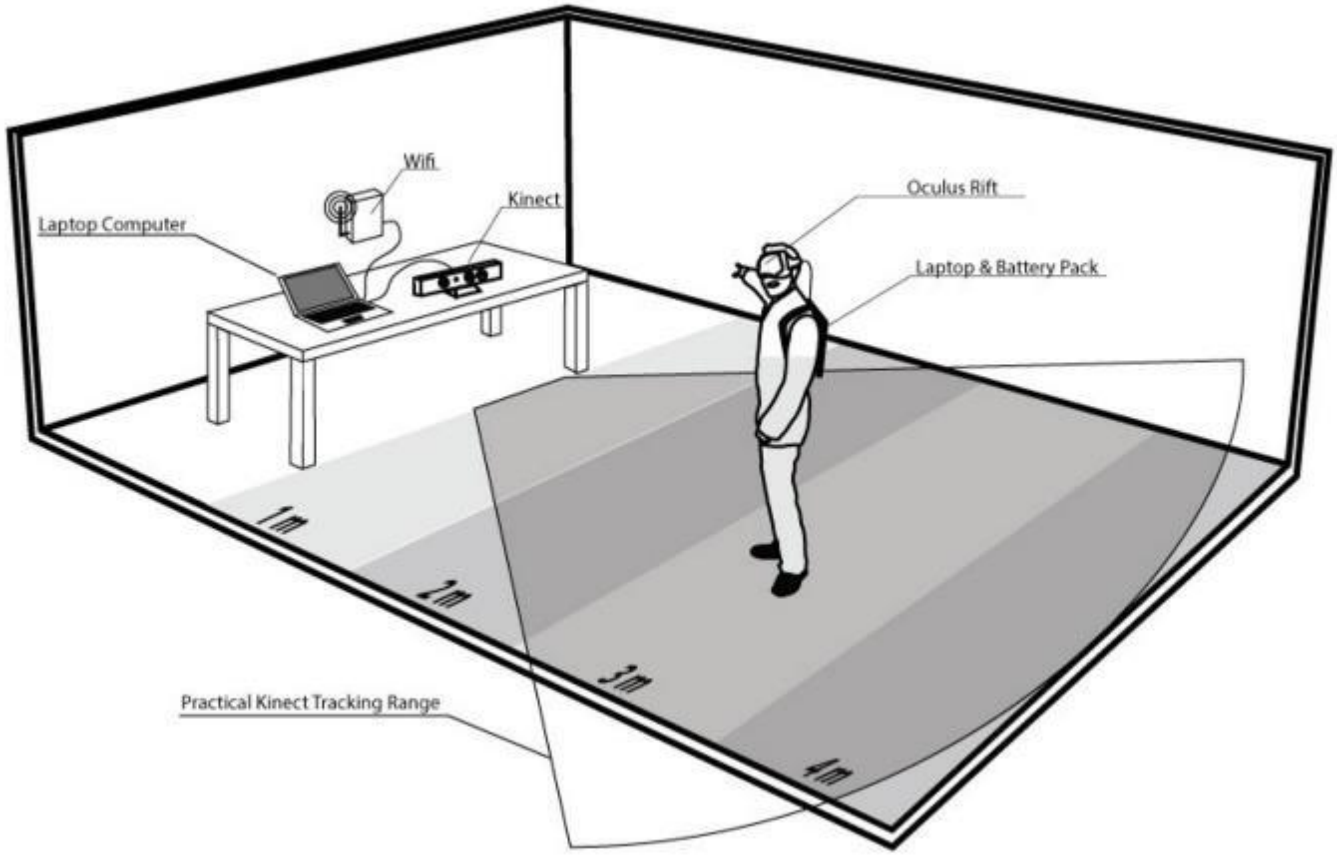
```
public Vector3 GetDirection(JointType joint)
{
    return bufferNew[joint] - bufferOld[joint];
}
```

Can you optimize this?

- *KinectManager* calls *AcquireLatestFrame* once for each game frame. This means that the game frame rate will be forced to the Kinect's frame rate!
- Possible solution: multi-threading! A thread reads data from Kinect and writes into a buffer. The game thread reads the buffer when needed.
- … locks?



Kinect and VR



- You can try this combination for your final project in the lab!