Multidimensional Sensing Techniques

Assignment N°3: Light Sensors

Cédric Léonard, 14/10/2021, student ID: 2100775

Introduction

This third Multidimensional Sensing Techniques assignment aims to get familiarized with light sensors. This assignment will be about simulating a circuit containing an Arduino UNO and 4 photoresistors (Light sensors). We will implement a simple low-level data fusion application across sensors and an algorithm to detect faulty sensor. You can find additional informational on the following GitHub repository.

What you did? How you did it?

Set up my work environment

The assignment took place on www.tincercad.com. After an account creation I created a new circuit design containing a breadboard, an Arduino UNO and 4 photoresistors (LDRs) and their corresponding resistors. Each LDR is connected to an analog input pin of the Arduino using some Voltage Divider network. Figure N°1 shows the schematized circuit, red wires are connected to a power source (5V), black wires to the ground (0V) and yellow wires to the analog input pins.

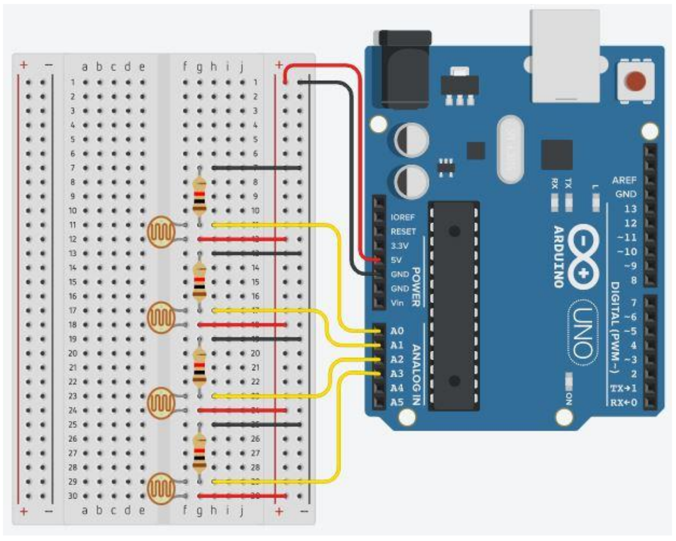


Figure N°1: Screenshot of the studied circuit schemated on **tinkercad**

Arduino program

After connecting the circuit, I created an Arduino script containing C++ code. The starting point of this code was found on Moodle. The main point of this code is the call of the `analogRead()` function to read a value from an analog input pin:

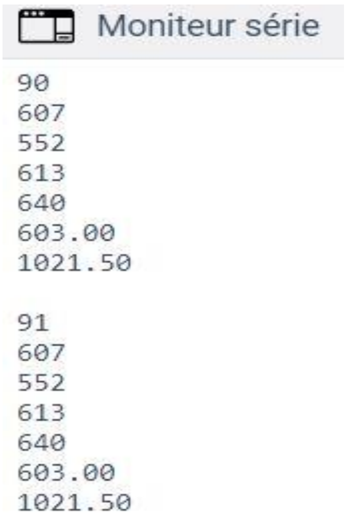
```
// Value = analogRead(PinID);
LDRValues[0][index] = analogRead(LDR1Pin);
```

In our case, a way to complete the assignment was to create a sliding window buffer. This buffer, called `LDRValues`, is in fact a matrix with 4 lines, one for each sensor, and `LDRBufferWidth` columns (which corresponds to how many values we store before overwriting the first ones).

Task N°5 asked us to compute the global average and variance of the current 4 sensors values. This is simply done in the main loop just after reading and storing each value in `LDRValues`:

```
// ---------- Compute statistics ----------
  float avg = 0; float var = 0;
  // --- Average Computation ---
  for(int i = 0; i < nbSensors; i++)
    avg += LDRValues[i][index];
  avg /= nbSensors;
  // --- Variance Computation ---
  for(int i = 0; i < nbSensors; i++)
    var += (LDRValues[i][index] - avg)*(LDRValues[i][index] - avg);
  var /= nbSensors;
```

After the code above, some printing for debug purposes is done and produces Figure N°2.


Moniteur série

```
90
607
552
613
640
603.00
1021.50

91
607
552
613
640
603.00
1021.50
```

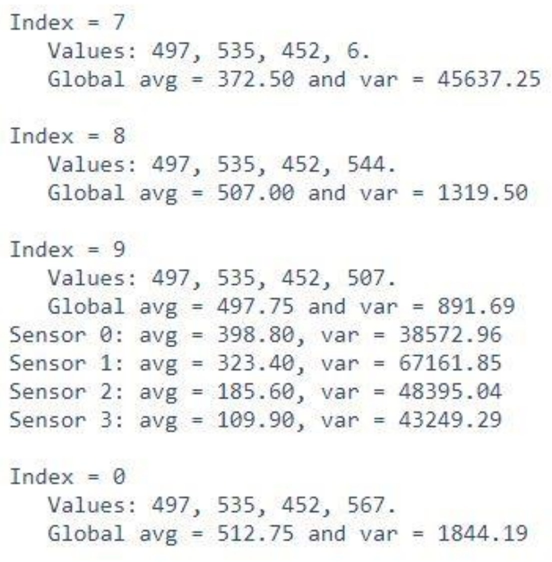
Figure N°2: Screenshot of the serial monitor showing each sensor value and global average and variance

Variables range

In the example shown in Figure N°2 the amount of light the photoresistors get has been set on purpose to the slightly same amount i.e. ~ 600. Therefore, the global variance is quite low and doesn't excess 8191. This was important when I first used the `int` type for my variables. Indeed, as explained in the *Assignment3_Instructions.pdf* each variable type corresponds to a different number of bytes and so, a different range. I was expecting my `int` variables to overflow (and to show negative number) outside the range of $[-32{,}768;\ 32{,}767]$ but the range seemed to be $[-8192;\ 8191]$ on my computer which is weird but represents the same problem. A way of fixing this is to expand the number of bytes allowed for the variable storage and therefore increase the variable possible range. This can be done with `long` variables or in my case with `float` which are also stored on 4-bytes and fit better than `int` for statistics computations.

Individual statistics

The next step was about computing individual averages and variances for each sensor. This is done when the sliding window buffer is filled i.e. that `index` equals `LDRBufferWidth`. The calculations used are simple and we then print the results for each sensor as shown in Figure N°3 (with $LDRBufferWidth = 10$).



```
Index = 7
   Values: 497, 535, 452, 6.
   Global avg = 372.50 and var = 45637.25

Index = 8
   Values: 497, 535, 452, 544.
   Global avg = 507.00 and var = 1319.50

Index = 9
   Values: 497, 535, 452, 507.
   Global avg = 497.75 and var = 891.69
Sensor 0: avg = 398.80, var = 38572.96
Sensor 1: avg = 323.40, var = 67161.85
Sensor 2: avg = 185.60, var = 48395.04
Sensor 3: avg = 109.90, var = 43249.29

Index = 0
   Values: 497, 535, 452, 567.
   Global avg = 512.75 and var = 1844.19
```

Figure N°3: Screenshot of the serial monitor showing individual averages and variances computations.

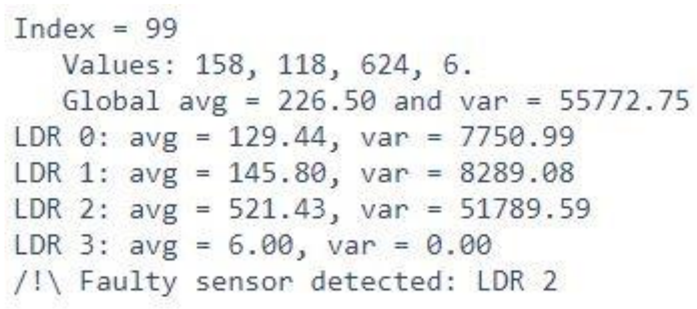
Defect detection algorithm

With these new individual statistics, we can now set up an algorithm to detect a possible faulty sensor. Defect detection is a really hard topic and can quickly become very complex. Hence, we will keep with a very simple solution.

We will consider a LDR as faulty if:

- the global variance is huge (> 40.000);
- its individual mean over the last `LDRBufferWidth` samples is far from the global mean (absolute distance > 150).

Moreover, as this algorithm may highlight several sensors (because we only have 4 different sensors and we are using the arithmetic mean formula for the average) we will then check every absolute distance between individual mean and the global mean and select the biggest. Such an algorithm produces the following result, Figure N°3:



```
Index = 99
   Values: 158, 118, 624, 6.
   Global avg = 226.50 and var = 55772.75
LDR 0: avg = 129.44, var = 7750.99
LDR 1: avg = 145.80, var = 8289.08
LDR 2: avg = 521.43, var = 51789.59
LDR 3: avg = 6.00, var = 0.00
/!\ Faulty sensor detected: LDR 2
```

Figure N°3: Screenshot of the serial monitor, faulty LDR detection

On Figure N°3 we can observe the current output of each sensor, 3 are low (< 200) and LDR2 outputs a high value (> 500). With the algorithm explained previously (in the `detectFaultySensor` function) the sensor LDR 2 is detected as faulty.

Minimal and maximal voltage and resistance in the circuit

For the last task we were asked to calculate minimal and maximal voltage applied to the analog inputs and same for the resistance value of the photoresistor.

When simulating, we measured that the minimal value returned by the `analogRead()` function is $A_{min} = 6$ and its maximal value: $A_{max} = 679$. We also know from the assignment instructions that behind each analog input of the Arduino UNO there is an ADC converting an input voltage range (called V_{out}), 0 to $5V$, to a digital value 0 to 1023.

Therefore, $V_{min} = \frac{5V*A_{min}}{1023} = 0{,}029V$ and $V_{max} = \frac{5V*A_{max}}{1023} = 3.32V$.

Now we want to know the minimal and maximal resistance of the photoresistor (LDR). This can easily be computed as our circuit can be assimilated as a Voltage Divider Network. Hence, we can deduce the following formula:

$$V_{out} = \frac{R_1}{R_1 + R_{LDR}} V_{in} \quad \Leftrightarrow \quad R_{LDR} = \frac{V_{in} R_1}{V_{out}} - R_1$$

Then, by substituting R_1 with $1k\Omega$, V_{in} with $5V$ and V_{out} with $V_{min} = 0.029V$ and $V_{max} = 3.32V$ we get:

- $R_{LDRmax} = 169k$;
- $R_{LDRmin} = 506$.

How your solution can be improved/generalized?

The defect detection algorithm can easily be improved with for examples some dynamic threshold (instead of a global variance > 40.000 or a distance between individual and global mean > 150). The program is also made that we run the defect detection algorithm once the circular buffer is full, i.e. every `LDRBufferWidth` samples. This is convenient, but can be improved with, for example a dynamic call of the function when huge or weird value are detected.

What did you learn? Did anything surprise you? Did you find anything challenging? Why?

This assignment was not particularly challenging, it was only about implementing features from the Assignment 2 but in C++. I thought it would be more complicated to create a circular buffer in C++ than in Python, but it was not the case.

Did you find anything satisfying? Why?

I discovered www.tincercad.com and its feature for creating circuit, including Arduino components and I found this very convenient and satisfying. However, the interface to write the C++ code is really poor and does not facilitate the job, same thing for the compiler.