FEDERAL STATE BUDGET EDUCATIONAL INSTITUTION OF HIGHER EDUCATION
"SAINT PETERSBURG STATE UNIVERSITY"
(SPbSU)

Educational program "Engineering-oriented physics"

Coursework report
6th semester
“Creation of a portable circuit for generating and measuring signals to implement a technique for diagnosing high-voltage conductivity of liquid dielectrics”

Completed by a 3rd year undergraduate student:
Tyuterev M. I.

Scientific adviser:
Vasilkov S. A.

Saint Petersburg
2022

Table of contents

Introduction

At the moment, diagnostics of high-voltage conductivity of liquids is carried out exclusively in the laboratory, but this is not always convenient. It would be much better to carry it out directly in the field. For these purposes, it was proposed to create a small device programmed to generate and measure a signal. As a result, for field testing you will only need a laptop, a measuring cell (for oil), a miniature high-voltage source and this most compact signal generation and measurement module.

Objectives of this work

Describe the concept of the device and create its prototype

Tasks completed during work

- Creation of a portable unit for generating a signal of a given shape, controlling a compact high-voltage source using this signal, measuring the resulting voltage and current, and transmitting data to a computer
- Hardware implementation of voltage generation and signal measurement is carried out using Arduino
- Managing the corresponding block, as well as processing the recorded data in Python

Theoretical description

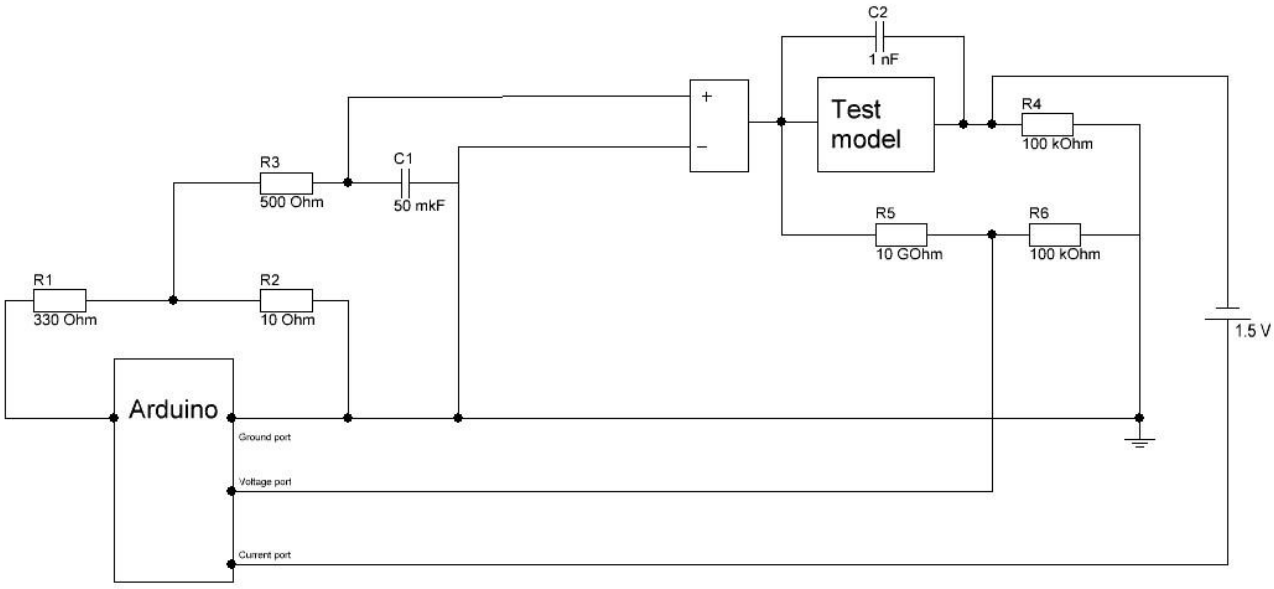
The work will use Arduino in signal generator and receiver mode. This device generates a PWM signal up to 5 V, the amplitude of which is determined by the duty cycle of the pulses. To control a high-voltage source, a smoothly varying voltage is used, so in order to receive a constant signal, you need to pass this PWM signal through a low-pass filter with a certain parameter $\tau = RC$.

Diagnostics of liquid dielectrics is carried out using DCVC (dynamic current-voltage characteristic). You need to be able to get rid of currents caused by parasitic capacitances. For this, an additional signal (impulse response) is created, and it is believed that after the cessation of the impact, the discharge of parasitic capacitances will begin, from the graph of which the value of this capacitance will be found.

Main results of the practice

Installation diagram

The complete installation diagram is shown in the figure:



Drawing1. Installation diagram

Now let's take a closer look at the description of this installation.

A PWM signal of up to 5 V is supplied from Arduino, however, to generate voltage from a high-voltage source, it is necessary to get rid of both PWM modulation and lower the voltage to 0.3 V (at this voltage value supplied to the generator, a voltage of 10 kV will be output). Therefore, to begin with, the signal is fed to a voltage divider across resistors, which reduces the amplitude of the signal. This smaller amplitude signal is fed to a low-pass filter with a pre-selected parameter $\tau = RC$, which allows you to get rid of modulation.

Then the signal supplied to the generator increases to 10 kV and is applied to the element under test. A capacitor is connected in parallel to this element to test one of the parts of the program. In a real experiment, capacitor C2 must be turned off.

Finally, we need to move on to measuring current and voltage. The current is taken through a measuring resistor, and the voltage is, again, through a voltage divider. Also, due to the fact that Arduino only measures positive voltage values, it is impossible to measure negative current, which may appear due to the presence of parasitic capacitances, it was decided to add an additional 1.5 V source.

Writing code

First of all, it was necessary to decide in which programming language this program should be written. There were 3 options: Matlab, python and the Arduino language itself (aka C++). Matlab and its package for working with Arduino were not purchased, in addition, the sampling frequency of the received signal was about 10 Hz (it is necessary to take data from Arduino with a frequency of about 500 Hz). So, it was decided not to use Matlab.

It seemed easier to work with python than with Arduino itself, so it was decided to check with what frequency this language can interact with Arduino in order to understand whether it is suitable in this case. When using the firmware for the Arduino Pyfirmata board, the maximum sampling frequency turned out to be about 500 Hz. This value is suitable for generating and collecting data, so it was decided to use python as the programming language. The firmware itself is basic for the selected board and is found in the code examples for Arduino in the IDE

It was proposed to identify functionally separate blocks (classes) in the code using OOP, in which it would be convenient to receive, store and carry out operations with data.

Classes:

1. HardwareMaster is a class for controlling Arduino. It communicates with it and performs the procedure for generating and collecting a signal.
2. DCVC is a class with all the data. Stores all the data of a specific experiment, creates a function that will be generated, writes and reads the file
3. SignalProcessing – class of functions for processing an existing signal
4. Experiment is a generic class. It describes possible scenarios for conducting the experiment.

The principle of implementing the program to obtain TWO:

1) Communication with Arduino is in progress
2) A pulse signal is created
3) It is generated, data collected and filtered from the carrier frequency
4) The function is minimized for parameter C (stray capacitance) and its value is found
5) The TWO signal is created
6) Point 3 is repeated
7) The parasitic currents are subtracted and the result is TWO

Comparison with LCard

Building a Model in Simulink

To test the operation of the Arduino board on a low-voltage source, the entire circuit in Figure 1 is not needed. Therefore, a simplified model system was chosen. Only the low-pass filter remained in it and a 1000 Ohm resistor was added to measure the current (see the diagram in Simulink in Figure 2).

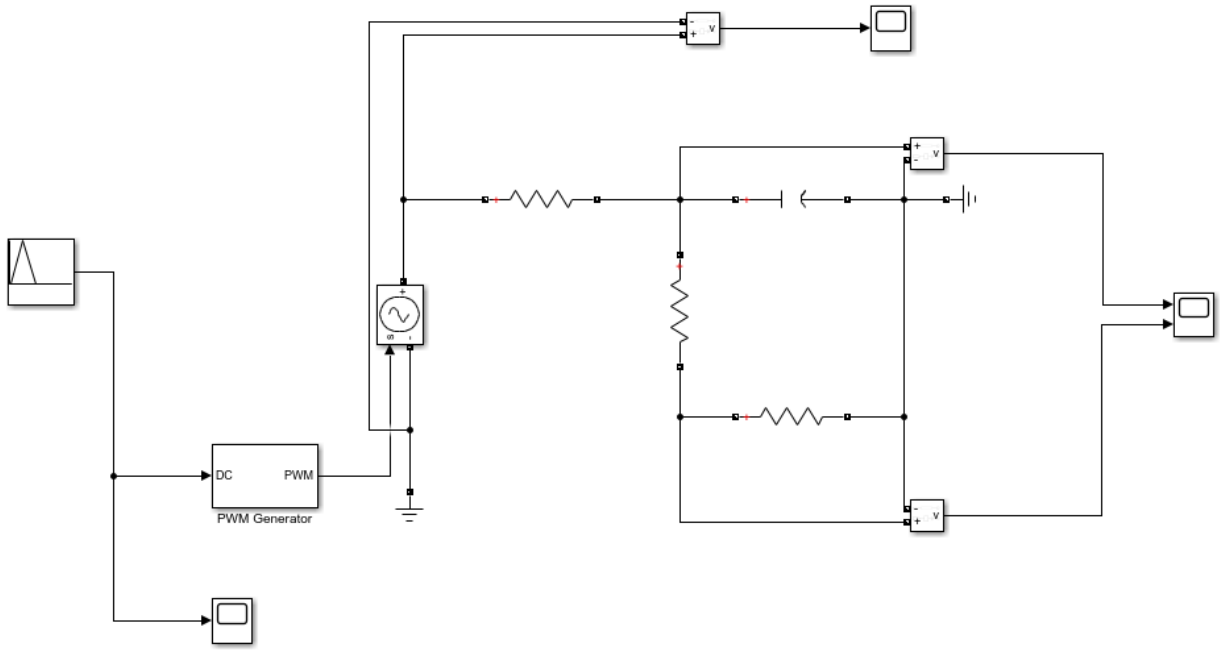


Figure 2. Circuit in Simulink

To better understand which filter parameters need to be selected, we will use the Simulink program. Let's draw a diagram representing our filter (in Figure 2) and select parameters. It was decided to settle on resistances R1 = 500 Ohm, capacitance C = 50 μF (similar to the circuit in Figure 1) and R2 = 1000 Ohm (current is removed from it).

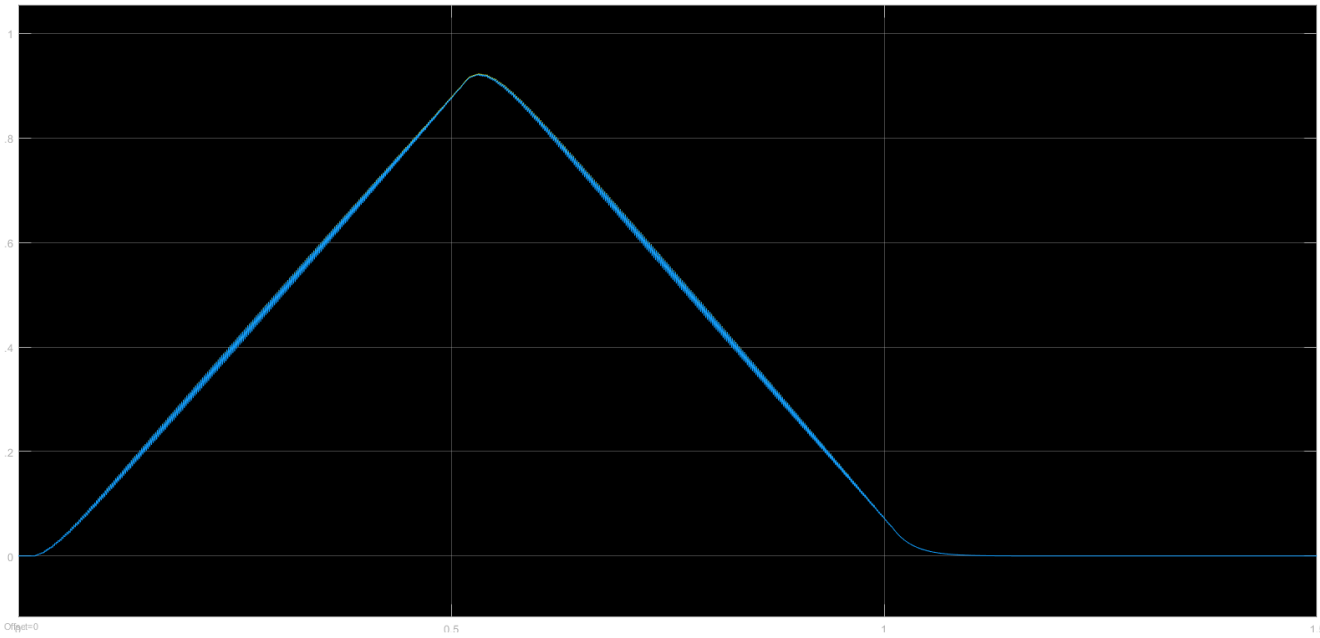


Figure 3. Simulink

A signal similar to that supplied by Arduino was supplied to the filter input (see Figure 4)

Low voltage source

In order to understand whether the data we are receiving using Arduino is generally correct, it was decided to take several test characteristics using a special LCard program.

The following dependency has been removed:

TWO 50 counts (see Figure 4). This is the signal that is sent by the program to the Arduino board. Then PWM modulation of this signal is performed and fed to the circuit.

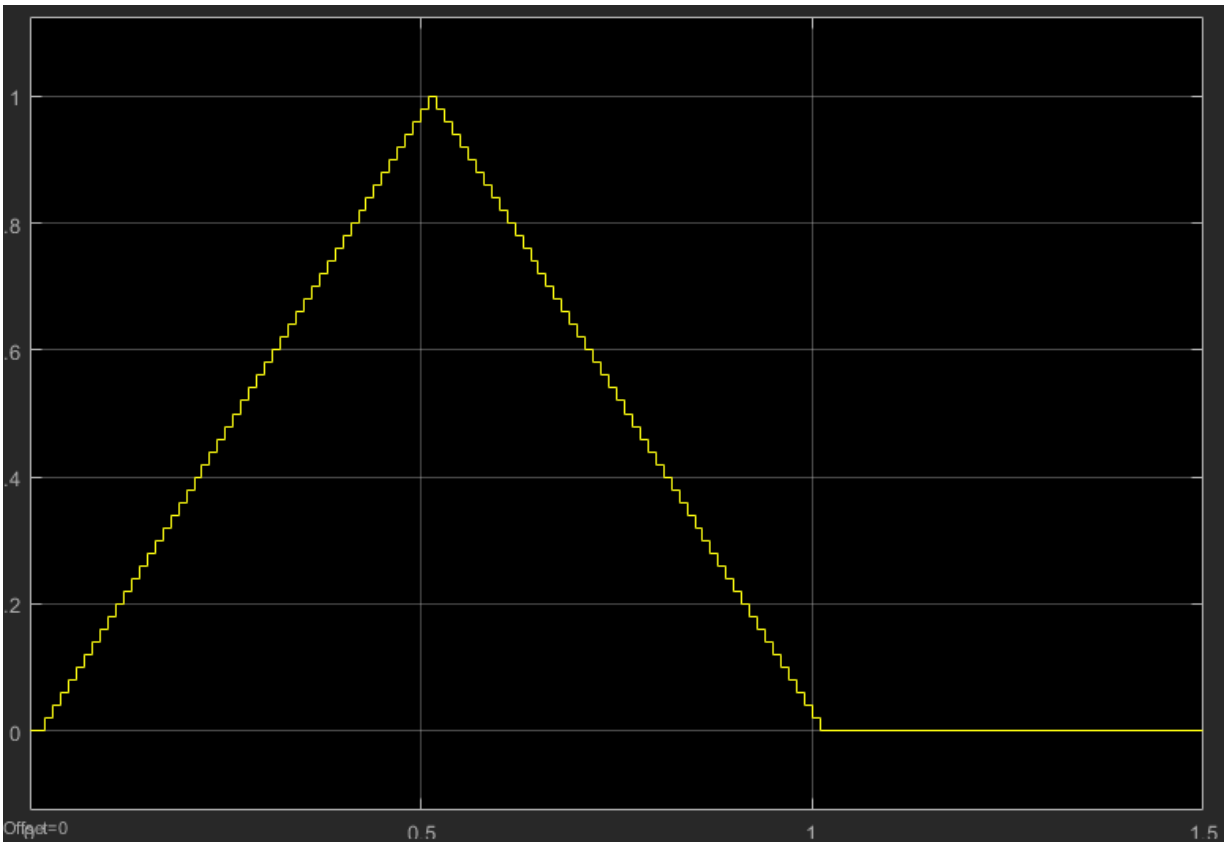


Figure 4. Signal generated by the program (50 samples)

Let's check how much the data obtained using Arduino and LCard converge. Here are graphs of voltage and current versus time

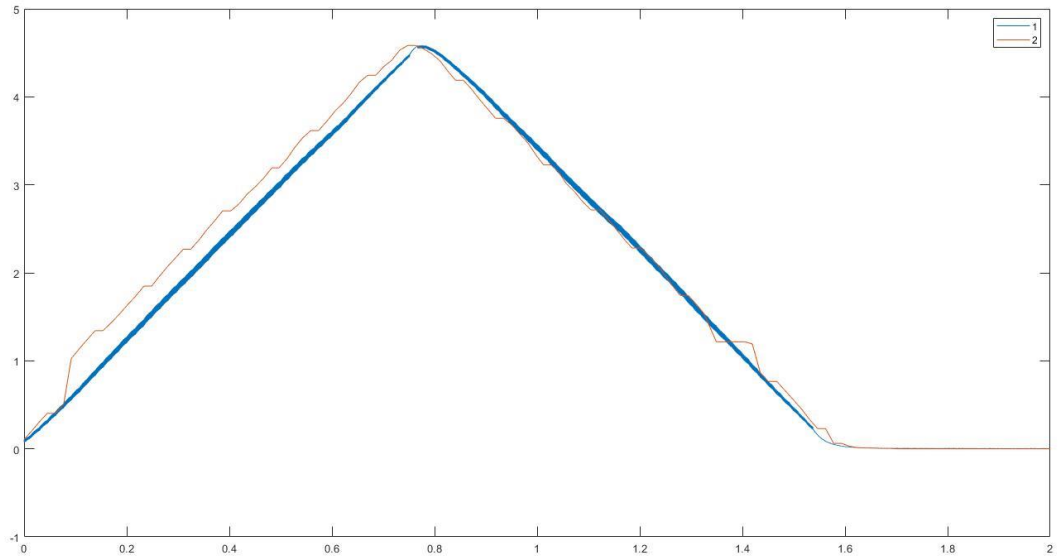


Figure 5. Voltage

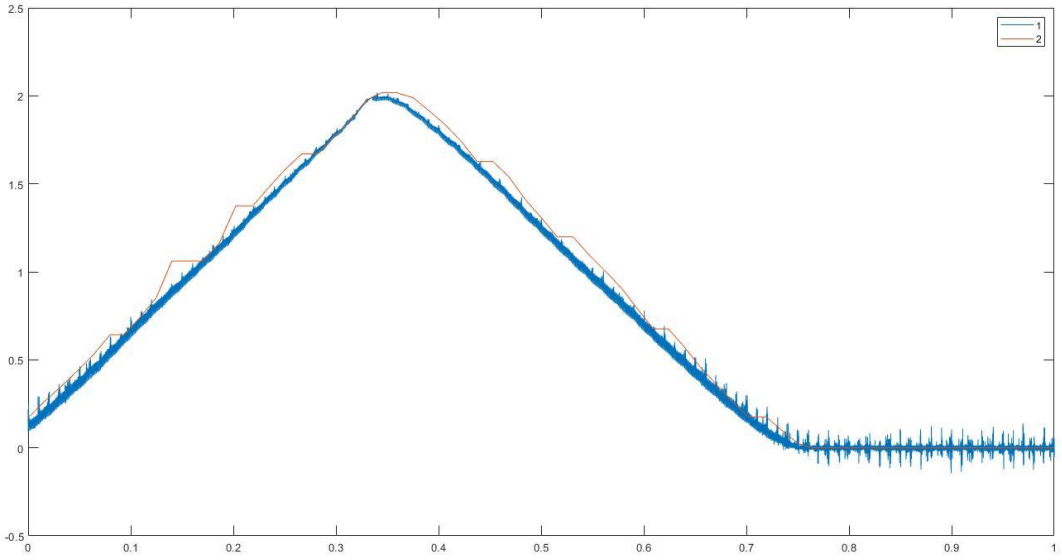


Figure 6. Current

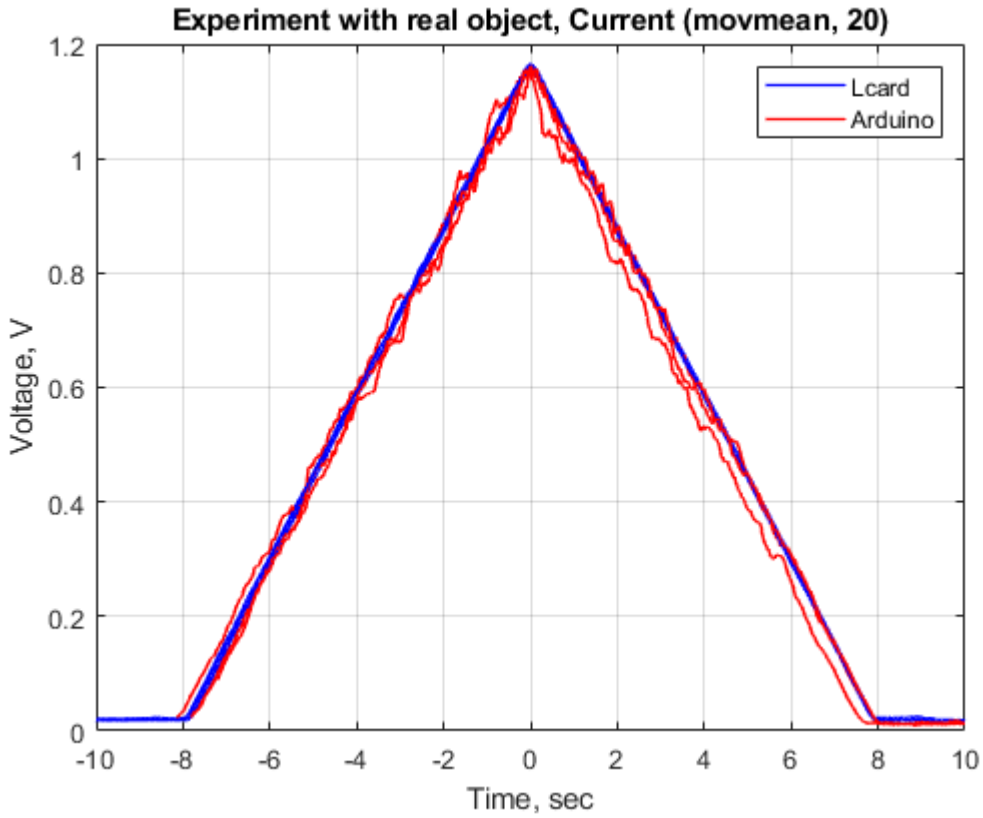
A similar picture is observed in other graphs. So, we can conclude that the Arduino ports are working normally.

As you can see, the signal generation time differs by approximately 1.5 times (not 1, but 1.5 seconds). This is due to the not entirely correct setting of the sampling frequency. At the moment the program is slowing down for this time. This is not the most efficient implementation option and is subsequently recommended to be improved.

Experiment with high voltage source

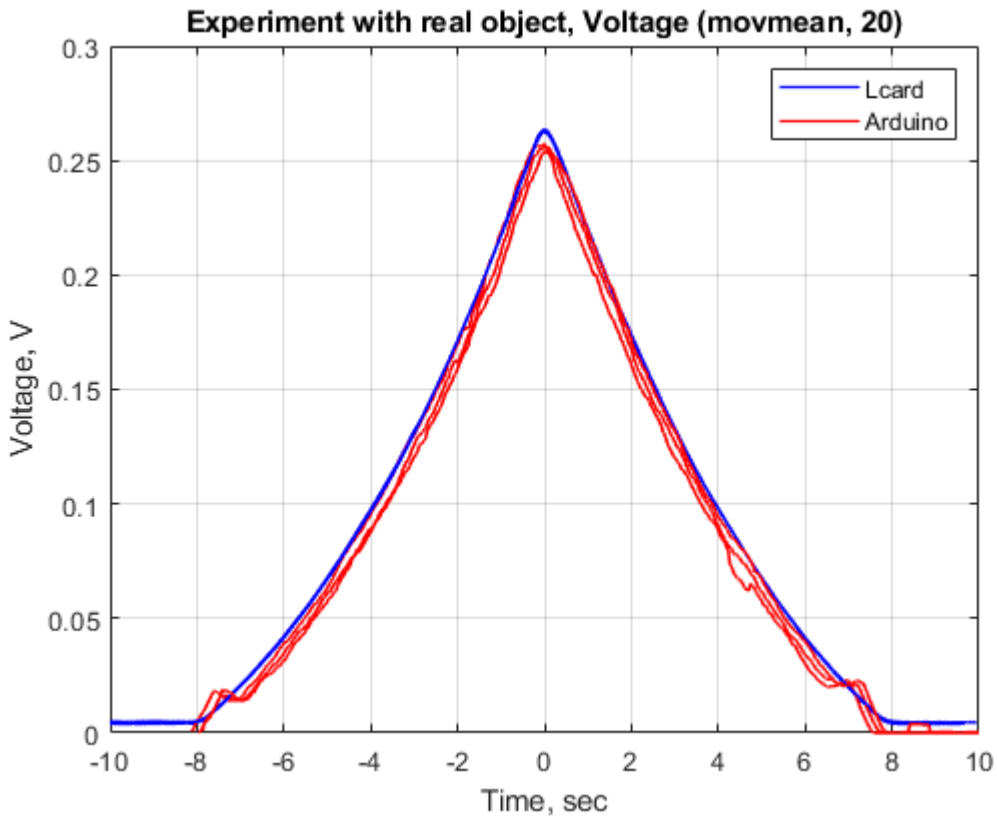
After checking the operation of the Arduino at low voltage, we will assemble the installation with a high-voltage generator and carry out a series of measurements to check the operation of the Arduino together with the high-voltage generator. This generator must produce a linearly increasing voltage with a maximum of 10 kV. This voltage corresponds to the voltage supplied to the generator similar to that shown in Figure 1, but with a maximum = 0.3 V. In order to achieve such a maximum, we pass the signal through a voltage divider across resistors. To lower the voltage from 5V to 0.3V, resistances R1 = 10 Ohms and R2 = 330 Ohms were selected. Let's change the pulse generation time from 1 to 10 seconds (in fact it will be about 15 seconds) so that there are more counts during generation. Arduino starts recording the singal simultaneously with the modulation, so zero noise is not observed.

After passing through the high-voltage part of the installation, the voltage is again divided and supplied for measurement in the LCard and Arduino. Just as in the last part of the work, we will compare how well the microcontroller measures voltage and current. To get rid of noise, let's pass the signal through the movmean function, which averages the function over neighboring values. 3 measurements were taken to check the Arduino for random errors.



Drawing2. Dependence of voltage on time.

These graphs are combined at the maximum point, because the graphs are recorded separately. The current is then determined by recalculation (dividing the voltage by the known resistance from which this voltage was removed)

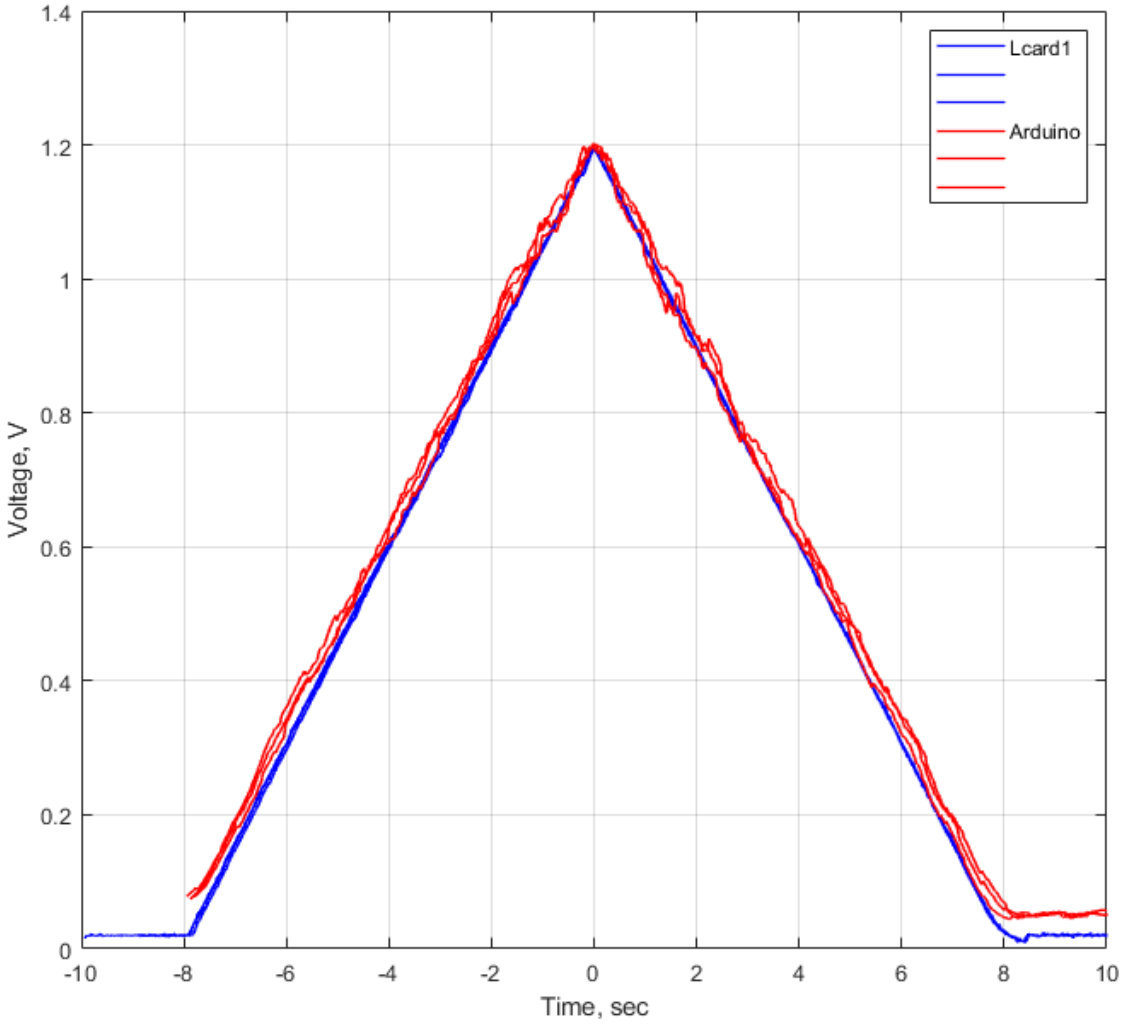


Drawing3. Current versus time

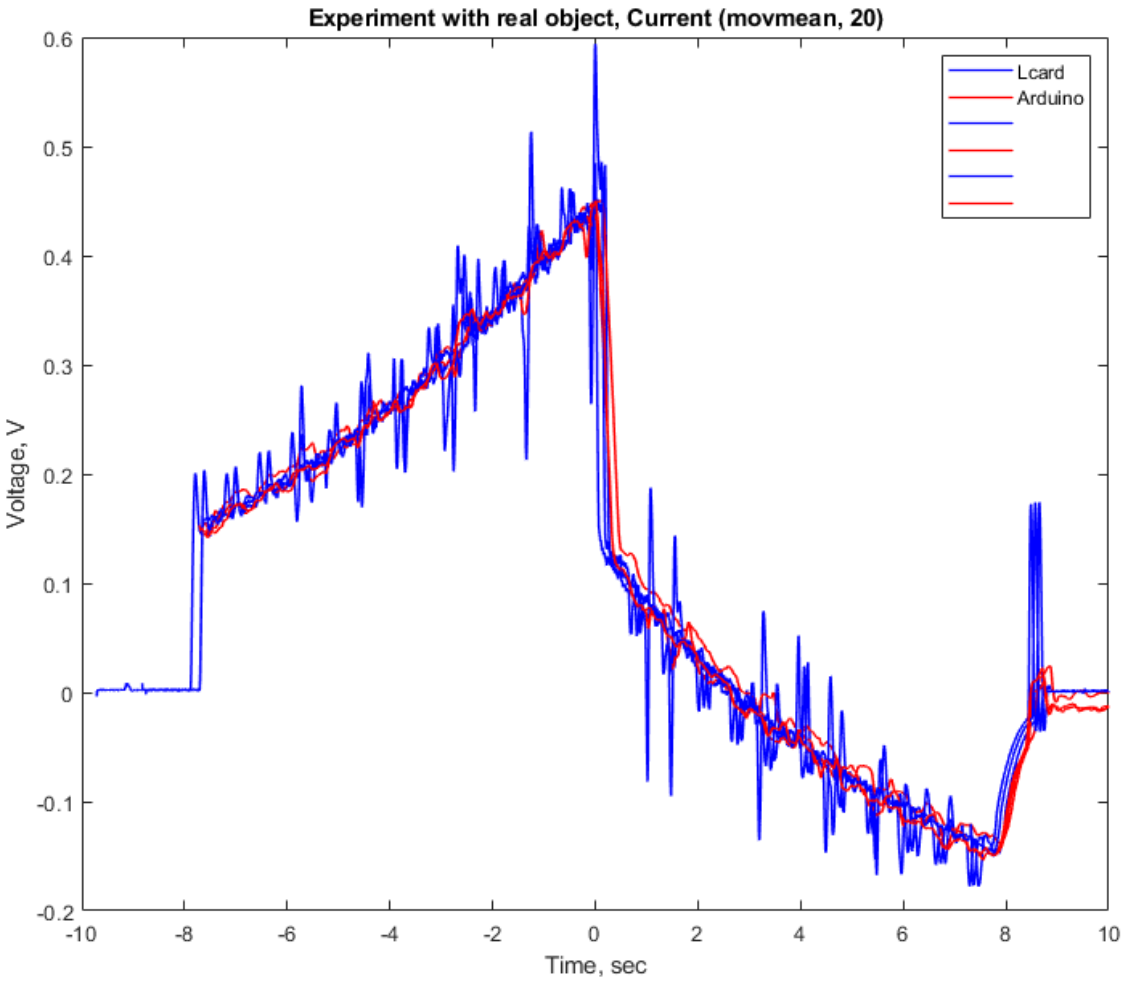
As you can see, the microcontroller measures the values quite well. We can conclude that this device can be used in the field in the future.

Subtraction of capacitive currents

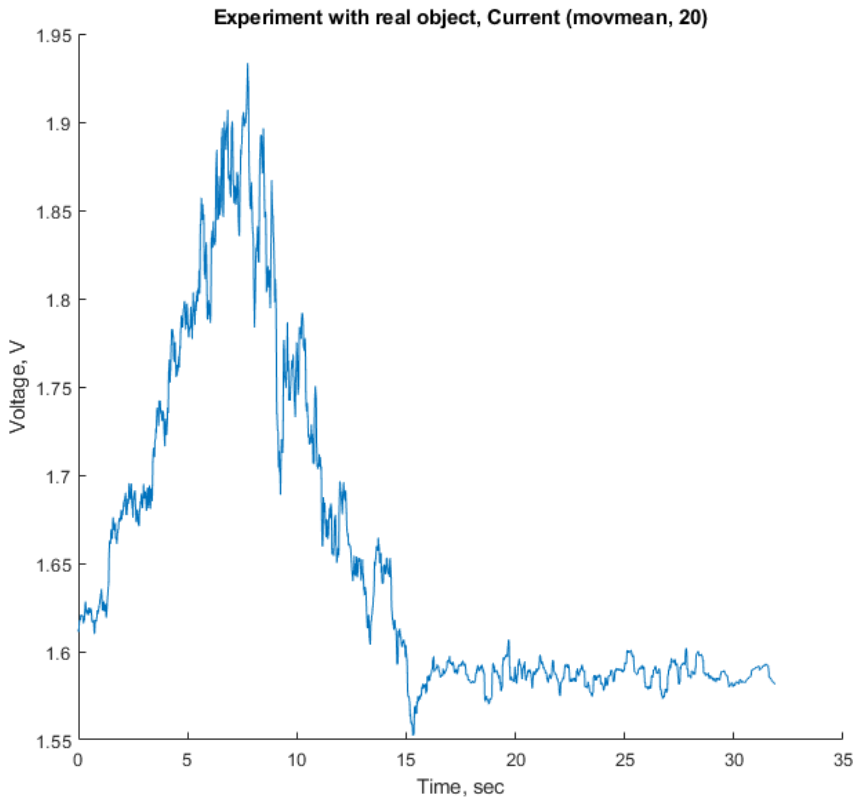
In real measurements, quite significant capacitive currents can occur due to parasitic capacitances. Post-processing of the results should help eliminate them. In this part of the work, we will check how well this code copes with this task. It's worth noting first that Arduino only measures positive values, and stray currents can lead to negative voltages. So that they can also be measured, we connect a 1.5V AA battery in series to the measured value to slightly shift the measured value. Thus, we will try to analyze the system by adding a capacitor with a capacitance C = 10-9 F in parallel to our installation and determine the current by discarding the capacitive one.



Drawing4. Voltage



Drawing5. Current before parasitic subtraction (battery voltage excluded)



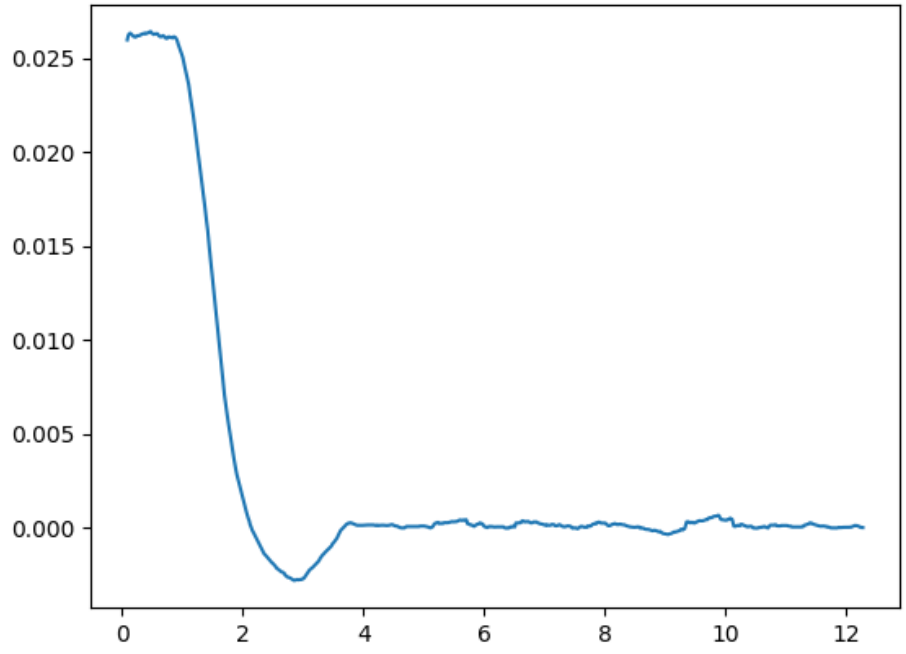
Drawing6. Current after getting rid of parasitic currents (battery voltage remains)

As you can see, the values, even with averaging, turned out to be very jumpy. This is due to how exactly the voltage derivative (the difference of the nearest values) was taken.

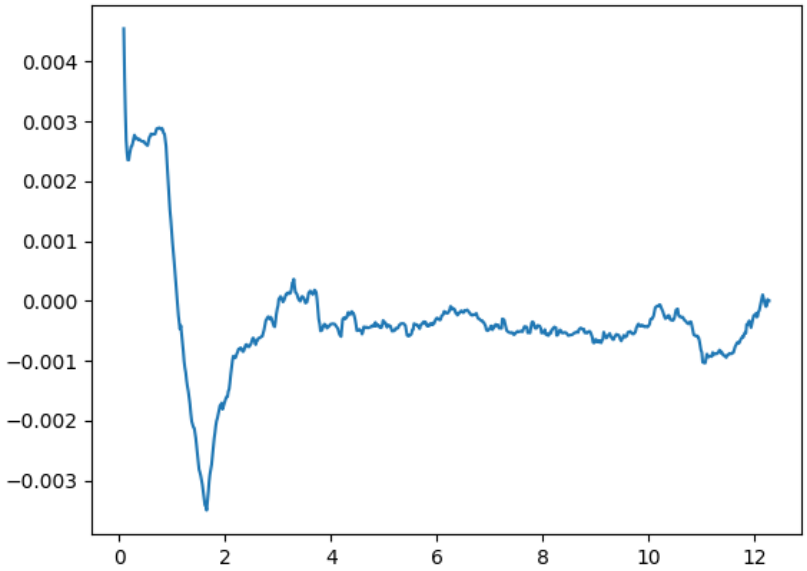
Determination of parasitic capacitances

In a real system, the value of parasitic capacitance is obviously unknown and, accordingly, must be determined. To do this, let's pass a synchronization pulse through the system (a signal of about 1 kV with a duration of about 1 second). Let's determine the dependences of voltage and current (in Figures 12 and 13) and then find the capacitances

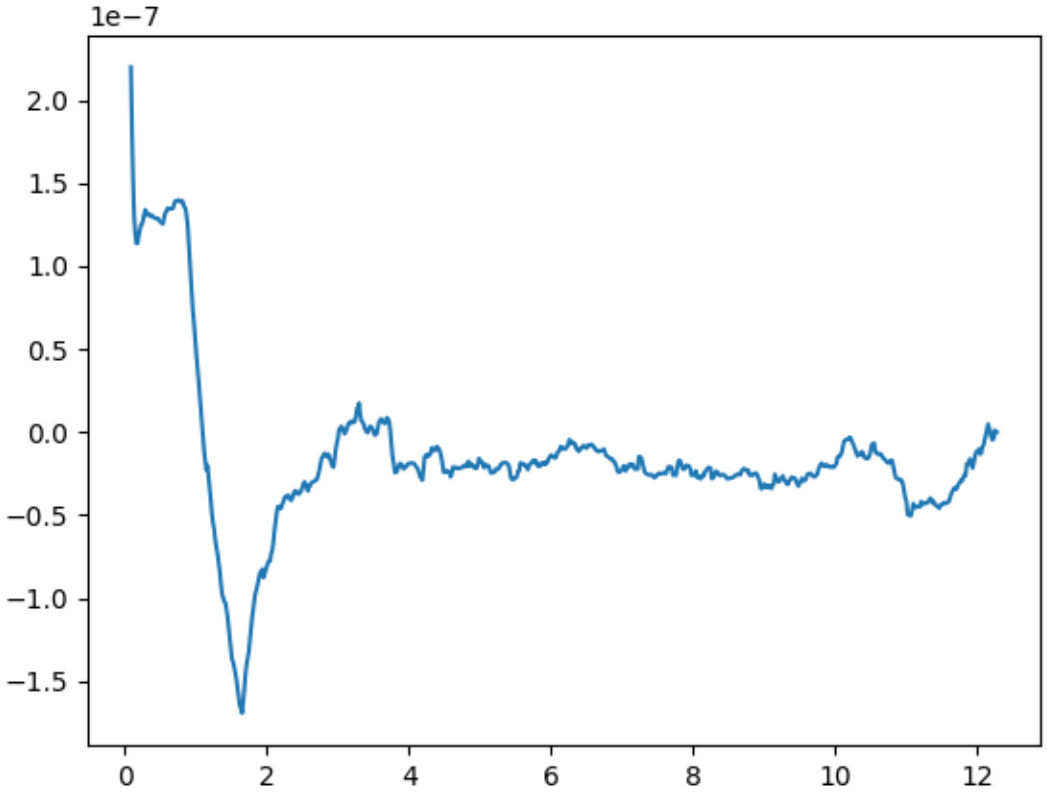
by searching for the minimum of the function of the current difference and the voltage derivative multiplied by an unknown value.



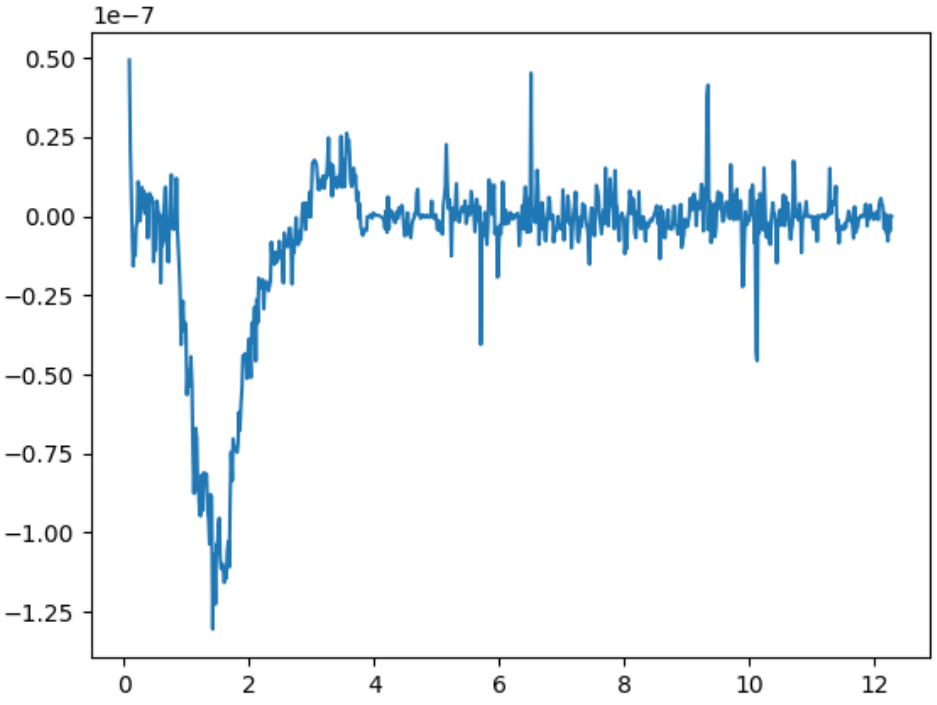
Drawing7. Voltage (value on Arduino)



Drawing8. Current (value on Arduino)



Drawing9. Real current value



Drawing10. Derivative of voltage multiplied by the found capacitance value

The capacitance value turned out to be = 7*10-11 F, which is an order of magnitude less than the capacitance of the capacitor under study. Apparently, this is again due to the inaccuracy of determining the derivative and the large number of emissions that this type of implementation produces.

Further development:

First of all, it should be noted that in this version the installation is already ready for operation. The hardware and code are written, the setup is assembled and can be used to calculate TWO high-voltage systems. However, there are a few points worth noting for improvement:

1. It would be more convenient to create a graphical interface for more convenient work with this code and selection of all parameters.
2. Change the approach to signal sampling. Instead of "sleep", add a comparison with the time function.
3. When stopping the program, Arduino does not stop producing a residual signal, this also needs to be fixed
4. As was shown in the previous part of the work, the graphs of voltage and current are not smooth functions, but have a large number of outliers. Accordingly, searching for the voltage derivative as the difference of two neighboring values potentially leads to even larger overshoots. It is proposed to replace such a derivative search with a more complex type (for example, the derivative of an approximating function).
5. In the future I would like to add new functionality for Arduino, such as a temperature sensor.
6. Add surge protection device
7. Replace the additional source (battery) with the correct determination of negative currents and voltages
8. Add calibration to Arduino reference value

Conclusion

Based on the results of this coursework, a prototype device was designed and created to determine the DCVC of the model under study.