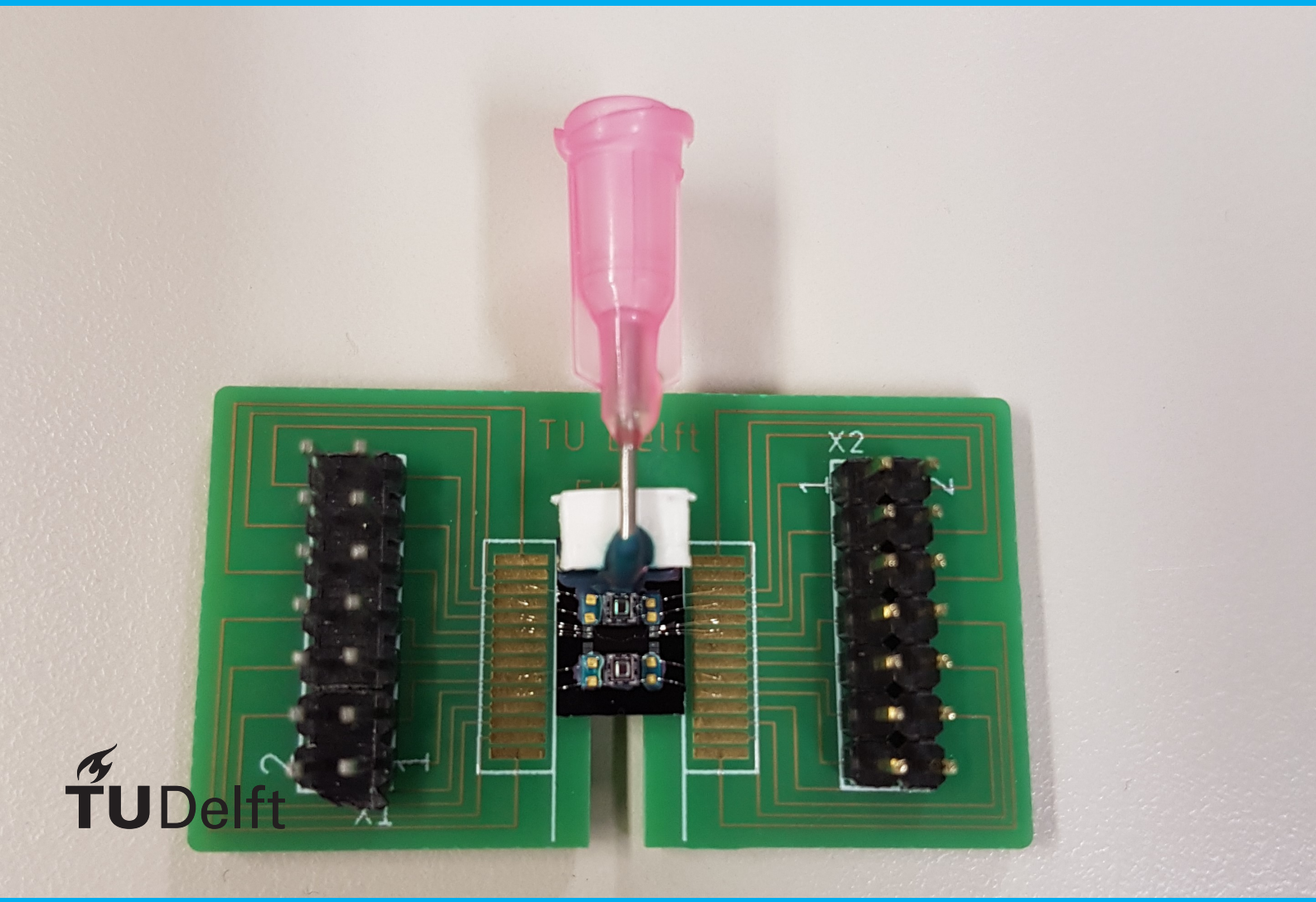
Control and data-acquisition of a heater/temperature sensor

For performance evaluation of vaporizing liquid micro-thrusters

G. Breysens
Marjolein Rebers

Version 1, June 2019



TUDelft

Control and data-acquisition of a heater/temperature sensor

For performance evaluation of vaporizing liquid micro-thrusters

by

G. Breysens
Marjolein Rebers

For the bachelor graduation project
at Delft University of Technology.
Figure on titlepage: thruster with water inlet and pressure sensors

Supervisor/Project Proposer	Henk van Zeijl
Supervisor	Brahim el Mansouri
Chairman	Jaap Hoekstra
Jury	Borbála Hunyadi
Jury	Anton Montagne

TU Delft

Abstract

The goal of this bachelor thesis is to develop a control system that controls the temperature of a microthruster. This system also needs to acquire data for research. The microthruster contains a resistor that is used both as a heater and as a sensor. To facilitate the acquisition of data and the testing of the control system, a lab setup with a Keithley 2450 SourceMeter power supply was used. LabVIEW was used to control the power supply and to execute the control algorithms. The final system consists of a microcontroller that runs the control algorithms based on proportional-integral-derivative (PID) control developed in this thesis. The PID values can be adapted with use of the graphical user interface (GUI). A read-out circuit and current supply will be part of the integrated system. These circuits will be developed by other groups that are part of this bachelor project.

Preface

This thesis was written in context of the Bachelor Graduation Project. The project was proposed by dr. ing. Henk van Zeijl with the goal to design a data-acquisition/control system for the research of microthrusters. We would like to express our gratitude to our supervisors dr. ing. Henk van Zeijl and ir. Brahim el Mansouri, and the designer of the thrusters Alisher Kurmanbay. We would like to thank ir. Anton Montagne, dr.ir. Chris Verhoeven, dr. Angelo Cervone, ing. Xavier van Rijnsoever and dr. Riccardo Ferrari for their time and helpful advice. We would also like to thank dr.ir. Ger de Graaf and dr.ir. Chris Verhoeven for lending us equipment. Finally, we would like to thank Koen Lam, Mirza Mrahorović, Coen Straathof and Rijk van Wijk for cooperating in this project.

G. Breysens & Marjolein Rebers
Delft, June 2019

Acronyms and software

Acronyms

ADC	Analog to Digital Converter
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
CPHA	Clock PHAse
CPOL	Clock POLarity
DAC	Digital to Analog Converter
DMA	Direct Memory Access
GUI	Graphical User Interface
LSB	Least Significant Bit
MCU	Micro Controller Unit
MEMS	MicroElectroMechanical Systems
MSB	Most Significant Bit
OS	Overshoot
PID	Proportional Integral Derivative
RX	Received data
SCPI	Standard Commands for Programmable Instruments
SPI	Serial Peripheral Interface
SSE	Steady State Error
TX	Transmit data
UART	Universal Asynchronous Receiver Transmitter
UI	User Interface
VI	Virtual Instrument
VISA	Virtual Instrument Software Architecture
VLM	Vaporizing Liquid Thrusters

Software

The used software can be found in Table 1.

Table 1: Software that was used during this thesis.

NI LabVIEW 2018	systems engineering software for applications that require test, measurement, and control with rapid access to hardware and data insights
STM32CUBEIDE 1.0.0	C/C++ development platform with configuration, code generation, code compilation, and debug features for STM32 microcontrollers
MATLAB R2018B	numerical analysis environment
FLIR R&D software	thermal analysis software package for FLIR R&D cameras
DinoCapture 2.0	microscope imaging software for Windows

Contents

1 Introduction

Pico- and nanosatellites have increased in popularity over the years [1]. In 2018, 244 nanosatellites were launched and it is predicted that in 2023 already 703 nanosatellites will be launched [2]. Nanosatellites can be used in a wide variety of fields, such as space debris removal [3], formation flying [4] or swarm missions [5]. They are lightweight and low-cost, but usually lack propulsion capability. There is still much research to be done into the micro-propulsion system for these satellites. The need for further development of these systems is also mentioned in the technology roadmap set by NASA [6]. MEMS (micro electromechanical systems)can play an important role in the creation of micro-propulsion systems. With systems used in space it is very important to know that they work correctly because changes are not easily made, therefore it is important to have system that can be used for testing. The background information, state-of-the-art analysis, project objective and task division of project is the same for all the sub-parts of the project [7, 8].

Background information

Micro-propulsion systems have many different variations based on their working principle, ranging from solar sails, cold gas propulsion systems, electric propulsion systems and chemical propulsion systems [9]. These types have different efficiency and size. This project focuses on the resistojet thruster, which falls in the category of electric propulsion systems. There are in general two main types of microresistojets: the vaporizing liquid microthruster (VLM) and low-pressure microresistojet (LPM) [10]. In this project, we work with a VLM that is part of a research project [11].

A VLM generally consists of a resistive heating element and a liquid channel, as shown in Figure 1.1. Thrust is delivered by heating a gas or liquid, accelerating it through a nozzle and expelling it into space. In our case, the propellant that will be tested is water. When the water starts to boil after the resistor heats it up, bubbles appear inside the channel. These bubbles are a change in state of water and form a layer of thermal insulation and affect the heat transfer from the heater to the channel. This in turn affects the temperature of the resistive element and the pressure in the channel. Therefore, the amount of steam generated tends to oscillate without control. As the thrust is dependent on the mass flow rate [12], unwanted variations in the temperature and pressure can therefore lead to a fluctuating thrust. Furthermore, uncontrolled heating could even cause thermal runaway and this would destroy the device. Controlling the temperature of the heater element is thus of very high importance. The control can be done with feedback because the resistive heater is also a sensor.

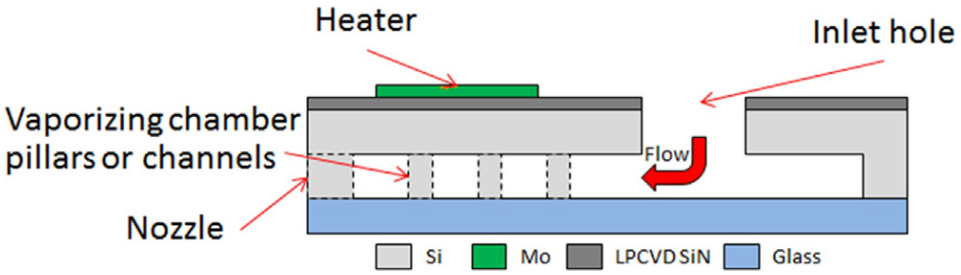


Figure 1.1: Schematic cross-section of the thruster taken from [12]

State-of-the-art analysis

Small satellites like CubeSats have several constraints that form a limitation in the design, e.g. its mass, dimensions, available power and propulsion [13]. Hence, there is a need for high performance, highly miniaturized and integrated micro-propulsion systems that meet these constraints [14].

Tummala and Dutta compared different micro-propulsion systems for CubeSats [15]. Their research shows that resistojets provide a relatively low specific impulse, have the highest average thrust-to-power ratio compared to other electric propulsion systems and can deliver a thrust between 0.1 mN and about 50 mN. This relatively low thrust can also be seen in the microresistojet propulsion concepts of Cervone, Zandbergen, Guerrieri, De Athayde Costa e Silva, Krusharev, and Zeijl [16]. In this work, Cervone, Zandbergen, Guerrieri, De Athayde Costa e Silva, Krusharev, and Zeijl state that the presented concepts have the most potential in nanosatellite applications where a thrust between 0.5 mN and 10 mN is required. Due to their relatively low thrust, resistojets are also used on larger satellites for attitude control [15].

When the thrust is controlled in magnitude and direction, it could make a big difference in the use of micro-propulsion systems [16].

A possible solution to control the thrust is by trying to reduce bubble forming. The transient behavior of bubble formation on micro-resistors is an active field of study [17]. Tsai and Lin have observed that the temperature of the heater rises with the increase of bubble size to reach a new equilibrium temperature, because it is believed that the heat dissipation path from the microresistor surface to the liquid is partially blocked by the vapor bubble. Several experiments have also been performed on microheaters to better understand the boiling process [18]. In this project an attempt will be made to control the temperature of the microheater and therefore the bubbling process can be researched.

Project objective

In an attempt to stabilize the temperature, we have defined the following goals for the project:

1. To acquire data about the temperature of the heating element during dynamic liquid/vapor phenomenon.

2. To control the temperature of the heating element in an effort to reduce bubble forming.

To achieve these goals, we implement the required hardware and software compatible with multiple resistojets. These resistojets have pressure sensors integrated that create additional information about the system.

The microthruster is supposed to work in space on nanosatellites. However, this project is part of a research project on manufacturing microthrusters. Therefore, we assume that our system is only going to work on earth as a first prototype. Phenomena such as space radiation, high pressure and extreme temperature variations will be omitted and constraints such as limited space, limited power and extreme robustness will not be given priority in the design. This simplifies the design while providing valuable information to reach the objectives of the project.

The project can be considered a success if the following deliverables are accomplished.

1. Design and implementation of circuitry for real-time data acquisition from a heater array and from multiple EPCOS pressure sensors and control of the resistance.

2. Design of front end data acquisition software.

3. Implementation of a control algorithm for temperature

The results obtained from this project enable researchers to further investigate the effect of the bubble forming in microthrusters.

Task division of project

In order to reach the objectives of the project, the project is divided into subgroups.

The temperature of the heater device is affected by supplying an electric input signal to the heater. The accuracy of this input signal determines the precision of the temperature which can be set. This is the challenge of the first subgroup: the supply group.

Implementing a feedback system poses some challenges, as there is no model available of the microthruster. Furthermore, effects such as the heat dissipation and stochastic bubble forming make modelling even more

complex. Nonetheless, the need for an adequate feedback system is of uttermost importance for controlling the temperature. This is the focus of another subgroup: the control group.

Providing reliable input to the feedback system means that the resistance of the heater element should be read out accurately. The temperature dependency makes this a challenging task. Therefore, another subgroup is formed which tackles this challenge: the read-out subgroup.

A high level overview of the system is given in Figure 1.2. To summarize, the following three subgroups are formed.

- Control: responsible for the control of the system.
- Read-out: responsible for the hardware read-out circuitry.
- Supply: responsible for the supply for the heater.

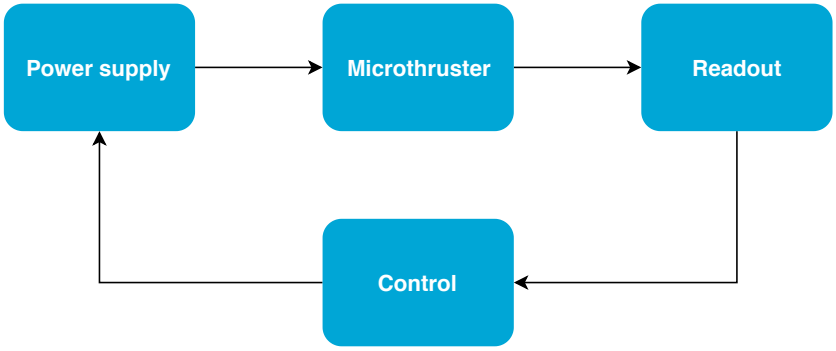


Figure 1.2: Overview of the subgroups

Technical analysis

Although the control of a microheater system has been done before [19]. This is a special case because the resistance is at the same time the heater and sensor. The control system should also be able to control thrusters with different resistances. Another reason why this project is special, is because the water in the thruster causes bubbles, which make the system more difficult to control.

Document structure

To get a better understanding of the thruster information about the theoretical model is first given. Then, the control type will be explained. Thereafter, the system with lab equipment will that is used for measurements will be elaborated on. With these measurements, a model is derived. At last the implementation of the system with use of the microcontroller will be explained and the test results will be reported.

2
Program of Requirements

In this chapter the requirements for the total system will be given, The requirements that are focused on by the control will be highlighted.

Program of requirements entire system

The system which that ought to be build should control the temperature of the heater element in a resistojet and provide temperature and pressure readings. The Key Performance Indicators (KPI) are listed below and identified whether they are a mandatory requirement (MR) or trade-off requirement (ToR).

1. Functional
   (a) The read-out must have an accuracy of +/- 1 °C. [MR]
   (b) The temperature must be within +/- 10 °C of the set-point. [MR]
   (c) Two EPCOS pressure sensors per truster need to be read out. [MR]
   (d) The resistor must be used as the heater. [MR]
   (e) The resistor must be used as the sensor. [MR]
   (f) The measurement needs to be a 4-probe measurement. [MR]
   (g) The system must have front-end data acquisition software. [MR]
   (h) The system should preferably read out an array of three thrusters. [ToR]
2. Ecological embedding in the environment
   (a) The maximum temperature should not exceed 300 °C. [MR]
   (b) A maximum voltage needs to be limited to 40 V. [MR]
3. System requirements
   (a) The circuit has to be portable. [MR]
   (b) The system should not use more than 10 W. [ToR]
   (c) The system should be able to be connected to different loads (i.e. thrusters) ranging from 100 to 1000 Ω. [ToR]
4. Development of manufacturing methodologies
   (a) The system should preferably be implemented on a PCB. [ToR]

Program of requirements Control subsystem

For the control subsystem the main focus is on requirement 1b, 1c, 1g, 1h,2a, 2b and 3c. Besides the requirements, there are assumptions made about the timing:

Assumptions

1. A sampling frequency of 100 Hz is enough to see the bubbling effect.

3
Theoretical model

In this chapter, more information about the theoretical model of the thruster will be given. First, the relation of the resistance to temperature will be examined. Secondly, the thermal dynamics will be studied.

3.1. Resistance

The resistance temperature coefficient α describes the relative change of the resistance R that is associated with a given change in temperature dT. This relation is defined by Equation 3.1.

$$\frac{dR}{R} = \alpha dT \tag{3.1}$$

Because the thruster is used in the temperature region in which the temperature coefficient does not change, the resistance can be calculated according to Equation 3.2 [20].

$$R(T) = R(T_0)[1 + \alpha(T - T_0)] \tag{3.2}$$

Where

- $R(T_0)$ = resistance at temperature T_0
- α = temperature coefficient of the resistance ($0.0024\,\mathrm{K}^{-1}$ for the thrusters)
- T = temperature

Because the resistance is linearly dependent on the temperature, the resistance can be used as a sensor. Since the resistance is dependent on the temperature, the VI characteristic becomes non-ohmic. When the resistor has a constant current as input, it heats up due to power dissipation. This results in a higher resistance and thus a higher voltage. Because the current stays constant, the power increases because the resistance increases as the thruster heats up. This is called the thermal runaway effect. This effect stops when the heating up and cooling down are in balance.
Given that the temperature needs to be controlled within a range of 10 °C and the resistance of the thruster is between 150-1000 Ω, the maximum deviation of the resistance can be calculated with Equation 3.3 to be 3.6 Ω.

$$dR = \alpha R(T_0) dT \tag{3.3}$$

3.2. Thermal dynamics

The heat transfer from a body to the ambient is proportional to the temperature difference between the body and the ambient as shown in Equation 3.4:

$$F = hA_s(T(t) - T_a) \tag{3.4}$$

where

- F = heat transfer
- h = heat transfer coefficient
- A_s = surface area

- $T(t)$ = body temperature at time t
- T_a = ambient temperature

When heat is lost to the ambient, the temperature drop in the body can be described by Equation 3.5:

$$\rho c_p V \frac{dT}{dt} = -F \tag{3.5}$$

where

- ρ = density
- c_p = specific heat
- V = body volume

The following equation can be derived using the thermal energy balance of the system [21]. Equating Equation 3.4 and Equation 3.5 for the heat transfer gives us Equation 3.6.

$$\rho c_p V \frac{dT}{dt} = -hA_s(T(t) - T_a) \tag{3.6}$$

This can be rewritten as

$$\frac{dT}{dt} + \frac{1}{\tau}T = \frac{1}{\tau}T_a \tag{3.7}$$

with

$$\tau = \frac{\rho c_p V}{hA_s}$$

This equation models a first-order LTI system. An electric equivalent with power input is described by Equation 3.8

$$C_T \frac{dT_T}{dt} = -\frac{1}{R_{Ta}}(T_T - T_a) + IV \tag{3.8}$$

where

- IV = power
- T_T = resistor temperature
- T_a = ambient temperature
- R_{Ta} = thermal resistance between environment and resistor
- C_T = thermal capacitance of resistor
- t = time

Rewriting using $\Delta T = T_T - T_a$ and using the fact that T_a is approximately constant leads to the following equation.

$$R_{Ta}C_T \frac{d\Delta T}{dt} + \Delta T = R_{Ta} IV \tag{3.9}$$

The above equation shows that the system dynamics are described by a first-order differential equation with time constant $\tau = R_{Ta}C_T$. The system shows the behaviour of a low-pass filter. This time constant can be determined from measurements.

Although the first-order approximation can work, the real system is more complicated because of environmental effects such as the heat capacity of water and the casing of the thruster, and the phase change when heating. This makes the system non-linear. To take into account more effects, a second-order model provides a more accurate description.

4 Control

In this chapter, PID control will be explained. This is a closed-loop negative feedback control method. PID control was chosen instead of bang-bang control, because it is more efficient due to less power use and because it is more accurate and gives less oscillations.

4.1. PID control

PID control is a commonly used control strategy. The PID equation can be expressed in the time-domain as follows [22]:

$$u(t) = K_p e(t) + K_i \int_0^t e(t) + K_d \frac{de(t)}{dt} \tag{4.1}$$

where

- $u(t)$ = output of the controller
- $e(t)$ = error at time t
- K_p, K_i and K_d are the proportional, integral and derivative constants respectively

Equation 4.1 is represented in the Laplace domain as follows.

$$C(s) = \frac{U(s)}{E(s)} = K_p + \frac{K_i}{s} + K_d s$$

The controller will be digital because digital controllers are insensitive to environmental factors such as temperature, they are flexible and it makes it easy to implement multiple functions.
To find the discrete form of PID control, the first-order bilinear transform $s = \frac{2}{T}\frac{1-z^{-1}}{1+z^{-1}}$ is used [23]. This results in Equation 4.2.

$$u[k] = ae[k] + be[k-1] + ce[k-2] + u[k-1] \tag{4.2}$$

- $u[k]$ = signal
- $e[k]$ = error
- $a = K_p + \frac{T_s K_i}{2} + \frac{K_d}{T_s}$
- $b = -K_p + \frac{T_s K_i}{2} - \frac{K_d}{T_s}$
- $c = \frac{K_d}{T_s}$
- T_s = sample time

Table 4.1 shows the effects that increasing the PID constants has on the rise time, settling time, overshoot, and steady-state error.

Table 4.1: Performance effects of tuning. The parameters that are effected are the rise time T_r, the settling time T_s, the overshoot OS, and the steady-state error SSE.

	Effect on performance			
	T_r	T_s	OS	SSE
K_p	Decrease	Small change	Increase	Decrease
K_i	Decrease	Increase	Increase	Eliminate
K_d	Small change	Decrease	Decrease	Small change

4.2. Control loop

Figure 4.1 shows a typical control system where

- r = setpoint
- e = error
- y = output
- y_m = measured output

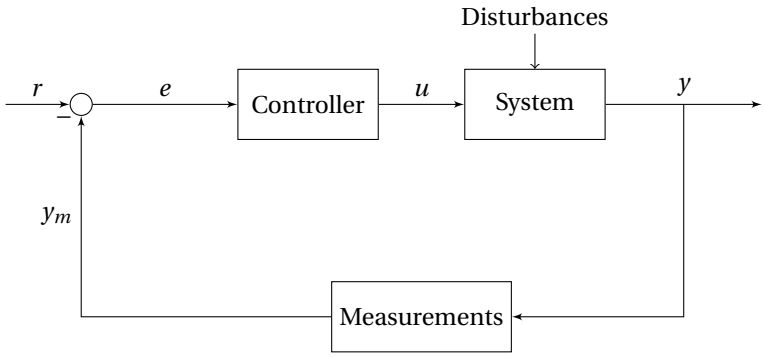


Figure 4.1: Generic control system

The control system has a temperature setpoint, which is converted to an equivalent resistance setpoint. The resistance control works with a PI controller. A derivative term is not used because it makes the system more sensitive to noise and stochastic effects, such as the bubbling effect when the water is boiled.
Two options exist to control the resistance. The PID controller could either control current or power. Because a model has been made with the relation of power to the temperature/resistance, power control is preferred. When the PI controller controls the amount of power that the system will deliver, this power needs to be converted to an equivalent current output. To convert the power to current, the voltage and current measurements and Equation 4.3 are used, the new resistance of the thruster can be calculated and fed back to form a negative feedback loop. Figure 4.2 shows a block diagram of the temperature to resistance conversion, the control loop and the heater with measurements.

$$I_{\text{supply}} = \sqrt{\frac{P_{\text{sp}} V_{\text{meas}}}{I_{\text{meas}}}} \tag{4.3}$$

Where:

- I_{supply} = send current to the supply
- P_{sp} = power setpoint
- V_{meas} = measured voltage
- I_{meas} = measured current

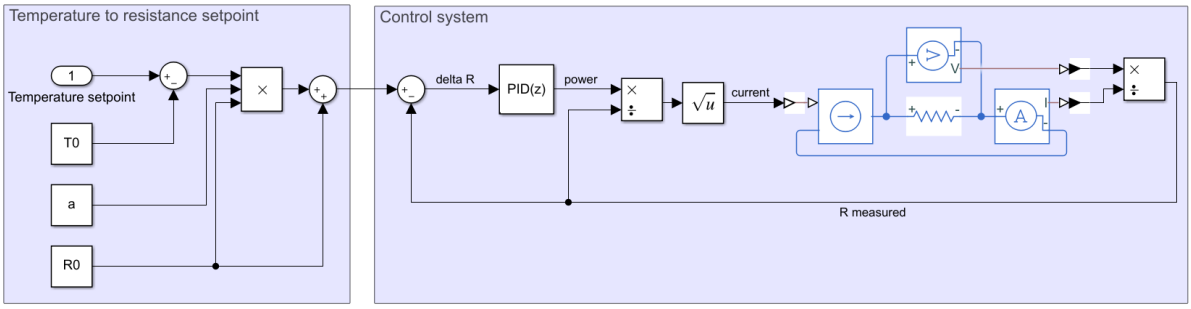


Figure 4.2: Temperature control using power

5
System with lab equipment

To get more information about the thruster without having the final system ready, a measurement setup with lab equipment was made. The measurements can be used to make a model of the thruster and to implement a control system. In this chapter, the setup is explained.

5.1. Setup

An overview of the set-up can be seen in Figure 5.1. For the test setup, the Keithley 2450 SourceMeter was

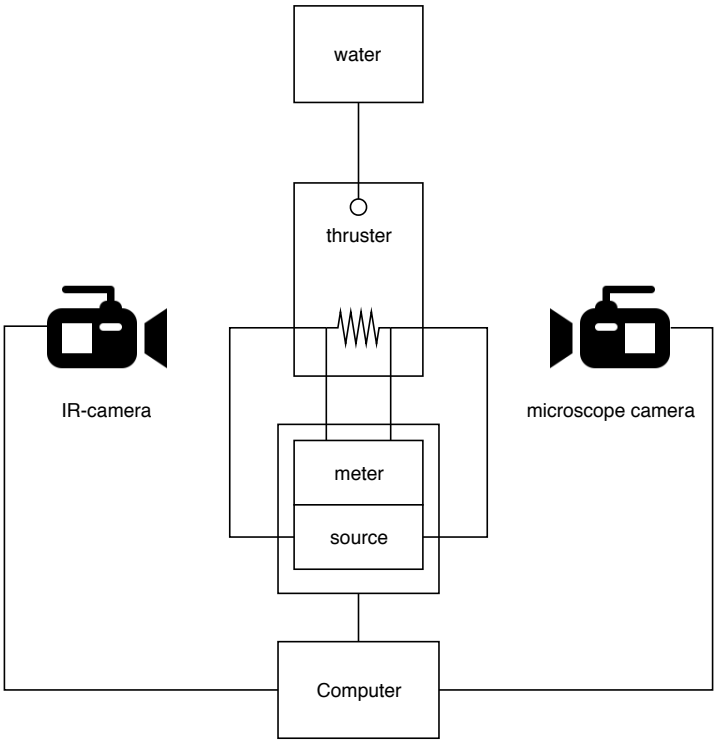


Figure 5.1: Set-up with lab equipment

used . The Keithley is useful because it has two source ports and two sense ports. This means it is suited for a 4-wire measurement. The Keithley can also deliver power up to 20 W. The power envelope of the Keithley describes that a current amplitude of 1 A can be achieved when the amplitude of the voltage is smaller than 20 V. A voltage amplitude of 200 V can be achieved when the amplitude of the current is smaller than 100 mA. The maximum amount of samples per second is 3000 for a 4.5-digit resolution measurement, and 59 for a 6.5-digit resolution measurement. This is not feasible when the measured data is sent to a remote interface [24].
For the SCPI (standard commands for programmable instruments) commands, a USB cable is used (this cable could be replaced with a GPIB cable). The information about the commands was found in the reference manual [25]. An Ethernet cable is also connected for the use of the virtual front panel as shown in Figure 5.2.

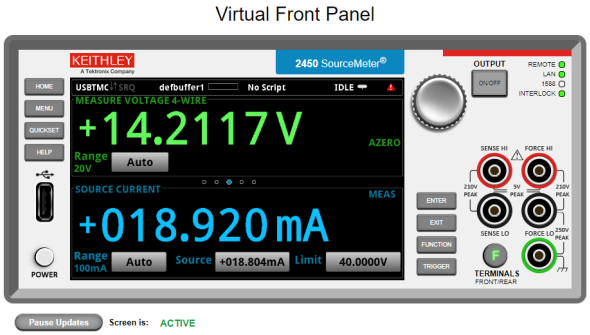


Figure 5.2: Virtual Front Panel

Microscope camera The DinoCapture microscope camera is used to observe the bubbling process in the thruster. An image from the camera can be seen in Figure 5.3. The figure shows the heating resistor surrounded by water that starts to boil. On the left side of the picture, the nozzle can be seen.

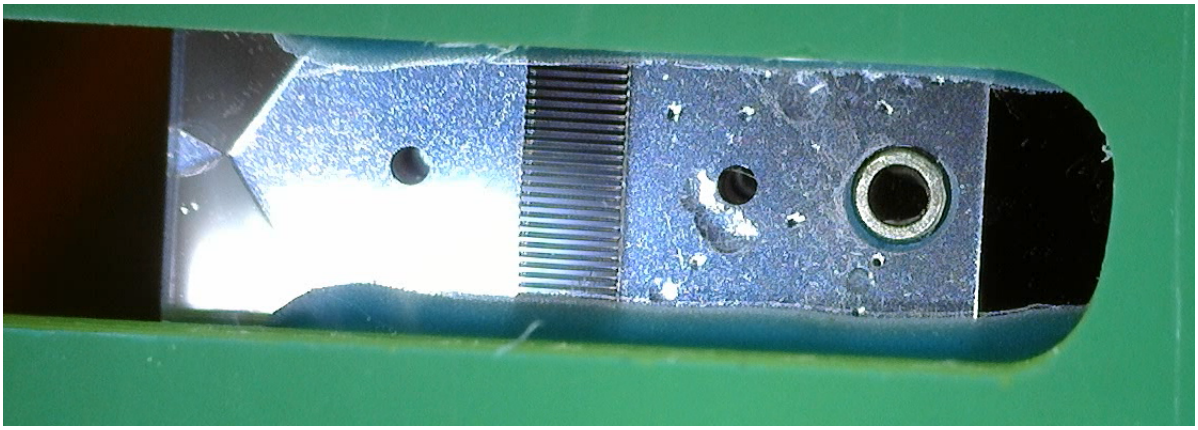


Figure 5.3: Bubbling process in microthruster

IR-camera The FLIR IR-camera is used to measure the temperature of the thruster. This camera will be used to verify the temperature control. We used the FLIR SC305 to perform thermal imaging. The SC305 can supply 16-bit 320 x 240 images at a maximum rate of 60 Hz via an Ethernet connection. Images from the camera can be seen in Appendix A.

5.2. LabVIEW

Multiple LabVIEW VIs (virtual instruments) were made to communicate with the Keithley. The VIs have a GUI and can be exported as an .exe file. All programs work in a similar manner. After initializing the communication, the commands are sent via VISA (virtual instrument software architecture). First, a start current is sent. Next, a loop starts. In this loop the measurement data is acquired and appended to a .csv file. What happens in the loop is dependent on the chosen functionality. The result is a measurement that has a frequency of approximately 10 Hz. This is much slower than of the Keithley itself. The main delay is in the communication with LabVIEW.

Power control The user can input a power setpoint in the GUI. LabVIEW calculates the necessary current to deliver to the thruster to reach this power by using the measured resistance.

Resistance control For resistance control, a PI controller is used to control the power, which is then converted to a current setpoint. The PI controller is implemented using Equation 4.2.

Figure 5.4 shows the power and resistance control GUIs in LabVIEW. The saved data can be plotted by MATLAB. The code that was used for this can be found in Subsection C.2.1.

(a) Power control GUI

(b) Resistance control GUI

Figure 5.4: LabVIEW GUIs

6
Measured model

In this chapter, a model is made from the measurements that were acquired using the Keithley 2450 SourceMeter.

6.1. System model

A system model has been made for a thruster with resistance of 188 Ω.

Dry system

To derive a model of the thermal dynamics, the step response data of the system is recorded where the input is a power setpoint and the resistance response is measured. A first-order transfer function is shown below, where K is the DC gain and τ is the time constant of the system.

$$P(s) = \frac{K}{\tau s + 1} \qquad \left[\mathrm{A}^{-2}\right] \tag{6.1}$$

To determine this transfer function, the step response of the system has to be measured.

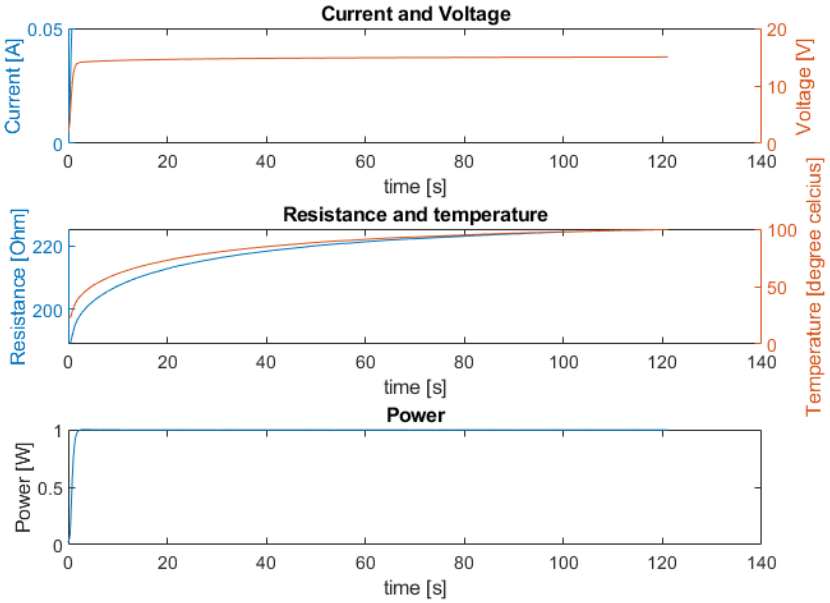


Figure 6.1: Dry system response for 1 W input with $R_0 = 188\,\Omega$

We will consider the output of the system to be the deviation ΔR from the resistance at room temperature. The time constant is the time it takes for the system to achieve 63% of its total change. Figure 6.2 shows only the resistance response of the system for a 1 W step input.

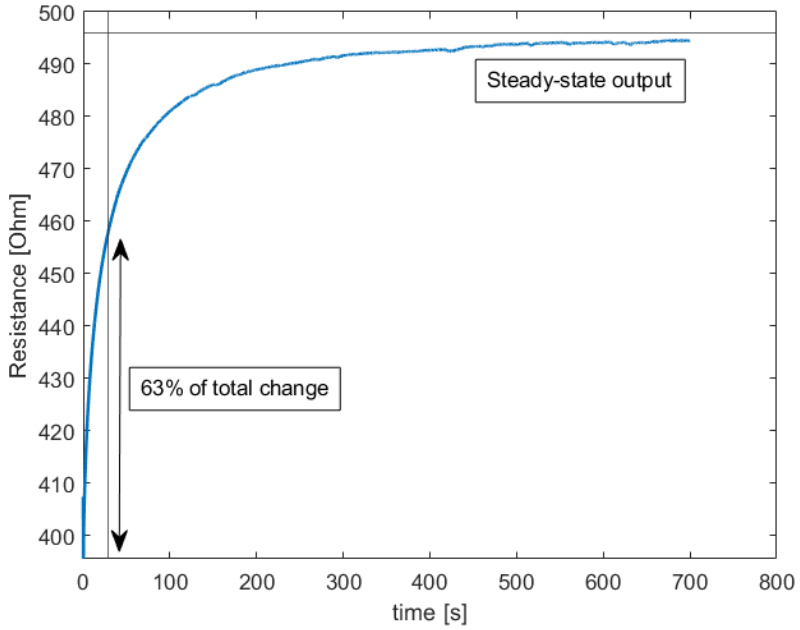


Figure 6.2: First-order model derivation

Using this response, we find that the DC gain is $K = \frac{496\,\Omega - 395.4902\,\Omega}{1\,\mathrm{W}} = 100.5098\ \mathrm{A}^{-2}$ and $\tau = 28.8105\,\mathrm{s}$. Therefore, the system is described by the following transfer function.

$$P(s) = \frac{100.5098}{28.8105s + 1} \qquad \left[\mathrm{A}^{-2}\right] \tag{6.2}$$

Figure 6.3 compares the measurements with the first-order model. The model does not provide a good fit (55.37 %). The reason for this is that the first-order model does not take into account the leakage effects to the environment.

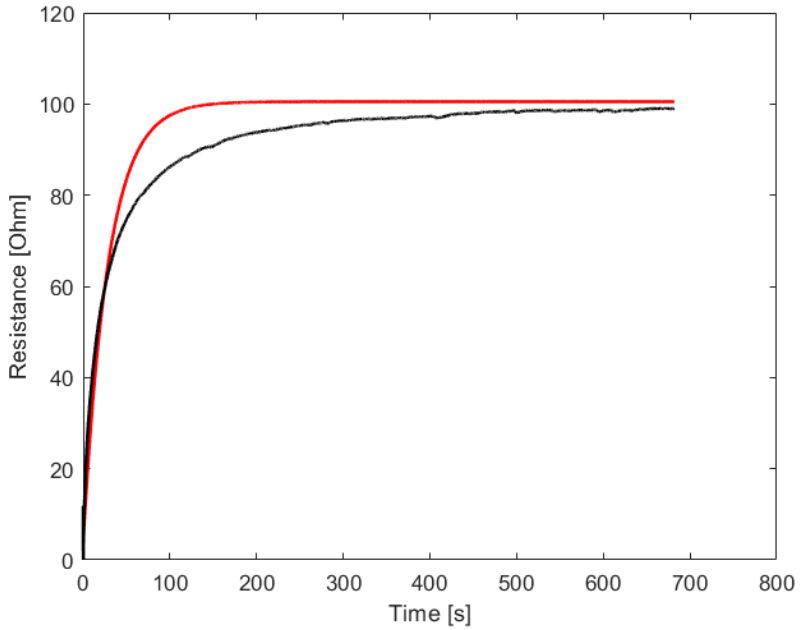


Figure 6.3: First-order model verification (55.37 % fit). The black plot shows the measured data and the red plot shows the simulated output of the first-order model.

To take into account the leakage effects, a second-order model can be found. Using the System Identification toolbox in MATLAB, the following transfer function of the second-order model was found.

$$P(s) = \frac{6.565s + 0.1001}{s^2 + 0.1006s + 0.001019} \quad \left[\mathrm{A}^{-2}\right] \tag{6.3}$$

This model provides a good fit (94.63 %). Figure 6.4 shows the measured data and the simulated output using the second-order model.

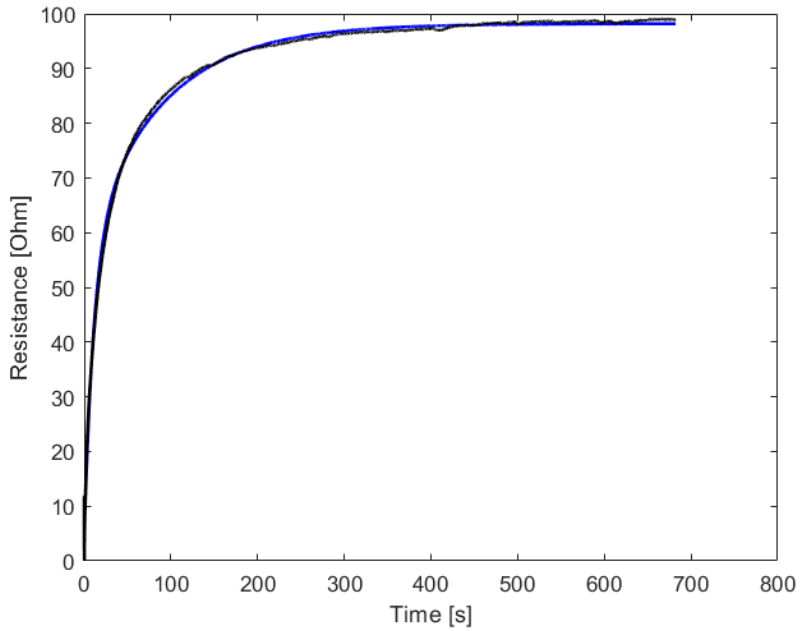


Figure 6.4: Second-order model verification (94.63 % fit). The black plot shows the measured data and the blue plot shows the simulated output of the second-order model.

Wet system

Figure 6.5 shows the response of the system in water when a power step of 1 W is applied. The rise curve is similar to the dry system.

6.2. Resistance control

To design the controller, several problems have to be taken into account. While this thesis only focuses on the control of one thruster, the next step would be to control an array of thrusters. It is possible to make models of all the thrusters, but this would be very time consuming. It is better to design one controller that will work on a variety of thrusters. It was found that a PI controller with $K_p = 1.71$ and $K_i = 1.81$ worked well. Various other K_p and K_i can be used, but it is essential that the magnitude of a and b as calculated in Equation 4.2 don't become too large, as the system will start to display oscillations. Figure 6.6 shows the measurement results of using resistance control with different setpoints.

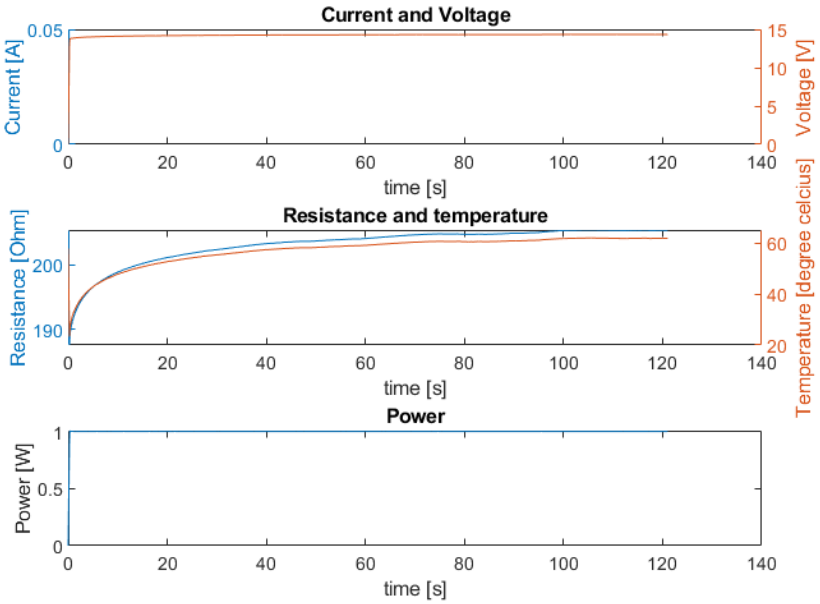


Figure 6.5: Wet system response for 1 W input with $R_0 = 186\,\Omega$

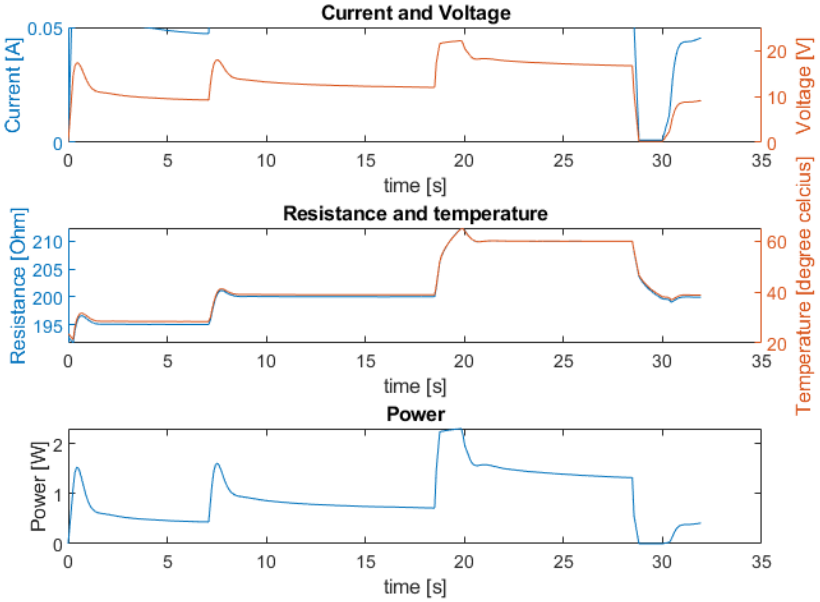


Figure 6.6: Resistance control measurements with different setpoints 195 Ω, 200 Ω, 210 Ω and 200 Ω were used in this order as the setpoints. $R_0 = 186\,\Omega$

7 System with microcontroller

In this chapter, the overview of microcontroller implementation of the control system will be given. With this implementation the sourcemeter is replaced by the supply circuit [7], read-out circuit [8] and microcontroller. The choice of the MCU (microcontroller) will be explained. The details of the communication protocol and control loop implementation will be given.

7.1. Overview

An overview of the total system with MCU is given in Figure 7.1. The voltage and current are used in the control loop to calculate the resistance and power. The pressure measurements are not used to control the system. The communication signals will be explained in Section 7.3. The microscope camera and IR-camera are not synchronized with the MCU and are used with the same programs as in Section 5.1. The controller is very flexible because the PID values can be changed by commands from the computer.

7.2. Microcontroller

The control is done on a MCU, because this makes it more flexible to change the control parameters, the setpoint and thruster type, than with the use of analog components. The MCU chip STM32f767zi was chosen because it could be used with floating-point libraries, which simplifies the implementation of the control. Moreover, the MCU has a high clock speed and the STM32F7 series has a HAL driver that provides an API (Application Programming Interface) with simple functions for communication. The MCU can also send and receive data in three different ways: blocking mode, interrupts (non-blocking) and DMA (direct memory access, non-blocking). For testing and debugging the NUCLEO board is used.
STM32 CUBE IDE is used to write and debug the code. It has a separate interface for device configuration, writing the code and debugging.

7.3. Communication Protocol

For communication, UART (universal asynchronous receiver transmitter) and SPI (serial peripheral interface) are used. The UART serial communication with the computer is convenient because it can go over the same cable that powers the MCU. The SPI protocol is necessary because the used DAC and ADC can only communicate over SPI.

SPI

SPI is a full duplex communication protocol that uses four signals as shown in Table 7.1. In the communication between two devices, there is a master and a slave device. The master device sends out the clock and the slave select. One master devices can have multiple slaves. In the application, the MCU is the master and the read-out and supply devices are slaves.

Table 7.1: SPI signals

CLK	Clock
MISO	Master In Slave Out
MOSI	Master Out Slave In
NSS	Slave Select

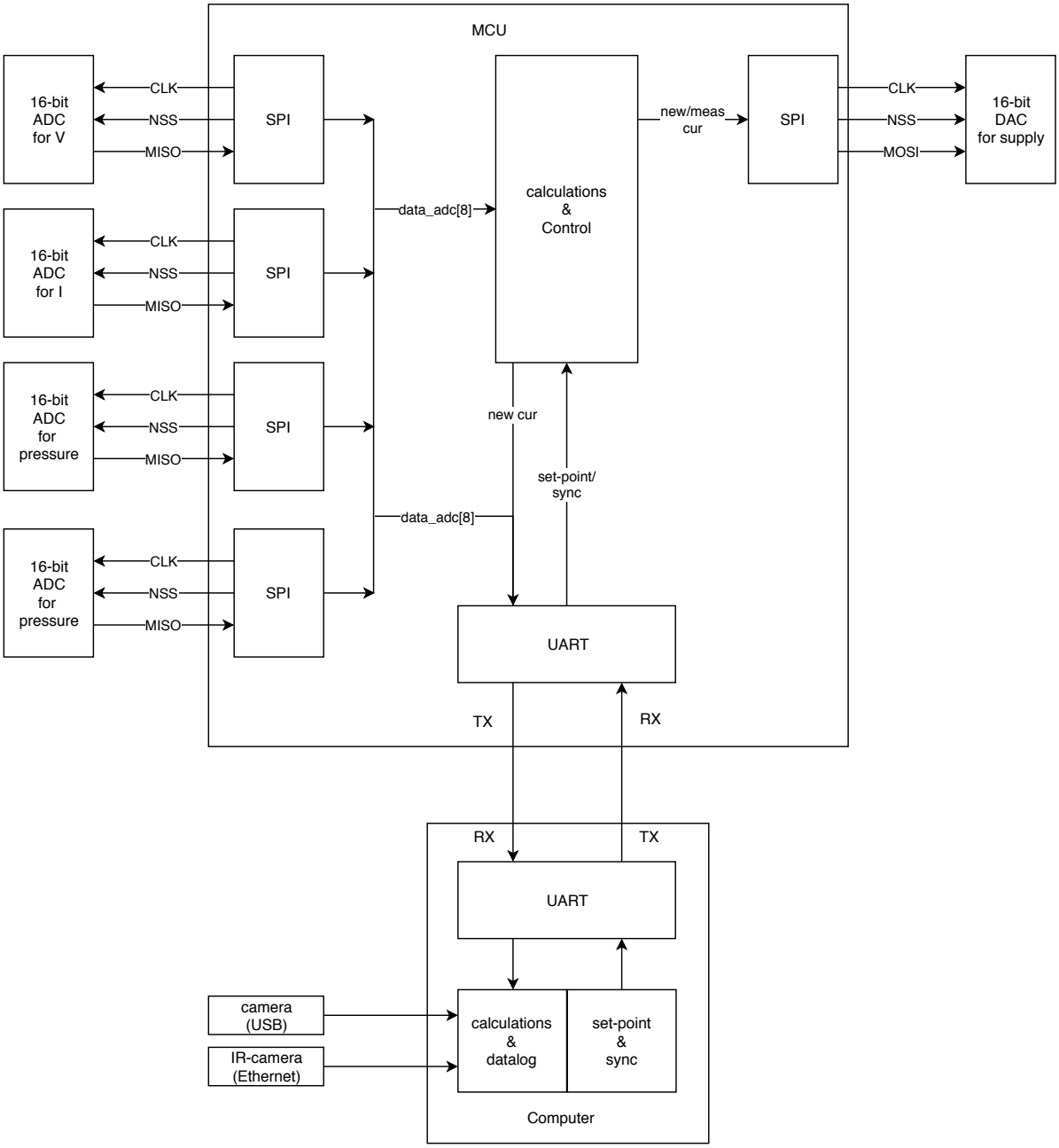


Figure 7.1: Overview system with MCU

ADC The information about the SPI settings for the 16-bit ADC (analog to digital converter) as shown in Table 7.2 were found in the datasheet [26]. Because the MCU only needs to receive from the ADC, the master is set as receive-only. This means that the MOSI is not used.
The data is received in two's complement format. This results in a range of $V_{\text{ref}} - V_{\text{LSB}}$ to $-V_{\text{ref}}$.
The clock speed is lower than the maximum of the ADC, is because a higher clock speed makes and long cables the system more susceptible for noise. The speed is also limited by to a time delay of 6 ns due to a digital isolator.
The software slave-select is used instead of the hardware, because the hardware slave-select stays low after finishing communication. Although all the ADC masters have the same configuration and are used in blocking mode, the choice was made not to use one master with multiple slaves. This made it possible to test the non-blocking mode settings. The non-blocking mode settings are not used because they gave timing issues due to the continuous clock and software slave select.

Table 7.2: ADC Settings

	ADC	MCU
Max SPI frequency	100 Mhz	12.5 Mhz
Min sync high	710 ns	-
CPOL	1	1
CPHA	1	1
	16-bit	16-bit
	MSB-first	MSB-first

DAC The information about the SPI settings for the 16-bit DAC (digital to analog converter) as shown in Table 7.3 were found in the datasheet [27]. Because the MCU only needs to transmit to the DAC, the master is set as transmit-only. This means that the MISO is not used. The clock speed is lower than the maximum of the DAC, is because a higher clock speed and the long cables makes the system more susceptible for noise. The speed is also limited by a time delay of 6 ns due to a digital isolator. Because of the constant frequency of 1 kHz, the minimal sync high time is always met. The master is again used in blocking mode due to the software slave select.

Table 7.3: DAC Settings

	DAC	MCU
Max SPI frequency	30 MHz	12.5 Mhz
Min sync high	12 ns	-
CPOL	0	0
CPHA	1	1
	24-bit	3x8-bit
	MSB-first	MSB-first

The data that needs to be sent has to be in the correct format. Before every 16 bits of data, 8 bits of zeros need to be sent, resulting in a total of 24 bits, see Table 7.4. After the data has the right format, it is sent to the DAC.

Table 7.4: Send format DAC

0000	0000	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0

UART

UART is a full duplex asynchronous communication protocol. The data will be sent in packages of 8-bit. Because this happens asynchronously, a start and stop sequence needs to be sent with each package. The signals needed for this protocol can be found in Table 7.5. The RX from the computer is connected to the TX of the MCU, and the TX from the computer is connected to the RX of the MCU. The speed is 115 200 bits/s

UART uses interrupt mode on the MCU, because it is non-blocking. On the computer, the program Putty

Table 7.5: UART signals

RX	Receive data
TX	Transmit data

was first used to send and receive. This program can log the data to a file, and this file can be read out with MATLAB. Because Putty sends everything that is typed in the terminal in ASCII , the received data needs to be converted from ASCII to integers on the MCU. Therefore the choice was made to use MATLAB to send and receive. The data can be sent as integers.

7.4. State diagram

The state diagram can be seen in Figure 7.2. To get a constant sampling frequency, a timer of 1 kHz is used. With this constant frequency, the time vector can be made on the computer without sending timing information along each sample. There are three states. During the off and operation states, it is checked if a signal is received from the computer and if the states needs to be changed. The blue led on the board is on when the device is in the operation state. The code can be found in Section C.1.

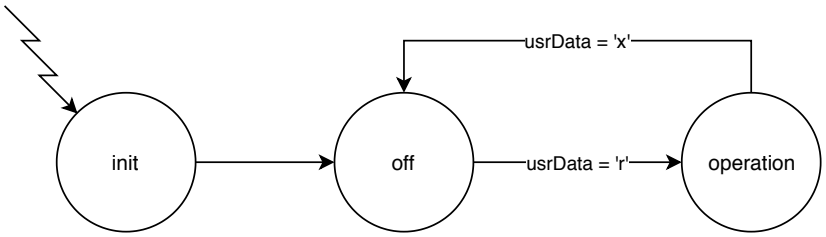


Figure 7.2: State diagram

Initialization

During the Initialization state, the UART starts receiving usrData. The output of the DAC is also set to zero to make sure that there is no unknown current coming from the current supply.

Off

During the off state, different commands can be received. To change the type of control, the command needs to start with 't'. For PID power control the integer '0' needs to be sent, a '1' needs to be sent for PID resistance control and a '2' for a current sweep and a '3' for power with calculation. To change the setpoint, the command needs to start with an 's'. There are two kinds of setpoints: a power setpoint and a resistance setpoint. Because both the setpoints are sent as floats, the same function on the MCU can be used. For the power setpoint, the 'double' 64-bit in MATLAB is converted to a 'single' 32-bit and this is set in an array of 4 bytes. These bytes are then sent. On the MCU, the array is converted back to a float. For the resistance setpoint, first a calculation is done from the start resistance and temperature setpoint to a resistance setpoint, with use of Equation 3.2. This 'double' is transmitted in the same way as the power setpoint. The values used for the PID control can be changed when the command starts with 'p', 'i' or 'd'. These values are also sent as floats.
When the command 'r' is sent, the a, b, c values from Equation 4.2 are calculated and the device goes to the operation state.

Operation

During the operation state it is checked if the next state needs to be the off state (the command 'x' has been sent) and if the setpoint is changed (command 's' and the new setpoint has been sent). Then it depends on the type of control which function is executed.

Power control PID During power control, the data from the ADCs are first received. Then, the new value that needs to be sent to the DAC is calculated. This is calculated using information from the read-out system. The 16-bit data is converted to a float voltage and current. From this, the power, error and new current are calculated. A voltage limit is created by setting the output current to zero when the voltage becomes higher than 40 V. The current value output from the PID control is converted to a 16-bit integer that can be sent to the DAC. Before this conversion, there is a check on overflow and a check on a negative number. If the 'float' new current creates an overflow, the highest possible 16-bit value is sent. If the new current is negative, a zero will be sent. This is due to the fact that a negative current will not cool down the system (the system would heat up).
The data from the ADCs and the new current are sent to the computer. The data that will be sent can be seen in Figure 7.3. The green arrow is the starting point with correct synchronization. The red arrows are wrong starting points, if the computer starts receiving on one of this points, the data is shifted and the wrong plots will be made. There are different ways to synchronize the data. One way is to send a synchronization signal with every 8 bytes to indicate where the begin is. This is not feasible because the data can get any value and

there is no sequence that can work as the synchronization signal. Therefore MATLAB start reading right after it has sent the 'r' signal to indicate that is ready to receive.
The new current is sent to the DAC and every 1 ms, the loop starts again.

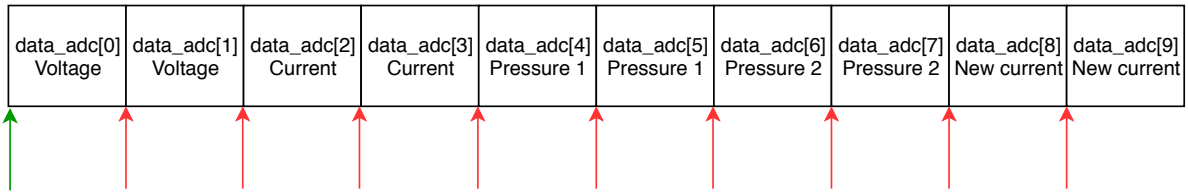


Figure 7.3: Synchronization UART

Power calculated To get a constant power, the current can also be calculated directly with Equation 4.3. The values are measured and sent in the same way as power control PID, only the calculation of the new current uses the power current relation.

Current sweep The current sweep can be used to verify the starting resistance. After 1 ms the new current is increased with 1 bit.

Resistance control PID During resistance control, there is a change that the control sets the current to 0 A. When this happens, the resistance can not be calculated any more because there are no voltages over the ADCs. Therefore, a measurement current is sent periodically. The ADCs start measuring, the new current and the data transmit to the computer and DAC are done in a similar way as power control. After the new current (the heating current) is sent, it waits 9 ms in which the resistance can heat up or cool down. After that, the constant measurement current is sent and the ADCs begin measuring again. Due to the heating time, the sampling time in this control mode is 100 Hz.

Resistance control During resistance control, a new output power value is determined by PID control. This power setpoint is followed by the power calculated function.

7.5. User interface

The MATLAB code is split into sections: open com, send data, receive data, close com, save data and plot data. The received data that is plotted are: measured voltage, measured current send to DAC. The calculated data that is plotted are: power, resistance and temperature. This code can be found in Subsection C.2.2.
To get a UI, the app designer from MATLAB is used. The app has less options than the script (the data can not be seen and only the voltage and current are plotted). The UI can be seen in Figure 7.4.

7.6. Testing

Different tests were performed to ensure that the subsystems work. At the end, tests were done with the total system.

Communication

The SPI communication is first tested by checking the communication signals from Table 7.1 on the oscilloscope. Then, a known positive signal was put on the input of one of the ADCs. The received data was forwarded to the DAC and the output of the DAC was shown on the oscilloscope.

The transmit to the MCU via UART was first tested with the program Putty and the debug option in the CUBEIDE. With help of breakpoints it is checked that the data that is received is the same as the data transmitted. With MATLAB, the same tests were done.
Transmission from the MCU to the computer was tested by sending a known periodic data signal, and plotting the received data.

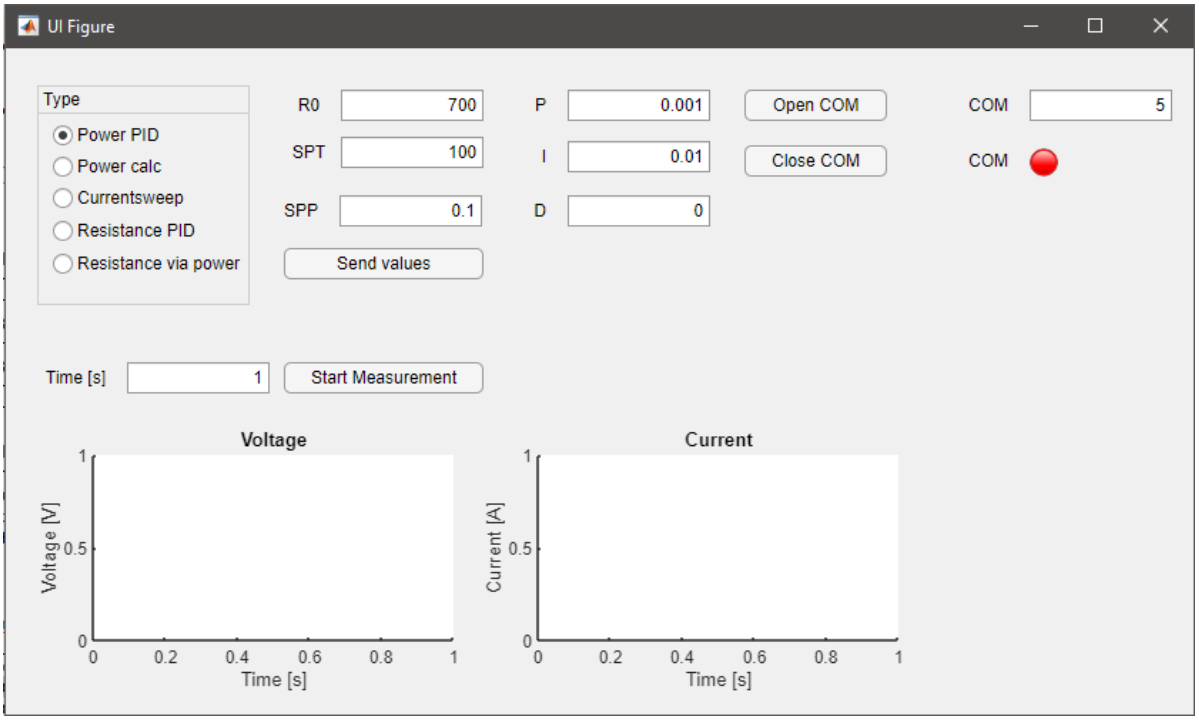
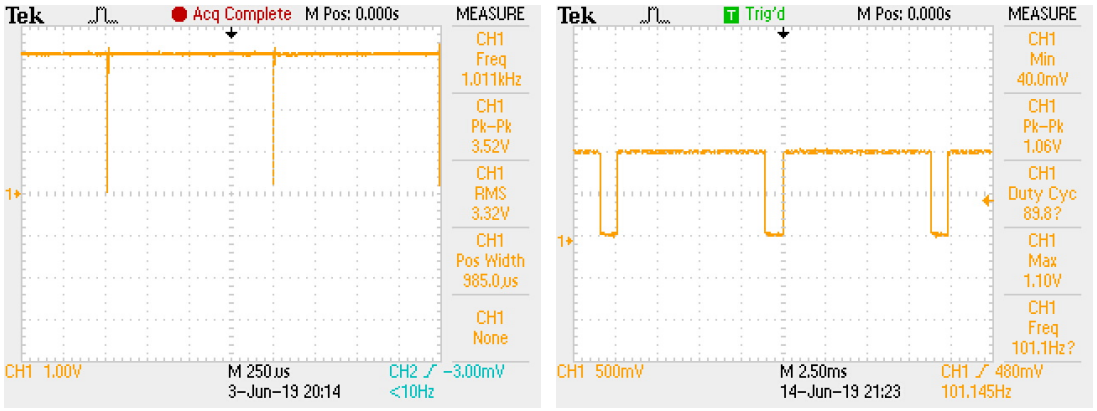


Figure 7.4: User interface MATLAB

Timing of the system

The timing is checked with help of the oscilloscope. To verify if the 1 ms timer works, the oscilloscope is used to determine the difference between the NSS. In Figure 7.5a, the NSS of the first ADC (begin of state machine loop) can be seen. It is concluded that the timer of 1 ms works.



(a) NSS first ADC 1 ms timer

(b) Output DAC resistance timing

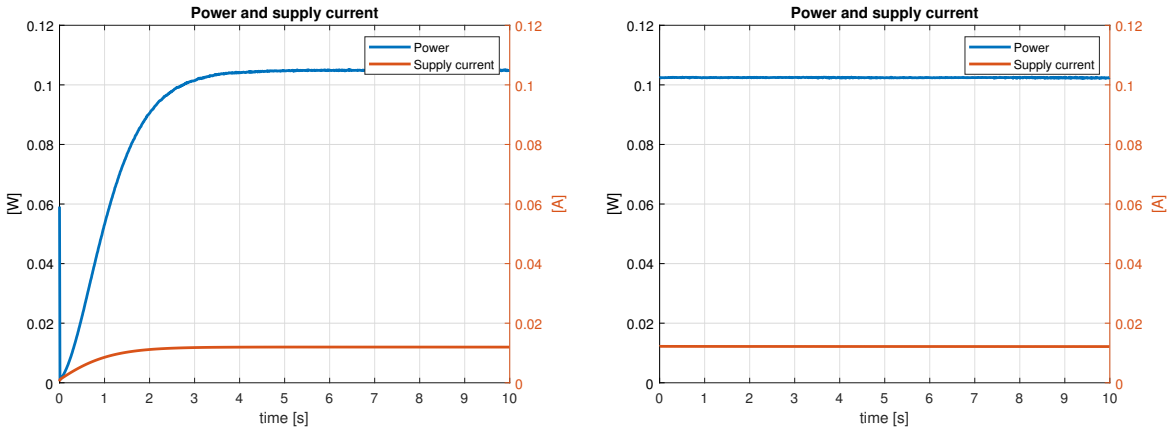
Figure 7.5: Test timing

To verify if the timing for resistance control works, the output of the DAC was displayed on the oscilloscope. In Figure 7.5b, instead of real control values, the heating current is set to 0xFFFF and the measurement current to 0x0F00. In Section B.1 the result of the timing during resistance control in the total system can be seen. The measurement to determine the settling time for the supply subgroup are explained in Section B.2.

Integration of total system

During the measurements of the total system, the IR-camera and microscope-camera were not connected. The received values from the pressure sensor were transmitted back to the computer. Because the pressure sensor was not connected, the received value stayed zero. The total system is dry-tested in all modes on a

thruster of 735 Ω. During the dry tests with the total system, there are sometimes spikes in the voltage measurement that are not expected. These spikes are not filtered because with water measurements a stochastic bubbling effect is expected and a filter would also filter this effect from the measurement data.
In Figure 7.6a an example of a power PID control measurement can be seen. An example of the same power setpoint by calculation can be seen in Figure 7.6b. These measurement are done with multiple power setpoints. There are small differences between the setpoint and the steady-state value, this is due to the floating-point calculations.

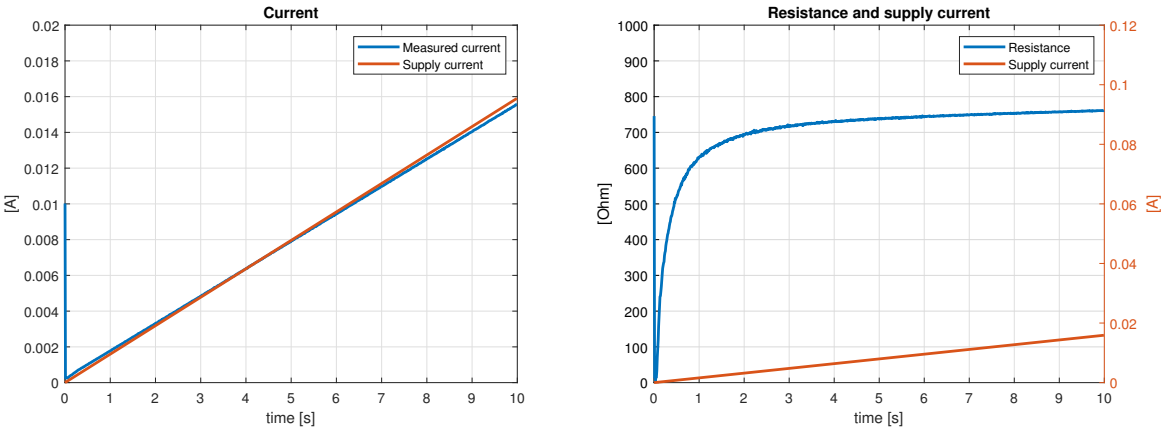


(a) PID, $P = 0.1, I = 0.01, D = 0$

(b) Calculated

Figure 7.6: 0.1 W control in two different ways, $R_0 = 735\,\Omega$

In Figure 7.7a an example of a current sweep is given, the difference between the current sent to the supply and the measured current can be seen. In Figure 7.7b the resistance during this sweep can be seen.

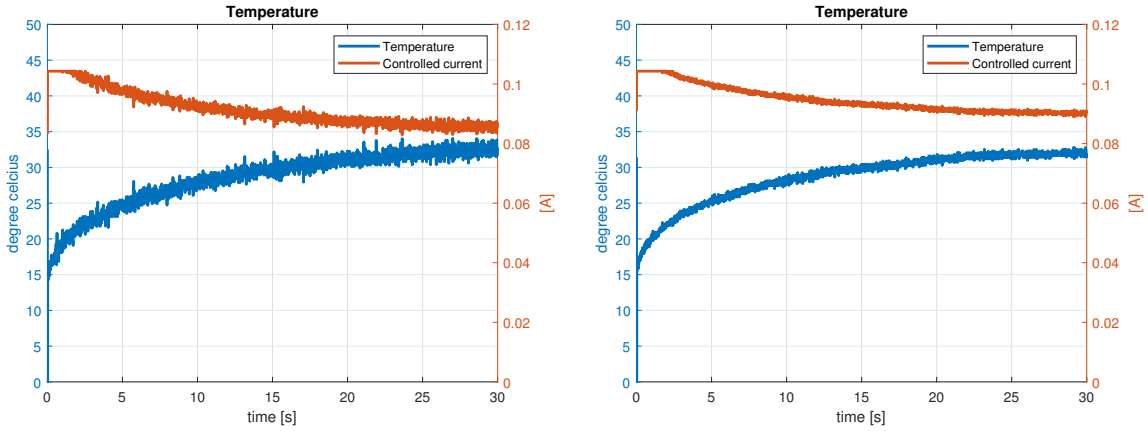


(a) Sweep current

(b) Resistance during current sweep

Figure 7.7: Current sweep

In Figure 7.8a, the temperature control to a setpoint of 30 °C with control via direct PID control can be seen. In Figure 7.8b the temperature control to a setpoint of 30 °C with power control can be seen. Although the PID values are not optimal, it can be seen that the control reacts on a difference in resistance/temperature.



(a) Resistance control via current output, $P = 0.01, I = 0.00001, D = 0$

(b) Resistance control via power setpoint, $P = 0.1, I = 0.0001, D = 0$

Figure 7.8: 30 °C control in two different ways, $R_0 = 735\,\Omega$

8 Conclusion and discussion

For different thrusters and different PID values, the oscillation and rise time differ. The PID constants can be altered for both the Keithley and the MCU system to allow the control within 10 °C of the setpoint for various thrusters (Requirement 1b, Requirement 3c). For both systems, the data can be saved and reloaded for later research. The system can not read out an array of three thrusters (Requirement 1h).

System with lab equipment The resistance control with the Keithley 2450 SourceMeter has been tested and it was able to control a thruster within 10 °C in a dry or wet environment. This system can not read out the pressure sensors (Requirement 1c). The LabVIEW VIs can be used as frond-end data acquisition software (Requirement 1g). The maximum temperature is limited by setting a limit to the temperature setpoint in the VI (Requirement 2a). The voltage limit is set by sending a voltage limit command (Requirement 2b).

System with MCU For the MCU system, it has not been tested if the temperature stays within 10 °C of the setpoint.
There are SPI channels reserved for the read-out of two pressure sensor and the data is sent back to the computer, this has not been tested with the sensors connected (Requirement 1c).
The MATLAB app can be used as frond-end data acquisition software for the system with MCU (Requirement 1g).
In the MATLAB code there is a limit of the temperature setpoint (Requirement 2a).
The output current is set to zero on the MCU when the measured voltage becomes too large (Requirement 2b).

Future work In the future, more measurements could be done for testing the system. The pressure sensors could be connected and read out. The system could be adapted for newer versions of the read-out circuit. For the different thrusters, research could be done on the PID values and the output of the system. Because of the strong variation of the process parameters such as the transfer resistance and heat capacitance, an adaptive control system would be required to achieve better performance. The system could be expanded to read out an array of three thrusters.
We recommend replacing the pressure sensors in the system to one with a higher temperature range and higher pressure range.

Bibliography

[1] J. Bouwmeester and J. Guo, "Survey of worldwide pico- and nanosatellite missions, distributions and subsystem technology", *Acta Astronautica*, vol. 67, no. 7, pp. 854–862, 2010, ISSN: 0094-5765. DOI: https://doi.org/10.1016/j.actaastro.2010.06.004. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0094576510001955.

[2] E. Kulu, *Nanosats database*. [Online]. Available: www.nanosats.eu.

[3] M. A. Silva, M. Shan, A. Cervone, and E. Gill, "Fuzzy control allocation of microthrusters for space debris removal using cubesats", *Engineering Applications of Artificial Intelligence*, vol. 81, pp. 145–156, 2019, ISSN: 0952-1976. DOI: https://doi.org/10.1016/j.engappai.2019.02.008. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0952197619300314.

[4] E. Gill, P. Sundaramoorthy, J. Bouwmeester, B. Zandbergen, and R. Reinhard, "Formation flying within a constellation of nano-satellites: The qb50 mission", *Acta Astronautica*, vol. 82, no. 1, pp. 110–117, 2013, 6th International Workshop on Satellite Constellation and Formation Flying, ISSN: 0094-5765. DOI: https://doi.org/10.1016/j.actaastro.2012.04.029. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0094576512001440.

[5] L. Felicetti and F. Santoni, "Nanosatellite swarm missions in low earth orbit using laser propulsion", *Aerospace Science and Technology*, vol. 27, no. 1, pp. 179–187, 2013, ISSN: 1270-9638. DOI: https://doi.org/10.1016/j.ast.2012.08.005. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1270963812001253.

[6] "Nasa technology roadmaps", *TA 2: In-Space Propulsion Technologies*, 2015.

[7] K. Lam and M. Mrahorović, "Design and implementation of a power supply for mems vaporizing liquid microthrusters", (to appear), 2019.

[8] C. Straathof and R. van Wijk, "Measuring device for controlling a vaporising microthruster", (to appear), 2019.

[9] K. Lemmer, "Propulsion for cubesats", *Acta Astronautica*, vol. 134, pp. 231–243, 2017, ISSN: 0094-5765. DOI: https://doi.org/10.1016/j.actaastro.2017.01.048. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0094576516308840.

[10] M. A. Silva, D. C. Guerrieri, A. Cervone, and E. Gill, "A review of mems micropropulsion technologies for cubesats and pocketqubes", *Acta Astronautica*, vol. 143, pp. 234–243, 2018, ISSN: 0094-5765. DOI: https://doi.org/10.1016/j.actaastro.2017.11.049. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0094576517304290.

[11] A. Kurmanbay, "Design, fabrication, and characterization of mems based micro heater for vaporizing liquid microthruster", (to appear), 2019.

[12] M. A. Silva, D. C. Guerrieri, H. van Zeijl, A. Cervone, and E. Gill, "Vaporizing liquid microthrusters with integrated heaters and temperature measurement", *Sensors and Actuators A: Physical*, vol. 265, pp. 261–274, 2017, ISSN: 0924-4247. DOI: https://doi.org/10.1016/j.sna.2017.07.032. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0924424717306490.

[13] D. Selva and D. Krejci, "A survey and assessment of the capabilities of cubesats for earth observation", *Acta Astronautica*, vol. 74, pp. 50–68, 2012, ISSN: 0094-5765. DOI: https://doi.org/10.1016/j.actaastro.2011.12.014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0094576511003742.

[14] A. Cervone, B. Zandbergen, J. Bouwmester, and J. Guo, "Micro-propulsion research; challenges towards future nano-satellite projects", *Leonardo Times*, 2013. [Online]. Available: http://resolver.tudelft.nl/uuid:c1757611-8f73-4061-a573-d748aaa36b23.

[15] A. Tummala and A. Dutta, "An overview of cube-satellite propulsion technologies and trends", *Aerospace*, vol. 4, Dec. 2017. DOI: 10.3390/aerospace4040058.

[16] A. Cervone, B. Zandbergen, D. C. Guerrieri, M. De Athayde Costa e Silva, I. Krusharev, and H. van Zeijl, "Green micro-resistojet research at delft university of technology: New options for cubesat propulsion", *CEAS Space Journal*, vol. 9, no. 1, pp. 111–125, Mar. 2017, ISSN: 1868-2510. DOI: 10.1007/s12567-016-0135-3. [Online]. Available: https://doi.org/10.1007/s12567-016-0135-3.

[17] J. Tsai and L. Lin, "Transient thermal bubble formation on polysilicon micro-resisters", *Journal of Heat Transfer*, vol. 124, pp. 375–382, 2002. DOI: 10.1115/1.1445136.

[18] K. T. Wen-Jei Yang, "Overview of boiling on microstructures - macro bubbles from micro heaters", *Microscale Thermophysical Engineering*, vol. 4, no. 1, pp. 7–24, 2000. DOI: 10.1080/108939500199600. eprint: https://doi.org/10.1080/108939500199600. [Online]. Available: https://doi.org/10.1080/108939500199600.

[19] M. Chambers, "Design and fabrication of a microheater control system", PhD thesis, University of Utah, 2006. [Online]. Available: https://my.ece.utah.edu/~mchamber/final-report.pdf.

[20] J. Kuo, L. Yu, and E. Meng, "Micromachined thermal flow sensors—a review", *Micromachines*, pp. 550–573, 2012. [Online]. Available: https://www.mdpi.com/2072-666X/3/3/550.

[21] M. Kutz, *Temperature control*. John Wiley & Sons, Inc, 1986.

[22] G. Franklin, J. Powell, and A. Emami-Naeini, *Feedback control of dynamic systems*. Pearson, 2015, pp. 216–228.

[23] M. Kondratiuk, L. Ambroziak, E. Pawluszewicz, and J. Janczak, "Discrete pid algorithm with non-uniform sampling – practical implementation in control system", *AIP Conference Proceedings*, vol. 2029, no. 1, p. 020 029, 2018. DOI: 10.1063/1.5066491. [Online]. Available: https://aip.scitation.org/doi/abs/10.1063/1.5066491.

[24] *2450 sourcemeter smu instrument datasheet*, Keithley, a Tektronix Company. [Online]. Available: https://www.tek.com/sites/default/files/media/media/resources/1KW-60904-0_2450_DataSheet_0.pdf.

[25] *2450 sourcemeter smu instrument reference manual*, 2450-901-01, Rev.B, Keithley, a Tektronix Company, 2013, ch. 5-1, 6-1. [Online]. Available: https://smt.at/wp-content/uploads/smt-handbuch-keithley-2450-englisch.pdf.

[26] *1 msps 16-bit differential input sar adc*, MCP33131D-10, Microchip Technology Inc., 2018. [Online]. Available: http://www.farnell.com/datasheets/2608040.pdf.

[27] *16-bit vout nanodac*, AD5061, Rev.C, Analog Devices, 2017. [Online]. Available: http://www.farnell.com/datasheets/2250565.pdf.

A

IR-camera

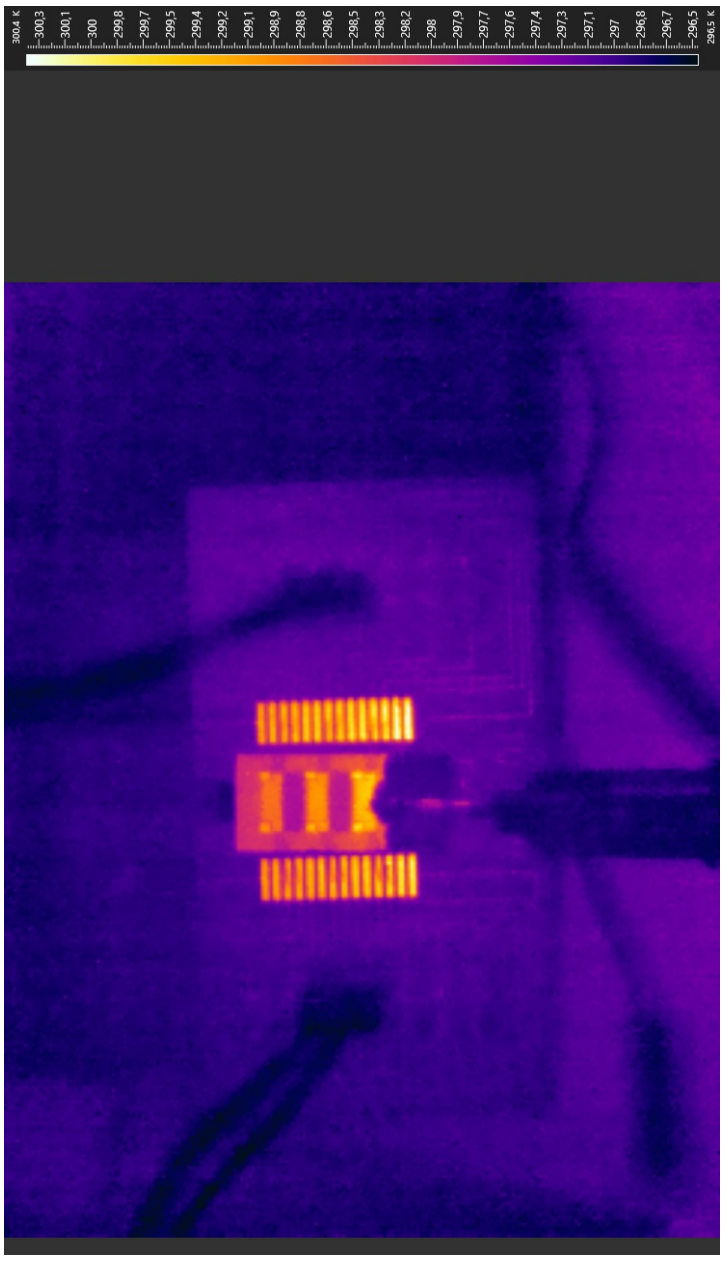


Figure A.1: IR-camera on three thrusters. The middle thruster is heated

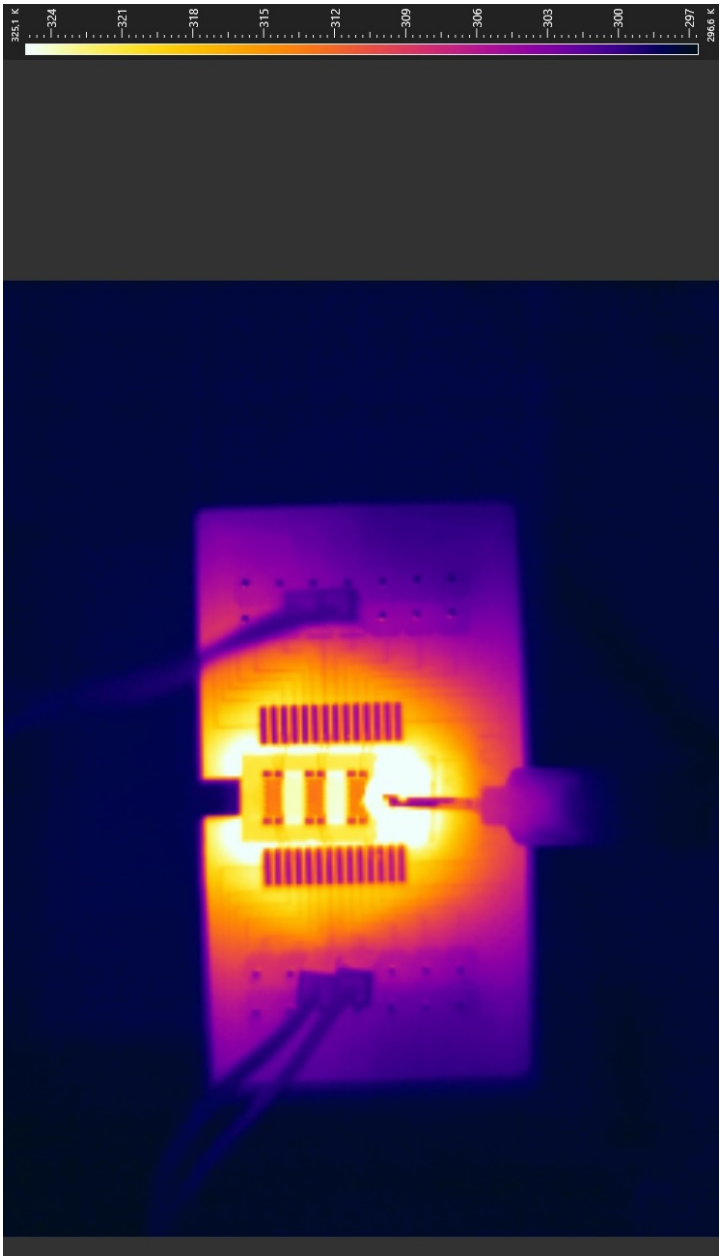


Figure A.2: IR-camera on three thrusters. The system is cooling down

B

Additional MCU testing results

In this appendix additional MCU testing results can be found.

B.1. Measurement pulses

In Figure B.1 the measured and supply heating current can be seen during a resistance control measurement. The measured current does not follow the supply heating current because during the measurement, the measurement current is supplied. In Figure B.2 it can be seen that the measured current follows the supplied current during the measurement.

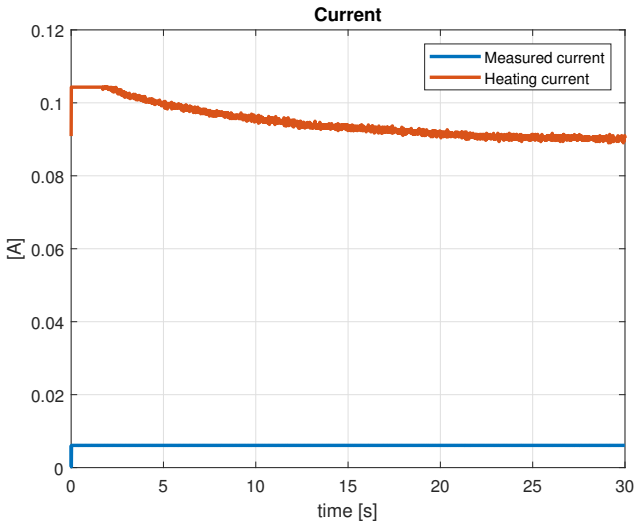


Figure B.1: Supplied and received current

B.2. Timing requirement supply group

In Figure B.3, the difference between the NSS signal from the DAC and the NSS signal of the first ADC. By measuring the delay between these two signals, the maximal settling time of the supply can be found. The maximum settling time is 24 µs

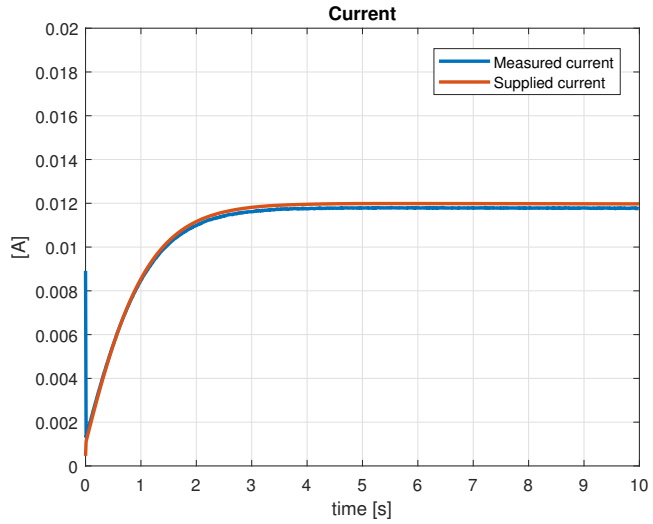


Figure B.2: Currents during power control with PID

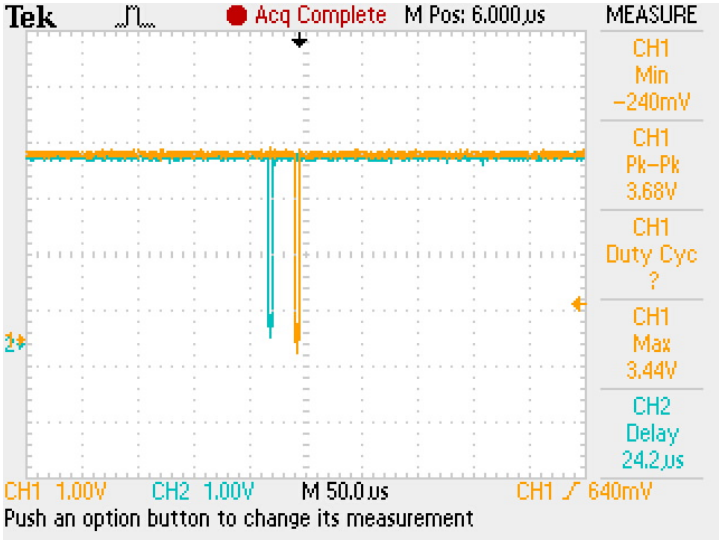


Figure B.3: Channel 1: NSS of the first ADC, channel 2: NSS of the DAC

C
Code

In this appendix the MCU and MATLAB code can be found.

C.1. MCU code

The code used on the MCU stands in this section.

C.1.1. App

```
/*
 * app.h
 *
 *  Created on: 15 jun. 2019
 *      Author: Marjolein Rebers
 */

#ifndef INC_APP_H_
#define INC_APP_H_

// includes
#include "main.h"
#include "getdata.h"
#include "senddata.h"
#include "calcon.h"
#include "main.h"
#include "spi.h"
#include "tim.h"
#include "usart.h"
#include "gpio.h"

// Defines
#define BUF_SIZE 20
#define HEARTBEAT_MAX_CNT 200
#define CONTROL_MAX_CNT0 10
#define CONTROL_MAX_CNT1 9


// typedefs and structs
typedef struct s_command {
        uint16_t len;
        uint8_t data[BUF_SIZE];
}t_command;

// Statemachine
enum t_state {
        off, opperation, error
};
```

```
39
40 // Initialize the application and the hardware
41 void init_app();
42
43 // Run the app
44 void run_app();
45
46 // uart handle for in ISR
47 void uart_recv_handle(UART_HandleTypeDef *huart);
48
49 #endif /* INC_APP_H_ */
```

```
1 /*
2  * app.c
3  *
4  *  Created on: 15 jun. 2019
5  *      Author: Marjolein Rebers
6  */
7
8 // Includes
9 #include "app.h"
10
11
12
13 // Variables
14 // Communication data
15 t_command intData;
16 t_command usrData;
17 uint8_t usrDataRdy;
18
19 // Heartbeat
20 uint16_t heartbeatCnt = 0x0;
21
22 // Current data
23 uint8_t start_cur[2] = { 0 };
24 uint8_t meas_cur[2] = { 0x00, 0x0f };
25
26 //known buffer used for testing uart
27 //uint8_t data[8] = { 1, 1, 1, 2, 0, 3, 0,4 };
28
29 // Control
30 uint8_t type = 0;
31 uint8_t controlCnt0 = 0x0;
32 uint8_t controlCnt1 = 0x0;
33 uint8_t data[10] = {0};
34 float sp = 0;
35 float Kp = 0;
36 float Ki = 0;
37 float Kd = 0;
38 float a = 0;
39 float b = 0;
40 float c = 0;
41 float Ts = 0.001;
42
43
44
```

```
45 // SysTick counters
46 uint32_t sysTick = 0;
47 uint32_t prevTick = 0;
48
49 // Statemachine
50 enum t_state state = off;
51
52 // Satemachine functions
53 void off_state();
54 void opperation_state();
55 void error_state();
56
57 // Control functions
58 void PowerControl();
59 void ResistanceControlPID();
60 void ResistanceControl();
61 void CurrentSweep();
62 void PIDPowerControl();
63
64 // Initialize the application and the hardware
65 void init_app() {
66         int i;
67
68         //Stop timer during debug
69         DBGMCU->APB1FZ |= (DBGMCU_APB1_FZ_DBG_TIM3_STOP);
70
71         //Start with 0 A
72         send_data_dac(start_cur, 2);
73
74         // Set systick to 1 KHz
75         if (HAL_GetTickFreq() != HAL_TICK_FREQ_1KHZ) {
76                 HAL_SetTickFreq(HAL_TICK_FREQ_1KHZ);
77         }
78
79         // set all the command data to zero
80         for (i = 0; i < BUF_SIZE; i++) {
81                 intData.data[i] = 0x0;
82                 usrData.data[i] = 0x0;
83         }
84         intData.len = 0x0;
85         usrData.len = 0x0;
86         usrDataRdy = 0x0;
87
88         // start a data request
89         HAL_UART_Receive_IT(&huart3, intData.data, BUF_SIZE);
90 }
91
92 // Run the app
93 void run_app() {
94         int i;
95
96         // Get current sysTick count
97         sysTick = HAL_GetTick();
98
99         // Check if the sysTick count is increased
100        if (prevTick != sysTick) {
```

```
101                 // Set previous sysTick to the current sysTick
102                 prevTick = sysTick;
103
104                 // Check if we are have new data
105                 if (HAL_UART_GetState(&huart3) != HAL_UART_STATE_BUSY_RX) {
106                         // check if we filled the entire buffer
107                         if (intData.len == 0) {
108                                 intData.len = BUF_SIZE;
109                         }
110
111                         // copy the data to a local buffer
112                         for (i = 0; i < intData.len; i++) {
113                                 usrData.data[i] = intData.data[i];
114                                 intData.data[i] = 0x0;
115                         }
116                         usrData.len = intData.len;
117                         intData.len = 0x0;
118                         usrDataRdy = 0x1;
119
120                         // start a new data request
121                         HAL_UART_Receive_IT(&huart3, intData.data, BUF_SIZE);
122                 }
123
124                 // Toggle LED 1 as heart beat
125                 if (heartbeatCnt == HEARTBEAT_MAX_CNT) {
126                         HAL_GPIO_TogglePin(LED1_GPIO_Port, LED1_Pin);
127                         heartbeatCnt = 0x0;
128                 } else {
129                         heartbeatCnt += 0x1;
130                 }
131
132                 // Switch through the state machine
133                 switch (state) {
134                 case off:
135                         off_state();
136                         break;
137                 case opperation:
138                         opperation_state();
139                         break;
140                 case error:
141                         error_state();
142                         break;
143                 }
144         }
145 }
146
147 // Uart handler for custom data length
148 void uart_recv_handle(UART_HandleTypeDef *huart) {
149         huart->pRxBuffPtr--;
150         uint8_t tmp0 = *huart->pRxBuffPtr;
151         huart->pRxBuffPtr--;
152         uint8_t tmp1 = *huart->pRxBuffPtr;
153         huart->pRxBuffPtr++;
154         huart->pRxBuffPtr++;
155
156         // check for line end character
```

```
157         if (tmp0 == '\n' && tmp1 == '\r') {
158                 uint16_t index = huart->RxXferSize - huart->RxXferCount - 2;
159                 intData.len = index;
160                 HAL_UART_AbortReceive(huart);
161         }
162 }
163
164 // Statemachine functions
165 void off_state() {
166         // Set state LEDs
167         HAL_GPIO_WritePin(LED2_GPIO_Port, LED2_Pin, 0x0);
168         HAL_GPIO_WritePin(LED3_GPIO_Port, LED3_Pin, 0x0);
169         send_data_dac(start_cur, 2);
170         resetcalcvalue();
171
172         // Process the communication data
173         if (usrDataRdy == 0x1) {
174                 if (usrData.data[0] == 'r' || usrData.data[0] == 'R') {
175                         a = calc_a(Kp, Ki, Kd, Ts);
176                         b = calc_b(Kp, Ki, Kd, Ts);
177                         c = calc_c(Kd, Ts);
178                         state = opperation;
179                 } else if ((usrData.data[0] == 'E' || usrData.data[0] == 'e')
180                                 && usrData.data[1] == 'r' && usrData.data[2] == 'r')
                                        {
181                         state = error;
182                 }else if (usrData.data[0] == 'S' || usrData.data[0] == 's') {
183                         sp = *(float *)((usrData.data + 1));
184                 }else if (usrData.data[0] == 'T' || usrData.data[0] == 't' ) {
185                         type =  usrData.data[1];
186                         if(type == 1){
187                                 Ts = 0.01;}
188                         else {Ts = 0.001;}
189                 }else if (usrData.data[0] == 'P' || usrData.data[0] == 'p') {
190                         Kp = *(float *)((usrData.data + 1));
191                 }else if (usrData.data[0] == 'I' || usrData.data[0] == 'i') {
192                         Ki = *(float *)((usrData.data + 1));
193                 }else if (usrData.data[0] == 'D' || usrData.data[0] == 'd') {
194                         Kd = *(float *)((usrData.data + 1));
195                 }
196                 usrDataRdy = 0x0;
197
198         }
199
200         // Check for errors
201         // TODO: Error checking
202 }
203
204 void opperation_state() {
205         // Set state LEDs
206         HAL_GPIO_WritePin(LED2_GPIO_Port, LED2_Pin, 0x1);
207         HAL_GPIO_WritePin(LED3_GPIO_Port, LED3_Pin, 0x0);
208
209         // Process the communication data
210         if (usrDataRdy == 0x1) {
211                 if (usrData.data[0] == 'x' || usrData.data[0] == 'X') {
```

```
212                     state = off;
213             } else if ((usrData.data[0] == 'E' || usrData.data[0] == 'e')
214                             && usrData.data[1] == 'r' && usrData.data[2] == 'r')
                                    {
215                     state = error;
216             }else if (usrData.data[0] == 'S' || usrData.data[0] == 's') {
217                     sp =  *(float *)(&usrData.data+1);
218             }
219             usrDataRdy = 0x0;
220     }
221
222     // Run the controller
223     if (type == 0) {
224             PIDPowerControl();
225     } else if (type == 1) {
226             PowerControl();
227     } else if (type == 2){
228             CurrentSweep();
229     } else if (type == 3){
230             ResistanceControlPID();
231     } else{
232             ResistanceControl();
233     }
234
235
236     // Check for errors
237     // TODO: Error checking
238 }
239
240 void error_state() {
241     // Set state LEDs
242     HAL_GPIO_WritePin(LED2_GPIO_Port, LED2_Pin, 0x0);
243     HAL_GPIO_WritePin(LED3_GPIO_Port, LED3_Pin, 0x1);
244
245     // Process the communication data
246     if (usrDataRdy == 0x1) {
247             if ((usrData.data[0] == 'r' || usrData.data[0] == 'R')
248                             && usrData.data[1] == 's' && usrData.data[2] == 't')
                                    {
249                     init_app();
250                     state = off;
251             }
252             usrDataRdy = 0x0;
253     }
254     //Set output current to 0
255     send_data_dac(start_cur, 2);
256
257
258 }
259
260 void PowerControl() {
261     //Control and acquisition
262     get_data_adc(data);
263     calc_pow(sp, data);
264     send_data_com(data, 10);
265     send_data_dac(data, 10);
```

```
266
267 }
268
269 void ResistanceControl(){
270         float psp;
271         // Keep timing and execute the commands
272                 if (controlCnt0 == CONTROL_MAX_CNT0 && controlCnt1 == 0x0) {
273                         //Control and acquisition
274                         get_data_adc(data);
275                         psp = calc_sppower(sp, data, a, b, c);
276                         calc_pow(psp, data);
277                         send_data_com(data, 10);
278                         send_data_dac(data, 10);
279                         // Add to the counters
280                         controlCnt1 += 0x1;
281                 } else if (controlCnt0 == CONTROL_MAX_CNT0
282                                 && controlCnt1 != CONTROL_MAX_CNT1) {
283                         // Add to the counters
284                         controlCnt1 += 0x1;
285                 }
286                 else if (controlCnt0 != CONTROL_MAX_CNT0
287                                 && controlCnt1 != CONTROL_MAX_CNT1) {
288                         // Add to the counters
289                         controlCnt0 += 0x1;
290                 } else {
291                         // Send data
292                         send_data_dac(meas_cur, 2);
293                         // Reset counters
294                         controlCnt0 = 0x0;
295                         controlCnt1 = 0x0;
296                 }
297
298 }
299
300 void PIDPowerControl(){
301         //Control and acquisition
302         get_data_adc(data);
303         calc_control(type, sp, data, a,b,c);
304         send_data_com(data, 10);
305         send_data_dac(data, 10);
306 }
307
308 void ResistanceControlPID() {
309         // Keep timing and execute the commands
310         if (controlCnt0 == CONTROL_MAX_CNT0 && controlCnt1 == 0x0) {
311                 //Control and acquisition
312                 get_data_adc(data);
313                 calc_control(type, sp, data, a,b,c);
314                 send_data_com(data, 10);
315                 send_data_dac(data, 10);
316                 // Add to the counters
317                 controlCnt1 += 0x1;
318         } else if (controlCnt0 == CONTROL_MAX_CNT0
319                         && controlCnt1 != CONTROL_MAX_CNT1) {
320                 // Add to the counters
321                 controlCnt1 += 0x1;
```

```
322         }
323         else if (controlCnt0 != CONTROL_MAX_CNT0
324                         && controlCnt1 != CONTROL_MAX_CNT1) {
325                 // Add to the counters
326                 controlCnt0 += 0x1;
327         } else {
328                 // Send data
329                 send_data_dac(meas_cur, 2);
330                 // Reset counters
331                 controlCnt0 = 0x0;
332                 controlCnt1 = 0x0;
333         }
334
335 }
336
337 void CurrentSweep() {
338         //current sweep
339         get_data_adc(data);
340         cur_sweep(data);
341         send_data_com(data, 10);
342         send_data_dac(data, 10);
343 }
```

C.1.2. Receive

```
1  /*
2   * getdata.c
3   *
4   *  Created on: May 30, 2019
5   *      Author: Marjolein Rebers
6   */
7
8  #include "main.h"
9  #include "spi.h"
10 #include "usart.h"
11 #include "gpio.h"
12
13 #include "stdio.h"
14 #include "stdlib.h"
15 #include "math.h"
16
17 #include "getdata.h"
18 #include "senddata.h"
19
20
21
22 int get_control_type(void){
23         uint8_t type[2] = {0};
24         HAL_UART_Receive_IT(&huart3, type, 2);
25         while( HAL_UART_GetState(&huart3) == HAL_UART_STATE_BUSY_RX)
26         {__NOP();}
27
28
29         return type[0];
30 }
31
32
```

```
33
34 float get_set_point_matlab(){
35
36         uint8_t receivedSP[5]= {0};
37
38         float sp =0; // set-point
39
40
41
42         HAL_UART_Receive_IT(&huart3, receivedSP, 5);
43                         while( HAL_UART_GetState(&huart3) == HAL_UART_STATE_BUSY_RX)
44                         {__NOP();}
45
46         //sp = ((receivedSP[3]<<24) |(receivedSP[2]<<16) |(receivedSP[1]<<8) |
               receivedSP[0]);
47         sp =  *(float *)&receivedSP;
48
49
50         __NOP();
51
52         return (sp);
53
54
55 }
56
57
58
59 void get_data_adc(uint8_t *data) {
60
61         //Receive Voltage
62         HAL_GPIO_WritePin( NSS1_GPIO_Port, NSS1_Pin, GPIO_PIN_RESET); // NSS1 low
63         HAL_SPI_Receive(&hspi1, data, 1, 1);
64         HAL_GPIO_WritePin( NSS1_GPIO_Port, NSS1_Pin, GPIO_PIN_SET); // NSS1 high
65
66         //Receive Current
67         HAL_GPIO_WritePin( NSS3_GPIO_Port, NSS3_Pin, GPIO_PIN_RESET); // NSS3 low
68         HAL_SPI_Receive(&hspi3, data+2, 1, 1);
69         HAL_GPIO_WritePin( NSS3_GPIO_Port, NSS3_Pin, GPIO_PIN_SET); // NSS3 high
70
71
72         //Receive Pressure 1
73         HAL_GPIO_WritePin( NSS4_GPIO_Port, NSS4_Pin, GPIO_PIN_RESET); // NSS4 low
74         HAL_SPI_Receive(&hspi4, data+4, 1, 1);
75         HAL_GPIO_WritePin( NSS4_GPIO_Port, NSS4_Pin, GPIO_PIN_SET); // NSS4 high
76
77         //Receive Pressure 2
78         HAL_GPIO_WritePin( NSS6_GPIO_Port, NSS6_Pin, GPIO_PIN_RESET); // NSS6 low
79         HAL_SPI_Receive(&hspi6, data+6, 1, 1);
80         HAL_GPIO_WritePin( NSS6_GPIO_Port, NSS6_Pin, GPIO_PIN_SET); // NSS6 high*/
81
82         return;
83 }
```

```
1 /*
2  * getdata.h
3  *
```

```
 *  Created on: May 30, 2019
 *      Author: Marjolein Rebers
 */

#ifndef GETDATA_H_
#define GETDATA_H_

int get_control_type(void);

float get_set_point(void);

float get_set_point_matlab();

void get_data_adc(uint8_t *data);

void check_stop(void);

#endif /* GETDATA_H_ */
```

C.1.3. Send

```
/*
 * senddata.c
 *
 *  Created on: May 29, 2019
 *      Author: Marjolein Rebers
 */

#include "main.h"
#include "spi.h"
#include "usart.h"
#include "gpio.h"
#include "stdio.h"


void send_data_com(uint8_t* data, int length) {

        while( HAL_UART_GetState(&huart3) == HAL_UART_STATE_BUSY_TX)
                                                                {__NOP();}

        HAL_UART_Transmit_IT(&huart3, data, length);



        //Test synchronization with known periodic data
        //data[3] = data[3]+1;
        //data[0] = data[0]+1;


        return;
}

void send_data_dac(uint8_t* data, int length){
        //Use in main function : send_data_dac(new_cur);

        uint8_t supply[3] = { 0x00, 0x00, 0x00 };

```

```
37          if(length ==2){
38                  supply[2] = data[0];
39                  supply[1] = data[1];
40          }
41
42
43          else{
44          supply[2] = data[8];
45          supply[1] = data[9];
46          }
47
48          HAL_GPIO_WritePin( NSS2_GPIO_Port, NSS2_Pin, GPIO_PIN_RESET); // NSS2 low
49          HAL_SPI_Transmit(&hspi2, supply, 3, 1);
50          HAL_GPIO_WritePin( NSS2_GPIO_Port, NSS2_Pin, GPIO_PIN_SET); // NSS2 high
51
52 }
53
54
55 void transfer_data_adc_dac(uint8_t *senddata) {
56          //Use in main function: transfer_data_adc_dac(data_adc);
57
58
59          uint8_t supply[3] = { 0x00, 0x00, 0x00 };
60
61          senddata[0] = senddata[0]<<1;
62          senddata[1] = senddata[1]<<1;
63
64          supply[2] = senddata[0];
65          supply[1] = senddata[1];
66
67          HAL_GPIO_WritePin( NSS2_GPIO_Port, NSS2_Pin, GPIO_PIN_RESET); // NSS2 low
68          HAL_SPI_Transmit(&hspi2, supply, 3, 1);
69          HAL_GPIO_WritePin( NSS2_GPIO_Port, NSS2_Pin, GPIO_PIN_SET); // NSS2 high
70
71 }
```

```
1 /*
2  * senddata.h
3  *
4  *  Created on: May 29, 2019
5  *      Author: Marjolein Rebers
6  */
7
8 #ifndef SENDDATA_H_
9 #define SENDDATA_H_
10
11 void send_data_com(uint8_t *data, int length);
12
13 //void send_data_dac(uint8_t *senddata);
14
15 void send_data_dac(uint8_t* data, int length);
16
17 void transfer_data_adc_dac(uint8_t* senddata);
18
19
20 #endif /* SENDDATA_H_ */
```

C.1.4. Control

```
/*
 * calcon.c
 *
 *  Created on: May 30, 2019
 *      Author: Marjolein Rebers
 */

#include "main.h"

#include "stdio.h"
#include "stdlib.h"
#include "math.h"

#include "calcon.h"









static float error_1 = 0;
static float error_2 = 0;
static float current = 100;
static float psp = 0; //power set point
static uint16_t sweep = 0;


void resetcalcvalue(){
        error_1 = 0;
        error_2 = 0;
        current = 0;
        psp = 0; //power set point
        sweep = 0;
        return;
}


float calc_sppower(float sp, uint8_t *data, float a, float b, float c){


        float error;
        float change;
        float res;

        float voltage_meas;
        float current_meas;

        voltage_meas = calc_vol(data);
        current_meas = calc_cur(data);

        //calculate resistance
        res = voltage_meas / current_meas;
        error = sp-res;

        //calculate change
```

```
56          change = a*error + b*error_1+c*error_2;
57          psp = psp + change;
58          error_2 = error_1;
59          error_1 = error;
60
61
62          return psp;
63  }
64
65
66  void calc_pow(float power,uint8_t *data){
67          uint16_t new_cur;
68          float res;
69          float cur_transfer = (10.2/1.043)*pow(2,16);
70          float voltage_meas;
71          float current_meas;
72          float send_current;
73
74
75
76          voltage_meas = calc_vol(data);
77          current_meas = calc_cur(data);
78
79          res = voltage_meas / current_meas;
80          current = sqrt(power/res)*cur_transfer;
81
82          //overvoltage protection, undercurrent limit
83          if(voltage_meas>40 || current < 0)
84          {
85                  send_current =0;
86                  current = 0;
87          }
88          //Check overflow
89          else if(current>65535){
90                  send_current = 65535;
91                  current = 65535;
92          }
93          else if((voltage_meas<0) || (current_meas<0)){
94                  send_current = 65535;
95          }
96          else
97          {
98                  send_current = current;
99          }
100
101
102         new_cur = (uint16_t) (send_current);
103
104         data[8] = new_cur&0xff;
105         data[9] = (new_cur>>8)&0xff;
106
107         return;
108
109
110 }
111
```

```
112
113 void calc_control(int type, float sp, uint8_t *data, float a, float b, float c) {
114
115         uint16_t new_cur;
116
117         //control
118         float error;
119         float change;
120         float power =0;
121
122
123         float cur_transfer = (10.2/1.043)*pow(2,16);
124         float voltage_meas;
125         float current_meas;
126         float send_current;
127
128
129         voltage_meas = calc_vol(data);
130         current_meas = calc_cur(data);
131
132         float res;
133
134         //calculate error
135         if(type ==0){
136                 //calculate power
137                 power = voltage_meas*current_meas;
138                 error = sp-power;
139         }
140         else{
141                 //calculate resistance
142                 res = voltage_meas / current_meas;
143                 error = sp-res;
144         }
145
146
147
148         //calculate change
149         change = a*error + b*error_1+c*error_2;
150         current = current + (change*cur_transfer);
151
152         error_2 = error_1;
153         error_1 = error;
154
155
156
157
158         //overvoltage protection, undercurrent limit
159         if(voltage_meas>40 || current < 0)
160         {
161                 send_current =0;
162                 current = 0;
163         }
164         //Check overflow
165         else if(current>65535){
166                 send_current = 65535;
167                 current = 65535;
```

```
168         }
169         else
170         {
171                 send_current = current;
172         }
173
174
175         new_cur = (uint16_t) (send_current);
176
177         data[8] = new_cur&0xff;
178         data[9] = (new_cur>>8)&0xff;
179
180
181
182
183
184         return;
185 }
186
187 float calc_vol(uint8_t *data){
188
189         int16_t vol;
190         float vref = 5.047;
191         float vsup = 20;
192         float heat_transfer = vref*15.4679/(pow(2,15));
193         float voltage_meas;
194
195         vol = (int16_t) ((data[1] << 8) | data[0]);
196
197         voltage_meas = vol*heat_transfer-0.01253*vsup;
198
199         return voltage_meas;
200
201
202 }
203
204 float calc_cur(uint8_t *data){
205         float vref = 5.047;
206         int16_t cur;
207         float shunt_transfer = vref/(10.2*4.83*pow(2,15));
208         float current_meas;
209
210         cur = (int16_t) ((data[3] << 8) | data[2]);
211
212         current_meas = cur*shunt_transfer;
213
214         return current_meas;
215 }
216
217 void cur_sweep(uint8_t *data){
218
219
220         //sweep = 0x0f00;
221         sweep = sweep+1;
222
223         if(sweep>=65535){
```

```
224                     sweep = 0x0000;
225             }
226
227
228             data[8] = sweep&0xff;
229             data[9] = (sweep>>8)&0xff;
230
231 }
232
233
234 float calc_a(float Kp, float Ki, float Kd, float Ts){
235             float a;
236             a = (Kp+Ts*Ki/2+Kd/Ts);
237             return a;
238 }
239
240
241 float calc_b(float Kp, float Ki, float Kd, float Ts){
242             float b;
243             b = (-Kp+Ts*Ki/2-Kd/Ts);
244             return b;
245 }
246
247 float calc_c(float Kd, float Ts){
248             float c;
249             c = (Kd/Ts);
250             return c;
251 }
```

```
1  /*
2   * calcon.h
3   *
4   *  Created on: May 30, 2019
5   *      Author: Marjolein Rebers
6   */
7
8  #ifndef CALCON_H_
9  #define CALCON_H_
10
11 // Function prototypes
12 void resetcalcvalue();
13
14 float calc_sppower(float sp, uint8_t *data, float a, float b, float c);
15
16 void calc_pow(float sp,uint8_t *data);
17
18 void calc_control(int type, float spres, uint8_t *data, float a, float b, float c);
19
20 float calc_vol(uint8_t *data);
21
22 float calc_cur(uint8_t *data);
23
24 void cur_sweep(uint8_t *data);
25
26 float calc_a(float Kp, float Ki, float Kd, float Ts);
27
```

```
28 float calc_b(float Kp, float Ki, float Kd, float Ts);
29
30 float calc_c(float Kd, float Ts);
31
32 #endif /* CALCON_H_ */
```

C.2. MATLAB code

The MATLAB code can be found in this section.

C.2.1. Plot results LabVIEW

```
1  clear all; close all; clc;
2
3  %% Set constants
4  R0 =191;
5  T0 = 20;
6  alpha = 0.0025;
7
8  %% Create right format
9  A = readtable('Measurements\file.txt');
10 current = A(:,2);
11 curr = table2array(current);
12 voltage = A(:,1);
13 volt = table2array(voltage);
14 time = A(:,3);
15 time = table2array(time);
16
17
18 %% Calculate resistance, temperature, power
19 R = volt./curr;
20 T =(((R/R0)-1)/alpha)+T0;
21 P = curr.*volt;
22 st = time(end)/length(time);
23 f = 1/st
24 %% Plot
25 subplot(3,1,1)
26 title('Current and Voltage')
27 xlabel('time [s]')
28 yyaxis left
29 plot(time,curr)
30 ylabel('Current [A]')
31 ylim([0,5E-2])
32 yyaxis right
33 plot(time,volt)
34 ylabel('Voltage [V]')
35
36 subplot(3,1,2)
37 title('Resistance and temperature')
38 xlabel('time [s]')
39 yyaxis left
40 plot(time,R)
41 ylabel('Resistance [Ohm]')
42 yyaxis right
43 plot(time,T)
44 ylabel('Temperature [degree celcius]')
45
```

```
subplot(3,1,3)
plot(time,P);
xlabel('time [s]')
ylabel('Power [W]')
title('Power')
```

C.2.2. MCU control

```
close all; clear all; clc;

%% begin values

% time
times = 10; %time in seconds


%plot
R0 = 735;   %resistance at T0
T0 = 20;
alpha = 0.0024;

%% Open com
s = serial('COM5');
set(s,'BaudRate',115200);
fopen(s)

%% send type
type = 1; %0 = power PID, 1 = Powercalc, 2 = currentsweep, 3 = resistancePID 4=
    resistance
fprintf(s, ['t', char(type), 13]) % 13 = '/r' and creates end of line with standard
    '/n' that is send
pause(0.001);

%% send setpoint
if (type <= 2)
sp = 0.1;
dt = 0.001;
Kp = 0.01;
Ki = 0.1;
else
Kp = 0.01;
Ki = 0.00001;
SPT = 30;%set point of temperature
sp = single(R0 * (1 + alpha * (SPT - T0)));
dt = 0.01;
end


sp = single(sp);
sp = typecast(sp, 'uint8');
fprintf(s,['s', char(sp(1)),char(sp(2)), char(sp(3)), char(sp(4)), 13])
pause(0.001);

%% send Kp

Kp = single(Kp);
Kp = typecast(Kp, 'uint8');
```

```
48 fprintf(s,['p', char(Kp(1)),char(Kp(2)), char(Kp(3)), char(Kp(4)), 13])
49 pause(0.001);
50
51 %% send Ki
52 Ki = single(Ki);
53 Ki = typecast(Ki, 'uint8');
54 fprintf(s,['i', char(Ki(1)),char(Ki(2)), char(Ki(3)), char(Ki(4)), 13])
55 pause(0.001);
56
57 %% send Kd
58 Kd = 0;
59 Kd = single(Kd);
60 Kd = typecast(Kd, 'uint8');
61 fprintf(s,['d', char(Kd(1)),char(Kd(2)), char(Kd(3)), char(Kd(4)), 13])
62 pause(0.001);
63
64 %% read
65
66 % send sync/start signal
67 fprintf(s,['r', 13])
68
69 %%
70 samples = times/dt;
71 for j = 1:samples
72     for i =1:5
73         data = uint8(fread(s, 2));
74         dataint(i) = (typecast(uint8(data), 'int16'));
75
76     end
77     voltage(j) = dataint(1);
78     current(j) = dataint(2);
79     pressure1(j) = dataint(3);
80     pressure2(j) = dataint(4);
81     send_dac(j) = dataint(5);
82
83 end
84
85 %%
86 % send exit/off signal
87 fprintf(s, ['x', 13])
88
89 %% close com
90 fclose(s)
91 delete(s)
92 clear s
93
94 %% save matlab workspace
95 save('measure.mat', 'voltage', 'current', 'send_dac', 'dt' , 'sp' , 'Kp', 'Ki', 'Kd'
       );
96
97
98 %% Plot
99 V_sup = 14;
100 vref = 5.047;
101 shunt_transfer = vref/(50*2^15);
102 heat_transfer = vref*16.155/(2^15); %old measurements
```

```
%heat_transfer = vref*15.4679/(2^15);
cur_transfer = 1.043/(10*2^16);
%vol = double(voltage)*heat_transfer-0.01253*V_sup;
vol = double(voltage)*heat_transfer-0.089654*V_sup; %old measurments
cur = double(current)*shunt_transfer;
cur_dac = double(typecast(int16(send_dac), 'uint16')) * cur_transfer;



plotfactor = 10; %reduces points in plot, this enables to save plots without error

vol = vol(1:plotfactor:end);
cur = cur(1:plotfactor:end);
cur_dac = cur_dac(1:plotfactor:end);

pow = vol.*cur;
res = vol./cur;
temp =(((res./R0 )-1)./alpha)+T0;


t = [0:dt*plotfactor:(length(vol)-1)*dt*plotfactor];



plot(t,vol)
title('Voltage')
xlabel('time [s]')
ylabel ('[V]')

figure
plot(t, cur)
title('Current')
xlabel('time [s]')
ylabel ('[A]')
hold on
ylim([0,0.12])
plot(t, cur_dac)


figure
plot(t, vol)
title('Measured voltage and current')
xlabel('time [s]')
yyaxis left
ylabel ('[V]')
hold on
yyaxis right
ylim([0,0.12])
ylabel ('[A]')
plot(t, cur)
legend('Voltage', 'Current')

figure
plot(t, pow)
title('Power and supply current')
xlabel('time [s]')
```

```
159 yyaxis left
160 %ylim([0,0.12])
161 ylabel ('[W]')
162 hold on
163 yyaxis right
164 ylim([0,0.12])
165 ylabel ('[A]')
166 plot(t, cur_dac)
167 legend('Power', 'Supply current')
168
169
170
171 figure
172 plot(t, res)
173 title('Resistance and temperature')
174 xlabel('time [s]')
175 yyaxis left
176 ylim([0,1000])
177 ylabel ('[Ohm]')
178 hold on
179 yyaxis right
180 ylabel('[A]')
181 plot(t,cur_dac)
182 ylim([0,0.12])
183 legend('Resistance', 'Temperature')
184
185 figure
186 yyaxis left
187 plot(t, temp)
188 title('Temperature')
189 xlabel('time [s]')
190 ylim([0,50])
191 ylabel ('degree celcius')
192 hold on
193 yyaxis right
194 ylim([0,0.12])
195 ylabel ('[A]')
196 plot(t, cur_dac)
197 legend('Temperature', 'Controlled current')
```