LOGBOOK

DROP COUNTER

This logbook describes the development of the Drop Counter. Which parts are used, how the Drop Counter is assembled and which elements are used in the electrical circuits.

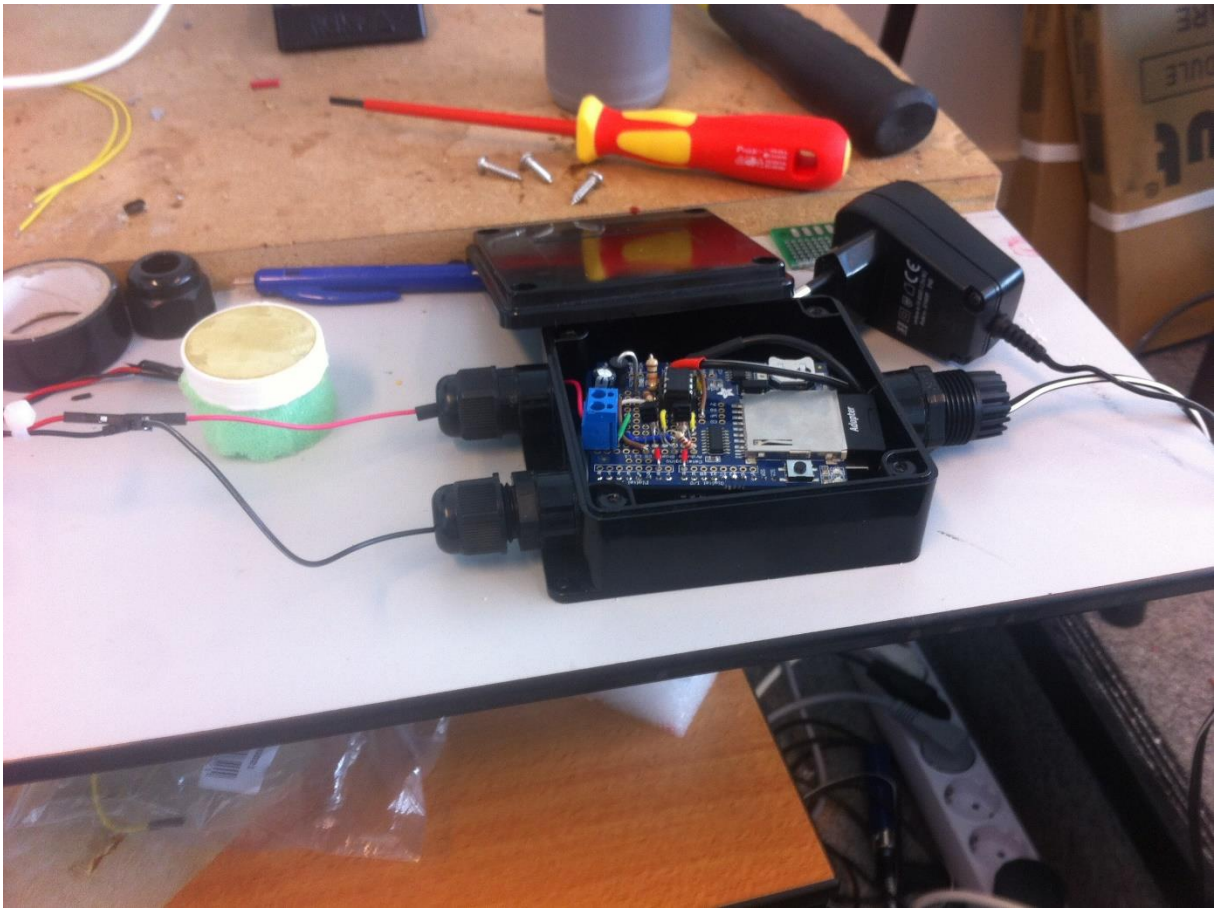

Figure 1: Developing of the Drop Counter

INTRODUCTION

This logbook is used to describe the development of the Drop Counter. The Drop Counter is a city science project which is aimed to provide a simple rain gauge based on the design of an acoustic disdrometer. As the name suggests, the Drop Counter counts the amount of drops per time unit and the intervals between the drops. Apart from that, the Drop Counter is required to be cheap and easy to produce by hobby meteorologists. The principle and the aims of the Drop Counter can also be found at a Github repository[1].

First the assembly of the Drop Counter and the parts which are used are described. Secondly some recommendations are made regarding the installation of the Drop Counter outdoors. Next the effects of different electrical circuits on the performance of the Drop Counter are observed. Finally a small experiment is performed to assess the minimum drop sizes which can be detected with the device.

ASSEMBLY

The drops in the Drop Counter are registered by making use of a piezo element as a sensor. The sensor is fixed on a 3D-printed holder with superglue (Figure 2A). An important note here is to only glue the piezo element around the edges so that the sensor can freely vibrate in the centre and no amplification is required. To damp noisy signals which are not drops, the 3D printed holder is glued on top of a kitchen sponge. It is important to guide the wires of the sensors through the hole in the bottom of the 3D printed holder and to make a hole in the sponge to guide the wires through. The metallic parts of the piezo element should be protected from corrosion. This is realized by applying a film of nail polish or car spray on top of the sensor. Finally a small gauze bandage can be fixed over the sensor so the top of the sensor is hydrophilic. This is applied in order to prevent the damping of drops which fall on a puddle of water formed on the sensor (Figure 2B).

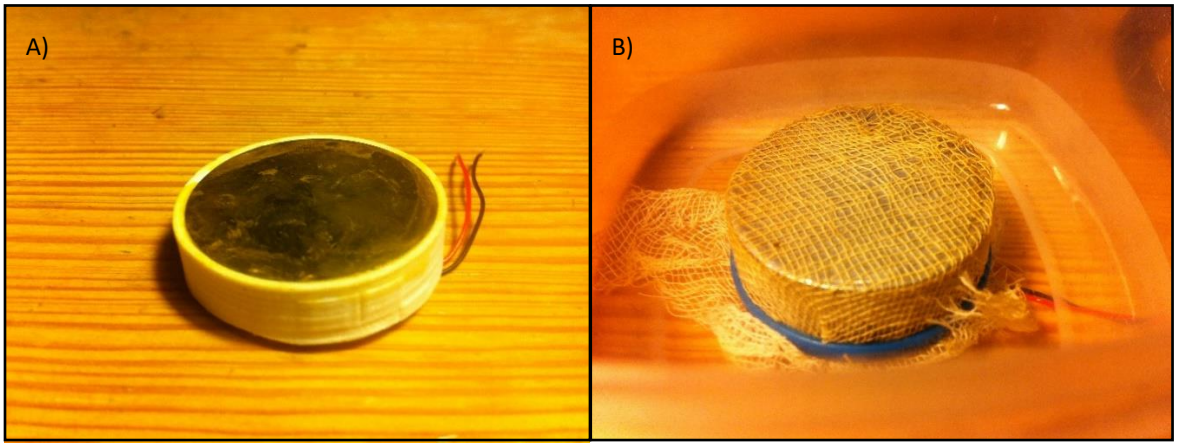

Figure 2: A) Piezo element glued on top of the 3D printed holder. B) piezo element covered with car spray to protect the metal from corrosion and gauze bandage to make the sensor hydrophilic.

Now the signals from the sensor are processed by a data logger. A very well documented and widely used part of equipment are the Arduino boards. For this sensor an Arduino Uno is used in combination with an Adafruit assembled datalogging shield, schematically depicted in Figure 3. There are some elements attached to the datalogging shield in order to process the signals correctly. The different schemes and their effects on the logging of the signals are explained in chapters Scheme 1 to Scheme 4.

[1] https://github.com/nvandegiesen/Intervalometer/wiki/Intervalometer

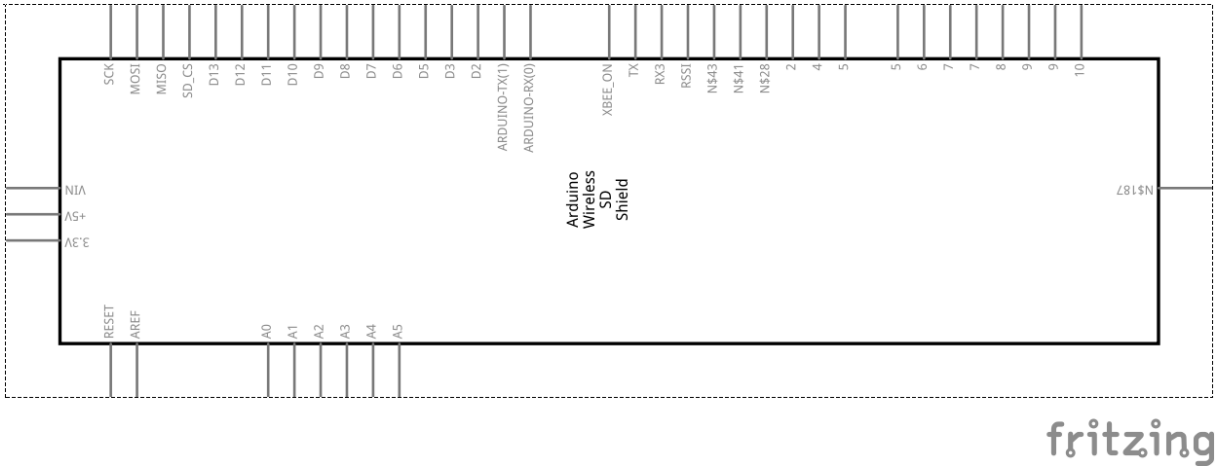

Figure 3: Arduino SD shield

In order to keep the electronics dry the Arduino board is stored in a rubber-sealed box with three inputs. All three inputs are sealed with a rubber ring which prevents water to flow into the box by flowing along the wires (see Figure 1). If the wires are too thin for the seal to tightly bind them, the wires should be thickened with wire tape. One of the inputs is used for the cables from the sensor, note that if the cables are too short they can be extended by soldering additional wire. But it is important to keep the wiring as short as possible to prevent electromagnatic interferention (EMF noise). On top of that; the two wires should be twisted around each other to prevent the EMF noise.

When the sensor and the datalogger are assembled (for the right configuration of the logger shield, see chapters Scheme 1 to Scheme 4) all that rests is to provide the Arduino shield with power and place the sensor outside. The power supply can be realized by using a 12V adapter and by fixing the wires to the Arduino board. Further reading for powering Arduino boards is suggested here: (https://playground.arduino.cc/Learning/WhatAdapter). For the placement in the field the choice has been made to fix the sensor, including the holder and the damping-sponge, in a PVC pipe in order to fix the sensor to the rubber-sealed box. The later can be realized by using cloth tape or by pinning the PVC pipe into a sandy soil with a skewer.

Table 1: Bill of materials

Item	Price	Number	Remarks	Subtotal
Arduino UNO	$ 23.38	1		$ 23.38
Data cable	$ 2.07	1		$ 2.07
Adafruit Data Logging Shield	$ 13.95	1		$ 13.95
Screw terminals	$ 0.57	1	(price at ten pieces)	$ 0.57
Jumper wires	€ 18.36	0		$ 21.70
Pull down resistors 2200 Ohm	€ 42.16	0	kit for all resistors, 100 each	$ 49.83
Pull down resistor 1 10kOhm				$ -
Pull down resistor 2 1MOhm				$ -
Transistor 2N5551	€ 0.15	5		$ 0.86
Zener diodes 3V	$ 0.14	1		$ 0.14
Mosfet BS170	$ 0.48	2		$ 0.96
			Total	$ 113.46
		Fixed costs wires and resistors		$ 73.60
		Price per meter		$ 39.86

CONCEPTUAL LAYOUT DROP COUNTER

Figure 4: Conceptual layout Drop Counter

INSTALLING DROP COUNTER OUTDOORS

During a period of measurements it appeared that the Drop Counter failed to register data due to some errors made in the installation of the device. This chapter will describe some practical aspects of installing the Drop Counter outdoors.

WATERTIGHT

First of all it is important to make the logger of the Drop Counter watertight. This is realized by putting the logger inside the logger box. However, in practice it appeared that the logger box can still be flooded. Hence it is important to make sure that the wires going into the logger box, are sloped downwards from the input (like in Figure 5). I.e. when moisture is attached to the wiring, it can only move downwards and not into the input.

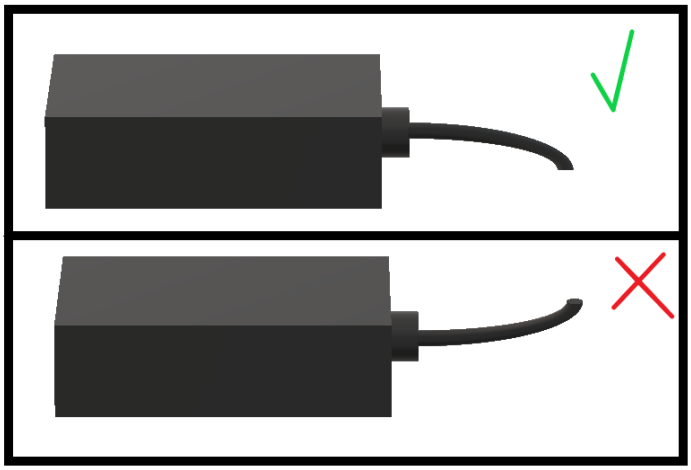

Figure 5: Example of how to install the wiring at the logger box.

WIND PROOF

During some measurements, a lot of the data was lost due to stormy weather which damaged the Drop Counter. The first damage occurred when the sponge with the piezo sensor blew out of the PVC pipe. Therefore it is important to fix the sponge firmly in the PVC pipe with for example superglue. In addition, make sure that the PVC pipe and logger box are attached firmly to the ground or a solid surface (e.g. a rooftop or shed). This can be done by using pins to attach the logger box and PVC pipe in the ground or screws to attach it to a solid surface. Apart from that, when an alternative power source is used like a combination with a battery and a solar panel, the solar panels also have to be firmly attached.

TEMPERATURE PROOF

Finally it is important that all the equipment which is used in the Drop Counter and in the logger box can still function in extreme weather conditions. This is very site specific, but as an example consider that the equipment should still be able to function at -40°C in the polar regions and at +80°C in the tropical regions. Note that the latter temperature is rather high, but can still be reached inside the logger box when the box is exposed to high intensities of solar radiation for a long time. For example some batteries will have trouble operating at these temperatures. Therefore it is advised to carefully pick a type of insulation and colour (which influences the albedo of the logger box) when the Drop Counter is installed in regions with these temperature ranges.

SCHEME 1

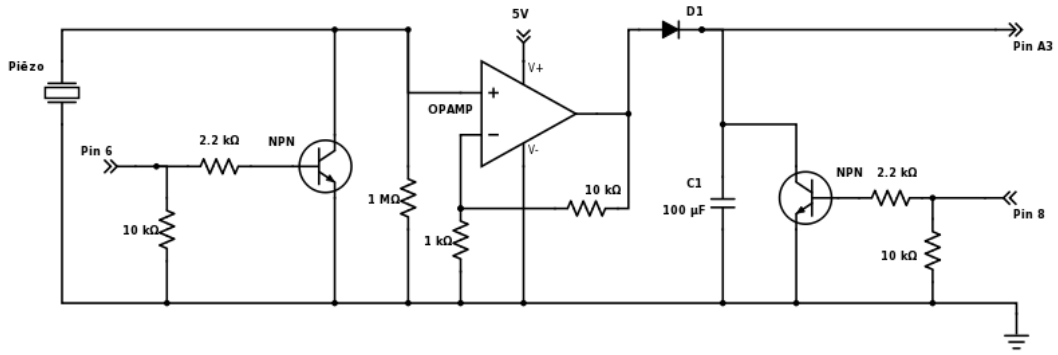

Figure 6: Drop Counter Concept. Including OpAmp, diode and capacitor.

SOFTWARE DROP COUNTER V3

HARDWARE

Arduino: Arduino UNO + Adafruit Assembled Data Logging shield for Arduino
MOSFET: Mosfet BS170
OpAmp: OpAmp LMC6482
Diode: Diode Schottky 30V 2A Through Hole
Capacitor: Elektrolytical capacitor Radial wired 5 mm 100 µF 63 V 20 % (Ø x l) 10 mm x 12.5 mm

OBSERVATIONS IN LINE WITH AIMS OF THE DROP COUNTER

- If drop hits the piezo, the signal is processed to SD card.
- Interval between two drops can by down to 50 ms, so the chance of missing drops is very small.

NOTES

If there is no pull-down resistance between the non-invertible and invertible output of the piezo, the analog pin receives a constant noisy signal.

Instead of the transistors, it is also possible to use MOSFETs. However, MOSFETs are very sensitive! Soldering MOSFETs on the SD shield caused the MOSFET to fail and consequently caused a finite resistance (several kΩ) between the gate and source. Wearing rubber gloves, earthing the solder pistol or your body didn't prevent the MOSFETs from failing. That is why it is assumed to be easier to use transistors which are harder to kill. The MOSFETs used were of the kind 2N5551 B331.

PROBLEMS

- If you wait for a while there is noise arriving at the analog pin, causing random values to be processed to the SD card.

ANALYSIS AND POSSIBLE SOLUTION

Reasons are yet unclear. So the circuit is stripped down to a simpler version of the Drop Counter without the capacitor, depicted in Figure 7.

SCHEME 2

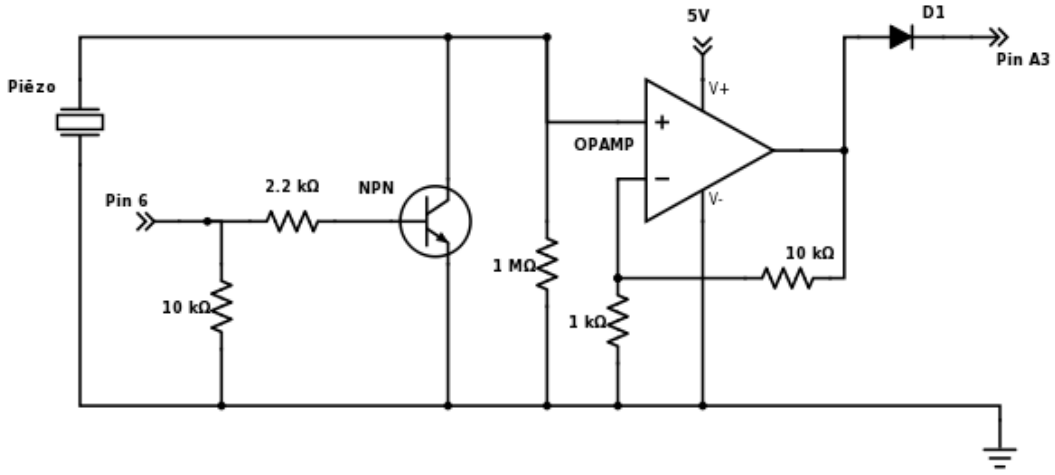

Figure 7: Scheme 2, with OpAmp and diode, no condensator.

SOFTWARE DROP COUNTER V3

HARDWARE

Arduino: Arduino UNO + Adafruit Assembled Data Logging shield for Arduino
Transistor: 2N5551 B331
OpAmp: OpAmp LMC6482
Diode: Diode Schottky 30V 2A Through Hole

OBSERVATIONS IN LINE WITH AIMS OF THE DROP COUNTER

- If drop hits the piezo, signal is processed to SD card.
- Interval between two drops can by down to 50 ms, so the chance of missing drops is very small.

NOTES

Including or excluding the diode in the scheme leads to the same problems. Apart from that, different resistances have been used in order to vary the amplifying factor. (10kΩ : 1kΩ and 10kΩ : 2.2kΩ) All leading to the same results.

PROBLEMS

- If you wait for a while there is noise arriving at the analog pin, causing random values to be processed to the SD card. This can be observed by reading the serial prints (Figure 8).
- If you hit the piezo element the writing of intervals stops
- If you wait for a while the random writing of intervals starts again.
- Without sensor the ‘floating pin’ effect occurs. So the serial readings from the analog sensor shows random values.

ANALYSIS AND POSSIBLE SOLUTION

Maybe the noise is caused by the fact that very small signals caused by wind or Electro-Magnetic Field (EMF) noise are amplified and processed to the analog pin. Therefore the analog pin is showing a constant incoming signal observed in the example serial output in Figure 8. A possible solution might be to exclude the OpAmp in order to limit the noise from low amplitude signals. Therefore the scheme is stripped to the simple knock-sensor example from Arduino (https://www.arduino.cc/en/Tutorial/Knock) in Scheme 3.

Figure 8: Serial output example; output value(0 - 1024, which is analogue to 0 - 5V) printed on first line, interval (ms) printed on second line.

SCHEME 3

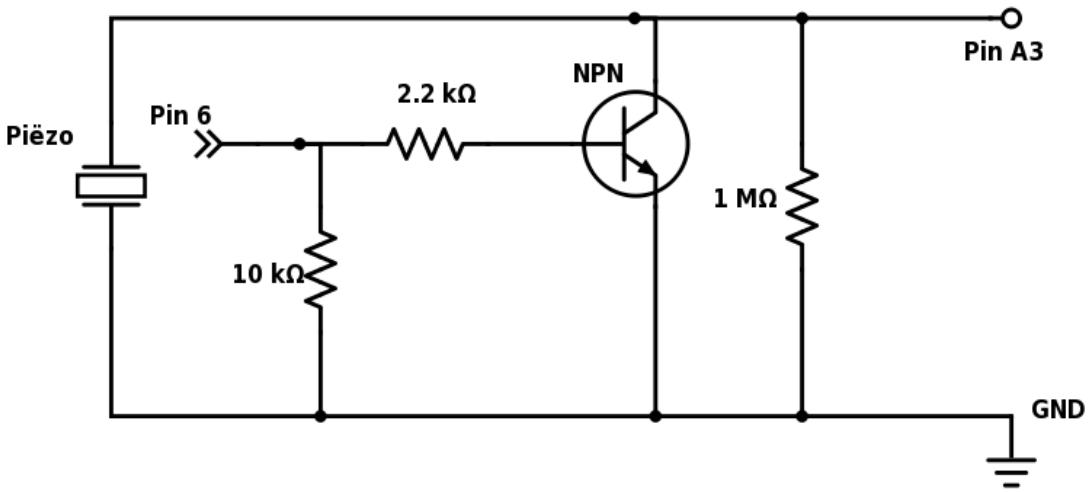

Figure 9: Scheme 3; basic knock sensor example with extra pin to shorten piezo from shaking after incoming signal.

SOFTWARE DROP COUNTER V3

HARDWARE

Arduino: Arduino UNO + Adafruit Assembled Data Logging shield for Arduino
Transistor: 2N5551 B331

OBSERVATIONS IN LINE WITH AIMS OF THE DROP COUNTER

- If drop hits the piezo, the time interval between the signal and the previous signal is processed to SD card.
- Interval between two drops can by down to 50 ms, so the chance of missing drops is very small.

NOTES

A threshold is implied in the software in order to limit the (noisy) low amplitude signals. Now that the OpAmp is excluded from the circuit, the small drops may be too low in amplitude to register. Hence it is tried to imply the internal reference of 1.1V as a reference for the analog pin, so that the signal is roughly 5 times amplified compared to the normal reference of 5V. However, when the threshold is lowered to approximately DC 10, AC 50mV with the normal reference of 5V, noisy signals start to arrive at the analog pin. Therefore the amplitude caused by the impact of small drops (lower than 0.8mm in diameter) is smaller than the noise registered at the analog pin. Hence the signals from the piezo do not have to be amplified in order to be registered by the analog pin.

For the setup of the Drop Counter a plastic logger box with three inputs is used to store the datalogger and to keep it dry in rainy conditions(see Figure 1). In the previous schemes the two small inputs are used for respectively the invertible and non-invertible outputs from the piezo. However, this limits the possibility to twist the wires and consequently seems to trigger the noise picked up from EMF. Twisting the wires up to the screwing station and using only one input from the logger box seems to limit the noise picked up by the wiring.

PROBLEMS

Although the noise problem mentioned in the Including or excluding the diode in the scheme leads to the same problems. Apart from that, different resistances have been used in order to vary the amplifying factor. (10kΩ : 1kΩ and 10kΩ : 2.2kΩ) All leading to the same results.

PROBLEMS FROM SCHEME 1

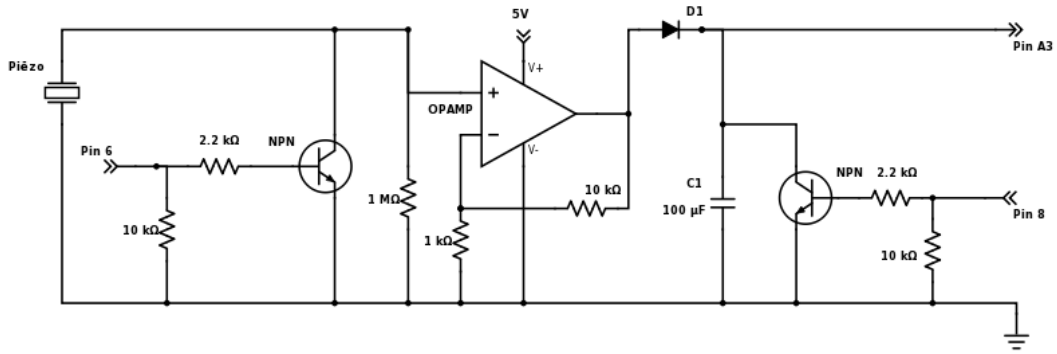

Figure 6: Drop Counter Concept. Including OpAmp, diode and capacitor.

SOFTWARE DROP COUNTER V3

HARDWARE

Arduino: Arduino UNO + Adafruit Assembled Data Logging shield for Arduino
MOSFET: Mosfet BS170
OpAmp: OpAmp LMC6482
Diode: Diode Schottky 30V 2A Through Hole
Capacitor: Elektrolytical capacitor Radial wired 5 mm 100 µF 63 V 20 % (Ø x l) 10 mm x 12.5 mm

OBSERVATIONS IN LINE WITH AIMS OF THE DROP COUNTER

- If drop hits the piezo, the signal is processed to SD card.
- Interval between two drops can by down to 50 ms, so the chance of missing drops is very small.

NOTES

If there is no pull-down resistance between the non-invertible and invertible output of the piezo, the analog pin receives a constant noisy signal.

Instead of the transistors, it is also possible to use MOSFETs. However, MOSFETs are very sensitive! Soldering MOSFETs on the SD shield caused the MOSFET to fail and consequently caused a finite resistance (several kΩ) between the gate and source. Wearing rubber gloves, earthing the solder pistol or your body didn't prevent the MOSFETs from failing. That is why it is assumed to be easier to use transistors which are harder to kill. The MOSFETs used were of the kind 2N5551 B331.

PROBLEMS

- If you wait for a while there is noise arriving at the analog pin, causing random values to be processed to the SD card.

ANALYSIS AND POSSIBLE SOLUTION

Reasons are yet unclear. So the circuit is stripped down to a simpler version of the Drop Counter without the capacitor, depicted in Figure 7.

- Scheme 2 seems to be reduced, there is still noise arriving at the analog pin.
- Sometimes it seems that a drop is missed when it hits the sensor. Maybe because the baud rate is too slow in order to pick the high amplitude signal up.
- Sometimes it takes a long time (24h) after there is a sudden income of noise at the analog pin. The noise is manifested as a constant signal at the analog pin. The cause is hard to find.
- If you pull and turn the wire from the piezo to the datalogger the noise seems to be triggered. This supports the idea that the noise is caused by EMF noise.

ANALYSIS AND POSSIBLE SOLUTION

Since the noise is still available and sometimes starting without a clear reason, filtering the signals entering the analog pin by implying an amplitude threshold only does not seem to do the job. This gives rise to the idea to imply a band frequency filter. So signals with a frequency which is too low or too high to be caused by the impact of a drop on the piezo are not registered at the analog pin. This can be done by implying a hardware filter, but this will complicate the circuit and does not comply to the aim of producing a simple data logger. Therefore it is better to imply the band filter in the software. So the band frequency is implied in the Software Drop Counter V4 package.

SOFTWARE DROP COUNTER V4

OBSERVATIONS IN LINE WITH AIMS OF THE DROP COUNTER

- If drop hits the piezo, the time interval between the signal and the previous signal is processed to SD card.

NOTES

A band filter is implied in the software which excludes the frequencies higher than 2kHz and lower than 10Hz. The frequencies are empirically determined by knocking on the sensor with varying force.

PROBLEMS

- Sometimes it seems that a drop is missed when it hits the sensor. This can either be due to the detection threshold which is too high or the baud rate which is too slow to register two drops which hit the sensor in a very short time range (shorter than the limits of the baud rate).

ANALYSIS AND POSSIBLE SOLUTION

The problem of missing drops by the slow baud rate may be solved by including a capacitor in the circuit to ‘remember’ the signal of the drop. This is tried in the next scheme. However, lowering the detection threshold already seems to solve the problem.

SCHEME 4

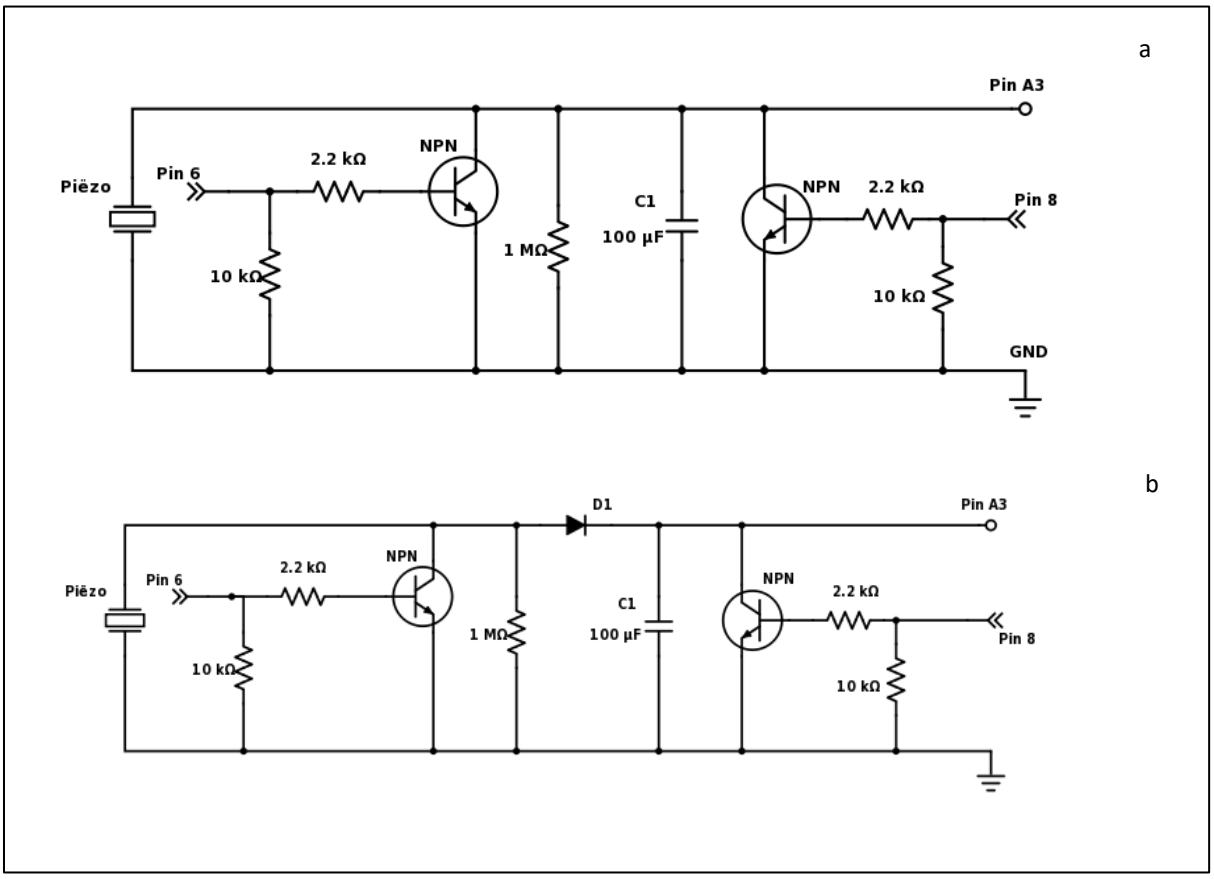

Figure 10: Scheme 4; basic knock sensor example with extra pin to shorten piezo from shaking after incoming signal and a capacitor + shortcut to 'remember' the signal. a) excluding diode, b) including diode.

SOFTWARE DROP COUNTER V4

OBSERVATIONS IN LINE WITH AIMS OF THE DROP COUNTER

- No readings at analog pin

NOTES

Capacitor used: 100 μF, 25V. Maybe the capacitor discharges through the pull-down resistor before the signal reaches the analog pin. However, placing a diode between the pull down resistor and the capacitor to prevent the capacitor to discharge this way (Figure 10b) does not fix the problem.

PROBLEMS

- The analog pin does not register a signal anymore. The reason is unclear.

ANALYSIS AND POSSIBLE SOLUTION

Maybe a different capacitor can be used in order to solve this problem. However, since the reason for the problem is unclear, so is the solution. Hence scheme 3 is considered a better option.

MINIMUM RAINDROP SIZE

EXPERIMENT DESCRIPTION

Now that the Drop Counter is yielding reasonable results it is of import to know what the minimum raindrop size is which can be registered with this device. This is initially tested by dropping artificial drops of tap-water on the sensor from a height of 1m. The drops are produced by pushing the water through an electro valve which can be opened and closed with a temporal accuracy of 1ms. Hence the drop sizes can be varied by applying different amounts of water to the valve, regulated by the open time of the valve. At the end of the electro valve a needle is fixed to help produce even smaller drops. See Figure 11 for a visualization of the experimental setup. The drop size is determined by weighing the drops and assuming they are of perfect cylindrical shape. In order to gain accuracy the average weight of 5 identical drops is taken as a standard. The energy of the drops is calculated by making use of a drag force working on a perfect sphere falling through air. A simple calculating tool can be found at: http://hyperphysics.phy-astr.gsu.edu/hbase/Mechanics/fallq.html.

Note that the electro valve is limited to produce drops down to 1.93mm in diameter, whereas raindrops can range down to lower than 0.5mm in diameter. Since there is a limit to the drop sizes which can be produced with the electro valve, the fall height is decreased in small steps in order to get an idea of the minimum energy which can be registered with the Drop Counter. The minimum energy can subsequently be linked to a minimum drop size.

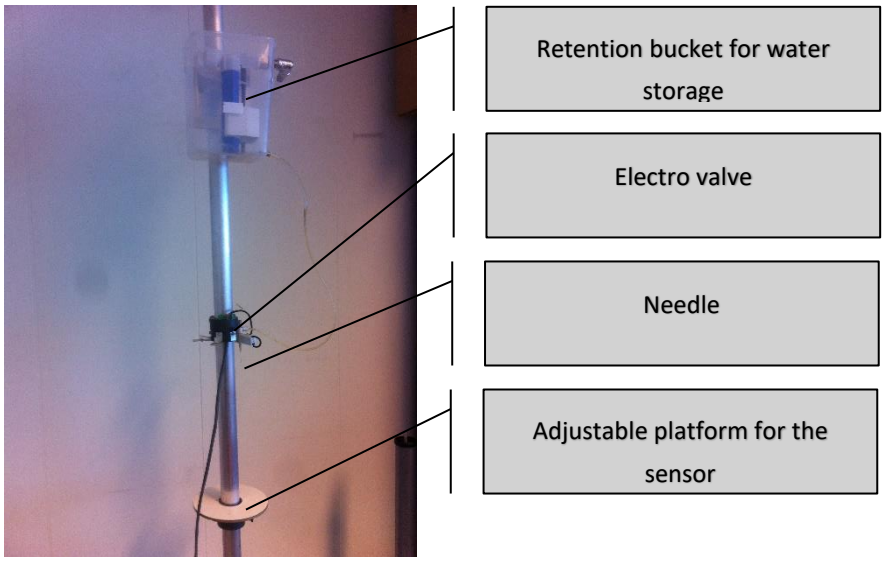

Figure 11: Experimental setup to determine the minimum detectable drop size.

RESULTS

In Table 2 the results are shown for a test with the smallest drop which can be produced with the electro valve. With decreasing fall heights it appears that the drops down to 0.95mm in diameter can still be detected with the current setup of the Drop Counter. Note however that the observations are not very stable.

NOTES DURING OBSERVATIONS

- When the sensor is fully covered by a puddle of water and drops are falling in the centre of the sensor, the readings are not decreased. Apparently the water is elastic enough to transfer the signal to the piezo element.
- This is not the case when the puddle is moving (wiggling); then the drops are not transferred to the sensor.

- When the drops fall next to the puddle, the signal is not transferred for all drops. In addition, if the puddle is very small and the drop falls exactly on this puddle (two to ten drops of 0.0038g), the signal is also damped.
- Especially when the drops are falling on the edge of the sensor, the readings are not very stable. Some drops are registered and some are not, with no clear differences.
- The cork ring is not a very well isolator when soaked wet; signals of the electro valve are read when the cork ring stands on the ring. The kitchen sponge seems to do the job better.
- When the drops of 1.93mm diameter fall from over a meter high the readings are more stable.
- When the sensitivity is optimal, a strong blow of breath can trigger the piezo as well. This gives rise to the question whether strong winds can trigger the piezo as well.

Constants	Number	Unit			
Weigth weighing scale	2.431	g			
No. Drops	50	#			
Density water	1000000	g/m^3			
Terminal velocity at	1	m			
g	9.81	m/s^2			
A0	943				
a	1.77				
n	1.147				
Needle 3 (white)					
Valve_time = 200 ms	Weigth (g) scale+water	Weigth (g) water	Weight (g) per drop	Volume (m^3) of drop	Drop diameter
Weight_1	2.633	0.202	0.00404	4.04E-09	1.976033248
Weight_2	2.618	0.187	0.00374	3.74E-09	1.925858287
Weight_3	2.626	0.195	0.0039	3.9E-09	1.952938925
Weight_4	2.629	0.198	0.00396	3.96E-09	1.962903071
Weight_5	2.632	0.201	0.00402	4.02E-09	1.972767071
	2.6276	0.1966	0.003932	3.932E-09	1.95810012
		0.1966	0.003932	3.932E-09	1.958265753
from http://hyperphysics.phy-astr.gsu.edu/hbase/Mechanics/fallq.html					
Fall Heigths (m)	V_impact (m/s)	E_kin (J)	Maesured?	Note	
0.6	3.192633028	2.00393E-05	check	-	
0.5	2.948671116	1.70937E-05	check	-	
0.4	2.668593121	1.40007E-05	check	-	
0.3	2.338660335	1.07527E-05	check	-	
0.2	1.932497029	7.34212E-06	check	-	
0.1	1.383070558	3.76073E-06	check	Not everytime, not at far edge	
Drop diameter (mm)	Volume (m^3)	Weigth (g)	terminal velocity (m/s)	E_kin (J)	Still measurable
1.5	1.76715E-09	0.001767146	5.512494617	2.68497E-05	yes
1.25	1.02265E-09	0.001022654	5.032196083	1.29483E-05	yes
1	5.23599E-10	0.000523599	4.500933007	5.30364E-06	yes
0.95	4.48921E-10	0.000448921	4.386966838	4.31984E-06	yes
0.9	3.81704E-10	0.000381704	4.269959969	3.47972E-06	no
0.8	2.68083E-10	0.000268083	4.025756866	2.17237E-06	no
0.7	1.79594E-10	0.000179594	3.765750729	1.2734E-06	no

Table 2: Results of electro valve tests regarding the minimum drop sizes.

CONCLUSION

As different setups of the Drop Counter have been tested the best option can be found by applying scheme 3 with software package V4 to the Arduino board. The rest of the assembly for the sensor is described in the Assembly section. This leads to a sensor which only registers the time interval between drops if a raindrop hits the sensor, provided that the wind speed is low enough and no external triggering is occurring (e.g. insects hitting the sensor). The minimum raindrop size which can be registered is approximately 1mm in diameter.

SOFTWARE VERSIONS (ARDUINO CODE FILES)

SOFTWARE DROP COUNTER V3

```
#include <Wire.h>
#include "RTClib.h"
#include <SD.h>
#include <Filters.h>

// RTC_DS1307 rtc;          // Define the RTC
RTC_PCF8523 rtc;            // Define the RTC

File myFile;                // Declare myFile
const int CS_Pin = 10;      // pin to communicate with SD card
const int analogPin = A2;   // pin that the sensor is attached to
const int threshold = 5;    // threshold level to determine if drop has hit surface
const int under = 1;        // Value used to reset for drift
const int check = 30000;    // time to check if the board is still alive
int checkLight = 3;         // LED burns if drop is detected
int resetDrum = 6;          // Pin to short out piezo drum
int resetCap = 8 ;          // Pin to short out capacitor
int sensorValue = 0;        // Variable to store reading from analogPin
unsigned long time1;        // Time last drop fell (ms)
unsigned long time2;        // Time new drop fell (ms)
unsigned long time3;        // Auxilliary debug variable (ms)
unsigned long time4;        // Auxilliary debug variable 2 (ms)
unsigned long interval;     // Time interval between drop (ms)
String myString;            // Declare myString
char daysOfTheWeek[7][12] = {"Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"};

//   SETUP SECTION //
void setup() {
 Serial.begin(9600); // baud rate
   while (!Serial) {
   ; // wait for serial port to connect.
 }

   Serial.print("Initializing SD card...");

 if (!SD.begin(CS_Pin)) {
   Serial.println("initialization failed!");
   return;
 }
 Serial.println("initialization done.");

 if (! rtc.begin()) {
   Serial.println("Couldn't find RTC");
   while (1);
```

```
}

if (! rtc.initialized()) {              // for RTC_DS1307: rtc.isrunning, RTC_PCF8523 rtc.initialized
  Serial.println("RTC is NOT running!");
}

// Write SD file on base of RTC time.
// Upload the following line once, if RTC is not correct:
// rtc.adjust(DateTime(F(__DATE__), F(__TIME__)));

DateTime now = rtc.now() ; // manually adjusted for Myanmar UTZ (normal RTC adjustment did not seem to work)
myString = String(now.day()) + "_" +  String(now.hour()) + "_" + String(now.minute()) ;
myFile = SD.open(String(myString + ".txt"), FILE_WRITE);

// To check if the right time is used, print to serial:
if (myFile) {
  Serial.println("'Writing initial time to  DD_HH_MM..." );
  Serial.println(String("Filename = " + myString));
  Serial.print(now.year(), DEC);
  Serial.print('/');
  Serial.print(now.month(), DEC);
  Serial.print('/');
  Serial.print(now.day(), DEC);
  Serial.print(" (");
  Serial.print(daysOfTheWeek[now.dayOfTheWeek()]);
  Serial.print(") ");
  Serial.print(now.hour(), DEC);
  Serial.print(':');
  Serial.print(now.minute(), DEC);
  Serial.print(':');
  Serial.print(now.second(), DEC);
  Serial.println();

  Serial.print(" since midnight 1/1/1970 = ");
  Serial.print(now.unixtime());
  Serial.print("s = ");
  Serial.print(now.unixtime() / 86400L);
  Serial.println("d");
  myFile.println(now.unixtime());
  // close the file:
  myFile.close();
  Serial.println("done.");
} else {
  // if the file didn't open, print an error:
  Serial.println(String("error opening " + myString ) );
}
```

```
 pinMode(resetDrum, OUTPUT);
 pinMode(resetCap, OUTPUT);
 time1 = millis();            // Start time
}

//    DATA ACQUIRING SECTION    //
void loop() {
 // read the input on analog pin:
 DateTime now = rtc.now() ; // manually adjusted for Myanmar UTZ (normal RTC adjustment did not seem to work)
 // analogReference(INTERNAL);
 sensorValue = analogRead(analogPin);   // Analog reading, which goes from 0 - 1023 (voltage 0V - 5V):

 if (sensorValue > threshold) {
  time2 = millis();               // Sets time drop fell

  digitalWrite(checkLight, HIGH);   // LED on
  digitalWrite(resetDrum, HIGH);    // Shorten piezo drum to stop it from vibrating
  digitalWrite(resetCap, HIGH);     // Discharge capacitor

  interval = time2 - time1;        // Interval between drops
  time1 = time2;                  // Reset old time

  Serial.println(interval);
  Serial.println(sensorValue);

  myFile = SD.open(String(myString + ".txt"), FILE_WRITE);
  myFile.println(interval);
  myFile.close();

  delay(20);
  digitalWrite(checkLight, LOW);   // LED off
  delay(30);
  digitalWrite(resetDrum, LOW);    // Stop shortening piezo drum
  digitalWrite(resetCap, LOW);     // Stop discharging capacitor
  delay(1);               }

 // serial check if the meter is still alive
 if (millis() - time3 > 10000) {
  Serial.println("Still alive!");
  time3 = millis();             }
 // check if the meter is still alive every hour and write to SD
 if (millis() - time4 > check) {
  myFile = SD.open(String(myString + ".txt"), FILE_WRITE);
  myFile.println(now.unixtime());
  myFile.close();
  time4 = millis();             }
 }
```

SOFTWARE DROP COUNTER V4

```
#include <Wire.h>
#include "RTClib.h"
#include <SD.h>
#include <Filters.h>

// There are two possible RTC's, uncomment the one neededed.

RTC_DS1307 rtc;              // Define the RTC
//RTC_PCF8523 rtc;           // Define the RTC

/* ----------------------------------------*/
//            DECLARATIONS          //
/* ----------------------------------------*/
File signals;                       // Declare signals file
File myFile;                        // Declare myFile
const int CS_Pin = 10;              // pin to communicate with SD card
const int analogPin = A2;             // pin that the sensor is attached to
const int threshold = 8;              // threshold level to determine if drop has hit surface
const int under = 1;                 // Value used to reset for drift
const int check = 30000;              // time to check if the board is still alive
int checkLight = 3;                  // LED burns if drop is detected
int resetDrum = 6;                   // Pin to short out piezo drum
int resetCap = 8 ;                   // Pin to short out capacitor
int sensorValue = 0;                 // Variable to store reading from analogPin
unsigned long time1;                 // Time last drop fell (ms)
unsigned long time2;                 // Time new drop fell (ms)
unsigned long time3;                 // Auxilliary debug variable (ms)
unsigned long time4;                 // Auxilliary debug variable 2 (ms)
unsigned long interval;              // Time interval between drop (ms)
String myString;                    // Declare myString
float HighFilterFrequency = 10.0;        // filters out changes slower that 10 Hz.
float LowFilterFrequency = 2000.0;       // filters out changes faster that 2 kHz.
char daysOfTheWeek[7][12] = {"Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"};



/* ----------------------------------------*/
//            SETUP SECTION          //
/* ----------------------------------------*/
void setup() {
 Serial.begin(9600); // baud rate
   while (!Serial) {
   ; // wait for serial port to connect.
 }

   Serial.print("Initializing SD card...");
```

```
if (!SD.begin(CS_Pin)) {
  Serial.println("initialization failed!");
  return;
}
Serial.println("initialization done.");

if (! rtc.begin()) {
  Serial.println("Couldn't find RTC");
  while (1);
}

if (! rtc.isrunning()) {              // for RTC_DS1307: rtc.isrunning, RTC_PCF8523 rtc.initialized
  Serial.println("RTC is NOT running!");
}

// Write SD file on base of RTC time.
// Upload the following line once, if RTC is not correct:
// rtc.adjust(DateTime(F(__DATE__), F(__TIME__)));

DateTime now = rtc.now() ;
myString = String(now.day()) + "_" +  String(now.hour()) + "_" + String(now.minute()) ;
myFile = SD.open(String(myString + ".txt"), FILE_WRITE);
signals = SD.open(String("signals.txt"), FILE_WRITE);

// To check if the right time is used, print to serial:
if (myFile) {
  Serial.println("Writing initial time to  DD_HH_MM..." );
  Serial.println(String("Filename = " + myString));
  Serial.print(now.year(), DEC);
  Serial.print('/');
  Serial.print(now.month(), DEC);
  Serial.print('/');
  Serial.print(now.day(), DEC);
  Serial.print(" (");
  Serial.print(daysOfTheWeek[now.dayOfTheWeek()]);
  Serial.print(") ");
  Serial.print(now.hour(), DEC);
  Serial.print(':');
  Serial.print(now.minute(), DEC);
  Serial.print(':');
  Serial.print(now.second(), DEC);
  Serial.println();

  Serial.print(" since midnight 1/1/1970 = ");
  Serial.print(now.unixtime());
  Serial.print("s = ");
```

```
  Serial.print(now.unixtime() / 86400L);
  Serial.println("d");
  myFile.println(now.unixtime());
  // close the file:
  myFile.close();
  Serial.println("done.");
 } else {
  // if the file didn't open, print an error:
  Serial.println(String("error opening " + myString ) );
 }

 pinMode(resetDrum, OUTPUT);
 pinMode(resetCap, OUTPUT);
 time1 = millis();            // Start time

}

/* ----------------------------------------*/
//              LOOP SECTION            //
/* ----------------------------------------*/
void loop() {
 // read the input on analog pin:
 // analogReference(INTERNAL);
 FilterOnePole highpassFilter(HIGHPASS,HighFilterFrequency ); // create a one pole (RC) highpass filter
 FilterOnePole lowpassFilter( LOWPASS, LowFilterFrequency  ); // create a one pole (RC) lowpass filter

 while( true ) {
  DateTime now = rtc.now() ;
  sensorValue = lowpassFilter.input( highpassFilter.input( analogRead( analogPin ) )  );    // Analog reading, which goes from 0 - 1023
(voltage 0V - 5V):

  if (sensorValue > threshold) {
   time2 = millis();               // Sets time drop fell
   Serial.println(sensorValue);
   digitalWrite(checkLight, HIGH);   // LED on
   digitalWrite(resetDrum, HIGH);    // Shorten piezo drum to stop it from vibrating
   digitalWrite(resetCap, HIGH);     // Discharge capacitor

   interval = time2 - time1;        // Interval between drops
   time1 = time2;                  // Reset old time

   Serial.println(interval);
   Serial.println(sensorValue);

   myFile = SD.open(String(myString + ".txt"), FILE_WRITE);
   myFile.println(interval);
   myFile.close();
```

```
   digitalWrite(checkLight, LOW);   // LED off
   digitalWrite(resetDrum, LOW);    // Stop shortening piezo drum
   digitalWrite(resetCap, LOW);     // Stop discharging capacitor
   delay(1);                }

// serial check if the meter is still alive
if (millis() - time3 > 10000) {
   Serial.println("Still alive!");
   time3 = millis();             }
// check if the meter is still alive every hour and write to SD
if (millis() - time4 > check) {
   myFile = SD.open(String(myString + ".txt"), FILE_WRITE);
   myFile.println(now.unixtime());
   myFile.close();
   time4 = millis();             }

}
}
```