Application of Microsoft Kinect in controlling computer Keyboard and Mouse input

Antti Salopuro
Lahti University of Applied Sciences, Finland

Sources

- http://msdn.microsoft.com/en-us/library/hh855347.aspx
- http://www.microsoft.com/en-us/kinectforwindows/develop/resources.aspx
- http://www.renauddumont.be/post/2012/06/01/Kinect-SDK-15-Face-tracking-in-Csharp
- http://msdn.microsoft.com/en-us/library/jj130970.aspx (Face tracking)
- Webb, J & Ashley, J., Beginning Kinect Programming with the Microsoft Kinect SDK, Apress, 2012
- Miles, R, Learn the Kinect API, Microsoft, 2012
- http://channel9.msdn.com/coding4fun/projects/Coding4Fun-Kinect-Toolkit
- http://inputsimulator.codeplex.com/
- https://github.com/jera/lazyconsumer/blob/master/NativeMethods.cs
- http://kinectmerrychristmas.codeplex.com/ (sample project of AR)
- http://www.codeplex.com/site/search?query=Kinect&ac=4 (All Kinect sample projects at codeplex)

Kinect Samples

Contents

1. Kinect development system setup
2. General application development with Kinect
   - Augmented reality
3. Controlling the mouse and keyboard with Kinect joint positions
4. Face tracking
5. Speech recognition

PART I

KINECT DEVELOPMENT SYSTEM SETUP – TOOLS REQUIRED

Kinect setup

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

Software libraries required

- MS Kinect for Windows runtime & SDK v1.7 & Developer toolkit (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)
- Handy library if working with avatars etc. game features: XNA 4.0 (http://www.microsoft.com/download/en/details.aspx?id=23714)

System and application setup

- Install
  - MS Kinect for Windows SDK 1.7
- Create a new project in Visual Studio 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)

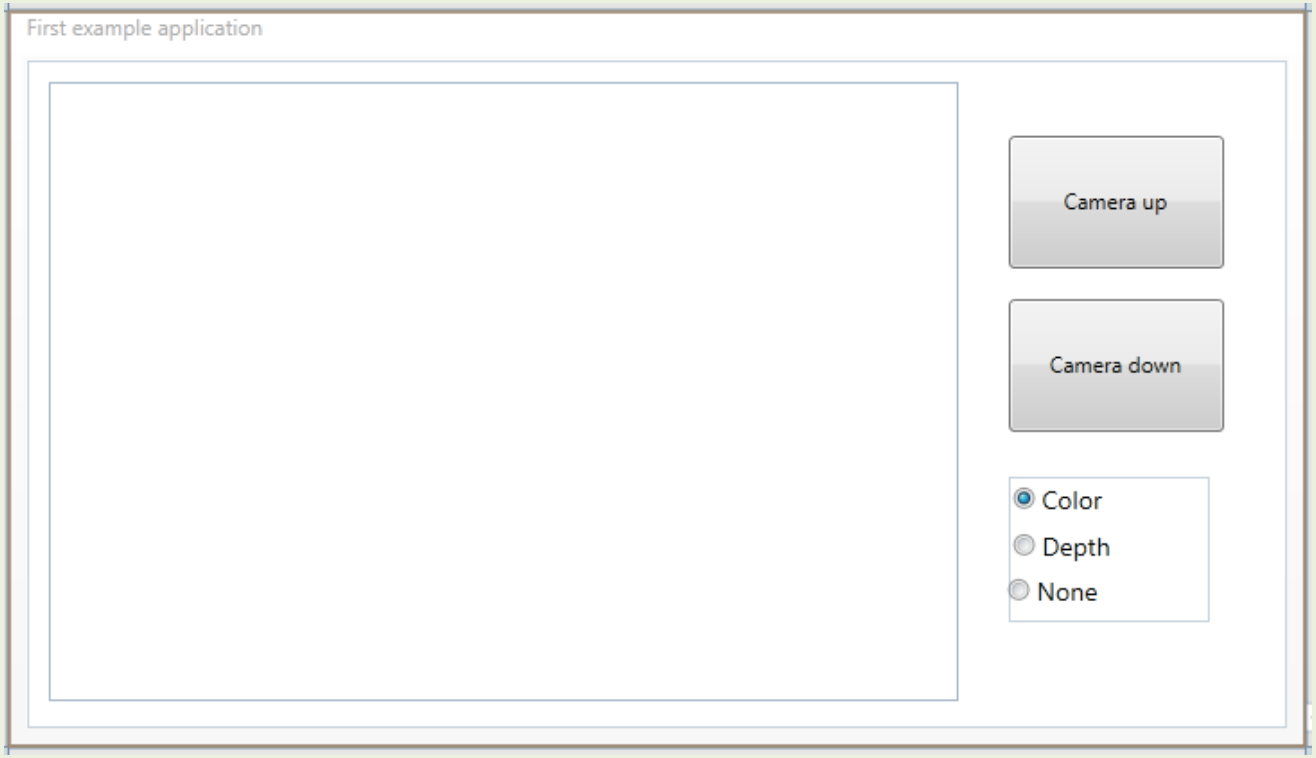
PART II

WPF APPLICATION DEVELOPMENT WITH KINECT SDK GESTURE CONTROL

Sample application

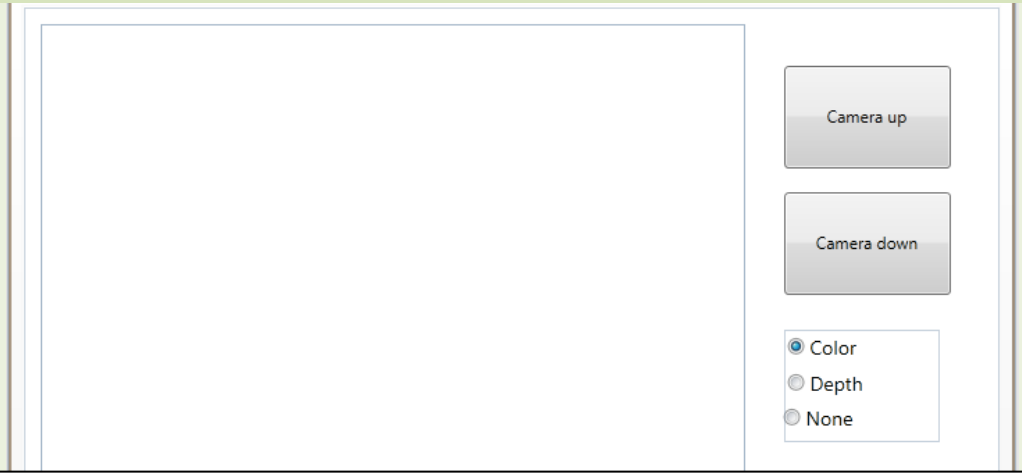
- Controls for adjusting the camera elevation angle
- Image control for showing the color or depth image
- Canvas control for showing two ellipses connected to hand positions
- Radiobuttons to select what is shown
  - Color, depth, only ellipses

First sample application UI



First sample application

Copy - paste the XAML to create the UI




```
 Title="MainWindow" Height="845" Width="1432" Loaded="Window_Loaded">
<Grid>
    <Image Height="773" HorizontalAlignment="Left" Margin="12,12,0,0" Name="image" Stretch="Fill" VerticalAlignment="Top" Width="1058" />
    <Button Content="Camera up" Height="113" HorizontalAlignment="Left" Margin="1186,39,0,0" Name="CamUpButton" VerticalAlignment="Top" Width="174"
      Click="CamUpButton_Click" />
    <Button Content="Camera down" Height="124" HorizontalAlignment="Left" Margin="1186,169,0,0" Name="CamDownButton" VerticalAlignment="Top" Width="174"
      Click="CamDownButton_Click" />
    <StackPanel Height="132" HorizontalAlignment="Left" Margin="1186,389,0,0" Name="stackPanel1" VerticalAlignment="Top" Width="163">
        <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="773" HorizontalAlignment="Left" Margin="12,12,0,0" Name="canvas" VerticalAlignment="Top" Width="1058">
    </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;
        private CoordinateMapper myMapper;

        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

..and coordinate mapper, which is required when converting data between color and depth images

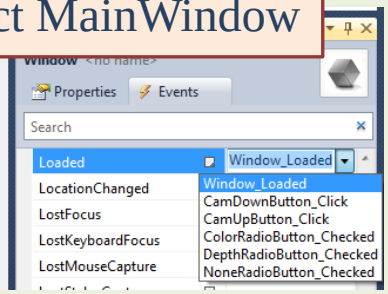
Sensor initialisation tasks

- Get access to the sensor
  - `camDevice = KinectSensor.KinectSensors[0];`
- Create global object for mapping the coordinates
  - `myMapper = new CoordinateMapper(camDevice);`
- Start sensor 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];
        myMapper = new CoordinateMapper(camDevice);
        camDevice.Start();
    }
    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()
    {
            Correction = 0.5f,
            JitterRadius = 0.05f,
            MaxDeviationRadius = 0.05f,
            Prediction = 0.5f,
            Smoothing = 0.5f
    });

    camDevice.AllFramesReady += camera_AllFramesReady;
```

Hook this method to event Loaded of object MainWindow



Sensor initialization code

```
private void Window_Loaded(object sender, RoutedEventArgs e)
{
    try
    {
        camDevice = KinectSensor.KinectSensors[0];
        myMapper = new CoordinateMapper(camDevice);
        camDevice.Start();
    }
    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()
    {
            Correction = 0.5f,
            JitterRadius = 0.05f,
            MaxDeviationRadius = 0.05f,
            Prediction = 0.5f,
            Smoothing = 0.5f
    });

    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, we implement this later

Sensor parameters specific to Kinect for Windows sensor

- The XBOX 360 game version of Kinect sensors is applicable at distances beyond 1.5m and assumes the player is in standing position
- With Kinect for Windows sensors the sensor can be set on a near mode allowing distances as low as 30 cm being measured
- Moreover the sensor can be set to near mode where lower joints are being neglected and detection of skeleton easier happens on seated person

```
camDevice.SkeletonStream.EnableTrackingInNearRange = true;
camDevice.SkeletonStream.TrackingMode = SkeletonTrackingMode.Seated;
```

With XBOX sensor version setting these parameters will end in exception

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 = 50;
   rightEllipse.Width = 50;
   rightEllipse.Fill = Brushes.Aqua;

   leftEllipse = new Ellipse();
   canvas.Children.Add(leftEllipse);
   leftEllipse.Height = 50;
   leftEllipse.Width = 50;
   leftEllipse.Fill = Brushes.PaleVioletRed;
}
```

Canvas is a control on WPF UI where it is possible to draw

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 sample application we will only implement event handler for this event

`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)
{
    ColorImageFrame colorImageFrame = null;
    DepthImageFrame depthImageFrame = null;
    SkeletonFrame skeletonFrame = null;

    try
    {
        colorImageFrame = e.OpenColorImageFrame();
        depthImageFrame = e.OpenDepthImageFrame();
        skeletonFrame = e.OpenSkeletonFrame();

        if (DepthRadioButton.IsChecked.Value)
            image.Source = depthImageFrame.ToBitmapSource();
        else
            if (ColorRadioButton.IsChecked.Value)
                image.Source = colorImageFrame.ToBitmapSource();
            else
                image.Source = null;
                  //Continue
```

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 = 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
```

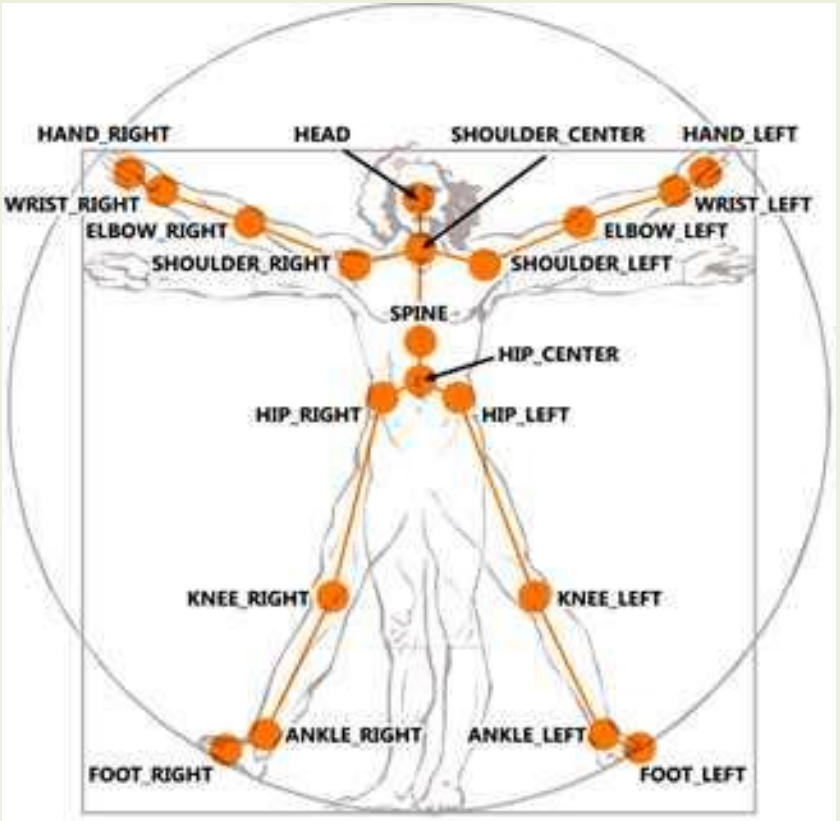
Get access to the skeleton frame

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)

But what is the skeleton and what are the joints?

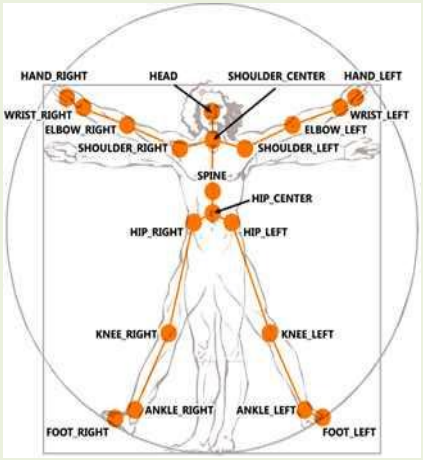


Skeleton array

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

Done already on slide Global references

```
// 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

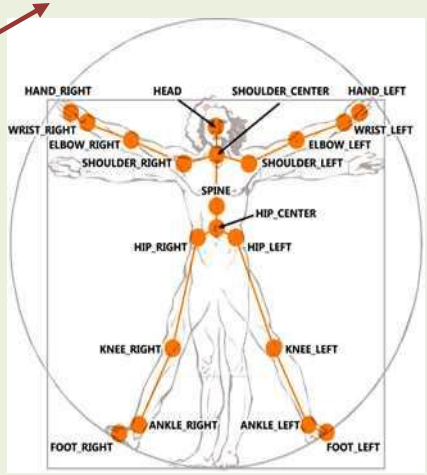
```
// get current skeleton data to the array after
skeletonFrame.CopySkeletonDataTo(this.allSkeletons);
```

Done on step 2

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];
```

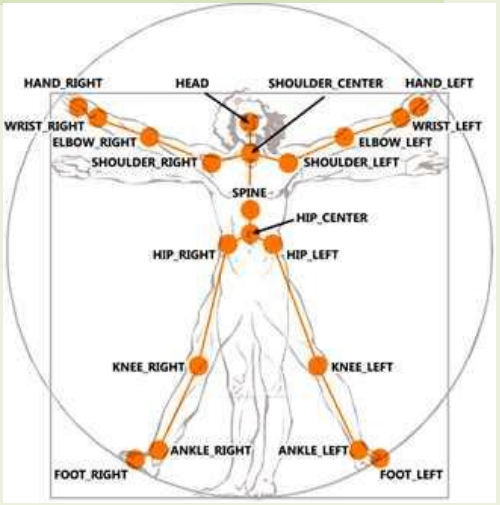


… or by taking the first one found being tracked

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

Skeleton class properties

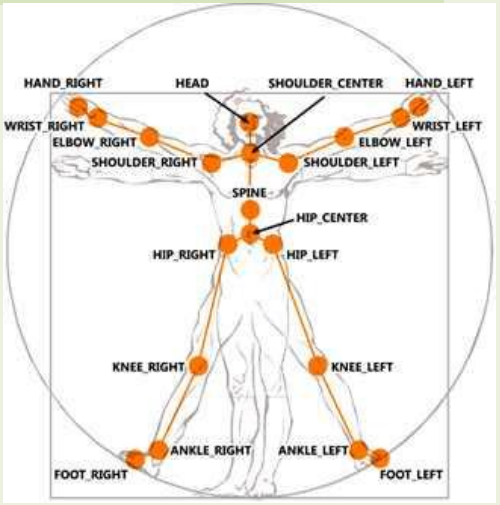
- `ClippedEdges`
  - which parts of the skeleton are clipped by the edges of the frame
- `Position`
  - skeleton position in X-Y-Z space
- `TrackingId`
- `TrackingState`
  - `NotTracked`, `PositionOnly`, `Tracked`
- `Joints`
  - Collection of 20 different body points



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

Joint class properties

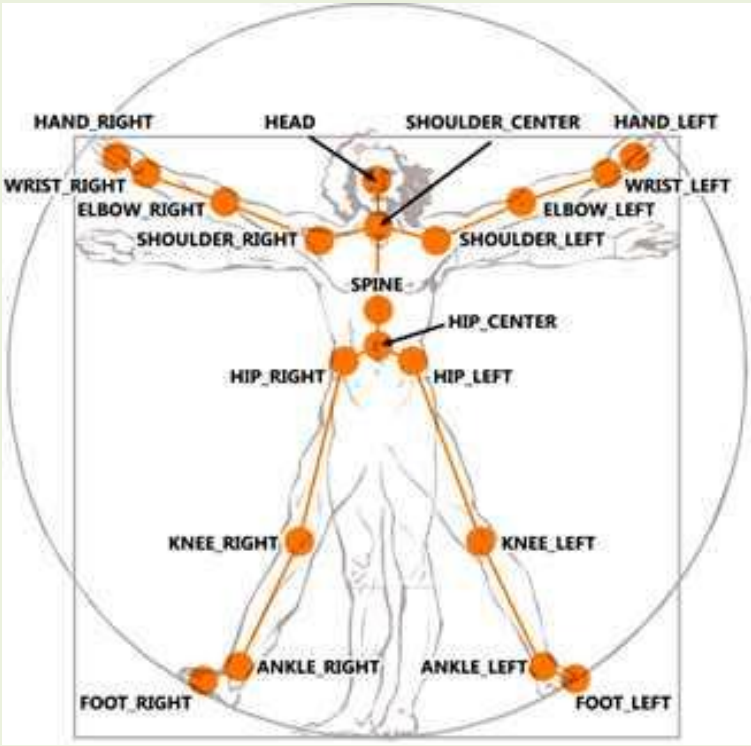
- `JointType`
  - Enum type for indexing the specific joints in the array of joints
- `Position`
  - skeleton position in X-Y-Z space
- `TrackingState`
  - `NotTracked`, `Inferred`, `Tracked`



`Inferred` joint is not currently seen by the sensor

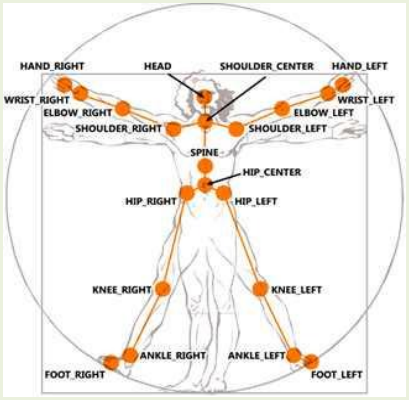
JointType enumeration

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.

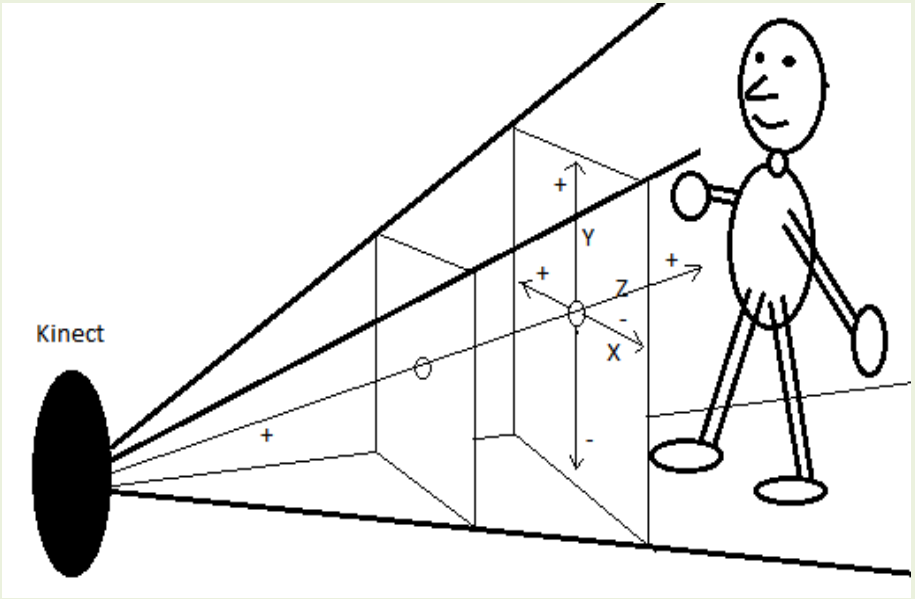
Y = Vertical position measured as the distance, in meters from the Kinect along the Y Axis.

Z = Distance from Kinect measured in meters

Joint position coordinate system

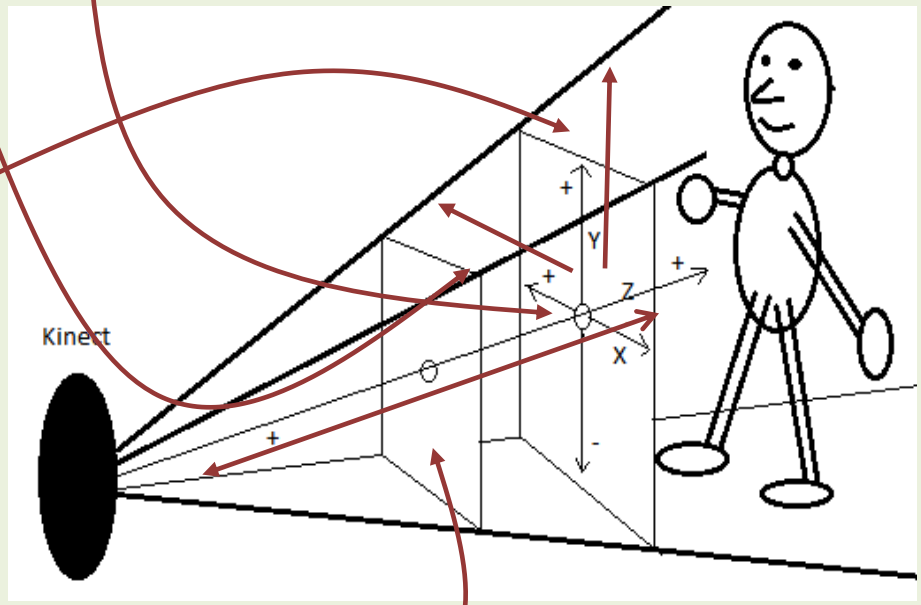
- The origo of the X-Y coordinate system is roughly at the center point of the taken image
- From the camera point of view X coordinate value increases to the left (from human point of view to the right) vertical centerline being the zero axis.

- 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)

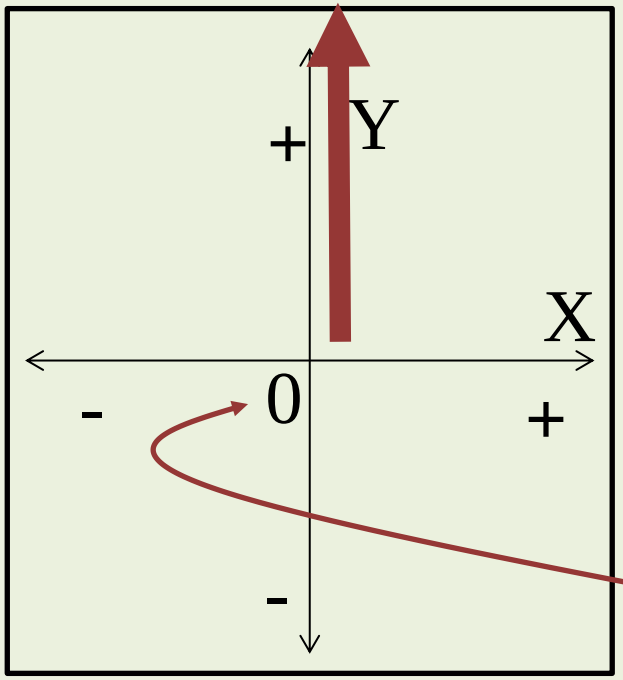
```
//Continue only if both hands are being tracked
if (RightHand.TrackingState == JointTrackingState.Tracked &&
    LeftHand.TrackingState == JointTrackingState.Tracked)
{

    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, ColorImageFo                                                    as);
    }
```

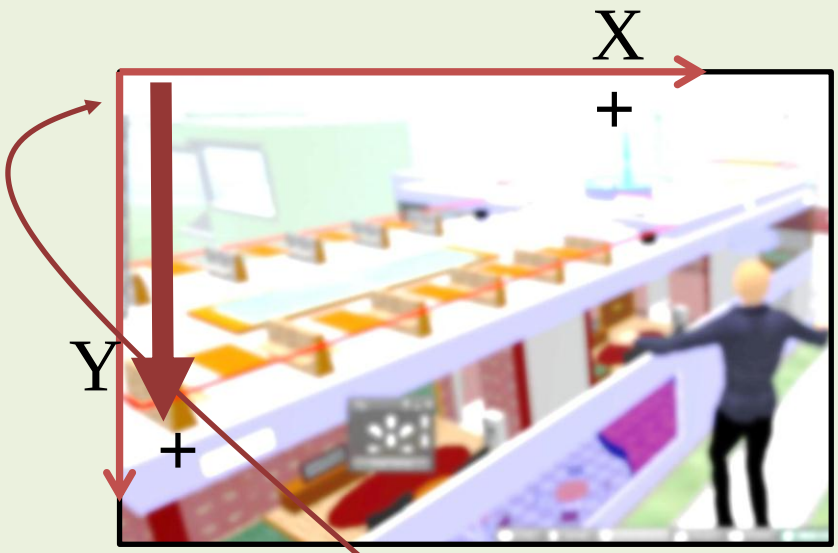
On the very first frame initialise the two ellipses

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

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

```
MapSkeletonPointToColorPoint(SkeletonPoint, ColorImageFormat)
MapSkeletonPointToDepthPoint(SkeletonPoint, DepthImageFormat)
```

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

```
MapDepthPointToSkeletonPoint
MapDepthPointToColorImagePoint
```

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

COLOR IMAGE

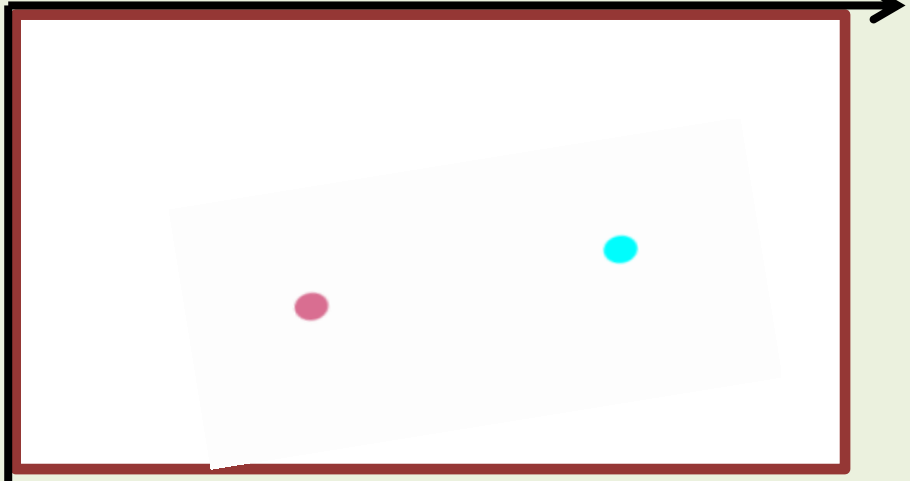
640

480

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

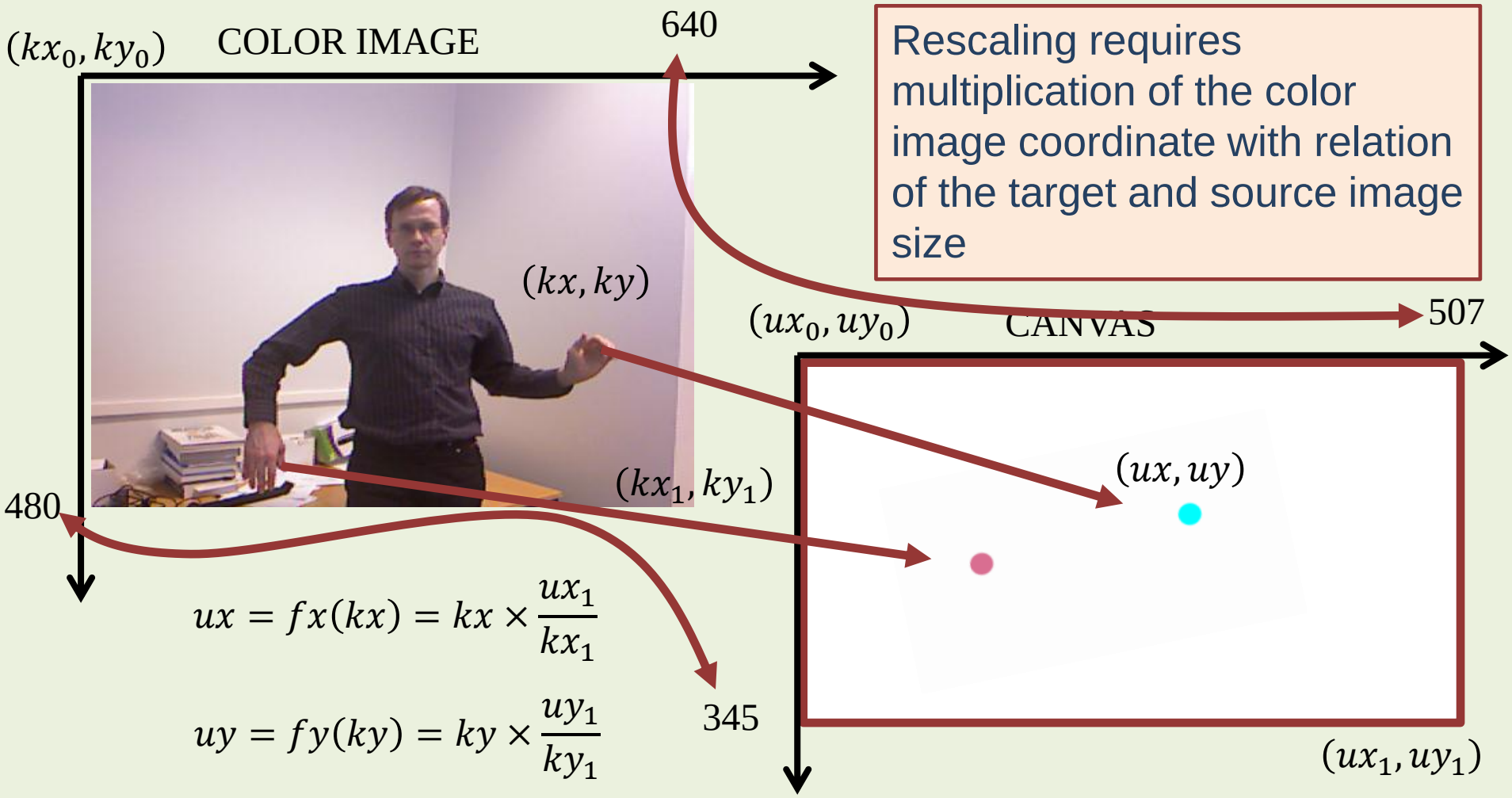
CANVAS

507



345

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

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

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

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

```
plotOnColorImage(RightHand, //joint
         ColorImageFormat.RgbResolution640x480Fps30, //resolution of the image
         rightEllipse, //UI element
         canvas); //target UI canvas
```

This call was added on previous slide

```
//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 = myMapper.MapSkeletonPointToColorPoint(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);
}
```

This implementation plots on color image, make similar method for depth image by changing the word Color to word Depth

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 = myMapper.MapSkeletonPointToColorPoint(myJoint.Position, myFormat);
```

First, map the skeleton point on color image

```
    //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);
}
```

Second, rescale the color image point to the canvas point

Augmented reality AR

- This concept means combining additional images, videos or animations on video image
- With WPF –UI technology this can be implemented sufficiently easily
- WPF UI can be constructed from several layers of images that are loaded on each other
- The layout order of the images can be adjusted and also the opasity property of objects allows objects being seen through

Adding static image on video image

- So far we draw two ellipses on the video frame
- In addition to adding shapes, it is possible to insert images (jpg, gif, png, …) on the UI
- To do this, a control of type `Image` need to be added on the `Canvas` – object
- The content of the `Image` object can then be read from an image file, location and opacity (and other parameters) can be set either in XAML code/Properties view or programmatically

Process

1. Create/find the image file
2. Add the image in the resources of the current Visual Studio WPF project
   - Add existing item→browse for the image file
3. Set the property `Source` of the `Image` object pointing to the wished image file
4. Since `Image` object is also an object of type `UIElement` you can pass it as well to method `plotOnColorImage`

Instead of static image, the `Source` contents can also be a video file or stream (like the image stream of Kinect sensor in our sample project)

Disposing the image frames

```
private void camera_AllFramesReady(object source, AllFramesReadyEventArgs e)
{
    ColorImageFrame colorImageFrame = null;
    DepthImageFrame depthImageFrame = null;
    SkeletonFrame skeletonFrame = null;
    try
    {
        colorImageFrame = e.OpenColorImageFrame();
         //and everything else here ...

    }
    finally
    {
        if (colorImageFrame != null)
            colorImageFrame.Dispose();

        if (depthImageFrame != null)
            depthImageFrame.Dispose();

        if (skeletonFrame != null)
            skeletonFrame.Dispose();
    }
}
```

This step is important to clean up and free resources, makes code running more smoothly

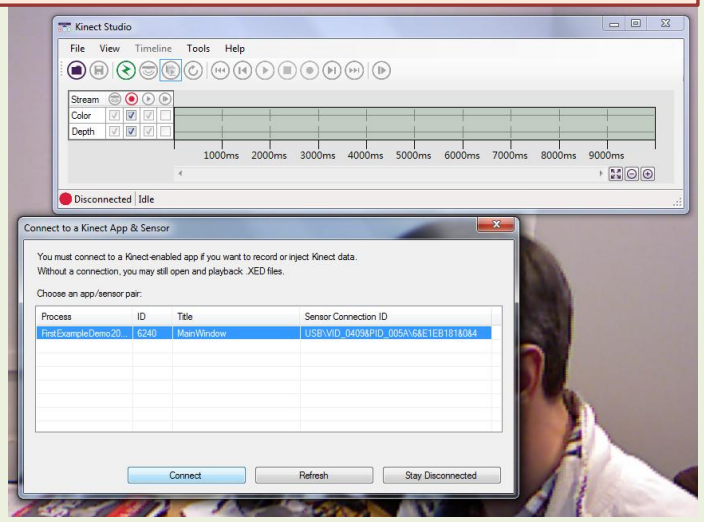
Let us try it!

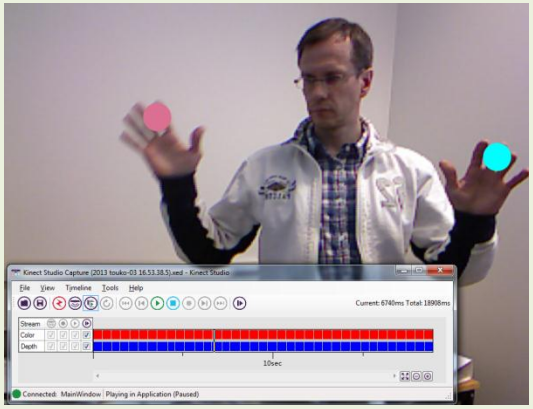
- Build and Debug

Recording data

- With Kinect Studio it is possible record and play Kinect sensor data streams

Launch Kinect Studio and connect an active Kinect software to it





After recording it is possible to play back and run Kinect application without sensor

Render the skeleton

- If we make pairs of the joints that are connected and draw a line between each pair we can reconstruct a graphical interpretation of the skeleton
- Sample code on following two slides implements this with two methods: `addLine` and `drawSkeleton`

addLine

```
private void addLine(Joint j1, Joint j2, Color color, KinectSensor sensor ,Canvas canvas)
{
    Line boneLine = new Line();
    boneLine.Stroke = new SolidColorBrush(color);
    boneLine.StrokeThickness = 5;
    ColorImagePoint j1p = sensor.CoordinateMapper.
        MapSkeletonPointToColorPoint(j1.Position, ColorImageFormat.RgbResolution640x480Fps30);
    //Rescale points to canvas size
    boneLine.X1 = (double)j1p.X/ camDevice.ColorStream.FrameWidth * canvas.Width;
    boneLine.Y1 = (double)j1p.Y / camDevice.ColorStream.FrameHeight * canvas.Height;
    ColorImagePoint j2p = sensor.CoordinateMapper.
        MapSkeletonPointToColorPoint(j2.Position, ColorImageFormat.RgbResolution640x480Fps30);
    boneLine.X2 = (double)j2p.X/ camDevice.ColorStream.FrameWidth * canvas.Width;
    boneLine.Y2 = (double)j2p.Y / camDevice.ColorStream.FrameHeight * canvas.Height;
    canvas.Children.Add(boneLine);
}
```

drawSkeleton

```
private void drawSkeleton(Skeleton skeleton, Canvas target, Color color)
{
    //Spine
    addLine(skeleton.Joints[JointType.Head],skeleton.Joints[JointType.ShoulderCenter],color,camDevice, target);
    addLine(skeleton.Joints[JointType.ShoulderCenter], skeleton.Joints[JointType.Spine],
                                                                    color, camDevice, target);

    //Left leg
    addLine(skeleton.Joints[JointType.Spine], skeleton.Joints[JointType.HipCenter], color, camDevice, target);
    addLine(skeleton.Joints[JointType.HipCenter], skeleton.Joints[JointType.HipLeft], color, camDevice, target);
    addLine(skeleton.Joints[JointType.HipLeft], skeleton.Joints[JointType.KneeLeft], color, camDevice, target);
    addLine(skeleton.Joints[JointType.KneeLeft], skeleton.Joints[JointType.AnkleLeft], color, camDevice, target);
    addLine(skeleton.Joints[JointType.AnkleLeft], skeleton.Joints[JointType.FootLeft], color, camDevice, target);

    //Same with right leg . . .

    //Left arm
    addLine(skeleton.Joints[JointType.ShoulderCenter], skeleton.Joints[JointType.ShoulderLeft],
                                                                    color, camDevice, target);
    addLine(skeleton.Joints[JointType.ShoulderLeft], skeleton.Joints[JointType.ElbowLeft],
                                                                    color, camDevice, target);
    addLine(skeleton.Joints[JointType.ElbowLeft], skeleton.Joints[JointType.WristLeft], color, camDevice, target);
    addLine(skeleton.Joints[JointType.WristLeft], skeleton.Joints[JointType.HandLeft], color, camDevice, target);

    //Same with right arm . . .
}
```

```
//Sample call from client app
drawSkeleton(sd,canvas,Colors.Green);
```

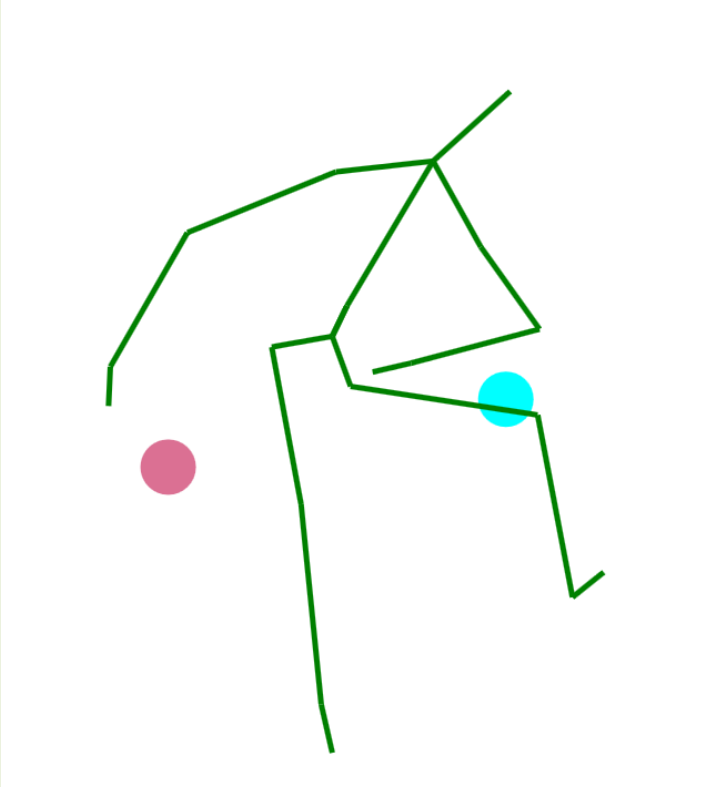
Cleaning previous skeletons

- If we use the previous sample, each frame will cause a completely new skeleton being drawn
- The old ones are however not taken away unless exlicitly commanded→ canvas is soon painted with skeleton color
- Solution 1: clean up the canvas by command

```
canvas.Children.Clear();
```

- Solution 2: create skeleton lines on global level and update their location

Result



PART III

CONTROLLING THE MOUSE AND KEYBOARD WITH KINECT JOINT POSITIONS

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

Tools required

- Kinect SDK, 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 display pixel

  `(x,y) = (245,334)`

  with no mouse button pressed is implemented as:

```
NativeMethods.SendMouseInput(245, //x coordinate
        334, //y coordinate
        (int)SystemParameters.PrimaryScreenWidth, // display width
        (int)SystemParameters.PrimaryScreenHeight, //display height
        false); //no mouse button pressed
```

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 methods 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);
```

StringBuilder class requires namespace System.Text

For getting access to foreground window properties

For changing the foreground window

Select foreground application window, example for Notepad

```
//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

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!!

Joint position scaling to screen for mouse control

- 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

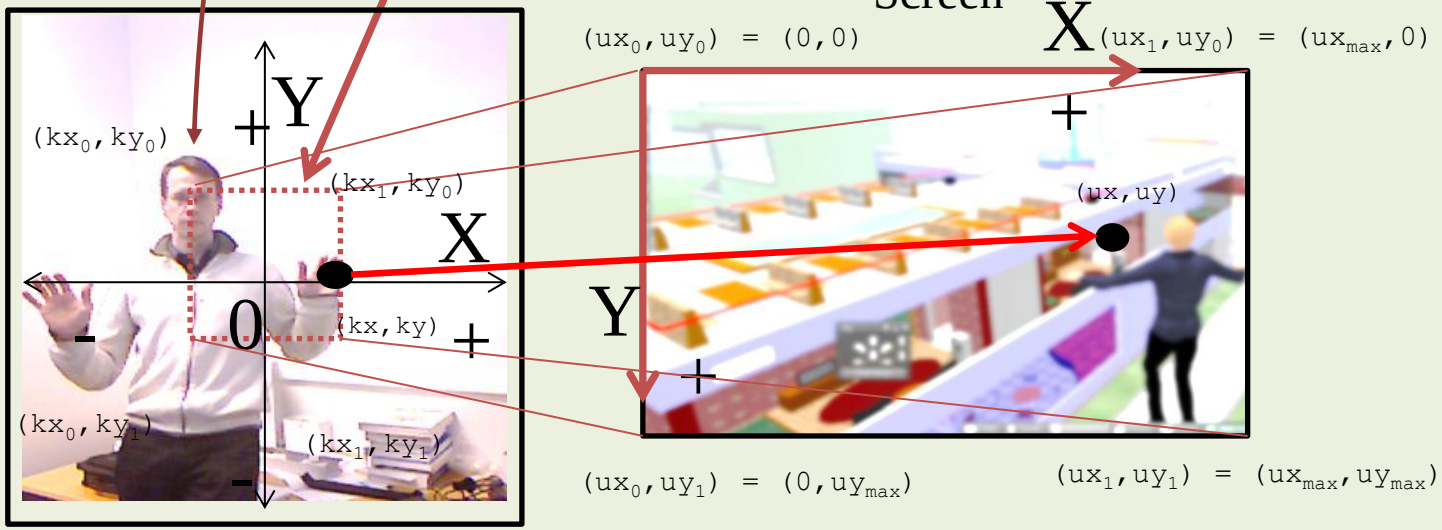
ScaleTo: Scaling factors

```
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

Left hand 30 cm above X – axis zero equals mouse click, right hand gives coordinates for mouse cursor

```
private void mouse(Joint LeftHand, Joint RightHand)
{
    bool mClick = (LeftHand.Position.Y > 0.30);

    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);
}
```

Calling mouse

With implementation on the previous page the mouse can be operated, for example with each frame as

```
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)
    {
        mouse(LeftHand, RightHand);
    }
```

PART III/b

CONTROLLING THE MOUSE AND KEYBOARD WITH KINECT JOINT POSITIONS: GESTURES

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

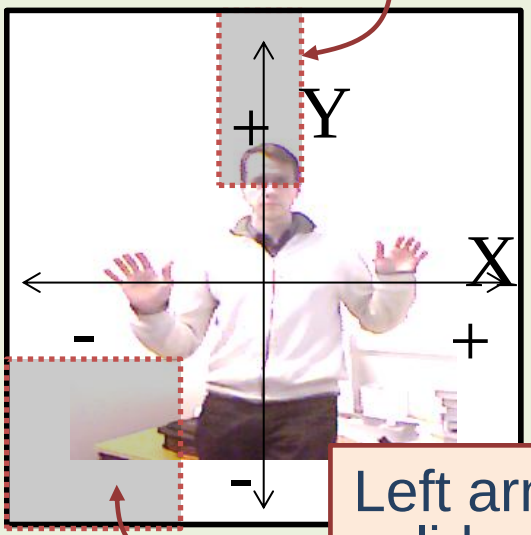
Both hands may have 8 different positions (not all applicable)

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

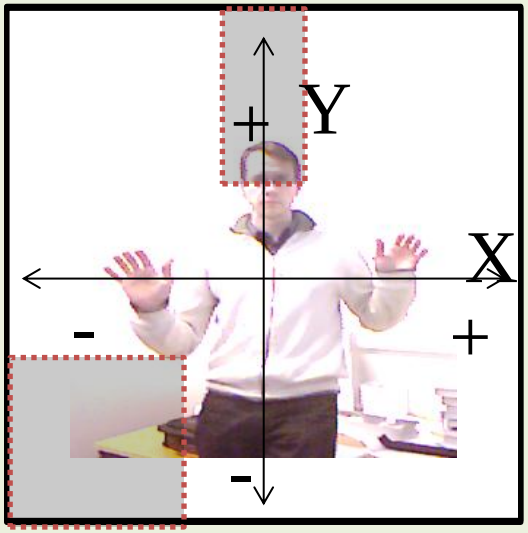


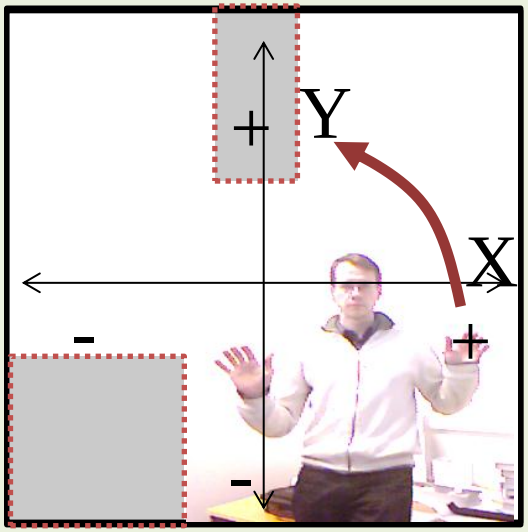
Left arm on low left valid range

```
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));
```

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



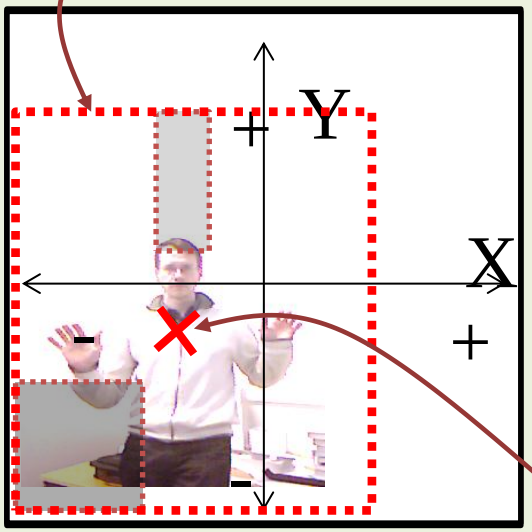


Approach 2: relative position

- 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 at the center of the body

Part IV

ERGONOMY

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
- Especially in 3D - games, it may be 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

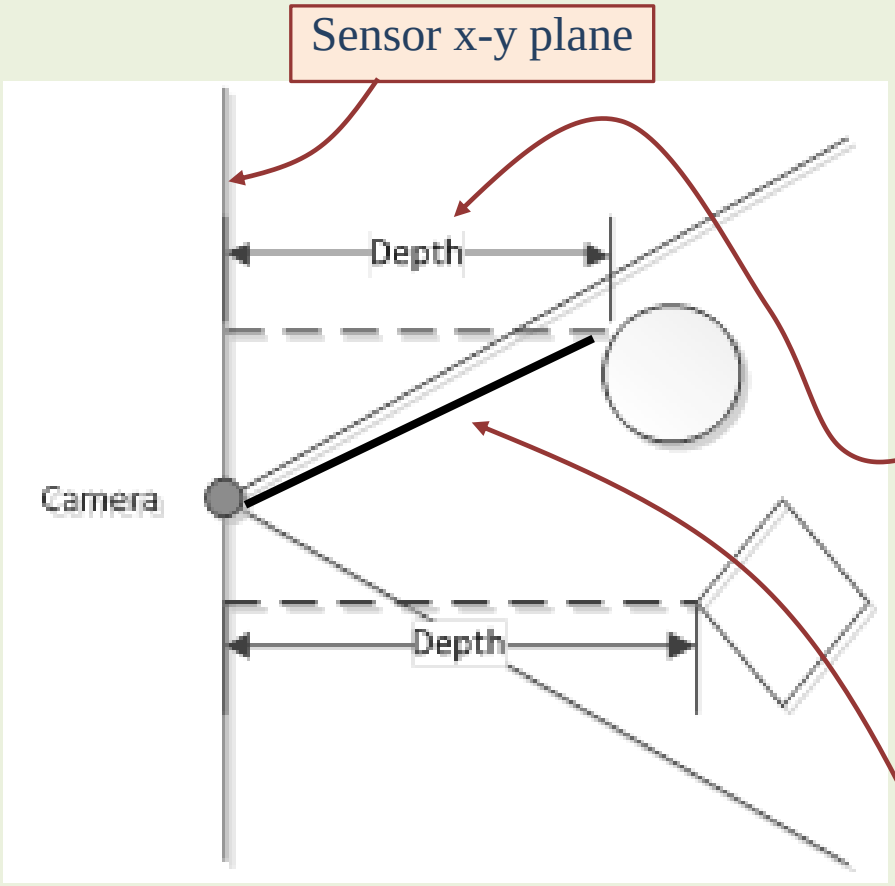
But we needed the second hand for other purposes!!!

Z - dimension

- Z – dimension of the Joint position provides one 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

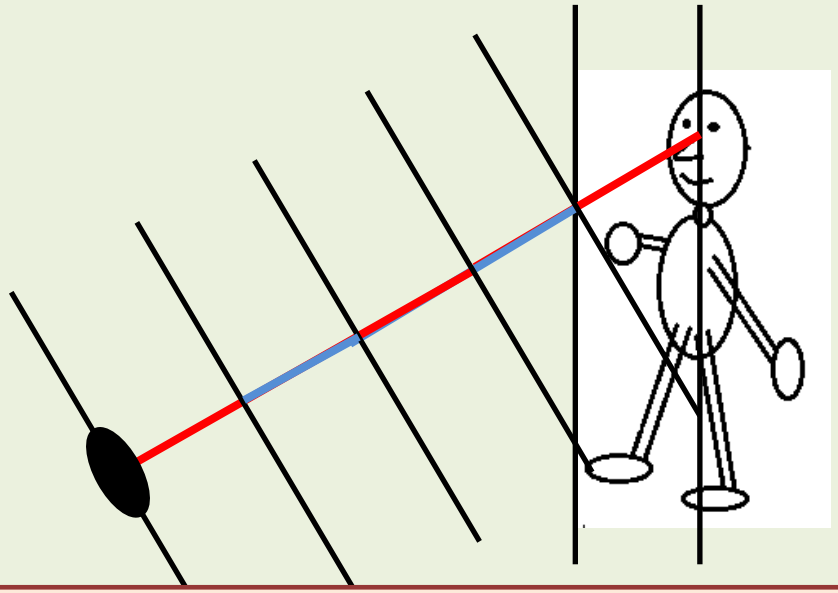
```
float mouseActiveThreshold = 0.3f; //30 cm
Joint Center = sd.Joints[JointType.ShoulderCenter];

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

Z – dimension coordinate system



Z coordinate (depth) value of an object is the distance from the camera sensor **x-y plane** to the object which is not the same as distance from object to sensor

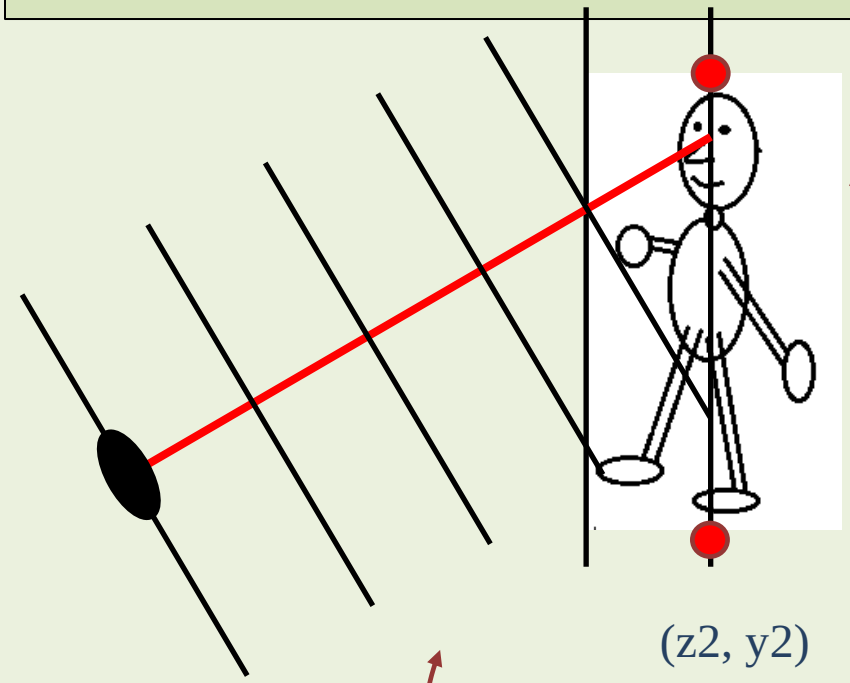
Sensor location effect on Z – dimension



If sensor is positioned low and tilted upwards or positioned high and not tilted, makes some difference in Z – directional position measure while the measure is taken from sensor point of view

Sensor point of view, low position and tilted angle

- Distance between camera and legs is different from distance between camera and head



Body midline is at line

$$y = kz + y_1 - kz_1$$

where $k = \frac{y_2 - y_1}{z_2 - z_1}$

Threshold line is then

$$y = k(z + t) + y_1 - kz_1$$

How big is this effect?

- If, for example, [`hand.Z` < `head.Z` – 20 cm] is required, how far the hand must be pushed to reach this level if sensor is far/close/bottom/up
  - Assume head and hand are both 1.5 m high

Case 1: sensor at head level

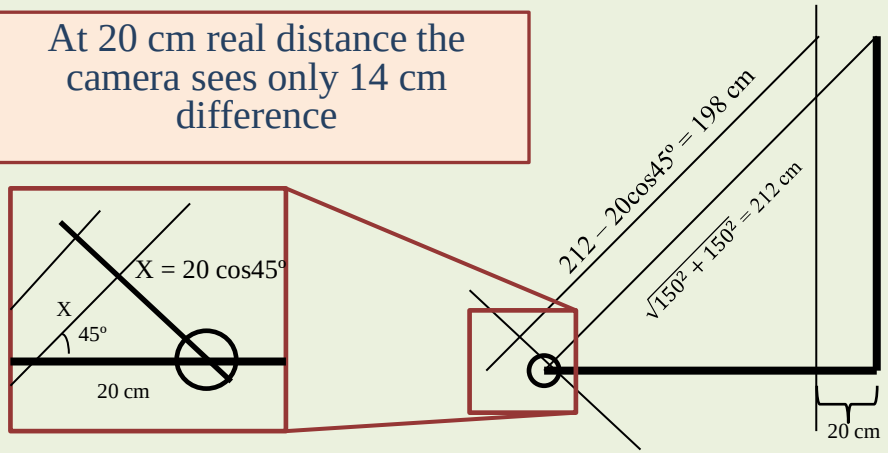
Sensor distance does not have any effect, 20 cm real difference is also seen by the sensor

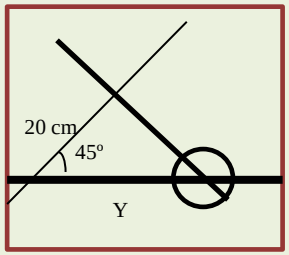
How big is this effect?

- If, for example, [`hand.Z` < `head.Z` – 20 cm] is required, how far the hand must be pushed to reach this level if sensor is far/close/bottom/up
  - Assume head and hand are both 1.5 m high

Case 2: sensor on floor, 1.5 m away from body

At 20 cm real distance the camera sees only 14 cm difference





$$y \cos 45° \, cm = 20cm \leftrightarrow$$
$$y = \frac{20}{\cos 45°} = 28cm$$

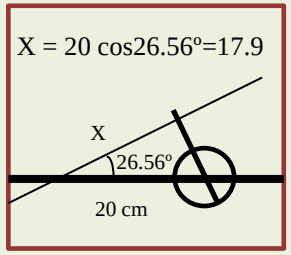
To make the difference up to 20 cm from sensor point of view, the hand must be pushed 28 cm forward

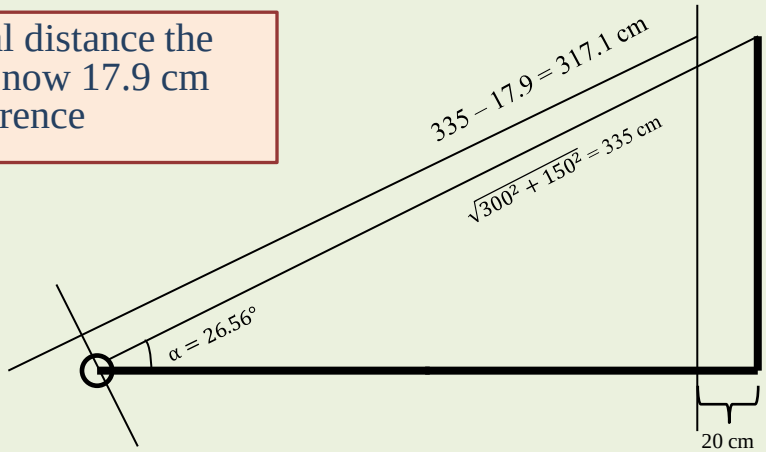
How big is this effect?

- If, for example, [`hand.Z` < `head.Z` – 20 cm] is required, how far the hand must be pushed to reach this level if sensor is far/close/bottom/up
  - Assume head and hand are both 1.5 m high

Case 3: sensor on floor, 3 m away from body

At 20 cm real distance the camera sees now 17.9 cm difference





$$y \cos 26.56° \, cm = 20cm \leftrightarrow$$
$$y = \frac{20}{\cos 26.56°} = 22.4cm$$

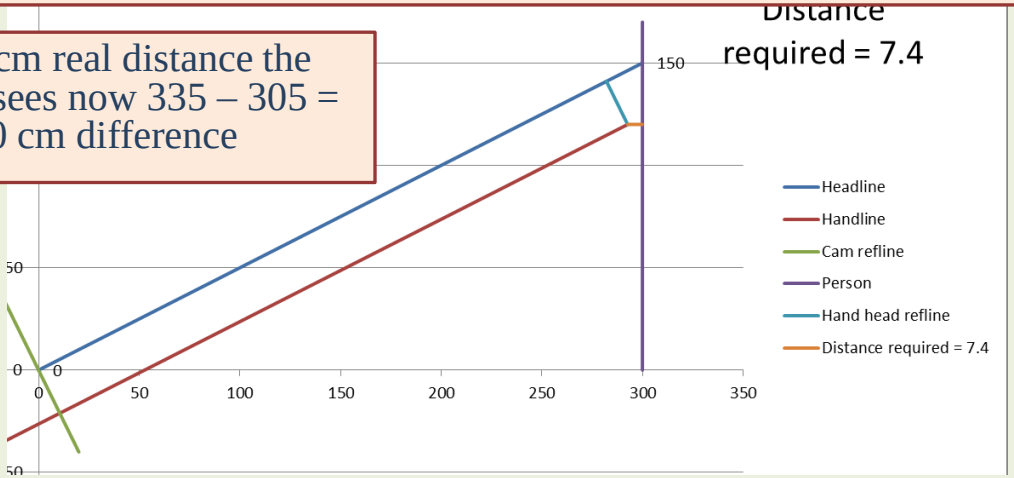
To make the difference up to 20 cm from sensor point of view, the hand must be pushed 22 cm forward

Control and reference joints at different heights?

- If, for example, [`hand.Z` < `head.Z` – 20 cm] is required, how far the hand must be pushed to reach this level if sensor is far/close/bottom/up
  - Assume head 1.5 m high but hand is 30 cm lower

Case 4: sensor 3 m away from body

At 20 cm real distance the camera sees now 335 – 305 = 30 cm difference



To make the difference up to 20 cm from sensor point of view, the hand must now be pushed only 7.4 cm forward !!!

Cancelling the sensor location effect

- No effect if control Joint (`hand`) and reference Joint (`head`) are close to each other and sensor is far or at the same height with them
- If sensor location is not known 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 tilt calibration

- While standing straight, record the Z and Y measures for `Head` (z1, y1) and some lower Joint in the middle, for example `CenterHip` (z2, y2)
- Determine linear function Z = f(Y) corresponding the straight line going through the two points by

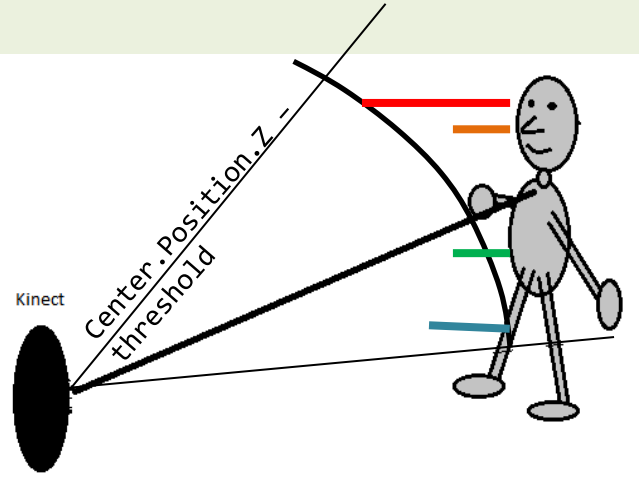
$$z = \frac{y-b}{k}, \text{ where k} = \frac{y_2-y_1}{z_2-z_1} \text{ and b} = y_1 - kz_1$$

- This equation determines the position of the body in z – axis, the equation which describes the location of the threshold line in y axis is now defined as

$$z = \frac{y-b}{k} - t, \text{ where k} = \frac{y_2-y_1}{z_2-z_1}, \text{ b} = y_1 - kz_1$$

and t is the threshold

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

$$z = \frac{y-(y_1-(\frac{y_2-y_1}{z_2-z_1})z_1)}{(\frac{y_2-y_1}{z_2-z_1})} = \frac{y-(0.45-2.269\times1.90)}{2.269} = \frac{y+3.8611}{2.269}$$

Application of calibrated Z – dimensional reference point

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

```
private float calibratedZ(float ref1_y, float ref1_z, float ref2_y, float ref2_z, float control_y)
{
    float k = (ref2_y - ref1_y) / (ref2_z - ref1_z);
    return (control_y - (ref1_y - (k * ref1_z))) / k;
}
```

Application

```
//function getZ returns the calibrated Z body reference value corresponding to hand joint height
float mouseThres = 0.2f;
float calZ = calibratedZ(Head.Position.Y, Head.Position.Z,
                    CenterHip.Position.Y, CenterHip.Position.Z, RightHand.Position.Y);
bool mouseOn = RightHand.Position.Z < calZ - mouseThres;
```

Part IV

FACE TRACKING

Face tracking

- Face tracking is a feature available with the Kinect SDK
- Requires extension available with Kinect SDK `Microsoft.Kinect.Toolkit.FaceTracking`
  - Reference
  - Namespace
  - Dynamic library FaceTrackLib.dll must be manually copied in the bin folder of the project
- Provides 3D (real space) and 2D (image space) coordinates of 87 different poits of human face
- Also provides several predefined indices corresponding facial expressions as *Animation Unit Coefficients*

Global objects

- The FaceTracker class provides the face tracking service
- As well it is useful to store the image data needed by the face tracker to global objects

```
private FaceTracker faceTracker;
byte[] colorPixelData;
short[] depthPixelData;
private Skeleton trackedSkeleton;
```

Object initialization

- Object initialization is best made in window initialization method (`Window_loaded`)
- FaceTracker object construction connects the object to the Kinect sensor object

```
colorPixelData = new byte[camDevice.ColorStream.FramePixelDataLength];
depthPixelData = new short[camDevice.DepthStream.FramePixelDataLength];
faceTracker = new FaceTracker(camDevice);
```

Get access to frame data

- In event handler of AllFramesReady the face tracking data is accessed through color image, depth image and skeleton frame data

```
private void camera_AllFramesReady(object source, AllFramesReadyEventArgs e)
{
    ColorImageFrame colorImageFrame = null;
    DepthImageFrame depthImageFrame = null;
    SkeletonFrame skeletonFrame = null;
    try
    {
        colorImageFrame = e.OpenColorImageFrame();
        depthImageFrame = e.OpenDepthImageFrame();
        skeletonFrame = e.OpenSkeletonFrame();

        //If we lost the facetracker, try recover even if it takes long time
        if (faceTracker == null)
            faceTracker = new FaceTracker(camDevice);
        if (faceTracker == null || colorImageFrame == null ||
            depthImageFrame == null || skeletonFrame == null)
                        return; //Not worth continuing
```

Frame data for face tracking

- Continue by copying the frame image data into pixel data objects
- Make sure there is a skeleton being tracked

```
//Get pixel and skeleton data from frames
colorImageFrame.CopyPixelDataTo(colorPixelData);
depthImageFrame.CopyPixelDataTo(depthPixelData);
skeletonFrame.CopySkeletonDataTo(allSkeletons);

//And make sure we have one skeleton tracked
if (trackedSkeleton == null) //If not from previous frame, try find one
    trackedSkeleton = allSkeletons.FirstOrDefault(s => s.TrackingState == SkeletonTrackingState.Tracked);

if (trackedSkeleton == null)
    return; //Not worth continuing if no skeletons available

//If we get this far, we might as well output the image
image.Source = colorImageFrame.ToBitmapSource();
```

FaceTrackFrame object

- FaceTrackFrame object is constructed from `colorPixelData`, `depthPixelData` and `trackedSkeleton`
- This object provides the face tracking data

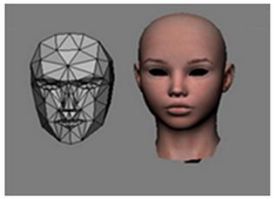
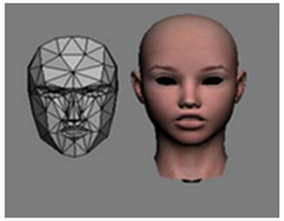
```
//If resources are available, first create the FaceTrackFrame object
//with static method Track of the FaceTracker class
FaceTrackFrame faceFrame = faceTracker.Track(camDevice.ColorStream.Format,
                        colorPixelData, camDevice.DepthStream.Format,
                        depthPixelData, trackedSkeleton);

//And now, if face is succesfully tracked, get the required tracking data...
if (faceFrame.TrackSuccessful)
{
    ...
```

Face tracking data: AU coefficients

- Face tracking data includes 2D, 3D, triangles and indices corresponding to emotional expressions as *Animation unit coefficients*

```
// Gets the AU coeffs, this is some prederfined facial expressions
EnumIndexableCollection<AnimationUnit, float> AUCoeff =
                                    faceFrame.GetAnimationUnitCoefficients();

// Gets the AU coeffs, this is some prederfined facial expressions
EnumIndexableCollection<AnimationUnit, float> AUCoeff =
faceFrame.GetAnimationUnitCoefficients();
//brow lowerer = 0: neutral, +1: fully lowered, -1: raised all the way
float browLowererAU = AUCoeff[AnimationUnit.BrowLower];
//jaw lowerer = 0: closed, +1: fully open, -1: closed (as 0)
float jawLowererAU = AUCoeff[AnimationUnit.JawLower];
```

AU coefficient values

AU Name and Value	Avatar Illustration	AU Value Interpretation
Neutral Face (all AUs 0)		
AU0 – Upper Lip Raiser (In Candid3 this is AU10)		0=neutral, covering teeth 1=showing teeth fully -1=maximal possible pushed down lip
AU1 – Jaw Lowerer (In Candid3 this is AU26/27)		0=closed 1=fully open -1= closed, like 0
AU2 – Lip Stretcher (In Candid3 this is AU20)		0=neutral 1=fully stretched (joker's smile) -0.5=rounded (pout) -1=fully rounded (kissing mouth)
AU3 – Brow Lowerer (In Candid3 this is AU4)		0=neutral -1=raised almost all the way +1=fully lowered (to the limit of the eyes)
AU4 – Lip Corner Depressor (In Candid3 this is AU13/15)		0=neutral -1=very happy smile +1=very sad frown
AU5 – Outer Brow Raiser (In Candid3 this is AU2)		0=neutral -1=fully lowered as a very sad face +1=raised as in an expression of deep surprise

Face tracking data: 2D coordinates

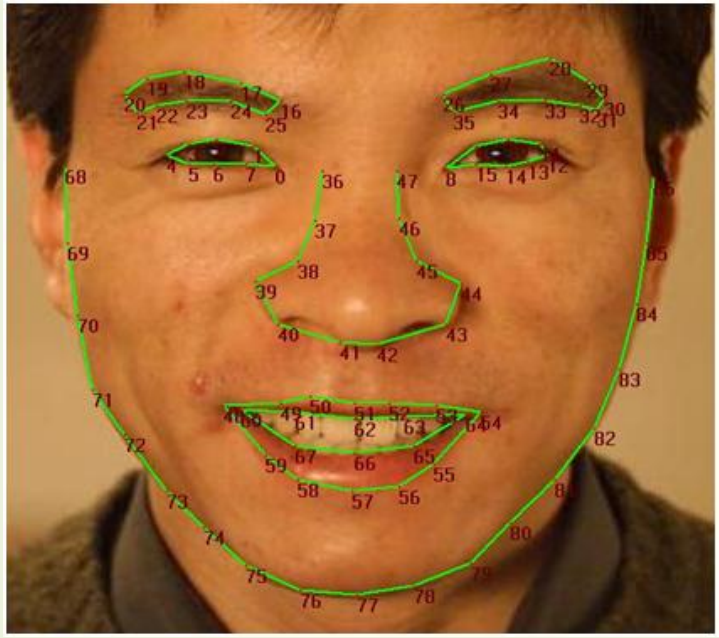
- There exists 100 points of face that can be tracked for location in 2D color image coordinates
- This data is accessed from `FaceTrackFrame` object by method `GetProjected3DShape`

```
EnumIndexableCollection<FeaturePoint, PointF> face2DFrame =
                                    faceFrame.GetProjected3DShape();

//PointF is a struct of FaceTracking Toolkit, has properties X and Y
PointF leftEye = face2DFrame[FeaturePoint.OuterTopLeftPupil];
PointF rightEye = face2DFrame[FeaturePoint.OuterTopRightPupil];
//Do not try this at home!
bool crossEyed = (leftEye.X > rightEye.X);
```

Any of these points can be applied to plot objetcs on image in the same way as with the skeleton joint data or, for example, to operate mouse cursor

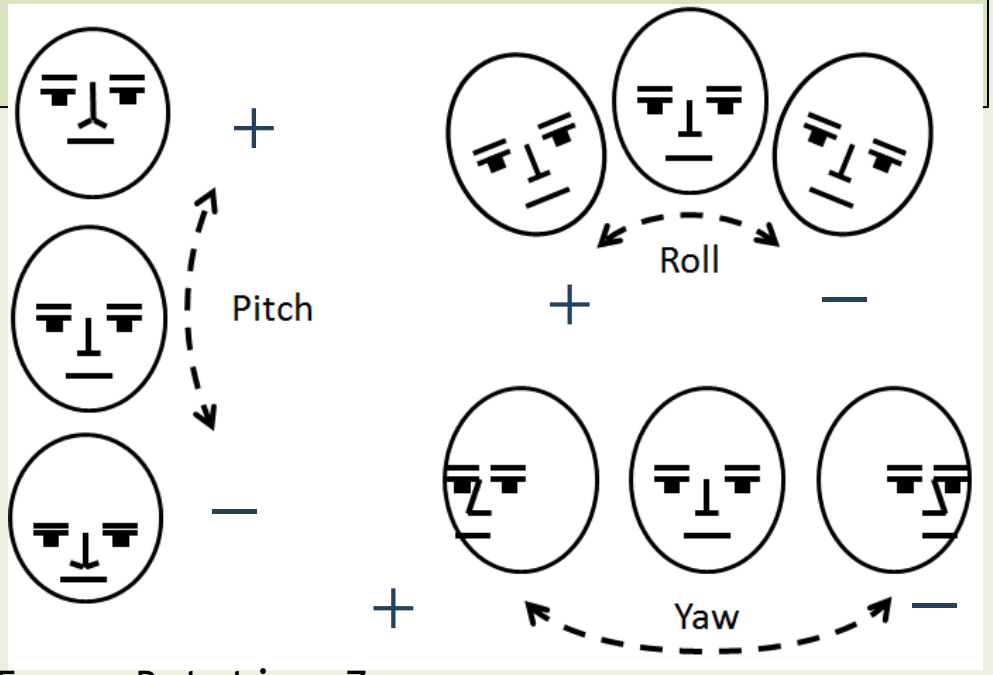
Feature point enumeration

- There exists 120 points of face in collection `Face2DFrame`
- Enum type `FeaturePoint` lists 71 of them



Head Pose

- Head pose can be accessed through the property Rotation of the object FaceTrackFrame




```
txtRoll.Content = "Roll: " + faceFrame.Rotation.Z;
txtPitch.Content = "Pitch: " + faceFrame.Rotation.X;
txtYaw.Content = "Yaw: " + faceFrame.Rotation.Y;
```

Face tracking data: 3D coordinates

- Same 100 points of face that can also be tracked for location in 3D real space coordinates
- This data is accessed from `FaceTrackFrame` object by method `Get3DShape`

```
EnumIndexableCollection<FeaturePoint, Vector3DF> face3DFrame =
                                                    faceFrame.Get3DShape();

//Vector3DF is struct of FaceTracking Toolkit, properties X and Y
Vector3DF leftEye3D = face3DFrame[FeaturePoint.OuterTopLeftPupil];
Vector3DF rightEye3D = face3DFrame[FeaturePoint.OuterTopRightPupil];
//Do not try this at home!
bool crossEyed = (leftEye3D.X > rightEye3D.X);
```

These points can be applied to solve location of face points in 3D space

Part VI

SPEECH RECOGNITION

Required SDK libraries

- Speech Platform SDK installed (v11)
  - Should be installed already with Kinect SDK
- Language package for wished languages
- Reference to Microsoft.Speech
- Namespaces
  - `Microsoft.Speech.AudioFormat`
    - For input format only
  - `Microsoft.Speech.Recognition`
- To produce speech, this SDK is not required, apply standard library `System.Speech` for that

Make computer talk

- This is very easy
  - Reference to System.Speech
- Create SpeechSynthesizer object
- Make it speak

```
SpeechSynthesizer mySynth = new SpeechSynthesizer();
mySynth.Speak("It is so easy to talk like this");
PromptBuilder myPrompt = new PromptBuilder();
myPrompt.AppendText("We can also build");
myPrompt.AppendText("text from pieces to a Prompt object");
mySynth.Speak(myPrompt);
```

Unfortunately the pronunciation is by default in English, but it can be controlled by applying so called set of International Phonetic Alphabet (IPA) and method `AppendTextWithPronunciation`

Speech recognition sample

- In application, for the speech control, the key classes are the `SpeechRecognitionEngine` and `KinectAudioSource`
- This sample applies a helper class `SpeechRecognizer` that you can copy as it is and modify to your purposes
  - Actually this sample is also modified from ShapeGame which comes with the SDK sample projects

SpeechRecognizer helper class

- Creates references to Kinect speech recognition objects of `SpeechRecognitionEngine` and `KinectAudioSource` and takes care of their initialisation and disposing
- Vocabularies implemented as C# collection type of `Dictionary<,>`, where words or phrases are adjoined with activities
- Build of complete grammar of the application from these vocabularies
- Interface `SaidSomething` to event handlers of speech commands and raising of these events when sensible words or sentences were said

Grammar structure

- Grammar is a structured property of the `SpeechRecognitionEngine` (`SRE`) object, once grammar is created, it is uploaded to `SRE` with method `LoadGrammar`
- Grammar object constructs from set of allowed words (`Choises`) and the rules of how to create legal sentences (`GrammarBuilder`) by combination of the words
  - This is something you learn more on the second week of TrabHCI
- Event handler `SpeechRecognized` of the SRE raises event of type SaidSomething with reasonable sentence

Shortcuts to the code files

SpeechRecognizer.cs

MainWindow.xaml.cs

MainWindow.xaml