Supplementary Information *concerning the manuscript*

Studying Physics, getting to know Python: RC circuit, simple experiments and coding with Raspberry Pi

Submitted to **Computing in Science and Engineering, 2020**

Contents

Listings

1 Part list

- Raspberry Pi3 model B (`www.raspberrypi.org`)
  - Power supply
  - Micro SD card with Raspbian Operating System (some sellers provide the Operating System already loaded on the micro SD card).
  - (suggested) Case with opening for access to the GPIO contacts

  - Keyboard USB
  - Mouse USB
  - Display with HDMI port (we use a Samsung S24D330 Monitor 24” Full HD, 1920 x 1080)
  - HDMI cable
- Extension cable for the GPIO (Kuman)
- Solderless breadboard (Kuman)
- Jumpers (Kuman)
- LED
- $330\,\Omega$ resistor
- MCP 3008 Analog to Digital Converter (on Amazon, DigiKey, Mouser)
- $1\,k\Omega$ resistor (3 pcs)
- $10\,k\Omega$ resistor
- $100\,\mu F$ (electrolytic) capacitor

2 Experiment zero: turn a LED on and off

Lighting a Light Emitting Diode. A 330 Ω resistor is connected in series to the diode in order to limit the current that flows from the RPi through the LED [1]. One terminal of the series is connected to a ground (GND) pin of the GPIO, the other terminal of the series is connected to the GPIO pin 17, a pin selectable for digital output. The LED has to be correctly polarized, connecting the shorter leg (the cathode) to the lower potential.
An individual GPIO pin can only safely provide 16 mA [2], [3]. If we use a maximum voltage of +3.3 V, according to Ohm's law the minimum resistance connected between the high voltage pin and the ground pin should be $R_{min} = +3.3\,V/16\,mA \simeq 206\,\Omega$. Moreover, the GPIO pins can provide all together a maximum current of 50 mA, distributed on all the pins [2], [4].

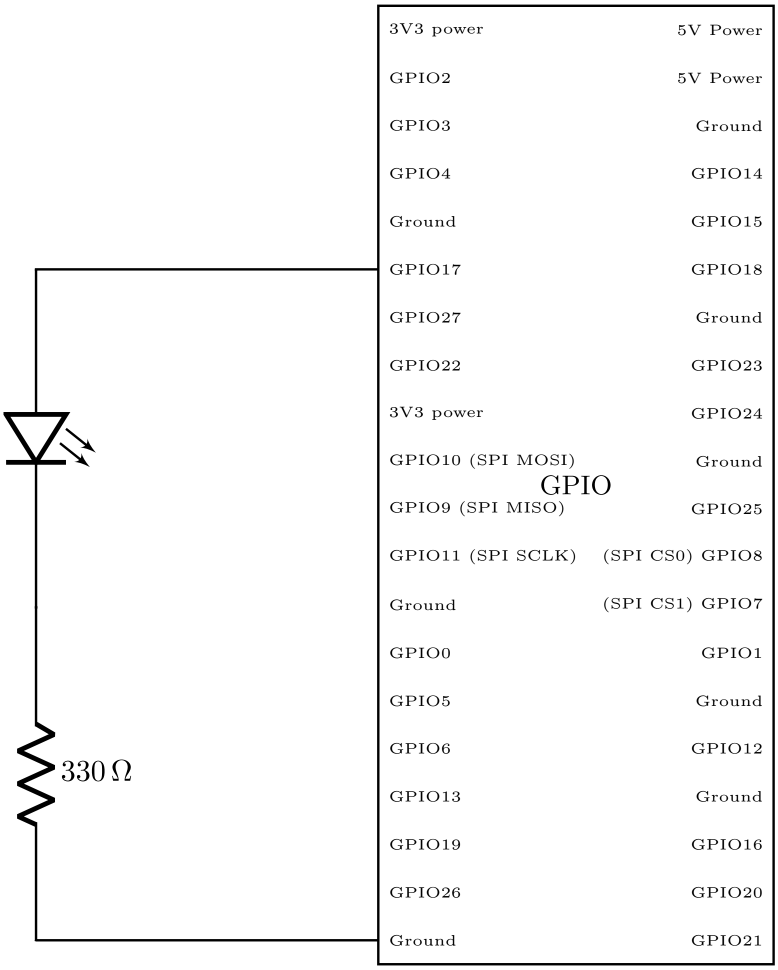

Figure S.1: Experiment Zero.

Listing 1: **gpio_switching.py**

```
from gpiozero import LED
from time import sleep

led=LED(17)

while True:
    led.on()
    sleep(0.5)
    led.off()
    sleep(0.5)
```

Code from the examples in [5], [6].

3 Read analog signals: measure a constant voltage

3.1 Connecting the MCP 3008

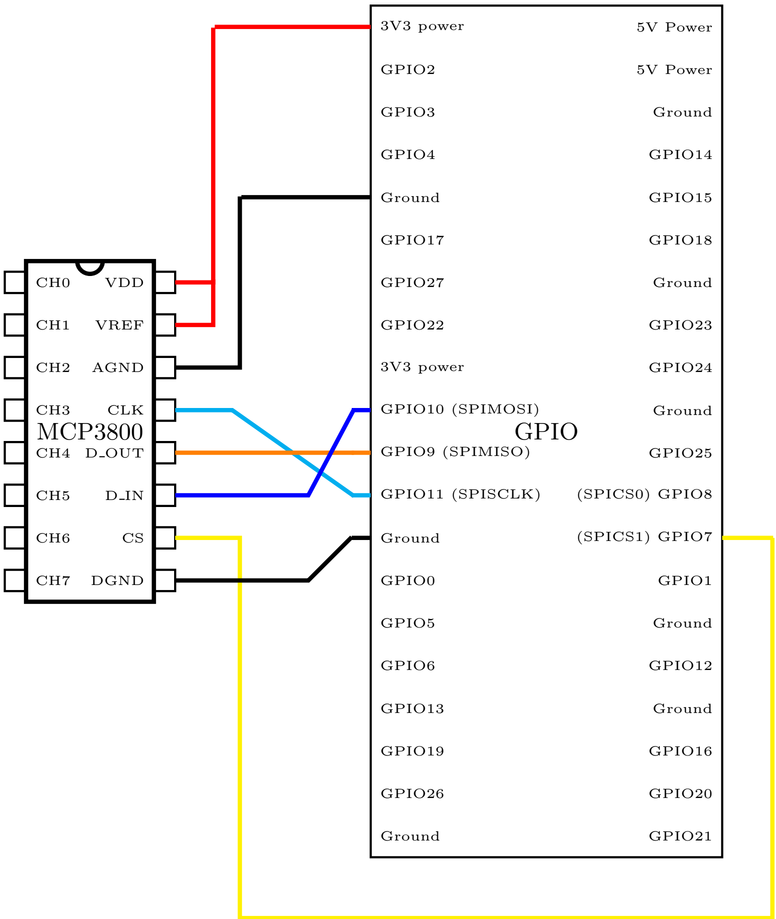

Figure S.2: Reading analog signals. Connections between the Raspberry Pi and the Analog to Digital Converter, the integrated circuit MCP3008.

3.2 Reading analog signals

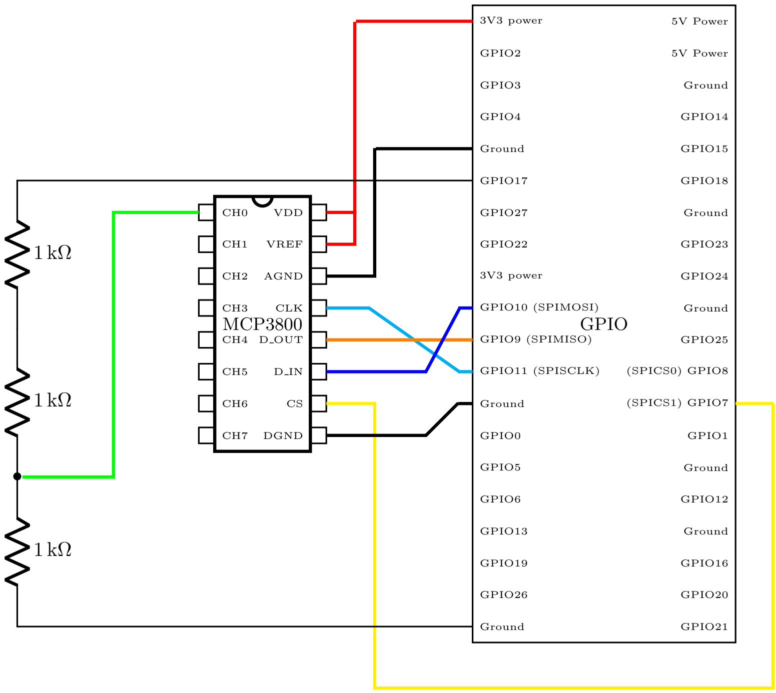

Figure S.3: Measuring on a voltage divider.

Measuring the voltage on a resistor of a voltage divider. We use the pin GPIO17 for applying a 3.3 V tension to the series of three equal resistors, so that a 1.1 V voltage is expected between the two terminals of each resistor. We choose the resistor connected to GND and we connect the other terminal of the resistor to the channel CH0 of the ADC converter.

The following script **measure_voltage.py** allows to read and print the voltage on one of the resistors of the series when the circuit is powered on and off, respectively. Since the values are transmitted from the MCP3008 to the RPi using the SPI protocol (Serial Peripheral Interface), the RPi configuration has to be previously modified in order to activate the SPI support [7], [8][1].

Listing 2: **measure_voltage.py**

```python
from gpiozero import LED, MCP3008
from time import sleep

power = LED(17)
pot = MCP3008(0)
Vdc = 3.3

while True:
    power.on()
    print(pot.value * Vdc)
    sleep(1)
    power.off()
    print(pot.value * Vdc)
    sleep(1)
```

Code based on the example in [9].

[1]For activating the SPI support: Raspberry Menu, Preferences, Raspberry Pi Configuration, Interfaces, SPI: Enable.

4 Transient response of the RC circuit

4.1 Wiring diagram

Measuring the voltage while charging and discharging the capacitor. One end of the RC series is connected to a GND pin of the GPIO, the other end is connected to the pin GPIO17. The maximum current drawn from the RPi to the circuit is $I = V_{DC}/R = 33\,\mu A$, well below the recommended limit of 16 mA.

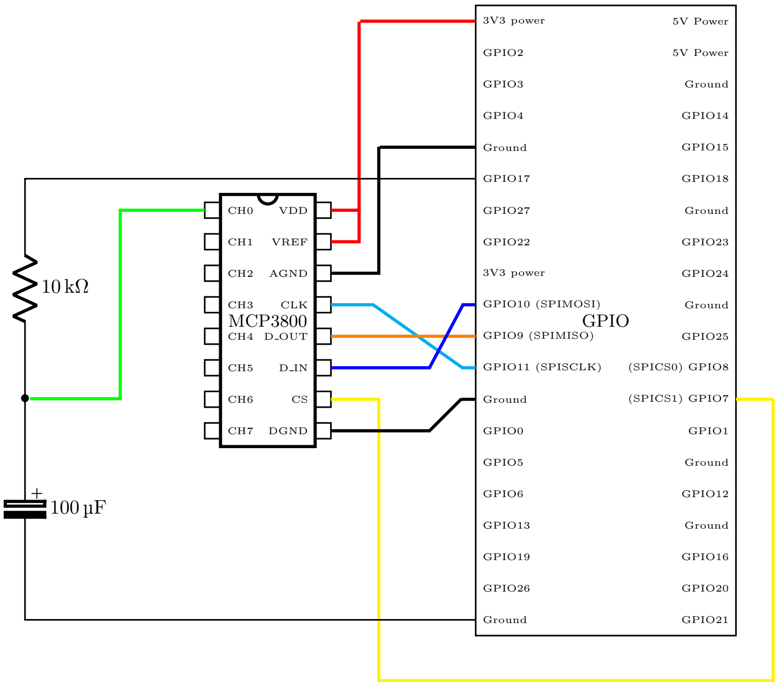

Figure S.4: RC circuit. Wiring diagram.

4.2 Code

Listing 3: **measure_RCtransients.py**

```
from gpiozero import LED, MCP3008
from time import sleep, perf_counter
import numpy as np

power = LED(17)
pot = MCP3008(0)

sleep_time = 0.1
exec_time = 40
v_high_time = 20
Vdc = 3.3
outputFile='voltageLog.txt'

def measurement(time_limit):
    while True:
        time = perf_counter() - start_time
        times.append(time)
        voltage = pot.value * Vdc
        voltages.append(voltage)
        if time > time_limit:
            break
        sleep(sleep_time)

times = [ ]
```

```
voltages = [ ]

start_time = perf_counter()

# charging
t_limit = v_high_time
power.on()
measurement(t_limit)

# discharging
t_limit = exec_time
power.off()
measurement(t_limit)

t = np.array(times)
v = np.array(voltages)

np.set_printoptions(precision = 20)
np.savetxt(outputFile, np.column_stack((t, v)))
```

Listing 4: **plot_voltage.py**

```
#   plot_voltage.py

import numpy as np
import matplotlib.pyplot as plt

# read data from file
t, v = np.loadtxt('voltageLog.txt', delimiter=' ', unpack = True)

# plot of the experimental data
plt.plot(t, v, 'o', color='blue', markersize = 3)
plt.axhline(color = 'gray', zorder = -1)
plt.axvline(color = 'gray', zorder = -1)
plt.xlabel('time  $t$ (s)')
plt.ylabel('voltage $V_C$ (V)')
plt.text(27, 1.8, 'R = 10 k$\Omega$'+'\n'+'C = 100 $\mu$F')

plt.savefig('plot_voltage.pdf')
plt.show()
```

5 Analysis. Modelling the experimental data

From the original file containing the experimental data we extract two subsets of data: one corresponding to charging, one corresponding to the discharging.

The fitting function cannot reproduce discontinuous changes such those occurring at the time t_0 at which the charging process sets in, and at the time t_1 at which the discharging process starts. For this reason from the dataset concerning the charging process we excluded the early points at about 0.0 V, and from the dataset corresponding to the discharging process we excluded the voltage values equal to the V_{dc} plateau value.

5.1 Curve fitting: charging process

Listing 5: **curveFit_charging.py**

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
import scipy.optimize

res = 3.3 / 1023
inputFile = 'chargingProcessData.txt'

# define fitting function
def ChargeProcess(t, V0, t0, tau):
    return V0 * (1-np.exp(-(t-t0)/tau))

# read the experimental data from file
t, v = np.loadtxt(inputFile, delimiter = ' ', unpack = True)

# assign the experimental uncertainty on Voltage values
dv = np.ones(v.size) * res

# initial guesses for fitting parameters
V0_guess, t0_guess, tau_guess = 3., 0., 1.

# fitting procedure
nlfit, nlpcov = \
        scipy.optimize.curve_fit(ChargeProcess, t, v, \
                                p0 = [V0_guess, t0_guess, tau_guess],
                                    sigma = dv,\
                                bounds = (0, 100))
          # we apply bounds to the free parameters so that only positive
              values are allowed.

# obtaining parameters from the best fit procedure
V0, t0, tau = nlfit

# obtaining uncertainties associated with fitting parameters
dV0, dt0, dtau = \
      [np.sqrt(nlpcov[j, j]) for j in range(nlfit.size)]

# create fitting function from fitted parameters
t_fit = np.linspace(t.min(), t.max(), 128)
v_fit = ChargeProcess(t_fit, V0, t0, tau)

# residuals and reduced chi squared
resids = v - ChargeProcess(t, V0, t0, tau)
redchisqr = ((resids/dv)**2).sum()/float(t.size-3)
ndf = t.size-3
# where 3 is the number of free parameters


```

```python
# create figure window to plot data
fig = plt.figure(1, figsize = (8,8))
gs = gridspec.GridSpec(2, 1, height_ratios = [6, 2])

# plotting data and fit
ax1 = fig.add_subplot(gs[0])
ax1.plot(t_fit, v_fit)
ax1.errorbar(t, v, yerr = dv, fmt = 'or', ecolor = 'black', markersize = 2)
ax1.set_xlabel(' time (s)')
ax1.set_ylabel('voltage $V_C$ (V)')
ax1.text(0.5, 0.50, r'$V_0$ = {0:6.4f}$\pm${1:0.4f}'.format(V0, dV0), transform = ax1.transAxes, fontsize = 14)
ax1.text(0.5, 0.40, r'$\tau$ = {0:6.4f}$\pm${1:0.4f}'.format(tau, dtau), transform = ax1.transAxes, fontsize=14)
ax1.text(0.5, 0.30, r'$t_0$= {0:5.4f}$\pm${1:0.4f}'.format(t0, dt0), transform = ax1.transAxes, fontsize=14)
ax1.text(0.5, 0.20, r'$\chi_r^2$ = {0:0.1f}, ndf = {1}'.format(redchisqr, ndf), transform = ax1.transAxes, fontsize=14)

# plotting residuals
ax2 = fig.add_subplot(gs[1])
ax2.errorbar(t, resids, yerr = dv, ecolor = 'black', fmt ='ro', markersize = 2)
ax2.axhline(color = 'gray', zorder = -1)
ax2.set_xlabel('time (s)')
ax2.set_ylabel('residuals (V)')
plt.savefig('ChargingProcessDataAndFit.pdf')
plt.show()
```

Code based on the examples in [10].

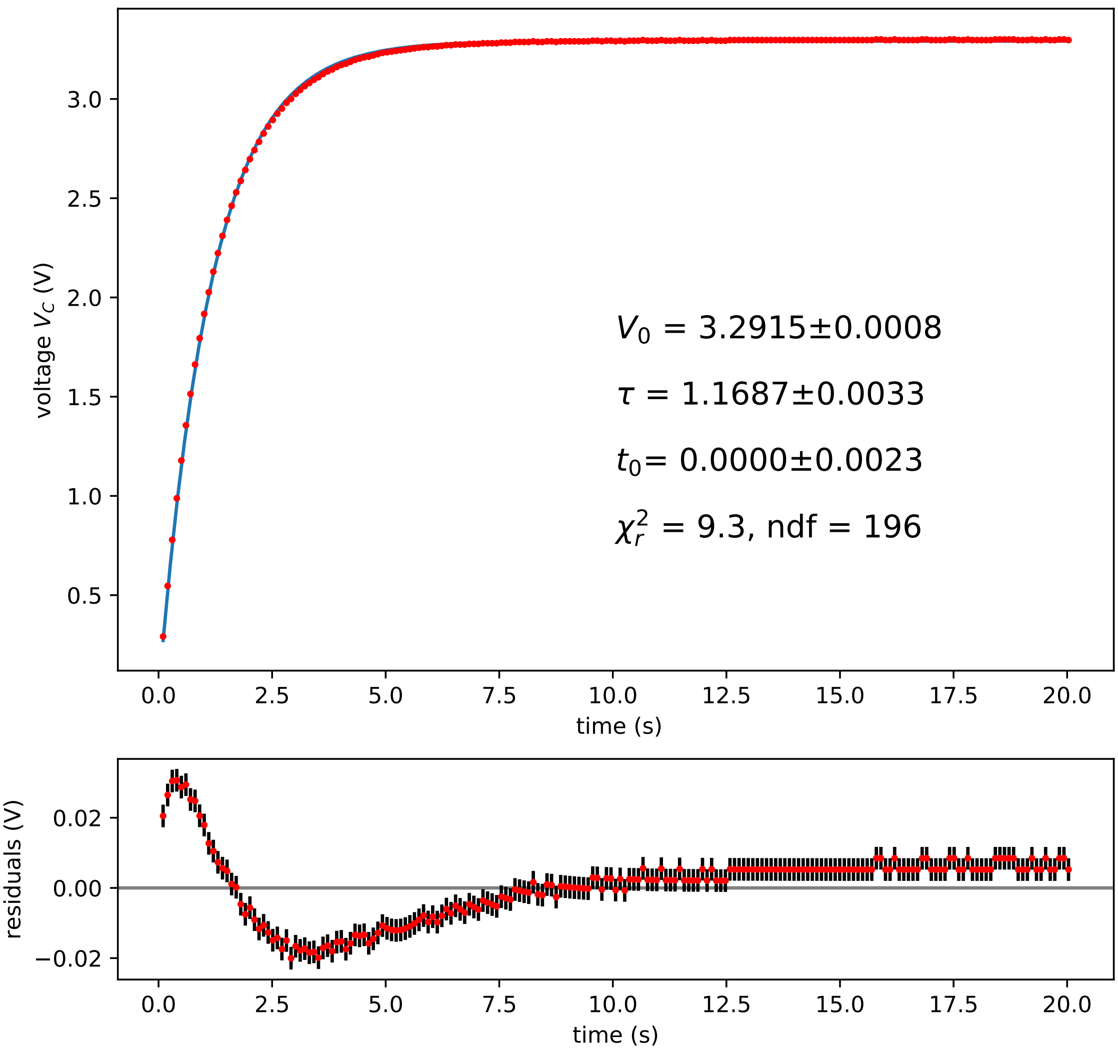

Figure S.5: *Upper panel.* Experimental values (dots) of the voltage on the capacitor in an RC circuit during the charging process measured as a fuction of time and best fit (solid line). *Lower panel.* Plot of the residuals (experimental value *minus* computed value) versus time. Error bars are plotted assuming that the uncertainty on the voltage measurements is equal to the voltage resolution.

5.2 Curve fitting: discharging process

Listing 6: **curveFit_discharging.py**

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
import scipy.optimize

Vdc = 3.293551538837322212
res = 3.3 / 1023
inputFile = 'DischargeProcessData.txt'

# define fitting function
def DischargeProcess(t, t0, tau):
    return Vdc * (np.exp(-(t-t0)/tau))

# read the experimental data from file
t, v = np.loadtxt(inputFile, delimiter = ' ', unpack = True)

# assign the experimental uncertainty on Voltage values
dv = np.ones(v.size) * res

# initial guesses for fitting parameters
t0_guess, tau_guess = 20, 1

# fitting procedure
nlfit, nlpcov = \
        scipy.optimize.curve_fit(DischargeProcess, t, v, \
                                p0 = [t0_guess, tau_guess], sigma = dv,\
                                bounds = (0, 100))

# obtaining parameters from the best fit procedure
t0, tau = nlfit

# obtaining uncertainties associated with fitting parameters
dt0, dtau = \
      [np.sqrt(nlpcov[j, j]) for j in range(nlfit.size)]

# create fitting function from fitted parameters
t_fit = np.linspace(t.min(), t.max(), 128)
v_fit = DischargeProcess(t_fit, t0, tau)

# residuals and reduced chi squared
resids = v - DischargeProcess(t, t0, tau)
redchisqr = ((resids/dv)**2).sum()/float(t.size-2)
ndf = t.size - 2
# where 2 is the number of free parameters

# create figure window to plot data
fig = plt.figure(1, figsize = (8,8))
gs = gridspec.GridSpec(2, 1, height_ratios = [6, 2])

# plotting data and fit
ax1 = fig.add_subplot(gs[0])
ax1.plot(t_fit, v_fit)
ax1.errorbar(t, v, yerr = dv, fmt = 'or', ecolor = 'black', markersize =
    2)
ax1.set_xlabel(' time (s)')
ax1.set_ylabel('voltage $V_C$ (V)')
```

```
ax1.text(0.6, 0.80, r'$\tau$ = {0:6.4f}$\pm${1:0.4f}'.format(tau, dtau),
    transform = ax1.transAxes, fontsize = 14)
ax1.text(0.6, 0.70, r'$t_0$= {0:5.4f}$\pm${1:0.4f}'.format(t0, dt0),
    transform = ax1.transAxes, fontsize = 14)
ax1.text(0.6, 0.60, r'$\chi_r^2$ = {0:0.1f}, ndf = {1}'.format(redchisqr,
    ndf), transform = ax1.transAxes, fontsize = 14)

# plotting residuals
ax2 = fig.add_subplot(gs[1])
ax2.errorbar(t, resids, yerr = dv, ecolor = 'black', fmt ='ro', markersize
     = 2)
ax2.axhline(color = 'gray', zorder = -1)
ax2.set_xlabel('time (s)')
ax2.set_ylabel('residuals (V)')
plt.savefig('DischargingDataAndFit.pdf')
plt.show()
```

Code based on the examples in [10].

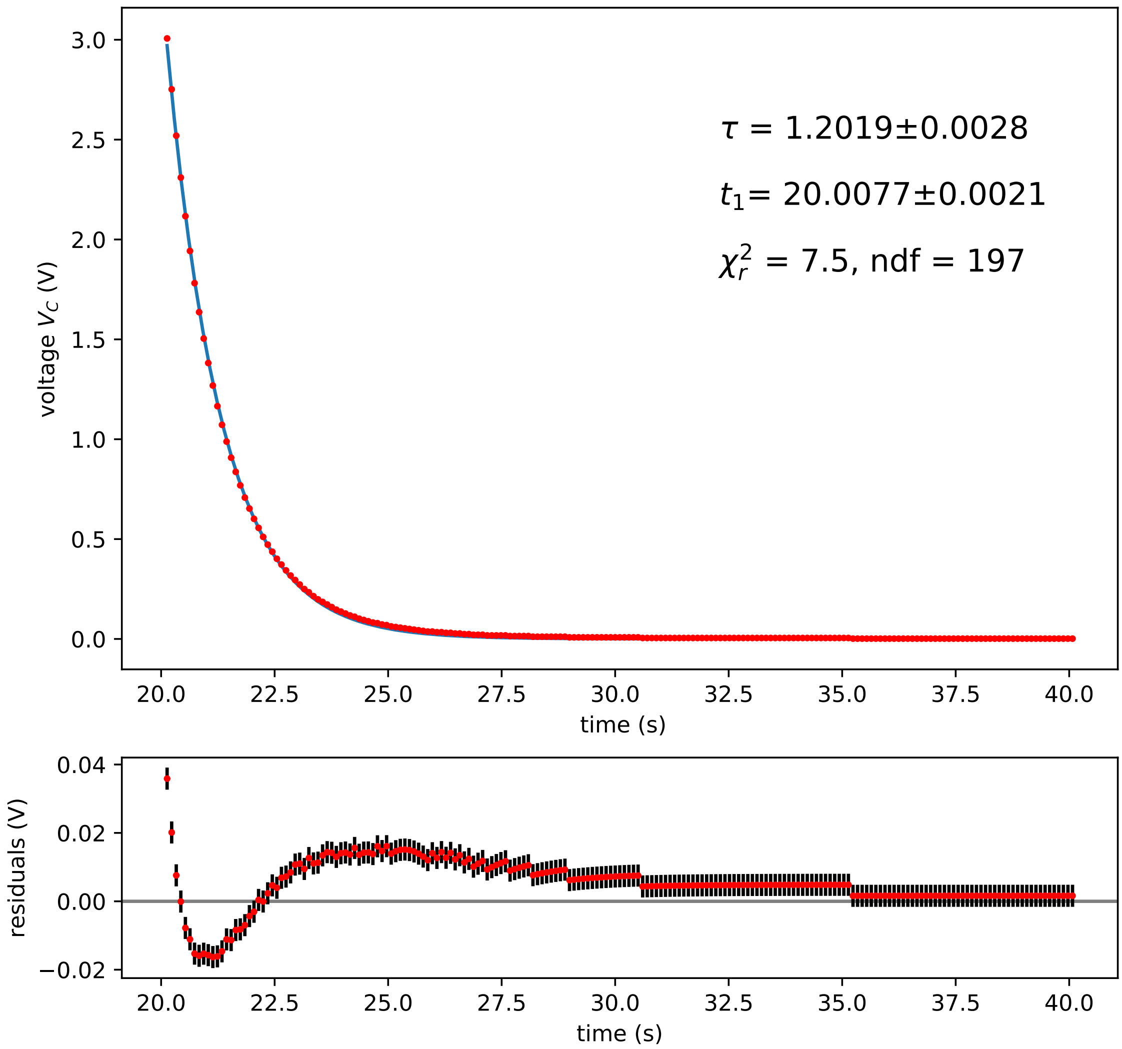

Figure S.6: *Upper panel.* Voltage on the capacitor in an RC circuit during the discharging. The solid line represents the best fit of the model equation $V_C(t) = V_0 \exp[-(t-t_1)/\tau]$ to the experimental data. *Lower panel.* Plot of the residuals.

Listing 7: **plot_voltage_and_fit.py**

```python
import numpy as np
import matplotlib.pyplot as plt

# define fitting function
def ChargeProcess(t, V0, t0, tau):
    return V0 * (1-np.exp(-(t-t0)/tau))

# define fitting function
def DischargeProcess(t, V0, t0, tau):
    return V0 * (np.exp(-(t-t0)/tau))


# read data from file
t, v = np.loadtxt('voltageLog.txt', delimiter=' ', unpack = True)

# plot of the experimental data
plt.plot(t, v, 'o', color='blue', markersize = 3)
plt.axhline(color = 'gray', zorder = -1)
plt.axvline(color = 'gray', zorder = -1)
plt.xlabel('time  $t$ (s)')
plt.ylabel('voltage $V_C$ (V)')
plt.text(27, 1.8, 'R = 10 k$\Omega$'+'\n'+'C = 100 $\mu$F')

# model curve for the charge process
t_fit_charge = np.linspace(0, 20, 128)
V0, t0, tau = 3.2915, 0, 1.1687
v_fit_charge = ChargeProcess(t_fit_charge, V0, t0, tau)
plt.plot(t_fit_charge, v_fit_charge, color='green')
#plt.draw()


# model curve for the discharge process
t_fit_discharge = np.linspace(20.01, 40, 128)
V0, t0, tau = 3.3, 20.0053, 1.2019
v_fit_discharge = DischargeProcess(t_fit_discharge, V0, t0, tau)
plt.plot(t_fit_discharge, v_fit_discharge, color='green')
#plt.draw()


plt.savefig('plot_voltage_and_fit.pdf')
#plt.draw()
plt.show()
```

5.3 Semilog plots

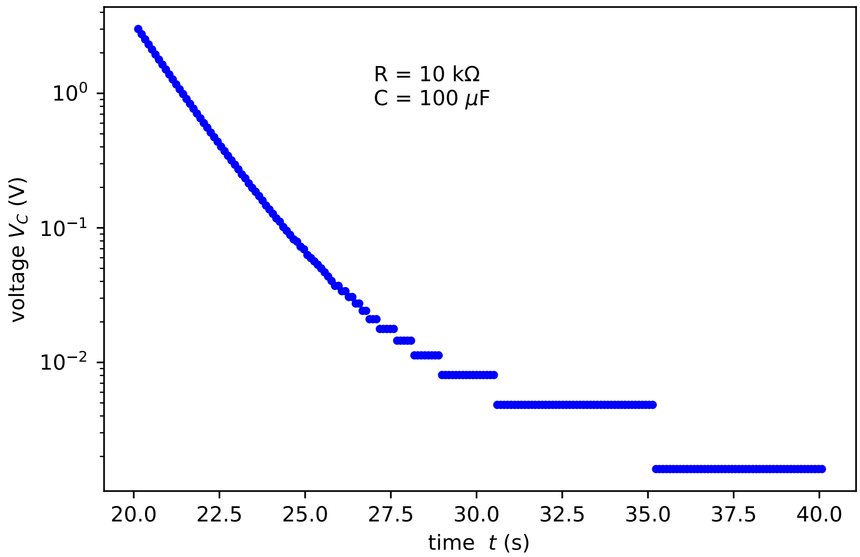

Figure S.7: Semilogarithmic plot of the voltage on the capacitor in the RC circuit studied while discharging the capacitor.

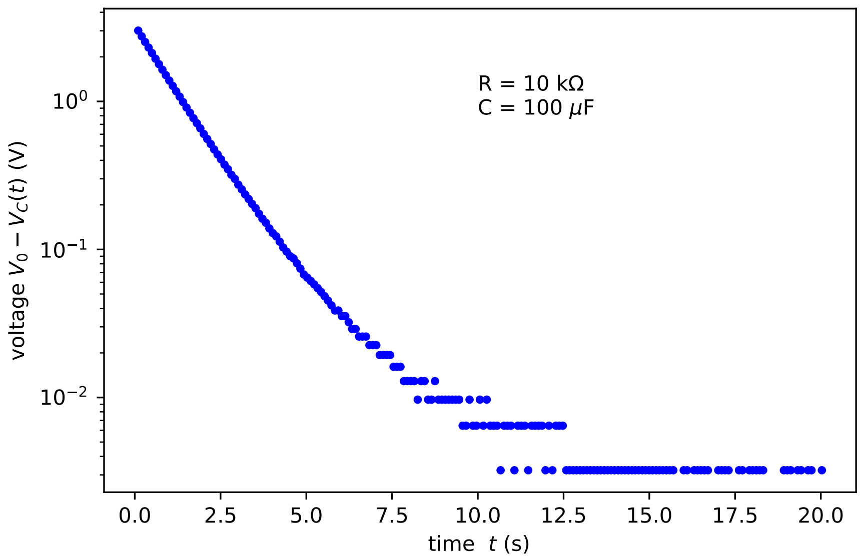

Figure S.8: Semilogarithmic plot of the difference $[V_0 - V_C(t)]$ versus time during the charging of the capacitor in the RC circuit studied.

6 Insights

6.1 Measurement rate

Listing 8: **rate_analysis.py**

```
import numpy as np
import matplotlib.pyplot as plt

inputDataFile = 'ChargeProcessDataZeroSleepOneSecond.txt'
outputDataFile = 'timeLapsesChargeProcessOneSecond.txt'
outParFile = 'ChargeDeltatParamsOneSecond.txt'

# read data from the file
t = np.loadtxt(inputDataFile, delimiter = ' ', usecols=(0), unpack = True)

# compute time lapses
delta_t = np.diff(t)
t_prime = t[:-1] + (delta_t/2)

# save the results as text
np.set_printoptions(precision = 20)
np.savetxt(outputDataFile, np.column_stack((t_prime, delta_t)))

# statistical analysis on time lapses
dt_mean = np.mean(delta_t, dtype = np.float64)
dt_min = np.min(delta_t)
dt_max = np.max(delta_t)
dt_std = np.std(delta_t, dtype = np.float64)

print('Delta t mean value = ', dt_mean)
print('Delta t minimum = ', dt_min)
print('Delta t mean value = ', dt_mean)
print('Delta t maximum = ', dt_max)

# write the results of the statistical analysis to a text file
fout = open(outParFile, 'w')
fout.write('Statistical analysis on times from the file: \n')
fout.write(outputDataFile + '\n' + '\n')
fout.write('Delta_t mean value = ' + str(dt_mean) + '\n')
fout.write('Delta_t minimum = ' + str(dt_min) + '\n')
fout.write('Delta_t maximum = ' + str(dt_max) + '\n')
fout.write('Delta_t standard deviation = ' + str(dt_std))
fout.close()


# plot of Delta_t vs- t
plt.plot(t_prime, delta_t*1e6, 'o', color = 'blue', markersize = 1)
plt.xlabel('time $t$ (s)')
plt.ylabel('time lapses $\Delta t$ ($\mu$s)')
plt.tight_layout()
plt.ylim(140, 360)
# save the plot to a file
plt.savefig('rate_analysis_zeroSleepOneSecond.pdf')
#plt.draw()
plt.show()
```

The statistical analysis of a set of results, chosen as example, suggests that the minimum time lapse is of 161 μs, the average value is 167 μs with a standard deviation of 9 μs. The values of the time lapses as a function of time are shown as a scatter plot in the following figure.

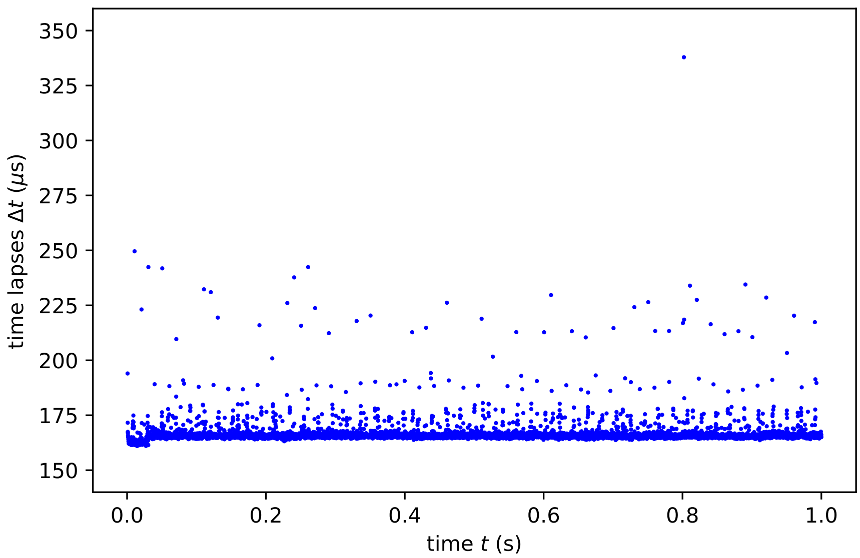

Figure S.9: Time lapses between two subsequent voltage measurements obtained if the code `measure_RCtransients.py` is modified commenting the `sleep()` command. In this condition the mean value of time lapses is approximately 170 µs.

6.2 Acquisition based on the spidev module

Listing 9: **measure_voltage_SPI.py**

```
from gpiozero import LED
from time import sleep, perf_counter
import spidev
import numpy as np

power = LED(17)

# open SPI bus
spi = spidev.SpiDev()
spi.open(0,0)
spi.max_speed_hz = 1000000

# read the data corresponding to a selected channel
# of the ADC MCP3008
def ReadChannel(channel):
    adc = spi.xfer2([1, (8 + channel)<<4, 0])
    data = ((adc[1]&3) << 8) + adc[2]
    return data

# convert data to voltage
def ConvertVolts(data):
    volts = (data * 3.3)/float(1023)
    return volts

# define channel
voltage_channel = 0

# function to perform the measurements
def measurement(time_limit):
    while True:
        # read the time
        time = perf_counter() - start_time
        times.append(time)
        # read the voltage
        voltage_level = ReadChannel(voltage_channel)
        voltage_value = ConvertVolts(voltage_level)
        voltages.append(voltage_value)
        if time > time_limit:
            break
        sleep(sleep_time)


sleep_time = 0.1
exec_time = 40
v_high_time = 20
outputFile = 'VoltageSPI-Log.txt'

times = [ ]
voltages = [ ]

start_time = perf_counter()

# charging the capacitor
t_limit = v_high_time
power.on()
measurement(t_limit)

```

```
# discharging the capacitor
t_limit = exec_time
power.off()
measurement(t_limit)


t = np.array(times)
v = np.array(voltages)

np.set_printoptions(precision = 20)
np.savetxt(outputFile, np.column_stack((t, v)))
```

Code based on the example in [7].

The statistic analysis on the time lapses from a sample data set indicates that the minimum time lapse is of about 50 μs, the average value is 54 μs, and the standard deviation is 12 μs. See the scatter plot of the data in Fig.S.10 as well as an histogram of the time lapses concerning a larger data set in Fig.S.11. For comparison, considering that the MCP3008 could work at an acquisition rate of 75 ksps (when $V_{DC} = 2.7$ V) [11], this would correspond to time lapses of 13.3 μs.

Summarizing, the acquisition rate depends not only on the sampling rate of the ADC, but also on the software used for controlling the MCP3008 chip through the GPIO. Aiming to achieve better performances is convenient to exploit the **spidev** library rather than the **gpiozero** module. Comparative tests on all the experiments carried out using the two different modules can be instructive. In particular, it can be verified whether the minimum detectable voltage corresponds to zero. Moreover, it should be considered that Python on RPi runs under a multitasking operating system so that other processes may gain priority over the CPU while the acquisition managed through the Python code would be slowed down. Faster execution and better timing could be achieved using C codes [12], [13]. However, Python is simpler if compared to C, it is readily available on the RPi, and the acquisition rate achieved with the settings described in this work can be suitable to carry out educational projects.

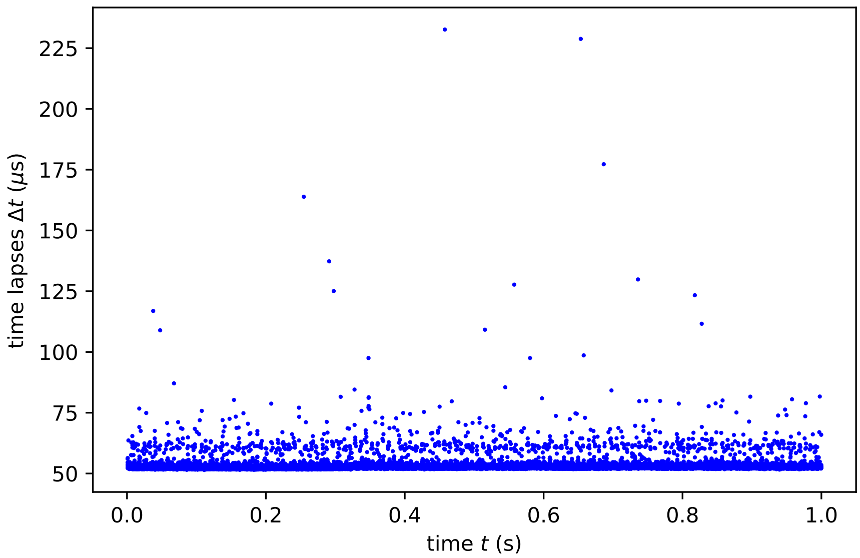

Figure S.10: Time lapses between two subsequent voltage measurements obtained when the code `measure_voltage_SPI.py` is used for the experiment, commenting the `sleep()` instruction. The mean value of the time lapses is of about 55 µs. The distribution of the time lapses acquired during a longer time interval is shown in Fig.S.11 as a histogram.

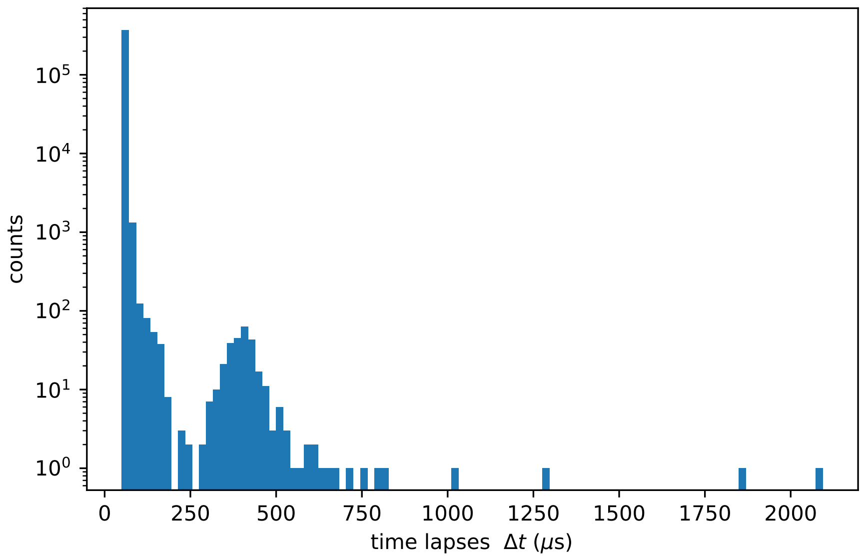

Figure S.11: Distribution of the time lapses Δt during the voltage measurements performed with the code `measure_voltage_SPI.py` where the `sleep()` instruction is commented (371737 data points, 100 bins). The data are acquired in a time interval of 20 s.

Listing 10: **histoTimeLapsesCharging.py**

```
import numpy as np
import matplotlib.pyplot as plt

DataFile='timeLapsesChargeProcess.txt'
OutputFile='histoLapsesCharging.pdf'
bins=100

# read data from file
t, Delta_t = np.loadtxt(DataFile, delimiter=' ', unpack = True)
# plot
plt.hist(Delta_t*1e6, bins, log = True)
plt.xlabel('time lapses  $\Delta t$ ($\mu$s)')
plt.ylabel('counts')
plt.tight_layout()
plt.savefig(OutputFile)
plt.show()
```

6.3 Optimized acquisition rate

Since the voltage varies rapidly when charging and discharging begin, it can be useful to impose a faster acquisition rate during the early steps of the process. To this aim the following code can be used for the measurements.

Listing 11: **log_spacedTimes.py**

```
from gpiozero import LED
from time import sleep, perf_counter
import spidev
import numpy as np

power = LED(17)

# open SPI bus
spi = spidev.SpiDev()
spi.open(0,0)
spi.max_speed_hz = 1000000

# read the data corresponding to a selected channel
# of the ADC MCP3008
def ReadChannel(channel):
    adc = spi.xfer2([1, (8 + channel)<<4, 0])
    data = ((adc[1]&3) << 8) + adc[2]
    return data

# convert data to voltage
def ConvertVolts(data):
    volts = (data * 3.3)/float(1023)
    return volts

# define channel
voltage_channel = 0

# function to perform the measurements
def measurement():
    for time_lapse in lapses:
        # read the time
        time = perf_counter() - start_time
        times.append(time)
        # read the voltage
        voltage_level = ReadChannel(voltage_channel)
        voltage_value = ConvertVolts(voltage_level)
        voltages.append(voltage_value)
        #
        sleep(time_lapse)


times = [ ]
voltages = [ ]

outputFile = 'VoltageSPI-LogSpaced.txt'
Delta_t_min = 50e-6
v_high_time = 20
points = 1000

start = np.log10(Delta_t_min)
stop = np.log10(v_high_time)
spots = np.logspace(start, stop, points)
lapses = np.diff(spots)

```

```
start_time = perf_counter()

# charging the capacitor
power.on()
measurement()

# discharging the capacitor
power.off()
measurement()


t = np.array(times)
v = np.array(voltages)

np.set_printoptions(precision = 20)
np.savetxt(outputFile, np.column_stack((t, v)))
```

References and Notes

[1] (), [Online]. Available: https://thepihut.com/blogs/raspberry-pi-tutorials/27968772-turning-on-an-led-with-your-raspberry-pis-gpio-pins.

[2] (), [Online]. Available: https://www.raspberrypi.org/documentation/hardware/raspberrypi/power/.

[3] (), [Online]. Available: https://www.raspberrypi.org/documentation/hardware/raspberrypi/gpio/.

[4] (), [Online]. Available: www.raspberrypi.org/documentation/hardware/raspberrypi/gpio/gpio_pads_control.md.

[5] B. Nuttall. (), [Online]. Available: https://gpiozero.readthedocs.io/en/stable/index.html. See also
https://www.raspberrypi.org/blog/gpio-zero-v1-5/.

[6] (), [Online]. Available: https://projects.raspberrypi.org/en/projects/physical-computing/3.

[7] (), [Online]. Available: https://www.raspberrypi-spy.co.uk/2013/10/analogue-sensors-on-the-raspberry-pi-using-an-mcp3008/.

[8] (), [Online]. Available: https://gpiozero.readthedocs.io/en/stable/api$%5C_$spi.html.

[9] (), [Online]. Available: https://projects.raspberrypi.org/en/projects/physical-computing/13.

[10] D. J. Pine, *Introduction to Python for Science and Engineering*. CRC Press, 2018, ISBN: 9781138583894.

[11] MCP3004/3008 datasheets, 2008, Microchip Technology Inc.

[12] (), [Online]. Available: https://pypi.org/project/RPi.GPIO/.

[13] (), [Online]. Available: https://www.circuits.dk/everything-about-raspberry-gpio/.