This document will show to the lecturer the results of the research to handle different face points using a Kinect sensor to detect people's gestures.

Gestures detection using Kinect

Face action units and Kinect

Jaime Jarrín Valencia

Contents

Table of figures

Index of tables

1. What is a Kinect?

Kinect is a line of motion sensing input devices by Microsoft for Xbox 360 and Xbox One video game consoles and Windows PCs. Based around a webcam-style add-on peripheral, it enables users to control and interact with their console/computer without the need for a game controller, through a natural user interface using gestures and spoken commands.

Nowadays, two different Kinect models had been launched into the market the version 1 and version 2, in Table 1 we can see the key differences between them.

Table 1 Kinect versions comparisons

Functions	Kinect 1 (SDK 1.8)	Kinect 2 (SDK 2.0)
Resolution	640x480 30fps 4:3	1920x1080 30fps 16:9
Vision angle	57 degrees horizontal, 43 degrees vertical	70 degrees horizontal, 60 degrees vertical
Minimal distance	1.82 m	1.37 m
Active IR (night vision)	No	Yes
Latency	102 ms	20 ms
Motor Manual adjustment	Yes	No
Simultaneous people detected	4	6
Simultaneous body point	20 / people	25 / people
Finger and wrist detection	No	Yes
Muscle detection	No	Yes
Pulse detection	No	Yes

One of key factors of Kinect 2.0 is its resolution, with a higher resolution the brings the possibility of get better images to process them and acquire features with more “quality” if we compared with the ones acquired with the version 1.

In the Figure 1 the Kinect 1 is shown, by the time it was designed to work exclusively with Xbox 360, however there were a few releases of the SDK to develop applications on windows based systems, right now the latest SDK available is the 1.8

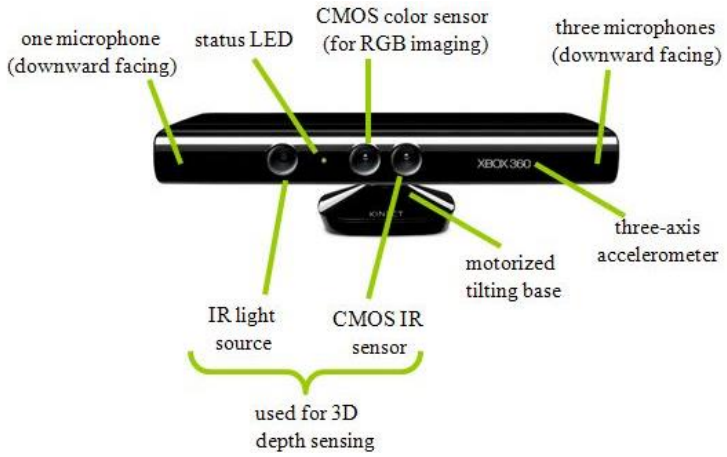

Figure 1 Kinect 1

In the other hand, the newest and more powerful sensor is shown in Figure 2, we can see the different sensors available on this device. With this device the SDK 2.0 is needed to start developing applications.

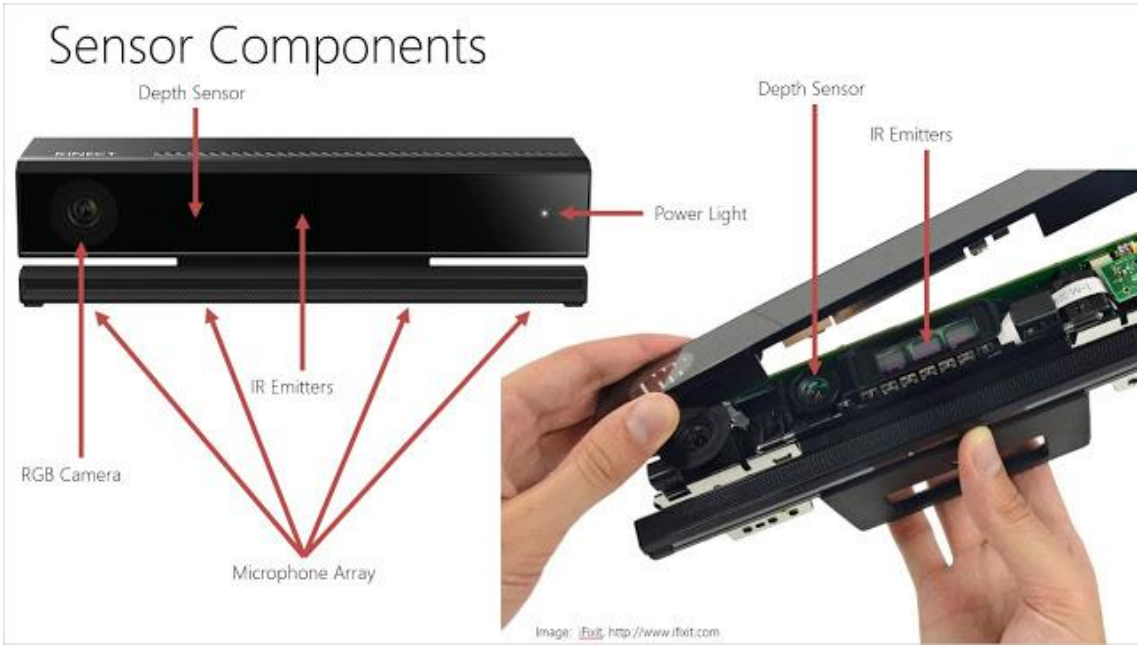

Figure 2 Kinect 2

Sadly, to use this Kinect with a PC it is mandatory to connect it through an adapter which is shown in Figure 3. This device is sold separately and it converts the ports designed to work with the Xbox one into an USB 3.0 port and standard 110/220 V plug.

Figure 3 PC adapter for Kinect 2

2. Research objectives

Once we know the devices which this research is going to work with the research objectives can be established which the following are:

- Study and recognition of facial gestures
- Study of functionalities of Kinect 1 and 2
- Study of development environment of SDK 1.8 and SDK 2.0
- Study and understand previous implementations using SDK 1.8 to recognize facial gestures.

The key objective is to get a facial and gestures recognition using the both versions of Kinect and get some differences between them.

3. Face gestures and Action units

Facial Action Coding System (FACS) is a system to taxonomize human facial movements by their appearance on the face, based on a system originally developed by a Swedish anatomist named Carl-Herman Hjortsjö. Movements of individual facial muscles are encoded by FACS from slight different instant changes in facial appearance. It is a common standard to systematically categorize the physical expression of emotions, and it has proven useful to psychologists and to animators. Typically this coding system associated to muscle's face are called action units (AU) and are "patterns" that will express different emotions of the person of study.

In Figure 4 we can see 90 points defined which represent main muscles of the face and then this points are translated to action units to detect emotions.

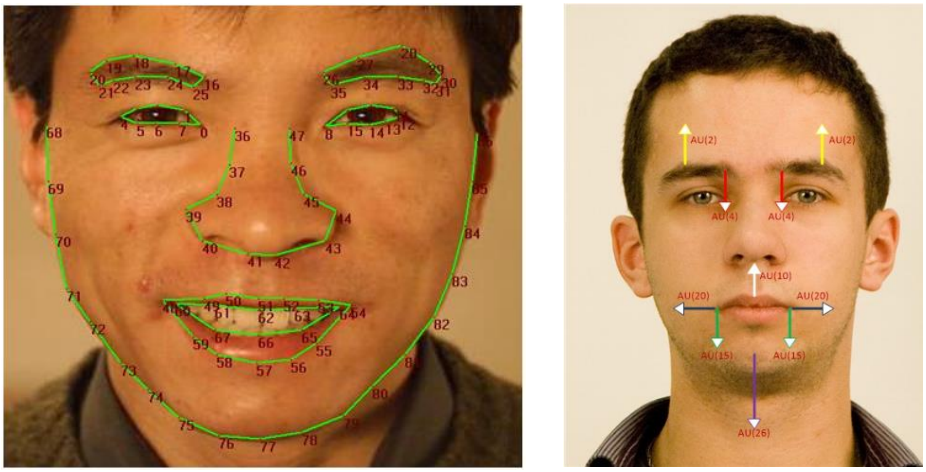

Figure 4 Face points

The mapping of the typical action units for the motions is the Table 2. For a complete mapping of the action units you can refer to [2].

Table 2 Action units and emotions

Emotion	***Action Units***
Happiness	6+12
Sadness	1+4+15
Surprise	1+2+5B+26
Fear	1+2+4+5+7+20+26
Anger	4+5+7+23
Disgust	9+15+16
Contempt	R12A+R14A

To conclude this section, action units are the key feature that any sensor would like to recognize in order to detect gestures and emotions, and this is exactly what Kinect does. However, due some lack of resolution, processing power and constant change of the environment this could be a hard task to fulfill. Next sections will show us the methods used to test the action units with Kinect 1 and 2.

4. Working with Kinect

4.1. Coding Language and environment

As a Microsoft application the code can be developed with C, C++ and C#. In this, C# is used exclusively and Microsoft Visual Studio 2015. If this language is new to you please refer to [2] to get involved into it.

4.2. SDK 1.8

In the research group the previous work made by Estefania Ferreira was used as base to perform this work. What I've done is to complete one library to get the points of the face. To do this the Microsoft.Kinect.Toolkit.FaceTracking (see Figure 5) was modified to include the method GetShapePoints(), in order to get the face points in real time. Although, this method was declared on the library SDK was never implemented.

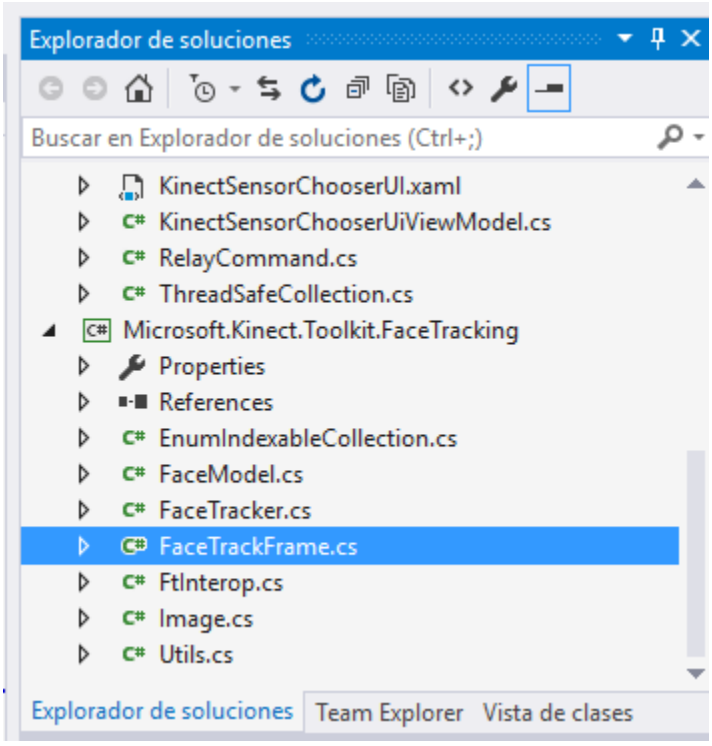

Figure 5 FaceTrackFrame in solution explorer Microsoft Visual Studio 2015

```
// populates an array for the ShapePoints
    public PointF[] GetShapePoints()
    {
      // get the 2D tracked shapes
      IntPtr pointsPtr = IntPtr.Zero;
      uint pointCount = 0;
  this.faceTrackingResultPtr.Get2DShapePoints(out pointsPtr, out pointCount);
      if (pointCount == 0)
      {
        return null;
      }

      // create our array to hold the points
      PointF[] shapePoints = new PointF[pointCount];

      int sizeInBytes = Marshal.SizeOf(typeof(PointF));
      for (int i = 0; i < pointCount; i++)
      {
        IntPtr faceShapePointsPtr;
```

```
            if (IntPtr.Size == 8)
            {
                // 64bit
        faceShapePointsPtr = new IntPtr(pointsPtr.ToInt64() + (i * sizeInBytes));
            }
            else
            {
                // 32bit
        faceShapePointsPtr = new IntPtr(pointsPtr.ToInt32() + (i * sizeInBytes));
            }

            // copy the data
            shapePoints[i] = (PointF)Marshal.PtrToStructure(faceShapePointsPtr, typeof(PointF));
          }

          return shapePoints;
        }

      }
    }
```

Above, we can see the code included to get the shape points in 2D, this library was modified only inside this project so I copied the toolkit into project folder and modified from there. Do not modify the toolkit located at C:/ folder to avoid incompatibilities with different functions. After that I created a project based on the test development kit of SDK 1.8, in Figure 6 and Figure 7 we can observe the results. In yellow are displayed the 2D points of the face detected by the Kinect by a mesh. In green, we can see the points that Kinect detects as an AU, so the points in green are the one which gives the most valuable information to process it later.

Figure 6 2D mesh points

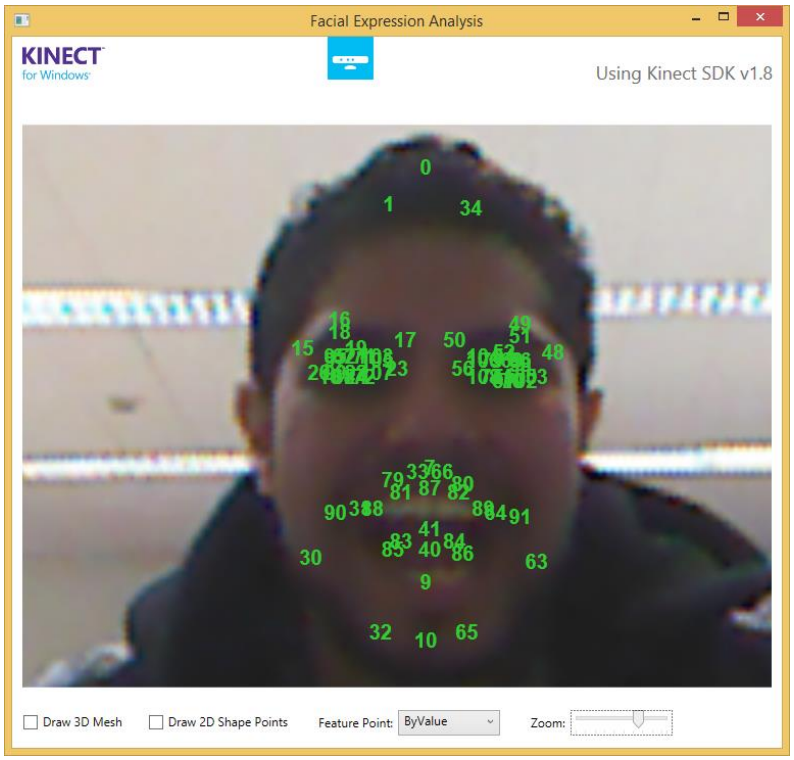

Figure 7 2D Mesh point by feature

Due lack of time in this research part, we had not enough time to contrast and measure the values of each AU and detect emotions as itself. But the main achievement acquired was to obtain the points of the face in 2D to process them.

4.3. Working with Kinect 2.0

The second part of this research is to understand the basis of the newer Kinect, thus in this section we will cover the achievements acquired.

4.3.1. Install Kinect sdk 2.0

In order to handle the newest Kinect (the one which come with Xbox-One) we need to download the proper sdk, this is version 2, and we can do this using next link:

https://www.microsoft.com/en-us/download/details.aspx?id=44561

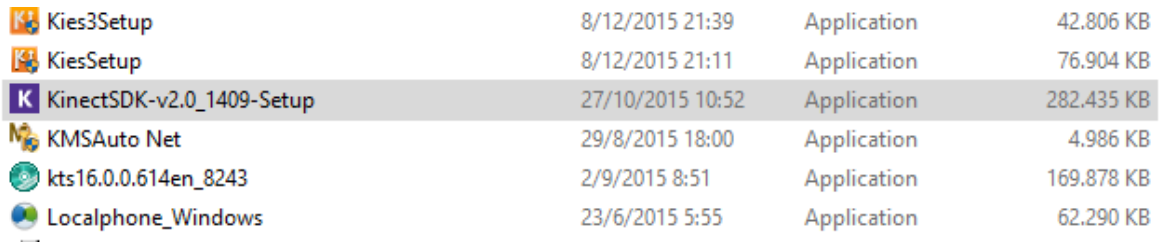

Figure 8 Installation file for the Kinect

Once download is complete, please open the file and Figure 9 will pop-up.

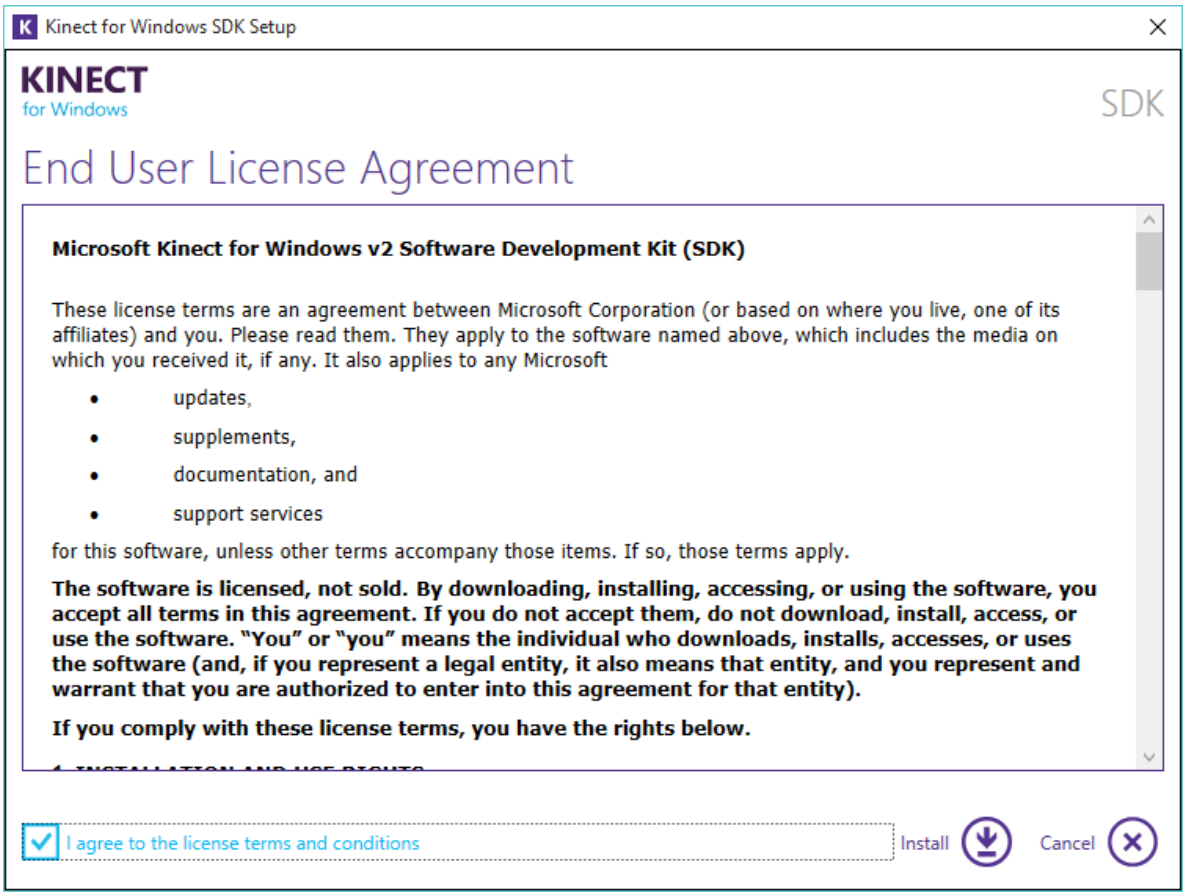

Figure 9 User agreement

Accept license terms and click on install, the whole average process will start and when it is complete will show Figure 10

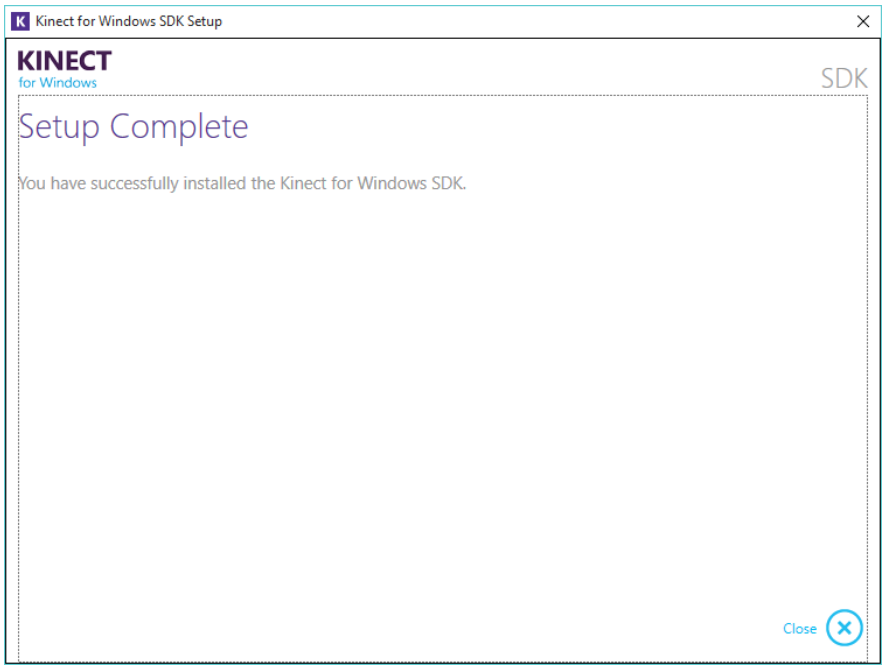

Figure 10 Kinect SDK v2.0 install complete

Now, we need to test if SDK is working, now please go to the following path:

C:\Program Files\Microsoft SDKs\Kinect\v2.0_1409\Tools\ConfigurationVerifier

Connect Kinect v2 to the PC and run *KinectV2ConfigurationVerifier*

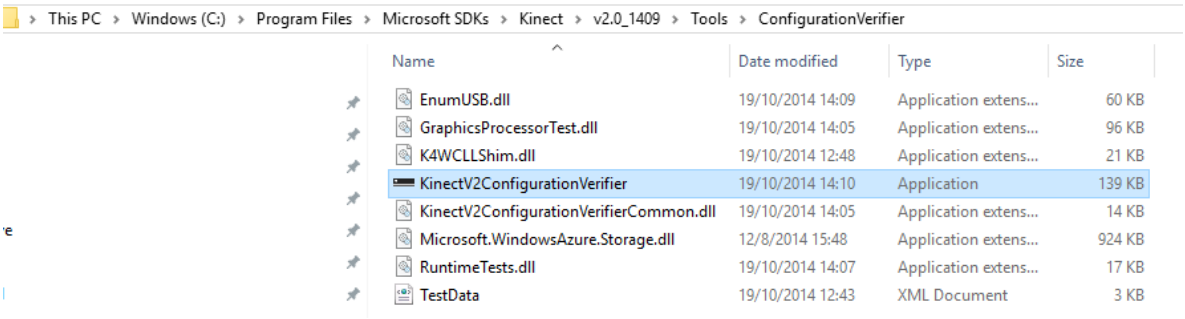

Figure 11 Configuration Verifier

If everything is running ok Figure 12 will be shown, take in mind that in order to Kinect V2 works it should be plugged into an USB 3.0 port.

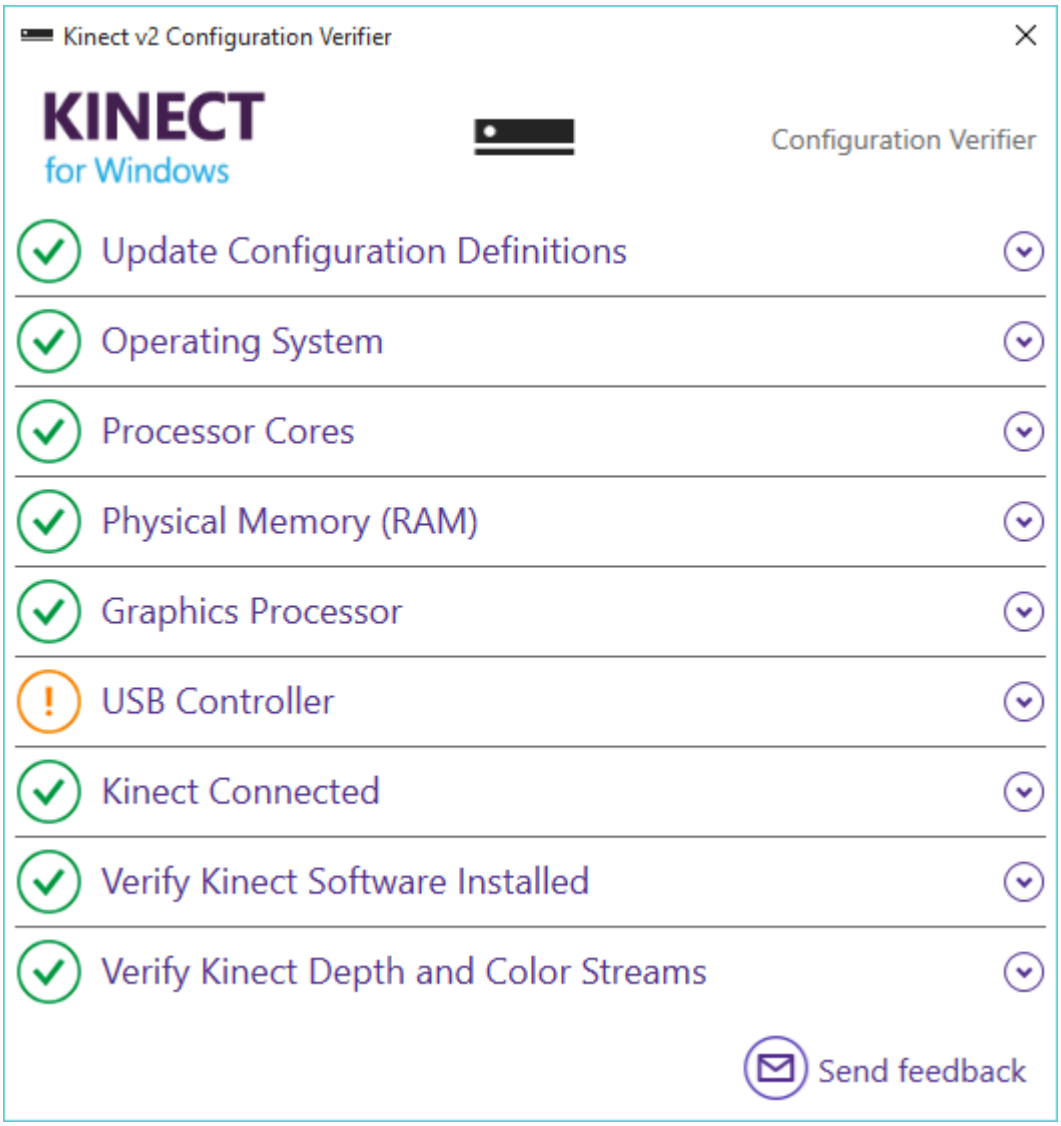

Figure 12 Configuration Verify

In our case USB controller is mark as warning because we are not working with an USB 3.0 standardized by Microsoft, however it will work perfectly.

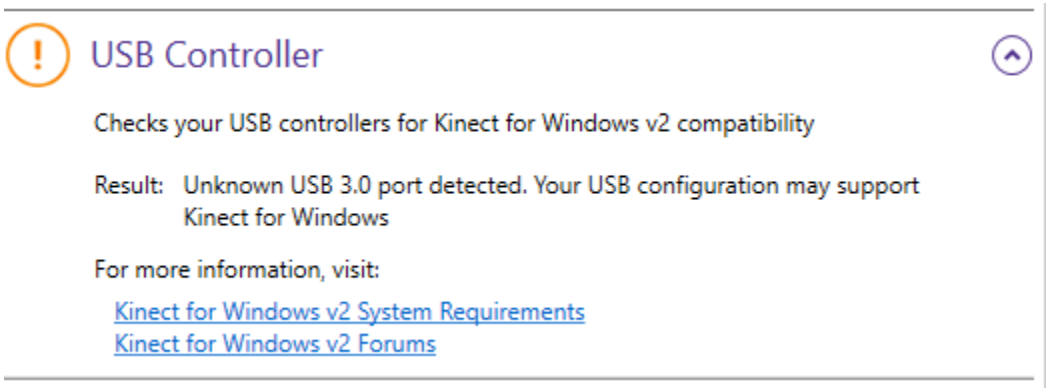

Figure 13 USB 3.0 verify

Now, we have finished Kinect V2 installation.

4.3.2. Testing Kinect functionalities

In order to test some of the functionalities that SDK v2.0 provides, please go to the path:

```
C:\Program Files\Microsoft SDKs\Kinect\v2.0_1409\Tools\SDKBrowser
```

And open the *SDKBrowser* application:

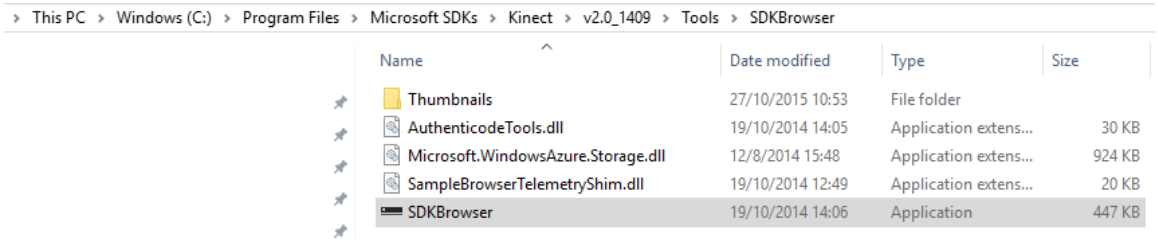

Figure 14 SDK browser application

Select the Face Basis-D2D to check what we can do with faces with this Kinect.

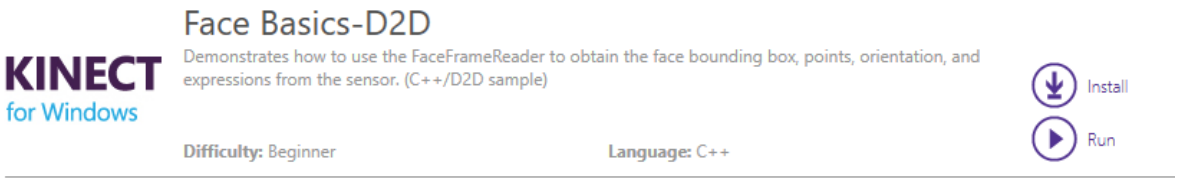

Figure 15 Face Basiscs-D2D

With this feature of Kinect we can detect face movements and expressions such as:

- Happy
- Engaged
- Wearing glasses
- Left/Right eye closed

- Mouth open

In Figure 16, Figure 17 AND Figure 18 show some examples of this library.

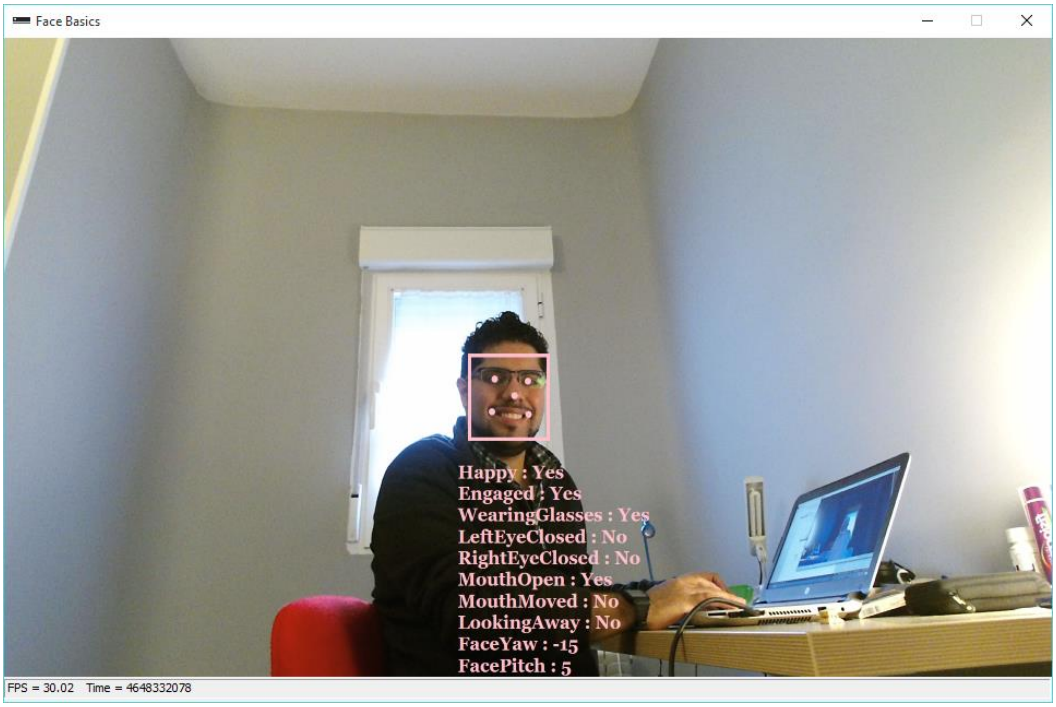

Figure 16 Face detection while wearing glasses

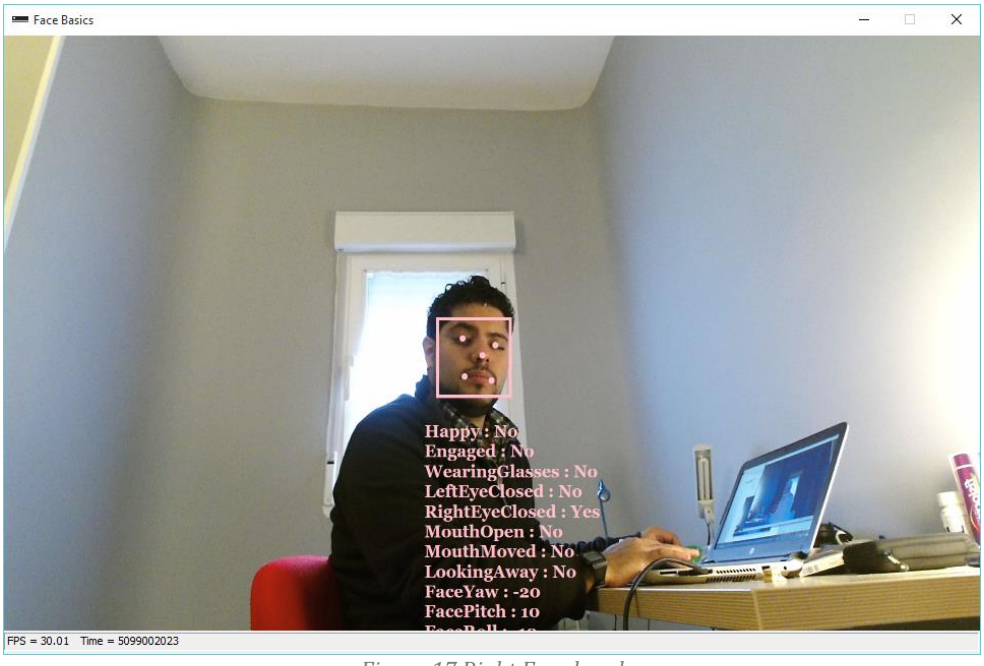

Figure 17 Right Eye closed

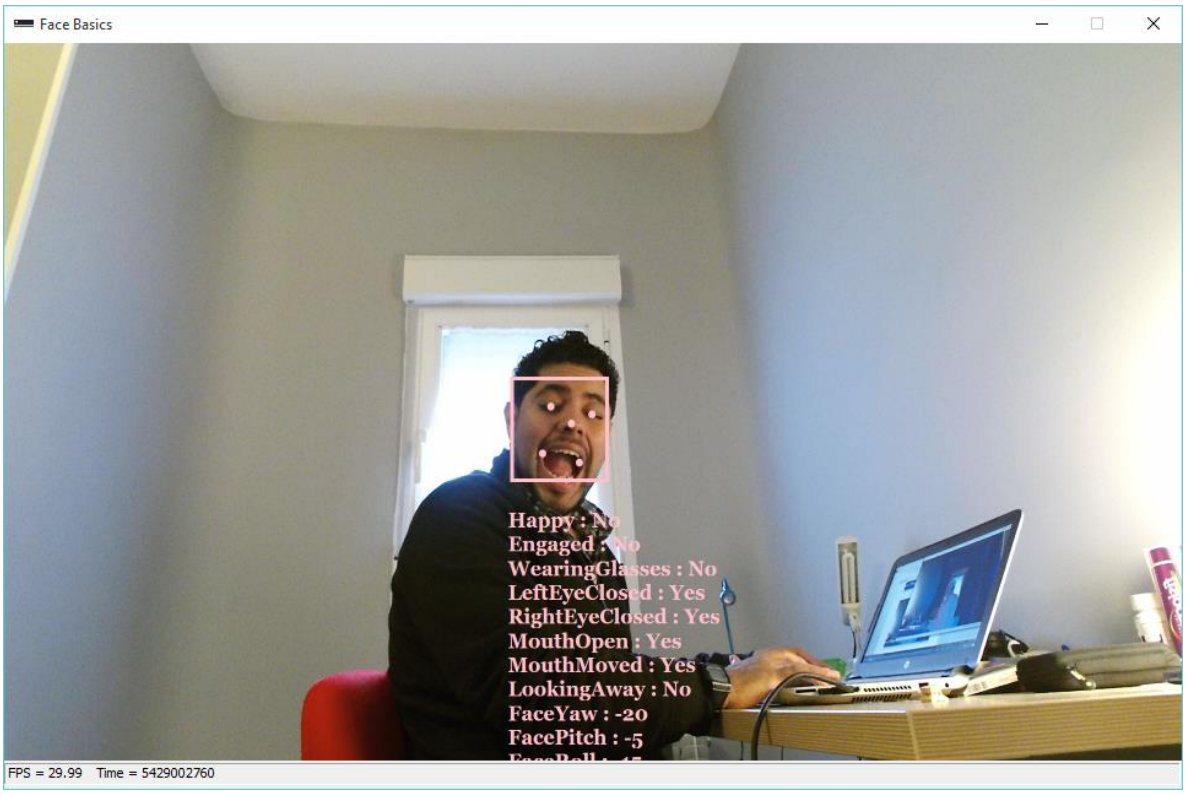

Figure 18 mouth opened and both eyes closed

With this is important to know that Kinect 2 has included native method to detects AU, in this way it is easier to program know features/emotions as mentioned above. However, if you want to detect and create your own applications new code must be implemented.

One thing that should be cared before programing in this new Kinect is the "Sources" Kinect 1 consider only 1 source of information, however Kinect 2 consider independently each source, color, depth, infrared, body, face. In this way it can process different things in different ways, however the methods used in the SDK 1.8 are not the same in SDK 2.0 due to this change.

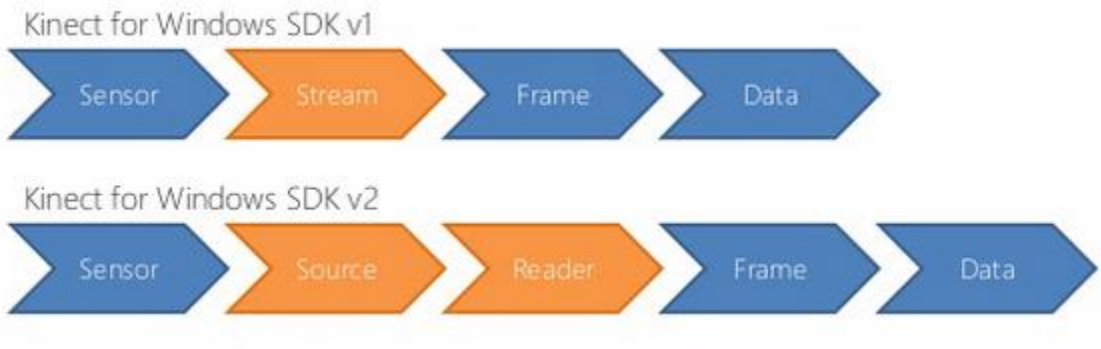

Figure 19 Programing flow

4.3.3. Key differences in program

Imports

The headers are different due use of different SDK, in this case we only consider the face features. However, if you want to use body should import the correspondingly libraries.

```
using System.Windows;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using Microsoft.Kinect;
using Microsoft.Kinect.Face;
```

In this example we are using the face in 2D, if we want to use the face in 3D one more library must be included Microsoft.Kinect.HDFace;

Definitions of sources and readers

As been said before, the sensor has multiples readers and this must be defined and initialized, in this case we use a bodyReader to track a body and a face source to get the face points.

```
/// <summary>
    /// Body reader
    /// </summary>
    public BodyFrameReader _bodyReader;

    /// <summary>
    /// Collection of all tracked bodies
    /// </summary>
    public Body[] _bodies;

    /// <summary>
    /// Requested face features
    /// </summary>

    /// <summary>
    /// Face Source
    /// </summary>
    private FaceFrameSource _faceSource;

    /// <summary>
    /// Face Reader
    /// </summary>
    private FaceFrameReader _faceReader;
```

The source reader is the object that has implemented method to get the AUs:

```
private const FaceFrameFeatures _faceFrameFeatures = FaceFrameFeatures.BoundingBoxInInfraredSpace
                                                   | FaceFrameFeatures.PointsInInfraredSpace
                                                   | FaceFrameFeatures.MouthMoved
                                                   | FaceFrameFeatures.MouthOpen
                                                   | FaceFrameFeatures.LeftEyeClosed
                                                   | FaceFrameFeatures.RightEyeClosed
                                                   | FaceFrameFeatures.LookingAway
                                                   | FaceFrameFeatures.Happy
                                                   | FaceFrameFeatures.FaceEngagement
                                                   | FaceFrameFeatures.Glasses;
```

4.3.4. Working with Face library

With this a complete code can be made (check for annex section for full code), the results are shown in

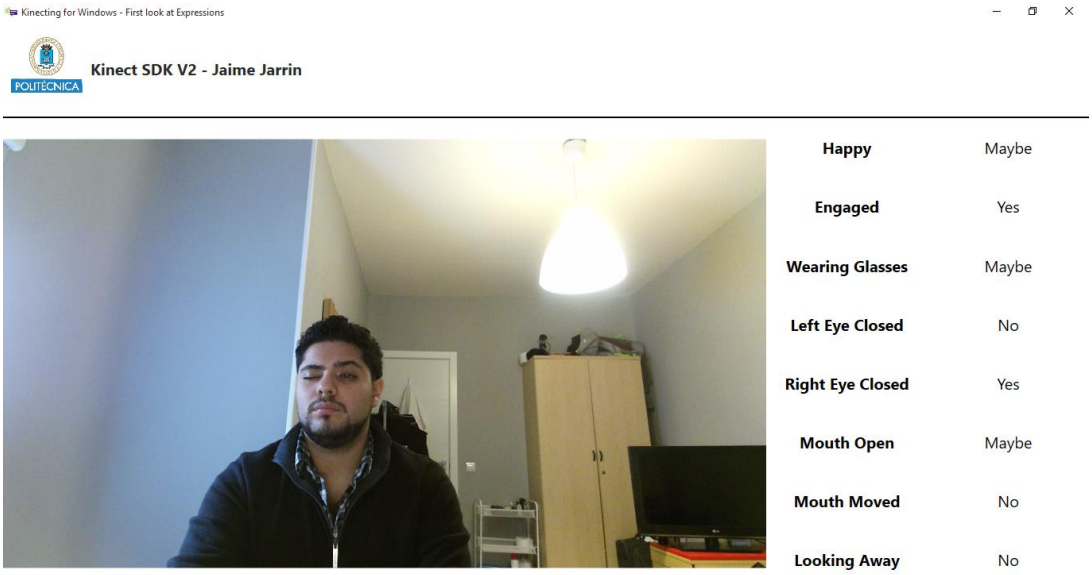

Figure 20 Test environment – right eye closed

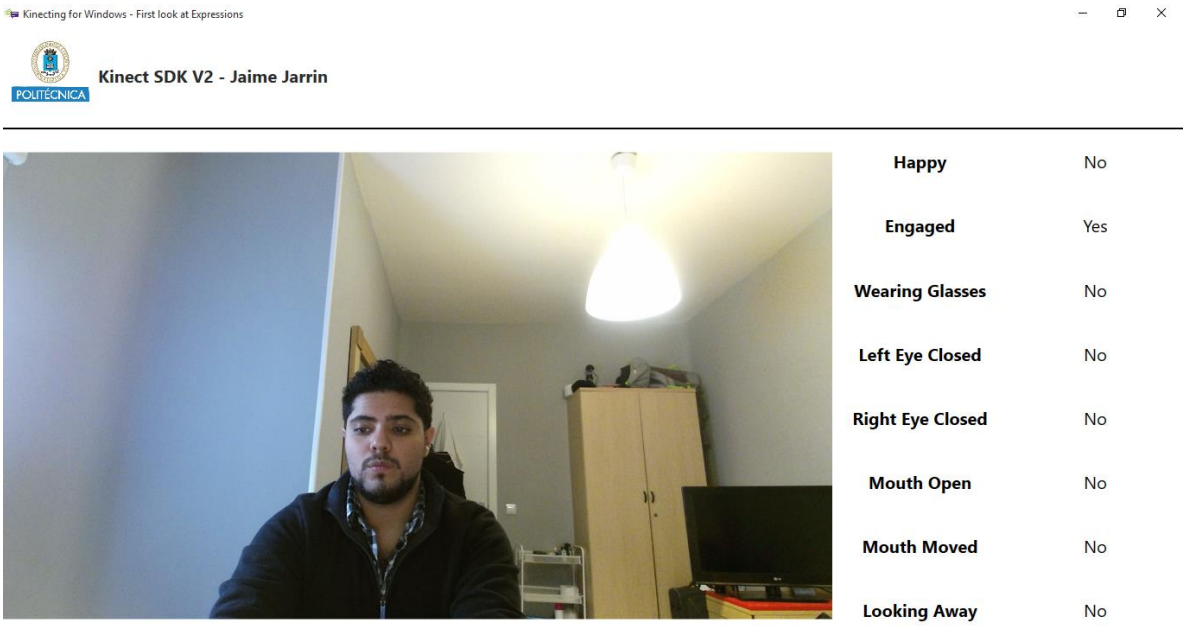

Figure 21 test environment – normal status

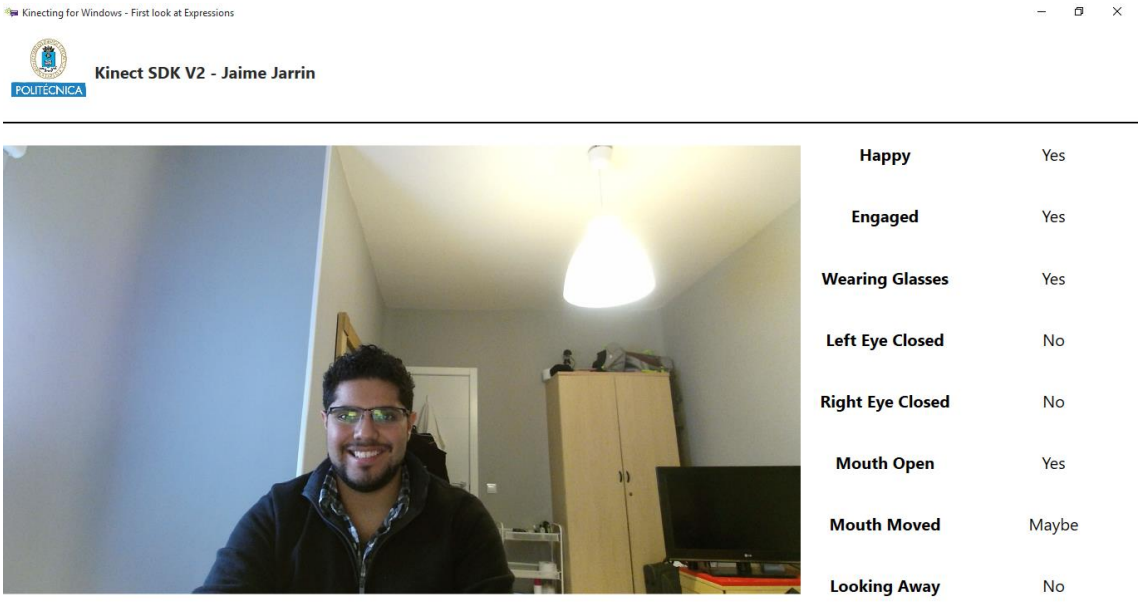

Figure 22 Happy wearing glasses

4.3.5. Working with HD Face Library

This is a 3D capabilities library which can detect many features of the face, thus it can sense more muscle's movements and has more sensibility. In fact, this library, detects around 1330 points on the face, however the ones that give us valuable information are the following:

```
EyeLeft = 0
LefteyeInnercorner = 210
LefteyeOutercorner = 469
LefteyeMidtop = 241
LefteyeMidbottom = 1104
RighteyeInnercorner = 843
RighteyeOutercorner = 1117
RighteyeMidtop = 731
RighteyeMidbottom = 1090
LefteyebrowInner = 346
LefteyebrowOuter = 140
LefteyebrowCenter = 222
RighteyebrowInner = 803
RighteyebrowOuter = 758
RighteyebrowCenter = 849
MouthLeftcorner = 91
MouthRightcorner = 687
MouthUpperlipMidtop = 19
MouthUpperlipMidbottom = 1072
MouthLowerlipMidtop = 10
MouthLowerlipMidbottom = 8
NoseTip = 18
NoseBottom = 14
NoseBottomleft = 156
NoseBottomright = 783
NoseTop = 24
```

NoseTopleft = 151
NoseTopright = 772
ForeheadCenter = 28
LeftcheekCenter = 412
RightcheekCenter = 933
Leftcheekbone = 458
Rightcheekbone = 674
ChinCenter = 4
LowerjawLeftend = 1307
LowerjawRightend = 1327

With this amount of points it is possible to plot faces as shown in Figure 23. A 3D mask is put in top of the face to detect the characteristics of muscles movements.

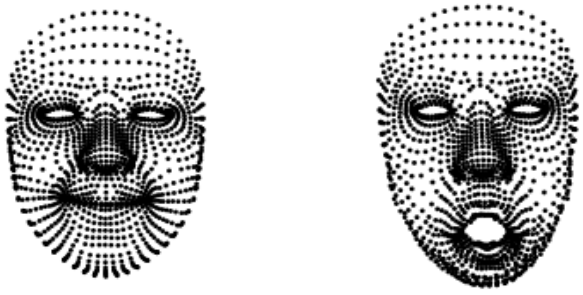

Figure 23 HD face

(Happy on the left – open mouth on the right)

Due the complexity of the second version of Kinect, it wasn't possible to automatically detect emotions using the HD face library, however we can store the values in a vector (X,Y,Z) and compare it with a base to detect emotions. This can be related as future work.

Useful Links

- Key Introduction to know the initial procedures of Kinect 2.0

  http://es.slideshare.net/SugiuraTsukasa/kinect-v2-introduction-and-tutorial

- Tutorial to get into Kinect sdk 2.0 (tutorial and examples)

  http://kinect.github.io/tutorial/

- Face tracking and HD Face webinar

  https://blogs.msdn.microsoft.com/kinectforwindows/2014/01/31/mysteries-of-kinect-for-windows-face-tracking-output-explained/

- Explanation of Kinect points in version 1.8

https://social.msdn.microsoft.com/Forums/en-US/13f8722e-c17e-4aa5-b0dd-67ffa8ff9f79/confused-about-face-tranking-points-and-animations?forum=kinectv2sdk

- Explanation of face points on Kinect 2

  https://msdn.microsoft.com/en-us/library/microsoft.kinect.face.highdetailfacepoints.aspx

  http://stc.fs.cvut.cz/pdf13/2659.pdf

- Frame processing in Kinect 1 vs Kinect 2

  http://www.kinectingforwindows.com/2014/03/03/gen-ii-kinect-basics-overview/

- Microsoft Virtual Academy

  https://mva.microsoft.com/en-us/training-courses/programming-kinect-for-windows-v2-jump-start-9088?l=WiRzgLf4_904984382

- Kinect API

  https://ptgmedia.pearsoncmg.com/images/9780735663961/samplepages/9780735663961.pdf

5. References

[1] https://msdn.microsoft.com/en-us/library/aa645597%28v=vs.71%29.aspx
[2] https://en.wikipedia.org/wiki/Facial_Action_Coding_System
[3] http://research.microsoft.com/en-us/events/fs2011/jancke_kinect_programming.pdf
[4] https://channel9.msdn.com/Series/Programming-Kinect-for-Windows-v2/01
[5] https://msdn.microsoft.com/en-us/library/dn785525.aspx
[6] https://msdn.microsoft.com/en-us/library/microsoft.kinect.face.highdetailfacepoints.aspx

6. Annex

6.1. Full code of version 1.8, Main window

```
// ------------------------------------------------------------------------
// <copyright file="MainWindow.xaml.cs" company="Microsoft">
//     Copyright (c) Microsoft Corporation.  All rights reserved.
// </copyright>
// ------------------------------------------------------------------------

namespace FaceTrackingBasics
{
    using System;
    using System.Windows;
    using System.Windows.Data;
    using System.Windows.Media;
    using System.Windows.Media.Imaging;
    using Microsoft.Kinect;
    using Microsoft.Kinect.Toolkit;

    public enum DrawFeaturePoint : int
    {
        None = 0,
        ByName = 1,
        ByValue = 2,
    }

    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        private static readonly int Bgr32BytesPerPixel = (PixelFormats.Bgr32.BitsPerPixel + 7) / 8;
        private readonly KinectSensorChooser sensorChooser = new KinectSensorChooser();
        private WriteableBitmap colorImageWritableBitmap;
        private byte[] colorImageData;
        private ColorImageFormat currentColorImageFormat = ColorImageFormat.Undefined;

        public MainWindow()
        {
            InitializeComponent();

            var faceTrackingViewerBinding = new Binding("Kinect") { Source = sensorChooser };
            faceTrackingViewer.SetBinding(FaceTrackingViewer.KinectProperty, faceTrackingViewerBinding);

            sensorChooser.KinectChanged += SensorChooserOnKinectChanged;

            sensorChooser.Start();
        }
```

```
    private void SensorChooserOnKinectChanged(object sender, KinectChangedEventArgs
kinectChangedEventArgs)
    {
      KinectSensor oldSensor = kinectChangedEventArgs.OldSensor;
      KinectSensor newSensor = kinectChangedEventArgs.NewSensor;

      if (oldSensor != null)
      {
        oldSensor.AllFramesReady -= KinectSensorOnAllFramesReady;
        oldSensor.ColorStream.Disable();
        oldSensor.DepthStream.Disable();
        oldSensor.DepthStream.Range = DepthRange.Default;
        oldSensor.SkeletonStream.Disable();
        oldSensor.SkeletonStream.EnableTrackingInNearRange = false;
        oldSensor.SkeletonStream.TrackingMode = SkeletonTrackingMode.Default;
      }

      if (newSensor != null)
      {
        try
        {
          newSensor.ColorStream.Enable(ColorImageFormat.RgbResolution640x480Fps30);
          newSensor.DepthStream.Enable(DepthImageFormat.Resolution640x480Fps30);
          try
          {
            // This will throw on non Kinect For Windows devices.
            newSensor.DepthStream.Range = DepthRange.Near;
            newSensor.SkeletonStream.EnableTrackingInNearRange = true;
          }
          catch (InvalidOperationException)
          {
            newSensor.DepthStream.Range = DepthRange.Default;
            newSensor.SkeletonStream.EnableTrackingInNearRange = false;
          }

          newSensor.SkeletonStream.TrackingMode = SkeletonTrackingMode.Seated;
          newSensor.SkeletonStream.Enable();
          newSensor.AllFramesReady += KinectSensorOnAllFramesReady;
        }
        catch (InvalidOperationException)
        {
          // This exception can be thrown when we are trying to
          // enable streams on a device that has gone away.  This
          // can occur, say, in app shutdown scenarios when the sensor
          // goes away between the time it changed status and the
          // time we get the sensor changed notification.
          //
          // Behavior here is to just eat the exception and assume
          // another notification will come along if a sensor
          // comes back.
        }
      }
    }

    private void WindowClosed(object sender, EventArgs e)
    {
```

```
        sensorChooser.Stop();
        faceTrackingViewer.Dispose();
    }

    private void KinectSensorOnAllFramesReady(object sender, AllFramesReadyEventArgs
allFramesReadyEventArgs)
    {
        using (var colorImageFrame = allFramesReadyEventArgs.OpenColorImageFrame())
        {
            if (colorImageFrame == null)
            {
                return;
            }

            // Make a copy of the color frame for displaying.
            var haveNewFormat = this.currentColorImageFormat != colorImageFrame.Format;
            if (haveNewFormat)
            {
                this.currentColorImageFormat = colorImageFrame.Format;
                this.colorImageData = new byte[colorImageFrame.PixelDataLength];
                this.colorImageWritableBitmap = new WriteableBitmap(
                    colorImageFrame.Width, colorImageFrame.Height, 96, 96, PixelFormats.Bgr32, null);
                ColorImage.Source = this.colorImageWritableBitmap;
            }

            colorImageFrame.CopyPixelDataTo(this.colorImageData);
            this.colorImageWritableBitmap.WritePixels(
                new Int32Rect(0, 0, colorImageFrame.Width, colorImageFrame.Height),
                this.colorImageData,
                colorImageFrame.Width * Bgr32BytesPerPixel,
                0);
        }
    }

    private void chkDraw3DMesh_Click(object sender, RoutedEventArgs e)
    {
        bool selected = false;
        if (chkDraw3DMesh.IsChecked == true)
        {
            selected = true;
        }
        faceTrackingViewer.DrawMesh(selected);
    }

    private void chkDrawShapePoints_Click(object sender, RoutedEventArgs e)
    {
        bool selected = false;
        if (chkDrawShapePoints.IsChecked == true)
        {
            selected = true;
        }
        faceTrackingViewer.DrawShapePoints(selected);
    }

    private void cboDrawFeaturePoints_SelectionChanged(object sender,
System.Windows.Controls.SelectionChangedEventArgs e)
    {
```

```
        if (cboDrawFeaturePoints.SelectedIndex > -1)
        {

faceTrackingViewer.DrawFeaturePoints((DrawFeaturePoint)cboDrawFeaturePoints.SelectedIndex);
        }
      }

  }
}
```

6.2. Full code of version 2 Kinect Face Library

```
using System.Windows;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using Microsoft.Kinect;
using Microsoft.Kinect.Face;

namespace K4W.Expressions
{
  public partial class MainWindow : Window
  {
    /// <summary>
    /// Instance of Kinect sensor
    /// </summary>
    private KinectSensor _Kinect;

    /// <summary>
    /// Body reader
    /// </summary>
    public BodyFrameReader _bodyReader;

    /// <summary>
    /// Collection of all tracked bodies
    /// </summary>
    public Body[] _bodies;

    /// <summary>
    /// Requested face features
    /// </summary>
    private const FaceFrameFeatures _faceFrameFeatures =
FaceFrameFeatures.BoundingBoxInInfraredSpace
                                | FaceFrameFeatures.PointsInInfraredSpace
                                | FaceFrameFeatures.MouthMoved
                                | FaceFrameFeatures.MouthOpen
                                | FaceFrameFeatures.LeftEyeClosed
                                | FaceFrameFeatures.RightEyeClosed
                                | FaceFrameFeatures.LookingAway
                                | FaceFrameFeatures.Happy
                                | FaceFrameFeatures.FaceEngagement
                                | FaceFrameFeatures.Glasses;

    /// <summary>
```

```
        /// Face Source
        /// </summary>
        private FaceFrameSource _faceSource;

        /// <summary>
        /// Face Reader
        /// </summary>
        private FaceFrameReader _faceReader;

        /// <summary>
        /// Default CTOR
        /// </summary>
        public MainWindow()
        {
            // Initialize Components
            InitializeComponent();

            // Initialize Kinect
            InitializeKinect();
        }

        /// <summary>
        /// Initialize Kinect
        /// </summary>
        private void InitializeKinect()
        {
            // Get Kinect sensor
            _Kinect = KinectSensor.GetDefault();

            if (_Kinect == null) return;

            // Initialize Camera
            InitializeCamera();

            // Initialize body tracking
            InitializeBodyTracking();

            // Start receiving
            _kinect.Open();
        }

        /// <summary>
        /// Initialize body tracking
        /// </summary>
        private void InitializeBodyTracking()
        {
            // Body Reader
            _bodyReader = _kinect.BodyFrameSource.OpenReader();

            // Wire event
            _bodyReader.FrameArrived += OnBodyFrameReceived;
        }

        /// <summary>
        /// Process body frames
        /// </summary>
        private void OnBodyFrameReceived(object sender, BodyFrameArrivedEventArgs e)
```

```
    {
        // Get Frame ref
        BodyFrameReference bodyRef = e.FrameReference;

        if (bodyRef == null) return;

        // Get body frame
        using (BodyFrame frame = bodyRef.AcquireFrame())
        {
            if (frame == null) return;

            // Allocate array when required
            if (_bodies == null)
                _bodies = new Body[frame.BodyCount];

            // Refresh bodies
            frame.GetAndRefreshBodyData(_bodies);

            foreach (Body body in _bodies)
            {
                if (body.IsTracked && _faceSource == null)
                {
                    // Create new sources with body TrackingId
                    _faceSource = new FaceFrameSource(_Kinect, body.TrackingId, _faceFrameFeatures);

                    // Create new reader
                    _faceReader = _faceSource.OpenReader();

                    // Wire events
                    _faceReader.FrameArrived += OnFaceFrameArrived;
                    _faceSource.TrackingIdLost += OnTrackingIdLost;
                }
            }
        }
    }

    /// <summary>
    /// Process the face frame
    /// </summary>
    private void OnFaceFrameArrived(object sender, FaceFrameArrivedEventArgs e)
    {
        // Retrieve the face reference
        FaceFrameReference faceRef = e.FrameReference;

        if (faceRef == null) return;

        // Acquire the face frame
        using (FaceFrame faceFrame = faceRef.AcquireFrame())
        {
            if (faceFrame == null) return;

            // Retrieve the face frame result
            FaceFrameResult frameResult = faceFrame.FaceFrameResult;

            // Display the values
            HappyResult.Text = frameResult.FaceProperties[FaceProperty.Happy].ToString();
            EngagedResult.Text = frameResult.FaceProperties[FaceProperty.Engaged].ToString();
```

```
            GlassesResult.Text = frameResult.FaceProperties[FaceProperty.WearingGlasses].ToString();
            LeftEyeResult.Text = frameResult.FaceProperties[FaceProperty.LeftEyeClosed].ToString();
            RightEyeResult.Text = frameResult.FaceProperties[FaceProperty.RightEyeClosed].ToString();
            MouthOpenResult.Text = frameResult.FaceProperties[FaceProperty.MouthOpen].ToString();
            MouthMovedResult.Text = frameResult.FaceProperties[FaceProperty.MouthMoved].ToString();
            LookingAwayResult.Text = frameResult.FaceProperties[FaceProperty.LookingAway].ToString();
        }
    }

    /// <summary>
    /// Handle when the tracked body is gone
    /// </summary>
    private void OnTrackingIdLost(object sender, TrackingIdLostEventArgs e)
    {
        // Update UI
        HappyResult.Text = "No face tracked";
        EngagedResult.Text = "No face tracked";
        GlassesResult.Text = "No face tracked";
        LeftEyeResult.Text = "No face tracked";
        RightEyeResult.Text = "No face tracked";
        MouthOpenResult.Text = "No face tracked";
        MouthMovedResult.Text = "No face tracked";
        LookingAwayResult.Text = "No face tracked";

        // Reset values for next body
        _faceReader = null;
        _faceSource = null;
    }

    #region CAMERA
    /// <summary>
    /// Color WriteableBitmap linked to our UI
    /// </summary>
    private WriteableBitmap _colorBitmap = null;
    /// <summary>
    /// Array of color pixels
    /// </summary>
    private byte[] _colorPixels = null;

    /// <summary>
    /// FrameReader for our coloroutput
    /// </summary>
    private ColorFrameReader _colorReader = null;
    /// <summary>
    /// Size fo the RGB pixel in bitmap
    /// </summary>
    private readonly int _bytePerPixel = (PixelFormats.Bgr32.BitsPerPixel + 7) / 8;
    private void InitializeCamera()
    {
        if (_Kinect == null) return;

        // Get frame description for the color output
        FrameDescription desc = _kinect.ColorFrameSource.FrameDescription;

        // Get the framereader for Color
        _colorReader = _kinect.ColorFrameSource.OpenReader();
```

```
            // Allocate pixel array
            _colorPixels = new byte[desc.Width * desc.Height * _bytePerPixel];

            // Create new WriteableBitmap
            _colorBitmap = new WriteableBitmap(desc.Width, desc.Height, 96, 96, PixelFormats.Bgr32, null);

            // Link WBMP to UI
            CameraImage.Source = _colorBitmap;

            // Hook-up event
            _colorReader.FrameArrived += OnColorFrameArrived;
        }
        /// <summary>
        /// Process color frames & show in UI
        /// </summary>
        private void OnColorFrameArrived(object sender, ColorFrameArrivedEventArgs e)
        {
            // Get the reference to the color frame
            ColorFrameReference colorRef = e.FrameReference;

            if (colorRef == null) return;

            // Acquire frame for specific reference
            ColorFrame frame = colorRef.AcquireFrame();

            // It's possible that we skipped a frame or it is already gone
            if (frame == null) return;

            using (frame)
            {
                // Get frame description
                FrameDescription frameDesc = frame.FrameDescription;

                // Check if width/height matches
                if (frameDesc.Width == _colorBitmap.PixelWidth && frameDesc.Height ==
_colorBitmap.PixelHeight)
                {
                    // Copy data to array based on image format
                    if (frame.RawColorImageFormat == ColorImageFormat.Bgra)
                    {
                        frame.CopyRawFrameDataToArray(_colorPixels);
                    }
                    else frame.CopyConvertedFrameDataToArray(_colorPixels, ColorImageFormat.Bgra);

                    // Copy output to bitmap
                    _colorBitmap.WritePixels(
                            new Int32Rect(0, 0, frameDesc.Width, frameDesc.Height),
                            _colorPixels,
                            frameDesc.Width * _bytePerPixel,
                            0);
                }
            }
        }
        #endregion CAMERA
    }
}
```