External communication between Unity, Arduino and Processing

About this project

For my master's thesis, my study group and I set out to explore if we could develop a game, which could fit the difficulty to the player, based on how hard they press on a keyboard. In a previous study, we found a statistical correlation between the difficulty and the amount of force the player exerts on a keyboard. For this project I developed some code, which would allow for a custom pressure reader, Unity and Processing to communicate.

The pressure reader

The script referenced in this section can be found in "reader\reader.ino"

The pressure reader waits until there is a connection in the other end of the serial port. The value is between 0-1023. As the data is sent as an array of bytes in the serial port, the data is normalized between 0-255. This means that the data only fills one byte and is easier to read in Unity. (As adding each element in the byte array is no longer needed)
The code is delayed depending on how many readings is desired per second, following this formula. $\frac{1000ms}{desired\ readings\ per\ second}$ (The delay is measured in ms).

The code for this can be seen below.

```
void setup() {
  Serial.begin(9600);
  while(!Serial){
    Serial.println("Waiting for Serial Port to open");
  }

  timeStep = 1000/fsrReadingsPrSec;
}

void loop() {
  fsrReading = analogRead(a_FSRPin);

  fsrReading = map(fsrReading, 0, 1023, 0, 255);

  //Used while hz is outcommented
  SendData(fsrReading);

  delay(timeStep);
}

void SendData(int lr){ /*, int rr*/
  Serial.write(lr);
}
```

Unity code

The script referenced in this section can be found under “ExternalComm.cs”.

The C# script is responsible for receiving the data, convert it from resistance to conductance, send the conductance value to the right place in the game and then send it to Processing for visualisation.

For this to work, the script utilizes the libraries for serial port connections and UDP-connections.

The conversion from resistance to conductance was because the signal received from the sensor is logarithmic (The harder the user presses, the less the sensor reacts), but the conductance is linear. The code for calculating this can be found below.

```
private void ConvertResistanceToConductance() {
    //Calculating resistance
    fsrVoltage = NormalizeValue(currentPressureValue, 0, 255, 1, 5001);
    fsrResistance = 5001 - fsrVoltage;

    fsrResistance *= 10000;
    fsrResistance /= fsrVoltage;

    //Calculating conductance
    fsrConductance = (1000000 / fsrResistance);
}
```

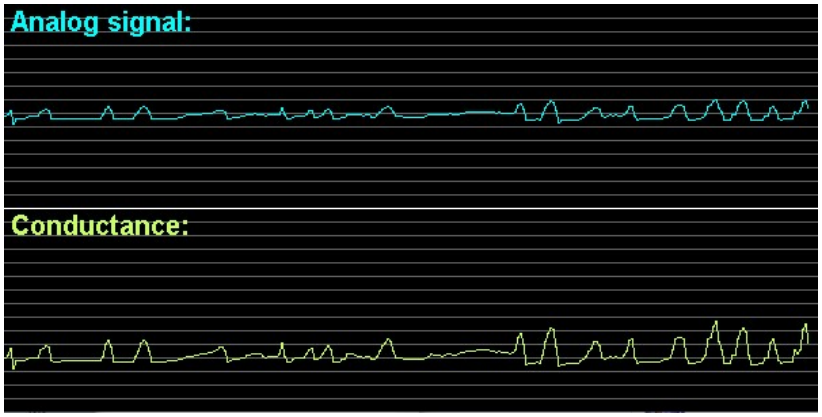
Normally the fsrVoltage value should be mapped between 0-5000 as there is sent 5V through the Arduino, but that would risk a division by 0. Therefore I mapped it to an offset of 1-5001. This means that after mapping the value, saying 5001 – fsrVoltage will still return the actual value between 0-5000, but without the risk of dividing by 0.

Processing code

The script referenced in this section can be found under “PressureVisualisation\ PressureVisualisation.pde”.

As processing was only use for visualisation, corruption in the data didn’t matter. Therefore, it utilized a UDP connection. noloop() was used to avoid the program from running unless processing received data. After receiving the data, the draw function would be run once, to make a line on the graph. The two most recent data points would be kept for both the analogue signal and conductance. A line would be drawn between these points, which would visualise a continuous signal. Whenever the signal reaches the end of the window, the signal is removed, to avoid overlapping.

An illustration of the graph can be seen below.



The code for drawing this can be seen below.

```
void draw(){
  if(positionIndex > width){
    ResetSketch();
  }

  stroke(0,255,255);
  line(  positionIndex-1,
         height/2 - pressureValue[1],
         positionIndex,
         height/2 - pressureValue[0]
         );

  stroke(200,255,100);
  line(  positionIndex-1,
         height/2 - conductanceValue[1] + height/2,
         positionIndex,
         height/2 - conductanceValue[0] + height/2
       );
}
void ResetSketch(){
  background(0);

  stroke(100);
  for(int i = height; i > 0; i-=10){
    line(0,i,width,i);
  }

  fill(0,255,255);
  textSize(16);
  textFont(createFont("Arial bold", 20));
  text("Analog signal:", 5, 20);
  fill(200,255,100);
  text("Conductance:", 5, 20+height/2);

  stroke(255);
  line(0, height/2, width, height/2);
  positionIndex = 0;
}
```