Visão Computacional:

Reconhecimento da Mão Humana e Seus Movimentos

Projeto Final de Licenciatura

VOLUME II

ANEXOS

Código C# do Projeto

Aluno: Tiago Manuel Saraiva Marques

Aluno nº 20111501

Orientador: Professor Doutor Sérgio Nunes

Coorientador: Professor Doutor Mário Macedo

Barcarena, Novembro de 2014

Ficheiro: MainWindow.xaml

```
<Window x:Class="WpfApplication1.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="MainWindow" ResizeMode="NoResize"
SizeToContent="WidthAndHeight" WindowStartupLocation="CenterScreen"
Closed="Window_Closed" mc:Ignorable="d"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
d:DesignWidth="839">
    <StackPanel>
        <StackPanel Orientation="Horizontal">
            <StackPanel></StackPanel>
            <StackPanel>
                <Image Height="300" Name="depth" Width="400" />
            </StackPanel>
            <Grid Height="298" Name="grid1" Width="424">
                <Grid.ColumnDefinitions>
                    <ColumnDefinition Width="209*" />
                    <ColumnDefinition Width="215*" />
                </Grid.ColumnDefinitions>
                <Grid.RowDefinitions>
                    <RowDefinition Height="206*" />
                    <RowDefinition Height="92*" />
                </Grid.RowDefinitions>
                <Slider Height="23" HorizontalAlignment="Right"
Margin="0,38,43,0" Maximum="200" Minimum="-200" Name="sldYmodify"
VerticalAlignment="Top" Width="157" Grid.Row="1"
ValueChanged="sldYmodify_ValueChanged" />
                <Slider Height="23" HorizontalAlignment="Right"
Margin="0,10,43,0" Maximum="200" Minimum="-200" Name="sldXmodify"
VerticalAlignment="Top" Width="157" Grid.Row="1"
ValueChanged="sldXmodify_ValueChanged" />
                <Label Content="0" Height="23" HorizontalAlignment="Right"
Margin="0,9,3,0" Name="lblXmodify" VerticalAlignment="Top" Width="42"
Grid.Row="1" />
                <Label Content="0" Height="23" HorizontalAlignment="Right"
Margin="0,37,3,0" Name="lblYmodify" VerticalAlignment="Top" Width="42"
Grid.Row="1" />
                <Border BorderBrush="Silver" BorderThickness="1"
Height="210" HorizontalAlignment="Left" Name="borderLeft"
VerticalAlignment="Top" Width="209" Margin="0,-4,0,0">
                    <Image Height="187" Name="imgLeftHand" Stretch="Fill"
Width="187" />
                </Border>
                <Border BorderBrush="Silver" BorderThickness="1"
Height="210" HorizontalAlignment="Left" Margin="0,-4,0,0"
Name="borderRight" VerticalAlignment="Top" Width="209" Grid.Column="1">
                    <Image Height="187" Name="imgRightHand" Stretch="Fill"
Width="187" />
                </Border>
                <TextBox Grid.Column="1" Grid.Row="1" Height="23"
HorizontalAlignment="Left" Margin="25,35,0,0" Name="txtAngle"
VerticalAlignment="Top" Width="162" Text="7" />
                <Button Content="Apply" Grid.Column="1" Grid.Row="1"
```

```
Height="23" HorizontalAlignment="Left" Margin="127,64,0,0" Name="btnApply"
VerticalAlignment="Top" Width="60" Click="btnApply_Click" />
                <Label Content="Elevation Angle(-27~+27)" Grid.Column="1"
Grid.Row="1" Height="28" HorizontalAlignment="Left" Margin="25,12,0,0"
Name="label1" VerticalAlignment="Top" />
            </Grid>
        </StackPanel>
        <StackPanel Orientation="Horizontal"></StackPanel>
    </StackPanel>
</Window>
```

Ficheiro: MainWindow.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using Microsoft.Kinect;

namespace WpfApplication1 {
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window {
        KinectSensor sensor;
        //globais usadas para a controlar a conversao da informação de
profundidade em falsa cor (escala de cinza)
        const int RED_IDX = 2;
        const int GREEN_IDX = 1;
        const int BLUE_IDX = 0;
        short[] depthFrame32 = new short[320 * 240 * 4];
        int HandLeftX, HandLeftY, HandRightX, HandRightY;
        public MainWindow() {
            InitializeComponent();
            sldXmodify.Value = -40;
            sldYmodify.Value = -30;
            // verifica se existe algum sensor ligado
            if (KinectSensor.KinectSensors.Count == 0){
                MessageBox.Show("Não foi detectado nenhum sensor kinect",
"Camera Viewer");
                Application.Current.Shutdown();
            }
            // caso exista liga-se ao primeiro
            sensor = KinectSensor.KinectSensors[0];
            // activa o kinect com a camara de profundidade e trata eventuais
erros
            try {
                sensor.SkeletonStream.Enable();

sensor.DepthStream.Enable(DepthImageFormat.Resolution320x240Fps30);
                sensor.Start();
            }
            catch {
                MessageBox.Show("Inicialização falhada", "Camera Viewer");
                Application.Current.Shutdown();
            }
            sensor.AllFramesReady += new
```

```
EventHandler<AllFramesReadyEventArgs>(sensor_AllFramesReady);
            txtAngle.Text = sensor.ElevationAngle.ToString();
        }
        bool isMakingAFist(ImageSource imgHand) {
            bool wasBlack = false;
            int BlackWidth = 0;
            int BlackTimes = 0;
            for (int yy = 10; yy < imgHand.Height - 10; yy += 10) {
                for (int xx = 3; xx </*MaxX*/ imgHand.Width; xx++) {
                    if (PixelColor(imgHand, xx, yy) == Colors.Black) {
                        if (!wasBlack) {
                            if (BlackWidth > 1 && BlackWidth < 15)
                            { BlackTimes++; }
                            BlackWidth = 0;
                        }
                        else{
                            BlackWidth++;
                        }
                        wasBlack = true;
                    }
                    else{
                        wasBlack = false;
                    }
                }
                if (BlackTimes > 1) { return false; }
            }
            return true;
        }
        Color PixelColor(ImageSource img, int PixelX, int PixelY) {
            /// guarda a cor do pixel num array (RGB) e retorna-a
            CroppedBitmap cb = new CroppedBitmap((BitmapSource)img, new
Int32Rect(PixelX, PixelY, 1, 1));
            byte[] pix = new byte[4];
            cb.CopyPixels(pix, 4, 0);
            return Color.FromRgb(pix[2], pix[1], pix[0]);
        }
        private Point getDisplayPosition(DepthImageFrame depthFrame, Joint
joint) {
            float depthX, depthY;
            DepthImagePoint depthPoint =
sensor.MapSkeletonPointToDepth(joint.Position,
DepthImageFormat.Resolution320x240Fps30);

            depthX = depthPoint.X;
            depthY = depthPoint.Y;

            depthX = Math.Max(0, Math.Min(depthX * 320, 320));
            depthY = Math.Max(0, Math.Min(depthY * 240, 240));

            int colorX, colorY;
            ColorImagePoint colorPoint =
depthFrame.MapToColorImagePoint(depthPoint.X, depthPoint.Y,
sensor.ColorStream.Format);
            colorX = colorPoint.X;
            colorY = colorPoint.Y;

            return new Point((int)(depth.Width * colorX / 640.0) -30,
```

```
(int)(depth.Height * colorY / 480) - 30);
        }
        void sensor_AllFramesReady(object sender, AllFramesReadyEventArgs
e) {
            using (var depthFrame = e.OpenDepthImageFrame()) {
                if (depthFrame == null) { //MessageBox.Show("depthFrame
null");
                    return;
                }
                var bits = new short[depthFrame.PixelDataLength];
                depthFrame.CopyPixelDataTo(bits);

                short[] convertedDepthFrame = convertDepthFrame(bits);

                depth.Source = BitmapSource.Create(depthFrame.Width,
depthFrame.Height, 96, 96, PixelFormats.Bgr32, null, convertedDepthFrame,
depthFrame.Width * 4);
                try {
                    //if (HandLeftX == 0) { return; }

                    int intLeftX = HandLeftX + (int)sldXmodify.Value;
                    int intLeftY = HandLeftY + (int)sldYmodify.Value;

                    imgLeftHand.Source = new
CroppedBitmap((BitmapSource)depth.Source.CloneCurrentValue(), new
Int32Rect((intLeftX < 0) ? 0 : intLeftX, (intLeftY < 0) ? 0 : intLeftY,
(intLeftX + 50 >= depthFrame.Width) ? depthFrame.Width - intLeftX : 50,
(intLeftY + 50 >= depthFrame.Height) ? depthFrame.Height - intLeftY : 50));
                    borderLeft.Background =
(isMakingAFist(imgLeftHand.Source)) ? Brushes.Red : Brushes.White;

                    int intRightX = HandRightX + (int)sldXmodify.Value - 10;
                    int intRightY = HandRightY + (int)sldYmodify.Value;

                    imgRightHand.Source = new
CroppedBitmap((BitmapSource)depth.Source.CloneCurrentValue(), new
Int32Rect((intRightX < 0) ? 0 : intRightX, (intRightY < 0) ? 0 : intRightY,
(intRightX + 50 >= depthFrame.Width) ? depthFrame.Width - intRightX : 50,
(intRightY + 50 >= depthFrame.Height) ? depthFrame.Height - intRightY : 50));
                    borderRight.Background =
(isMakingAFist(imgRightHand.Source)) ? Brushes.Red : Brushes.White;
                }
                catch { }

                using (SkeletonFrame skeletonFrame = e.OpenSkeletonFrame())
{
                    if (skeletonFrame == null) {
//MessageBox.Show("skeletonFrame null");
                        return;
                    }
                    Skeleton[] esqueletos = new
Skeleton[skeletonFrame.SkeletonArrayLength];
                    skeletonFrame.CopySkeletonDataTo(esqueletos);
                    foreach (Skeleton esqueleto in esqueletos) {
                        if (SkeletonTrackingState.Tracked ==
esqueleto.TrackingState) {
                            // Draw joints
```

```
                            foreach (Joint joint in esqueleto.Joints) {
                                #region Update XY
                                switch (joint.JointType.ToString()) {
                                    case "HandLeft":
                                        Point getPl =
getDisplayPosition(depthFrame, joint);
                                        HandLeftX = (int)getPl.X;
                                        HandLeftY = (int)getPl.Y;
                                        break;
                                    case "HandRight":
                                        Point getPr =
getDisplayPosition(depthFrame, joint);
                                        HandRightX = (int)getPr.X;
                                        HandRightY = (int)getPr.Y;
                                        break;
                                }
                                #endregion Update XY
                            }
                        }
                    } // for each skeleton
                }
                //
            }
        }
        short[] convertDepthFrame(short[] depthFrame16) {
            // Converts a 16-bit grayscale depth frame which includes player
indexes into a 32-bit frame
            for (int i16 = 0, i32 = 0; i16 < depthFrame16.Length && i32 <
depthFrame32.Length; i16 += 2, i32 += 4) {
                int player = depthFrame16[i16] & 0x07;
                int realDepth = (depthFrame16[i16 + 1] << 5) |
(depthFrame16[i16] >> 3);
                // transform 13-bit depth information into an 8-bit intensity
appropriate
                // for display (we disregard information in most significant
bit)
                byte intensity = (byte)(255);//(255 - (255 * realDepth /
0x0fff));

                depthFrame32[i32 + RED_IDX] = 0; //(byte)(255 - intensity);
                depthFrame32[i32 + GREEN_IDX] = 0; //(byte)(255 - intensity);
                depthFrame32[i32 + BLUE_IDX] = 0; //(byte)(255 - intensity);
                if (player == 0) {
                    depthFrame32[i32 + RED_IDX] = (byte)(intensity / 2);
                    depthFrame32[i32 + GREEN_IDX] = (byte)(intensity / 2);
                    depthFrame32[i32 + BLUE_IDX] = (byte)(intensity / 2);
                }
            }
            return depthFrame32;
        }
        #region Change Label Value
        private void sldXmodify_ValueChanged(object sender,
RoutedPropertyChangedEventArgs<double> e) {
            lblXmodify.Content = sldXmodify.Value;
        }
        private void sldYmodify_ValueChanged(object sender,
RoutedPropertyChangedEventArgs<double> e) {
```

```
            lblYmodify.Content = sldYmodify.Value;
        }
        #endregion
        private void btnApply_Click(object sender, RoutedEventArgs e) {
            int i;
            bool CanBeInt = int.TryParse(txtAngle.Text, out i);
            if (CanBeInt && i > -28 && i < 28) { sensor.ElevationAngle = i;
}
        }
        private void Window_Closed(object sender, EventArgs e) {
            sensor.Stop();
            Environment.Exit(0);
        }
    }
}
```