Realising and Applied Gaming Ecosystem

Research and Innovation Action

Grant agreement no.: 644187

D2.3 – Real-time Emotion Detection Asset - Installation tutorial

RAGE – WP2 – D2.3

Project Number	H2020-ICT-2014-1
Actual Date	February 2018
Document Author/s	Dessislava Vassileva
Version	1.0
Status	Final version

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 644187

About the Real-Time Arousal Detection Using Galvanic Skin Response Asset

Galvanic Skin Response (GSR), also referred to Electro-Dermal Activity (EDA), Skin Conductance Response (SCR), Psycho-Galvanic Reflex (PGR), or Skin Conductance Level (SCL), is related to the activity of the sweat glands, which are regulated by the sympathetic nervous system. When being open and functioning intensively, they emit water solution (sweat) which creates channels of higher conductivity toward the deeper skin layers. EDA represents the electrical conductivity of the skin, which is directly dependent on the activity of the sweat glands, and is often used to index the autonomic arousal. GSR offers a popular and affordable way for detection of player's arousal in adaptive digital games and other affective computing applications. The asset produces real-time features of GSR signal measured from particular player such as: mean tonic activity level, phasic activity represented by mean and maximum amplitude of skin conductance response (all in μS, i.e. micro-siemens), rate of phasic activity (response peaks/sec), SCR rise time, SCR 1/2 recovery time, and slope of tonic activity (in μS/sec). The level of arousal may be useful for emotion detection and for adaptation purposes. The asset receives a filtered raw signal from a simple, low cost biofeedback device allowing sampling rate up to 0.8Khz. Measurements are carried out with two electrodes placed on two adjacent fingers. Recording, filtering and feature extraction might be executed on a computer (server) different than the game machine, in order to speed up all the required processing. The results will be communicated from the server-side to the client component in order to be used for game adaptation.

Document scope

The present document provides installation instructions of Real-Time Arousal Detection Using Galvanic Skin Response asset and, next, presents how to integrate and use the asset in a Unity 3D-based C# game.

Asset installation steps

The asset installation process includes following steps:

1. Download from address: https://github.com/ddessy/RealTimeArousalDetectionUsingGSR/tree/master/Drivers, the zip file named *GSRDrivers.zip* with GSR device's drivers;

2. Unzip the file *GSRDrivers.zip* and install drivers (from folders *SerialCOMDriver* and *drivers_ft232*) (if there are not available at your computer);

3. Download the archive file *RealTimeArousalDetectionUsingGSRBin.rar* from: *https://github.com/ddessy/RealTimeArousalDetectionUsingGSR/blob/master/RealTimeArousalDetectionUsingGSRBin/RealTimeArousalDetectionUsingGSRBin.rar* with *exe* and *dll* files implementing the asset functionality;

4. Unzip the archive file and copy the asset files in a folder at your local file system;

5. Check the configuration file located at *./Resources/RealTimeArousalDetectionAssetSettings.xml* and make changes for some of its settings, where is it needed. The setting value *LogFile* (<LogFile>C:/Users/XXX/logs/log.txt</LogFile>) is mandatory to be changed. It is the path to the log file of the asset and it (the path) has to exist in the local file system;

6. Plug the sensor's cable into the GSR measuring device.

7. Plug the USB cable into the GSR measuring device and connect the other side of the cable to a free USB port of the computer.

8. Put the two sensors on the fingers of the hand of the user. The hand should be clean and dry. In order to avoid any noise and artefacts, the hand should be hold in still static position.

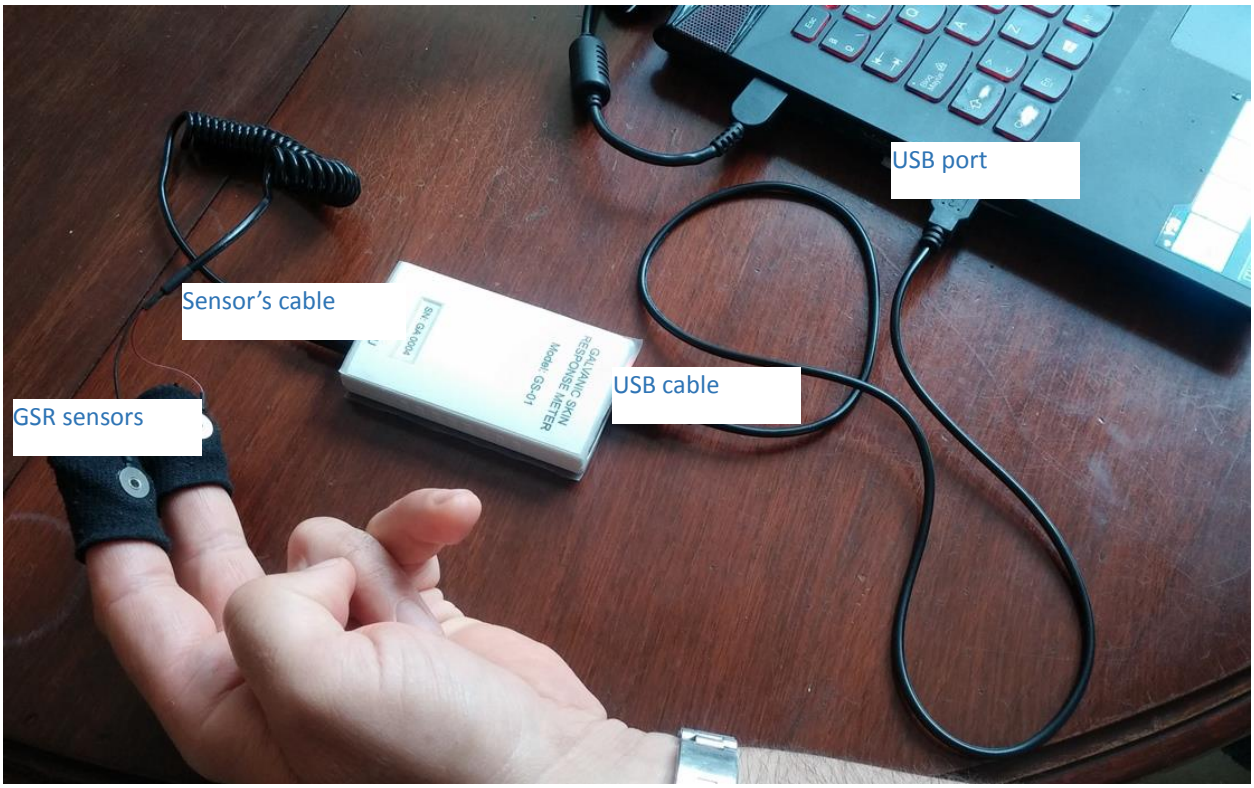

Fig. 1: Connecting the GSR measuring device to the computer

9. If you like to run the application in background mode (with no visualization of the signals), set the value of the *FormMode* property in the *./Resources/realTimeArousalDetectionAssetSettings.xml* configuration file to be equal to `BackgroundMode`. Otherwise, the value should be as given below: `<FormMode>NoBackgroundMode</FormMode>`

10. Start the *DisplayGSRSignal.exe* file by clicking onto it and after then click on the *Start* button in order to visualize the GSR signal. The socket will be started at the port specified in the property *SocketPort* (i.e. <SocketPort>10116</SocketPort>) from the configuration file (*./Resources/realTimeArousalDetectionAssetSettings.xml*), and the device starts measuring. You can do the following:

    1. You can stop and start the socket alternatively by clicking the "Start/Stop socket" button.

    2. In order to start visualization of the GSR signal, press the "Start" button. At any moment you can stop the device by pressing the "Stop" button and, next, to start it again.

    3. You can switch on/off the visualization denoised signal visualization by check/uncheck the checkbox over the "Start" button.

    4. After stopping the device, you can use the mouse wheel in order to zoom in/out each one of the axes, while the mouse cursor is placed on graphic. For resetting the zoom on the axis X or Y, just click on the circle button in the end of the scrollbar of the corresponded axis (fig. 2).

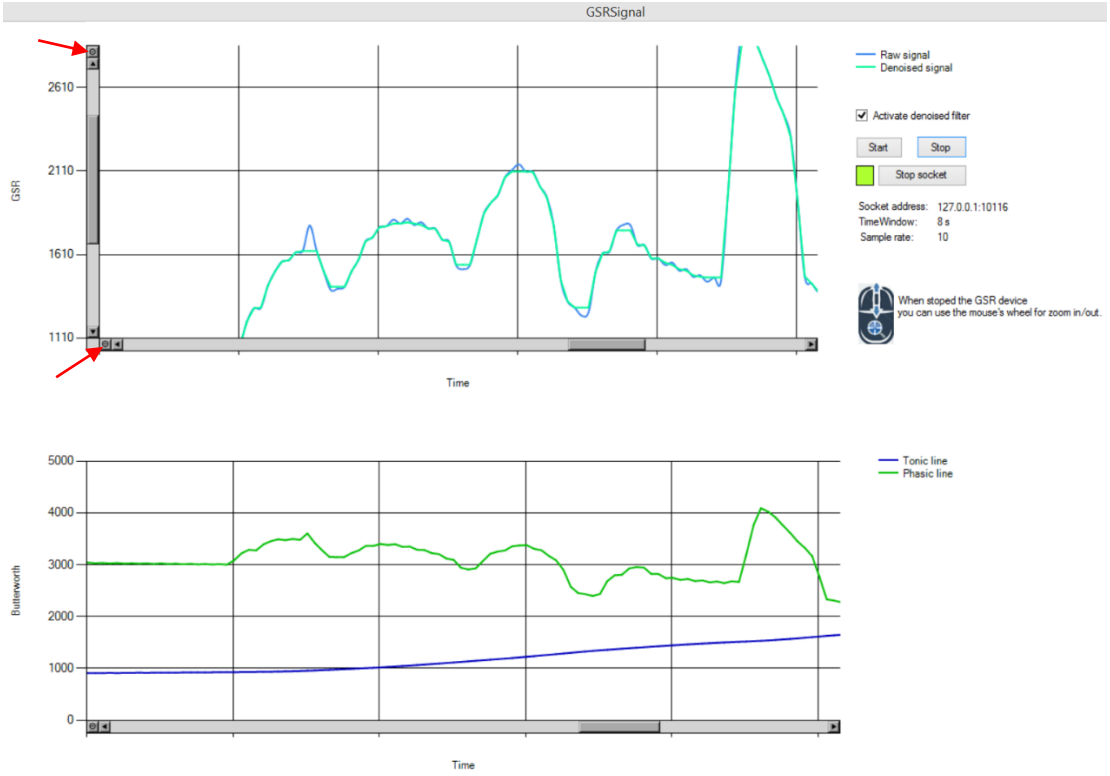

Fig. 2: A screenshot of the visualization application with zoomed signal on X and Y

11. Go to the section “Asset integration steps”.

Application settings

1. Basic application settings

The basic application settings in *./Resources/realTimeArousalDetectionAssetSettings.xml* are as follows:

- SocketPort – the number of port of the socket that will communicate with the asset;
- SocketIPAddress – IP address of the socket that will communicate with the asset;
- COMPort – define the GSR device COM port. If it has value `N.A.`, the asset works with one random chosen COM port (but usually only one COM port is available);
- FormMode – the application run in background mode if it has value `BackgroundMode`;
- LogFile – path to the log file;
- DefaultTimeWindow - default value for the time window;
- SamplerateLabel - sample rate of the GSR device;
- ArousalLevel - number of arousal levels.

2. Advanced application settings

The basic application settings in *./Resources/realTimeArousalDetectionAssetSettings.xml* are as follows:

- MinGSRDeviceSignalValue - the smallest possible value that can be detected by the GSR device;
- MaxGSRDeviceSignalValue - the largest possible value that can be detected by the GSR device;
- CalibrationTimerInterval – in the calibration period it is the time interval during which the asset measures the arousal status of the current user;
- ButterworthPhasicFrequency – frequency used in the Butterworth filter for the phasic signal;
- ButterworthTonicFrequency – frequency used in the Butterworth filter for the tonic signal;

- ApplicationMode – if it has value *TestWithoutDevice* the asset works with data from a file (that is specified in the setting TestData), but not with real data from the GSR device;
- TestData – path to the file that specified the static GSR data when value of the setting ApplicationMode is *TestWithoutDevice*.

Asset integration steps

The asset integration process includes following steps:

1. Start the asset by clicking onto *DisplayGSRSignal.exe* and, if the mode is set to *NoBackgroundMode*, click next on the *Start* button in order to visualize the GSR signal (the GSR measuring device should be attached to the same computer); else, in *BackgroundMode*, the asset will start without visualization window;
2. Check if the asset's socket server is available;
3. Run a socket client on the address (that is stated in the setting *SocketIPAddress*) and port (that is stated in the setting *SocketPort*) specified in the application settings;
4. At the beginning, send to the asset a message with content *"SOCP"* for starting the calibration period;
5. After the calibration period came up to its end, send a message *"EOCP"* for ending calibration period;
6. When you need information about the status of emotional arousal of the current user, send a message to the asset with the content *"GET_EDA"*;
7. At the end of the arousal measurement of the current user, send a message *"EOM"*.

Example of using the asset

The asset can be used in a game following the next steps:

1. Create a socket client class such as the following:

```
using System;
using System.Text;
using System.Net.Sockets;

public class SocketClient
{
    private int portNum;// = 10116;
    private string hostName;// = "127.0.0.1";
    TcpClient tcpClient = new TcpClient();
```

```
    NetworkStream networkStream = null;

    public void StartSocketClient(int portNum, string hostName)
    {
           this.portNum = portNum;
           this.hostName = hostName;
           tcpClient.Connect(hostName, portNum);

           networkStream = tcpClient.GetStream();
           if (networkStream.CanWrite && networkStream.CanRead)
           {
                  //Log("Start socket connection.");
           }
           else if (!networkStream.CanRead)
           {
                  //Log("You can not write data to this stream");
                  networkStream.Close();
                  tcpClient.Close();
           }
           else if (!networkStream.CanWrite)
           {
                  //Log("You can not read data from this stream");
                  networkStream.Close();
                  tcpClient.Close();
           }
    }

    public bool IsSocketConnected()
    {
           return tcpClient.Client.IsBound;
    }

    public object SendMessageToSocketServer(String message)
    {
        string returnData = "";
        if (networkStream != null && networkStream.CanWrite && networkStream.CanRead)
        {
            //Log("Send to the socket message: " + message);
            Byte[] sendBytes = Encoding.ASCII.GetBytes(message);
            networkStream.Write(sendBytes, 0, sendBytes.Length);

            // Reads the NetworkStream into a byte buffer.
            byte[] bytes = new byte[tcpClient.ReceiveBufferSize];
            int BytesRead = networkStream.Read(bytes, 0,
(int)tcpClient.ReceiveBufferSize);

           // Returns the data received from the host to the console.
           returnData = Encoding.ASCII.GetString(bytes, 0, BytesRead);
           //Log("Socket answer: \r\n" + returnData);
        }

        return returnData;
    }

    public void CloseSocketConnection()
    {
           networkStream.Close();
           tcpClient.Close();
```

```
        //Log("Close socket connection.");
    }
}
```

2. Define a socket client in your game:

```
SocketClient client = new SocketClient();
bool endOfCalibrationPeriod = false;
bool hasSocketConnection = false;
```

3. Start the socket with the port and IP address specified in the setting file (i.e. *<SocketPort>10116</SocketPort>; <SocketIPAddress>127.0.0.1</SocketIPAddress>*):

```
try
{
    client = new SocketClient();
    client.StartSocketClient(portNum, hostName);
    hasSocketConnection = true;
}
catch (SocketException se)
{
    hasSocketConnection = false;
    Debug.Log("The socket connection can't be established.");
};
```

4. Send a message to the asset for starting a calibration period:

```
string socp = (string)client.SendMessageToSocketServer("SOCP");
```

5. Send a message to the asset for ending the a calibration period:

```
if (!endOfCalibrationPeriod && hasSocketConnection)
{
    if (calibrationTime <= 0.0f)
    {
        string eocp = (string)client.SendMessageToSocketServer("EOCP");
        endOfCalibrationPeriod = true;
    }
}
```

The calibration period is appropriate to be in the range between 2 - 4 minutes.

6. Use data from the asset when you need them – general arousal level or/and other arousal statistics data (i.e. SCR arousal area, SCR achieved arousal level, etc.):

```
if (hasSocketConnection && endOfCalibrationPeriod)
{
    string currentEDA = (string)client.SendMessageToSocketServer("GET_EDA");
    Debug.Log("currentEDA=" + currentEDA);
    currentEDA = currentEDA.Substring(currentEDA.IndexOf('{'));
    currentEDA = currentEDA.Replace("NaN", "0");

    ArousalStatistics currentArousalStatistics = new ArousalStatistics();
    JsonUtility.FromJsonOverwrite(currentEDA, currentArousalStatistics);
    arousalLevel = currentArousalStatistics.GeneralArousalLevel;
    Debug.Log("GeneralArousalLevel = " + arousalLevel);
}
```

The EDA data are sent from the asset in form of JSON file (an example of such file is given in the next section). In order to use them more convenient you can prepare a class reflected the JSON file structure and deserialize it like an object (i.e. using `JsonUtility.FromJsonOverwrite(JSONString, object)`). An example of such class is given below:

```
public class ArousalStatistics
{
		public double SCRArousalArea;
		public double SCRAchievedArousalLevel;
		public double SCLAchievedArousalLevel;
		public double GeneralArousalLevel;
		public ArousalFeature SCRAmplitude;
		public ArousalFeature SCRRise;
		public ArousalFeature SCRRecoveryTime;
		public TonicStatistics TonicStatistics;
		public double MovingAverage;
		public double LastValue;
		public double LastRawSignalValue;
		public double LastMedianFilterlValue;
		public double HighPassSignalValue;
		public double LowPassSignalValue;

		public ArousalStatistics()
		{
			//super();
		}
}

public class ArousalFeature
{
		public double Minimum;
		public double Maximum;
		public decimal Mean;
		public decimal StdDeviation;
		public string Name;
		public double Count;

		public ArousalFeature()
		{
			//super();
		}

		public ArousalFeature(string name)
		{
			this.Name = name;
		}
}

public class TonicStatistics
{
		public double slope;
		public double meanAmp;
		public double minAmp;
		public double maxAmp;
		public decimal stdDeviation;
		public TonicStatistics ()
```

```
		{
			//super();
		}
}
```

Example of a JSON object returned by the asset


```
{
  "SCRArousalArea": 1433323.455078125,
  "SCRAmplitude": {
    "Minimum": 1573,
    "Maximum": 1731,
    "Mean": 1659.96932515337,
    "StdDeviation": 37.6450036301976,
    "Count": 40.75,
    "Name": "Amplitude"
  },
  "SCRRise": {
    "Minimum": 0.99169921875,
    "Maximum": 6935.13232421875,
    "Mean": 47.4074355116836,
    "StdDeviation": 463.337091469114,
    "Count": 27.75,
    "Name": "Rise time"
  },
  "SCRRecoveryTime": {
    "Minimum": 0,
    "Maximum": 16.0113525390625,
```

```
  "Mean": 7.93012146288128,
  "StdDeviation": 1.51291360666324,
  "Count": 26.125,
  "Name": "Recovery time"
 },
 "SCRAchievedArousalLevel": 52,
 "GeneralArousalLevel": 38,
 "TonicStatistics": {
  "Slope": -0.0053473949851672508,
  "MeanAmp": 1673.5,
  "MinAmp": 1566,
  "MaxAmp": 1707,
  "StdDeviation": 52.8446780669539
 },
 "SCLAchievedArousalLevel": 49,
 "MovingAverage": 1654.0258992805755,
 "LastValue": 2977.817,
 "LastMedianFilterValue": 1628,
 "LastRawSignalValue": 1626,
 "HighPassSignalValue": 2977.817,
 "LowPassSignalValue": 1665.54
}
```