Application of Microsoft Kinect in controlling computer Keyboard and Mouse input

Technologies to Reduce the Access Barrier in Human Computer Interaction
Erasmus Intensive Programme 2011-1-FI1-ERA10-06251
IP2012, Zaragoza, Spain 19.3. – 30.3.2012
Antti Salopuro
Lahti University of Applied Sciences, Finland

LAHDEN AMMATTIKORKEAKOULU
Lahti University of Applied Sciences

Education and Culture
Lifelong Learning Programme
ERASMUS

Contents

1. Kinect development system setup
2. Application development with Kinect
3. Controlling the mouse and keyboard with Kinect joint positions

PART I

KINECT DEVELOPMENT SYSTEM SETUP

Antti Salopuro
Lahti University of Applied Sciences, Finland

Kinect setup

- Software libraries required
- System setup
- Application setup
  - Namespaces required

Antti Salopuro
Lahti University of Applied Sciences, Finland

Software libraries required

- XNA 4.0 (http://www.microsoft.com/download/en/details.aspx?id=23714)
- MS Kinect for Windows SDK v1 (http://www.microsoft.com/en-us/kinectforwindows/develop/)
- Coding4Fun Kinect Toolkit (http://channel9.msdn.com/coding4fun/projects/Coding4Fun-Kinect-Toolkit)
  - scaling joint positions to some given frame, for example screen
  - transferring the image frame to wpf ImageSource control format
- Input Simulator (http://inputsimulator.codeplex.com/)
  - for sending keyboard commands to the computer
  - Windows forms provide also a method SendKeys, but it only simulates text entry, not actual keystrokes
- NativeMethods.cs for commanding the mouse (https://github.com/jera/lazyconsumer/blob/master/NativeMethods.cs)

System and application setup

- Install
  - XNA 4.0
  - MS Kinect for Windows SDK
- Create a new project in Visual Studio 2010 as standard C# WPF application
- Add project references to required namespaces
  - Solution Explorer/References/Add Reference/ .NET or Browse
  - Include required namespaces to your classes with `using` directive:

```
using Microsoft.Kinect;
using Coding4Fun.Kinect.Wpf;
using WindowsInput;
using System.Windows.Forms;
using System.Diagnostics;
```

- Add class `NativeMethods.cs` to your project (Solution Explorer)

PART II

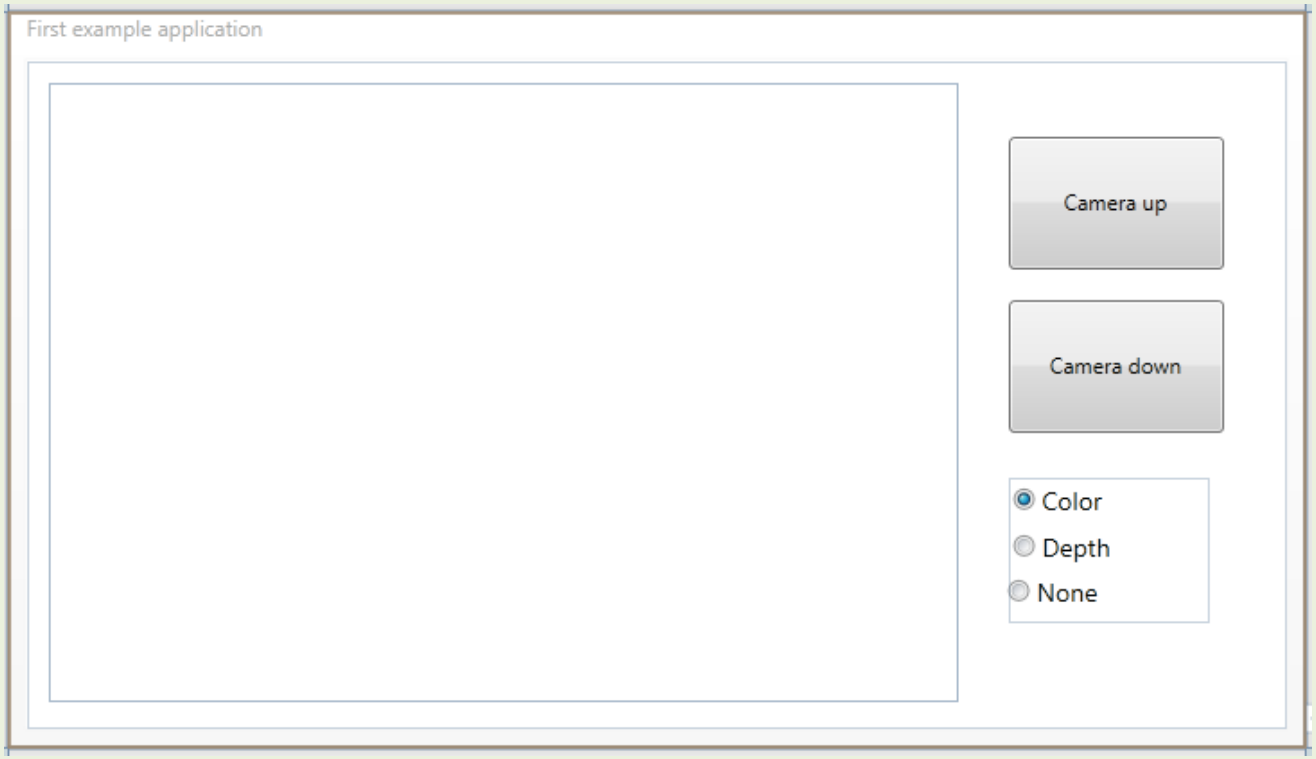
WPF APPLICATION DEVELOPMENT WITH KINECT SDK

Antti Salopuro
Lahti University of Applied Sciences, Finland

Sample application

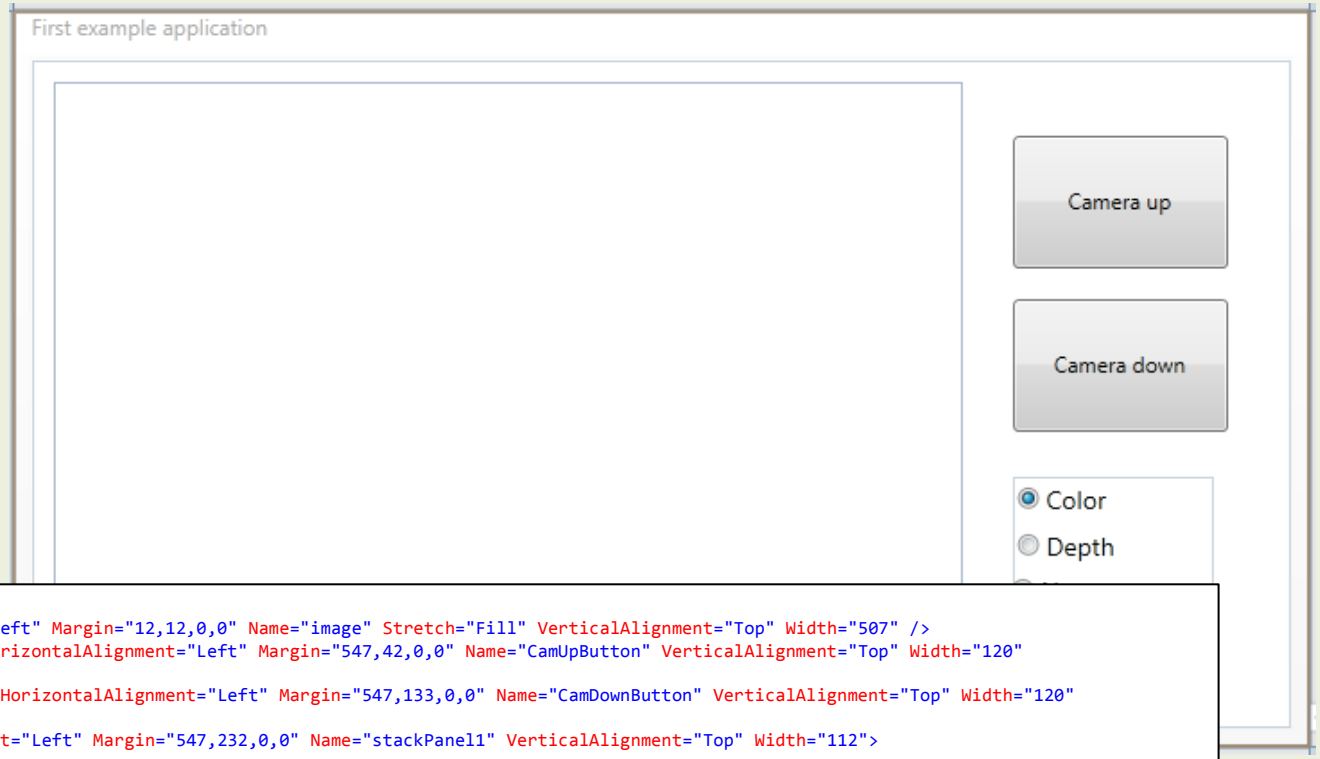
- Controls for adjusting the camera elevation angle
- Image control for showing the color or depth image
- Canvas control for showing two ellipses connected to hand positions
- Radiobuttons to select what is shown
  - Color, depth, only ellipses

First sample application UI



First sample application

Copy - paste the XAML to create the UI




```
<Grid>
       <Image Height="345" HorizontalAlignment="Left" Margin="12,12,0,0" Name="image" Stretch="Fill" VerticalAlignment="Top" Width="507" />
       <Button Content="Camera up" Height="74" HorizontalAlignment="Left" Margin="547,42,0,0" Name="CamUpButton" VerticalAlignment="Top" Width="120"
      Click="CamUpButton_Click" />
       <Button Content="Camera down" Height="74" HorizontalAlignment="Left" Margin="547,133,0,0" Name="CamDownButton" VerticalAlignment="Top" Width="120"
      Click="CamDownButton_Click" />
       <StackPanel Height="81" HorizontalAlignment="Left" Margin="547,232,0,0" Name="stackPanel1" VerticalAlignment="Top" Width="112">
           <StackPanel.BindingGroup>
               <BindingGroup Name="ImageToShowGroup" SharesProposedValues="True" />
           </StackPanel.BindingGroup>
           <RadioButton Content="Color" Height="20" Name="ColorRadioButton" Margin="3" FontSize="14" IsChecked="True" HorizontalAlignment="Left"
      Checked="ColorRadioButton_Checked" />
           <RadioButton Content="Depth" Height="22" Name="DepthRadioButton" Margin="3" FontSize="14" Width="91" HorizontalAlignment="Left"
      Checked="DepthRadioButton_Checked" />
           <RadioButton Content="None" FontSize="14" Height="24" Name="NoneRadioButton" Width="91" HorizontalAlignment="Left" Checked="NoneRadioButton_Checked" />
       </StackPanel>
       <Canvas Height="345" HorizontalAlignment="Left" Margin="12,12,0,0" Name="canvas" VerticalAlignment="Top" Width="507">
       </Canvas>
   </Grid>
```

Global references

```
public partial class MainWindow : Window
    {
        private KinectSensor camDevice;
        private const int skeletonCount = 6;
        private Skeleton[] allSkeletons = new Skeleton[skeletonCount];

        private Ellipse rightEllipse, leftEllipse;

        public MainWindow()
        {
                … …
```

It is useful to define a global reference for the sensor object and for the collection of the skeletons…

and for the two ellipses drawn on hands

Sensor initialisation tasks

- Get access to the sensor
  - `camDevice = KinectSensor.KinectSensors[0];`
- Start device
  - `camDevice.Start();`
- Enable required video streams
  - Color, depth and skeleton streams
- Hook FrameReady events to event handlers

Sensor initialization code

```
private void Window_Loaded(object sender, RoutedEventArgs e)
{
    try
    {
        camDevice = KinectSensor.KinectSensors[0];
    }
    catch (Exception ex)
    {
        System.Windows.MessageBox.Show("Could not find Kinect camera: " + ex.Message);
    }

    camDevice.DepthStream.Enable(DepthImageFormat.Resolution640x480Fps30);
    camDevice.ColorStream.Enable(ColorImageFormat.RgbResolution640x480Fps30);
    camDevice.SkeletonStream.Enable(new TransformSmoothParameters()
                {
                    Smoothing = 0.75f,
                    Correction = 0.5f,
                    Prediction = 0.3f,
                    JitterRadius = 0.5f,
                    MaxDeviationRadius = 0.04f
                });

    camDevice.AllFramesReady += camera_AllFramesReady;
```

Sensor initialization code

```
private void Window_Loaded(object sender, RoutedEventArgs e)
{
    try
    {
        camDevice = KinectSensor.KinectSensors[0];
    }
    catch (Exception ex)
    {
        System.Windows.MessageBox.Show("Could not find Kinect camera: " + ex.Message);
    }

    camDevice.Start();

    camDevice.DepthStream.Enable(DepthImageFormat.Resolution640x480Fps30);
    camDevice.ColorStream.Enable(ColorImageFormat.RgbResolution640x480Fps30);
    camDevice.SkeletonStream.Enable(new TransformSmoothParameters()
                {
                    Smoothing = 0.75f,
                    Correction = 0.5f,
                    Prediction = 0.3f,
                    JitterRadius = 0.5f,
                    MaxDeviationRadius = 0.04f
                });

    camDevice.AllFramesReady += camera_AllFramesReady;
```

Sensor initalisation is best done together with the `Window_Loaded` event handler

Wrap all initialisation statements inside try-catch blocks

Depth and color streams require resolution settings, skeleton stream may take smoothing parameters

Hook an event handler to AllFramesReady event

Sensor resource release

```
private void Window_Closed(object sender, EventArgs e)
{
   camDevice.Stop();
}
```

Sensor resources are released in the Window_Closed event handler

Camera elevation angle adjustment

Event handlers for the Camera up/down buttons

```
private void CamDownButton_Click(object sender, RoutedEventArgs e)
{
    try
    {
        if (camDevice.ElevationAngle > camDevice.MinElevationAngle + 5)
            camDevice.ElevationAngle -= 5;
    }
    catch
    {
        System.Windows.MessageBox.Show("Elevation angle change not succesful");
    }
}
```

Method to add two ellipses on canvas control

```
private void createEllipses()
{
   rightEllipse = new Ellipse();
   canvas.Children.Add(rightEllipse);
   rightEllipse.Height = 25;
   rightEllipse.Width = 25;
   rightEllipse.Fill = Brushes.Aqua;

   leftEllipse = new Ellipse();
   canvas.Children.Add(leftEllipse);
   leftEllipse.Height = 25;
   leftEllipse.Width = 25;
   leftEllipse.Fill = Brushes.PaleVioletRed;
}
```

This method need to be called before processing the corresponding joint data

Event handlers hooked on events

- An event is a message sent by an object to signal the occurrence of an action
- Event handlers can be hooked on these events
  - Event handler hooked to an event is a method run every time the event happens
  - Event handler must have the correct signature, i.e. correct set of input parameters
- Each image stream triggers an event every time a new image frame has been captured

Events raised by image frame objects

- Each different frame triggers an event every time a new frame has been captured
  - `ColorFrameReady`
  - `DepthFrameReady`
  - `SkeletonFrameReady`
- There exists also an event triggered after all different image frames have been renewed
  - `AllFramesReady`
  - In this application we will only implement this event handler

`AllFramesReady` event handler tasks in this application

1. Get access to and plot color or depth image frame on image control
   - Image plotted depends on radio button selection
2. Get access to skeleton frame and one skeleton data in it
3. Get access to joints of both hands of the found skeleton
4. Draw ellipses on both hands (on canvas)
   - The exact position of ellipses depend on the background image (color or depth)

1. Get access to and plot color or depth image frame on image control

```
private void camera_AllFramesReady(object source, AllFramesReadyEventArgs e)
{
    if (DepthRadioButton.IsChecked.Value)
        image.Source = e.OpenDepthImageFrame().ToBitmapSource();
    else
        image.Source = e.OpenColorImageFrame().ToBitmapSource();

        //Code continues…
```

2. Get access to skeleton frame and to one skeleton data in it

```
private void camera_AllFramesReady(object source, AllFramesReadyEventArgs e)
{
    //image source processing was here

    SkeletonFrame skeletonFrame = e.OpenSkeletonFrame();

    if (skeletonFrame != null)
    {
          if ((this.allSkeletons == null) ||
          (this.allSkeletons.Length != skeletonFrame.SkeletonArrayLength))
          {
               this.allSkeletons = new Skeleton[skeletonFrame.SkeletonArrayLength];
          }

          skeletonFrame.CopySkeletonDataTo(this.allSkeletons);


          foreach (Skeleton sd in allSkeletons)
          {
                    if (sd.TrackingState == SkeletonTrackingState.Tracked)
                    {
                        //Now sd will refer to a tracked skeleton
                        //continue processing skeleton data
```

2. Get access to skeleton frame and to one skeleton data in it

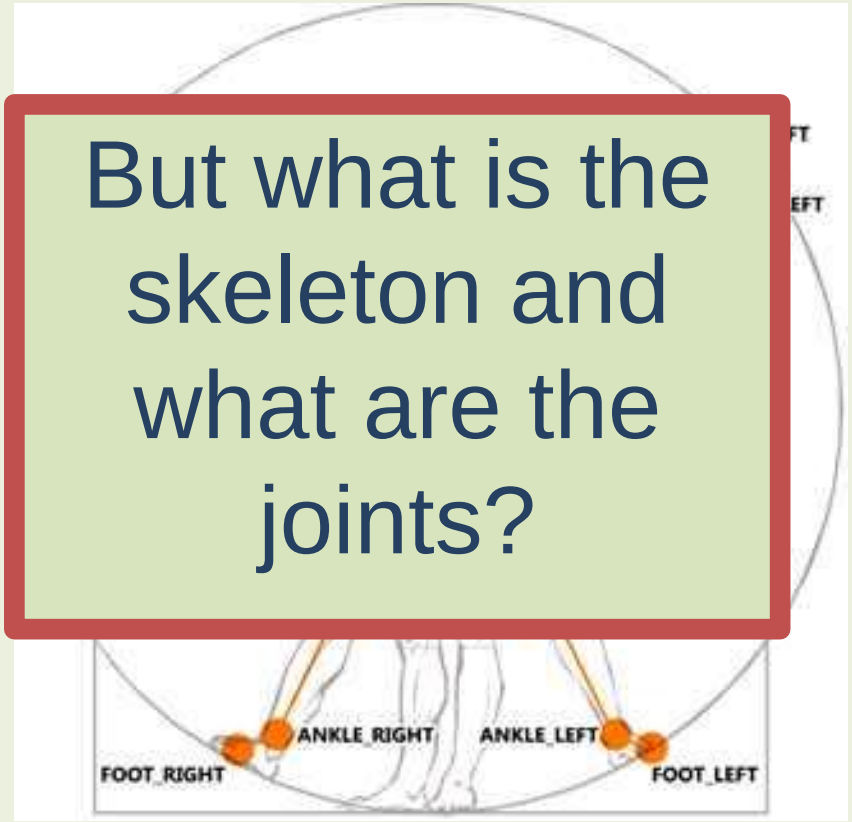
```
private void camera_AllFramesReady(object source, AllFramesReadyEventArgs e)
{
    //image source processing was here

    SkeletonFrame skeletonFrame = e.OpenSkeletonFrame();

    if (skeletonFrame != null)
    {
        if ((this.allSkeletons == null) ||
        (this.allSkeletons.Length != skeletonFrame.SkeletonArrayLength))
        {
            this.allSkeletons = new Skeleton[skeletonFrame.SkeletonArrayLength];
        }

        skeletonFrame.CopySkeletonDataTo(this.allSkeletons);


        foreach (Skeleton sd in allSkeletons)
        {
            if (sd.TrackingState == SkeletonTrackingState.Tracked)
            {
                //Now sd will refer to a tracked skeleton
                //continue processing skeleton data
```

Get access to the skeleton frame

Get access to the skeleton array, this need to be done only once

Search for the first skeleton being tracked

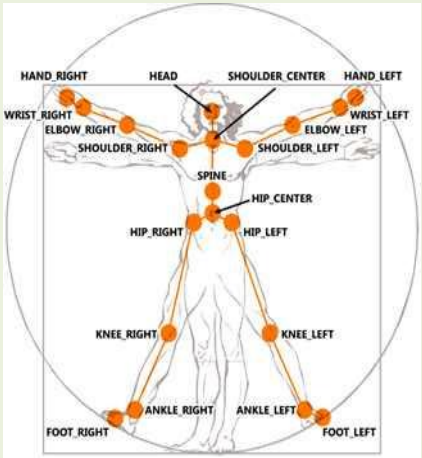
3. Get access to joints of both hands of the found skeleton (sd)



Skeleton array

The `skeletonFrame` object of the sensor includes an array of max 6 `skeleton` objects

```
// global reference to the skeleton array
private const int skeletonCount = 6;
private Skeleton[] allSkeletons = new Skeleton[skeletonCount];
```



`skeletonFrame` object offers the method `CopySkeletonData` for getting access to the skeleton array

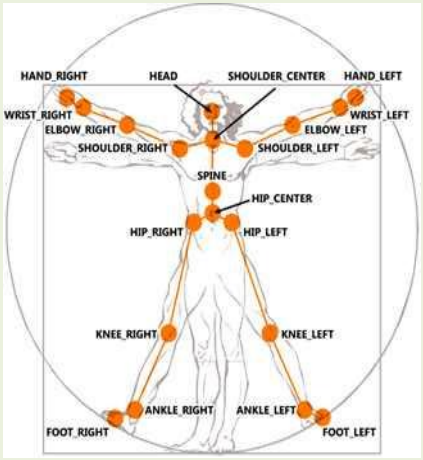
```
// get current skeleton data to the array after
skeletonFrame.CopySkeletonDataTo(this.allSkeletons);
```


Antti Salopuro
Lahti University of Applied Sciences, Finland

Get access to skeleton

Once the skeleton array has been found, the active skeleton can be picked from it, for example, by direct index value…

```
Skeleton sd;
if (allSkeletons[0].TrackingState == SkeletonTrackingState.Tracked)
    sd = allSkeletons[0];
```

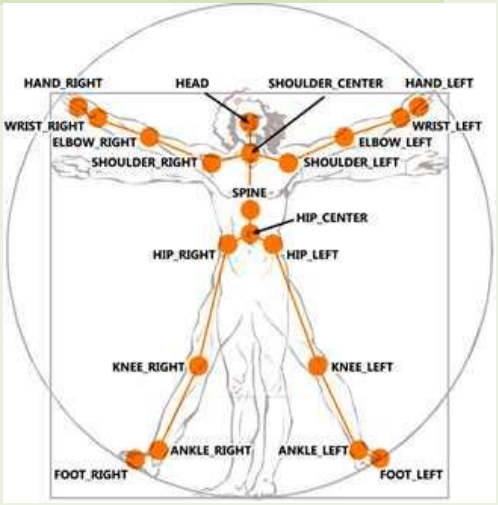


… or by taking the first one found being tracked

```
foreach (Skeleton sd in allSkeletons)
{
   if (sd.TrackingState == SkeletonTrackingState.Tracked)
   {
```

Skeleton class properties

- `ClippedEdges`
  - which parts of the skeleton are clipped by the edges of the frame
- `Position`
  - skeleton position in X-Y-Z space
- `TrackingId`
- `TrackingState`
  - `NotTracked`, `PositionOnly`, `Tracked`
- `Joints`
  - Collection of 20 different body points

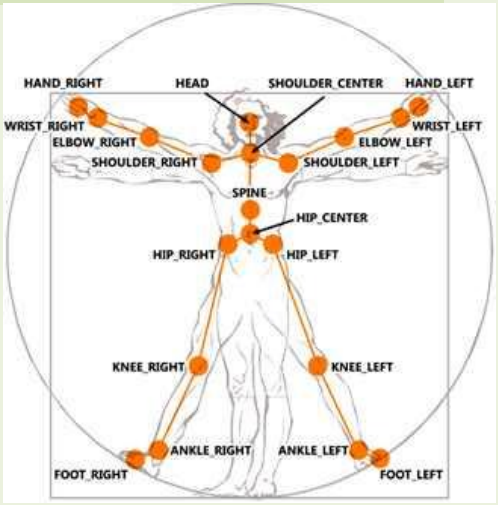


`Joints` is the skeleton property containing the positions of different body parts

Antti Salopuro
Lahti University of Applied Sciences, Finland

Joint class properties

- `JointType`
  - Enum type for indexing the specific joints in the array of joints
- `Position`
  - skeleton position in X-Y-Z space
- `TrackingState`
  - `NotTracked`, `Inferred`, `Tracked`

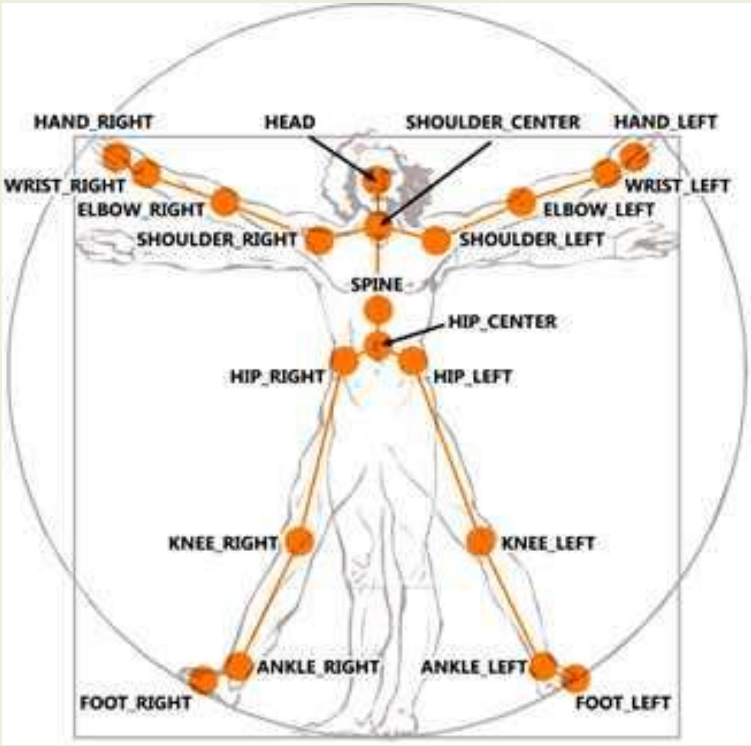


`Inferred` joint is not currently seen by the sensor

Antti Salopuro
Lahti University of Applied Sciences, Finland

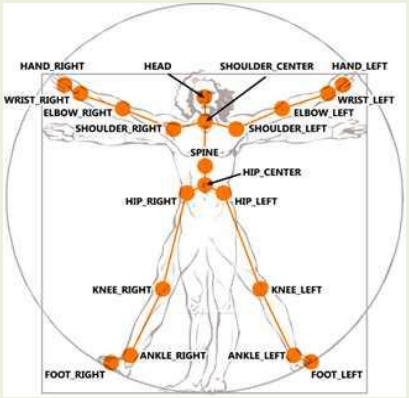
JointType enumeration

Name	Description
AnkleLeft	Left ankle.
AnkleRight	Right ankle.
ElbowLeft	Left elbow.
ElbowRight	Right elbow.
FootLeft	Left foot.
FootRight	Right foot.
HandLeft	Left hand.
HandRight	Right hand.
Head	Head.
HipCenter	Center, between hips.
HipLeft	Left hip.
HipRight	Right hip.
KneeLeft	Left knee.
KneeRight	Right knee.
ShoulderCenter	Center, between shoulders.
ShoulderLeft	Left shoulder.
ShoulderRight	Right shoulder.
Spine	Spine.
WristLeft	Left wrist.
WristRight	Right wrist.



Position property

- Location of the physical part in the human body in 3D space
  - X, Y, Z



X = Horizontal position measured as the distance, in meters from the Kinect along the X Axis.

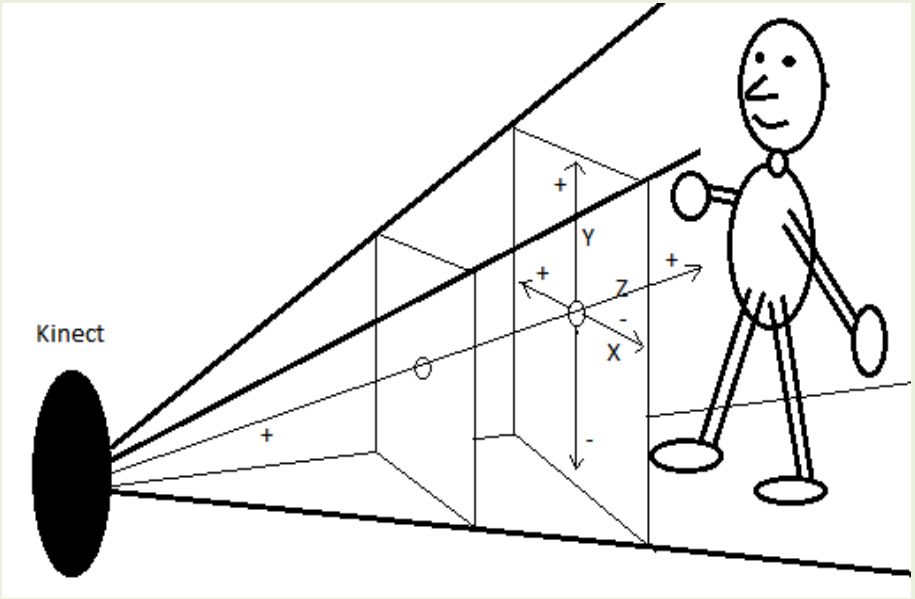
Y = Vertical position measured as the distance, in meters from the Kinect along the Y Axis.

Z = Distance from Kinect measured in meters

Joint position coordinate system

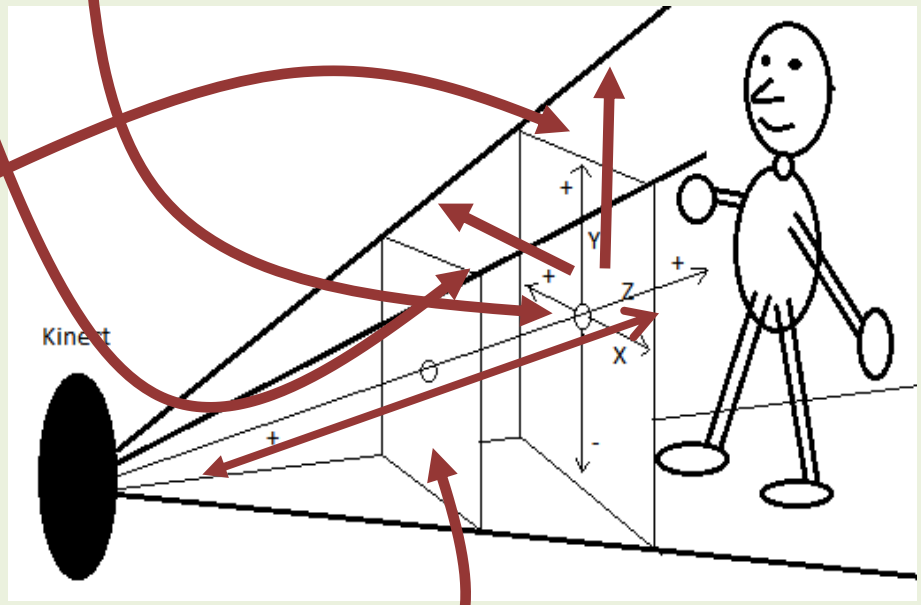
- The origo of the X-Y coordinate system is roughly at the center point of the taken image
- From the camera point of view X coordinate value increases to the left (from human point of view to the right) vertical centerline being the zero axis.

- Correspondingly Y value increases upwards, zero axis being the horizontal centerline.
- Max and Min values of X and Y depend therefore on the human distance from the camera.



Joint position coordinate system

- The origo of the X-Y coordinate system is roughly at the center point of the taken image
- From the camera point of view X coordinate value increases to the left (from human point of view to the right) vertical centerline being the zero axis.
- Correspondingly Y value increases upwards, zero axis being the horizontal centerline.
- Max and Min values of X and Y depend therefore on the human distance from the camera.



3. Get access to joints of both hands of the found skeleton (sd)

```
foreach (Skeleton sd in allSkeletons)
{
    if (sd.TrackingState == SkeletonTrackingState.Tracked)
    {
          Joint LeftHand = sd.Joints[JointType.HandLeft];
          Joint RightHand = sd.Joints[JointType.HandRight];

          //Continue only if both hands are being tracked
          if (RightHand.TrackingState == JointTrackingState.Tracked &&
              LeftHand.TrackingState == JointTrackingState.Tracked)
          {
```

3. Get access to joints of both hands of the found skeleton (sd)

```
foreach (Skeleton sd in allSkeletons)
{
    if (sd.TrackingState == SkeletonTrackingState.Tracked)
    {
        Joint LeftHand = sd.Joints[JointType.HandLeft];
        Joint RightHand = sd.Joints[JointType.HandRight];

        //Continue only if both hands are being tracked
        if (RightHand.TrackingState == JointTrackingState.Tracked &&
            LeftHand.TrackingState == JointTrackingState.Tracked)
        {
```

enum type `JointType` to pick the wished joint

References to both hands

Make sure both hands are being tracked

4. Draw ellipses on both hands (on canvas control)

```
//Continue only if both hands are being tracked
if (RightHand.TrackingState == JointTrackingState.Tracked &&
    LeftHand.TrackingState == JointTrackingState.Tracked)
{

    if (rightEllipse == null || leftEllipse == null)
        createEllipses();

    if (DepthRadioButton.IsChecked.Value)
    {
        plotOnDepthImage(RightHand, DepthImageFormat.Resolution640x480Fps30, rightEllipse, canvas);
        plotOnDepthImage(LeftHand, DepthImageFormat.Resolution640x480Fps30, leftEllipse, canvas);
    }
    else
    {
        plotOnColorImage(RightHand, ColorImageFormat.RgbResolution640x480Fps30, rightEllipse, canvas);
        plotOnColorImage(LeftHand, ColorImageFormat.RgbResolution640x480Fps30, leftEllipse, canvas);
    }
```

4. Draw ellipses on both hands (on canvas control)

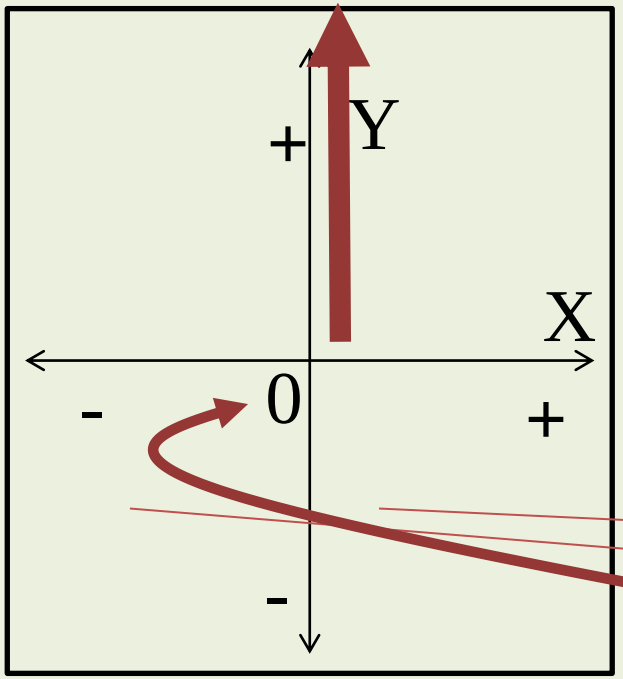
```
//Continue only if both hands are being tracked
if (RightHand.TrackingState == JointTrackingState.Tracked &&
    LeftHand.TrackingState == JointTrackingState.Tracked)
{

    if (rightEllipse == null || leftEllipse == null)
        createEllipses();

    if (DepthRadioButton.IsChecked.Value)
    {
        plotOnDepthImage(RightHand, DepthImageFormat.Resolution640x480Fps30, rightEllipse, canvas);
        plotOnDepthImage(LeftHand, DepthImageFormat.Resolution640x480Fps30, leftEllipse, canvas);
    }
    else
    {
        plotOnColorImage(RightHand, ColorImageFormat.RgbResolution640x480Fps30, rightEllipse, canvas);
        plotOnColorImage(LeftHand, ColorImageFormat.RgbResolution640x480Fps30, leftEllipse, canvas);
    }
```

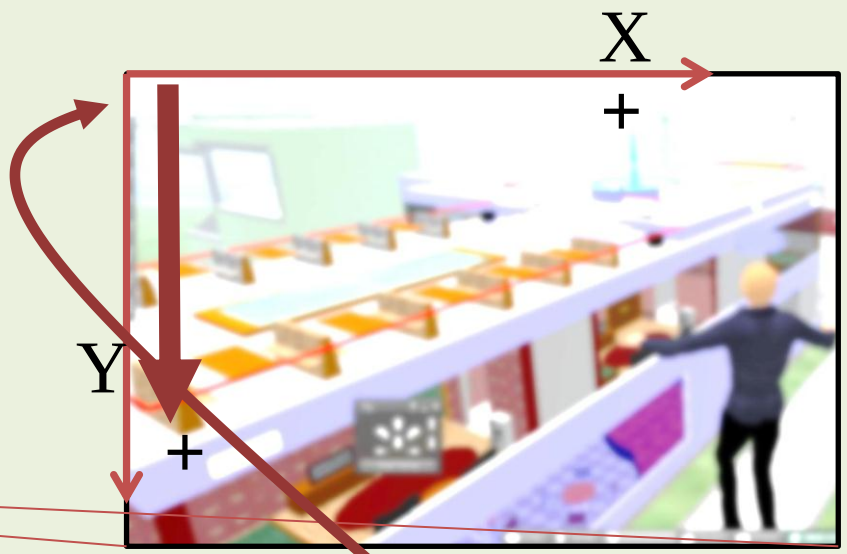
On the very first frame initialise the two ellipses

Select the correct image format by radiobuttons

These plot methods take the given joint position, calculate the corresponding location on the target image and draw the given UI – element on canvas… BUT HOW?

Coordinate systems of joint position and UI – control are different





Coordinate Y - axis directions are opposite between joint image and target UI - component

Origo points positioned in a different way as well

Mapping joint position on image

Sensor object has 2 methods to map position from joint coordinates to pixel coordinates of images or UI - panels

```
MapSkeletonPointToColor(SkeletonPoint, ColorImageFormat)
MapSkeletonPointToDepth(SkeletonPoint, DepthImageFormat)
```

In addition, there exists corresponding method for mapping from depth to skeleton point and one more from depth to color images

```
MapDepthToSkeletonPoint
MapDepthToColorImagePoint
```

Rescaling from color (or depth) image to UI – element (canvas)

COLOR IMAGE

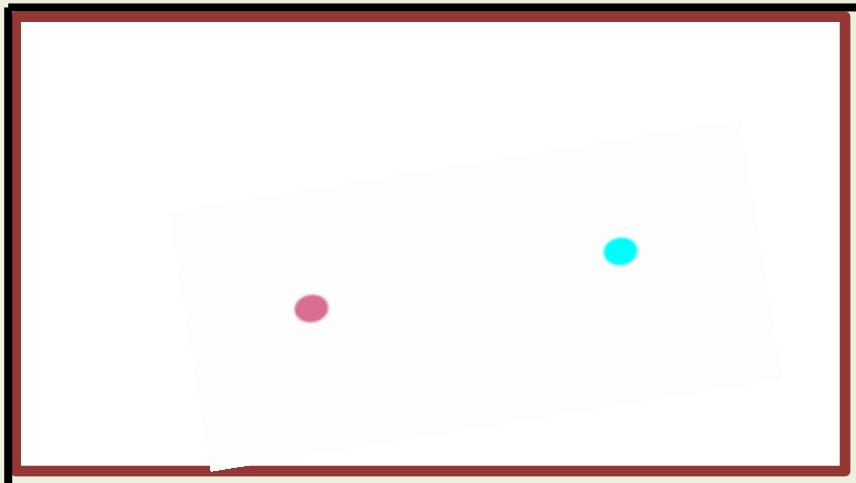
640

480

By default, the source and target rectangles are not of the same sizes

CANVAS

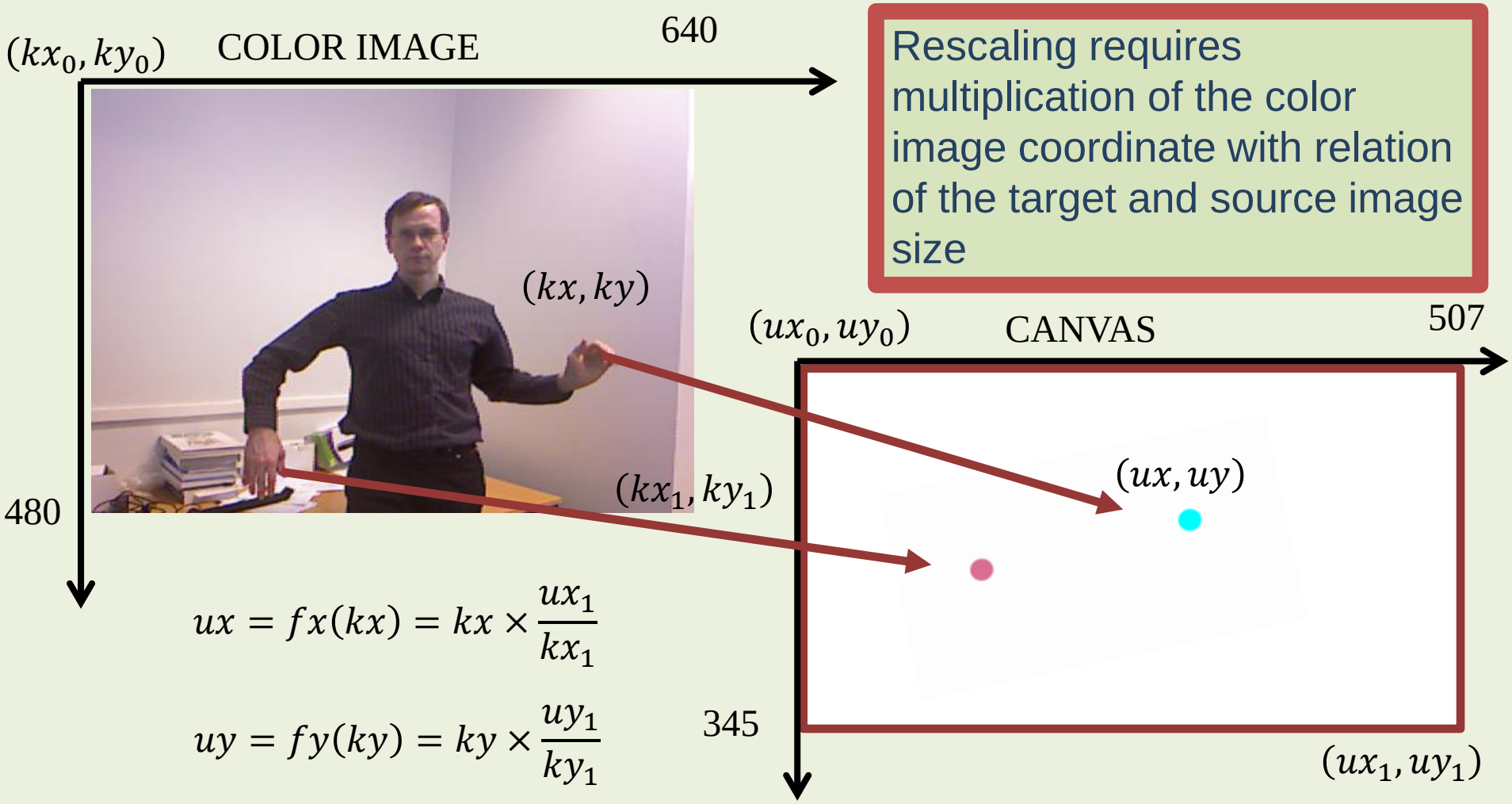
507



345

Antti Salopuro
Lahti University of Applied Sciences, Finland

Rescaling from color (or depth) image to UI – element (canvas)



Rescaling requires multiplication of the color image coordinate with relation of the target and source image size

$$ux = fx(kx) = kx \times \frac{ux_1}{kx_1}$$

$$uy = fy(ky) = ky \times \frac{uy_1}{ky_1}$$

Antti Salopuro
Lahti University of Applied Sciences, Finland

Plotting UI element on a canvas and on a position of a joint

```
//Remember the call function to draw ellipse on right hand joint (see previous slide)
plotOnColorImage(RightHand, //joint
         ColorImageFormat.RgbResolution640x480Fps30, //resolution of the image
         rightEllipse, //UI element
         canvas); //target UI canvas
```

```
//Function implementation
private void plotOnColorImage(Joint myJoint, ColorImageFormat myFormat, UIElement myObject, Canvas
    tgtCanvas)
{
    //Transform the joint position from skeleton coordinates to color image position
    ColorImagePoint colP = camDevice.MapSkeletonPointToColor(myJoint.Position, myFormat);

    //Define the UI element (ellipse) top left corner position inside the canvas
    Canvas.SetTop(myObject, (double)colP.Y / camDevice.ColorStream.FrameHeight * tgtCanvas.Height -
    myObject.RenderSize.Height / 2);

    Canvas.SetLeft(myObject, (double)colP.X / camDevice.ColorStream.FrameWidth * tgtCanvas.Width -
    myObject.RenderSize.Width / 2);
}
```

Plotting UI element on a canvas and on a position of a joint

```
//Function implementation
private void plotOnColorImage(Joint myJoint, ColorImageFormat myFormat, UIElement myObject, Canvas
    tgtCanvas)
{
    //Transform the joint position from skeleton coordinates to color image position
    ColorImagePoint colP = camDevice.MapSkeletonPointToColor(myJoint.Position, myFormat);

    //Define the UI element (ellipse) top left corner position inside the canvas
    Canvas.SetTop(myObject, (double)colP.Y / camDevice.ColorStream.FrameHeight * tgtCanvas.Height -
    myObject.RenderSize.Height / 2);

    Canvas.SetLeft(myObject, (double)colP.X / camDevice.ColorStream.FrameWidth * tgtCanvas.Width -
    myObject.RenderSize.Width / 2);
}
```

First, map the skeleton point on color image

Second, rescale the color image point to the canvas point

ScaleTo method

The Coding4Fun extension for Kinect SDK offers also a method `ScaleTo` for scaling directly from joint position into a given rectangle size

```
Joint colP = myJoint.ScaleTo((int)tgtCanvas.Width,
                             (int)tgtCanvas.Height);

Canvas.SetTop(myObject, (double)colP.Position.Y);

Canvas.SetLeft(myObject, (double)colP.Position.X);
```

Not directly applicable when plotting on sensor images while the skeleton position does not quite match with real image positions – offset remains – but very useful when we start playing with mouse!!

PART III

CONTROLLING THE MOUSE AND KEYBOARD WITH KINECT JOINT POSITIONS

Antti Salopuro
Lahti University of Applied Sciences, Finland

Contents

- Technology
  - Tools required
  - Programmatic control of mouse
  - Programmatic control of keyboard
  - Programmatic control of active application
  - Joint position scaling to screen size for mouse control
- Gestures
  - Absolute X – Y positions in gesture control
  - Relative X – Y positions in gesture control
  - Relaxation of body
  - Depth dimension in gesture control

PART IIIa

CONTROLLING THE MOUSE AND KEYBOARD WITH KINECT JOINT POSITIONS: TECHNOLOGY

Antti Salopuro
Lahti University of Applied Sciences, Finland

Tools required

- Kinect SDK, XNA 4.0, Visual Studio 2010, Kinect sensor
- Input Simulator (http://inputsimulator.codeplex.com/)
  - for sending keyboard commands to the computer
  - Windows forms provide also a method SendKeys, but it only simulates text entry, not actual keystrokes
- NativeMethods.cs for commanding the mouse (https://github.com/jera/lazyconsumer/blob/master/NativeMethods.cs)
  - Include the namespace in the project

```
using NativeMethods;
```

- Some additional references to `user32.dll` native methods to get access to System resources

Programmatic control of the mouse

- In NativeMethods.cs there is method `SendMouseInput` for sending commands to the mouse
- Method takes the mouse position (X and Y coordinates), screen size and mouse button press as parameters
- Example: set position at pixel `(x,y) = (245,334)`, no mouse button pressed

```
NativeMethods.SendMouseInput(245,
                             334,
                             (int)SystemParameters.PrimaryScreenWidth,
                             (int)SystemParameters.PrimaryScreenHeight,
                             false);
```

Programmatic control of keyboard

- `SendKeys` of namespace `System.Windows.Forms`
  - Two methods available
    - Send() , only applicable with Windows messaging
    - SendWait()
  - Documentation at: http://msdn.microsoft.com/en-us/library/system.windows.forms.sendkeys.aspx
- `InputSimulator` tool provided by an open source project
  - Several different nethods available
  - Documentation and download at: http://inputsimulator.codeplex.com/

SendKeys

- Send()
  - Sends the key and continues
  - Application must handle Windows messaging
- SendWait()
  - Sends the key and waits until the message has been processed
- Both send methods take a string or the key(s) as parameter

SendKeys parameters

```
SendKeys.SendWait("Kinect");

SendKeys.SendWait(“{ENTER}")
// 10 subsequent Enters:
SendKeys.SendWait("{ENTER 10}");
SendKeys.SendWait("{DOWN}");

// CTRL + C (copy)
SendKeys.SendWait(“^C")
// CTRL + ALT + delete
SendKeys.SendWait(“^%{DEL}");
// SHIFT + E + C
SendKeys.SendWait(“+EC");
```

Send plain string

Buttons with no symbol showing on screen, see documentation for keys

Control (^), Alt (%) and Shift (+) characters coupled with one or more other characters

InputSimulator

- 5 different methods for controlling keyboard
  - `SimulateTextEntry` for plain text string entry
  - `SimulateKeyPress` for pressing single key
  - `SimulateKeyDown` for holding single key down
  - `SimulateKeyUp` for lifting single key up
  - `SimulateModifiedKeyStroke` for special keys and combined keys

InputSimulator method parameters

```
InputSimulator.SimulateTextEntry("Kinect");
```

Send plain string

```
InputSimulator.SimulateKeyDown(VirtualKeyCode.SHIFT);
InputSimulator.SimulateKeyPress(VirtualKeyCode.VK_K);
InputSimulator.SimulateKeyPress(VirtualKeyCode.VK_I);
InputSimulator.SimulateKeyPress(VirtualKeyCode.VK_N);
InputSimulator.SimulateKeyUp(VirtualKeyCode.SHIFT);
InputSimulator.SimulateKeyPress(VirtualKeyCode.VK_E);
InputSimulator.SimulateKeyPress(VirtualKeyCode.VK_C);
InputSimulator.SimulateKeyPress(VirtualKeyCode.VK_T);
```

Send single key presses

```
// CTRL + C (copy)
InputSimulator.SimulateModifiedKeyStroke(
                    VirtualKeyCode.CONTROL,
                    VirtualKeyCode.VK_C);

// CTRL + ALT + C + K
InputSimulator.SimulateModifiedKeyStroke(
          new[] {   VirtualKeyCode.CONTROL,
                    VirtualKeyCode.ALT },
          new[] {   VirtualKeyCode.VK_C,
                    VirtualKeyCode.VK_K });
```

Control (^), Alt (%) and Shift (+) i.e. characters combined with one or more other characters

Programmatic control of currently active application

- For safety reasons it is necessary to restrict the gesture commands to specific applications
- Namespace `System.Diagnostics` provides method `Process.GetProcesses()` for retrieving a list of currently running processes
- In `user32.dll` there exists methods that provide control over foreground application, for example
  - `GetForegroundWindow()`
    - Get access to foreground window properties
  - `SetForegroundWindow()`
    - Set given application on foreground

Importing the user32.dll methods

Add the following code in your project, for example in NativeMethods.cs class file

```
[DllImport("user32.dll", CharSet = CharSet.Auto, ExactSpelling = true)]
public static extern IntPtr GetForegroundWindow();

[DllImport("user32.dll")]
static extern int GetWindowText(IntPtr hWnd, StringBuilder text, int count);

public static string GetActiveWindowTitle()
{
    const int nChars = 256;
    IntPtr handle = IntPtr.Zero;
    StringBuilder Buff = new StringBuilder(nChars);
    handle = GetForegroundWindow();
    if (GetWindowText(handle, Buff, nChars) > 0)
    {
        return Buff.ToString();
    }
    return null;
}

[DllImport("user32.dll")]
public static extern bool
SetForegroundWindow(IntPtr hWnd);
```

For getting access to foreground window properties

For changing the foreground window

Select foreground application window, Notepad - example

```
//Retrieve list of running processes
Process[] pList = Process.GetProcesses();
foreach (Process pr in pList)
{
    if (pr.MainWindowTitle.StartsWith("Untitled - Notepad"))
    {       //when first Notepad found set it on foreground
        NativeMethods.SetForegroundWindow(pr.MainWindowHandle);
            //and terminate for-each loop
        return;
    }
}

//Make sure that Notepad is active
if (string.Compare(NativeMethods.GetActiveWindowTitle(), "Untitled - Notepad") == 0)
{
            //Write and copy – paste on Notepad
           InputSimulator.SimulateTextEntry("Kinect");
           SendKeys.SendWait("{HOME}");
           SendKeys.SendWait("+{END}");
           SendKeys.SendWait("^C");
           SendKeys.SendWait("{DOWN}");
           SendKeys.SendWait("{ENTER}");
           SendKeys.SendWait("^V");
}
//Return to original (this) application
this.Activate();
```

Open an empty Notepad application, compile and execute this application

Joint position scaling to screen for mouse control

- Method `ScaleTo` of `Joint` object in the Coding4Fun extension on Kinect SDK
- Two overloads: first one takes only screen size parameters (width and height), second one takes also scaling factors
- Scaling factors define the relative size of the source image (skeleton image) window mapped on whole target screen
  - To cover the whole screen the hand does not necessarily have to cover the whole image area

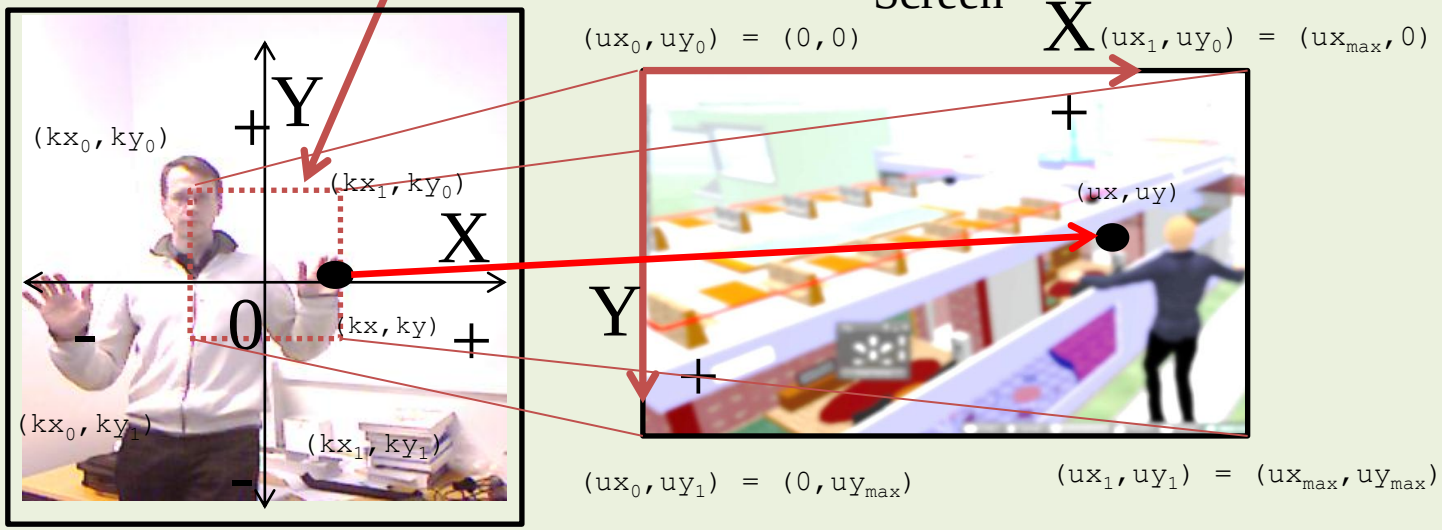
ScaleTo: Scaling factors

```
Joint rhSc = RightHand.ScaleTo(  (int)SystemParameters.PrimaryScreenWidth,
                                 (int)SystemParameters.PrimaryScreenHeight,
                                 0.2f, //X – direction scaling factor
                                 0.2f); //Y – direction scaling factor
```

Area easily reachable with the right hand

Skeleton image

Screen



Example: mouse control

```
bool mClick = (LeftHand.Position.Y > 0.0);

Joint rhSc = RightHand.ScaleTo((int)SystemParameters.PrimaryScreenWidth,
                               (int)SystemParameters.PrimaryScreenHeight,
                               0.2f,
                               0.2f);

NativeMethods.SendMouseInput((int)rhSc.Position.X,
                             (int)rhSc.Position.Y,
                             (int)SystemParameters.PrimaryScreenWidth,
                             (int)SystemParameters.PrimaryScreenHeight,
                             mClick);
```

Left hand above X – axis equals mouse click, right hand gives coordinates for mouse cursor

PART III/b

CONTROLLING THE MOUSE AND KEYBOARD WITH KINECT JOINT POSITIONS: GESTURES

Antti Salopuro
Lahti University of Applied Sciences, Finland

Selecting practical set of gestures

- Combining Kinect joint static positions to keyboard commands is possible by setting some active range(s) for one or more joints for activating the command
- Technically this is simple with the tools described in previous chapters
- Difficulty lies in the selection of the set of gestures in case the number of different commands is higher than just a few
- Moreover, at a relaxed position of the body parts, the gesture control should not send any commands to the system
  - The user should have possibility to relax at any time without a possible hazard to the system or applications

Naval semaphores example

Naval semaphores are signals sent by the positions of two arms emphazised with flags used in communication between, for example, two distant naval ships

Word ”Kinect” with naval semaphores

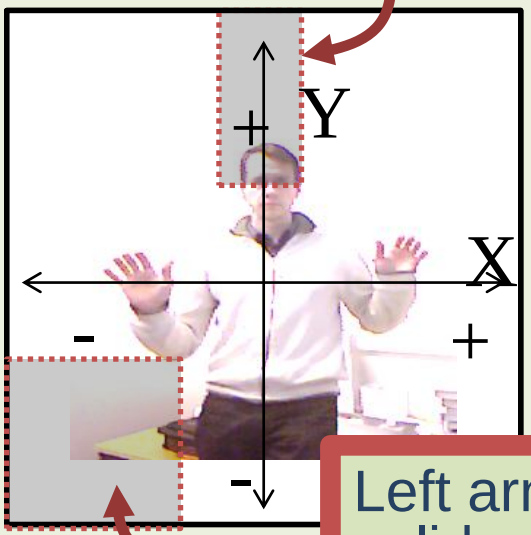
Both hands may have 8 different positions (not all applicable)

Approach 1: absolute position

- Naval semaphores is an example of a set of gestures applicable for computer control
  - Robust set, while it has been developed such that two codes are not easily mixed
- Requires only detection of the X – Y positions of the joints of both hands
- Definition of a gesture requires, for both hands, a specification of valid ranges where the key is activated
  - min and max values in both X and Y dimension

Example: definition of semaphore ’K’

Right arm up valid range

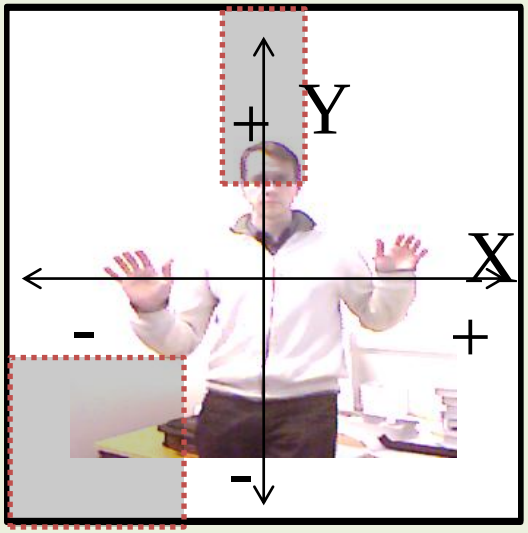


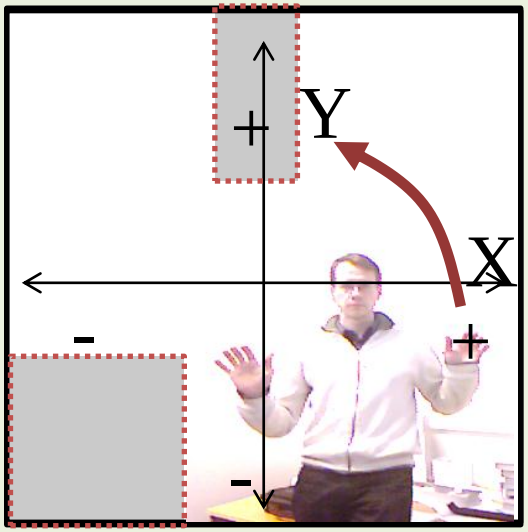

```
Joint RightHand = sd.Joints[JointType.HandRight];
Joint LeftHand = sd.Joints[JointType.HandLeft];

float k_XThresh_Left = -0.3f; //30 cm away from vertical image centerline
float k_YThresh_Left = -0.3f; //30 cm below horizontal image centerline
float k_XThresh_Right_right = 0.1f;
float k_XThresh_Right_left = -0.1f;//20 cm window around image centerline
float k_YThresh_Right = 0.3f;//30cm above horizontal image centerline

bool isK = ((LeftHand.Position.X < k_XThresh_Left)&&
            (LeftHand.Position.Y < k_YThresh_Left)&&
            (RightHand.Position.X > k_XThresh_Right_left)&&
            (RightHand.Position.X < k_XThresh_Right_right)&&
            (RightHand.Position.Y > k_YThresh_Right));
```

Left arm on low left valid range

Problem with absolute position

Application of absolute positions is practical only when the user stands in the middle of the image or close to it





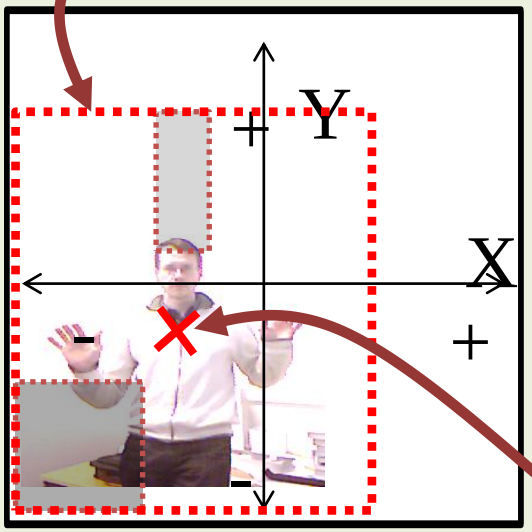
Antti Salopuro
Lahti University of Applied Sciences, Finland

Approach 2: relative position

- Relative position of joints means selecting a reference point from the human body and appying the joint position relative to the position of the reference point
- A valid reference point can be something that statically lies in the middle of the body
  - For example shoulder center or head do not deviate much from their positions
- Valid reference point can also be some point in the body that may change the position significantly
  - For example second hand: compare the vertical or horizontal position of both hands
  - Left hand on right means that arms are crossed

Example: definition of semaphore ’K’

ShoulderCenter
centered source frame




```
Joint RightHand = sd.Joints[JointType.HandRight];
Joint LeftHand = sd.Joints[JointType.HandLeft];
Joint Center = sd.Joints[JointType.ShoulderCenter];

float k_XThresh_Left = -0.3f; //30 cm away from vertical body centerline
float k_YThresh_Left = -0.3f; //30 cm below horizontal body centerline
float k_XThresh_Right_right = 0.1f;
float k_XThresh_Right_left = -0.1f;//20 cm window around body centerline
float k_YThresh_Right = 0.3f;//30cm above horizontal body centerline

bool isK = ((LeftHand.Position.X < Center.Position.X + k_XThresh_Left)&&
            (LeftHand.Position.Y < Center.Position.Y + k_YThresh_Left) &&
            (RightHand.Position.X > Center.Position.X + k_XThresh_Right_left) &&
            (RightHand.Position.X < Center.Position.X + k_XThresh_Right_right) &&
            (RightHand.Position.Y > Center.Position.Y + k_YThresh_Right));
```

New reference point in
the center of the body

Mouse and keyboard control in real applications

- When mouse is operated with one hand, both hands can not anymore be applied for keyboard commands as in the previous semaphore examples
- Quite often, especially in 3D - games, it is also necessary to control, for example, arrow buttons in addition to the mouse position and mouse click - and still be able to operate some extra commands
  - Should we need extra hands?
- Further, the body should be allowed to get into a relaxed position where no muscle need to be tensed without risking the application or system hazard

Relaxation of body parts

- If mouse position is read only from the X – Y position of the hand, there is no place for the hand to be relaxed
  - Hand hanging relaxed would make the mouse point in right (or left) lower corner of the screen
- One solution would be to activate the mouse only if, for example, the second hand is being raised up
  - Lowering both hands will disconnect gesture from mouse allowing body to rest

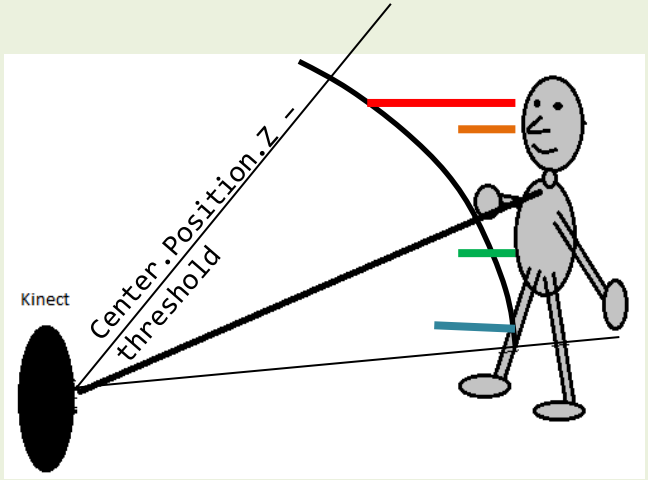
But we needed the second hand for other purposes!!!

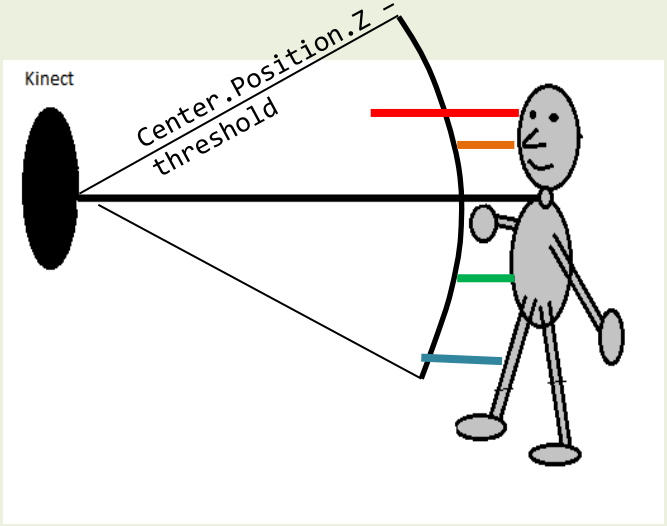
Z - dimension

- Z – dimension of the Joint position provides a solution for allowing a natural relaxation for mouse hand
- For example, activate mouse only if the hand is pushed forward by, say, 30 cm
  - Pulling hand towards the body releases mouse
- This is easy to implement by applying a body reference point in Z – dimension
  - For example `ShoulderCenter` again

```
float mouseActiveThreshold = 0.3f; //30 cm
Joint Center = sd.Joints[JointType.ShoulderCenter];

//Mouse activated only if right hand is far enough from the body
bool mouseOn = (RightHand.Position.Z < (Center.Position.Z - mouseActiveThreshold));
```

Sensor location effect on Z – dimension





If sensor is positioned low or high makes big difference on Z – directional position measure while the measure is taken from sensor point of view

Cancelling the sensor location effect

- If active range of a Joint in a gesture is short in Y – direction and sensor is far, effect is small if body reference point is close to the active Joint
  - Can be omitted
- If sensor location is not know in advance, prepare for cancelling the effect
  - Calibration?
  - Dynamically select the body reference point either from Head, Shoulder or Hip according to the hand height

Sensor location calibration

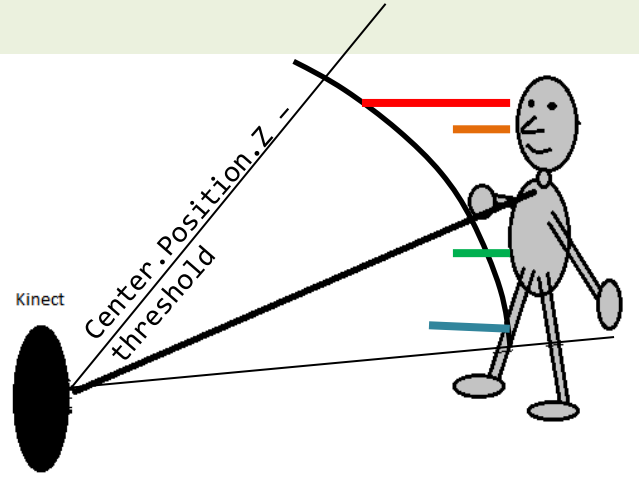
- While standing straight, record the Z and Y measures for Head and some lower Joint in the middle, for example CenterHip
- Determine linear function Z = f(Y) = a*Y + b corresponding the straight line going through the two points
  - For a and b, solve the pair of equations

$$\begin{cases} Z_{Head} = a \times Y_{Head} + b \\ Z_{Hip} = a \times Y_{Hip} + b \end{cases} \Leftrightarrow \begin{cases} a = \dfrac{Z_{Head} - Z_{Hip}}{Y_{Head} - Y_{Hip}} \\ b = Z_{Hip} - a \times Y_{Hip} \end{cases}$$

- Apply the function value in Joint Y – position as the body reference point in Z dimension

Antti Salopuro
Lahti University of Applied Sciences, Finland

Calibration example



Sensor positioned low at about 2m distance from body

Hd_Z = Head Z - position = 1.90
Hd_Y = Head Y - position = 0.45
Hp_Z = Hip Z - position = 1.69
Hp_Z = Hip Y - position = -0.14

$$\begin{cases} 1.9 = a \times 0.45 + b \\ 1.69 = a \times (-0.14) + b \end{cases} \leftrightarrow \begin{cases} b = 1.9 - 0.45a \\ 1.69 = -0.14a + 1.9 - 0.45a \end{cases} \leftrightarrow \begin{cases} b = 1.9 - 0.45a \\ 0.59a = 1.9 - 1.69 = 0.21 \end{cases}$$

$$\leftrightarrow \begin{cases} b = 1.9 - 0.45a = 1.9 - 0.45 \times 0.36 = 1.74 \\ a = \dfrac{0.21}{0.59} = 0.36 \end{cases} \Rightarrow$$

$$f(Y) = 0.36Y + 1.74$$

Antti Salopuro
Lahti University of Applied Sciences, Finland

Application of calibrated Z – dimensional reference point

- As function Z = f(Y) has been defined, it can be used instead of static body reference point

```
//function getZ returns the calibrated Z body reference value corresponding to hand joint height
bool mouseOn = (RightHand.Position.Z < (getZ(RightHand.Position.Y) - mouseActiveThreshold));
```