TrabHCI
Technologies to Reduce the Access Barrier in Human Computer Interaction
Erasmus Intensive Programme IP29588-1-1731-10

Kinect: getting started

2

Michela Goffredo
University Roma TRE
goffredo@uniroma3.it

What's Kinect Sensor

KINECT™
for Windows®

Microsoft Kinect is a motion sensor by Microsoft Xbox which allows to extract:

- RGB video stream
- Depth video stream
- Skeleton data (i.e. human body model by means of segments)
- Speech

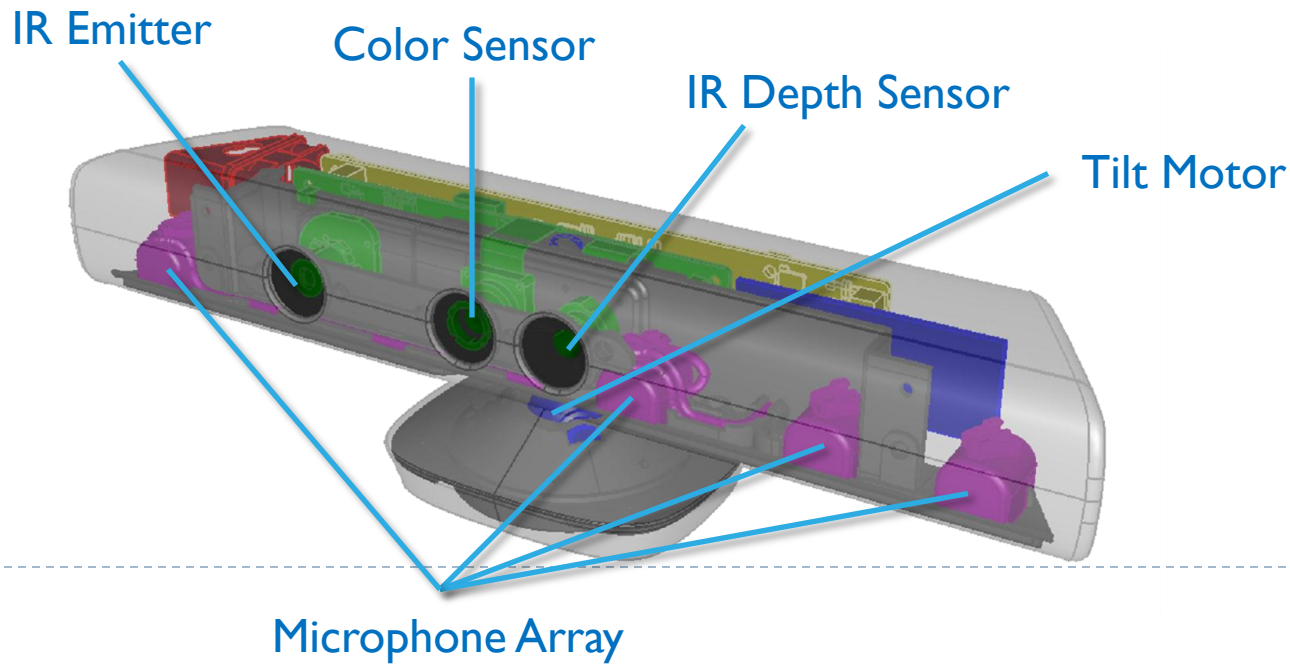
What's Kinect Sensor

http://www.youtube.com/watch?v=RT7hGBY5FZU

What's Kinect Sensor

Microsoft Kinect is composed of:

- **IR emitter** → IR pattern
- **IR depth sensor** → depth image from the analysis of the deformation of the IR pattern
- Color sensor → color camera
- Tilt motor
- Microphone array



Camera resolution

Color camera

- 12 fps 1280x960 pixel^2
- 15 fps 640x480 pixel^2 RAW
- 30 fps 640x480 pixel^2

Depth camera

- 30 fps 80x60; 320x240; 640x480 pixel^2

What is depth data

Depth data is:

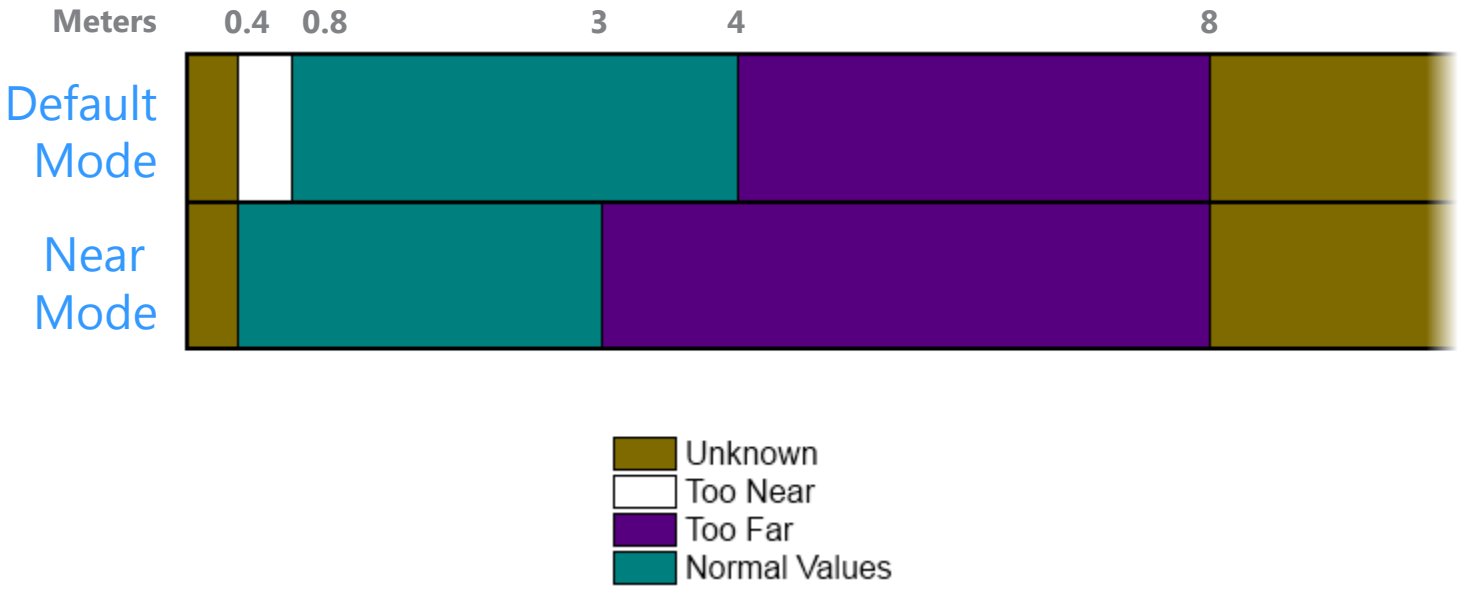
1. the distance (mm) of every pixel from the sensor
   320x240 → 76800 pixel!
2. a label indicating if the pixel belongs to a player

Kinect can work in to 2 different modes: default and near mode.

Differences mainly involve the possibility to extract the 19-joints skeleton model.

Mode	Depth & Player	Center Hip Joint	Other 19 Joints
Default	Yes	Yes	Yes
Near	Yes	Yes	No, for v1.0

Distances



Depth data is an array of *shorts* containing the distances and the player label.

The distances depend on the mode: the greenish area is ok!

- Default mode: 0.8 – 4 m
- Near mode: 0.4 – 3 m

Depth data

How to get distance data and the player label from depth data:

Distance data:

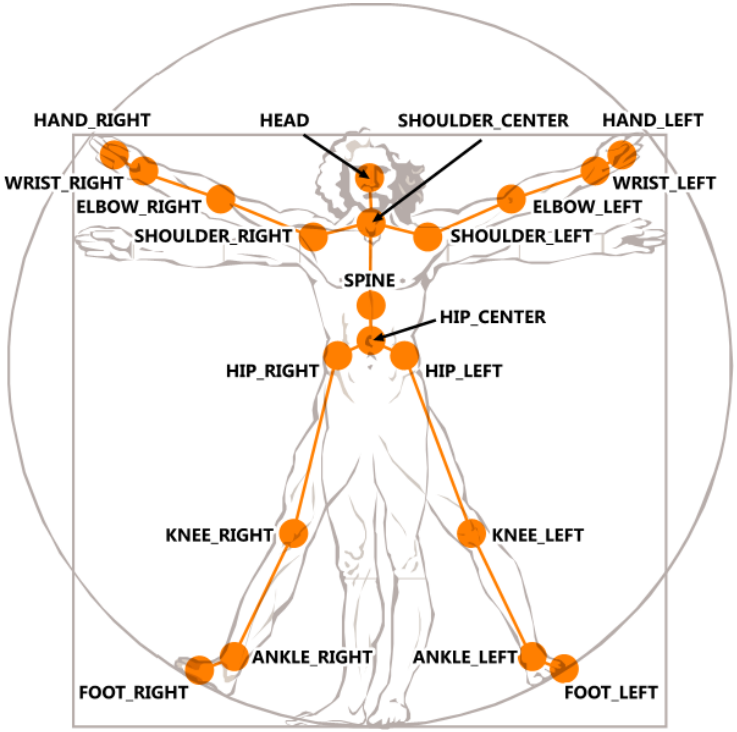
```
int depth = depthPoint >> DepthImageFrame.PlayerIndexBitmaskWidth;
```

Player label:

```
int player = depthPoint & DepthImageFrame.PlayerIndexBitmask;
```

Skeleton tracking

From the analysis of depth data (distance in mm + player label), Kinect applies a human body model composed of 20 joints connected with segments:



Skeleton tracking

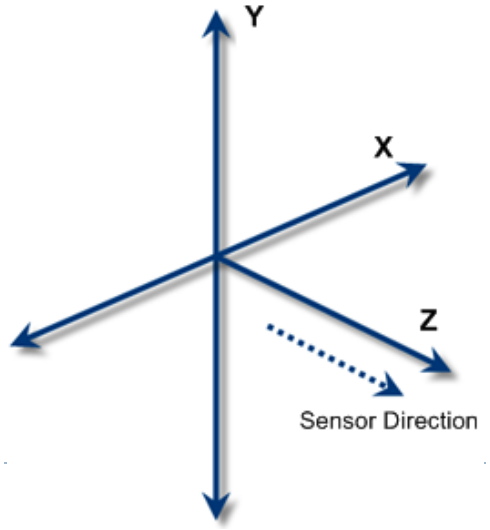
Kinect gives:

- a set of x, y, z joints in meters for each player;
- an associated state for each joint:
  - Tracked
  - Not tracked
  - Inferred (occluded, clipped, or low confidence joints)

More specifically:

- Maximum 2 players tracked at once
- Each skeleton has a unique identifier

Please note the reference system:



Getting started

What do you need?

1. Buy the Kinect sensor
2. Download the Kinect SDK v1.0 from the website and install it:

   www.microsoft.com/en-us/kinectforwindows

3. Have a C# development tools (i.e. Microsoft Visual Studio)
5. Microsoft .NET Framework 4.0
6. Windows 7

Getting started

HW requirements?

5. Dual-core CPU, 2.66-GHz or faster
6. ≥2 GB RAM

Useful stuff for developpers:

EMGU: cross platform .Net wrapper to the Intel OpenCV image processing library. It allows OpenCV functions to be called from .NET compatible languages such as C#.

First program

1. **Read** data from the hard disk:
   - Color data
   - Depth data
   - Skeleton data

   } gathered with the Kinect sensor

2. **Show data** into 3 different images into MainWindow
3. **Analyse skeleton data** for:
   - Extract the 3D coordinated of each joint
   - Recognise the posture/gesture

Settings

- Create a new Project WPF Application in C# in Open Visual Studio
- Destination Framework .NET4 (project-properties)
- Reference **System.Drawing**
- Reference the Kinect API:

  Reference/Add Reference/**Microsoft.Kinect**

MainWindow.xaml

- Resize the Window so that it is possible to fit **3 images** with size 320x240.
- Add the images into the Window and set width and height in the Properties Window (View/Properties Window).
- Add the following events to the MainWindow Properties::
  - Loaded
  - Closed
- …let's go in MainWindow.xaml.cs !

MainWindow.xaml.cs

- Add the following:

```
using Microsoft.Kinect;
using System.Drawing.Imaging;
using System.Drawing;
using System.IO;
using System.Timers;
using System.Windows.Threading;
```

Reading and showing a video file

In the Window_Loaded method:

```
Capture _captureColor
_captureColor = new Capture(@"C:\Miki\colorVideo.avi");


double fps =
_captureColor.GetCaptureProperty(Emgu.CV.CvEnum.CAP_PROP.CV_CAP_PR
OP_FPS);

My_Time.Tick += new EventHandler(My_Timer_Tick);
My_Time.Interval = new TimeSpan(0, 0, 0, 0, (int) ((1 / fps) *
1000));

My_Time.Start();
```

Reading and showing a video file

My_Timer_Tick is a method related to an event:

```
void My_Timer_Tick(object sender, EventArgs e)
```

where we actually capture the frame and show in in image1:

```
using (colorFrame = _captureColor.QueryFrame())
{
    if(colorFrame == null)
    {
        return;
        }
// transform colorFrame into BitmapSource (WPF)
colorImageBmp =
CreateBitmapSourceFromBitmap(colorFrame.ToBitmap(colorFrame.Width,
colorFrame.Height));

//show color image in image1
image1.Source = colorImageBmp;                    }
```

Reading and showing a video file

`CreateBitmapSourceFromBitmap` is a method for transforming colorFrame into BitmapSource (paste & copy & use it):

```
BitmapSource CreateBitmapSourceFromBitmap(Bitmap bitmap)   {
BitmapSource bs;
if (bitmap == null){
    return null;
}
try{
bs =
System.Windows.Interop.Imaging.CreateBitmapSourceFromHBitmap(bitmap.GetHbit
map(), IntPtr.Zero, Int32Rect.Empty, BitmapSizeOptions.FromEmptyOptions());
}
catch {
bs = null;
My_Time.Stop();
}
finally {
bitmap.Dispose();
}
    return bs;}
```

Reading and showing a video file

And of course we need to define the following:

```
DispatcherTimer My_Time = new DispatcherTimer();
Image<Bgr, Byte> colorFrame;
BitmapSource colorImageBmp;
```

Remember to stop `My_time` when the Window is closed!

In the Window_Closed method add:

```
My_Time.Stop();
```

First program

1. **Read** data from the hard disk:
   - Color data
   - Depth data
   - Skeleton data

   } gathered with the Kinect sensor

2. **Show data** into 3 different images into MainWindow
3. **Analyse skeleton data** for:
   - Extract the 3D coordinated of each joint
   - Recognise the posture/gesture

Reading and showing a video file

For depth video file it's exactly the same!

Let's do it…

How's depth video like? Why?

Note: when we're going to connect the Kinect sensor and gather color and depth data directly from the sensor, depth data will be different… see slide # 7

First program

1. **Read** data from the hard disk:
   - Color data
   - Depth data
   - Skeleton data

   } gathered with the Kinect sensor

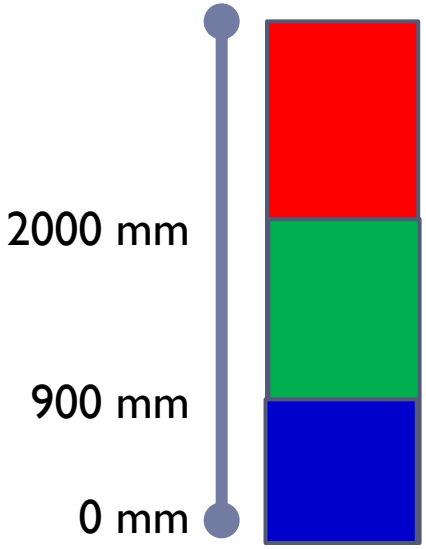
2. **Show data** into 3 different images into MainWindow
3. **Analyse skeleton data** for:
   - Extract the 3D coordinated of each joint
   - Recognise the posture/gesture

For depth data to a video file

Depth data from the Kinect sensor is composed of:

1. the distance (mm) of every pixel from the sensor
2. a label indicating if the pixel belongs to a player

Depth video file is obtained by associating a color to a range of depths, i.e:



We'll see it later on…

Reading skeleton data

While color and depth data are videos, skeleton data are saved in 2 different text files:

- `skeleton3D.txt`
  3D joints coordinates. [metres]

- `skeleton_norm.txt`
  2D joints coordinates projected on the sensor plane and normalised with respect to the Color Image resolution. For visualizing the skeleton on the image. [pixels]

Note: when we're going to connect the Kinect sensor and gather skeleton data directly from the sensor, joint data will be 3D!

Reading and drawing skeleton data

Back in the Window_Loaded method:

```
//3D skeleton data
sr1 = new StreamReader(@"C:\Miki\skeleton3D.txt");

//2D skeleton data
sr2 = new StreamReader(@"C:\Miki\skeleton_norm.txt");

//background image for skeleton visualization (black 640x480
bmp image)
bmp = new Bitmap(@"C:\Miki\black.bmp", true);

background = CreateBitmapSourceFromBitmap(bmp);
```

Reading and drawing skeleton data

Back in the My_Timer_Tick method:

```
//read 3D skeleton data from file
dataFile1 = ReadTxtFile(sr1);
skeleton3D = Read3DSkeletonFromTxt(dataFile1);

//read 2D skeleton data from file
skeleton2D = ReadTxtFile(sr2);

// draw skeleton data on the background image (i.e. black
image)
DrawSkeleton(skeleton2D, background);
```

Reading skeleton data

Where `ReadTxtFile` is the method for reading the text file and put data into an array of double (paste & copy & use it):

```
double[] ReadTxtFile(StreamReader sr) {
            string line = sr.ReadLine();
            char[] delimiters = new char[] { '\t' };
            string[] parts = line.Split(delimiters,
            StringSplitOptions.RemoveEmptyEntries);
            double[] skeleton = new double[parts.Length];
            for (int i = 0; i < parts.Length; i++)
            {
                skeleton[i] = Convert.ToDouble(parts[i]);
            }

            return skeleton;
}
```

Reading skeleton data

Where `Read3DSkeletonFromTxt` is the method for transforming the array of double of the 3D joint coordinates, into the `Skeleton` object (paste & copy & use it):

```
Skeleton Read3DSkeletonFromTxt(double[] dataFile) {
            Skeleton skeleton = new Skeleton();
            SkeletonPoint sp = new SkeletonPoint();
            Joint jt = new Joint();
            int i = 0;
            foreach (Joint joint in skeleton.Joints) {
                sp.X = (float)dataFile[i];
                sp.Y = (float)dataFile[i + 1];
                sp.Z = (float)dataFile[i + 2];
                jt = skeleton.Joints[joint.JointType];
                jt.Position = sp;
                // put joint back into skeleton
                skeleton.Joints[joint.JointType] = jt;
                i = i + 3;              }
            return skeleton;         }
```

Drawing skeleton data

Where `DrawSkeleton` is the method for drawing the skeleton on the background image (paste & copy & use it):

```
void DrawSkeleton(double[] skeleton, BitmapSource colorBmp)        {

DrawingGroup dGroup = new DrawingGroup();
            DrawingContext dc = dGroup.Open();
            dc.DrawImage(colorBmp, new Rect(0, 0, 640, 480));

            //drawing lines and ellipses on the image
            Double radius = 3;
            dc.DrawLine(new System.Windows.Media.Pen(new SolidColorBrush(Colors.Red), 5), new System.Windows.Point(skeleton[16], skeleton[17]), new System.Windows.Point(skeleton[28], skeleton[29]));
            dc.DrawLine(new System.Windows.Media.Pen(new SolidColorBrush(Colors.Red), 5), new System.Windows.Point(skeleton[28], skeleton[29]), new System.Windows.Point(skeleton[30], skeleton[31]));
            dc.DrawLine(new System.Windows.Media.Pen(new SolidColorBrush(Colors.Red), 5), new System.Windows.Point(skeleton[30], skeleton[31]), new System.Windows.Point(skeleton[4], skeleton[5]));
            dc.DrawLine(new System.Windows.Media.Pen(new SolidColorBrush(Colors.Red), 5), new System.Windows.Point(skeleton[4], skeleton[5]), new System.Windows.Point(skeleton[36], skeleton[37]));
            dc.DrawLine(new System.Windows.Media.Pen(new SolidColorBrush(Colors.Red), 5), new System.Windows.Point(skeleton[36], skeleton[37]), new System.Windows.Point(skeleton[12], skeleton[13]));
            dc.DrawLine(new System.Windows.Media.Pen(new SolidColorBrush(Colors.Red), 5), new System.Windows.Point(skeleton[28], skeleton[29]), new System.Windows.Point(skeleton[32], skeleton[33]));
            dc.DrawLine(new System.Windows.Media.Pen(new SolidColorBrush(Colors.Red), 5), new System.Windows.Point(skeleton[32], skeleton[33]), new System.Windows.Point(skeleton[6], skeleton[7]));
            dc.DrawLine(new System.Windows.Media.Pen(new SolidColorBrush(Colors.Red), 5), new System.Windows.Point(skeleton[6], skeleton[7]), new System.Windows.Point(skeleton[38], skeleton[39]));
            dc.DrawLine(new System.Windows.Media.Pen(new SolidColorBrush(Colors.Red), 5), new System.Windows.Point(skeleton[38], skeleton[39]), new System.Windows.Point(skeleton[14], skeleton[15]));
            dc.DrawLine(new System.Windows.Media.Pen(new SolidColorBrush(Colors.Red), 5), new System.Windows.Point(skeleton[28], skeleton[29]), new System.Windows.Point(skeleton[34], skeleton[35]));
            dc.DrawLine(new System.Windows.Media.Pen(new SolidColorBrush(Colors.Red), 5), new System.Windows.Point(skeleton[34], skeleton[35]), new System.Windows.Point(skeleton[18], skeleton[19]));
            dc.DrawLine(new System.Windows.Media.Pen(new SolidColorBrush(Colors.Red), 5), new System.Windows.Point(skeleton[18], skeleton[19]), new System.Windows.Point(skeleton[20], skeleton[21]));
            dc.DrawLine(new System.Windows.Media.Pen(new SolidColorBrush(Colors.Red), 5), new System.Windows.Point(skeleton[20], skeleton[21]), new System.Windows.Point(skeleton[24], skeleton[25]));
            dc.DrawLine(new System.Windows.Media.Pen(new SolidColorBrush(Colors.Red), 5), new System.Windows.Point(skeleton[24], skeleton[25]), new System.Windows.Point(skeleton[0], skeleton[1]));
            dc.DrawLine(new System.Windows.Media.Pen(new SolidColorBrush(Colors.Red), 5), new System.Windows.Point(skeleton[0], skeleton[1]), new System.Windows.Point(skeleton[8], skeleton[9]));
            dc.DrawLine(new System.Windows.Media.Pen(new SolidColorBrush(Colors.Red), 5), new System.Windows.Point(skeleton[18], skeleton[19]), new System.Windows.Point(skeleton[22], skeleton[23]));
            dc.DrawLine(new System.Windows.Media.Pen(new SolidColorBrush(Colors.Red), 5), new System.Windows.Point(skeleton[22], skeleton[23]), new System.Windows.Point(skeleton[26], skeleton[27]));
            dc.DrawLine(new System.Windows.Media.Pen(new SolidColorBrush(Colors.Red), 5), new System.Windows.Point(skeleton[26], skeleton[27]), new System.Windows.Point(skeleton[2], skeleton[3]));
            dc.DrawLine(new System.Windows.Media.Pen(new SolidColorBrush(Colors.Red), 5), new System.Windows.Point(skeleton[2], skeleton[3]), new System.Windows.Point(skeleton[10], skeleton[11]));

            for (int i = 0; i < (skeleton.Length); i=i+2)
            {
                dc.DrawEllipse(null, new System.Windows.Media.Pen(new SolidColorBrush(Colors.Yellow), 10), new System.Windows.Point(skeleton[i], skeleton[i+1]), radius, radius);
            }

            dc.Close();
            DrawingImage dImageSource = new DrawingImage(dGroup);
            image3.Source = dImageSource;
        }
```

Reading and drawing skeleton data

And of course we need to define the following:

```
StreamReader sr1;
StreamReader sr2;
Bitmap bmp;
BitmapSource background;

Skeleton skeleton3D = new Skeleton();
double[] dataFile1;
double[] skeleton2D;
```

First program

1. **Read** data from the hard disk:
   - Color data
   - Depth data
   - Skeleton data

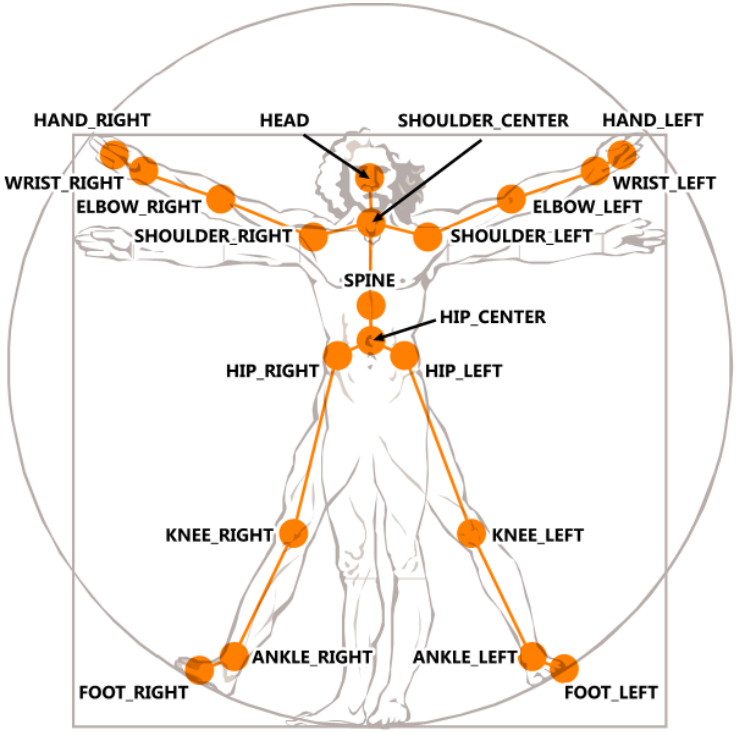
   } gathered with the Kinect sensor

2. **Show data** into 3 different images into MainWindow
3. **Analyse skeleton data** for:
   - Extract the 3D coordinated of each joint
   - Recognise the posture/gesture

Analyse 3D skeleton data

How to extract the x,y,z coordinates (`float`) of each joint:

```
skeleton3D.Joints[JointType.AnkleLeft].Position.X
skeleton3D.Joints[JointType.AnkleLeft].Position.Y
skeleton3D.Joints[JointType.AnkleLeft].Position.Z
```



Analyse skeleton data

In the folder you find the following

- Postures:
  1. None (hands close to the body)
  2. LeftHandOverHead
  3. RightHandOverHead

- Gestures:
  1. LeftHello
  2. RightHello
  3. Help
  4. GoStraight

Write the method for recognising each posture/gesture.

In real-time with the Kinect sensor…

1. **Read** data from the Kinect sensor:
   - Color data
   - Depth data
   - Skeleton data

2. **Show data** into 3 different images into MainWindow
3. **Analyse skeleton data** for:
   - Extract the 3D coordinated of each joint
   - Recognise the posture/gesture

Up to now

1. Image and video processing
2. Posture and gesture recognition
3. Kinect motion sensor and full body skeleton tracking

BUT in this situation

…the skeleton tracking is not working!

Next step

4. **Use image processing** for extracting information on the posture/gestures executed by a subject in the seated position
5. Use the potentiality of Kinect, i.e. **depth information**!