<Window x:Class="KinectApplication.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

Title="MainWindow" Height="570" Width="748">

<Grid>

<Label Content="ConnectionId:"

HorizontalAlignment="Left"

Margin="10,10,0,0"

VerticalAlignment="Top"

RenderTransformOrigin="-0.734,0.113"/>

<Label Content="Status:"

HorizontalAlignment="Left"

Margin="10,41,0,0"

VerticalAlignment="Top"/>

<Label x:Name="ConnectionID"

HorizontalAlignment="Left"

Margin="111,13,0,0"

VerticalAlignment="Top"

RenderTransformOrigin="-0.734,0.113"

Height="23"

Width="134"/>

<Label x:Name="Status"

HorizontalAlignment="Left"

Margin="268,13,0,0"

VerticalAlignment="Top"

Height="26"

Width="156"/>

<Button x:Name="StartStopButton"

Content="Start"

HorizontalAlignment="Left"

Margin="10,72,0,0"

VerticalAlignment="Top"

Width="75"

Click="Start\_Click"/>

<Canvas x:Name="ImageCanvas"

HorizontalAlignment="Left"

Height="161"

Margin="329,0,0,0"

VerticalAlignment="Top"

Width="188"/>

<Canvas x:Name="VideoCanvas"

HorizontalAlignment="Left"

Height="480"

Margin="90,41,-213,-202"

VerticalAlignment="Top"

Width="640"/>

</Grid>

</Window>

namespace KinectApplication

{

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.Forms;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;

using Coding4Fun.Kinect.Wpf;

using Microsoft.Kinect;

public partial class MainWindow : Window

{

[System.Runtime.InteropServices.DllImport("user32.dll")]

static extern void keybd\_event(byte bVk, byte bScan, uint dwFlags, UIntPtr dwExtraInfo);

private void PressKey(Keys key)

{

const int KEYEVENTF\_EXTENDEDKEY = 0x1;

const int KEYEVENTF\_KEYUP = 0x2;

keybd\_event((byte)key, 0x45, KEYEVENTF\_EXTENDEDKEY, (UIntPtr)0);

}

private void UpKey(Keys key)

{

const int KEYEVENTF\_EXTENDEDKEY = 0x1;

const int KEYEVENTF\_KEYUP = 0x2;

keybd\_event((byte)key, 0x45, KEYEVENTF\_EXTENDEDKEY | KEYEVENTF\_KEYUP, (UIntPtr)0);

}

private const string ButtonStartText = "Start";

private const string ButtonStopText = "Stop";

private const string ButtonGrayscale = "Grayscale";

private const string ButtonColor = "Color";

private KinectSensor sensor;

private DepthImagePixel[] depthImagePixels;

public MainWindow()

{

InitializeComponent();

}

private void Start\_Click(object sender, RoutedEventArgs e)

{

if (this.StartStopButton.Content.ToString() == ButtonStartText)

{

if (KinectSensor.KinectSensors.Any())

{

KinectSensor.KinectSensors.StatusChanged += (o, args) =>

{

this.Status.Content = args.Status.ToString();

};

sensor = KinectSensor.KinectSensors.First();

}

sensor.Start();

sensor.ElevationAngle = 0;

sensor.ColorStream.Enable();

sensor.DepthStream.Enable();

sensor.SkeletonStream.Enable();

sensor.SkeletonStream.TrackingMode = SkeletonTrackingMode.Default; // Default is standing

sensor.AllFramesReady += sensorAllFramesReady;

this.ConnectionID.Content = sensor.DeviceConnectionId;

this.StartStopButton.Content = ButtonStopText;

}

else

{

if (sensor != null && sensor.IsRunning)

{

sensor.Stop();

StartStopButton.Content = ButtonStartText;

}

}

}

void sensorAllFramesReady(object sender, AllFramesReadyEventArgs e)

{

depthImagePixels = new DepthImagePixel[sensor.DepthStream.FramePixelDataLength];

using (var frame = e.OpenDepthImageFrame())

{

if (frame == null)

{

return;

}

frame.CopyDepthImagePixelDataTo(depthImagePixels);

}

using (var frame = e.OpenColorImageFrame())

{

if (frame == null)

{

return;

}

var bitmap = CreateBitmap(frame);

VideoCanvas.Background = new ImageBrush(bitmap);

}

using (var frame = e.OpenSkeletonFrame())

{

if (frame == null)

{

return;

}

var skeletons = new Skeleton[frame.SkeletonArrayLength];

frame.CopySkeletonDataTo(skeletons);

var skeleton = skeletons.FirstOrDefault(sk => sk.TrackingState == SkeletonTrackingState.Tracked);

if (skeleton == null)

{

return;

}

var rigthHandPosition = skeleton.Joints[JointType.HandRight].Position;

var leftHandPosition = skeleton.Joints[JointType.HandLeft].Position;

var headPosition = skeleton.Joints[JointType.Head].Position;

var armsPosition = skeleton.Joints[JointType.ShoulderCenter].Position;

var shoulderLeftPosition = skeleton.Joints[JointType.ShoulderLeft].Position;

var shoulderRigthPosition = skeleton.Joints[JointType.ShoulderRight].Position;

var hipCenterPosition = skeleton.Joints[JointType.HipCenter].Position;

var mapper = new CoordinateMapper(sensor);

var rightHandCoord = mapper.MapSkeletonPointToColorPoint(rigthHandPosition, ColorImageFormat.RgbResolution640x480Fps30);

var headCoord = mapper.MapSkeletonPointToColorPoint(headPosition, ColorImageFormat.RgbResolution640x480Fps30);

var armsCenterCoord = mapper.MapSkeletonPointToColorPoint(armsPosition, ColorImageFormat.RgbResolution640x480Fps30);

var shoulderLeftCoord = mapper.MapSkeletonPointToColorPoint(shoulderLeftPosition, ColorImageFormat.RgbResolution640x480Fps30);

var shoulderRightCoord = mapper.MapSkeletonPointToColorPoint(shoulderRigthPosition, ColorImageFormat.RgbResolution640x480Fps30);

var leftHandCoord = mapper.MapSkeletonPointToColorPoint(leftHandPosition, ColorImageFormat.RgbResolution640x480Fps30);

var hipCenterCoord = mapper.MapSkeletonPointToColorPoint(hipCenterPosition, ColorImageFormat.RgbResolution640x480Fps30);

this.DetectGestures(headCoord, rightHandCoord, leftHandCoord, armsCenterCoord, shoulderLeftCoord, shoulderRightCoord, hipCenterCoord);

}

}

private void DetectGestures(ColorImagePoint headCoord,

ColorImagePoint rightHandCoord, ColorImagePoint leftHandCoord,

ColorImagePoint armsCenterCoord,

ColorImagePoint shoulderLeftCoord, ColorImagePoint shoulderRightCoord,

ColorImagePoint hipCenterCoord)

{

var isRightHandPassive = rightHandCoord.Y > shoulderRightCoord.Y && rightHandCoord.Y < hipCenterCoord.Y

&& rightHandCoord.X > shoulderRightCoord.X && rightHandCoord.X < shoulderLeftCoord.X;

var isLeftHandPassive = leftHandCoord.Y > shoulderLeftCoord.Y && leftHandCoord.Y < hipCenterCoord.Y

&& leftHandCoord.X < shoulderLeftCoord.X && leftHandCoord.X > shoulderRightCoord.X;

if (!isRightHandPassive || !isLeftHandPassive)

{

// Check if there are at least one hand on the top of the head

var isRightHandUp = (rightHandCoord.Y < headCoord.Y);

var isLeftHandUp = (leftHandCoord.Y < headCoord.Y);

if (isRightHandUp || isLeftHandUp)

{

// SendKeys.SendWait("{UP}");

PressKey(Keys.Up);

this.Status.Content = "Up";

}

else

{

UpKey(Keys.Up);

}

// Check for left turn

var isHandOnLeft = leftHandCoord.Y > headCoord.Y && leftHandCoord.Y < hipCenterCoord.Y;

if (isHandOnLeft)

{

//SendKeys.SendWait("{LEFT}");

PressKey(Keys.Left);

this.Status.Content = "Left";

}

else

{

UpKey(Keys.Left);

}

// Check for right turn

var isHandOnRight = rightHandCoord.Y > headCoord.Y && rightHandCoord.Y < hipCenterCoord.Y;

if (isHandOnRight)

{

// SendKeys.SendWait("{RIGHT}");

PressKey(Keys.Right);

this.Status.Content = "Right";

}

else

{

UpKey(Keys.Right);

}

// Check for brakes

var isLeftHandDown = leftHandCoord.Y > hipCenterCoord.Y;

var isRightHandDown = rightHandCoord.Y > hipCenterCoord.Y;

if (isLeftHandDown || isRightHandDown)

{

// SendKeys.SendWait("{DOWN}");

PressKey(Keys.Down);

this.Status.Content = "Down";

}

else

{

UpKey(Keys.Down);

}

}

else

{

UpKey(Keys.Up);

UpKey(Keys.Down);

UpKey(Keys.Left);

UpKey(Keys.Right);

}

}

private BitmapSource CreateBitmap(ColorImageFrame frame)

{

var pixelData = new byte[frame.PixelDataLength];

frame.CopyPixelDataTo(pixelData);

GrayscaleData(pixelData);

var stride = frame.Width \* frame.BytesPerPixel;

var bitmap = BitmapSource.Create(frame.Width, frame.Height, 96, 96, PixelFormats.Bgr32, null, pixelData, stride);

return bitmap;

}

private void GrayscaleData(byte[] pixelData)

{

// Mapping depth to color

var mapper = new CoordinateMapper(sensor);

var depthPoints = new DepthImagePoint[640 \* 480];

mapper.MapColorFrameToDepthFrame(ColorImageFormat.RgbResolution640x480Fps30, DepthImageFormat.Resolution640x480Fps30, depthImagePixels, depthPoints);

for (int i = 0; i < depthPoints.Length; i++)

{

var point = depthPoints[i];

if (point.Depth > 600 || KinectSensor.IsKnownPoint(point))

{

var pixelDataIndex = i \* 4;

var maxValue = Math.Max(pixelData[pixelDataIndex], Math.Max(pixelData[pixelDataIndex + 1], pixelData[pixelDataIndex + 2]));

pixelData[pixelDataIndex] = maxValue;

pixelData[pixelDataIndex + 1] = maxValue;

pixelData[pixelDataIndex + 2] = maxValue;

}

}

}

}

}