

POLITECNICO DI TORINO

Department of Mechanical Engineering



Master's Degree Thesis

Conceptual design, implementation, and testing
of a water vapor supply control for the purpose of
a full automated Solid Oxide Cell test rig

Politecnico di Torino's Supervisor

Prof. MASSIMO SANTARELLI

Leibniz Universität Hannover's Supervisors

Prof. STEPHAN KABELAC

M.Sc. JAN HOLLMANN

Candidate

SEYEDALI AYATI

A.Y. 2020/2021

Abstract

The Europe's energy distribution grid's continuing development makes finding innovative ways to energy storage imperative. One promising approach is the reversible solid oxide cell (ReSOC), with hydrogen as an energy carrier. At its core, a ReSOC is an electrochemical cell that can operate both as an electrolyzer and a fuel cell. Empirical research of such cells consists of large parts by conveying electrochemical impedance spectroscopy (EIS) at a wide range of different running conditions.

To improve the current research process, the test rig for the characterization of solid oxide fuel cells and electrolysis cells could be up to date. These characterizations consist of a set of operating points defined by the cell temperature, the fuel/anode gas composition, the cathode gas composition, and the current applied to the cell. These characterizations can be achieved nearly automated using software developed at the institute, except for the water vapor supply.

According to the water vapor supply, the test rig has two main problems. The first one, the water vapor supply, is not automated. The second one, the water vapor volume flow supplied to the test rig, is fluctuating flow; thus, this action directly impacts the Fuel Cell voltage.

As a novelty, the research aims to develop a mechanism to automatically control the water vapor control settings' adaptation to reach the required water volume flow and maintain the stable condition. This mechanism consists of a stepper motor to rotate the valve, the Coupling System, and an Arduino Uno control board to control the stepper motor to its desired position. The control variable is the steam pressure that is to be directed to a target value. The automated valve system runs with the Arduino software (IDE) to control the pressure valve position. The author also uses the Visual Studio code section to connect with Arduino IDE and run the stepper motor. Furthermore, for designing and assembling the Coupling System, the author utilizes SolidWorks software.

Eventually, the author will implement the automated valve system into the test rig, and consequently, the control's functionality will be getting more satisfying and more accurate.

Acknowledgements

First, I would like to express my gratitude to my thesis supervisor in Politecnico di Torino, Prof. Massimo Santarelli, my supervisor in Leibniz Universität Hannover, Prof. Stephan Kabelac, and my mentor, Mr. Jan Hollmann, for their availability and fast responses whenever I needed guidance through all the stages of the thesis. They selflessly shared their knowledge and experiments. They always showed a strong will to support me.

Moreover, this thesis is prepared in the Thermodynamic Institute of Leibniz Universität Hannover. I would like to thank all of the Institute staff, particularly Mrs. Dorit Schulte.

I would like to express my very great appreciation to Mr. Hossein Larki Harchegani, one of the most outstanding engineers in the Thermodynamic Institute of Leibniz Universität Hannover, for being supportive and friend during these eight months abroad. I should say that without his advising, guidance, and extraordinary skills in engineering subjects, I would have never been able to advance this thesis.

I would like to express my very great thankfulness to My friends who supported me during my master's degree, particularly Ehsan Lesani and Erfan Jalilian.

I would like to express my very great gratitude to my transcendent friend Milad Zandevakili. he has made Hannover a great place to work and to explore, making it like a second home for me. I should express that without him, life in Hanover would have been very difficult during the Lockdown period due to the Covid-19 pandemic.

In the end, I would like to express my very great gratitude to my beloved parents and my all-embracing brother, who have supported me selflessly during my growing up and education. I should say that I always owe them my entire life.

“Seyed Ali Ayati”
Germany, Hanover February 2021

Table of Contents

List of Figures	VI
List of Tables	VIII
Nomenclature	IX
1 Introduction	1
2 Fundamentals	3
2.1 Solid Oxide Cells	3
2.1.1 Operating Principle	3
2.1.2 Open Circuit Operation	5
2.1.3 Closed Circuit Operation	6
2.1.4 Loss Mechanisms	7
2.2 Current/Voltage Characteristics	9
2.3 Electrochemical Impedance Spectroscopy	11
2.3.1 Methodology	11
2.3.2 Linearity of Electrochemistry Systems	13
2.3.3 Steady State Systems	14
2.3.4 Equivalent Electrical Circuit	14
2.3.5 Distribution of Relaxation Times (DRT)	15
2.3.6 Analysis of DRTs	16
2.4 Test rig (Testing Station)	17
2.4.1 Basic Solid Oxide Fuel Cell Test Station Requirements	17
2.4.2 Test Conditions	18
2.5 State of Art	19
2.5.1 Review of the institute testing station characteristics	19
2.5.2 The aim of this thesis	21
3 Conceptual Design	23
3.1 components in the experiment	23

3.1.1	Arduino control board	24
3.1.2	Stepper motor driver controller	24
3.1.3	Stepper motor	26
3.1.4	Power supply	27
3.1.5	Jumper cable Arduino	28
3.2	CAD-Design	28
3.2.1	The methodology of the Coupling System design	28
3.2.2	The modeling of Coupling system by SolidWorks	30
3.3	Assembly the Coupling System and the other components	32
4	Implementation	34
4.1	Developing software (Control)	34
4.1.1	Arduino Software (IDE)	34
4.1.2	Arduino board control with Visual Studio	37
4.2	Building the valve automation	42
4.3	Integrating automated valve into the test rig	43
5	Future Research	46
5.1	PID controller	46
5.2	Feedforward controller	47
5.3	Arduino IDE codes for the automatic control system	48
5.4	Water supply pressure sensor into test rig	52
5.5	Limitations of this experiment.	53
6	Conclusion	54
A	Coupling System Drawing	55
B	The Arduino Sketch	56
C	Visual Studio Codes	60
	Bibliography	65

List of Figures

2.1	Schematic of a solid oxide cell, operating as both a fuel cell (left) and an electrolyzer (right) [1] [11] [12]	4
2.2	Typical U/j -characteristic of a solid oxide cell [1] [11] [12]	7
2.3	Schematic plot of voltage versus current density of a SOFC showing different types of polarisations: activation polarisation is dominant at low current densities; diffusion polarisation is dominant at high current densities when the transport of reactive species to the electrolyte / electrode interface becomes a limiting factor for the cell reaction [18] [19] [8]	10
2.4	Basic experimental setup for the impedance measurement of a real SOFC with an internal impedance Z_{cell} [22].	12
2.5	Corresponding I-U curve. A sinusoidal current of small amplitude $i(t)$ is superposed to a defined bias current I_{load} and the voltage response $u(t)$ is measured [22].	12
2.6	Typical Nyquist-plot recorded on a real anode supported SOFC single cell. The high frequency intercept (for $\omega \rightarrow \infty$) with the real axis corresponds to the purely ohmic resistance R_0 . The difference between the low and high frequency intercept is the so-called polarisation resistance R_{pol} of the cell [22].	13
2.7	Current versus Voltage Curve Showing Pseudo-Linearity [23].	14
2.8	Distribution of relaxation times and integral of distribution function derived from the impedance spectrum shown in Fig. 2.6 [8] [11] [12].	17
2.9	Schematic of the FuelCon Evaluator-HT testing station used in the experiment [11] [12].	20
2.10	Schematic of the cell housing, including electric contacts as well as gas supply and removal [11] [12].	21
3.1	Image of Aurduino Uno Board [27].	24
3.2	Image of L298N Stepper Motor Driver Controller with the description of its elements.	25
3.3	Image of the driving stepper with H-Bridge.	25

3.4	Image of stepper motor nema14-01.	26
3.5	The schematic of the A+, A-, B+, and B- wires.	26
3.6	The illustration of NEMA 14-1 Stepper Motor in connection with L298N and Arduino.	27
3.7	The image of Mean Well - 12V/1.25A Rail Din power supply used in this experiment.	27
3.8	The image of the MAKER FACTORY JKMF40 Jumper cable Arduino used in this experiment.	28
3.9	The image of Bellows-Sealed Metering Valves manufactured by Swagelok company [28].	28
3.10	The illustration of the pressure valve and its elements with description [28].	29
3.11	The image of modeling of the Coupling System on the view of close to the stepper motor shaft in SolidWorks.	30
3.12	The image of modeling of the Coupling System on the view of close to the pressure valve head in SolidWorks.	31
3.13	The image of a close view of the Coupling System after manufacturing.	31
3.14	The image of the electrical components are assembled	32
3.15	The image of the entire components are assembled	33
4.1	The image of the design surface of this windows form	41
4.2	The image of the LED pin 13 on the Arduino board	42
4.3	The image of the valve automation 1	43
4.4	The image of the valve automation 2	43
4.5	The image of the implementation of the automation valve into the testing station	44
5.1	The block diagram of the PID controller [31]	47
5.2	The block diagram of the Feedforward controller [32]	47
5.3	Pressure transmitter model A-10 made by WIKA LLC [34]	52
A.1	The image of the Coupling System Drawing	55
B.1	The view of the Arduino sketch for directly changing the stepper motor position.	56
B.2	The view of the Arduino sketch for directly changing the stepper motor position with sending the command from Visual Studio 2019.	57
B.3	The view of the Arduino sketch for automatically control the position of the stepper motor by PID controller (part 1).	58
B.4	The view of the Arduino sketch for automatically control the position of the stepper motor by PID controller (part 2).	59

List of Tables

5.1	Examples of data-sheet include steam pressure, steam volume flow, and corresponding valve position in different moments, which have already been measured.	48
-----	--	----

Nomenclature

A	Area
F	Faraday Constant
f_k	Fugacity of Species k
G	Gibbs Free Energy
$\Delta^R G$	Free Reaction Energy
j	Electric Current Density
p	Pressure
p^\ominus	Standard Pressure
R_m	General Gas Constant
Q	Electric Charge
T	Temperature [K]
t	Time
U	Voltage
U_{cell}	Cell Voltage
$U_{cell,0}$	Open Circuit Cell Voltage
z	Amount of Elementary Charges Carried
x_k	Mole fraction of any individual gas component in a gas mixture
p_k	Partial pressure of any individual gas component in a gas mixture
n_k	Moles of any individual gas component in a gas mixture

p_{tot}	Total pressure of the gas mixture
n_{tot}	Total moles of the gas mixture
e	Error Signal
K_p	Proportional Constant
K_I	Integral Constant
K_D	Derivative Constant
Y	Output of The PID Controller
η	Cell Overpotential
ν	Stoichiometric Coefficient
$()^\ominus$	With Regard to Standard Pressure
k	Species k

Chapter 1

Introduction

The Europe's energy distribution grid's continuing development is unique challenges: Due to the prolonged usage of energy vehicles with highly unpredictable yield, such as solar energy and wind, it is crucial to repay for alternating energy supply. For this purpose, novel approaches to energy storage are strongly inquired. One method of achieving such compensation is to use excess electrical energy for hydrogen electrolysis from water to convert electrical into chemical energy. This hydrogen can be stored until energy demand exceeds the supply and then be used to power a fuel cell, hence providing electrical energy to the grid.

One assuring method of using hydrogen as energy storage is the reversible solid-oxide cell (ReSOC). At its core, a ReSOC is an electrochemical cell that can operate both as an electrolyzer and a fuel cell. It aids the oxyhydrogen reaction [1].



The reaction's electrolysis direction (EC) is endothermic and needs an electrical energy input to occur [1]. In contrast, the fuel cell direction (FC) is exothermic and can transform chemical into electrical energy [1]. Storage systems employing the ReSOC principle are estimated to be very efficient in converting energy as well as economically viable [2]. However, before being used on a commercial scale, there are still technical challenges to be spoken.

The two most significant of these challenges are the ReSOC's long-term stability and energy conversion efficiency. To determine favorable system configurations on a commercial scale, modeling approaches must be practiced [3][4]. If such large-scale models predict the behavior of entire stacks of ReSOCs correctly, they first need to be able to portray single cell performance accurately. SOC performance is very dependent on the operating conditions, which are defined by multiple parameters. Furthermore, the physical processes governing the cell's behavior are not still wholly known. Therefore involved, partly empirical model equations are applied [5].It

means empirical coefficients need to be observed for models, and they must be approved for a wide variety of operating conditions. As a result, extensive empirical research is essential, as measurements have to be repeated at many different combinations for the parameters defining the cell's operating conditions.

For this purpose, the Thermodynamics institute of Leibniz Universität Hannover employs a high-temperature testing station designed explicitly for housing ReSOCs, as well as an impedance spectroscope. Electrochemical impedance spectroscopy (EIS) has established an invaluable tool for in-situ examination of electrochemical cells, particularly when combined with an analysis of the resulting spectra's distribution of relaxation times (DRT) [6] [7] [8] [9]. As electrochemical cell behavior is dependent on a multitude of different parameters, empirical research regularly includes EIS measurements attended at a large number of other operating conditions. To improve the current research process, the test rig for the characterization of solid oxide fuel cells and electrolysis cells could be up to date. These characterizations consist of a set of operating points defined by the cell temperature, the fuel/ anode gas composition, the cathode gas composition, and the current applied to the cell. These characterizations can be achieved nearly automated using software developed at the institute, except for the water vapor supply.

According to the water vapor supply, the test rig has two main problems. The first one, the water vapor supply, is not automated. The second one, the water vapor volume flow supplied to the test rig, is fluctuating flow; thus, this action directly impacts the Fuel Cell voltage.

As a novelty, the research aims to develop a mechanism to automatically control the water vapor control settings' adaptation to reach the required water volume flow and maintain the stable condition. This mechanism consists of a stepper motor to rotate the valve, the Coupling System, and an Arduino Uno control board to control the stepper motor to its desired position. The control variable is the steam pressure that is to be directed to a target value. The automated valve system runs with the Arduino software (IDE) to control the pressure valve position. The author also uses the Visual Studio code section to connect with Arduino IDE and run the stepper motor. Furthermore, for designing and assembling the Coupling System, the author utilizes SolidWorks software.

Eventually, the author will implement the automated valve system into the test rig, and consequently, the control's functionality will be getting more satisfying and more accurate. Also, in chapter 2, the author will demonstrate the basic definitions in solid oxide fuel cell and state of the art; in chapter 3, he will describe the Arduino control board, stepper motor, and the CAD-Design section; in chapter 4, he will define the Arduino program, the Visual Studio program and the implementation of the automated valve system into the test rig; in chapter 5; he will represent the automatic control program and the limitation of this research; in chapter 6, he will explain the summary all actions done in this thesis.

Chapter 2

Fundamentals

2.1 Solid Oxide Cells

2.1.1 Operating Principle

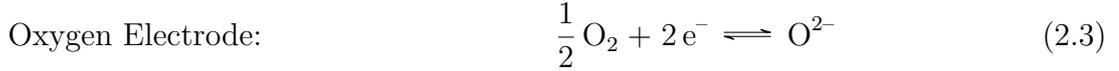
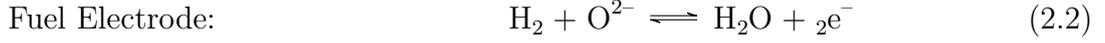
Solid oxide cells are electrochemical cells used as either a fuel cell (SOFC) or an electrolytic cell (SOEC) [1]. That means they can convert chemical energy stored in a fuel into electrical power and convert electrical into chemical energy [1]. Because of this capability, these cells are also called reversible solid oxide cells (ReSOC) [1]. However, this does not mean that they are thermodynamically reversible, but only that their operating principle can be reversed. SOFCs can be used with a multitude of fuels, including hydrogen and different natural gasses [1]. Conversely, SOECs can use water and carbon dioxide for electrolysis [1]. In the case of hydrogen and water as fuel, the redox reaction taking place in the cell is given by



Here FC denotes the fuel cell direction and EC the electrolysis direction of the reaction. Use the term fuel cell; the fuel cell reaction is the forward; the electrolysis reaction is the reverse reaction [1]. The operation principle in an electrochemical cell is that the oxidation and reduction reactions occur spatially separated at the electrodes [1]. An electrode is called Anode if the oxidation reaction occurs there and Cathode if it is the reduction site. The electrolyte provides the separation of Anode and Cathode [1]. In every electrochemical cell, the electrolyte serves several important purposes: The separation of the reacting species, the electrical isolation of both electrodes, and the transport of ions [1]. This is fundamental for the cell to work, as the charge is transported through the electrolyte by ions while electrons cannot pass it [1]. Instead, they can be transported via an electric circuit outside the cell and used to provide electric power [1]. Solid oxide cells typically employ

an electrolyte that allows passage to O^{2-} ions. Fig. 2.1 illustrates the operating principle for SOFC and SOEC mode [1].

As the reaction can proceed in both directions, both electrodes can be either the Anode or the Cathode. Therefore, it is customary to designate the electrodes as fuel and oxygen electrode when talking about reversible cells. The partial reactions taking place at each electrode are [10]



When in the fuel cell mode, the oxygen electrode acts as the Cathode. Here, molecular oxygen is combined with electrons to form O^{2-} ions. These can pass through the electrolyte to the fuel electrode, which is the Anode in that case [1]. At the Anode, molecular hydrogen is oxidized by the O^{2-} ions, resulting in water and free electrons [1]. As the electrons cannot permeate the electrolyte layer, they are conducted by an external electric circuit [1]. Because of the electrochemical potential difference between the electrodes, an electrical field across the cell occurs. The electrons in the external circuit can then be used to perform electrical work. The process is reversed in electrolysis mode: An external power supply generates an electric field that forces electrons to the fuel electrode, now acting as the Cathode [1]. At the Cathode, the electrons reduce water, forming hydrogen and O^{2-} ions. The

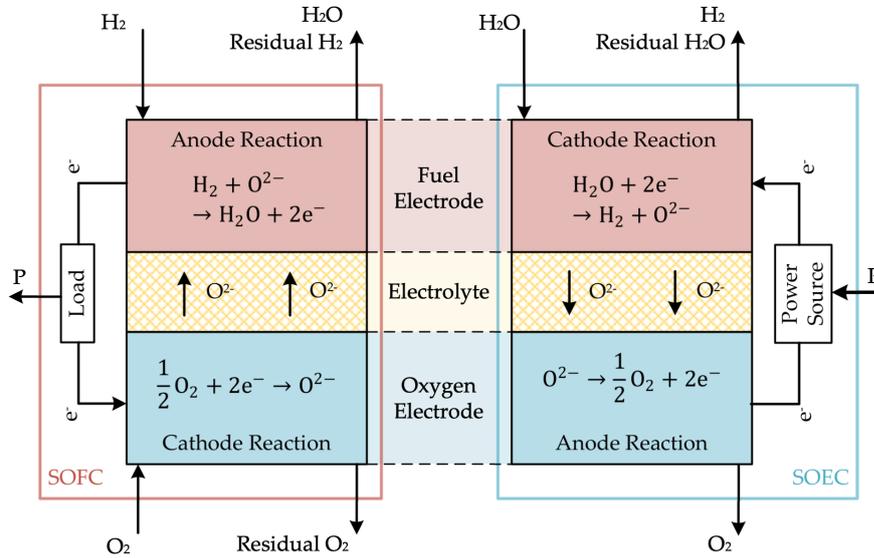


Figure 2.1: Schematic of a solid oxide cell, operating as both a fuel cell (left) and an electrolyzer (right) [1] [11] [12]

O^{2-} ions move through the electrolyte to the oxygen electrode, where they are oxidized, creating oxygen and electrons [1]. This way, electrical energy can be converted to chemical energy stored within the separated fuel and oxygen [1].

It is important to note that the cell reactions can only occur at the so-called triple phase boundaries (TPBs). A triple-phase boundary is a place where the solid electrolyte and electrode phases and the gas phase meet. Because each reaction includes species conducted in only one of the three phases, the reactions can't take place anywhere else [1].

Electrodes and electrolyte must match specific criteria for the cell processes to work. Solid oxide cells employ a solid ceramic electrolyte, typically made of yttria-stabilized zirconia (YSZ) (cf. O'HAYRE [1]). YSZ is permeable to O^{2-} , which is the most typical charge conductor for SOCs [1]. However, this conductivity is temperature-dependent, and use in electrochemical cells is only feasible at temperatures above 600 °C [1]. This results in typical operating temperatures of more than 800 °C, which means that all materials used must be resistant to heat. Additionally, electrolyte and oxygen electrode must withstand the oxidizing environment created by the use of oxygen at such temperatures [1].

The electrodes must also have a high ionic and electrical conductivity and increased permeability to the reactant gases since a lack of either would impede the electrochemical reaction. For the fuel electrode, the materials commonly used are porous metal-ceramic mixtures (cermets) [1]. In a cermet, the ceramic component provides ionic conductivity and metal electrical conductivity [13]. The most common oxygen electrode materials include lanthanum strontium manganite (LSM) and Lanthanum strontium cobalt ferrite (LSCF) [13].

2.1.2 Open Circuit Operation

An electrochemical cell with no electrical connection between the electrodes is referred to as being in open circuit operation. In open-circuit conditions, the electrochemical potential between the electrodes is in equilibrium [1]. In this state, the fuel cell's reaction rates and the electrolysis reaction are precisely equal, resulting in a net reaction rate of zero. It is noteworthy, however, that both reactions still take place. The maximum voltage between the electrodes of a thermodynamically reversible electrochemical cell at a given set of parameters and in open circuit conditions is the reversible cell voltage $U_{cell,0}$ [1]. It is strongly dependent on the redox reaction taking place in the cell: Assuming constant temperature T and pressure p , the relation between chemical and electrical work in a reversible electrochemical system is given by

$$W_{el}^{rev}(p, T) = -\Delta^R G(p, T) \quad (2.4)$$

with the electrical work W_{el} and Gibb's free energy difference of the underlying redox reaction $\Delta^R G$ [1] Electrical work is defined as

$$W_{el} = \Delta\Phi_{el} \cdot Q \quad (2.5)$$

with the electrical potential difference $\Delta\Phi_{el}$ and the electric charge Q . The charge of n moles of a species is dependent on the number of elementary charges one molecule carries z and Faraday's constant F :

$$Q = n \cdot z \cdot F \quad (2.6)$$

Combining Eqs. (2.4) through (2.6) and using the molar free energy difference $\Delta^R G_m = \Delta^R G/n$, one can obtain the maximum difference in electric potential between the electrodes, which equals the voltage between the electrodes. This voltage is called the reversible cell voltage at open circuit $U_{cell,0}$. It is customary to only include the temperature dependency and give the reversible cell voltage with respect to standard pressure p^\ominus :

$$U_{cell,0}(p^\ominus, T) = U_{cell,0}^\ominus(T) = \Delta\Phi_{el}^\ominus(T) = \frac{\Delta^R G_m^\ominus(T)}{zF} \quad (2.7)$$

To account for the effects of pressure and gas composition, the Nernst equation can be derived from the condition of electrochemical equilibrium [1]:

$$U_{cell,0} = U_{cell,0}^\ominus(T) - \frac{R_m T}{zF} \ln \left(\prod_k \left(\frac{f_k}{p^\ominus} \right)^{\nu_k} \right) \quad (2.8)$$

Here, R_m is the molar gas constant, f_k the fugacity, and ν_k the stoichiometric coefficient of the species k . For ideal gases, the fugacity becomes $f_k = p_k$, and the Nernst equation is simplified to

$$U_{cell,0} = U_{cell,0}^\ominus(T) - \frac{R_m T}{zF} \ln \left(\prod_k \left(\frac{p_k}{p^\ominus} \right)^{\nu_k} \right) \quad (2.9)$$

2.1.3 Closed Circuit Operation

Electrons can flow from one electrode to the other when closing the electric circuit, and a net flow of ions through the electrolyte occurs [1]. The electric current I is defined as charge flowing per time t . To compare different cells, the electrical current is usually expressed as the current density j [1]:

$$j = \frac{I}{A_{cell}} = \frac{1}{A_{cell}} \cdot \frac{dQ}{dt} = \frac{\dot{n}zF}{A_{cell}} \quad (2.10)$$

with the cell area A_{cell} and the charge carrier flow \dot{n} . At a net current density of $j = 0$, the circuit is open, and the cell voltage $U_{cell,0}$ is present across the cell [1]. Because of that, $U_{cell,0}$ is also called open-circuit voltage (OCV) [1]. Increasing the current density leads to a voltage drop caused by different cell loss mechanisms, such as ohmic losses, transport of substances, and the cell's reaction kinetics. These current density-dependent voltage losses are called overvoltages η_k [1]. The current density-dependent cell voltage in fuel cell mode can therefore be expressed as

$$U_{cell}(j) = U_{cell,0} - \sum_i |\eta_k(j)| \quad (2.11)$$

To achieve a negative current density, resulting in electrolysis, a higher voltage than the OCV must be externally applied to the cell [1]. For solid oxide cells at high temperatures, this relation between cell voltage and current density is nearly linear, as Fig. 2.2 illustrates [1].

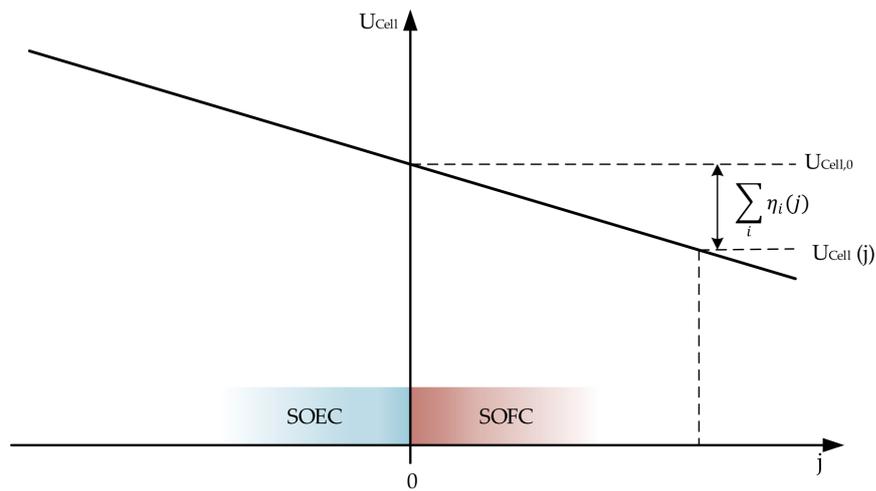


Figure 2.2: Typical U/j -characteristic of a solid oxide cell [1] [11] [12]

2.1.4 Loss Mechanisms

When a fuel cell is loaded with an electric current the cell voltage drops below the thermodynamically predicted (Nernst-voltage). This is due to several internal irreversible loss mechanisms. In the following subchapters a short description of these losses will be given [8].

Ohmic Losses

Ohmic losses occur during the electronic or ionic transport through the electrodes and the electrolyte. The overall ohmic resistance is the sum of each individual ohmic contribution R_k . According to Ohm's law, the ohmic overpotential linearly increases with the current density j [8].

$$\eta_{ohm} = j \cdot \sum_{i=k} R_k = j \cdot R_{ohm} \quad (2.12)$$

In solid oxide cells, the main contributors to the ohmic overpotential are the electrode's resistance to electron transport and the electrolyte's resistance to ion transport [8].

Activation Loss

Activation polarisation describes the electrochemical loss mechanisms taking place mainly at the three-phase boundary (TPB) where ionic-conducting, electronic-conducting and gas phase meet. An activation energy is necessary in order to overcome the energy barrier that prevents a spontaneous reaction. The higher the temperature, the higher the probability for reactants to gain the necessary activation energy, therefore the overpotentials are reduced. A commonly used equation for describing the influence of activation overpotential on current density is the well known Butler-Volmer equation [14]

$$j = j_0 \left(\exp\left(\frac{\alpha z F \eta_{act}}{R_m T}\right) - \exp\left(\frac{(1 - \alpha) z F \eta_{act}}{R_m T}\right) \right) \quad (2.13)$$

Here j is the (partial pressure and temperature dependent) exchange current density of anode/cathode, n the number of exchanged electrons, α the apparent charge transfer coefficient, and η_{act} the activation overpotential of the according electrode (anode or cathode). The charge transfer coefficient is an indicator of the symmetry of the activation energy barrier when a positive or negative overpotential is applied [15].

Diffusion Overpotential Loss

High current densities are correlated with an enhanced gas transport and gas conversion in the electrode, which leads to polarisation losses. The so-called diffusion polarisation at the anode results from an undersupply of the three-phase points with fuel. Simultaneously the reaction product (water) is being transported away from the reaction zone too slowly. At the cathode, losses due to diffusion polarisation occur, too. They are caused by an undersupply with oxidizing gas [8].

Calculation of the diffusion polarisation overpotential is based on the Nernst-equation (Eq. 2.9), from which the following Eqs. 2.14 and 2.15 can easily be derived [16]:

$$\eta_{conc,an} = \frac{R_m T}{2F} \ln \left(\frac{p_{H_2O_{an}}^{TPB} \cdot p_{H_2_{an}}}{p_{H_2O_{an}} \cdot p_{H_2_{an}}^{TPB}} \right) \quad (2.14)$$

$$\eta_{conc,cat} = \frac{R_m T}{4F} \ln \left(\frac{p_{O_2_{cat}}}{p_{O_2_{cat}}^{TPB}} \right) \quad (2.15)$$

$\eta_{conc,an}$ for Anode and $\eta_{conc,cat}$ for Cathode describe the overpotential resulting from a partial pressure difference between gas atmosphere and TPB. $p_{H_2O_{an}}$, $p_{H_2_{an}}$ and $p_{O_2_{cat}}$ are the known partial pressures of hydrogen, water and oxygen, respectively, in the gas channel. The unknown partial pressures at the TPB, $p_{H_2O_{an}}^{TPB}$, $p_{H_2_{an}}^{TPB}$ and $p_{O_2_{cat}}^{TPB}$, are determined by applying Fick's first law and assuming a linear concentration gradient as a function of the current density j . In this way the following Eqs. 2.16 and 2.17, which relate the diffusion-based voltage drop η_{conc} density j , are obtained [17]:

$$\eta_{conc,an} = \frac{R_m T}{2F} \ln \left(\frac{1 + \frac{R_m T L_{an}}{2F D_{H_2O}^{eff} p_{H_2O,an} p^\ominus} \cdot j}{1 - \frac{R_m T L_{an}}{2F D_{H_2}^{eff} p_{H_2,an} p^\ominus} \cdot j} \right) \quad (2.16)$$

$$\eta_{conc,cat} = \frac{R_m T}{4F} \ln \left(\frac{1}{1 - \frac{R_m T L_{cat} \left(1 - \frac{p_{O_2_{cat}}}{p^\ominus} \right)}{4F D_{O_2}^{eff} p_{O_2,cat} p^\ominus} \cdot j} \right) \quad (2.17)$$

L_{an} and L_{cat} denote the effective thickness of the diffusion layer on the anode and cathode side respectively. D_i^{eff} is the effective molecular diffusion coefficient, which is a function of the microstructural properties (pore-size, porosity and tortuosity) of the underlying diffusion layer.

2.2 Current/Voltage Characteristics

The effect of the different loss mechanisms, discussed above, on the actual voltage output of a real SOFC during loading is shown qualitatively in the following Fig. 2.3. As can be seen, even at open circuit condition the cell voltage (U_{OCV}) is less than the thermodynamically predicted Nernst-voltage U_{th} . The difference $U_{OCV} - U_{th}$ is called the ‘‘Overpotential’’. This first drop may be caused by different parasitic losses, like for example undesired electron leaks across the electrolyte or even not perfectly gastight electrolytes. These causes induce an unwanted fuel-utilization

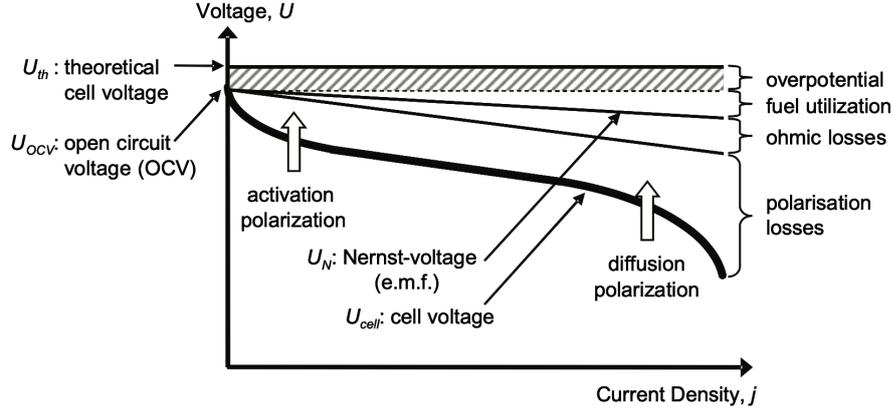


Figure 2.3: Schematic plot of voltage versus current density of a SOFC showing different types of polarisations: activation polarisation is dominant at low current densities; diffusion polarisation is dominant at high current densities when the transport of reactive species to the electrolyte / electrode interface becomes a limiting factor for the cell reaction [18] [19] [8]

already at open circuit, thus lowering the Nernst voltage.

A second cell voltage drop caused by the fuel utilization is the so-called gas conversion loss. An increasing current density results in an increased consumption of the fuel and oxidant gases. In the anode gas channel the hydrogen partial pressure decreases while the reaction product water increases. This results in an increasing oxygen partial pressure in the fuel gas mixture (p_{O_2an}). For the same reason the cathode gas becomes depleted in oxygen (p_{O_2cat} decreases). As a consequence the Nernst-voltage U_N , which arises between cathode and anode will decrease by increasing current density, following Eq. 2.18. U_N represents the driving force for the overall cell reaction and is therefore also called the “electromotive force” [20].

$$U_N = \frac{R_m T}{4F} \ln \left(\sqrt{\frac{p_{O_2cat}}{p_{O_2an}}} \right) \quad (2.18)$$

The two remaining losses responsible for the characteristic shape of the current voltage curve (I-U curve) depicted in Fig. 2.3 are the ohmic and polarisation losses. The polarisation loss is the sum of the activation and diffusion polarisation.

The strong (nonlinear) voltage drop at low current density is mainly caused by activation losses taking place at the TPB of both electrodes. In the medium current range the overall loss is dominated by the ohmic overpotential loss, therefore a more or less linear decrease of the cell voltage with increasing current density is observed. At high current densities, the voltage output of the fuel cell once again drops rapidly due to mass-transport limitations at the electrodes (gas diffusion

polarisation).

2.3 Electrochemical Impedance Spectroscopy

2.3.1 Methodology

In order to develop a physico-chemical model of SOFC single cells and to refine their efficiency and long term stability the performance-related polarisation processes have to be identified and proven. In contrast to I-U curves, where only the overall loss of a cell can be identified, the electrochemical impedance spectroscopy (EIS) is one of the most promising methods for unfolding complex electrochemical systems such as a SOFC [8].

The EIS method takes advantage from the fact that the polarisation loss mechanisms taking place in an electrochemical system differ in their characteristic time constant and frequency response [8].

The most common and standard approach to measure the dynamic behaviour (impedance) of an electrochemical system is by applying a sinusoidal current (or voltage) to the interface and measure the phase shift and amplitude, or real and imaginary parts, of the resulting voltage or (current) [8].

Here it should be noted that an impedance is only defined for systems that satisfy the conditions of causality, linearity, and time-invariance. Although many systems, like SOFCs, are usually non-linear, linearity can be assumed when the magnitude of the applied current stimulus is small enough to cause a linear response [21].

For a successful interpretation of impedance curves the measurement data quality is of crucial importance. The quality and amount of information that can be extracted from impedance data is implicitly connected to the noise-level and the compliance of the measured curve with the principles of causality, linearity and stability [8].

A well established method used to assess the consistency and quality of measured impedance spectra is the Kramers-Kronig validation [14]. The Kramers-Kronig relations are integral equations, which constrain the real and imaginary components of the impedance for systems that satisfy the conditions of causality, linearity, and stability [14].

The basic experimental arrangement for impedance measurement is shown in Fig. 2.4. A sinusoidal current of small amplitude $i(t) = i_0 \sin(\omega t)$ is superposed to a defined bias current I_{load} and the sinusoidal voltage response $u(t) = u_0(\omega) \sin[(\omega t + \phi(\omega))]$ is measured (Fig. 2.5). From the ratio between the complex variables of voltage and current the impedance is calculated as follows [22]:

$$\underline{Z}(\omega) = \frac{\underline{u}(t)}{\underline{i}(t)} = \frac{u_0(\omega)}{i_0} e^{j\phi(\omega)} = |\underline{Z}(\omega)| e^{j\phi(\omega)} = Re\{\underline{Z}(\omega)\} + jIm\{\underline{Z}(\omega)\} \quad (2.19)$$

Where $\omega = 2\pi f[s^{-1}]$ represents the angular frequency and $\phi(\omega)$ the frequency dependent phase shift between voltage and current. This measurement is generally

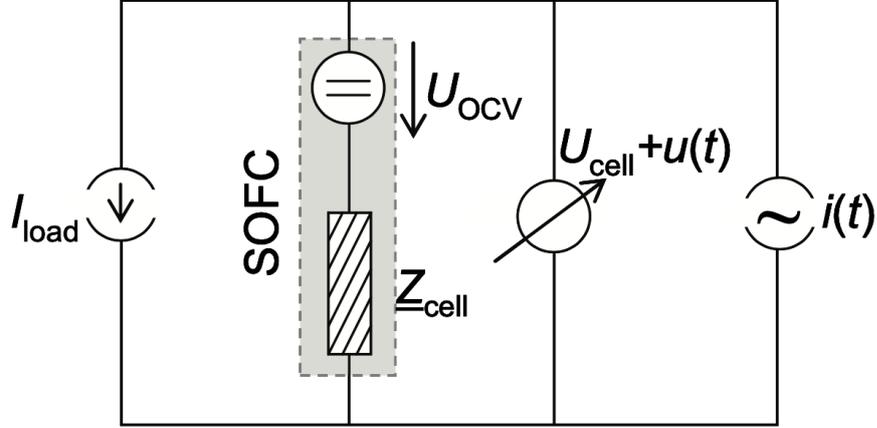


Figure 2.4: Basic experimental setup for the impedance measurement of a real SOFC with an internal impedance Z_{cell} [22].

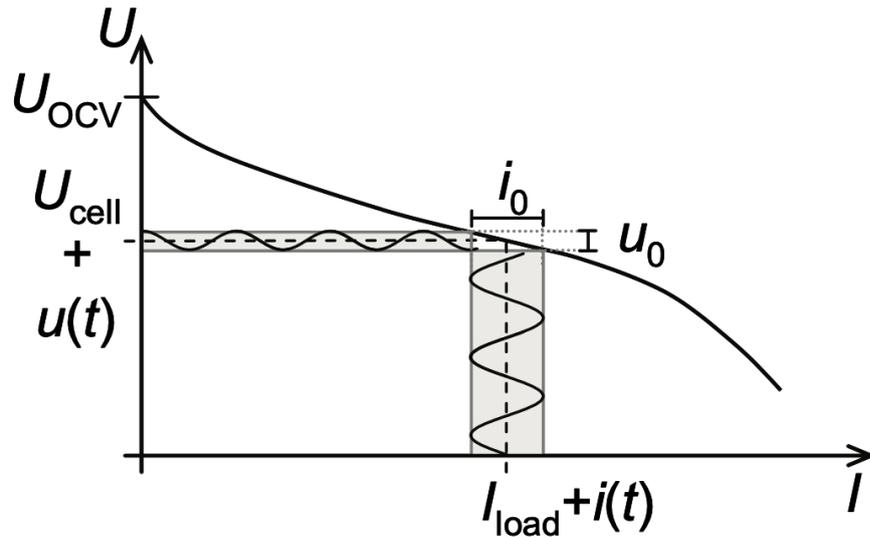


Figure 2.5: Corresponding I-U curve. A sinusoidal current of small amplitude $i(t)$ is superposed to a defined bias current I_{load} and the voltage response $u(t)$ is measured [22].

performed for a discrete quantity of frequency values in a defined frequency range and the recorded impedance values are usually plotted in the complex plane. The

resulting curve is also known as Nyquist-plot. Fig. 2.6 gives an example for a Nyquist plot measured on an anode supported SOFC single cell. The high frequency intercept (for $\omega \rightarrow \infty$) with the real axis corresponds to the purely ohmic resistance R_0 of the cell, whereas the intercept at the lower frequency (for $\omega \rightarrow 0$) is identical to the differential resistance which can be obtained from the corresponding I-U characteristic at the given operating point. The difference between the low and high frequency intercept is the so-called polarisation resistance R_{pol} of the cell. R_{pol} is the sum of each single polarisation resistance caused by the loss mechanisms explained in section 2.1.4.

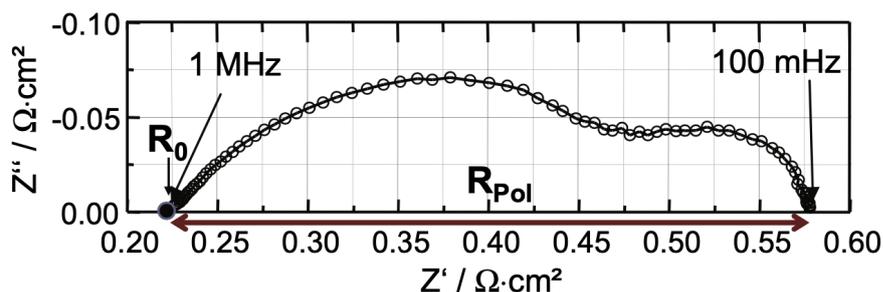


Figure 2.6: Typical Nyquist-plot recorded on a real anode supported SOFC single cell. The high frequency intercept (for $\omega \rightarrow \infty$) with the real axis corresponds to the purely ohmic resistance R_0 . The difference between the low and high frequency intercept is the so-called polarisation resistance R_{pol} of the cell [22].

2.3.2 Linearity of Electrochemistry Systems

Electrical circuit theory distinguishes between linear and non-linear systems (circuits) [23]. Impedance analysis of linear circuits is much easier than analysis of non-linear ones [23]. For a potentiostated electrochemical cell, the input is the potential and the output is the current. Electrochemical cells are not linear! Doubling the voltage will not necessarily double the current [23]. However, Fig. 2.7 shows how electrochemical systems can be pseudo-linear [23]. It appears to be linear to look at a small enough portion of a cell’s current versus voltage curve [23]. In normal EIS practice, a small (1 to 10 mV) AC signal is applied to the cell. With such a small potential signal, the system is pseudo-linear [23].

If the system is non-linear, the current response will contain harmonics of the excitation frequency. A harmonic is a frequency equal to an integer multiplied by the fundamental frequency [23]. For example, the “second harmonic” is a frequency equal to two times the fundamental frequency [23].

Some researchers have made use of this phenomenon. Linear systems should not generate harmonics, so the presence or absence of significant harmonic response

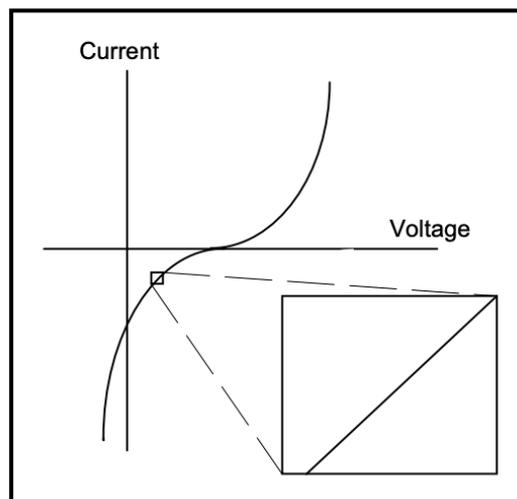


Figure 2.7: Current versus Voltage Curve Showing Pseudo-Linearity [23].

allows one to determine the linearity of the system [23]. Other researchers have intentionally used larger amplitude excitation potentials [23]. They use the harmonic response to estimate the curvature in the cell's current-voltage curve [24].

2.3.3 Steady State Systems

Measuring an EIS spectrum takes time (often up to many hours). The system being measured must be steady throughout the time required to measure the EIS spectrum. A common cause of problems in EIS measurements and analysis is drift in the system being measured [23].

In practice, a steady state can be difficult to achieve. The cell can change through adsorption of solution impurities, growth of an oxide layer, the build-up of reaction products in solution, coating degradation, or temperature changes, to list just a few factors [23].

Standard EIS analysis tools may give wildly inaccurate results on a system that is not at a steady state [23].

2.3.4 Equivalent Electrical Circuit

An approach for quantifying influences of different processes on voltage losses expected in literature is the substitution of the cell's impedance with a number of impedance elements known from electrical engineering. The idea is that a specific impedance element can model each physical process. They are connected as an equivalent circuit diagram; therefore, it results in a complete cell model. Different

elements can be found in literature, such as R, RC-, RQ-, Warburg- and Gerischer Phase-elements [8].

There are two main processes at the reaction layer simultaneously: Firstly, a double layer is formed at the electrode-electrolyte-interface. Secondly, ions pass the interface and take part in the electrode reaction while experiencing a resistance expressed by the Butler-Volmer equation (2.13). The double-layer can be simplified as a capacity C_{act} , as it is effectively an electrical field between two planes. The equivalent resistance of the reaction kinetics can be modeled as the faradaic resistance R_{act} . The faradaic resistance can be obtained by a linearization of Eq. (2.13) [8]:

$$R_{act} = \left. \frac{d\eta_{act}}{dj} \right|_{j=0} = \frac{R_m T}{zF \cdot j_0} \quad (2.20)$$

The electrode-electrolyte interface can therefore be modeled as an RC-element. The complex impedance of an RC-element is

$$\underline{Z}_{RC}(\omega) = \left(\frac{1}{R} + i\omega C \right)^{-1} = \frac{R}{1 + i\omega \cdot RC'} \quad (2.21)$$

where R is the ohmic resistance and C the capacity. Any RC-element is defined by its relaxation time τ_0 , which is defined as

$$\tau_0 = RC \quad (2.22)$$

2.3.5 Distribution of Relaxation Times (DRT)

Calculating the distribution of relaxation times (DRT) from the impedance spectrum of a cell is a method, that allows identifying different processes within the cell by their time-dependent behavior. The basic approach is to approximate the cell as a serial connection of RC-elements. The total polarisation impedance for such a connection is

$$\underline{Z}_{pol}(\omega) = R_{pol} \sum_{n=1}^N \frac{\gamma_n}{1 + i\omega\tau_n'} \quad \text{with} \quad \sum_{n=1}^N \gamma_n = 1 \quad (2.23)$$

where τ_n is the relaxation time of element n and N is the number of RC-elements. In this equation γ_n weighs the contribution of each element to the total impedance. If each RC-element represents a physical process in the cell, γ_n gives a measure of each process's contribution to the total polarization loss [8].

Usually, the number of contributing processes is not known for certain before the analysis. The DRT method addresses this problem by replacing the finite number of RC-elements shown in Eq. (2.23) with an infinite number of infinitesimally small

RC-elements with relaxation times ranging from 0 to ∞ . This approach yields the integral equation

$$Z_{pol}(\omega) = R_{pol} + \int_0^{\infty} \frac{\gamma(\tau)}{1 + i\omega\tau} d\tau \quad \text{with} \quad \int_0^{\infty} \gamma(\tau) d\tau = 1 \quad (2.24)$$

with the distribution function $\gamma(\tau)$. The main difficulty in calculating the DRT is obtaining this distribution function from the discrete data points resulting from measurements. This step is not explained in detail here, as there are finished algorithms available based on different works of literature [8] [6] [25]. The result of a finished DRT analysis is a distribution function $\gamma(f)$, with $f = \omega/(2\pi)$.

2.3.6 Analysis of DRTs

To identify the resistances of individual processes, the distribution function $\gamma(f)$ is used. Fig. 2.8 shows an example of $\gamma(f)$. Each peak indicates accumulations of infinitesimal RC-elements at a particular excitation frequency. Each peak $\gamma_p(f_p)$ in the distribution function shows an individual process taking place at the frequency f_p with the corresponding relaxation time τ_p [8]. Because $\gamma(f)$ weighs the influence of each approach, the area beneath one peak multiplied by the overall polarization resistance R_{pol} gives the individual resistance $R_{pol,k}$ of the process k :

$$R_{pol,k} = R_{pol} \cdot \int_{f_k^-}^{f_k^+} \gamma(f) df \quad (2.25)$$

where f_k^- and f_k^+ are the integration limits that are counted towards process k . To correctly identify these limits, the processes involved must be isolated from one another in the frequency domain. It is usually possible with the DRT method to find individual processes with a resolution of up to half a frequency decade [8]. However, processes that are not separated cannot be evaluated individually. Significantly, up to this point, the processes identified in this way are not associated with the actual physical processes taking place in the cell. Hence, the main challenge to researchers using the DRT is to attribute these processes to physical loss mechanisms correctly. To gain insights into the causes of individual polarization resistance, it is necessary to investigate the relations between the cells operating conditions and the DRT. This means running impedance spectroscopy measurements at each step while varying all essential parameters, i.e., cell temperature and fuel and oxygen electrode gas composition. This way, it is possible to discover which processes correlate strongly with the variation of which parameter. For example, a process strongly dependent on fuel gas composition can be associated with the fuel electrode. At the same time, strong oxygen concentration dependence indicates a process taking place at the oxygen electrode. Additionally, the temperature dependency of a process is telling of its activation energy. Therefore, processes with negligible

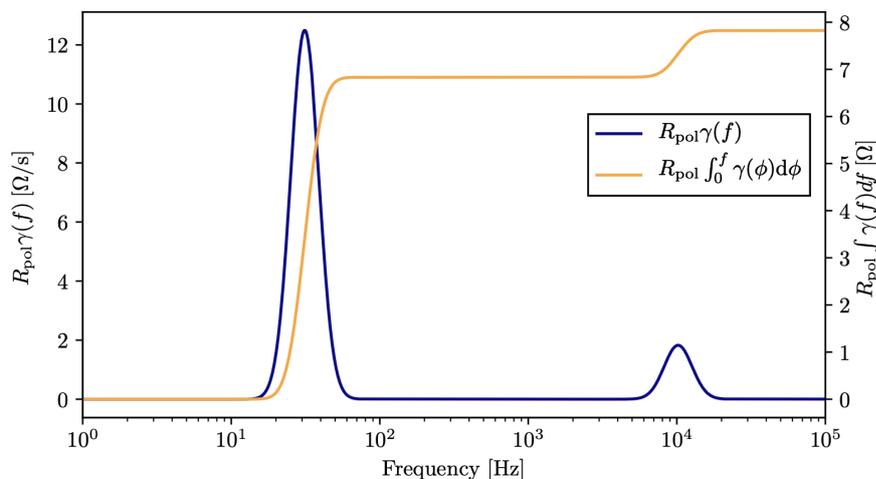


Figure 2.8: Distribution of relaxation times and integral of distribution function derived from the impedance spectrum shown in Fig. 2.6 [8] [11] [12].

thermal activation are likely to be caused by gas-phase diffusion, while those showing pronounced thermal activation most likely correspond to activation losses. Additionally, variations of the electrical load can yield insights regarding the cell's electrode reactions' symmetry and the corresponding transfer coefficient α [8].

2.4 Test rig (Testing Station)

2.4.1 Basic Solid Oxide Fuel Cell Test Station Requirements

As fuel cell performance strongly depends on the operating conditions, a good test setup must allow flexible control over the operating pressures, temperatures, humidity levels, and flow rates of the reactant gases [1].

Mass flow controllers, pressure gauges, and temperature sensors allow the operating conditions of the fuel cell to be continually monitored during testing. Electrochemical measurement equipment, usually including a potentiostat/galvanostat and an impedance analyzer, is attached to the fuel cell [1]. These measurement devices have at least two leads; one connects to the fuel cell cathode, while the other connects to the fuel cell anode. Often a third lead is provided for a reference electrode [1]. Most commercially available potentiostats can perform a wide range of potentiostatic/galvanostatic experiments, including j - V curve measurements, current interrupt, and cyclic voltammetry [1]. Electrochemical impedance spectroscopy often requires a dedicated impedance analyzer or an add-on unit in addition to the

potentiostat [1].

The fuel cell in a SOFC test station needs to reside inside a furnace with precise temperature control over a wide temperature range. Working at elevated temperatures presents special challenges, particularly in providing robust seals, electrical leads, and connections to/from the fuel cell [1]. Accurately monitoring the fuel cell conditions (such as temperature, pressure, and gas compositions) while at elevated temperatures is also challenging [1]. Designing a proper test station gets even more complicated when considering that SOFCs are frequently intended for use with hydrocarbon fuels [1]. Such fuels tend to crack at elevated temperatures and leave undesirable carbon coatings behind [1]. Methods for removing, burning, or controlling these carbon residues become essential in fuel cell test stations operating at high temperatures with hydrocarbon fuels [1].

2.4.2 Test Conditions

Test conditions will dramatically affect fuel cell performance. Therefore, care must be taken to fully document measurement operating conditions, testing procedures, device histories, and so on [1].

The most important testing conditions to document are now briefly discussed [1]:

Warm-up

To ensure that a fuel cell system is well equilibrated, it is customary to conduct a standardized warm-up procedure prior to cell characterization [1]. A typical warm-up procedure might involve operating the cell at a fixed current load for 30–60 min prior to testing. Failure to properly warm up a fuel cell system can result in highly nonstationary (non-steady-state) behavior [1].

Temperature

It is essential to document and maintain a constant fuel cell temperature during measurement. Both the gas inlet and exit temperatures should be measured as well as the temperature of the fuel cell itself [1]. Sophisticated techniques even allow temperature distributions across a fuel cell device to be monitored in real time [1]. In general, increased temperature will improve performance due to improved kinetics and conduction processes [1].

Pressure

Gas pressures are generally monitored at both the fuel cell inlets and outlets [1]. This allows the internal pressure of the fuel cell to be determined as well as the pressure

drop within the cell. Increased cell pressure will improve performance [1]. (However, increasing the pressure requires additional energy “input” from compressors, fans, etc [1].)

Flow Rate

Flow rates are generally set using mass flow controllers. During a j–V test, there are two main ways to handle reactant flow rates. In the first method, flow rates are held constant during the entire test at a flow rate that is sufficiently high so that even at the largest current densities there is sufficient supply [1]. This method is known as the fixed-flow-rate condition. In the second method, flow rates are adjusted stoichiometrically with the current so that the ratio between reactant supply and current consumption is always fixed. This method is known as the fixed-stoichiometry condition [1]. Fair j–V curve comparisons should be done using the same flow rate method. Increased flow usually improves performance [1].

Compression Force

For most fuel cell assemblies, there is an optimal cell compression force, which leads to best performance; thus, cell compression force should be noted and monitored [1]. Cells with lower compression forces can suffer increased ohmic loss, while cells with higher compression forces can suffer increased pressure or concentration losses [1].

2.5 State of Art

2.5.1 Review of the institute testing station characteristics

Experiments are conducted using an Evaluator-HT testing station manufactured by *FuelCon*. Its center is a thermally isolated furnace necessary for achieving a solid oxide cell’s operating temperature. This furnace is designed to work at temperatures up to 1100 °C and houses the examined cell. Fig. 2.9 shows the furnace as well as the peripheral systems composing the testing station. Both electrodes need a continuous supply of gas \dot{V}_{in}^{fuel} and \dot{V}_{in}^{ox} , respectively. These gas flows can be made up of multiple components: Water, hydrogen, and nitrogen on the fuel side, and either a mixture of oxygen and nitrogen or air on the oxygen side. However, the fuel side is not supplied with pure nitrogen but with protection gas, consisting of 95% nitrogen and 5% hydrogen.

This protection gas is also used to purge the cell while the testing station starts up, shutting down, or idle at a high temperature. This is done because the fuel electrode is not resistant to the effect of oxygen at high temperatures. By ensuring there is always a gas flow with at least 5% hydrogen content, the fuel electrode is

protected against oxidation. Any oxygen that might leak into the fuel gas channel immediately reacts with hydrogen. To supply deionized water to the fuel gas mixture, a peristaltic pump manufactured by *Knauer* is used. It is capable of delivering volume flows as little as 0.001 ml/min to a downstream heater. The heater evaporates the water, which is then stored in a heated container. The container is connected to the fuel line via a hand valve, which can regulate its pressure. This pressurized container decouples the steam flow to the cell from fluctuations in the liquid water flow caused by the pump.

On the oxygen side, there is the option to use either air or a custom mixture of

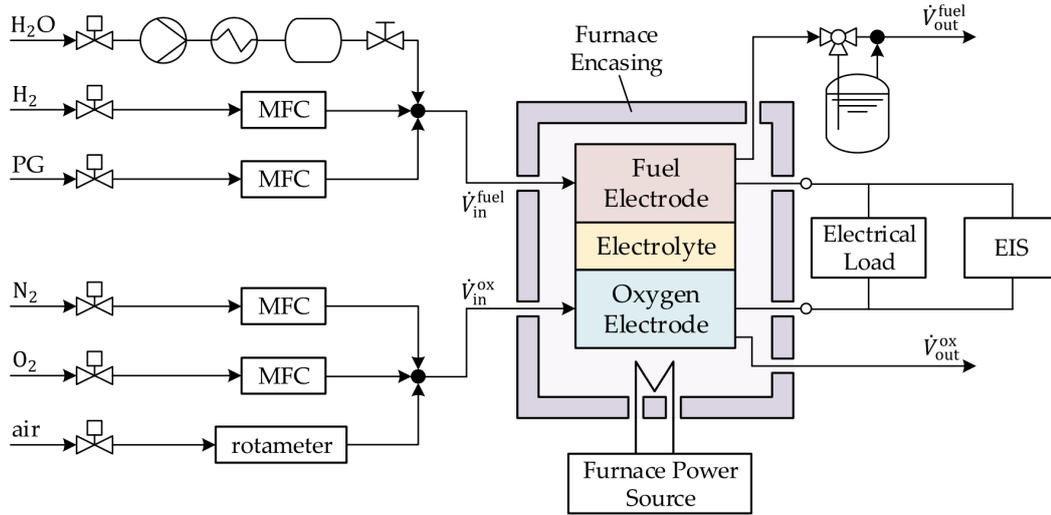


Figure 2.9: Schematic of the FuelCon Evaluator-HT testing station used in the experiment [11] [12].

nitrogen and oxygen. Air is filtered and dried, and fed to the cell at a constant pressure. It passes a rotameter, which measures its volume flow. All other gasses are supplied from pressurized gas cylinders, which are connected to valves via pressure regulators. Behind the valves are mass flow controllers (MFCs), which control each component's exact gas flow. All outgoing gas flows are fed into the building's exhaust system. For checking the fuel side for leaks, the fuel side exhaust gas can be directed through a transparent water container. The gas supply can then be reduced until no more bubbles are observed in the water container, determining the amount of gas currently leaking from the system.

The cell itself is situated within a ceramic housing within the furnace. The gasses are supplied and removed through the furnace's floor, as shown in Fig. 2.10. The cell is arranged horizontally, with the fuel electrode being the top layer and the oxygen electrode at the bottom. A concrete nickel block contacts the fuel electrode with gas channels machined into its lower side. A platinum mesh contacts the

oxygen electrode. The ceramic housing is split into two parts separated by a ceramic gasket, preventing gas exchange between oxygen and fuel side. The cell is contacted electronically in two ways: For maximally precise cell voltage measurements, two slim platinum lines lead out of the furnace floor directly beneath the cell. These are called sense lines and are used to measure U_{sense} , the primary source of information on the cell's current voltage. The electric load is attached via conductive metal rods that protrude from the furnace's top. Between the electric metal rods, a voltage

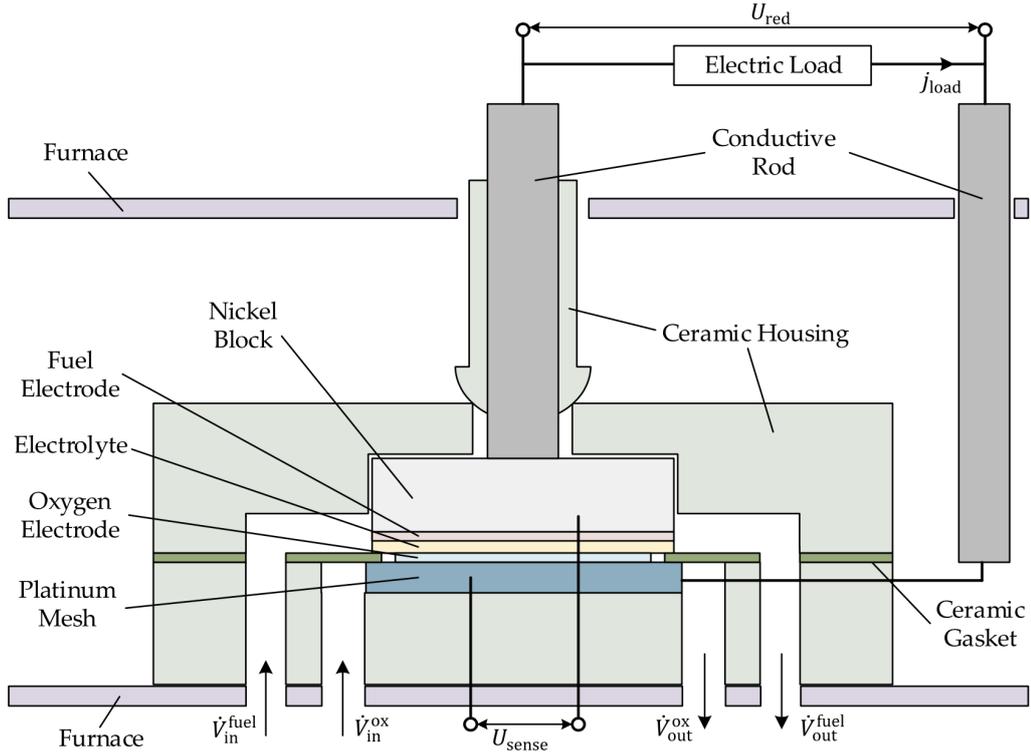


Figure 2.10: Schematic of the cell housing, including electric contacts as well as gas supply and removal [11] [12].

can be measured redundantly to the sense lines. This voltage U_{red} is only used for validating the readings from the sense lines while the cell is in an open-circuit state because it is distorted significantly by the conductive rod's resistances as soon as a current flows through them.

2.5.2 The aim of this thesis

This thesis aims to solve the problem related to the water vapor supply to fuel electrode. First of all, the author describes this obstacle by expressing the thesis

concept. As the author represented in the last section, the liquid water is supplied to the fuel gas mixture by *Knauer's* peristaltic pump. Here, the problem is occurred and makes the liquid water provide a fluctuating volume flow to the heater. Since the peristaltic pump is manufactured by a different company than the testing station, this problem happens.

The other issue the operator faces is changing the water vapor flow manually. Thus, the author needs to design and build a mechanism for fully controlling the water vapor flow and maintaining the stable condition. On the other hand, after making this mechanism and solving this problem, water vapor pressure won't oscillate, and then the system works very well.

In the following, the author describes the necessity of being pressure constant. The instability of the water supply makes partial pressure fluctuation, and gas composition won't be stable. Here, it should be expressed that the changing partial pressure directly affects the Nernst equation (Eq. 2.9). This experiment shows that the fuel side's ideal gas mixture at high temperature and low pressure consists of different components: hydrogen, nitrogen, and water vapor. The partial pressure at the ideal gas is equivalent to the molar fraction. Ideally, the ratio of partial pressures equals the ratio of the number of molecules. The mole fraction x_k of an individual gas component in an ideal gas mixture can be represented in terms of the component's partial pressure or the element's moles [26]. This concept showed in Eq. 2.26 [26]:

$$x_k = \frac{p_k}{p_{tot}} = \frac{n_k}{n_{tot}} \quad (2.26)$$

where x_k represents mole fraction of any individual gas component in a gas mixture, p_k partial pressure of any individual gas component in a gas mixture, n_k moles of any individual gas component in a gas mixture, n_{tot} total moles of the gas mixture, and p_{tot} total pressure of the gas mixture [26].

According to Eq. 2.26, changing the amount of the water vapor, which is one of the gas components (n_k), directly affects its partial pressure (p_k) through the molar fraction (x_k), and then changing the partial pressure (p_k) in the Nernst equation (Eq. 2.9) directly impacts on the cell voltage and makes the cell voltage oscillate. On the other hand, the fluctuating of cell voltage makes the trouble in useful Impedance Spectroscopy measuring. This consequence makes the steady-state condition of the testing station is deranged. Consequently, the operator can not distinguish the test rig's failure that means the appeared error would not be clear from a defeat in the test rig or water supply system.

Nevertheless, the author needs to design the mechanism for fully automated controlling the water vapor supply and maintaining the stable condition. In the next chapters, the author describes this mechanism's design, code programming, assembling all components, and eventually, the mechanism's implementation into the test rig.

Chapter 3

Conceptual Design

This project aims to develop a mechanism and control to automatically adapt the water vapor control settings to the required water volume flow. Therefore, it will consist of a stepper motor and relative components to rotate the valve, including holding the control valve and an (Arduino) based control board to control the stepper motor to its desired position. The control variable is the vapor pressure, which is directed to a target value.

For this reason, the author has designed the Coupling System for the desired connection between the pressure valve and the stepper motor by SolidWorks. Therefore, the operator will be able to control the vapor pressure automatically. In this chapter, the author will describe the whole procedure in three sections.

In the first section, he will speak about the fundamental components needed for running the stepper motor, including the stepper motor, driver controller, and Arduino board.

In the second section, he will explain the Coupling System that the author designed in the SolidWorks area and made by the Thermodynamic Institute workshop of Leibniz Universität Hannover. In the last section, the author will speak out of the Coupling System, stepper motor, and Arduino board assembly.

3.1 components in the experiment

In this chapter, the author will describe the fundamental components needed to develop this mechanism. The author will explain the Coupling System in section 3.2. The essential components are as follows:

1. Arduino control board
2. Stepper motor driver controller
3. Stepper motor

4. Power supply
5. Jumper cable Arduino
6. Coupling System

3.1.1 Arduino control board

The author utilizes the Arduino UNO, shown in Fig. 3.1, because it is the best and easiest board to get started with electronics and coding. Arduino Uno has open-source software, so it's relatively easy to implement control logic on this microcontroller board. Arduino Uno is a microcontroller board based on the ATmega328P (datasheet). It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz ceramic resonator (CSTCE16M0V53-R0), a USB connection, a power jack, an ICSP header and a



Figure 3.1: Image of Arduino Uno Board [27].

reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started [27].

3.1.2 Stepper motor driver controller

Stepper motors are operated by receiving the signal from the driver controller, shown in Fig. 3.2. In this experiment, the author utilizes the L298N Motor Driver. One of the most comfortable, best, and inexpensive way to control stepper motors is to interface L298N Motor Driver with Arduino. It can handle both the speed and spinning direction of any Bipolar stepper motor like NEMA 14.

As the L298N module has two H-Bridges, each H-Bridge will drive one of the electromagnetic coils of a stepper motor. By energizing these electromagnetic coils

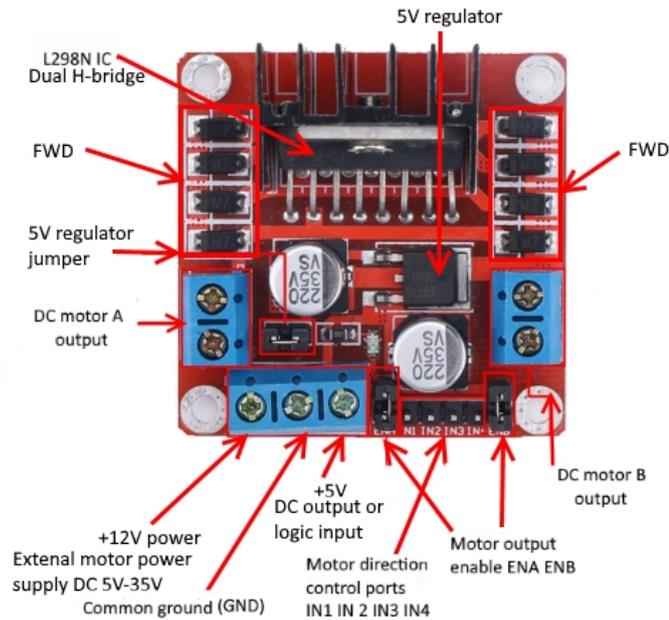


Figure 3.2: Image of L298N Stepper Motor Driver Controller with the description of its elements.

in a specific sequence, a stepper motor’s shaft would be able to move forward or backward precisely in small steps. However, the speed of a motor is determined by how frequently these coils energize. In Fig. 3.3, a driving stepper with H-Bridge is shown.

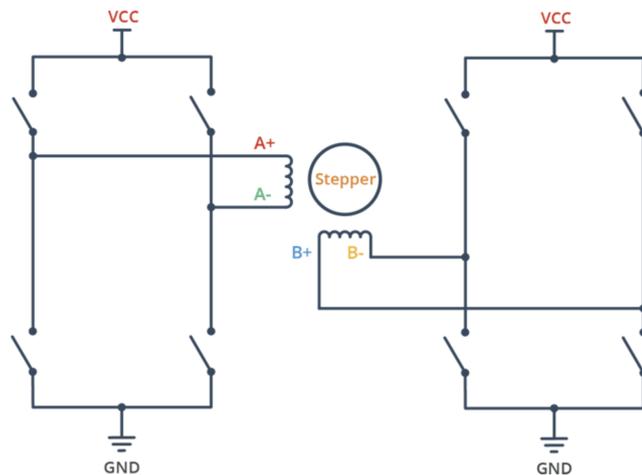


Figure 3.3: Image of the driving stepper with H-Bridge.

3.1.3 Stepper motor

In this experiment, the author utilizes NEMA 14-01 bipolar stepper shown in Fig. 3.4 rated at 12V. It offers 200 steps per revolution. Here, it needs to determine the



Figure 3.4: Image of stepper motor nema14-01.

A+, A-, B+, and B- wires on the motor. The best way to do this is to check the data sheet of the motor. In this case, these are red, green, blue, and yellow. In Fig. 3.5 shown the schematic of these wire. The connections between components

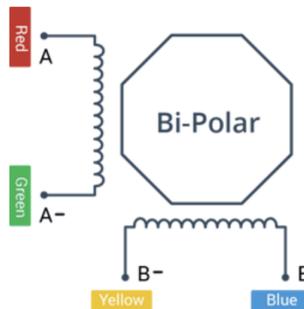


Figure 3.5: The schematic of the A+, A-, B+, and B- wires.

are relatively simple. It starts by connecting an external 12V power supply to the VCC terminal and then be kept the 5V-EN jumper in place.

It needs to be kept both the ENA and ENB jumpers in place, so the motor is always enabled. The input pins (IN1, IN2, IN3, IN4) of the L298N module are connected to four Arduino digital output pins (8, 9, 10, and 11). Eventually, connect the A+, A-, B+, and B- wires from the stepper motor to the module, as shown in Fig. 3.6.

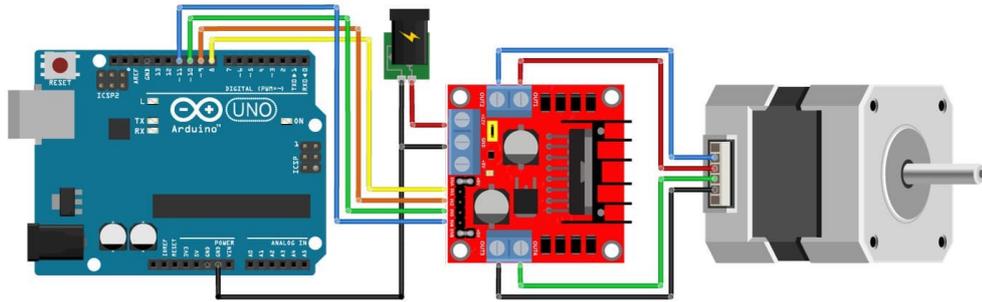


Figure 3.6: The illustration of NEMA 14-1 Stepper Motor in connection with L298N and Arduino.

3.1.4 Power supply

In this experiment, the Author utilizes the Mean Well - 12V/1.25A Rail Din power supply shown in Fig. 3.7. It is one of the best choices for this project because of its



Figure 3.7: The image of Mean Well - 12V/1.25A Rail Din power supply used in this experiment.

excellent reliability and unbeatable value for money. This type of power supply is ideal to power any type of device that requires 12VDC, and also it could be mounted directly on the DIN rail. The closed design and possession of touch-protected screw connections are the advantages of this kind of power supply.

3.1.5 Jumper cable Arduino

In this experiment, the Author utilizes the MAKER FACTORY JKMF40 Jumper cable Arduino shown in Fig 3.8. These jumper cables are easy to connect and

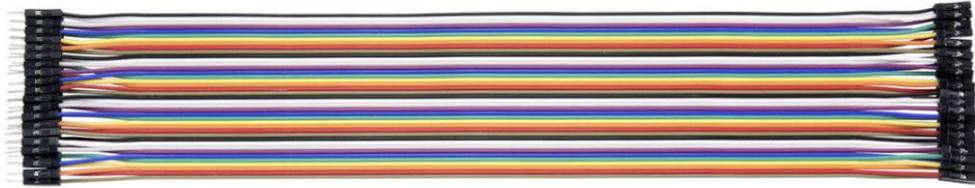


Figure 3.8: The image of the MAKER FACTORY JKMF40 Jumper cable Arduino used in this experiment.

disconnect, and the connecting and disconnecting are the advantages of these types of wires.

3.2 CAD-Design

3.2.1 The methodology of the Coupling System design

In this section, the author will explain the Coupling System. The first matter that is essential here is the connecting element between the stepper motor and the pressure valve. In the first view, it would be the easiest way to choose the shaft couplings made before and already exist in the market. However, to use the Coupling System, it has to be noted about the parts' dimensions. These parts



Figure 3.9: The image of Bellows-Sealed Metering Valves manufactured by Swagelok company [28].

are included of the pressure valve head and the stepper motor shaft. Since the dimensions of the stepper motor shaft and the head of the pressure valve are different, the author has decided to design a desired Coupling System for this experiment.

The pressure valve used in this experiment is Bellows-Sealed Metering Valves shown Fig. 3.9 manufactured by Swagelok company.

With due attention to the movement of the pressure valve head, the author noticed that the head part not only rotates around its axis but also moves along its axis or on the other hand, it has an axial movement. The author will show in Fig. 3.10 the illustration of the pressure valve and its elements with description. On the

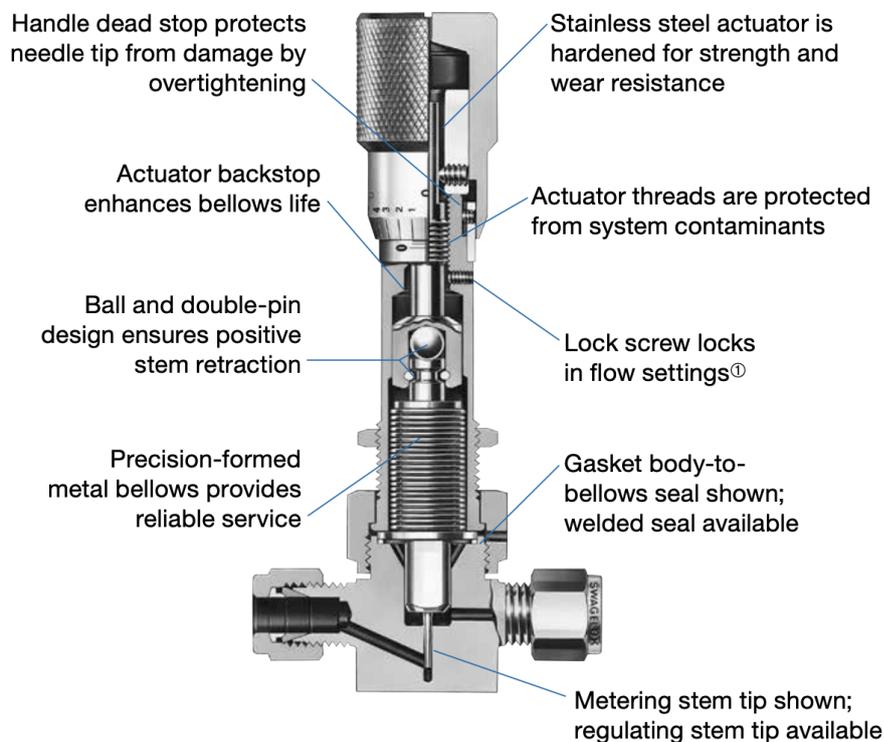


Figure 3.10: The illustration of the pressure valve and its elements with description [28].

other side, by looking at the stepper motor shaft, the author noticed that the shaft is not a full cross-section shaft, and it seems like a half cross-section shaft. Thus, the author utilized the stepper motor data-sheet to design this side of the Coupling System.

Nevertheless, the author has to design a Coupling System that allows the pressure valve to move smoothly along its axis. Since there is a plane on the stepper motor

shaft, the author utilized this feature of the shaft and made the Coupling System procedure move smoothly on this plane along the axis; simultaneously, the whole connection mechanism rotates around its axis.

3.2.2 The modeling of Coupling system by SolidWorks

In this part, the author designed the Coupling System by SolidWorks 2019. As SolidWorks is one of the bests in designing and modeling, the author's choice is SolidWorks. In modeling this part, the author considered the dimensions of each side of the Coupling System. on the side close to the stepper motor shaft; the author must design a hole for the shaft and make an appropriate way for the shaft to rotate and smoothly move along the axis. Nevertheless, the author modeled the hole with two screw thread holes M3 to fix the shaft on the spot, making the shaft moves smoothly. The Fig. 3.11 has shown the side close to the connecting of the stepper motor shaft. On the other side of the Coupling System, the author modeled

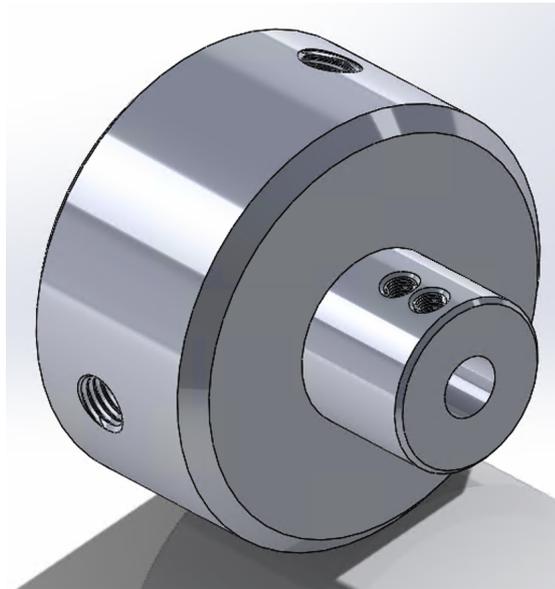


Figure 3.11: The image of modeling of the Coupling System on the view of close to the stepper motor shaft in SolidWorks.

a hole according to the pressure valve head and designed three screw thread holes around this hole to tighten the head of the pressure valve inside the Coupling System. the size of the thread holes is M5. Since the surface around the pressure valve head is roughly coarse, it would tighten and fix the head entirely. In Fig. 3.12 has shown a close view of the side close to the pressure valve head. Following ending the modeling with SolidWorks, the Coupling System was made by the



Figure 3.12: The image of modeling of the Coupling System on the view of close to the pressure valve head in SolidWorks.

Thermodynamic Institute workshop of Leibniz Universität Hannover. The material used for the Coupling system is Aluminium. the author utilized the Aluminium for making the Coupling System because of its lightness and strength. In Fig. 3.12 has shown a close view of the Coupling System after manufacturing.



Figure 3.13: The image of a close view of the Coupling System after manufacturing.

3.3 Assembly the Coupling System and the other components

At the end of this chapter, the author will speak out of assembling the whole components. In the first step for connecting the driver controller with the Arduino board, the input pins (IN1, IN2, IN3, IN4) of the L298N module are connected to four Arduino digital output pins (8, 9, 10, and 11).

The author connected the A+, A-, B+, and B- wires from the stepper motor to the driver controller in the second step. The next part is the connection between the power supply and driver controller connecting the wire coming from the voltage plus pole output on the power supply to the 12 voltage plus input on the driver controller.

In the last step of this part, the author creates a connection between three elements: voltage minus pole output on the power supply, the common ground (GND) on the driver controller, and eventually the common ground (GND) on the Arduino board.

The connection of the stepper motor, driver controller, and Arduino board is already made. According to the Arduino board and the computer's connection through a

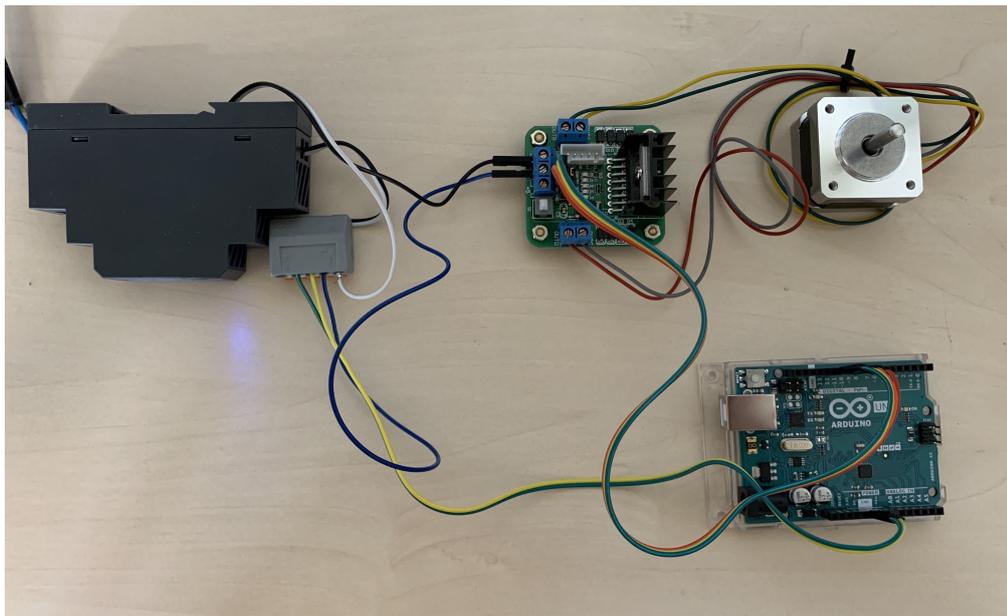


Figure 3.14: The image of the electrical components are assembled

USB cable, the Arduino board will be received the required electrical power from the computer through input USB (type B) on the Arduino Uno board. In Fig. 3.14 has shown the stepper motor, driver controller, power supply, and the Arduino

Uno board are assembled. It is possible to provide the required electrical power of the Arduino board through a five-plus voltage output of the driver controller. Since the Arduino program codes send through the USB cable and the USB cable will always be connected to the Arduino board, it would not be essential to provide the driver controller's necessary electrical power.

After finishing connecting the stepper motor, driver controller, power supply, and eventually the Arduino board, the author rejoined the Coupling System and the pressure valve head. In the end, the whole system is entirely assembled and will be

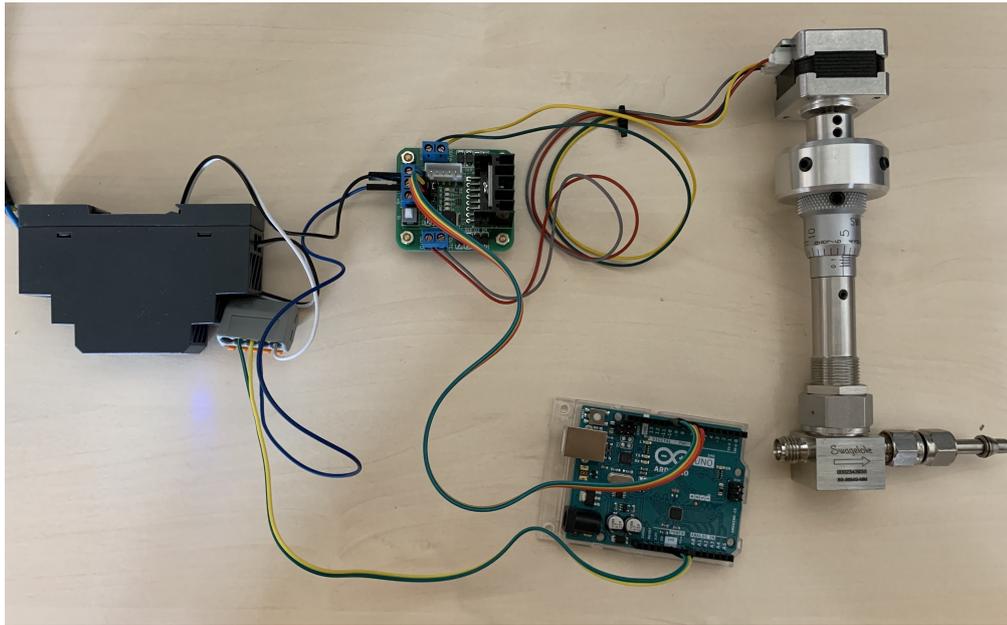


Figure 3.15: The image of the entire components are assembled

ready to run the desired command code. In Fig. 3.15 has shown the fundamental components are convened. In the next chapter, the author will be describing the running codes and then the implementation part.

Chapter 4

Implementation

In this chapter, the author will describe the implementation part. It consists of developing software, building the valve automation, and integrating automated valve into the test rig. The sections that the author mounts the whole electrical system include the Arduino board, power supply, driver controller, stepper motor, and Coupling System on the pressure valve already installed on the test rig. Eventually, the operating system will adequately run with the control codes sent from the operator's computer.

4.1 Developing software (Control)

The first section of this chapter is about developing software. The author will define and describe the Arduino Software (IDE) and Visual Studio and then explain the control codes using these programs to control and run the operating system.

4.1.1 Arduino Software (IDE)

To control the Arduino board, the user needs to utilize the Arduino Software (IDE), open-source software that makes it easy to write code and upload it to the board. This software can be used with any Arduino board [29].

In this section, the author will describe the control codes on Arduino Software (IDE) to run the operating system directly from this software.

The first lines of code are always allocated to the libraries in Arduino IDE. Thus, the author utilized the Stepper library to run the stepper motor in this code shown below.

```
// Include the Stepper library.  
#include <Stepper.h>
```

Moreover, the author defined the code line to fit the number of steps per revolution for the stepper motor shown below.

```
// change this to fit the number of steps per revolution for the  
//stepper motor.  
const int stepsPerRevolution = 200;
```

The next step is defining the code line to initialize the stepper motor library on pins 8 through 11 of the Arduino Uno board, which is displayed below.

```
// initialize the stepper library on pins 8 through 11:  
Stepper myStepper(stepsPerRevolution, 8, 9, 10, 11);
```

Before the next step, the author must define two functions in Arduino Software (IDE). The first function is void setup(), which is technically a function created at the top of each program. Inside the curly brackets is the code run one time as soon as the program starts running. The next function is void loop(), which is used as a part of its structure. The code inside the loop function repeatedly runs as long as the Arduino board is turned on. These functions have presented here.

```
void setup() {  
}  
  
void loop() {  
}
```

According to these two functions, the author utilized the "void setup" for adding the codes that will be run once. Thus, he defined the code line to initialize the serial port. "Serial. begin(9600)" is employed in this function, which means the Arduino to get ready to exchange messages with the Serial Monitor at a data rate of 9600 bits per second. That's 9600 binary ones or zeros per second and is commonly called a baud rate. This action has presented here.

```
// initialize the serial port:  
Serial.begin(9600);
```

The next step is about the speed of the stepper motor. For this reason, the author determined the code line still in the "void setup" function to set the speed of revolution, which is 100 rpm. This point should be added that the operator could change the speed of rotation.

```
// set the speed at 100 rpm:  
myStepper.setSpeed(100);
```

The last action for running the stepper motor is defining the code line to control the stepper motor steps to reach the desired point. For this reason, the author

utilized the "myStepper.step()" function to run the stepper motor. Inside of the parentheses, the operator will be able to insert the number of the desired steps. This number could be a positive number or a negative number. It means, if the number is positive, the stepper motor will rotate forward; otherwise, it will have a backward rotation.

```
// Run the Stepper motor forward at 100 steps/second until the motor  
//reaches desired steps (X/200 revolutions)(For the forward rotation,  
//the number inside of the parentheses at myStepper.step( ) must be  
//positive. For the backward rotation, this number must be negative.):  
  
myStepper.step( );
```

In the overall view, this code will be represented such as below

```
//sketch to control a stepper motor Nema 14-01 with L298N driver  
//controller and Arduino UNO.  
  
// Written by Seyed Ali Ayati  
  
// Include the Stepper library.  
#include <Stepper.h>  
  
// change this to fit the number of steps per revolution for the  
//stepper motor.  
const int stepsPerRevolution = 200;  
  
// initialize the stepper library on pins 8 through 11:  
Stepper myStepper(stepsPerRevolution, 8, 9, 10, 11);  
  
void setup() {  
  // initialize the serial port:  
  Serial.begin(9600);  
  
  // set the speed at 100 rpm:  
  myStepper.setSpeed(100);  
  
  // Run the Stepper motor forward at 100 steps/second until the motor  
//reaches desired steps (X/200 revolutions)(For the forward rotation,  
//the number inside of the parentheses at myStepper.step( ) must be
```

```
    //positive. For the backward rotation, this number must be negative.):  
  
    myStepper.step( );  
}  
  
void loop() {}
```

In the end, the code has to be verified. Thus, the operator should click on the Verify button, and then this code could be uploaded on the Arduino board.

4.1.2 Arduino board control with Visual Studio

In this section, the author will explain running the operating system with Visual Studio. First of all, it should be expressed that in case of using the Arduino Board, the user needs to utilize the Arduino Software (IDE) for communication with the Arduino board. However, the user could use other kinds of programming software such as Python, Matlab, and Visual Studio to control the Arduino board. This programming software uploads the commands to the Arduino board through Arduino IDE. Thus, the author executes a connection code between this programming software and Arduino IDE.

Concerning the connection between Arduino IDE and Visual Studio 2019, the author has written the code on the Arduino IDE area that the user will send the command by Visual Studio 2019.

The Arduino IDE division

The first lines of code are regularly designated to the libraries in Arduino IDE. Thus, the author utilized the Stepper library to command the stepper motor in this code shown here.

```
// Include the Stepper library.  
#include <Stepper.h>
```

For the next step, the author defined the variables according to the subsequent commands.

```
String data;  
char d1;  
String x;  
String y;  
int Stepperval;  
int Speedval;
```

Further, the author described the code line to fit the number of steps per rotation for the stepper motor shown below.

```
// change this to fit the number of steps per revolution for the  
//stepper motor.  
const int stepsPerRevolution = 200;
```

The next step is determining the code line to initialize the stepper motor library on pins 8 through 11 of the Arduino Uno board, which is presented here.

```
// initialize the stepper library on pins 8 through 11:  
Stepper myStepper(stepsPerRevolution, 8, 9, 10, 11);
```

At void setup() function, the author defined the code line to initialize the serial port ("Serial. begin(9600)"). Then, he represented the code line to set the pin 13 led on the standby situation.

```
void setup() {  
  // initialize the serial port:  
  Serial.begin(9600);  
  // set the pin 13 led on the standby situation:  
  pinMode(13, OUTPUT);  
}
```

All the following codes are at the void loop() function; the author utilized the "if" function to define the serial port object. This parameter is applied to get the number of bytes (characters) available for reading from the serial port, which is data that's already reached and stored in the serial receive buffer (which holds 64 bytes).

```
if(Serial.available()){}
```

At the "if" function, the author utilized the "data" variable for reading from the serial port, and then he changed this variable from String to Char. The new variable is defined, named "d1."

```
data = Serial.readString();  
d1 = data.charAt(0);
```

The next step is for selecting action based upon the first character. Thus, the author defined four cases to send the concerned command from Visual Studio to the Arduino board.

The first and second cases are related to turning on and off the pin 13 output led. These situations are for assuring the user of the lack of defect on the Arduino board.

```
switch(d1){ // select action based upon first character
  case 'A': //first character is an A = turn on pin 13 led
    digitalWrite(13, HIGH);
    break;
  case 'a': //second character is an a = turn off pin 13 led
    digitalWrite(13, LOW);
    break;
```

The next case is for sending the command from Visual Studio to the Arduino board to make the stepper motor rotate and then reach the desired position.

```
  case 'S': //third character is an S = set stepper motor steps
    x= data.substring(1);
    Stepperval = x.toInt();
    myStepper.step(Stepperval);
    delay(100); //wait for stepper motor to finish
    break;
```

The last case is concerning controlling the speed of the stepper motor.

```
  case 'C': //last character is an C = control stepper motor speed
    y= data.substring(1);
    Speedval = y.toInt();
    myStepper.setSpeed(Speedval);
    delay(100);
    break;
```

In the overall view, this code will be represented such as below

```
//sketch to control a stepper motor with sending the commands
//from Visual Studio to Arduino board.

// Writen by Seyed Ali Ayati

// Include the Stepper library.
#include <Stepper.h>

String data;
char d1;
String x;
String y;
int Stepperval;
int Speedval;

// change this to fit the number of steps per revolution for your motor.
const int stepsPerRevolution = 200;
```

```
// initialize the stepper library on pins 8 through 11:
Stepper myStepper(stepsPerRevolution, 8, 9, 10, 11);

void setup() {
  // initialize the serial port:
  Serial.begin(9600);

  // set the pin 13 led on the standby situation:
  pinMode(13, OUTPUT);
}

void loop() {
  if(Serial.available()){
    data = Serial.readString();
    d1 = data.charAt(0);
    switch(d1){ //select action based upon first character
      case 'A': //first character is an A = turn on pin 13 led
        digitalWrite(13, HIGH);
        break;
      case 'a': //second character is an a = turn off pin 13 led
        digitalWrite(13, LOW);
        break;
      case 'S': //third character is an S = set stepper motor steps
        x= data.substring(1);
        Stepperval = x.toInt();
        myStepper.step(Stepperval);
        delay(100); //wait for stepper motor to finish
        break;
      case 'C': //last character is an C = control stepper motor speed
        y= data.substring(1);
        Speedval = y.toInt();
        myStepper.setSpeed(Speedval);
        delay(100);
        break;
    }
  }
}
```

The Visual Studio 2019 division

In this section, the author explained the Visual Studio area how to control the Arduino board. Making the stepper motor easier to control, the Visual Studio is the existing way to reach this purpose.

The Visual Studio is utilized to create GUI applications using Windows Forms [30]. The layout can be controlled by housing the controls inside other containers or locking them to the form's side [30]. Commands that display data (like textbox, list box, and grid view) can be bound to data sources like databases or queries. Data-bound controls can be created by dragging items from the Data Sources window onto a design surface [30]. The UI is linked with code using an event-driven programming model. The author generates C# code for the application. In the Visual Studio code section, the author has written the proper codes to

The image shows a Windows Form titled "Form1" with the following elements:

- Title:** Water Valve Automation
- Text:** Please choose the proper COM Ports:
- Control:** A dropdown menu labeled "COM Ports".
- Text:** Please check the Arduino communication with turn on/off pin 13 on the Arduino board:
- Buttons:** "Turn On", "Turn Off", and "Reset position to 0".
- Buttons:** "Relais On" and "Relais Off".
- Text:** Please enter the desired speed (rpm):
- Control:** A text box containing "10" followed by "rounds per minute" and a "Send Speed" button.
- Text:** Please enter the desired degree change:
- Control:** A dropdown menu, a text box containing "0" followed by "degrees", and a "Send Degree" button.
- Text:** Current position:
- Control:** A text box containing "50".
- Text:** Turn to position:
- Control:** A text box containing "0" followed by "degrees" and a "Rotate to position" button.

Figure 4.1: The image of the design surface of this windows form

connect with Arduino IDE and then run the stepper motor. This code is displayed in the appendix. In Fig. 4.1, the author showed the design surface of this windows form. As shown in this figure, the first text box is related to choose the proper COM Ports that means the USB cable is plugged into which ports of the computer. The second text box is concerning checking the Arduino board communication with turn on/off pin 13 output LED shown in Fig. 4.2. The next button box is

programmed to rotate backward the valve to the beginning point. For this reason, by clicking this button, the valve location will reset to the start position. On the

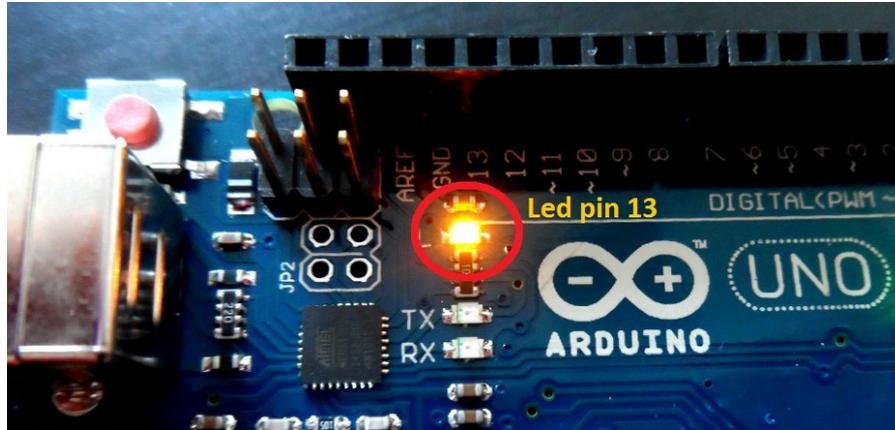


Figure 4.2: The image of the LED pin 13 on the Arduino board

other hand, the valve will be closed entirely.

There are two button boxes before the next text box for making a pause in time of shifting pressure valve position. It will act as a manual Switch and create a waiting time for the next rotation. The next text box is for entering the desired stepper motor speed, set by default in 10 rpm.

The next text box is related to the stepper motor position. For this purpose, two text boxes are assigned to change the stepper motor's direction and then change the stepper motor's position in degrees. The text box relating to switching the direction has two options "Open" for forwarding revolution and for opening the pressure valve, and "Close" is for backward rotation and for closing the pressure valve.

There is a displayed box representing the current position in the last steps, and a text box exists for applying the new position of the pressure valve. The previous text box only sends the difference between the new desired position and the last position. Consequently, it will make the operator's job more comfortable for the small pressure valve position changes.

Eventually, the author saved this program as an EXE file to make it easier to open the control file and then run the stepper motor. After all this debate, the operator will control the stepper motor only with Windows Forms as an EXE file, and technically the user would not utilize the Arduino software (IDE) anymore.

4.2 Building the valve automation

The automation valve is built with the fundamental components placed on the Aluminium Profile System, as displayed in two different sides of view Fig. 4.3 and

also Fig. 4.4.

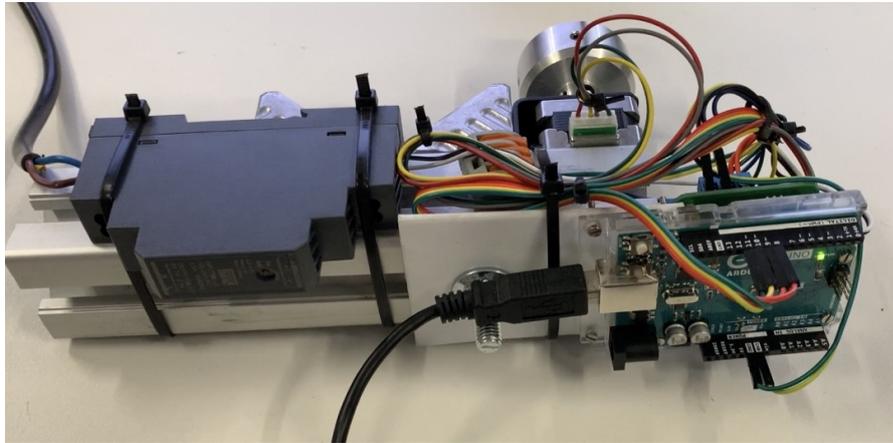


Figure 4.3: The image of the valve automation 1

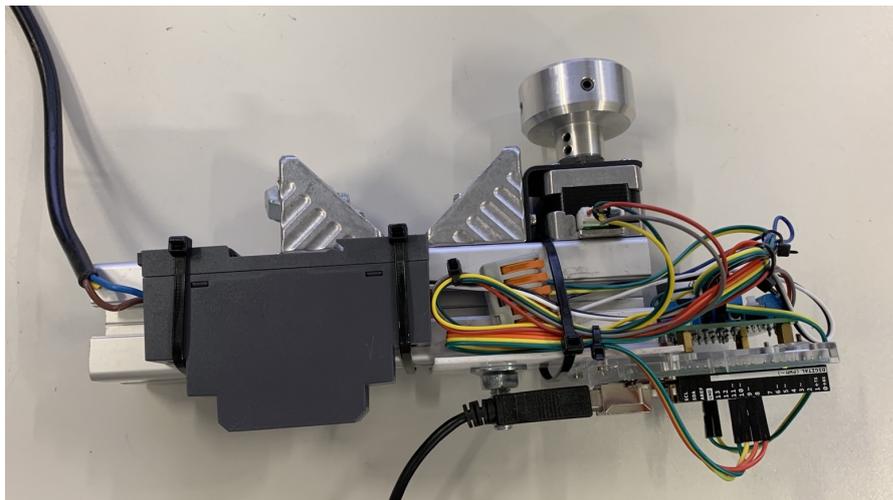


Figure 4.4: The image of the valve automation 2

4.3 Integrating automated valve into the test rig

In the last section, the author explains the implementation of the automation valve on the testing station as shown in Fig. 4.5. For mounting this system, some points must be observed. In this case, the essential point is the Coupling System's adjustment and the pressure valve head.

The alignment will always be a critical matter. For this reason, the Coupling

System needs to be in an appropriate position. Otherwise, it puts pressure on both the stepper motor shaft and also the pressure valve head.

The other point the author must observe is needing enough power for rotating the pressure valve head. In this case, this obstacle is determined by increasing the output voltage of the power supply. For this purpose, the voltage of the power



Figure 4.5: The image of the implementation of the automation valve into the testing station

supply changed from 8V to around 10V. Eventually, the automation system works

correctly without any problem. The operator will be able to run the testing station and then change the pressure valve position and measure the pressure and the cell voltage. On the other hand, the user will be able to gain the Impedance Spectroscopy measurements and measure the process's improvement by working with the new automation system.

Chapter 5

Future Research

The previous chapters were about the Arduino board's manual control and technically the stepper motor control for turning the pressure valve to set it on the desired position. That means the user could manage the water supply pressure all the time.

However, the control system, described in the current chapter, is about automatically managing water supply pressure. This automatic control system will work with the PID and Feedforward controller through Arduino software (IDE).

The author will describe the whole procedure, consisting of the PID controller, Feedforward controller, the Arduino IDE codes for running the stepper motor, the pressure sensor's introduction, which is already used in the test rig. At the end of the chapter, the author will explain the limitation of using the automatic control system, the restriction of performing both the manual control system and the automatic control system, and the lack of a chapter on the experimental measurement data.

5.1 PID controller

The PID controller is a generic control loop feedback mechanism (controller) widely used in industrial control systems; a PID is the most commonly used feedback controller [31]. Calculate an error value as the difference between the measured process variable and the desired response. The controller attempts to minimize the error by adjusting the process control input [31].

The PID controller calculation (algorithm) involves three constant parameters called the proportional (P), integral (I), and derivative (D) values. These values can be interpreted in terms of time [31]. (P) depends on the present error, (I) on the accumulation of past error, and (D) predicts future error based on the current change rate. The weighted sum of these three actions is used to adjust the process

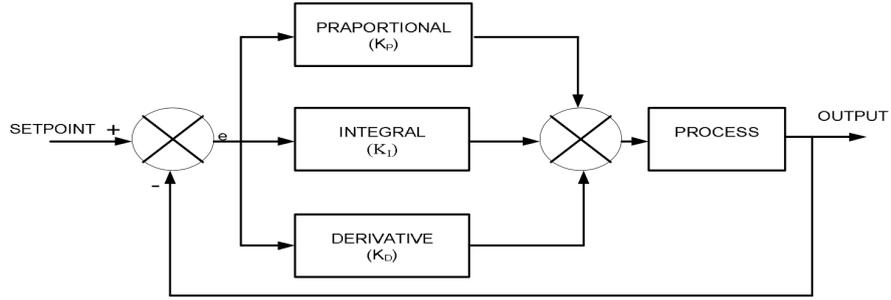


Figure 5.1: The block diagram of the PID controller [31]

via a control element such as a control valve position or power supplied to a heating element [31]. In Fig. 5.1 represented the block diagram of the PID controller, and in Eq. 5.1 represented the output of the PID controller [31]:

$$Y(t) = e(t) K_P + K_I \int_0^t e(t) dt + K_D \frac{de(t)}{dt} \quad (5.1)$$

where e represents Error Signal and K_P Proportional Constant, K_I Integral Constant, and then K_D Derivative Constant [31].

5.2 Feedforward controller

When a model of a system is well known it can be used to improve the performance of a control system by adding a Feedforward function, as pictured in Fig. 5.2 [32]. The Feedforward function is basically an inverse model of the process. When this

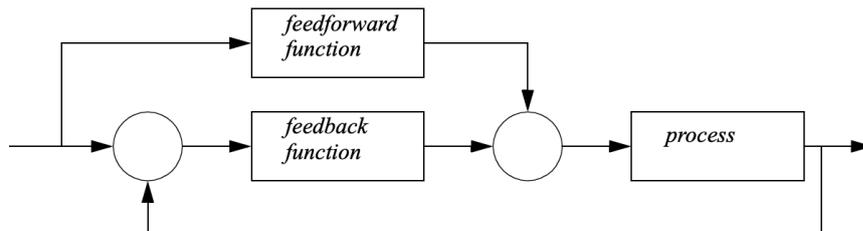


Figure 5.2: The block diagram of the Feedforward controller [32]

is used together with a more traditional feedback function, the overall system can outperform more traditional controllers function, such as the PID controller [32]. In this experiment, the PID controller measures the required pressure and sends the stepper motor's command to rotate and reach the desired position. This operation

will be run on the test rig, and the Feedforward will act like a technician and makes the controller work easier.

Since the operator has the data-sheet, such as Table 5.1, concerning steam pressure, the water vapor volume flow, and corresponding valve position, he will act like a

Steam Volume Flow (<i>ml/min</i>)	Steam Pressure (<i>bar</i>)	Valve Position (<i>round</i>)
0.241	2.9	6
0.04	2.08	1
0.121	2.553	3
0.2	2.601	5
0.28	2.548	7
0.362	2.594	8
0.019	2	0.5
0.096	2.5	2.5
0.193	2.47	4.5

Table 5.1: Examples of data-sheet include steam pressure, steam volume flow, and corresponding valve position in different moments, which have already been measured.

Feedforward controller. Thus he knows how much the stepper motor needs to rotate and then makes the pressure valve reach the proper position. On the other hand, the operator will leave the automated system on the stepper motor's accurate range. Then the PID controller runs the automated system and changes the pressure valve position a little bit. This type of Feedforward controlling makes the PID controller not work from the beginning.

Nevertheless, Feedforward control's benefits are significant and can often justify the time, making the system operate at high speeds.

5.3 Arduino IDE codes for the automatic control system

This section represents the codes that the author used in this control system, and as regularly, the Arduino IDE code commences with adding the required libraries. Since the author utilized the PID controller codes in this program, the PID controller library needs to be added.

```
// Include the Stepper library.
#include <Stepper.h>
```

```
// Include the PID library
#include <PID_v1.h>
```

Further, the author described the code line to fit the number of steps per rotation for the stepper motor shown below.

```
// change this to fit the number of steps per revolution for the
//stepper motor.
const int stepsPerRevolution = 200;
```

The next step is explaining the code lines to define the functions and variables that the author utilized for employing the PID controller in Arduino IDE. For this reason, the Setpoint shows desired pressure value, the Input presents the pressure, which is given by the sensor, and then the Output displays the process variable.

```
//Define Variables we'll be connecting to
double Setpoint ; // It will be desired pressuer value
double Input ; // The pressure which is shown by the sensor
double Output ; // Process Variable
// PID Parameters
double Kp=2 , Ki=0.5 , Kd=2 ;
// create PID instance
PID myPID(&Input, &Output, &Setpoint, Kp, Ki, Kd, DIRECT);
```

The next step is determining the code line to initialize the stepper motor library on pins 8 through 11 of the Arduino Uno board, which is presented here.

```
// initialize the stepper library on pins 8 through 11:
Stepper myStepper(stepsPerRevolution, 8, 9, 10, 11);
```

At void setup() function, the author defined the code line to convert the desired pressure into voltage. Since the Arduino board always receive the voltage as data, the author needs to do such conversion here. For this reason, the operator has to replace the pressure number with "x" in the following equation. In this line of code, the operator acts like a Feedforward controller, and he leaves the automated system on the stepper motor's reasonable range.

The next codes in void setup() function are related to launch the PID controller here. Due to the Feedforward controller, the PID controller changes the pressure valve position a little bit. Then, the author represented the code line to initialize the serial port ("Serial. begin(9600)"). Eventually, the code concerning the stepper motor speed will be inserted.

```
/void setup() {
//Insert the desired pressure, e.g. x bar
double converting_pressure_to_voltage = x *(5.0/16.0);
```

```
//Gain the Setpoint which is voltage
  Setpoint = converting_pressure_to_voltage;

//Turn the PID on
myPID.SetMode(AUTOMATIC);

// initialize the serial port:
Serial.begin(9600);

// set the speed at 100 rpm:
myStepper.setSpeed(100);
}
```

All the following codes are at the void loop() function; first, the author defined "sensorValue" for the input data from the sensor through the A0 pin on the Arduino board and then represented by the "analogRead()" function in the code lines. The "analogRead()" function Reads the value from the specified analog pin [33]. Arduino boards contain a multichannel, 10-bit analog to digital converter [33]. This means that it will map input voltages between 0 and the operating voltage(5V or 3.3V) into integer values between 0 and 1023 [33]. For instance, on an Arduino Uno, this yields a resolution between readings of 5 volts / 1024 units or 0.0049 volts (4.9 mV) per unit [33].

The next code in the void loop() is concerned with the computation with the new input in each loop. After this step, the stepper motor needs to rotate to reach the new desired position. Finally, the list of Input, Output, and Setpoint data will be exposed after each computation.

```
void loop() {
  // get the sensor value
  int sensorValue = analogRead(A0);
  Input= sensorValue * (5.0 / 1024.0);

  //computation with the new input in each loop
  myPID.Compute();

  //move a number of steps equal to the change in the sensor reading.
  myStepper.step(Setpoint - Input);

  Serial.print(Input);
  Serial.print(" ");
  Serial.println(Output);
  Serial.print(" ");
  Serial.println(Setpoint);
}
```

```
    // delay(100);  
}
```

In the overall view, this code will be represented such as below

```
//Sketch to set Automaticly the position of the stepper motor by PID controller  
  
// Writen by Seyed Ali Ayati  
  
// Include the Stepper library.  
#include <Stepper.h>  
  
// Include the PID library  
#include <PID_v1.h>  
  
// change this to fit the number of steps per revolution for the  
//stepper motor.  
const int stepsPerRevolution = 200;  
  
//Define Variables we'll be connecting to  
double Setpoint ; // It will be desired pressure value  
double Input ; // The pressure which is shown by the sensor  
double Output ; // Process Variable  
// PID Parameters  
double Kp=2 , Ki=0.5 , Kd=2 ;  
  
// create PID instance  
PID myPID(&Input, &Output, &Setpoint, Kp, Ki, Kd, DIRECT);  
  
// initialize the stepper library on pins 8 through 11:  
Stepper myStepper(stepsPerRevolution, 8, 9, 10, 11);  
  
void setup() {  
    //Insert the desired pressure,e.g. 3.0 bar  
    double converting_pressure_to_voltage = 2.5 *(5.0/16.0);  
  
    //Gain the Setpoint which is voltage  
    Setpoint = converting_pressure_to_voltage;  
  
    //Turn the PID on  
    myPID.SetMode(AUTOMATIC);  
  
    // initialize the serial port:  
    Serial.begin(9600);
```

```
// set the speed at 100 rpm:
myStepper.setSpeed(100);
}

void loop() {
  // get the sensor value
  int sensorValue = analogRead(A0);
  Input= sensorValue * (5.0 / 1024.0);

  //computation with the new input in each loop
  myPID.Compute();

  // move a number of steps equal to the change in the sensor reading.
  myStepper.step(Setpoint - Input);

  Serial.print(Input);
  Serial.print(" ");
  Serial.println(Output);
  Serial.print(" ");
  Serial.println(Setpoint);
  // delay(100);
}
```

5.4 Water supply pressure sensor into test rig

The pressure sensor used for water supply into test rig is Pressure transmitter Model A-10 made by WIKA LLC shown in Fig. 5.3. One of the output data in this



Figure 5.3: Pressure transmitter model A-10 made by WIKA LLC [34]

pressure transmitter is in the current. As the Arduino board used in this project, the whole data sent to this board must be in voltage. Thus, the current output of the pressure transmitter needs to be converted into voltage.

For this reason, using the current-to-voltage module is one of the ways to reach this goal. The current-to-voltage module linearly converts 0-25mA current signals into 0-5V voltage signals.

Pressure transmitter Model A-10 has the current signal output of 4-20 mA. Therefore, the Arduino control board can easily read the current signals output from the sensor. Typically, current signals lower than 4mA can be applied for fault diagnosis, and current signals higher than 20mA can be used for overrun detection. By affording this convertor and then install on the Arduino board, the user will be able to run the Arduino code which poses the PID controller. Then it would be possible to control the automation valve automatically.

5.5 Limitations of this experiment.

This experiment had some problems: test rig unpreparedness and difficulty gaining current from the pressure transmitter.

Regarding the test rig unpreparedness, the experimental procedure was impossible to perform due to the Covid-19 Pandemic situation. As the other colleagues had to implement the other concerned parts of the test rig and replace the new hardware, the testing station had not been ready until the end of this thesis. Thus, the condition of testing the codes related to the automatic control system and, indeed, the whole system was not available.

The other problem was related to difficulty receiving the data as current from the sensor. It would be obliged to connect the pressure transmitter to the current-to-voltage module directly to reach this point, and for this reason, it needs to take out the concerned current wire from the sensor. As the pressure transmitter is a valuable component, this action may make the sensor be harmed.

This part of the research will be able to be continued in the future after passing the Corona pandemic trouble situation. Under these circumstances, it can be hoped that the operator will test the automatic control plan, then its weaknesses will be revealed, and in the end, decide to improve the system.

Regarding the manual control plan, the Author examined the automated valve system, and it worked correctly. Still, due to the test rig's unpreparedness, it would not be capable of gaining the experiment results such as Impedance Spectroscopy measurement and improved test rig performance.

Chapter 6

Conclusion

In this thesis, the automated valve system is designed and built to control the water vapor volume flow to prevent the water vapor's oscillation. As a novelty, the research developed a mechanism to automatically control the water vapor control settings' adaption to reach the required water volume flow.

The automated valve system is controlled through the Arduino Uno control board. This control board sent the stepper motor's command for rotating the pressure valve to control the water vapor volume flow.

The uniqueness of the stepper motor shaft and pressure valve head brings us to the point that the author designed the specific type of the Coupling System by the SolidWorks. This design helped us to handle the pressure valve position.

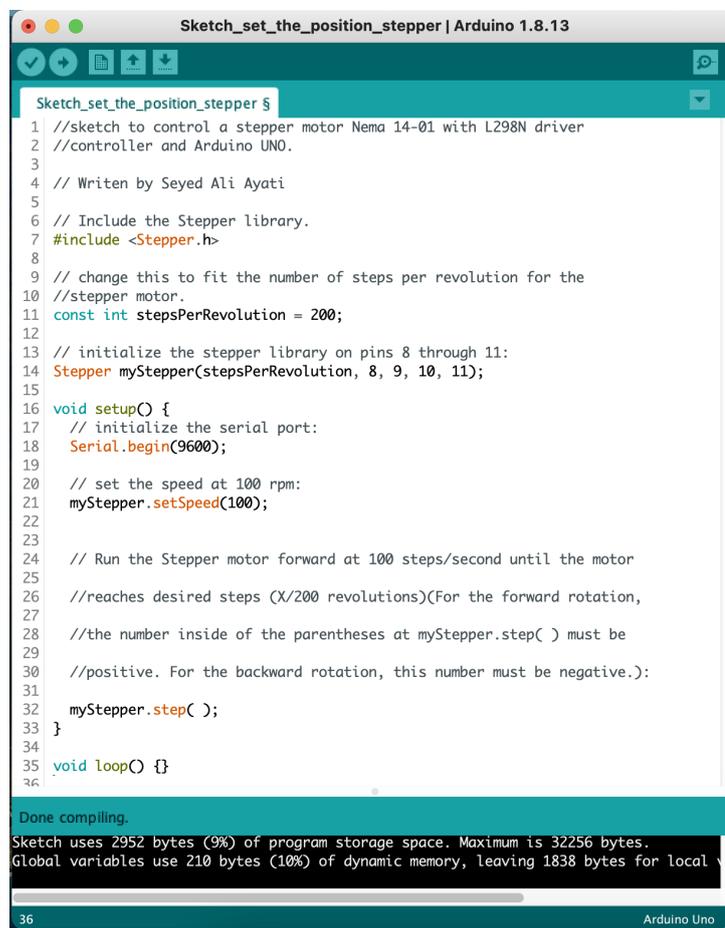
The author defined two different manners: the manual control program and the automatic control program. This definition helped us to control the automated valve system.

The automated valve system runs with the Arduino software (IDE) to control the pressure valve position. The author also used the Visual Studio code section to connect with Arduino IDE and run the stepper motor. The advantage of using the Visual Studio program was to control the automated valve system conveniently and save the control program as an EXE file. As we aimed, the automated valve system was examined and worked accurately.

The automated valve system considers the automatic control program to utilize the PID controller with Feedforward controller through Arduino software (IDE). One of the limitations that the thesis face was unable to run the PID controller to reach the desired point automatically. Due to the Covid-19 Pandemic situation, the testing station was unprepared, and gaining the experiment results such as Impedance Spectroscopy measurement and improved test rig performance by the automated valve system was impossible.

Appendix B

The Arduino Sketch

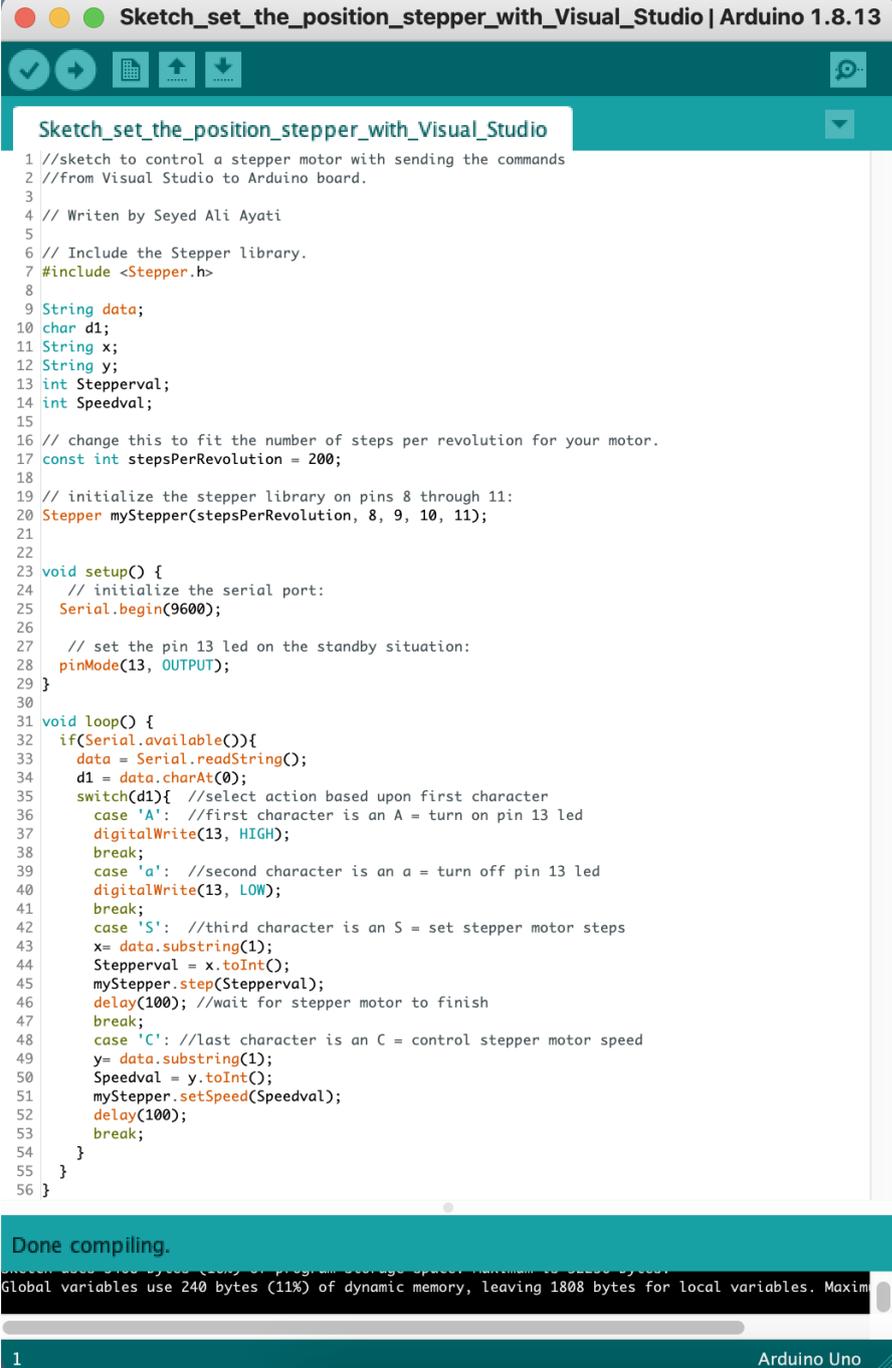


```
Sketch_set_the_position_stepper | Arduino 1.8.13
Sketch_set_the_position_stepper $
1 //sketch to control a stepper motor Nema 14-01 with L298N driver
2 //controller and Arduino UNO.
3
4 // Writen by Seyed Ali Ayati
5
6 // Include the Stepper library.
7 #include <Stepper.h>
8
9 // change this to fit the number of steps per revolution for the
10 //stepper motor.
11 const int stepsPerRevolution = 200;
12
13 // initialize the stepper library on pins 8 through 11:
14 Stepper myStepper(stepsPerRevolution, 8, 9, 10, 11);
15
16 void setup() {
17   // initialize the serial port:
18   Serial.begin(9600);
19
20   // set the speed at 100 rpm:
21   myStepper.setSpeed(100);
22
23
24   // Run the Stepper motor forward at 100 steps/second until the motor
25   //reaches desired steps (X/200 revolutions)(For the forward rotation,
26   //the number inside of the parentheses at myStepper.step( ) must be
27   //positive. For the backward rotation, this number must be negative.):
28   myStepper.step( );
29
30 }
31
32 void loop() {}
33
34
35
36
```

Done compiling.
Sketch uses 2952 bytes (9%) of program storage space. Maximum is 32256 bytes.
Global variables use 210 bytes (10%) of dynamic memory, leaving 1838 bytes for local

36 Arduino Uno

Figure B.1: The view of the Arduino sketch for directly changing the stepper motor position.



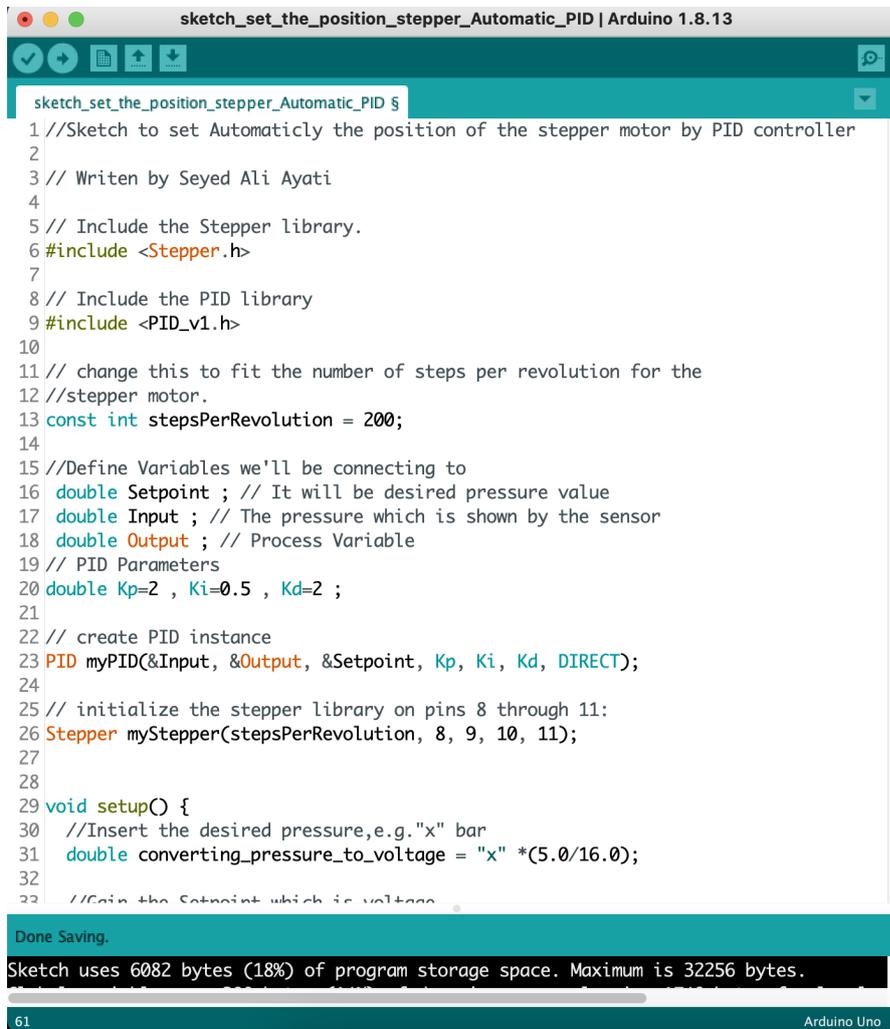
```
1 //sketch to control a stepper motor with sending the commands
2 //from Visual Studio to Arduino board.
3
4 // Writen by Seyed Ali Ayati
5
6 // Include the Stepper library.
7 #include <Stepper.h>
8
9 String data;
10 char d1;
11 String x;
12 String y;
13 int Stepperval;
14 int Speedval;
15
16 // change this to fit the number of steps per revolution for your motor.
17 const int stepsPerRevolution = 200;
18
19 // initialize the stepper library on pins 8 through 11:
20 Stepper myStepper(stepsPerRevolution, 8, 9, 10, 11);
21
22
23 void setup() {
24   // initialize the serial port:
25   Serial.begin(9600);
26
27   // set the pin 13 led on the standby situation:
28   pinMode(13, OUTPUT);
29 }
30
31 void loop() {
32   if(Serial.available()){
33     data = Serial.readString();
34     d1 = data.charAt(0);
35     switch(d1){ //select action based upon first character
36       case 'A': //first character is an A = turn on pin 13 led
37         digitalWrite(13, HIGH);
38         break;
39       case 'a': //second character is an a = turn off pin 13 led
40         digitalWrite(13, LOW);
41         break;
42       case 'S': //third character is an S = set stepper motor steps
43         x= data.substring(1);
44         Stepperval = x.toInt();
45         myStepper.step(Stepperval);
46         delay(100); //wait for stepper motor to finish
47         break;
48       case 'C': //last character is an C = control stepper motor speed
49         y= data.substring(1);
50         Speedval = y.toInt();
51         myStepper.setSpeed(Speedval);
52         delay(100);
53         break;
54     }
55   }
56 }
```

Done compiling.

Global variables use 240 bytes (11%) of dynamic memory, leaving 1808 bytes for local variables. Maxim

1 Arduino Uno

Figure B.2: The view of the Arduino sketch for directly changing the stepper motor position with sending the command from Visual Studio 2019.



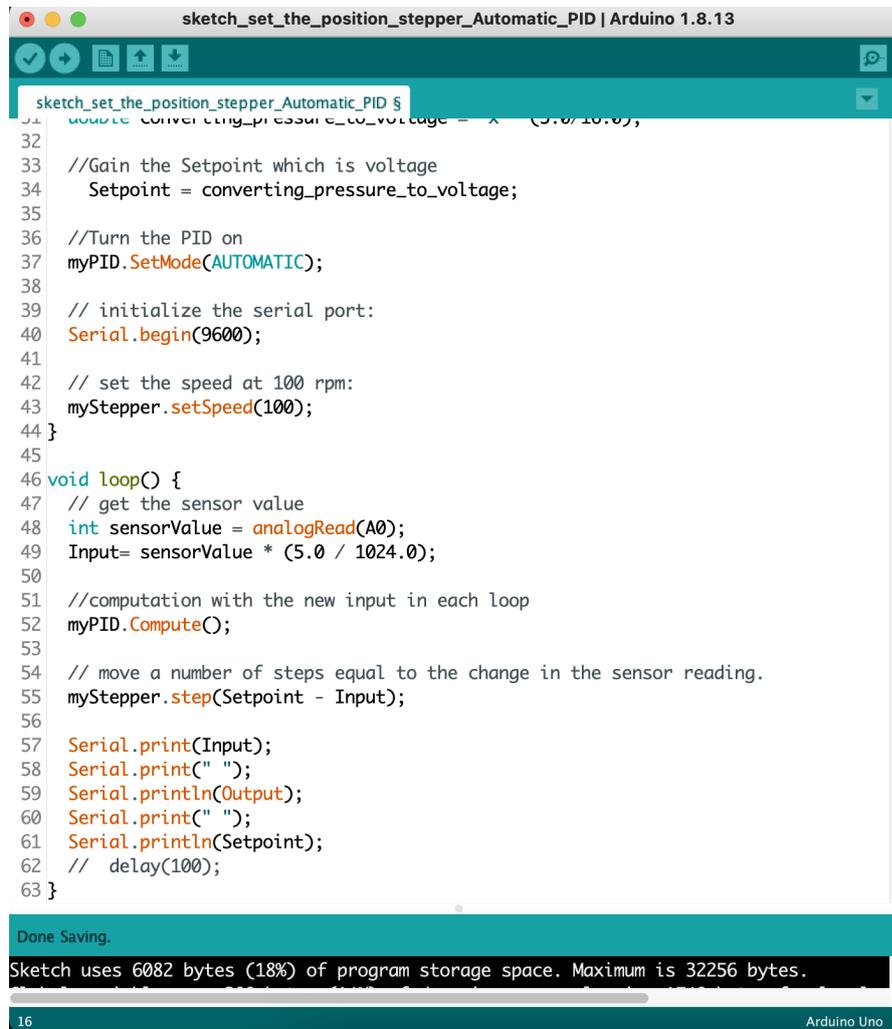
```
sketch_set_the_position_stepper_Automatic_PID | Arduino 1.8.13
sketch_set_the_position_stepper_Automatic_PID $
1 //Sketch to set Automatically the position of the stepper motor by PID controller
2
3 // Written by Seyed Ali Ayati
4
5 // Include the Stepper library.
6 #include <Stepper.h>
7
8 // Include the PID library
9 #include <PID_v1.h>
10
11 // change this to fit the number of steps per revolution for the
12 //stepper motor.
13 const int stepsPerRevolution = 200;
14
15 //Define Variables we'll be connecting to
16 double Setpoint ; // It will be desired pressure value
17 double Input ; // The pressure which is shown by the sensor
18 double Output ; // Process Variable
19 // PID Parameters
20 double Kp=2 , Ki=0.5 , Kd=2 ;
21
22 // create PID instance
23 PID myPID(&Input, &Output, &Setpoint, Kp, Ki, Kd, DIRECT);
24
25 // initialize the stepper library on pins 8 through 11:
26 Stepper myStepper(stepsPerRevolution, 8, 9, 10, 11);
27
28
29 void setup() {
30 //Insert the desired pressure,e.g."x" bar
31 double converting_pressure_to_voltage = "x" *(5.0/16.0);
32
33 //Gain the Setpoint which is voltage
```

Done Saving.

Sketch uses 6082 bytes (18%) of program storage space. Maximum is 32256 bytes.

61 Arduino Uno

Figure B.3: The view of the Arduino sketch for automatically control the position of the stepper motor by PID controller (part 1).



```
sketch_set_the_position_stepper_Automatic_PID | Arduino 1.8.13
31 double converting_pressure_to_voltage = x * (5.0 / 10.0);
32
33 //Gain the Setpoint which is voltage
34 Setpoint = converting_pressure_to_voltage;
35
36 //Turn the PID on
37 myPID.SetMode(AUTOMATIC);
38
39 // initialize the serial port:
40 Serial.begin(9600);
41
42 // set the speed at 100 rpm:
43 myStepper.setSpeed(100);
44 }
45
46 void loop() {
47 // get the sensor value
48 int sensorValue = analogRead(A0);
49 Input= sensorValue * (5.0 / 1024.0);
50
51 //computation with the new input in each loop
52 myPID.Compute();
53
54 // move a number of steps equal to the change in the sensor reading.
55 myStepper.step(Setpoint - Input);
56
57 Serial.print(Input);
58 Serial.print(" ");
59 Serial.println(Output);
60 Serial.print(" ");
61 Serial.println(Setpoint);
62 // delay(100);
63 }
```

Done Saving.

Sketch uses 6082 bytes (18%) of program storage space. Maximum is 32256 bytes.

16 Arduino Uno

Figure B.4: The view of the Arduino sketch for automatically control the position of the stepper motor by PID controller (part 2).

Appendix C

Visual Studio Codes

The codes used in the Visual Studio 2019

```
1 using System;
2 using System.Collections.Generic;
3 using System.ComponentModel;
4 using System.Data;
5 using System.Drawing;
6 using System.Linq;
7 using System.Text;
8 using System.Windows.Forms;
9 using System.IO.Ports;
10 using System.Threading;
11
12 namespace Visual_Manual_Part
13 {
14     public partial class Form1 : Form
15     {
16         public static double position = 0;
17         public static double max_position_value = 7 * 360;
18         public Form1()
19         {
20             InitializeComponent();
21             serialPort1.Open();
22         }
23     }
24
25     private void Form1_Load(object sender, EventArgs e)
26     {
27         string [] ports = SerialPort.GetPortNames();
28         cBoxCOMPORT.Items.AddRange(ports);
29         CurrentPosition.Text = Convert.ToString(position);
30     }
```

```
31
32     private void comboBox1_SelectedIndexChanged(object sender,
EventArgs e)
33     {
34
35     }
36
37     private void onButton_Click(object sender, EventArgs e)
38     {
39         // Send command to Arduino to turn pin 13 on
40         serialPort1.Write("A");
41     }
42
43     private void offButton_Click(object sender, EventArgs e)
44     {
45         // Send command to Arduino to turn pin 13 off
46         serialPort1.Write("a");
47     }
48
49     private void StepperMotorDeg_Click(object sender, EventArgs e
)
50     {
51         double sign;
52         switch (comboBox_Sign.SelectedItem)
53         {
54             case "open":
55                 sign = 1;
56                 break;
57             case "close":
58                 sign = -1;
59                 break;
60             default:
61                 sign = 0;
62                 break;
63         }
64
65         //Send Steps value to Stepper Motor
66         double doubletextbox1 = Convert.ToDouble(textBox1.Text);
67         double DoubleValue = 0.5555556; //(200 steps / 360 degree
= 0.5555556)
68         double ConversionDegreeToSteps = doubletextbox1 *
DoubleValue * sign;
69         double new_position = position + doubletextbox1 * sign *
-1;
70         if (new_position < 0)
71         {
72             string message = "Position value smaller than minimum
position.";
73             string caption = "Error";
```

```
74         MessageBoxButtons buttons = MessageBoxButtons.OK;
75
76         DialogResult result;
77
78         result = MessageBox.Show(message, caption, buttons);
79
80
81     }
82     else
83     if (new_position > max_position_value)
84     {
85         string message = "Position value higher than maximum
position.";
86         string caption = "Error";
87         MessageBoxButtons buttons = MessageBoxButtons.OK;
88
89         DialogResult result;
90
91         result = MessageBox.Show(message, caption, buttons);
92
93
94     }
95     else
96     {
97         string StringedConversionDegreeToSteps =
ConversionDegreeToSteps.ToString();
98         string m1 = "S" + StringedConversionDegreeToSteps;
99         serialPort1.Write(m1);
100        position = position + doubletextBox1 * sign * -1;
101        CurrentPosition.Text = Convert.ToString(position);
102    }
103 }
104
105 private void label5_Click(object sender, EventArgs e)
106 {
107
108 }
109
110 private void textBox2_TextChanged(object sender, EventArgs e)
111 {
112
113 }
114
115 e) private void StepperMotorSpeed_Click(object sender, EventArgs
116 {
117     string m2 = "C" + textBox2.Text;
118     serialPort1.Write(m2);
119 }
```

```
120
121     private void label6_Click(object sender, EventArgs e)
122     {
123
124     }
125
126     private void button2_Click(object sender, EventArgs e)
127     {
128         string message = "Rotate valve manually to minimum
position and press OK.";
129         string caption = "Error";
130         MessageBoxButtons buttons = MessageBoxButtons.OK;
131
132         DialogResult result;
133
134         result = MessageBox.Show(message, caption, buttons);
135         position = 0;
136         CurrentPosition.Text = Convert.ToString(position);
137
138     }
139
140
141     private void RotateToPosition_Click(object sender, EventArgs
e)
142     {
143
144
145         double new_position = Convert.ToDouble(DesiredPosition.
Text);
146
147         if (new_position < 0)
148         {
149             string message = "Position value smaller than 0.";
150             string caption = "Error";
151             MessageBoxButtons buttons = MessageBoxButtons.OK;
152
153             DialogResult result;
154
155             result = MessageBox.Show(message, caption, buttons);
156
157         }
158         else
159         if (new_position > max_position_value)
160         {
161             string message = "Position value higher than maximum
position.";
162             string caption = "Error";
163             MessageBoxButtons buttons = MessageBoxButtons.OK;
164
```

```
165         DialogResult result;
166
167         result = MessageBox.Show(message, caption, buttons);
168
169
170     }
171     else
172     {
173
174         double differenz = new_position - position;
175
176         double DoubleValue = 0.5555556; //(200 steps / 360
177         degree = 0.5555556)
178         double ConversionDegreeToSteps = differenz *
179         DoubleValue * -1;
180         string StringedConversionDegreeToSteps =
181         ConversionDegreeToSteps.ToString();
182         string ml = "S" + StringedConversionDegreeToSteps;
183         serialPort1.Write("R");
184         Thread.Sleep(3000);
185         serialPort1.Write(ml);
186         int waittime = (Convert.ToInt32(differenz)/360 * 60/
187         Convert.ToInt32(textBox2.Text))*1000 + 5000;
188         Thread.Sleep(waittime);
189         serialPort1.Write("r");
190
191         position = new_position;
192         CurrentPosition.Text = Convert.ToString(position);
193     }
194 }
195
196 private void RelaisOn_Click(object sender, EventArgs e)
197 {
198     serialPort1.Write("R");
199 }
200
201 private void RelaisOff_Click(object sender, EventArgs e)
202 {
203     serialPort1.Write("r");
204 }
```

Bibliography

- [1] Ryan P O’Hayre, Suk-Won Cha, Whitney G Colella, and Fritz B Prinz. *1014 Fuel Cell Fundamentals*. 2008 (cit. on pp. 1, 3–7, 17–19).
- [2] Paolo Di Giorgio and Umberto Desideri. «Potential of reversible solid oxide cells as electricity storage system». In: *Energies* 9.8 (2016), p. 662 (cit. on p. 1).
- [3] Martin Hauth et al. «Production and reliability oriented SOFC cell and stack design». In: *ECS Transactions* 78.1 (2017), p. 2231 (cit. on p. 1).
- [4] Christopher H Wendel and Robert J Braun. «Design and techno-economic analysis of high efficiency reversible solid oxide cell systems for distributed energy storage». In: *Applied energy* 172 (2016), pp. 118–131 (cit. on p. 1).
- [5] Kun Wang et al. «A review on solid oxide fuel cell models». In: *International journal of hydrogen energy* 36.12 (2011), pp. 7212–7228 (cit. on p. 1).
- [6] Bernard A Boukamp and Aurélie Rolle. «Use of a distribution function of relaxation times (DFRT) in impedance analysis of SOFC electrodes». In: *Solid state ionics* 314 (2018), pp. 103–111 (cit. on pp. 2, 16).
- [7] André Leonide, Volker Sonn, André Weber, and Ellen Ivers-Tiffée. «Evaluation and modeling of the cell resistance in anode-supported solid oxide fuel cells». In: *Journal of the Electrochemical Society* 155.1 (2007), B36 (cit. on p. 2).
- [8] A. Leonide. *SOFC Modelling and Parameter Identification by Means of Impedance Spectroscopy*. Schriften des Instituts für Werkstoffe der Elektrotechnik, Karlsruher Institut für Technologie. KIT Scientific Publ., 2010. ISBN: 9783866445383. URL: <https://books.google.de/books?id=kE30tpn2DS8C> (cit. on pp. 2, 7, 8, 10, 11, 15–17).
- [9] Helge Schichlein, Axel C Müller, Michael Voigts, Albert Krügel, and Ellen Ivers-Tiffée. «Deconvolution of electrochemical impedance spectra for the identification of electrode reaction mechanisms in solid oxide fuel cells». In: *Journal of Applied Electrochemistry* 32.8 (2002), pp. 875–882 (cit. on p. 2).
- [10] HD Baehr and S Kabelac. *Thermodynamics: Fundamentals and Technical Applications*. 14th edition. 2009 (cit. on p. 4).

- [11] Jan Hollmann. «Experimentelle Validierung und Erweiterung eines Modells zur Beschreibung des Betriebsverhaltens einer umkehrbaren Festoxidzelle (ReSOC)». MA thesis. Institut für Thermodynamik: Leibniz Universität Hannover, 2018 (cit. on pp. 4, 7, 17, 20, 21).
- [12] Johannes Kube. «Development of an Automated Test Sequence for the Characterisation of High Temperature Solid Oxide Cells (SOFC / SOEC)». MA thesis. Institut für Thermodynamik: Leibniz Universität Hannover, 2018 (cit. on pp. 4, 7, 17, 20, 21).
- [13] Sergio Yesid Gómez and Dachamir Hotza. «Current developments in reversible solid oxide fuel cells». In: *Renewable and Sustainable Energy Reviews* 61 (2016), pp. 155–174 (cit. on p. 5).
- [14] Mark E Orazem and Bernard Tribollet. «Electrochemical impedance spectroscopy». In: *New Jersey* (2008) (cit. on pp. 8, 11).
- [15] Allen J Bard, Larry R Faulkner, et al. «Fundamentals and applications». In: *Electrochemical Methods* 2.482 (2001), pp. 580–632 (cit. on p. 8).
- [16] S Primdahl and M Mogensen. «Gas diffusion impedance in characterization of solid oxide fuel cell anodes». In: *Journal of the electrochemical society* 146.8 (1999), p. 2827 (cit. on p. 9).
- [17] Jai-Woh Kim, Anil V Virkar, Kuan-Zong Fung, Karun Mehta, and Subhash C Singhal. «Polarization effects in intermediate temperature, anode-supported solid oxide fuel cells». In: *Journal of the Electrochemical Society* 146.1 (1999), p. 69 (cit. on p. 9).
- [18] A Weber. «Development and characterization of materials and components for the high-temperature fuel cell SOFC». PhD thesis. dissertation, University "a t Karlsruhe (TH), 2002 (cit. on p. 10).
- [19] Subhash C Singhal and Kevin Kendall. *High-temperature solid oxide fuel cells: fundamentals, design and applications*. Elsevier, 2003 (cit. on p. 10).
- [20] E Ivers-Tiffée. «Brennstoffzellen und Batterien, lecture notes». In: *Institut für Werkstoffe der Elektrotechnik (IWE), Karlsruher Institut für Technologie (KIT), Germany* (2009) (cit. on p. 10).
- [21] Axel Müller. «Multi-layer anode for the high-temperature fuel cell (SOFC)». PhD thesis. publisher cannot be determined, 2004 (cit. on p. 11).
- [22] Helge Schichlein. *Experimental modeling for the high-temperature fuel cell SOFC*. Mainz, 2003 (cit. on pp. 11–13).
- [23] Gamry Instruments. «Basics of electrochemical impedance spectroscopy». In: *G. Instruments, Complex impedance in Corrosion* (2007), pp. 1–30 (cit. on pp. 13, 14).

- [24] David Loveday, Pete Peterson, and Bob Rodgers. «Evaluation of organic coatings with electrochemical impedance spectroscopy». In: *JCT coatings tech* 8 (2004), pp. 46–52 (cit. on p. 14).
- [25] Yanxiang Zhang, Yu Chen, Mei Li, Mufu Yan, Meng Ni, and Changrong Xia. «A high-precision approach to reconstruct distribution of relaxation times from electrochemical impedance spectroscopy». In: *Journal of power sources* 308 (2016), pp. 1–6 (cit. on p. 16).
- [26] AD McNaught and A Wilkinson. *IUPAC Compendium of Chemical Terminology, 2nd edn.(1997)*. 1997 (cit. on p. 22).
- [27] Arduino LLC. *Arduino Uno Board Description*. 2021. URL: <https://store.arduino.cc/arduino-uno-rev3> (visited on 01/13/2021) (cit. on p. 24).
- [28] Swagelok LLC. *Bellows-Sealed Metering Valves*. 2021. URL: <https://www.swagelok.com/downloads/webcatalogs/en/MS-01-23.pdf> (visited on 01/22/2021) (cit. on pp. 28, 29).
- [29] Arduino LLC. *Arduino Software (IDE) Software Description*. 2021. URL: <https://www.arduino.cc/en/software> (visited on 01/19/2021) (cit. on p. 34).
- [30] Microsoft LLC. *Bind controls to data in Visual Studio*. 2021. URL: <https://docs.microsoft.com/en-us/previous-versions/visualstudio/visual-studio-2015/data-tools/bind-controls-to-data-in-visual-studio?view=vs-2015&redirectedfrom=MSDN> (visited on 01/22/2021) (cit. on p. 41).
- [31] Manoj Kushwah and Ashis Patra. «Tuning PID controller for speed control of DC motor using soft computing techniques-A review». In: *Advance in Electronic and Electric Engineering* 4.2 (2014), pp. 141–148 (cit. on pp. 46, 47).
- [32] Hugh Jack. «Dynamic system modeling and control». In: *Draft Ver 2* (1993) (cit. on p. 47).
- [33] Arduino LLC. *Arduino Software (IDE) "analogRead()" Description*. 2021. URL: <https://www.arduino.cc/reference/en/language/functions/analog-io/analogread/> (visited on 01/24/2021) (cit. on p. 50).
- [34] WIKA LLC. *Pressure transmitter, model A-10 description*. 2021. URL: https://www.wika.us/upload/DS_PE8160_en_co_1631.pdf (visited on 01/24/2021) (cit. on p. 52).