Gezamenlijke opleiding UHasselt en KU Leuven

Faculteit Industriële Ingenieurswetenschappen

master in de industriële wetenschappen: energie

Masterthesis

Fabrication and Implementation of an Operando Gas Analysis Setup for Pouch Cell Batteries, and Its Application in the Characterization of Gas Evolution

Senna Nijs
Karnpreet Singh

Scriptie ingediend tot het behalen van de graad van master in de industriële wetenschappen: energie

PROMOTOR :
Prof. dr. Mohammadhosein SAFARI

PROMOTOR :
dr. Saeed YARI

2024
2025

Faculteit Industriële Ingenieurswetenschappen

master in de industriële wetenschappen: energie

Masterthesis

Fabrication and Implementation of an Operando Gas Analysis Setup for Pouch Cell Batteries, and Its Application in the Characterization of Gas Evolution

Senna Nijs
Karnpreet Singh

Scriptie ingediend tot het behalen van de graad van master in de industriële wetenschappen: energie

PROMOTOR :

Prof. dr. Mohammadhosein SAFARI

PROMOTOR :

dr. Saeed YARI

Preface

This master's thesis is the result of a unique collaboration between two students from different specializations within the program of Engineering Technology. Senna Nijs, enrolled in the Master of Electromechanical Engineering Technology – Design & Production (Ma IIW EM-OP), and Karnpreet Singh, enrolled in the Master of Energy Engineering Technology (Ma IIW EN), have combined their complementary expertise to approach this research topic from a multidisciplinary perspective. The integration of our respective backgrounds allowed us to tackle the various aspects of this project with depth and precision, resulting in a final work that reflects both design-oriented thinking and energy-focused analysis.

We would like to sincerely thank our promoter, Prof. dr. ir. Momo Safari, whose support and advice greatly enhanced the quality and rigour of our research and methodology. We are equally grateful to our external promotor and supervisor, dr. Saeed Yari for his continuous guidance and valuable insights throughout the course of this research. His expertise and constructive feedback were crucial in shaping both the direction and the outcome of this work.

Special thanks are due to Simon-Pierre Verpoort for his valuable support and insightful contributions regarding the manufacturing of custom components.

In addition, we wish to acknowledge IMO-IMOMEC for their assistance and providing us necessary components. Additionally, we would like to thank EnergyVille for the usage of their infrastructure.

Lastly, we would like to emphasize how rewarding this collaboration has been. Working together across disciplines not only made this project technically stronger but also personally enriching. We hope this thesis contributes meaningfully to the field and provides valuable insights for future researchers and engineers interested in the coupling of gas analysis techniques with lithium-ion pouch cell systems.

Contents

List of Tables

List of Figures

Abstract

IMO-IMOMEC conducts research on battery storage systems, including analyses of pouch cell batteries. During the cycling process, undesired gases are generated that can compromise battery performance and safety. The existing gas analysis setup is only suitable for small coin-cell size batteries, while pouch cells require manual gas extraction, i.e. with a syringe, leading to inconsistent measurements, leakages and the inability to perform operando measurements. This master thesis aims to develop an automated setup that enables precise and reliable real-time gas analysis of pouch cell batteries.

Firstly, a coupling mechanism is designed and fabricated to encapsulate the pouch cell. The mechanism allows fast battery exchange and ensures sealed gas and electric connections. Furthermore, the system incorporates solenoid valves for automated measurements and tubing lengths, reduced by 65% compared to previous setups to reduce the dead volume.

The completed setup functions autonomously and is configurable via the custom software application. Hence, the system not only allows for operando gas analysis but also eliminates the need of an operator during the cycling process if measurement at specific intervals are required.

Mass spectrometry leak tests confirm excellent integrity with air infiltration below 800 counts/s. Measurements further deliver reliable and reproducible data. In conclusion, this setup marks a major improvement over manual methods, providing a reliable tool for analysing gas production in pouch cell batteries.

Abstract in Dutch

IMO-IMOMEC voert onderzoek uit naar energieopslagsystemen, waaronder pouchcelbatterijen. Tijdens het laden en ontladen ontstaan ongewenste gassen die de prestaties en veiligheid van batterijen kunnen aantasten. De bestaande gasanalyse-opstelling is enkel geschikt voor kleine coincelbatterijen. Voor pouchcellen is handmatige gasextractie vereist, bijvoorbeeld met een spuit, wat leidt tot inconsistente metingen, lekkages en maakt operando-metingen onmogelijk. Deze masterproef ontwikkelt een geautomatiseerde opstelling die nauwkeurige en betrouwbare realtime gasanalyse van pouchcellen mogelijk maakt.

Eerst werd een koppelsysteem ontworpen om de pouchcel luchtdicht te omsluiten. Dit maakt snelle batterijwissels mogelijk en garandeert afgesloten gas- en elektrische aansluitingen. Daarnaast bevat het systeem solenoïdekleppen voor automatische metingen en werd de buislengte met 65% gereduceerd om het dood volume te minimaliseren.

De opstelling werkt autonoom en is instelbaar via een eigen softwaretoepassing. Hierdoor zijn operando-metingen mogelijk zonder tussenkomst van een operator, zelfs bij metingen op vaste tijdstippen.

Lektests met massaspectrometrie tonen een uitstekende integriteit aan, met luchtinfiltratie onder 800 counts/s. Metingen leveren bovendien betrouwbare en reproduceerbare gegevens op. Deze opstelling vormt een duidelijke verbetering ten opzichte van manuele methoden en biedt een betrouwbaar instrument voor de analyse van gasvorming in pouchcelbatterijen.

Chapter 1

Introduction

The accelerated shift toward renewable energy has significantly intensified the demand for efficient and sustainable energy storage solutions. Lithium-ion batteries, as a cornerstone of modern electrochemical storage technologies, have emerged as a critical component in facilitating this global energy transition.

However, their widespread adoption also brings forth complex scientific and engineering challenges. Among these, the formation of gases during battery operation presents a crucial safety and performance issue, particularly in pouch cell configurations.

This chapter introduces the broader context in which this research is situated, emphasizing the relevance of gas analysis in lithium-ion batteries and the limitations of current experimental setups. The underlying problem and motivation for the development of a novel measurements setup are subsequently presented. Thereafter, the specific research objectives and methodological approach adopted throughout this project are outlined. The chapter concludes with a structured overview of the thesis, providing a roadmap of the subsequent chapters and their respective contributions to the research.

1.1 Context

The global energy transition is becoming increasingly important to slow down global warming. Currently, electrification is becoming the most widely used way to reduce CO_2 formation. In this regard, the research institute EnergyVille in Genk (Belgium) is conducting research into innovative technologies and materials for green energy production. A dedicated team here specifically focuses on researching energy storage systems. In the energy transition, energy storage systems such as batteries play a crucial role. The increasing understanding of lithium-ion battery performance, safety, and longevity has played a key role in enabling their rapid development and widespread adoption in portable electronics and electric vehicles.

Due to the wide range of applications for batteries, they come in various forms and topologies. One important type is pouch cell batteries. Pouch cell batteries are lightweight, flexible, and have a high energy density, making them suitable for applications where space and weight are critical factors. They are commonly used in electric vehicles, consumer electronics, and renewable energy storage systems.

During the charging and discharging process of pouch cell batteries undesired gases are generated [1]. These gases can accumulate within the cell, causing it to swell and potentially leading to a decrease in battery performance or a safety hazard. Incidents involving thermal runaway, leading to explosions in consumer electronics and electric vehicles, have garnered significant attention in recent years [1].

This paper focuses on designing and implementing a real-time measurement setup for those gases using a mass spectrometer. A mass spectrometer is an advanced analytical instrument that detects and identifies gas emissions with high sensitivity and precision, capable of analysing a wide range of gaseous species. Using this tool, the released gas composition can be accurately identified and quantified [2].

Previous studies have been conducted on gas formation in batteries, using various measurement setups. For pouch cell batteries, relatively simple setups are currently used. Figure 1.1 illustrates a method used for pouch cell batteries. In this setup, the gases from the pouch cell are manually extracted using a syringe through a tube and a coupling connection [3].

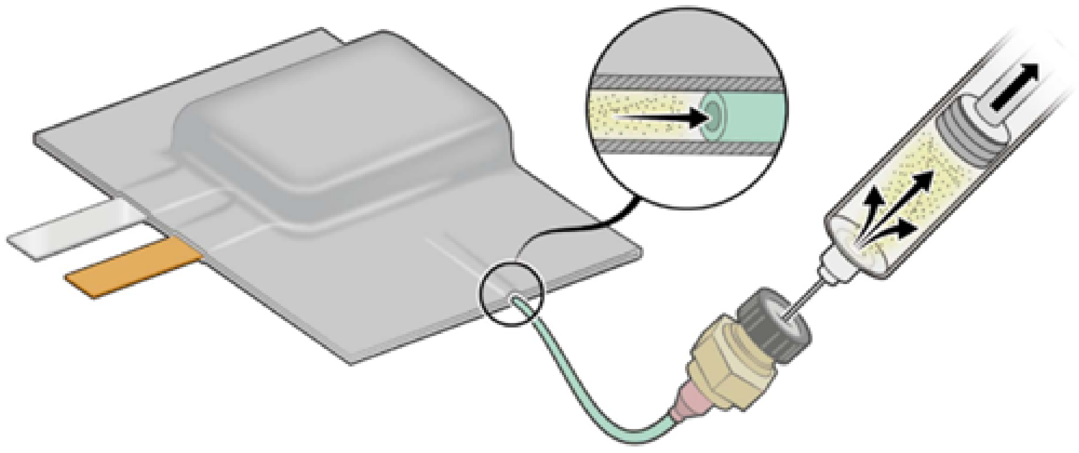


Figure 1.1: Schematic illustration of the manual gas sampling system for pouch cells [3]

1.2 Problem Statement

This project is particularly important for the battery research group of EnergyVille, which focuses on testing and optimizing the performance of energy storage systems. Due to the relative novelty of gas analysis techniques for batteries, research in this area has been limited.

To perform gas analysis on batteries, a mass spectrometer is used. However, connecting a mass spectrometer to pouch cell batteries, which come in a variety of shapes and sizes, presents a significant challenge. Creating a general and reliable connection for these different forms is crucial to accurately measure gas production during the charging and discharging processes.

Previous research has explored methods to connect a mass spectrometer to pouch cell batteries, but these studies used a rather simplistic coupling design [2]. This design comes with several limitations. For example, the produced gases have to be extracted manually from the pouch cells, which causes a less consistent measurement and more room for human mistakes. There is also an extraction method developed for coin cells. However, even this technique has challenges, such as frequent gas leaks caused by insufficient sealing of the tubes and the use of excessively long tubes. In fact, during preliminary tests, the presence of air leakage within the current setup was observed. Mass spectrometry measurements have revealed a significant concentration of atmospheric gases, particularly nitrogen and oxygen, inside the sampling line—indicating that ambient air is infiltrating the system. This compromises the integrity of the gas analysis by diluting or masking the signals of the gases evolved from the pouch cell itself. Such contamination not only reduces the accuracy of the quantitative analysis but also poses a risk of misinterpreting reaction pathways due to secondary gas formations.

Additionally, the current coupling device designed for coin cells is too small and incompatible in shape with pouch cells.

Another significant limitation is that, currently, the gases are released using a manual switch. This approach requires an operator to be physically present throughout the cycling process, that can take from one to 14 days of time based on the desired number of cycles and the C-rate.

1.3 Objectives

The primary objective of this master's thesis is to develop a coupling mechanism that enables accurate and reliable gas analysis of pouch cell batteries, that performs measurements autonomously without the presence an operator after starting. The ultimate goal is to create a system that allows for detailed real-time measurements of the gases released during charging and discharging, ensuring that these measurements are both precise and repeatable.

In order to achieve the research objectives, several specific goals must be met. First, a robust and airtight coupling mechanism must be designed to allow accurate and repeatable gas analysis of pouch cell batteries. The mechanism must be capable of capturing the gases generated within the pouch cell and directing them efficiently to the mass spectrometer for the analysis.

In addition, a method must be implemented to enable electrochemical cycling of the pouch cell while it is enclosed within the gas-tight containment mechanism. This requires the integration of reliable electrical feedthroughs that maintain the system's airtightness while allowing safe and uninterrupted connection to a battery cycler. The design must ensure minimal electrical resistance and avoid interference with the gas flow or mechanical sealing.

Furthermore, the coupling system must be constructed using chemically inert and non-reactive materials, such as stainless steel or perfluoroalkoxy alkane (PFA), to prevent interactions with the evolved gases. This material choice is essential to avoid contamination or the formation of secondary gas species, which could compromise the accuracy of the mass spectrometry analysis. Airtight sealing is not only essential to avoid leakage and to ensure that all generated gases are directed through the intended detection path but also to avoid the ingress of ambient air, which could contaminate the gas stream and interfere with accurate mass spectrometry measurements.

Secondly, the setup will be upgraded by integrating electronically actuated valves to remotely start gas measurements. To ensure both functional reliability and user safety, the electronic control system for the valve actuation must be designed according to best practices in embedded hardware engineering, with particular attention to fail-safe operation and electrical isolation where needed.

Moreover, a custom software interface will be developed, enabling the user to control valve operation remotely and define measurement conditions through various trigger options. These include time-based triggers, voltage thresholds, full-cycle, half-cycle and rest-period triggers. The software should support the simultaneous use of multiple trigger types and the software will log precise time and trigger-type data to allow post-synchronization with the data stream from the mass spectrometer.

By achieving these objectives, the goal is to design a coupling mechanism that is reliable and fully automated, ultimately allowing for accurate and repeatable gas analysis of pouch cell batteries.

1.4 Method

The first phase of this research involves the construction of a single layer anode and cathode pouch cell battery specially made for gas analysis using mass spectrometry using the materials and instruments provided by EnergyVille.

The first phase of this research involves designing the coupling using the Creo CAD Software. This mechanism will be built to meet the requirements outlined in the objectives.

Secondly, the electronics will be built to be able to control the solenoid valves. Additionally, the software to be able to control the valves remotely will be developed in Python. After this, a bill of materials will be made which will contain the costs of the materials that has been manufactured or ordered.

Progressively, a measurements setup will be built to connect the battery to the mass spectrometer integrating the coupling mechanism, the solenoid valves and the electronics to control them.

Subsequently, the mass spectrometer analyses will be conducted using the constructed setup. To this end, The results will be critically evaluated, and the mechanism will undergo further refinement based on the findings. The main objective of doing so is to ensure that the final coupling system gives reliable and consistent results.

In the final stage, analyses will be performed using the fully optimised setup on multiple pouch cells. The data obtained from these final tests will be examined in-depth.

1.5 Future Outlook

The subsequent chapters of this thesis will systematically present the design, implementation, and analysis of the test setup:

- **Chapter 2:** Provides an extensive literature review on the history and evolution of Li-ion battery technology, with a focus on current challenges and gas formation phenomena in battery cells.
- **Chapter 3:** Describes the mechanical design of the test cell, including material selection, fabrication procedures, and the integration of measurement instrumentation.
- **Chapter 4:** Discusses the electronics and automation system used in the test setup, including valve control and software implementation.
- **Chapter 5:** Details the fabrication procedure of the custom-made pouch cells, including materials, assembly steps, and quality control measures.

- **Chapter 6:** Presents the results obtained from the gas analysis experiments and provides an in-depth discussion of the findings in relation to the initial objectives and literature.
- **Chapter 7:** Concludes the thesis by summarizing the key findings, identifying limitations, and providing recommendations for future research.

Chapter 2

Literature Review

2.1 Introduction

2.1.1 Brief history of battery technology

Brief History of Battery Technology

The earliest known precursor of modern batteries is the Baghdad Battery. This artifact consists of a copper cylinder enclosing an iron rod within clay. It likely served for basic electroplating or chemical experiments in the Parthian era. Systematic investigation of electrochemical cells began in the late eighteenth century. In 1780 Luigi Galvani reported the phenomenon of animal electricity. In 1800 Alessandro Volta devised the voltaic pile. Volta's pile used alternating zinc and copper discs separated by pads soaked in electrolyte. This arrangement delivered the first stable and controllable source of electric current [4].

In 1836 John Frederic Daniell modified Volta's design by placing copper and zinc electrodes in separate compartments each with its own electrolyte. That innovation reduced the polarization that limited earlier cells. In 1859 Gaston Planté introduced the first lead acid accumulator. This invention inaugurated truly rechargeable cells. In the 1870s Georges Leclanché developed the dry cell and in 1899 Waldmar Jungner created the nickel cadmium battery. These advances powered telegraph networks and early portable lighting. Despite their impact these chemistries suffered from low energy density and significant weight. The search for higher specific energy culminated in 1991 when Sony commercialized the first lithium ion batteries. These devices combined rechargeability with greatly improved energy density and cycle life [4].

Historical Evolution of Lithium-Ion Battery Technologies

Since the first experimental lithium-ion cell appeared in the early 1970s the field has reached several key milestones. Figure 2.1 shows the main concepts and commercialization events through the decades. In the 1980s researchers introduced the rock-salt structure of $LiCoO_2$. In 1991 Sony launched the first commercial lithium-ion battery. Around the year 2000 $LiFePO_4$ and NMC cathode materials entered the market. After 2010 development shifted toward lithium-rich cathodes. During the latter half of that decade solid-state electrolytes and Li–S systems became the focus of intense research. More recent work addresses Li–air and Li–CO_2 cells aiming for maturation around 2030 [5].

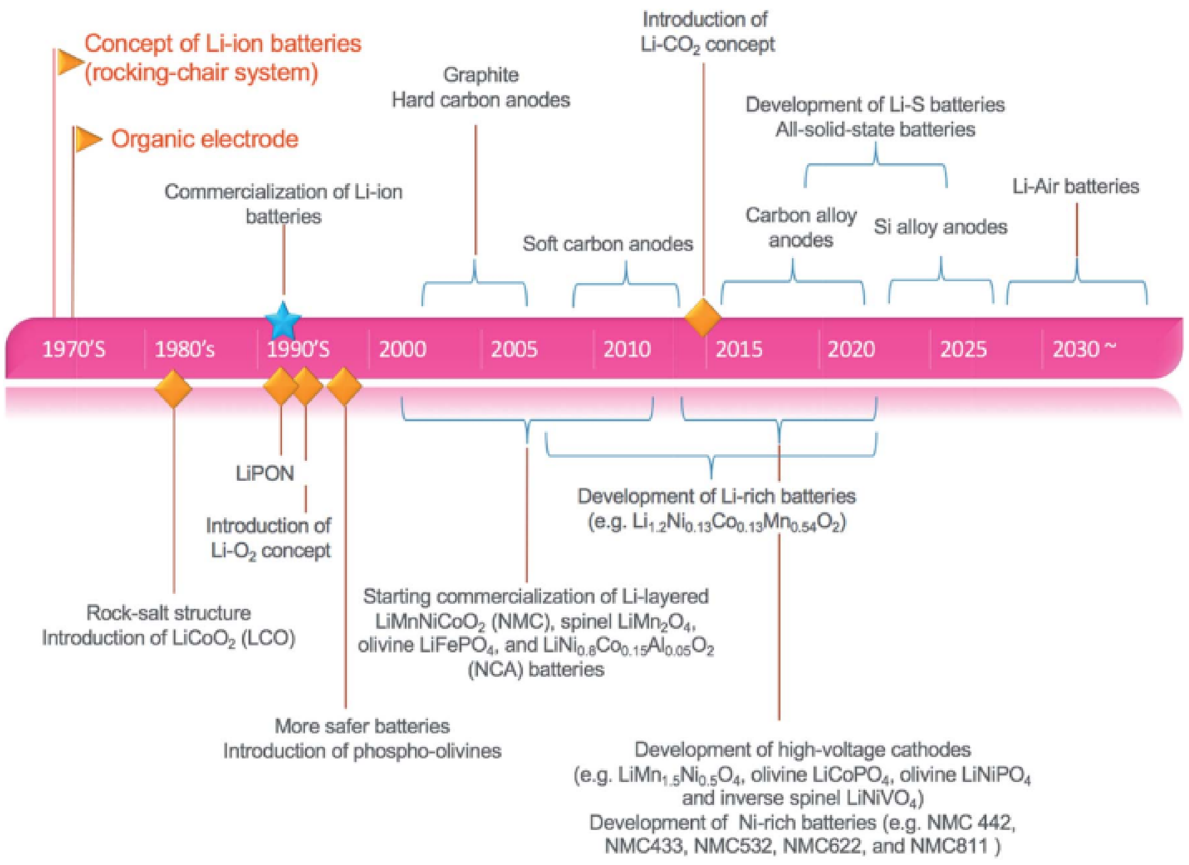

Concept of Li-ion batteries (rocking-chair system)
Organic electrode
Commercialization of Li-ion batteries
Graphite
Hard carbon anodes
Soft carbon anodes
Introduction of Li-CO$_2$ concept
Development of Li-S batteries
All-solid-state batteries
Carbon alloy anodes
Si alloy anodes
Li-Air batteries
1970'S
1980's
1990'S
2000
2005
2010
2015
2020
2025
2030 ~
LiPON
Introduction of Li-O$_2$ concept
Rock-salt structure
Introduction of LiCoO$_2$ (LCO)
Development of Li-rich batteries (e.g. Li$_{1.2}$Ni$_{0.13}$Co$_{0.13}$Mn$_{0.54}$O$_2$)
Starting commercialization of Li-layered LiMnNiCoO$_2$ (NMC), spinel LiMn$_2$O$_4$, olivine LiFePO$_4$, and LiNi$_{0.8}$Co$_{0.15}$Al$_{0.05}$O$_2$ (NCA) batteries
More safer batteries
Introduction of phospho-olivines
Development of high-voltage cathodes (e.g. LiMn$_{1.5}$Ni$_{0.5}$O$_4$, olivine LiCoPO$_4$, olivine LiNiPO$_4$ and inverse spinel LiNiVO$_4$)
Development of Ni-rich batteries (e.g. NMC 442, NMC433, NMC532, NMC622, and NMC811)

Figure 2.1: Historical evolution and key milestones of lithium-ion battery technology. [5].

2.1.2 The significance of Li-ion batteries in modern applications

Applications of Lithium-Ion Batteries Lithium-ion batteries have become the primary energy source for portable electronics such as smartphones and laptops due to their high gravimetric and volumetric energy densities [6]. By employing non-aqueous, water-free electrolytes and lithium salts, cell voltages near 4 V are achieved—far exceeding the ¡ 2 V limit of aqueous systems, and this enables significantly higher energy storage per unit mass and volume.

Beyond consumer devices, LIBs are critical for grid-scale energy storage applications, where they support renewable integration, peak-shaving and frequency regulation. In the transportation sector, electric vehicles (EVs) and hybrid electric vehicles (HEVs) rely on LIBs for their combination of power density, cycle life and efficiency. Emerging fields include aerospace power systems and self-powered autonomous sensors, each demanding careful optimization of energy density, safety, cost, cycle life, charge–discharge rates and environmental impact to meet specific operational requirements [6].

2.2 Basics of Li-ion Batteries

2.2.1 Components of Lithium Ion Batteries

Lithium ion cells consist of several key parts. Figure 2.2 shows a pouch type cell. This cell uses a flexible polymer bag as casing. The bag contains the electrode stack and the electrolyte. Two metal tabs connect each electrode to the external circuit. One tab is attached to the cathode foil and the other to the anode foil.

The cathode is an aluminum foil coated with a lithium containing active material that releases lithium ions during discharge. The anode is a copper foil coated with graphite layers that host lithium ions during charge. A porous separator lies between the two electrodes. It prevents electronic contact while allowing lithium ions to move. The electrolyte is a solution of lithium salt in organic solvents. It provides the medium for ionic transport throughout the cell [7].

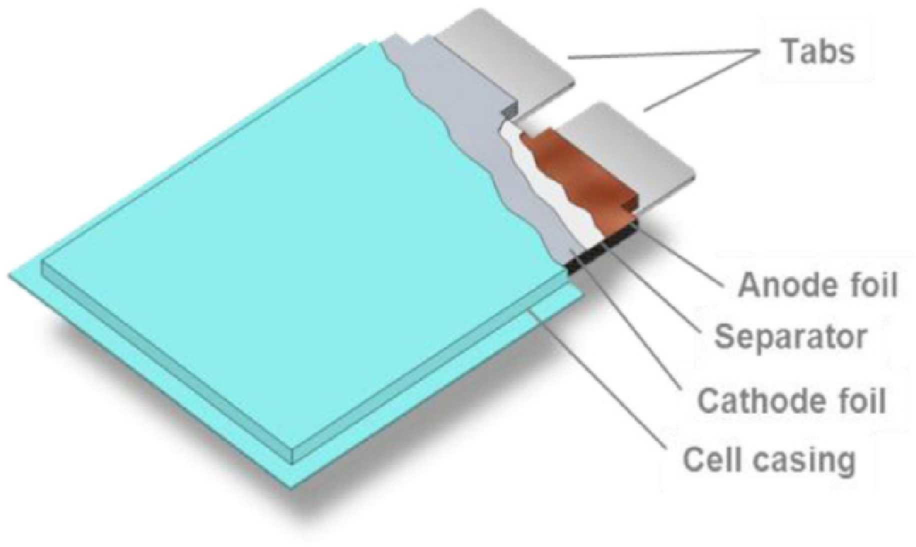


Figure 2.2: Structure of a pouch type lithium ion cell showing the cathode foil, separator, anode foil, cell casing and current collector tabs. Source: Zubi *et al.* [7]

2.2.2 Working Principle of Lithium Ion Batteries

The operation of a lithium ion battery is based on reversible movement of lithium ions between two electrodes. The anode serves as the negative electrode and the cathode acts as the positive electrode. During discharge lithium ions migrate from the anode through the electrolyte and separator to the cathode. Electrons travel via the external circuit to supply power to an electrical load. Charging reverses these processes by driving lithium ions back to the anode and returning electrons through the negative terminal [8].

The chemical changes in each electrode can be expressed by half reactions. At the cathode lithium is removed from its host lattice during charge and reinserted during discharge. At the anode lithium is inserted into graphite layers during charge and extracted during discharge. For a lithium cobalt oxide cathode and a graphite anode the reactions read

Energy storage in these cells relies on repeated intercalation and deintercalation of lithium ions into layered host structures. The nearly constant potential during these processes yields a flat discharge profile. Losses arise from ionic resistance in the electrolyte and electronic resistance in the electrodes and current collectors. Careful selection of electrode materials and electrolyte composition can reduce these losses and enhance efficiency and cycle life [8].

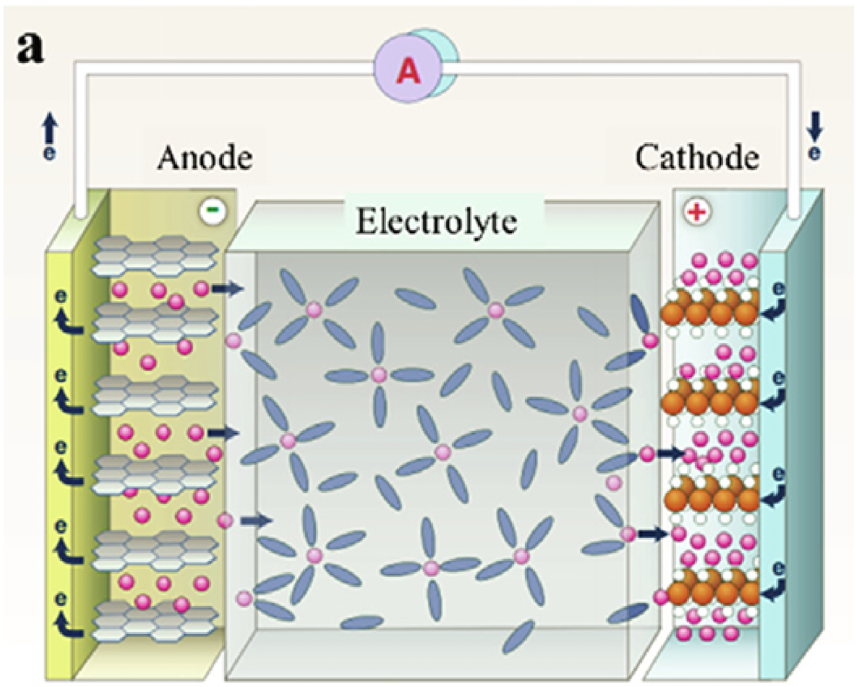


Figure 2.3: Diagram of lithium ion transport and electron flow during discharge and charge. Source: [8]

$$\text{Cathode}: \ LiCoO_2 \underset{\text{Discharge}}{\overset{\text{Charge}}{\rightleftharpoons}} Li_{(1-x)}CoO_2 + xLi^+ + xe^-$$

$$\text{Anode}: \ 6C + xLi_+ + xe^- \underset{\text{Discharge}}{\overset{\text{Charge}}{\rightleftharpoons}} Li_xC_6$$

$$\text{Reactions}: \ LiCoO_2 + 6C \underset{\text{Discharge}}{\overset{\text{Charge}}{\rightleftharpoons}} Li_{(1-x)}CoO_2 + Li_xC_6$$

Figure 2.4: Charge and discharge reactions at cathode and anode. Source: [8]

2.3 Advantages of Li-ion Batteries

Advantages of Lithium-Ion Batteries

Outstanding specific energy and power. Lithium-ion batteries store more energy per unit mass than most other rechargeable systems. They also sustain significant current output without large voltage drop. This combination makes them ideal for both portable electronics and electric vehicles [7].

Long calendar and cycle lives. These cells retain capacity over hundreds to thousands of charge cycles. The stable electrode materials and optimized electrolytes limit degradation. As a result they offer durable performance over many years [7].

High round trip efficiency. Energy losses during charge and discharge remain low in lithium-ion cells. Typical one-way efficiency exceeds ninety percent. This efficiency reduces wasted energy and lowers operating costs [7].

Low operation and maintenance requirements. After installation these systems require little intervention. They do not need regular fluid topping or complex balancing. This simplicity lowers lifecycle cost and eases system management [7].

Table 2.1: Key advantages of lithium-ion batteries [7]

Outstanding specific energy and power
Long calendar and cycle lives
High round trip efficiency
Low operation and maintenance requirements
Satisfactory operating temperature ranges
High reliability
Technological diversity with multiple chemistries
Intensive global research and development efforts
Availability of eco friendly materials
Reasonable self discharge rate
Relatively fast recharge

Satisfactory operating temperature ranges. Most chemistries operate effectively from minus twenty to plus sixty degrees Celsius. Performance losses outside this band remain moderate. This robustness suits diverse climates and applications [7].

High reliability. Properly managed cells rarely fail unexpectedly. Built-in safety features prevent damage from overcharge or temperature extremes. These safeguards ensure dependable operation in critical systems [7].

Technological diversity with multiple chemistries. A wide range of cathode and anode materials exists to tune energy, power and cost. This diversity allows designers to select the best chemistry for each use case. It supports tailored solutions across industries [7].

Intensive global research and development efforts. Both academic and industry teams continually explore new materials and architectures. Innovations aim to boost energy, lifetime and safety. Ongoing work promises further gains and cost reductions [7].

Availability of eco friendly materials. Some modern chemistries replace scarce or toxic metals with abundant elements. For example $LiFePO_4$ uses iron and phosphate. These choices reduce environmental impact and improve sustainability [7].

Reasonable self discharge rate. Lithium-ion cells lose only a small fraction of charge when idle. Typical self discharge is under five percent per month. This makes them suitable for standby and backup applications [7].

Relatively fast recharge. Many cells reach eighty percent state of charge within thirty minutes. High ion conductivity and electrode design support rapid charging. Fast recharge enhances system availability and user convenience [7].

2.4 Challenges in Li-ion Battery Technology

Despite their widespread success and high energy density, lithium-ion batteries face significant challenges that limit their performance, safety, and sustainability. One of the biggest challenges lies in capacity degradation over time, which reduces the effective lifespan of the battery and complicates state-of-health predictions. High charging rates, low temperatures, and elevated voltages can accelerate this degradation, posing a trade-off between performance and durability [9]. Another major issue concerns safety, especially related to thermal runaway (TR) events triggered by internal short circuits or uncontrolled side reactions. The use of flammable electrolytes and the potential for lithium plating and dendrite formation further amplify safety concerns. This section focuses on the degradation mechanisms and safety issues that most critically affect lithium-ion battery performance over time.

2.4.1 Degradation Mechanisms

Lithium-ion battery degradation can be conceptualized on three analytical levels: mechanisms, modes, and observable effects. Mechanisms represent the physical and chemical changes occurring at the microstructural level, such as the formation of passivation layers or structural breakdown of active materials. Modes group these mechanisms based on their functional consequences—for instance, loss of active material (LAM), loss of lithium inventory (LLI), impedance increase, and stoichiometric drift [10]. Finally, these modes manifest as the observable effects of degradation, notably capacity and power fade.

This multilevel framework helps in deciphering how local material changes evolve into performance losses, and why certain conditions exacerbate degradation disproportionately. The degradation modes themselves are tightly linked; for instance, LLI and LAM may occur simultaneously and lead to stoichiometric drift, thereby accelerating further degradation. Figure 2.5 shows the location and consequences of the de degradation mechanisms in Li-ion baterry cells, with primary mechanisms labelled in green and secondary mechanisms labelled in dark red.

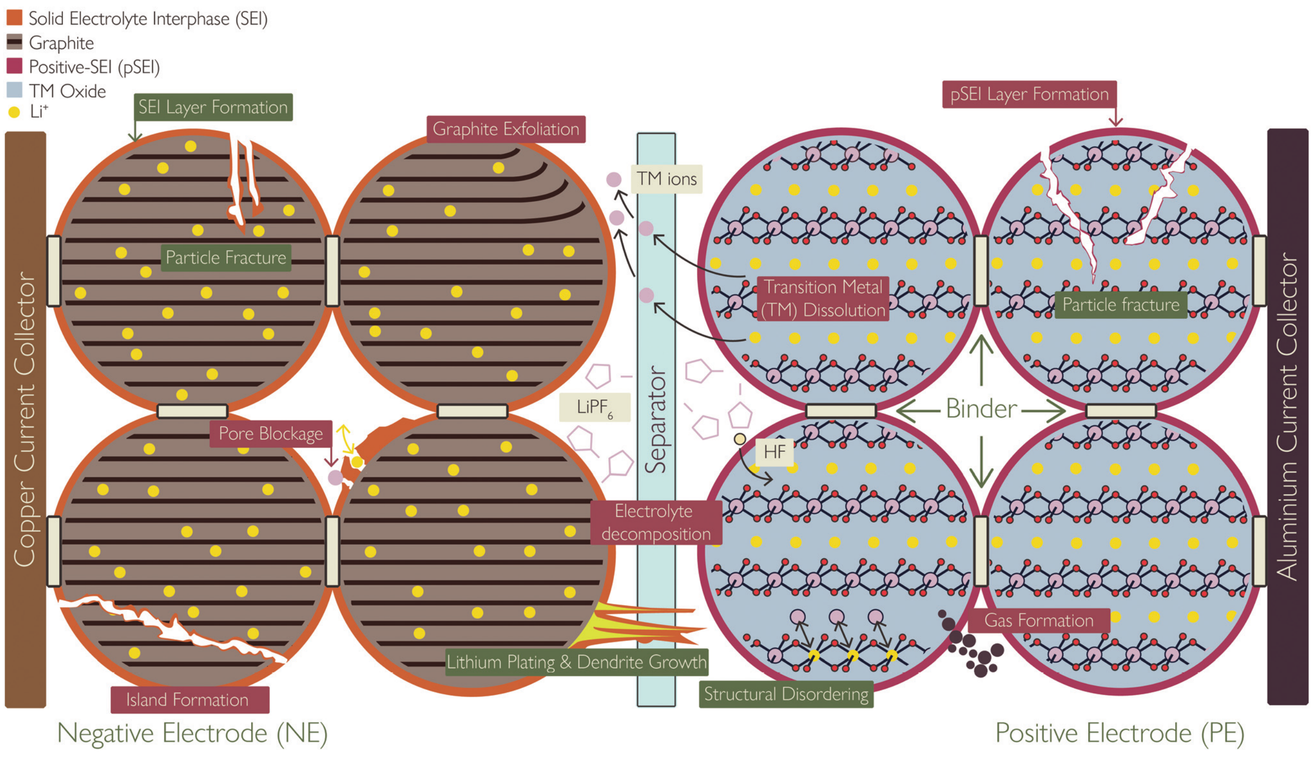


Figure 2.5: Schematic showing the location and consequences of the degradation mechanisms in Li-ion battery cells [11].

SEI Layer Formation and Growth

One of the earliest and most significant degradation mechanisms in LIBs is the growth of the solid electrolyte interphase (SEI) on the negative electrode [11].

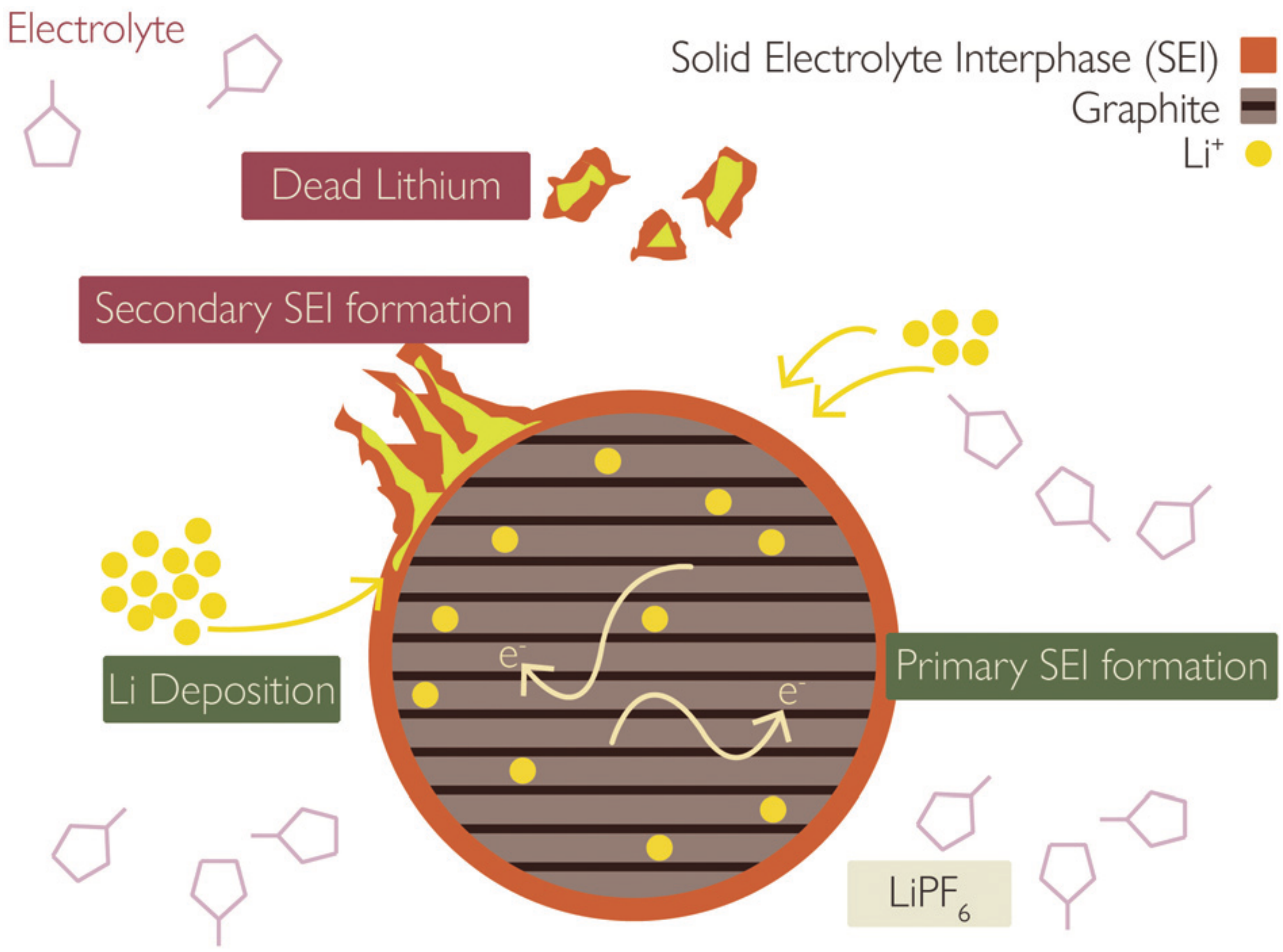


Figure 2.6: Schematic showing the various consequences and causes of transition metal dissolution, including the link with both SEI and pSEI formation and growth [11].

This layer, formed primarily during the initial charging cycles, results from the reduction of electrolyte solvents at the low potentials typical of graphite-based electrodes. While the SEI serves a protective function by preventing further electrolyte decomposition, its growth over time continues due to several reinforcing factors. These include solvent diffusion through the SEI, exposure of new electrode surfaces via particle cracking, and catalytic reactions involving dissolved transition metals.

The SEI is composed of a heterogeneous mixture of organic and inorganic compounds, including lithium carbonate, lithium ethylene dicarbonate, and lithium fluoride. Its growth consumes cyclable lithium, resulting in LLI, and increases ionic resistance, which contributes to power fade. Although it is a passive layer by design, the SEI can itself crack due to the mechanical stress of cycling, exposing fresh graphite surfaces and initiating further SEI growth—a feedback loop that significantly impacts cell longevity

Lithium Plating and Dendrite Growth

Another highly impactful degradation mechanism is lithium plating, where lithium deposits as a metal on the surface of the anode rather than intercalating into the graphite [12]. This typically occurs during fast charging, particularly at low temperatures, or when the anode becomes fully lithiated and cannot accommodate additional lithium. The phenomenon is both thermodynamically and kinetically driven, with high overpotentials and low lithium diffusivity in graphite being the main triggers.

Once plated, lithium metal is highly reactive with the electrolyte and rapidly forms additional SEI layers. If the deposited lithium becomes electrically isolated, either by being covered in SEI or detaching, it transforms into so-called "dead lithium," which no longer participates in the charge-discharge cycles. Lithium plating can also induce dendritic growth, where metallic filaments protrude into the separator, potentially causing internal short circuits and TR [11]. This mechanism is thus not only a contributor to LLI and impedance increase, but also a key safety concern in high-performance battery systems.

Positive Electrode Decomposition and Phase Transitions

While degradation at the negative electrode has been extensively studied, structural and chemical transformations at the positive electrode (typically layered transition metal oxides like NMC) are equally critical. These materials undergo complex degradation processes during high-voltage operation and repeated cycling, especially at high states of charge.

One of the central phenomena is phase transformation, where the layered structure of NMC transforms into spinel or rock salt phases [12, 13]. These new phases are less capable of lithium intercalation and present higher resistance to lithium-ion transport. This transformation is often accompanied by the release of lattice oxygen, which reacts with electrolyte components to form gaseous by-products such as CO_2 and O_2, further destabilizing the cell.

Concurrently, transition metal dissolution, particularly of Ni, Mn, and Co, occurs at the cathode surface and leads to their migration through the electrolyte to the anode. Once deposited on the graphite electrode, these metal ions catalyse further SEI formation and exacerbate impedance rise. Additionally, the similarity in ionic radii between Ni^{2+} and Li^{+} promotes cation mixing within the lattice, hindering lithium transport and raising cell impedance.

To some extent, the positive electrode develops a passivation layer analogous to the SEI, termed the cathode electrolyte interphase (CEI or pSEI). This layer forms from reactions between dissolved transition metals and fluoride ions in the electrolyte. Like its counterpart on the anode, the pSEI contributes to impedance rise and electrolyte consumption, adding to the overall degradation burden.

Mechanical Degradation: Particle Fracture

Mechanical degradation manifests most prominently as particle fracture, affecting both electrodes but especially significant in systems incorporating silicon-based anodes[14]. Silicon, while offering a much higher theoretical capacity than graphite, undergoes extreme volume expansion (up to 300%) during lithiation. This causes severe mechanical stress within particles, leading to fragmentation, loss of electrical contact, and irreversible capacity loss[15].

Even in graphite and NMC-based systems, repeated cycling causes expansion and contraction, which over time generates microcracks in active material particles. These cracks not only lead to the mechanical isolation of material fragments but also expose fresh surfaces to the electrolyte. This initiates further SEI or pSEI growth and accelerates degradation via the mechanisms described above.

Advanced imaging studies have confirmed the correlation between particle fracture and capacity fade. Mechanical modelling also shows that as stress accumulates, isolated regions form with higher current density, further increasing localized heating and fracture—a clear example of a positive feedback loop at the micro-scale.

2.4.2 Safety Issues

Thermal Runaway

TR represents the most severe safety concern in lithium-ion battery technology, characterized by an irreversible failure process that leads to the production of flammable gases and potential fire or explosion [16]. The TR process occurs when heat generated by exothermic reactions exceeds the heat dissipated to the environment, creating a self-sustaining reaction cascade (see Figure 2.7).

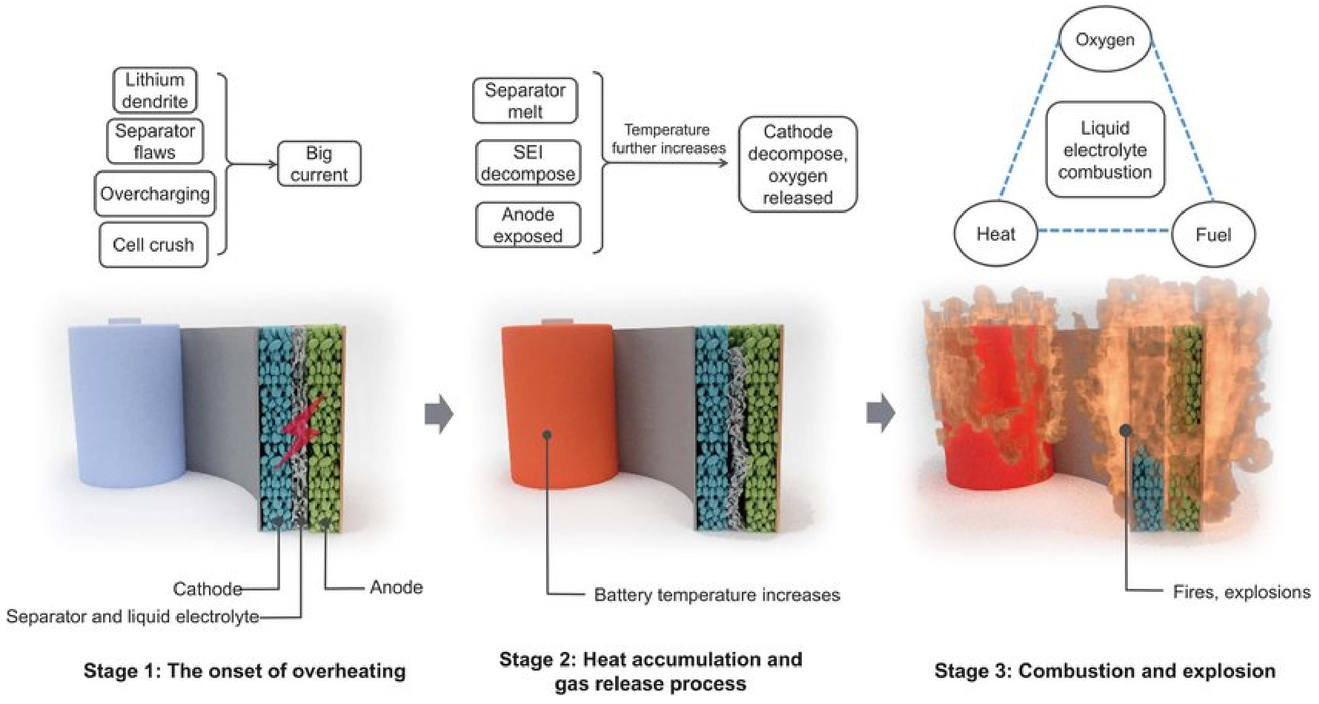

Lithium dendrite
Separator flaws
Overcharging
Cell crush
Big current
Separator melt
SEI decompose
Anode exposed
Temperature further increases
Cathode decompose, oxygen released
Oxygen
Liquid electrolyte combustion
Heat
Fuel
Cathode
Anode
Separator and liquid electrolyte
Battery temperature increases
Fires, explosions
Stage 1: The onset of overheating
Stage 2: Heat accumulation and gas release process
Stage 3: Combustion and explosion

Figure 2.7: Depiction of TR process initiation. [17]

The initiation of TR typically begins with the breakdown of the SEI layer, which serves as a passive stabilizing layer over the anode surface. Once compromised, continued electrolyte degradation occurs, leading to progressive heat generation and eventual TR. Various abuse conditions can trigger this process, including short circuits, mechanical damage, overcharging, over-discharging, and overheating [18, 16].

More key findings on gas generation during TR in aged lithium-ion batteries are discussed in 2.5.3.

Short Circuits

Short circuit conditions represent another critical safety concern, with lithium-ion batteries typically exhibiting much higher short circuit currents compared to traditional lead-acid batteries due to their lower internal resistance [19]. Experimental measurements have shown that lithium-ion batteries can produce short circuit currents exceeding 3.3 kA, significantly higher than the approximately 1.5 kA observed in comparable lead-acid systems.

The internal resistance of lithium-ion batteries can be as low as 5-10 mΩ for complete battery packs, with individual cells potentially reaching 0.3 mΩ each. This low resistance, while beneficial for efficiency and performance, creates substantial safety challenges during fault conditions. The high current capability necessitates robust protection systems and careful circuit design to manage fault currents safely.

2.5 Understanding Battery Degradation and Failure

2.5.1 Aging Processes in Li-ion Batteries

Battery aging represents a complex interplay of multiple degradation mechanisms that occur simultaneously during operation. The aging process involves both calendric aging (time-dependent degradation during storage) and cyclic aging (degradation due to charge-discharge cycling). Understanding these processes requires detailed analysis of the electrochemical, mechanical, and thermal factors that contribute to performance decline over time.

The SEI layer plays a central role in aging processes, as it continues to evolve throughout the battery's operational life. Computational studies have shown that SEI formation is not a one-time event but rather an ongoing process involving continuous electrolyte decomposition and film growth [20]. The composition and thickness of the SEI layer directly impact lithium-ion transport kinetics, with thicker or more resistive SEI layers leading to increased impedance and reduced capacity [21].

Aging also involves structural changes in electrode materials, including particle cracking, volume expansion and contraction during cycling, and gradual loss of active material. These mechanical degradation modes are often coupled with electrochemical processes, creating complex feedback mechanisms that accelerate overall degradation [21].

2.5.2 Chemical and Electrochemical Reactions During Cycling

The cycling process in lithium-ion batteries involves numerous parallel chemical and electrochemical reactions beyond the primary lithium intercalation/deintercalation reactions. Side reactions play a crucial role in determining battery lifetime and performance characteristics. These reactions include continued electrolyte decomposition, transition metal dissolution, and gas-generating reactions.

During normal cycling, the potential window and current density significantly influence the types and rates of side reactions. At high potentials, electrolyte oxidation becomes more favourable, while at low potentials, reduction reactions dominate. The dynamic nature of these reactions means that the battery's internal chemistry continuously evolves throughout its operational life [20].

Computational reaction networks have identified over 900 possible elementary reactions that can occur within lithium-ion battery systems [20]. These reactions compete with each other and with the primary electrochemical reactions, with their relative rates determining the overall battery behaviour and degradation patterns.

2.5.3 Gas Generation in Cells During Aging and Abuse Conditions

Gas generation represents a critical aspect of battery degradation and failure, occurring through multiple pathways during both normal operation and abuse conditions. During normal aging, gas generation primarily results from electrolyte decomposition reactions and continued SEI formation [18]. Experimental studies reveal that gas evolution in LFP batteries during aging primarily consists of:

- Hydrogen (H_2) - 40-60% of total gas volume [22]
- Carbon monoxide (CO) - 15-25%
- Carbon dioxide (CO_2) - 10-20%
- Hydrocarbons (CH_4, C_2H_4) - 5-15%
- Trace oxygen (O_2) - S <1%

Hydrogen Formation

The dominant H_2 generation originates from two primary pathways:

$$2\mathrm{H_2O} + 2e^- \rightarrow \mathrm{H_2} + 2\mathrm{OH}^- \quad \text{(Water reduction)[23]} \tag{2.1}$$

$$\mathrm{EC/LiPF_6} + \mathrm{H_2O} \rightarrow \mathrm{HF} + \mathrm{POF_3} + \mathrm{H_2} \quad \text{(Electrolyte hydrolysis)[24]} \tag{2.2}$$

Aging accelerates H_2 production due to:

- SEI layer degradation exposing fresh anode surfaces
- Moisture ingress through pouch cell seams
- Lithium plating side reactions [22]

Carbon Oxide Formation

CO and CO_2 generation occurs through electrolyte decomposition:

$$\mathrm{EC} \rightarrow \mathrm{CO} + \mathrm{C_2H_4} + \mathrm{Li_2CO_3} \quad (\Delta T > 60^\circ\mathrm{C})[25] \tag{2.3}$$

$$\mathrm{LiPF_6} + \mathrm{H_2O} \rightarrow \mathrm{POF_3} + 2\mathrm{HF} + \mathrm{CO_2} \quad \text{(Salt hydrolysis)[23]} \tag{2.4}$$

Hydrocarbon Generation

Ethylene (C_2H_4) forms through electrolyte decomposition (see Equation 2.3).

Methane (CH_4) production correlates with:

$$\mathrm{DMC} + e^- \rightarrow \mathrm{CH_4} + \mathrm{Li_2CO_3} \quad \text{(Solvent decomposition)[23]} \tag{2.5}$$

Ethylene carbonate (EC) and dimethyl carbonate (DMC) are organic solvents widely used in lithium-ion battery electrolytes. EC ($C_3H_4O_3$) is a cyclic carbonate with high dielectric constant, enabling effective lithium salt dissociation, while DMC ($C_3H_6O_3$) is a linear carbonate that reduces electrolyte viscosity and enhances ionic conductivity. Both solvents participate in solid-electrolyte interphase (SEI) formation and electrolyte decomposition reactions during battery operation.

However the gas composition changes substantially during aging. Table 2.2 shows long-term cycling induced changes.

Table 2.2: Gas composition evolution during aging (Data adapted from [22, 25])

Gas	**Fresh Cell (vol%)**	**Aged Cell (500 cycles)**
H_2	52.4	38.2
CO	18.7	27.4
CO_2	15.1	21.8
C_2H_4	9.3	8.1
CH_4	4.5	4.5

Key aging-related changes are:

- 40-60% H_2 reduction due to lithium inventory loss
- 50-70% CO/CO_2 increase from cumulative electrolyte decomposition and cathode surface oxidation [24]
- Hydrocarbon stabilization after initial SEI formation [24]

Under abuse conditions, particularly during TR, gas generation increases dramatically. Experimental measurements during TR events have quantified the concentrations of evolved gases, showing significant production of hydrogen (highly flammable), carbon monoxide (toxic), carbon dioxide, and hydrocarbon species [18]. The total volume and composition of these gases determine the flammability characteristics and potential explosion hazards.

The rate and extent of gas generation depend strongly on temperature, state of charge, and the specific abuse condition. Higher temperatures and states of charge generally lead to increased gas production rates and more diverse gas compositions. Understanding these relationships is crucial for developing safety systems and designing containment strategies for battery installations [18].

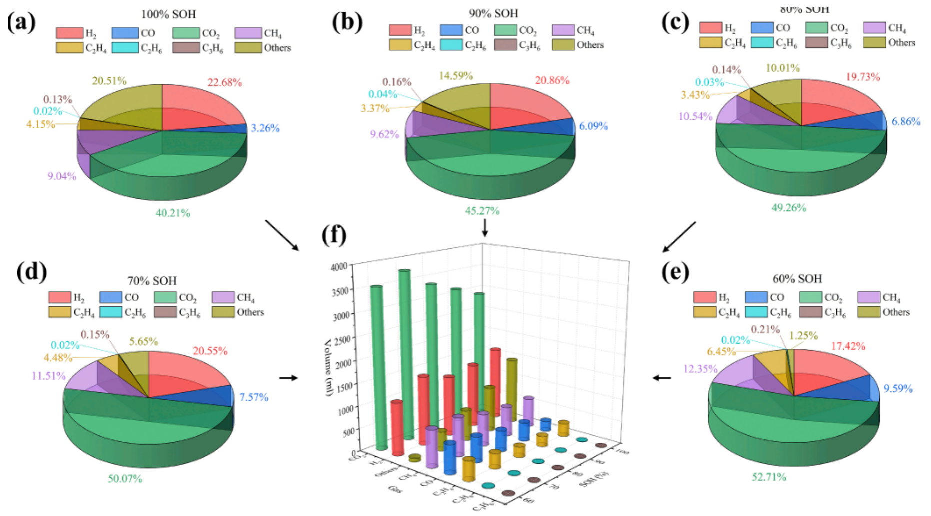


Figure 2.8: TR gas generation component and volume statistics with different SOHs. (a) Proportion of gas generation components of 100 % SOH; (b) Of 90 % SOH; (c) Of 80 % SOH; (d) Of 70 % SOH; (e) Of 60 % SOH; (d) Actual volume of gas generation components [22].

Overall, the key findings on gas generation during TR in aged lithium-ion batteries reveal a shift in gas composition as the state of health (SOH) declines [22]. Hydrogen (H_2) volume decreases with aging due to reduced lithium availability for reactions with binders, while carbon monoxide (CO) and carbon dioxide (CO_2) increase significantly, driven by enhanced reactions involving plated lithium and degraded cathode materials. Hydrocarbon gases like methane (CH_4) and ethylene (C_2H_4) also rise, contributing to flammability risks. Notably, aged cells (e.g., 60% SOH) exhibit a 40–60% drop in H_2 but a 2–3× increase in CO/CO_2 compared to fresh batteries, linked to electrolyte decomposition and cathode-derived oxygen reactions. This non-linear evolution in gas profiles underscores the need for tailored safety protocols, as reduced hydrogen lowers explosion risks but heightened CO/CO_2 and hydrocarbons amplify toxicity and combustion hazards.

The heating rate during TR initiation significantly affects the severity and characteristics of the event. Studies using various heating rates (5°C/min to 40°C/min) have shown that heating rates between 12.8 to 16.4°C/min can result in a mixture of standard and violent TR reactions [18]. Lower heating rates (less than 12°C/min) tend to produce less violent reactions, while higher rates increase the risk of explosive behaviour.

Furthermore, overcharging conditions significantly increase gas generation rates by driving electrolyte decomposition reactions beyond their normal operating ranges. At high potentials, water contamination in the electrolyte can lead to oxygen evolution, while continued charging beyond the normal voltage limits causes accelerated electrolyte breakdown and increased gas production.

2.6 Methods of Analysing Li-ion Batteries

2.6.1 Structural Analysis

Structural analysis techniques provide essential insights into the physical and chemical changes occurring within lithium-ion batteries during operation and degradation. X-ray diffraction (XRD) and scanning electron microscopy (SEM) represent two of the most important structural characterization methods for battery research.

In situ and operando XRD techniques allow real-time monitoring of structural changes in electrode materials during cycling. These methods can track phase transitions, lattice parameter changes, and the formation of new crystalline phases. The ability to observe these changes in real-time provides crucial information about degradation mechanisms and their kinetics.

SEM provides high-resolution imaging (see Figure 2.9) of electrode morphology, SEI formation, and dendrite growth. Recent advances in environmental SEM and in situ SEM techniques have enabled direct observation of dynamic processes, including real-time dendrite formation and SEI evolution. These techniques are particularly valuable for understanding the three-dimensional nature of degradation processes [25].

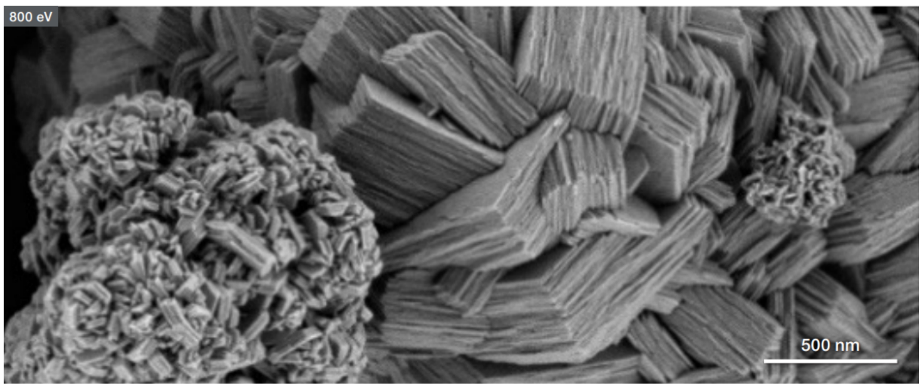


Figure 2.9: Imaging example with SEM [26].

Advanced synchrotron-based techniques, such as Dark-Field X-ray Microscopy (DFXM), provide unprecedented insights into the mechanical aspects of battery degradation. These methods can map strain fields and dislocation networks around defects such as dendrites, revealing the coupling between electrochemical and mechanical processes [27].

2.6.2 Electrochemical Analysis

Electrochemical impedance spectroscopy (EIS) and galvanostatic intermittent titration technique (GITT) represent fundamental electrochemical analysis methods for battery characterization. EIS provides information about the various resistance and capacitance components within the battery, including SEI resistance, charge transfer resistance, and diffusion limitations.

GITT enables measurement of lithium-ion diffusion coefficients within electrode materials, providing insights into the kinetic limitations of the battery system. These measurements are crucial for understanding rate capability and how diffusion properties change during aging and degradation.

The combination of multiple electrochemical techniques allows comprehensive characterization of battery behaviour under various conditions. These methods can track changes in performance parameters over time, providing quantitative measures of degradation rates and mechanisms.

2.7 Role of Gas Evolution in Li-ion Battery Behaviour

2.7.1 Sources of Gas Generation

Gas generation in lithium-ion batteries occurs through multiple pathways, with electrolyte decomposition representing the primary source under most conditions. During normal operation, the continuous growth of the SEI layer involves electrochemical reduction of electrolyte components, producing gases such as ethylene (C_2H_4), carbon monoxide (CO), and hydrogen (H_2) as byproducts as discussed in 2.5.3.

2.7.2 Impact of Gas Generation on Cell Performance and Safety

Gas generation has multiple impacts on battery performance and safety. From a performance perspective, gas generation represents a loss of active electrolyte, which can lead to increased impedance and reduced capacity over time. The gases produced can also accumulate within the cell, potentially causing mechanical stress and deformation of cell components.

Safety implications of gas generation are particularly significant during abuse conditions. The accumulation of flammable gases, particularly hydrogen and hydrocarbons, creates explosion hazards if the gases are released into confined spaces. The lower flammability limit (LFL) of the gas mixture determines the minimum concentration required for ignition [18].

Experimental measurements have quantified the flammability characteristics of TR gases, showing that the gas composition varies significantly with temperature and abuse conditions. The total volume of flammable gas produced during TR can create substantial safety hazards, particularly in enclosed battery installations [18].

2.7.3 Need for Precise Gas Analysis

The complex nature of gas generation in lithium-ion batteries necessitates precise analytical techniques capable of identifying and quantifying multiple gas species simultaneously. Real-time monitoring of gas evolution provides crucial information for understanding degradation mechanisms, predicting failure modes, and developing safety systems.

Mass Spectroscopy(MS), Fourier-transform infrared spectroscopy (FTIR), and non-dispersive infrared (NDIR) sensors are commonly used for gas analysis. FTIR enables simultaneous detection of various gases with good temporal resolution, but it cannot detect IR-inactive species such as H_2 and requires careful calibration [28]. NDIR sensors are compact and ideal for continuous monitoring of target gases like CO_2, but they lack versatility and selectivity [29]. Other tools like electrochemical sensors or colorimetric tubes are useful for specific applications, yet they often suffer from cross-sensitivity, limited accuracy, or one-time use [30]. MS offers detailed gas identification and is suitable for real-time analysis [31].

2.8 Introduction to Mass Spectroscopy

2.8.1 Basics of MS and How It Works

MS represents a powerful analytical technique for identifying and quantifying gas species based on their mass-to-charge ratios. The basic principle involves ionization of gas molecules, acceleration of the resulting ions in an electric field, and separation based on their mass-to-charge ratios using magnetic or electric fields [28].

For battery applications, MS typically employs electron impact ionization, where high-energy electrons collide with gas molecules to produce characteristic fragmentation patterns. These fragmentation patterns can serve as "fingerprints" for identifying specific gas species. However, in the absence of prior chromatographic separation, complex mixtures may require deconvolution or prior knowledge to resolve overlapping signals. [31].

The mass spectrometer consists of several key components: an ion source for ionization, a mass analyser for separation, and a detector for quantification. Different types of mass analysers, including quadrupole, time-of-flight, and magnetic sector instruments, offer various advantages depending on the specific analytical requirements [32].

2.8.2 Key Features: Sensitivity, Specificity, and Real-time Analysis Capability

MS offers several key advantages for battery gas analysis. High sensitivity allows detection of trace gas species at very low concentrations, which is crucial for early detection of degradation processes or safety hazards. Modern MS can detect gas concentrations in the parts-per-billion range or lower.

Specificity is achieved through the characteristic fragmentation patterns produced by different gas species. Even structurally similar compounds typically produce distinct mass spectra, allowing unambiguous identification of individual components in complex gas mixtures [33].

Real-time analysis capability is perhaps the most significant advantage for battery applications. MS can provide continuous monitoring of gas composition with response times on the order of seconds, enabling dynamic studies of gas evolution during battery operation or abuse conditions.

2.9 Application of MS in Li-ion Battery Analysis

2.9.1 Detection of Evolved Gases During Battery Operation and Failure

MS has emerged as a valuable tool for monitoring gas evolution during various battery operating conditions. During normal cycling, mass spectrometry can detect the gradual evolution of gases from SEI formation and continued electrolyte decomposition, providing insights into the aging process and degradation mechanisms.

The technique has proven particularly valuable for studying gas evolution during battery failure modes. Real-time monitoring during TR events can track the dynamic evolution of gas composition as the failure progresses, providing crucial data for understanding failure mechanisms and developing safety systems.

2.9.2 Insights into Electrolyte Decomposition Products

MS provides detailed information about electrolyte decomposition pathways by identifying specific decomposition products. The technique can distinguish between different reaction pathways and track how decomposition patterns change with temperature, state of charge, and other operating conditions.

Studies using MS have confirmed theoretical predictions about SEI formation mechanisms and have identified previously unknown decomposition pathways [33]. This information is crucial for developing improved electrolyte formulations and additives that minimize unwanted side reactions.

2.9.3 Tracking Thermal Runaway and Abusive Conditions

Real-time MS monitoring during thermal runaway provides unprecedented insights into the progression of failure events. The technique can track the evolution of different gas species as the TR progresses, providing information about the sequence of reactions and the conditions that lead to the most hazardous gas compositions.

This capability is particularly valuable for developing TR detection systems and for optimizing battery design to minimize gas generation during abuse conditions. Early detection of characteristic gas species can provide warning of impending TR before the most dangerous phases of the event occur.

2.9.4 Examples of Research Findings Using MS

Recent research using MS has provided new insights into battery degradation and failure mechanisms. Studies have quantified the rates of gas generation during different operating conditions and have identified correlations between gas evolution patterns and battery performance metrics [33, 34].

The technique has also been used to evaluate the effectiveness of various safety measures and design modifications. For example, MS can assess how different electrolyte additives affect gas generation patterns or how cell design modifications influence the composition of gases evolved during TR [35].

2.10 Conclusion with Respect to This Research

2.10.1 Relevance of Real-time Gas Analysis

Real-time gas analysis plays a crucial role in advancing the understanding of degradation and safety mechanisms in lithium-ion pouch cells over time. The ability to monitor gas evolution dynamically, rather than relying on static measurements, enables the detection of transient phenomena that are often missed in traditional test methods. This is particularly important in early-stage detection of hazardous conditions such as electrolyte decomposition, gas accumulation, or the onset of thermal runaway.

In this research, MS is employed as the analytical method due to its high sensitivity, broad chemical coverage, and time-resolved measurement capability. The integration of a gas-tight containment mechanism and an automated valve system allows for synchronized, trigger-based sampling, which makes the analysis both flexible and precise.

2.10.2 Justification for Focusing on MS

MS was selected over other gas analysis techniques for several reasons directly aligned with the goals of this project. MS allows for continuous or semi-continuous monitoring without delays. Its ability to detect a wide range of gas species, including those that are electrochemically reactive, short-lived, or present in low concentrations, makes it well-suited for detecting early-stage battery degradation.

Furthermore, the integration of MS with a custom-built hardware and software framework enables flexible control over when and how measurements are taken. Trigger-based sampling, based on parameters such as voltage, time, and cycle state, ensures that gas data can be accurately correlated with specific electrochemical events. This opens the door to deeper diagnostic insight into how and when critical reactions occur within the cell.

Overall, this research demonstrates that with proper integration of airtight design, inert materials, automation, and synchronized control logic, MS can be effectively applied to monitor and understand gas evolution in real-time. The developed methodology has the potential to serve as a foundation for future battery safety diagnostics and operando characterization platforms.

Chapter 3

Design and Production of the Test Setup

3.1 Introduction

This chapter presents the design and fabrication of a custom operando gas analysis setup intended for the characterization of gas evolution in lithium-ion pouch cells. The system enables real-time monitoring of gases released during electrochemical cycling and is tailored for integration with a mass spectrometer.

The development process involved the design of a modular, sealable test cell capable of housing standard-format pouch cells under controlled conditions. Mechanical, fluidic, and electrical subsystems were integrated to ensure compatibility with both laboratory infrastructure and safety protocols.

The following sections detail the functional requirements and design constraints, followed by an in-depth discussion of each subsystem and the associated manufacturing processes.

3.2 Design Requirements

The design of the test setup was subject to several essential functional and experimental requirements.

First, the system was required to be completely airtight once closed. Given that the setup is opened and closed in a dry room, but operated externally, the exclusion of ambient air during operation was critical. This requirement was imposed both for safety—due to the reactivity of lithium-containing materials with atmospheric moisture, and for the reliability of gas analysis, where even minor air ingress could compromise measurement accuracy.

Second, the system had to allow for the electrochemical cycling of the pouch cell from outside the sealed volume. This required uninterrupted electrical connectivity between the potentiostat and the cell terminals while maintaining a sealed internal environment.

Third, the internal arrangement of the pouch cell had to prevent the possibility of electrical short circuits. Proper electrical isolation and mechanical support were necessary to ensure safe operation under varying test conditions.

A further requirement was the minimization of dead volume throughout the gas path. Excessive internal volume would lead to dilution of the evolved gases by the carrier gas, reducing the sensitivity and resolution of the measurements. This concern was particularly relevant in light of previous setups where large dead volumes were shown to negatively impact the detection of gas species.

In addition to these primary requirements, the design was also expected to accommodate:

- Controlled flow of inert carrier gas through multiple selectable paths.
- Compatibility with real-time gas analysis instrumentation.
- Repeated mechanical assembly and disassembly without loss of sealing performance.
- Operability within the spatial and procedural constraints of a dry room environment.

3.2.1 Constraints

The design should be made within the given constrains.

3.3 Overview of the Setup

Figure 3.1 shows the complete operando gas analysis setup as assembled during the experimental phase. The system integrates all necessary components to enable airtight, safe, and controllable cycling of lithium-ion pouch cells under operando conditions. It also allows evolved gases to be analyzed in real time. The key elements are briefly introduced below.

- **Cell:** The central component of the setup, located in the middle of the gas circuit. It contains a lithium-ion pouch cell enclosed in a stainless steel housing that maintains airtight sealing during operation.
- **Circuit board:** Positioned at the top left of the image, the control board is responsible for sending digital signals to activate the solenoid valves. It also handles logic and timing, and provides power distribution to connected components.
- **Solenoid valves:** Distributed throughout the gas circuit, the solenoid valves manage the flow routing of the argon carrier gas. They allow switching between measurement lines and flushing paths. These are controlled programmatically via the circuit board.
- **Tubing:** The gas path is constructed using a combination of stainless steel and PTFE tubing. The layout is optimized to minimize dead volume and reduce contamination risk. It is also designed to ensure compatibility with pressurized argon operation.
- **Filter:** Placed downstream of the cell outlet, the filter prevents solid or liquid particles such as condensed electrolyte from reaching the mass spectrometer. This protects the detector from damage or contamination.
- **Argon inlet:** Located upstream in the flow circuit, the argon inlet provides a continuous supply of inert carrier gas to the system. The pressure and flow are regulated externally.

- **Fitok connection pieces:** Stainless steel Fitok connectors are used throughout the setup to ensure leak-free and high-pressure-compatible fittings between valves, tubing, and components. Their modularity allows quick disconnection and reconfiguration. This does not compromise system integrity.

Further sections elaborate on each subsystem and describe their roles in ensuring airtight operation, measurement accuracy, and experimental repeatability.

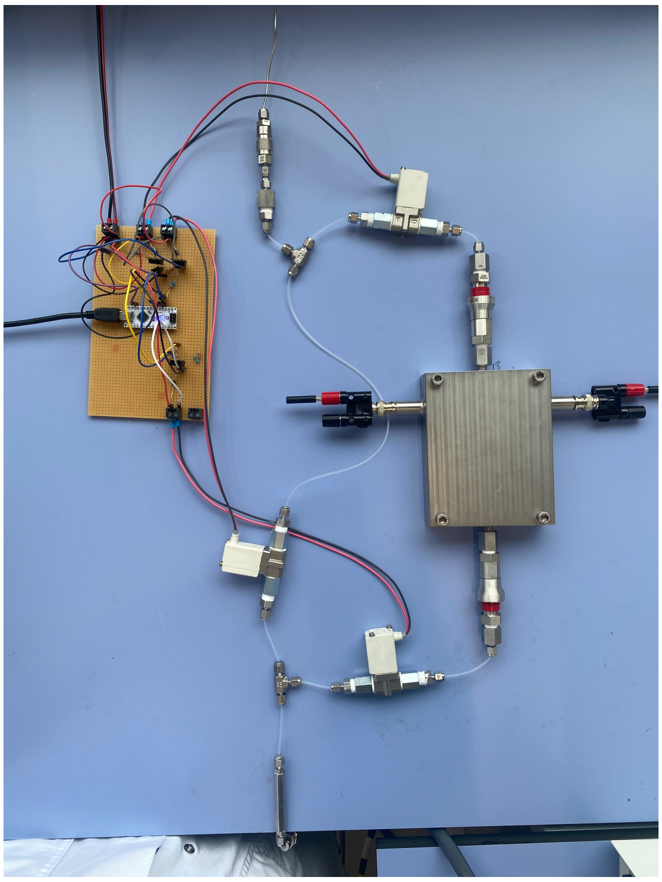


Figure 3.1: Top-view image of the complete test setup with labeled components.

3.4 Design of the Cell

The cell assembly forms the central component of the test setup. It is responsible for housing the pouch cell in a manner that guarantees gas-tight sealing, electrical accessibility, and mechanical stability throughout the test cycle. The design allows the operator to insert and remove the pouch cell in a controlled dry room environment, after which the system is closed and transferred to the measurement environment for operando gas analysis.

Figures 3.2 and 3.3 provide an overview of the mechanical structure of the cell. In the open configuration (Figure 3.2), the internal cavity and sealing groove are visible, along with the electrical contact components. In the closed configuration (Figure 3.3), the enclosure is fully sealed, with all valves and feedthroughs connected.

The remainder of this section discusses the structural elements, gas flow interfaces, sealing methods, and electrical integration that collectively define the functionality of the cell housing.

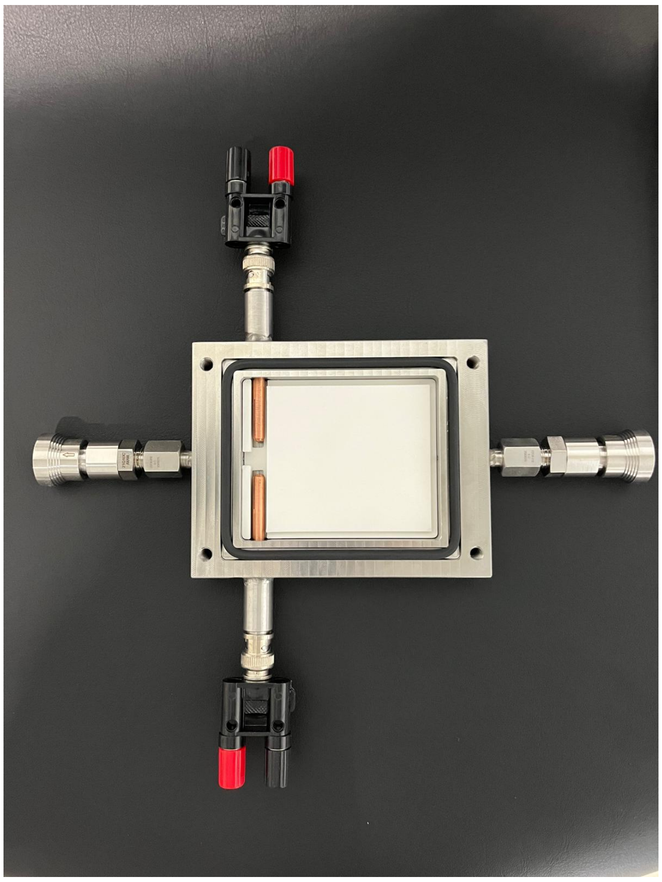


Figure 3.2: Top-view of the cell in opened configuration.

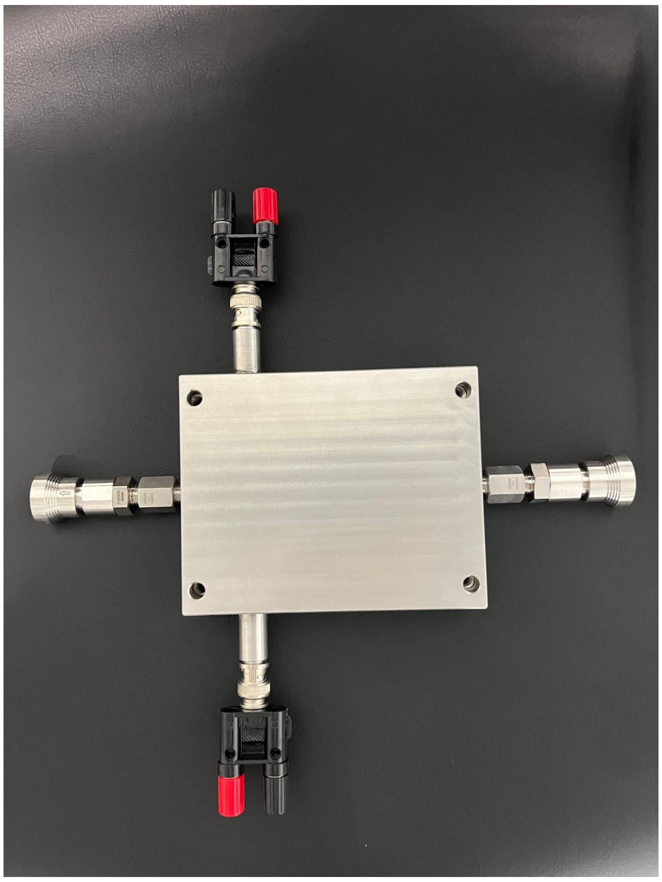


Figure 3.3: Top-view of the cell in closed configuration.

3.4.1 Cell Housing

The housing defines the core structure in which the pouch cell is placed during operando gas analysis. It is designed to enclose the cell tightly in order to contain all gases released during electrochemical cycling. A reliable and hermetic seal is essential to maintain the system's integrity and prevent contamination or leakage.

Material and Dimensions

The housing is machined from stainless steel (316L), selected for its high chemical resistance to both the electrolyte and the gases potentially released during operation. The material also provides mechanical stability and corrosion resistance, while being suitable for precision machining processes.

The dimensions of the enclosure were chosen to closely match the dimensions of the pouch cell, while minimizing internal volume to reduce dead space in the system. The external dimensions of the housing are 125 × 100 mm, and the internal chamber measures 84 × 69 mm.

Bottom Plate

The pouch cell is placed into a machined cavity in the bottom plate. This cavity will later be filled with an insulating insert to prevent short-circuiting between the electrodes and the housing. In the initial design stage, a gas distribution grid was integrated into the bottom plate to guide argon and evolved gases through the cell. However, this feature was relocated to the insulating insert to improve modularity and simplify manufacturing.

Additionally, a groove is machined into the bottom plate to accommodate an O-ring. This ensures proper sealing when the top plate is mounted. Mounting is achieved via M6 bolts, placed at the corners of the housing.

Channels are also integrated into the bottom plate to allow connection of the gas inlet and outlet tubing. These are positioned to align with the gas channels in the insulating insert. Holes for electrical feedthroughs are incorporated as well. Figure 3.4 shows the CAD model of the bottom plate including the machined features.

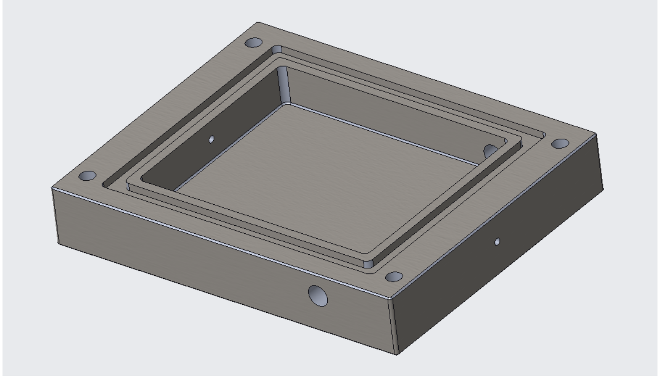


Figure 3.4: CAD model of the bottom plate.

Upper Plate

The upper plate shares the same external dimensions as the bottom plate. Internally, a shallow cavity is machined to define the upper boundary of the cell chamber. This cavity serves as a locating guide for the rubber sealing element, ensuring centered placement and uniform compression during assembly. Figure 3.5 shows the CAD model of the upper plate with relevant features.

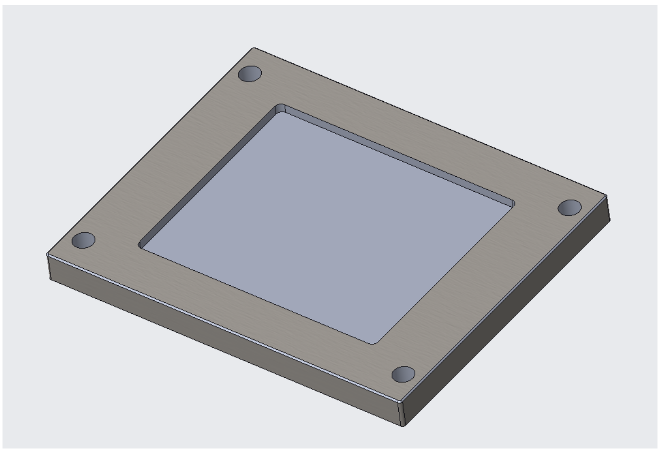


Figure 3.5: CAD model of the upper plate.

Production of the Cell Housing

The two housing components were produced using high-precision CNC milling. Figure 3.6 shows the machined components after initial manufacturing. Notably, the cavity in the bottom plate is slightly smaller than in the upper plate. This design choice was made to account for post-weld distortion. At the time of rough machining, the welding components such as the electrical feedthrough and gas connectors had not yet been delivered. Because welding introduces local shrinkage, leaving extra material in the cavity allowed for final finishing after welding, thereby avoiding misalignment that could compromise sealing.

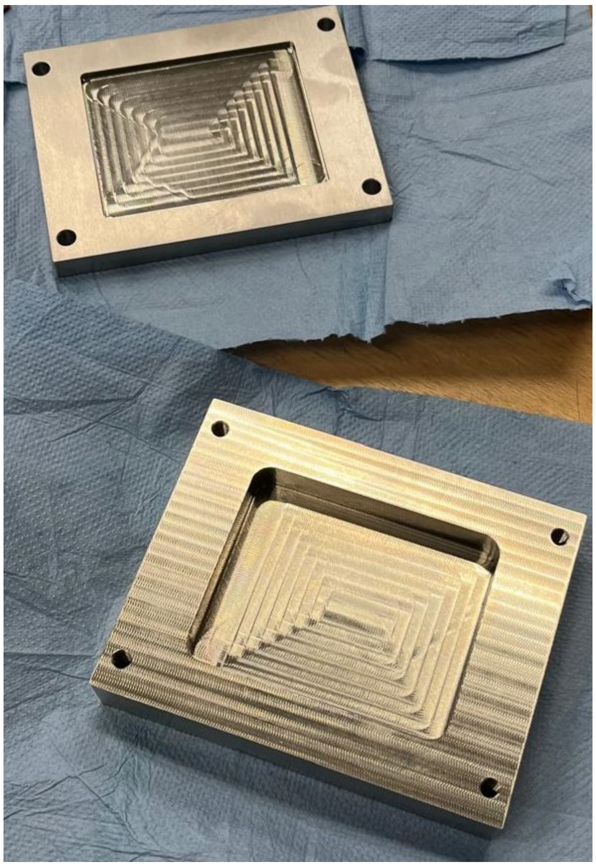


Figure 3.6: Milled top and bottom plates of the cell housing.

3.4.2 Gas Flow Management

Efficient gas flow control is essential for reliable operando measurements. The system must ensure that the carrier gas (argon) is routed correctly and that cross-contamination between measurement channels is avoided. This section describes the components and flow configuration used in the system, supported by the schematic in Figure 3.7.

Argon Gas Supply

Argon is used as an inert carrier gas. The laboratory argon supply was pre-installed and connected to a digital flowmeter and controller. This unit regulates the gas flow toward the solenoid valves.

Gas Flow Configuration

Figure 3.7 illustrates the gas flow path for a single line. In total, the system comprises three distinct lines:

- **Bypass line:** Provides a continuous reference flow of pure argon to the mass spectrometer. This ensures a stable baseline and compensates for any delays in valve switching or fluctuations during sampling.
- **Pouch cell line:** Connects to the custom-built cell. This path is activated during operando analysis of pouch cells.
- **Coin cell line:** Routes gas through the legacy setup for coin cell testing. Integrating this setup into the new system improves usability and reduces dead volume by using shorter

tubing.

Solenoid valves are used to switch between these lines automatically. The logic controlling these valves is discussed in a separate chapter on automation.

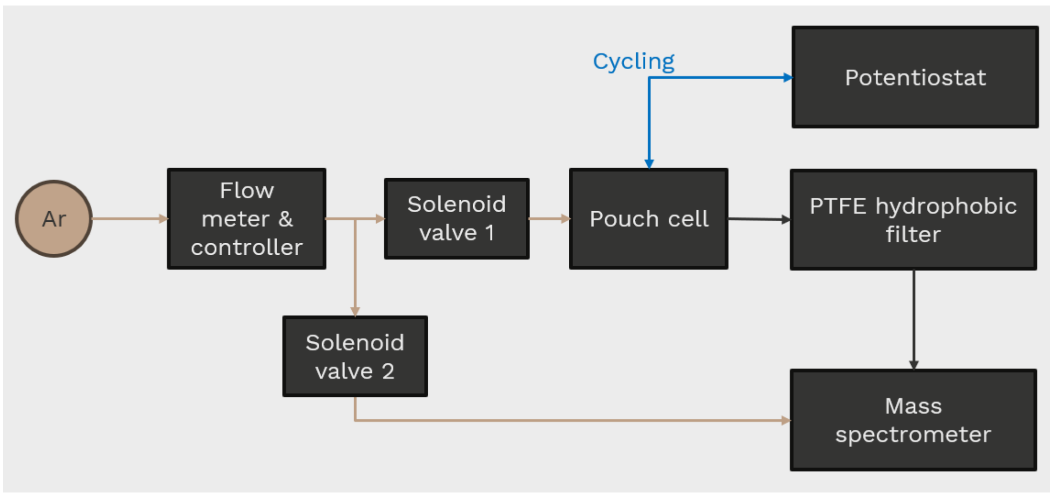


Figure 3.7: Schematic of the gas flow configuration for a single line.

Quick Connectors

Quick connectors sourced from Fitok are used to decouple the cell from the gas system. These connectors are normally closed, which ensures that no gas escapes when the cell is disconnected. This design facilitates rapid and safe exchange of the cell without depressurizing or venting the rest of the system.

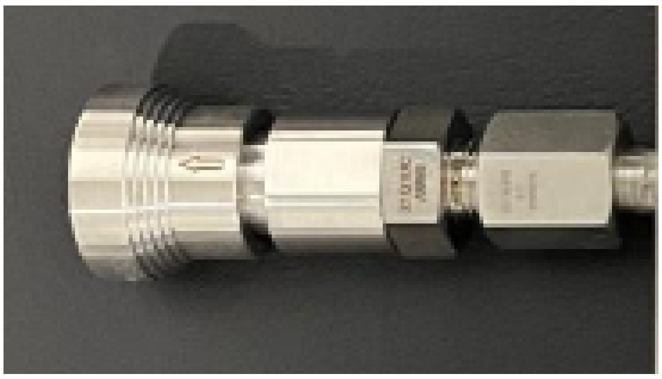


Figure 3.8: Female quick connector.

Figure 3.9: Male quick connector.

3D Printed Bottom Layer

Gas is introduced into the cell cavity through a channel embedded in the insulating bottom plate (Figure 3.10). When the system is closed, the pouch is pressed onto this plate by a sealing element, effectively isolating the internal volume. This ensures that gas can only travel from the inlet to the outlet via the intended lateral path. An outlet hole collects both carrier and evolved gases and directs them to the filter and mass spectrometer.

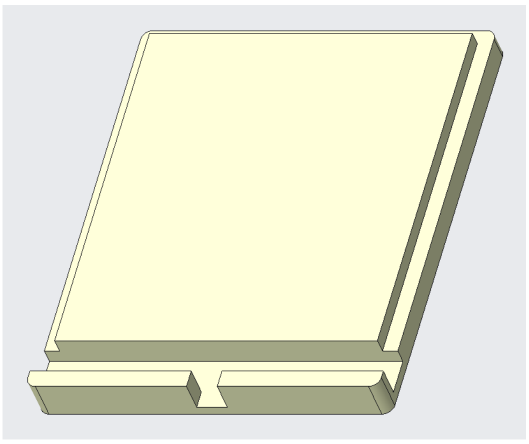


Figure 3.10: Insulating bottom plate with integrated gas flow channel.

3.4.3 Electrical Feedthrough

To enable electrochemical cycling of the pouch cell while preserving the gas-tightness of the sealed enclosure, dedicated electrical feedthroughs are integrated into the housing. These feedthroughs allow electrical signals to pass from the external environment into the internal cell chamber without permitting gas exchange. This is essential for maintaining measurement integrity during operando gas analysis.

The feedthroughs are welded into the stainless steel housing. This method ensures a permanent hermetic seal around the conductors, preventing leaks that could occur with threaded or compression-based fittings. The mechanical integration was designed to remain robust under differential pressure and repeated handling.

Externally, the feedthroughs are connected to a potentiostat, which controls the charge and discharge cycles of the pouch cell. The potentiostat also records voltage and current responses during the measurement. A figure of the potentiostat and its control interface will be added later in this document.

Short-Circuit Protection

Short-circuit prevention was a primary concern in the design of the internal electrical contacts. The pouch cell's positive and negative terminals, typically in the form of aluminum and copper tabs, must be isolated from each other and from the stainless steel housing. Improper alignment or insufficient insulation would present a risk of electrical shorting and cell failure. A custom insulating insert was developed to position and support the tabs securely within the housing.

Coaxial Connectors

The external electrical interface uses coaxial connectors to ensure robust signal transmission and electromagnetic shielding. These connectors maintain low resistance and physical integrity over repeated connection cycles. The coaxial geometry also contributes to the system's overall gas-tightness.

Tab Interface and Internal Connection Mechanism

Internally, a custom solution was developed to interface the cell tabs with the feedthroughs. Each welded feedthrough terminates inside the chamber with a copper rod. These rods feature a longitudinal slit at the rear side, into which the aluminum or copper tab of the pouch cell is inserted. Once inserted, the tab is mechanically folded and pressed against the copper rod to ensure consistent electrical contact during cycling. This connection method eliminates the need for soldering or thermal joining near the active cell.

This configuration allows for fast, tool-free installation while preserving full gas sealing, as the feedthroughs are fixed and hermetically welded into place. The physical layout of this connection is shown in Figures 3.11 and 3.12, which illustrate the internal arrangement both with and without the pouch cell inserted.

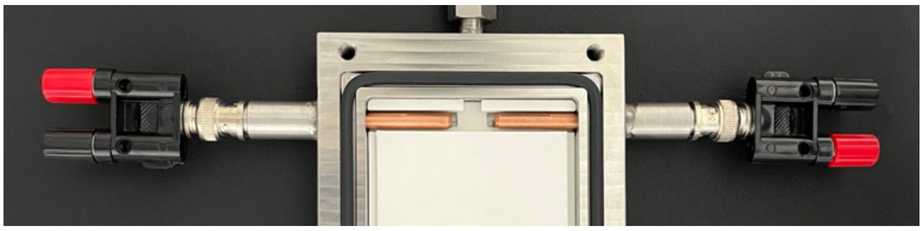


Figure 3.11: Internal view of the copper feedthrough connection system.

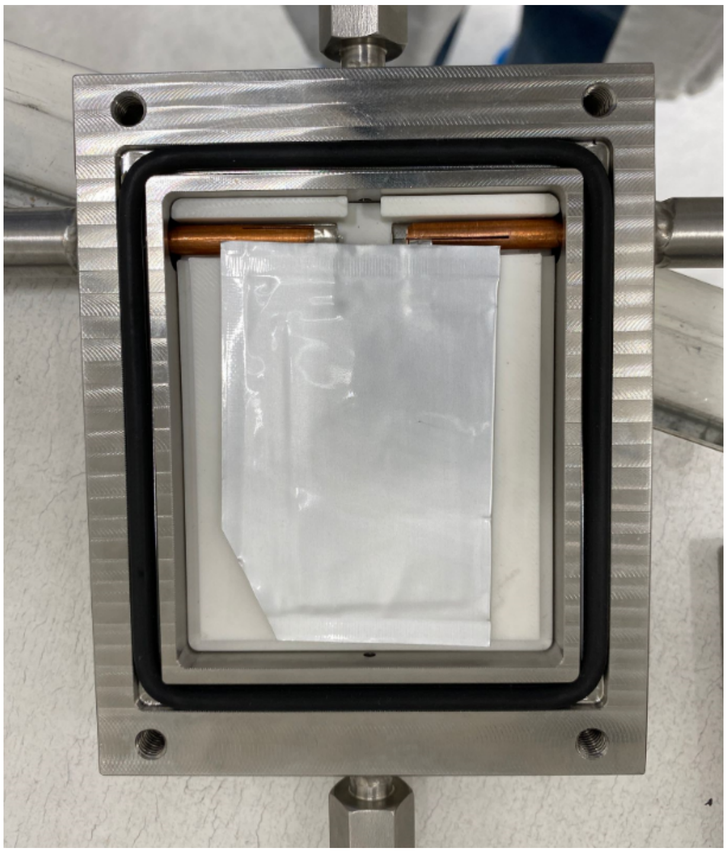


Figure 3.12: Pouch cell inserted and connected to the internal feedthrough rods.

3.4.4 Sealing Mechanism

To maintain airtight conditions during operando testing, a two-stage sealing concept was implemented. This design relies on both a primary O-ring and a secondary EPDM cellular rubber layer to ensure gas isolation between the upper and lower plates of the cell housing.

O-Ring Selection

The primary seal is provided by a circular O-ring placed in a machined groove along the inner perimeter of the bottom plate. The groove dimensions are 75 mm $\times$ 90 mm in width, with a depth of 2.5 mm and a channel width of 5.0 mm. The selected O-ring has an inner diameter of 107.5 mm and a cross-section (snoerdikte) of 3.53 mm. This O-ring geometry results in a radial compression of approximately 29%, which is within the recommended range for static sealing applications.

Based on the O-ring material properties (Viton, modulus $\approx$ 7.5 MPa) and a compression factor of 1.2 for the selected groove geometry, the resulting sealing pressure is estimated at approximately 2.6 MPa. This value is sufficient to ensure gas-tight operation under low-pressure conditions and prevents ingress or egress of argon or any evolved gases. The O-ring is compressed vertically between the top and bottom plates.

EPDM Cellular Rubber

A secondary sealing interface is achieved using a layer of EPDM (ethylene propylene diene monomer) cellular rubber placed on the underside of the top plate. This material was selected due to its relatively low Shore hardness, which makes it more easily deformable than the primary O-ring. This hierarchy of compressibility ensures that the O-ring absorbs the majority of the clamping force and forms the main seal, while the EPDM layer provides supplementary sealing around the pouch and electrical contacts.

The EPDM used features a closed-cell structure. This design prevents gas diffusion into the material, which would otherwise trap reaction products and hinder accurate detection by the mass spectrometer. If an open-cell foam were used, evolved gases could be absorbed into the microstructure, effectively acting as a gas-phase capacitor and introducing long-term release effects that degrade measurement accuracy.

Clamping Mechanism Selection

To compress the sealing elements, the upper and lower plates are fastened using four M6 bolts. Initial concepts considered using an external mechanical clamp, similar to the one employed in the coin cell configuration. While such a clamp is suitable for compact, lightweight setups, it was deemed impractical for the current design due to its larger dimensions and the need for repeated manipulation within a dry room environment.

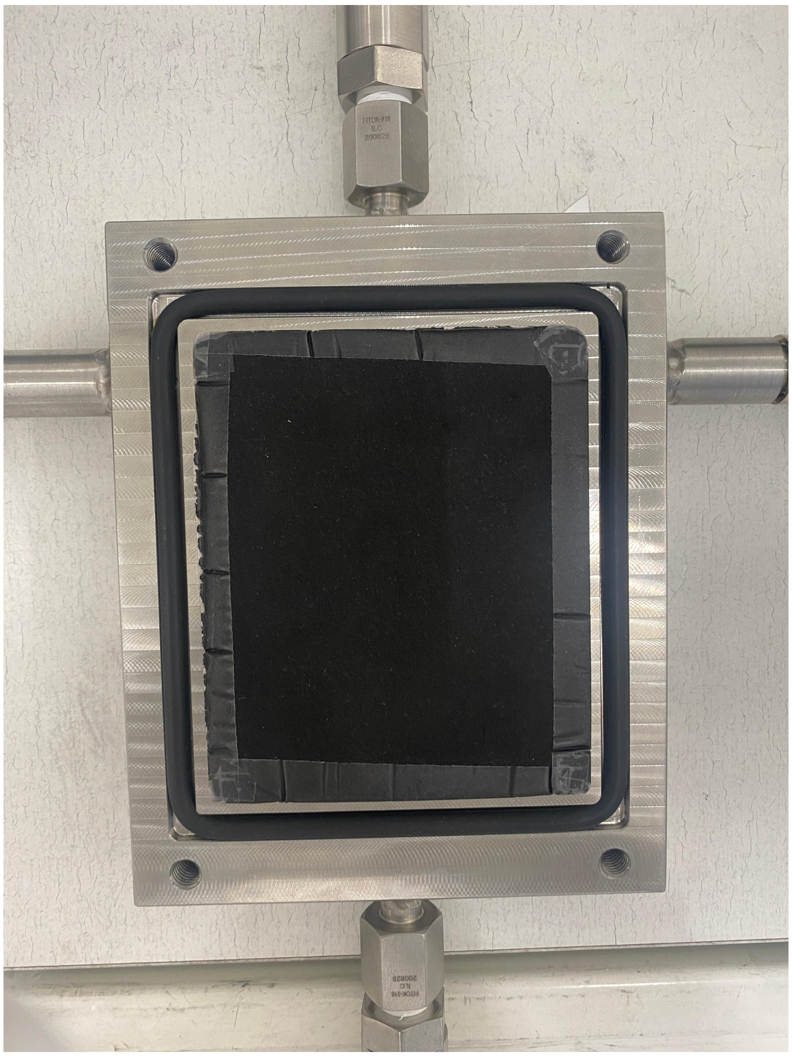


Figure 3.13: Detail view of the O-ring and EPDM rubber interface in the assembled cell.

The use of bolts offers several advantages. The fasteners are embedded directly in the housing, simplifying alignment during closure. They also provide reproducible and symmetric compression, which is critical for maintaining sealing pressure. In addition, the absence of an external frame improves the setup's portability and reduces its footprint during manipulation in the dry room.

Chapter 4

Automation

4.1 Introduction

To enable operando, precise and repeatable gas analysis of pouch cell batteries, the test setup must operate autonomously after starting. Manual operation is not only time-consuming, but it does not allow operando measurements as well. In addition, it introduces inconsistencies due to human error and exposes the operator to the electrolyte, which poses a chemical safety risk. Therefore, automation was integrated as a core part of the system design, ensuring that measurements can be performed autonomously, reliably, and with minimal supervision or intervention.

This chapter presents the full automation strategy developed for the gas analysis setup. Section 4.2 focuses on the electronics, including the circuit design, the microcontroller platform, and how key components such as solenoid valves, the flow meter, and the potentiostat are connected and controlled. Section 4.3 describes the software implementation, including the control logic, real-time data acquisition, user interface design, and communication with external devices. Finally, data logging and software testing procedures are also discussed, demonstrating the system's capability for robust long-term operation.

4.2 Electronics

This section outlines the electronic subsystem that enables automated control of the gas analysis setup. It describes the central role of the microcontroller in managing signal flow between sensors and actuators, including the solenoid valves, flow meter, and potentiostat. Additionally, the integration of protection components such as flyback diodes and the overall circuit design are discussed to ensure reliable and safe operation during long-term measurements.

4.2.1 Circuit Overview

To enable automated switching of the solenoid valves, each valve is controlled via an external power circuit based on logic-level N-channel MOSFETs. This approach ensures safe and efficient control of 24 V components using the 5 V logic signals of the Arduino Nano. Figure 4.1 shows the complete circuit used to control five solenoid valves.

Each control channel consists of a digital output pin from the Arduino Nano (D2, D3, D4, D5,

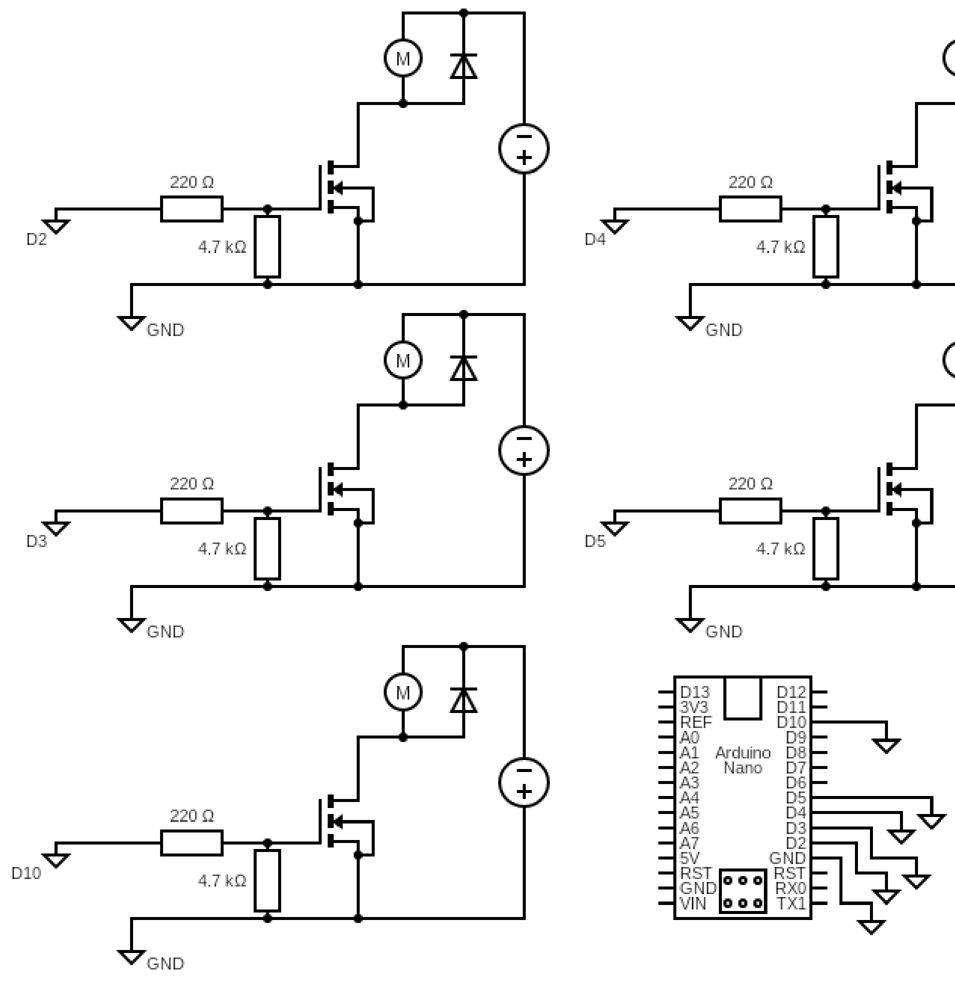


Figure 4.1: Circuit overview for controlling five 24 V solenoid valves using an Arduino Nano and logic-level N-MOSFETs

and D10), connected to the gate of a MOSFET through a 220 Ω resistor. This resistor limits gate charging current and protects the microcontroller pin. A 4.7 kΩ pull-down resistor is used on each gate to ensure the MOSFET remains off when the pin is not driven.

The solenoid valve is connected between the 24 V supply and the drain of the MOSFET. When the Arduino outputs a HIGH signal, the MOSFET turns on and allows current to flow through the valve, activating it. To protect the MOSFET against inductive voltage spikes generated by the solenoid, a flyback diode is placed in parallel with each valve coil, oriented to block current during normal operation and conduct during switch-off events.

All MOSFET sources are tied to ground, which is shared with the Arduino and the power supply to ensure proper signal reference and switching operation. This modular configuration ensures that each valve can be individually controlled while maintaining electrical isolation between control and load sections.

4.2.2 Microcontroller

The microcontroller used for this application is an Arduino Nano based on the ATmega328P (see Figure 4.2). This compact and cost-effective development board was selected due to its small footprint, sufficient number of digital I/O pins, and compatibility with the Arduino development environment. The Nano serves as the central control unit, generating logic signals to switch the solenoid valves and handling communication with external devices. It operates at 5 V and is powered via USB to the pc that contains the software application as well for interfacing, better explained later. Its ease of programming, wide community support, and integration with readily available libraries made it a robust choice for the automation system. More technical details can be found on the supplier's website [36].

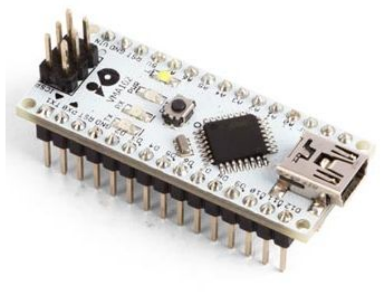


Figure 4.2: Arduino Nano microcontroller based on the ATmega328P

4.2.3 Solenoid Valves

The gas sampling system uses normally closed (NC) solenoid valves to selectively open or isolate flow paths during automated measurement sequences. An image of the solenoid valve is shown in Figure 4.3. Each valve operates at 24 V DC and is actuated electrically via the control circuit described in Section 4.2.1. When activated, the valve opens to allow gas flow and requires 0.125 mA or 3 W. When de-energized, it returns to the closed state thanks to a spring, ensuring safe default isolation of the gas lines.

The selected model is a miniature solenoid valve from RS PRO (product no. 873-2639), chosen for its compact form factor, fast switching speed, and compatibility with inert and corrosive gases. The valve is rated for 0–8 bar pressure and includes a 3/2 configuration, allowing it to function in either open/close or directional switching modes depending on the application. Further, the material of this valve is SS, which makes it well suited for use in automated gas handling in electrochemical experiments.

Full technical specifications are available on the supplier's website [37].

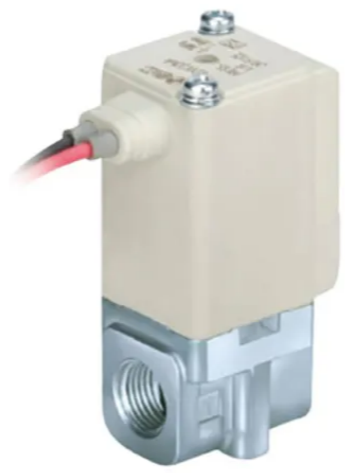
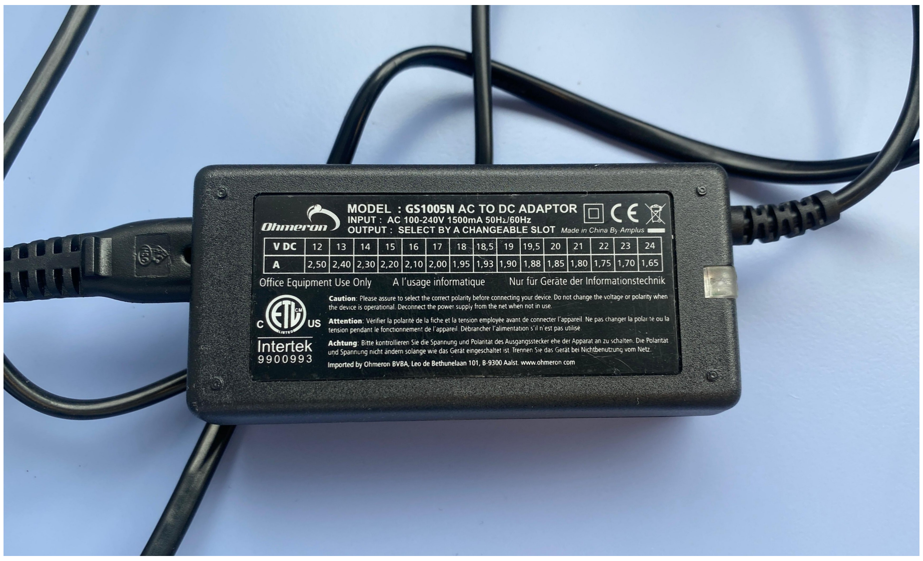


Figure 4.3: 24 V normally closed miniature solenoid valve (RS PRO 873-2639).

24 V Power Supply

The solenoid valves in the automation system are powered by an external 24 V DC power supply (see Figure 4.4) rated at up to 2 A. This provides sufficient current to activate multiple valves simultaneously, while maintaining voltage stability and ensuring reliable operation under varying load conditions.

Using a dedicated power supply separates the power domain of the actuators from the Arduino Nano's logic circuitry. This isolation helps reduce electrical noise, prevents voltage sag, and protects the microcontroller from potential current surges.


Ohmeron

MODEL : GS1005N AC TO DC ADAPTOR

INPUT : AC 100-240V 1500mA 50Hz/60Hz

OUTPUT : SELECT BY A CHANGEABLE SLOT Made in China By Amplus

V DC	12	13	14	15	16	17	18	18,5	19	19,5	20	21	22	23	24
A	2,50	2,40	2,30	2,20	2,10	2,00	1,95	1,93	1,90	1,88	1,85	1,80	1,75	1,70	1,65

Office Equipment Use Only A l'usage informatique Nur für Geräte der Informationstechnik

Caution: Please assure to select the correct polarity before connecting your device. Do not change the voltage or polarity when the device is operational. Deconnect the power supply from the net when not in use.

Attention: Vérifier la polarité de la fiche et la tension employée avant de connecter l'appareil. Ne pas changer la polarité ou la tension pendant le fonctionnement de l'appareil. Débrancher l'alimentation s'il n'est pas utilisé.

Achtung: Bitte kontrollieren Sie die Spannung und Polarität des Ausgangsstecker ehe der Apparat an zu schalten. Die Polarität und Spannung nicht ändern solange wie das Gerät eingeschaltet ist. Trennen Sie das Gerät bei Nichtbenutzung vom Netz.

C US Intertek 9900993

Imported by Ohmeron BVBA, Leo de Bethunelaan 101, B-9300 Aalst. www.ohmeron.com

Figure 4.4: 24 V DC power supply with a maximum output current of 2 A.

MOSFET Selection

For switching the solenoid valves, the IRFZ44N N-channel MOSFET was selected (see Figure 4.5). This transistor is widely used in low- to medium-power switching applications due to its high current handling capability (up to 49 A) and low drain-source on-resistance ($R_{\mathrm{DS}(on)} \approx 0.032\ \Omega$ at $V_{\mathrm{GS}} = 10\,\mathrm{V}$). Although it is not a logic-level MOSFET, the IRFZ44N can be sufficiently driven by the 5 V gate signals of the Arduino Nano in this application, due to the relatively low switching frequency and moderate current demand of the solenoid valves. More technical specifications are available on the shop's website [38].

Figure 4.5: 24 V DC power supply with a maximum output current of 2 A.

The MOSFETs are configured as low-side switches, with their sources connected to ground and the drains connected to the negative terminal of the solenoid valves. This configuration allows each valve to be activated by applying a `HIGH` signal from the Arduino to the corresponding gate. A gate resistor (220 Ω) is added to limit inrush current, and a pull-down resistor (4.7 kΩ) ensures the MOSFET remains off when the gate is floating. The robustness and wide availability of the IRFZ44N made it a suitable and cost-effective choice for this system.

Flyback Diode Selection

To protect the MOSFETs from high-voltage transients generated when switching off inductive loads, a flyback diode is placed in parallel with each solenoid valve. For this purpose, the 1N4007 general-purpose rectifier diode was selected. It is capable of withstanding reverse voltages up to 1000 V and conducting continuous currents up to 1 A, which is well within the requirements of the solenoids used in this setup.

When the solenoid is de-energized, the sudden collapse of the magnetic field induces a high reverse voltage across the coil. The flyback diode provides a safe discharge path for this current by conducting it back through the coil, thereby clamping the voltage spike and preventing damage to the MOSFET. The 1N4007's ruggedness and wide availability make it a reliable and economical choice for this type of inductive load protection.

4.2.4 Flow Meter Connection

The system uses a Brooks Instruments mass flow controller (MFC) in combination with a microprocessor-based read-out unit (model 0152), shown in Figure 4.6 to control and monitor gas flow rates during measurement sequences.

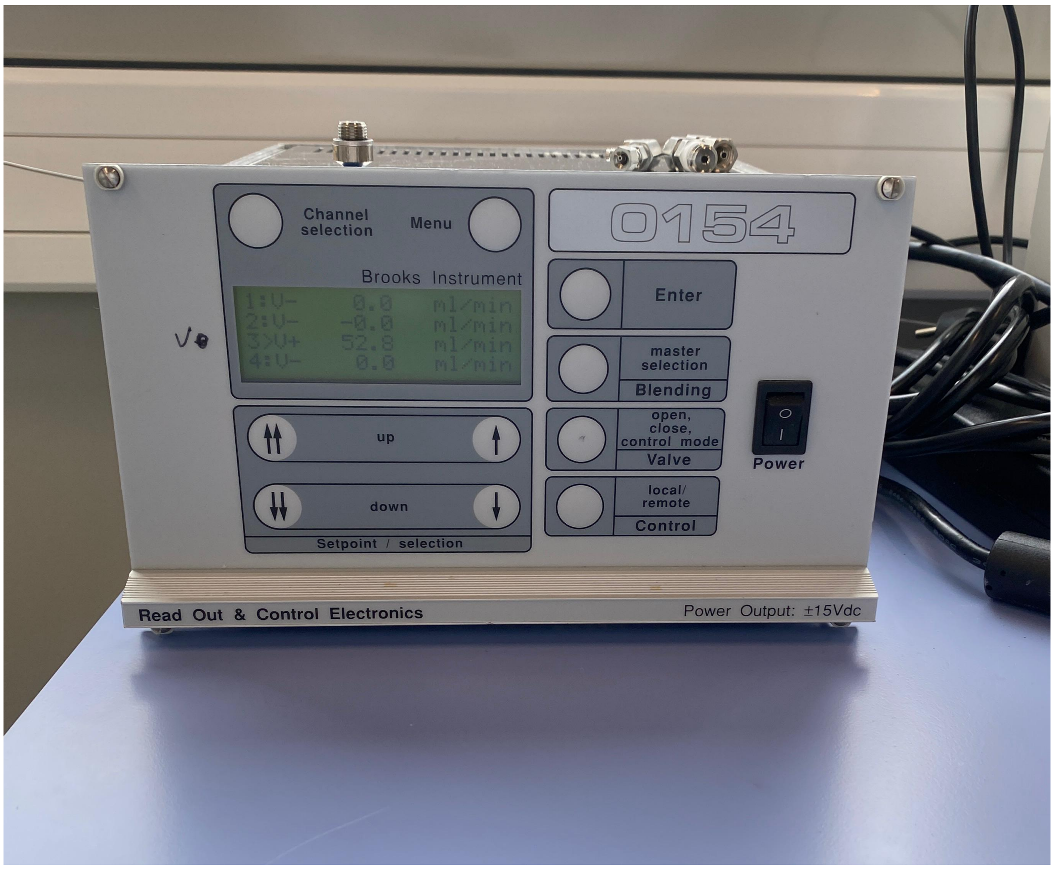


Figure 4.6: Brooks 0152 mass flow controller

An example schematic of the connection is shown in Figure 4.7. The Brooks 0152 unit also supports multiple I/O ranges (0–10 V, 0–20 mA, 4–20 mA), but in this case, only the default 0–5 V mode was used. The user's manual on the official website of Brooks Instruments explains all the possible working modes of the tool and how to communicate with it [39].

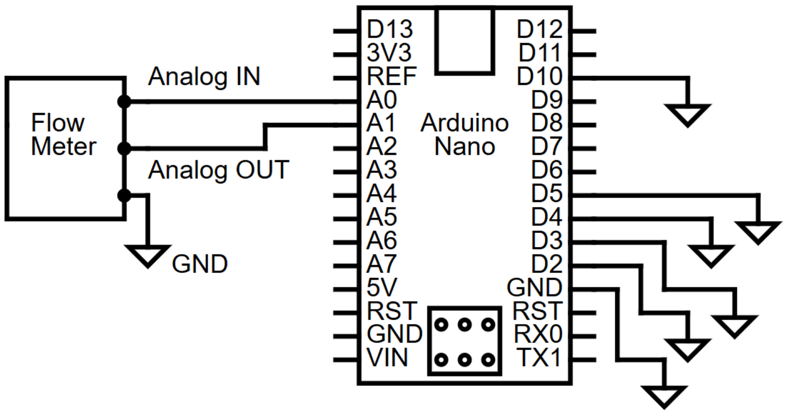


Figure 4.7: Wiring configuration for connecting the Arduino Nano to channel 3 of the Brooks 0154 read-out unit.

Each channel of the read-out unit supports analog setpoint input and flow feedback output, with voltage ranges typically scaled between 0–5 V to represent 0–100% of full-scale flow.

In this setup, channel 3 of the Brooks unit is used to interface with the Arduino Nano for both control and monitoring. Three connections were made:

- The ground reference of the Brooks unit (pin 23: `Setpoint ground channel 3`) was connected to the Arduino's GND pin.
- The analog setpoint input (pin 11: `Input Remote setpoint Volt channel 3`) was connected to an Arduino PWM-capable digital pin through a low-pass RC filter to provide a stable voltage control signal.
- The analog flow signal output (pin 5: `Output Flow Volt channel 3`) was connected to an analog input pin on the Arduino for real-time monitoring.

This configuration allows the Arduino to set a flow rate via analog voltage and simultaneously read back the actual flow as an analog voltage signal, scaled proportionally to the maximum calibrated range of the instrument. Since both the input and output operate in the 0–5 V domain, direct voltage compatibility with the 5 V Arduino system is ensured, without the need for level shifting.

4.3 Software

The automation system is controlled by a custom-developed Python application. This software manages all key functionalities, including valve switching, real-time data acquisition, user interaction, and experiment logging. It is designed to run on the same laptop used to operate the potentiostat, enabling seamless integration without additional hardware requirements.

The Python code is modularly structured into independent components, each responsible for specific aspects of system control. The communication with the Arduino is handled via a serial USB connection, while the flow meter data is read and controlled through analog voltage signals managed by the Arduino firmware. Trigger-based measurement logic is implemented to enable sampling actions based on voltage thresholds, time intervals, or cycling and rest states.

The software architecture was developed with flexibility and robustness in mind, allowing multiple measurement lines to operate under different modes—manual or automatic—each with a configurable set of parameters. A graphical user interface (GUI) built with the customtkinter library provides intuitive control and live visual feedback during operation.

In the following subsections the control logic, communication protocols, data acquisition approach, user interface, and data storage strategy are explained in detail.

4.3.1 Overview of the Control Logic

The core control logic of the automation system is implemented in Python and structured around two operational modes: automatic and manual. Each mode is managed in a dedicated module, with all underlying logic isolated from the graphical interface to ensure modularity and maintainability.

In `automaticLOGIC.py` (see Appendix A), the automatic mode is defined by a sequential flow of actions that mimic a full experimental cycle. The flow chart in Figure 4.8 shows the events order.

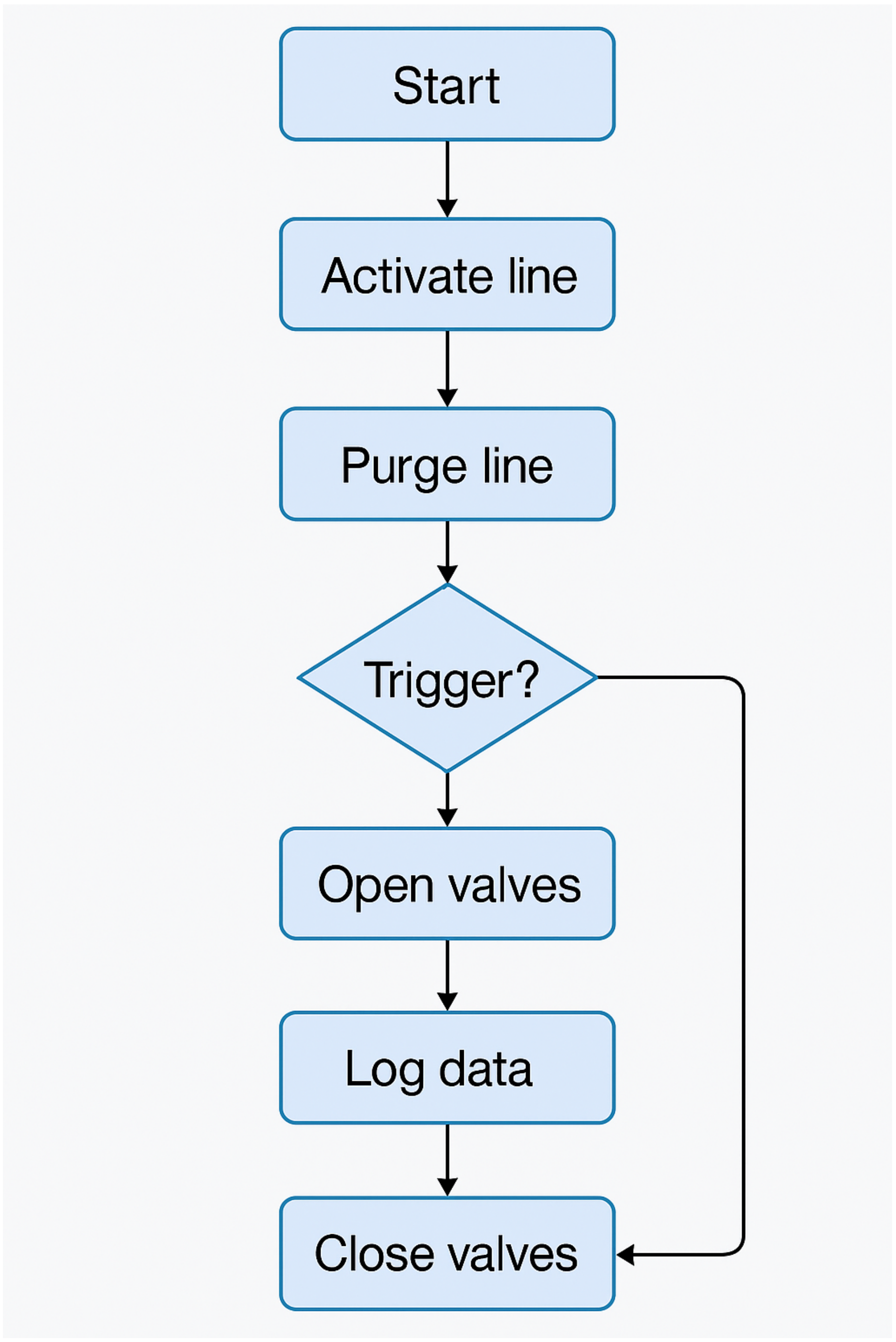


Figure 4.8: Flowchart of the automatic measurement logic implemented in `automaticLOGIC.py`.

Upon activating a sample line, an initial purge is executed by opening the two valves for the selected line for a given duration. After purging, the line valves shut down and the dedicated bypass valve (V0) opens to direct the argon gas flow directly towards the measurement device. This argon level will be used as reference level during the whole measurement.

The measurement process is event-driven and based on a flexible triggering system. Currently, five types of triggers are supported:

- **Time trigger:** activates a measurement periodically after a set interval.
- **Voltage trigger:** activates a measurement when a defined voltage threshold is reached. It contains a min. and max. limit as well to avoid detecting harsh peaks given by noise in

the potentiostat data.

- **Half cycle trigger:** activates when a half-cycle in the battery cycling is detected. This means that a measurement will happen each time the battery is fully charged or discharged.
- **Cycle trigger:** activates when a full cycle in the battery cycling is detected. The operator can choose to do a first measurements after X number of cycles and then a measurement after Y number of cycles. This is important because the most gases are produced during the first cycles. After the first cycles, the gas development drops significantly.
- **Rest trigger:** activates when a rest period in the battery cycling is detected.

Triggers can be combined, allowing for multi-condition events to initiate data logging. Once a trigger is active, the system opens the appropriate valves, logs the timestamp and contextual data, and then closes the valves after a user-defined sampling time.

Safety checks ensure that only one line is active at any given moment, preventing pressure drops or flow conflicts. All actions are logged and visualised in real-time within the user interface. The logic also includes safeguards to abort sampling if required parameters are missing or if conflicting operations are attempted.

Manual mode, implemented in `manualLOGIC.py` (see Appendix A), allows the user to directly toggle valve states and control flow without automated sequences. This is useful for calibration, maintenance, or exploratory measurements.

The modular structure allows future expansion of trigger types or integration of external control signals without major restructuring of the core logic. Furthermore the software is already implemented to control up to four lines. At the moment, only two lines are built as discussed before, one for the pouch cell measurements and one for the coin cell measurements.

4.3.2 Communication with External Devices

The automation software interfaces with two external devices to enable synchronized and autonomous sampling: the potentiostat and the mass spectrometer. These devices are essential for, respectively, detecting electrochemical state changes that trigger sampling events, and for analyzing the gaseous species released from the pouch cell.

Potentiostat – Real-Time Trigger Input The potentiostat, operated via EC-Lab software, performs the electrochemical cycling of the battery. To align gas sampling with specific states of charge or transitions, the automation system relies on real-time cycling data. This data is streamed continuously from EC-Lab using the “Online Text Export” function, which writes voltage, current, time, and state parameters to a text file in real time.

A dedicated Python module monitors this file and parses it to extract the relevant parameters. Trigger logic modules evaluate the live data to determine whether sampling conditions are met. This allows the system to react dynamically to voltage thresholds, cycle completions, or rest periods without requiring direct communication with the potentiostat hardware.

Mass Spectrometer – Passive Data Acquisition The mass spectrometer is responsible for identifying and quantifying gaseous species in the sample. It is not directly controlled by the automation software. Instead, it passively collects and analyzes gas whenever a sampling valve is opened and flow is directed to its inlet. The timing and duration of these sampling events are logged by the automation software and synchronized post hoc with the spectrometer data to correlate observed gas peaks with electrochemical events.

This indirect coordination ensures that the mass spectrometer operates independently of the control system, while still providing time-aligned analytical data. All valve actions and trigger events are timestamped, making retrospective alignment with the spectrometer output straightforward.

4.3.3 User Interface Design

The graphical user interface (GUI) was developed using the `customtkinter` library, a modern extension of Python's `tkinter` package that enables the creation of visually consistent and themable interfaces. The GUI is designed to support three primary operational views: *Automatic*, *Manual*, and *Visualisation*. Each mode is accessible via a tab navigation bar at the top of the window.

Automatic Tab The Automatic tab (Figure 4.9) allows users to configure sampling parameters per measurement line. Each line can be activated via a toggle switch, after which the user is required to input purge timing, sample duration, flow duration, and select one or more trigger conditions (e.g., time-based, voltage-based, half-cycle, etc.). If required parameters are missing or logically inconsistent, a red error message is shown and the process is blocked to ensure system safety. Once active, the system performs purging, monitors trigger conditions, and logs sampling actions automatically.

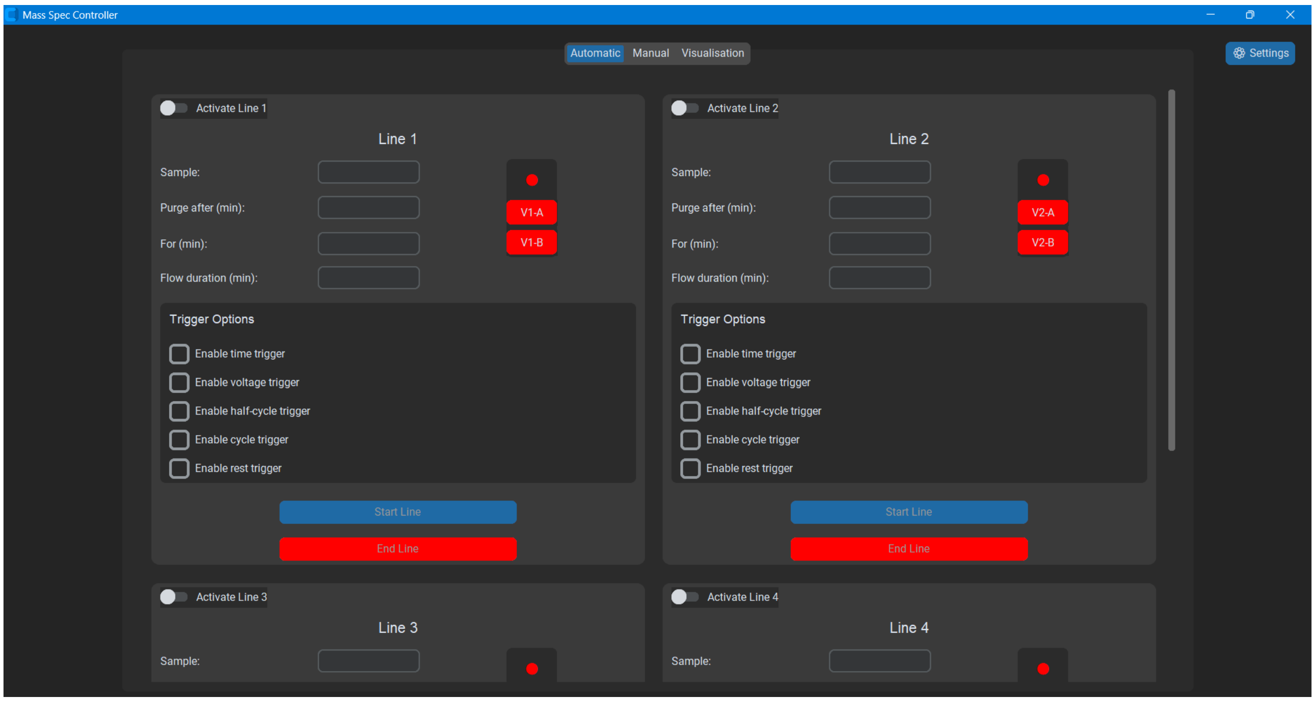


Figure 4.9: Automatic logic interface tab

Manual Tab The Manual tab (Figure 4.10) provides direct control over all solenoid valves in the system. Each valve is represented by a color-coded button—red indicating a closed state, green indicating an open state. This mode is primarily intended for maintenance, calibration, or manual intervention during debugging. It bypasses all automation logic, giving the user full control over valve states in real time.

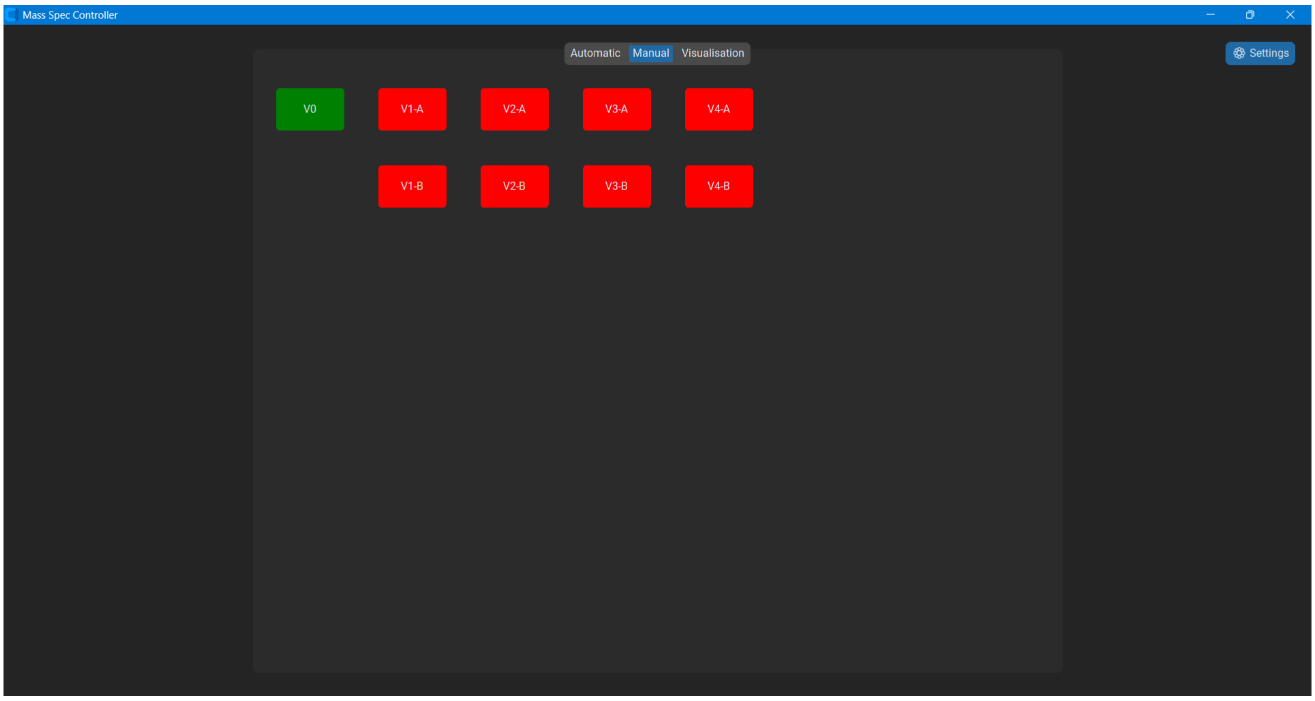


Figure 4.10: Manual interface tab

Visualisation Tab The Visualisation tab (Figure 4.11) offers a simplified overview of current valve states during system operation. It is optimized for real-time observation during automatic sampling procedures. Only the valve state for the active lines is visualized; no interaction is possible in this tab to prevent accidental interference. This view is especially useful during long-duration experiments where monitoring flow routing is important but direct control is not needed.

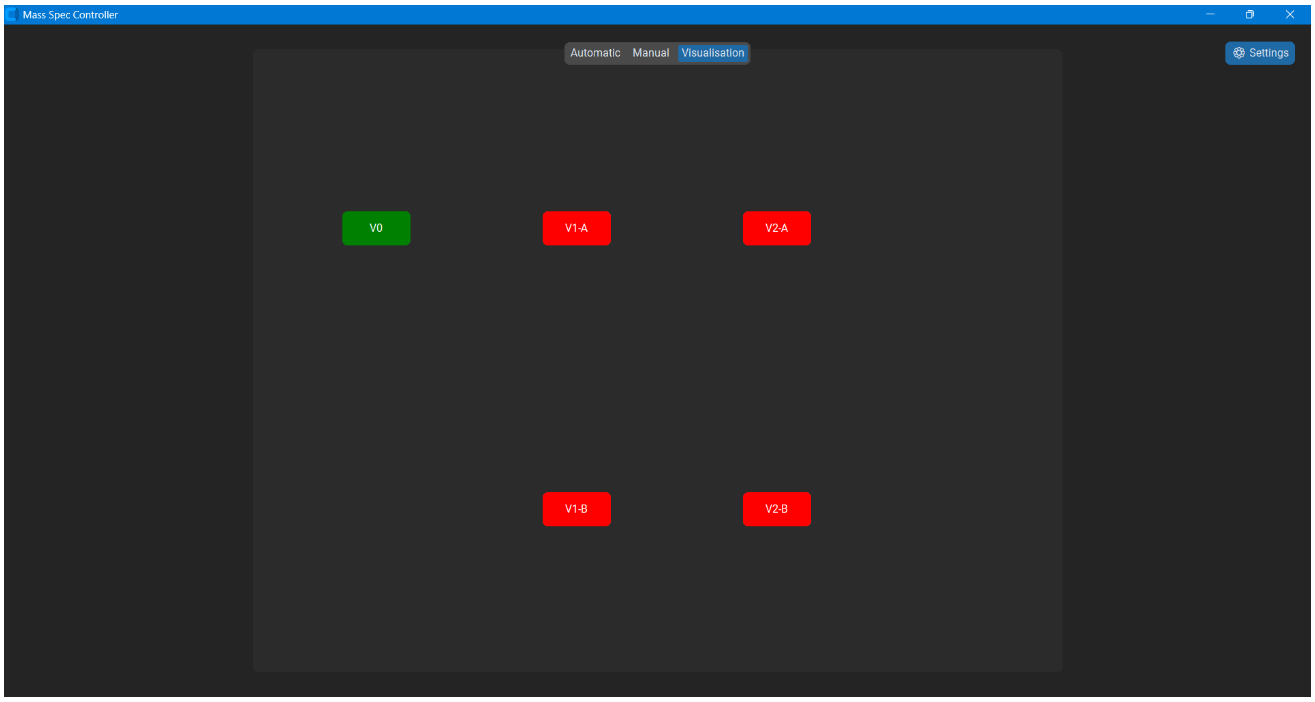


Figure 4.11: Visualisation interface tab

Design Considerations The interface was developed with usability, clarity, and modularity in mind. The structured layout and the strict parameter validation ensure intuitive use for the operators. All GUI events are linked to background processes via callbacks, ensuring real-time responsiveness while separating interface logic from control logic.

4.3.4 Data Logging and Storage

Data logging in the automation software is handled through a dedicated logging function, which is called each time a measurement is triggered. The logging system creates or updates a CSV or Excel file containing structured information about each sampling event. This file is saved locally on the PC running the control software and is uniquely associated with each experiment session.

When a trigger condition is met, the software collects the following data points:

- Timestamp (date and time)
- Elapsed time since experiment start
- Active line number
- Trigger type(s) that activated the measurement
- Sampling duration
- Instantaneous voltage read from the potentiostat

These values are passed to the logging function and written as a new row in the log file. A global timer, initialized at the start of the first active line, provides a consistent time reference across all measurements. The log is updated in real time to ensure no data is lost in case of interruption or system failure.

Each line is appended to the file using standard Python file I/O functions or a high-level data export library 'pandas'. The resulting log file allows the operator to perform post-experiment analysis, like aligning the data with MS results.

4.3.5 Software Testing and Validation

To ensure reliability and correct system behavior during operation, the developed software was extensively tested under both simulated and experimental conditions. The validation process focused on four key aspects: trigger response accuracy, valve actuation timing, data logging integrity, and interface stability.

Trigger Evaluation Each of the implemented triggers (time-based, voltage-based, half-cycle, full-cycle, and rest) was independently tested using previous potentiostat data. Boundary conditions were applied to confirm that triggers only activate within their configured thresholds and debounce logic (e.g., minimum/maximum voltage limits) correctly filters out noise-induced events.

Valve Control Verification All valve combinations were systematically activated through both manual and automatic modes to ensure safe switching logic. Special attention was given to exclusive line activation, purge sequence correctness, and transition timing between purge and sampling states. Physical observation and serial feedback confirmed proper execution.

Logging Consistency Each triggered measurement was checked for correct logging of timestamp, trigger type, sampling duration, and line ID. Edge cases such as simultaneous triggers and rapid reactivation were also tested to verify that duplicate or missed log entries do not occur. Log files were opened in spreadsheet software to confirm formatting, completeness, and time alignment.

Interface Robustness The GUI was tested with partial inputs, invalid settings, and user interruptions to ensure errors are caught and clearly reported. Blocking behavior (e.g., start button disabled until all fields are filled) was verified. The application was also tested over extended durations to confirm memory and performance stability.

Final Integration The complete system was finally validated using real cycling experiments with active gas sampling. Triggered measurements were compared against the real-time voltage profile, and corresponding valve actuation was verified via live visual feedback and the generated log file. These tests confirmed that the system performs reliably in real experimental conditions.

Chapter 5

Pouch Cell Fabrication Process

5.1 Introduction

This chapter outlines the procedure used to fabricate the pouch cells employed throughout this work. It describes the preparation of the individual components, the step-by-step assembly process, and the sealing method used to ensure reliable operation during electrochemical cycling and gas analysis. Photographic documentation is provided to illustrate each stage of the process and support reproducibility. The chapter concludes with a summary of common issues encountered during fabrication, highlighting critical points of attention for ensuring functional and consistent cells.

5.2 Fabrication Process

The fabrication of the pouch cells was performed step by step in the dry room. A dry room is a climate controlled environment designed to maintain extremely low humidity levels. This step is essential as residual water can react with the electrolyte, particularly $LiFePO_4$, forming hydrofluoric acid (HF), which degrades both the electrolyte and electrode surfaces. The pouch cells fabricated for this study follow a single-stack configuration, consisting of a double-sided LFP cathode placed between two graphite anode layers. The procedure includes preparation of materials, electrode punching, stacking, electrolyte filling, and vacuum sealing of the pouch. Each step is documented with corresponding images taken during production.

5.2.1 Materials Overview

The key materials used for pouch cell assembly include cathode and anode electrode foils, a porous separator, and aluminum-laminated pouch foil for the pouch. Additionally, pre-welded current collector tabs were prepared for integration during sealing. These base materials are shown in Figure 5.1 and Figure 5.2.

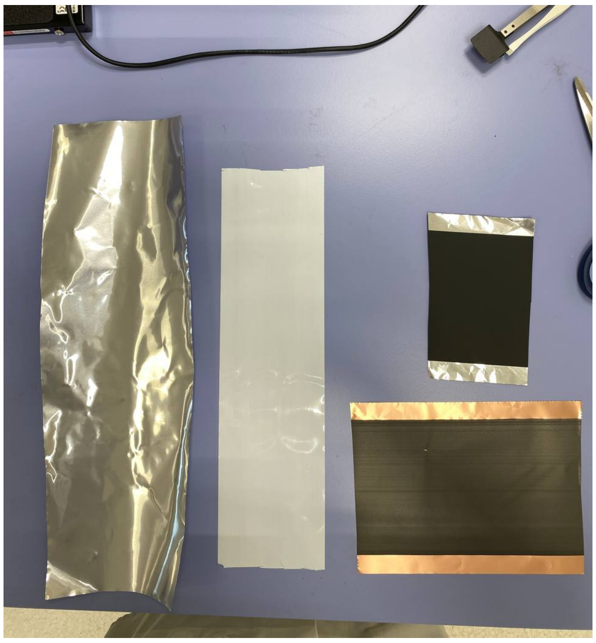


Figure 5.1: Overview of the primary cell components: anode foil, separator, cathode foil, and pouch foil.

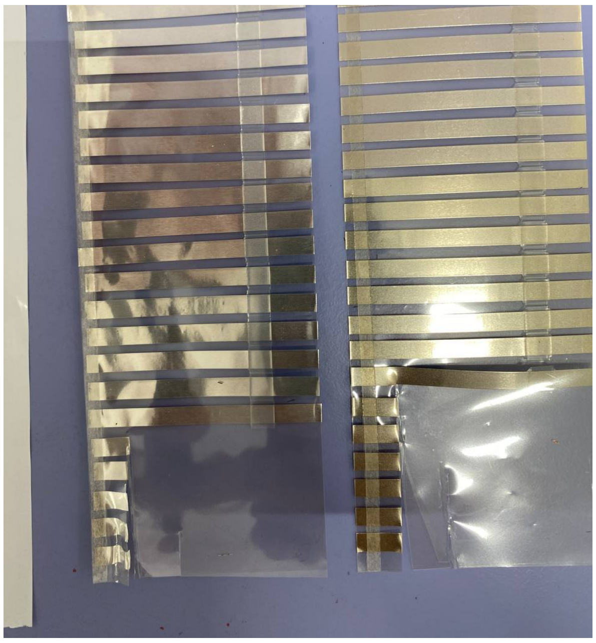


Figure 5.2: Current collector tabs for anode and cathode connection.

5.2.2 Electrode Punching

To prepare the electrodes for stacking, rectangular electrodes were punched from both the anode and cathode foil using a mechanical punching tool. The punched electrodes feature a dedicated tab area designed to facilitate secure welding of the current collector tabs during assembly (as shown in later steps). Care was taken to avoid contamination between active materials by cleaning the tools between operations. The punching process is illustrated in Figure 5.3, and the resulting electrodes are shown in Figure 5.4.

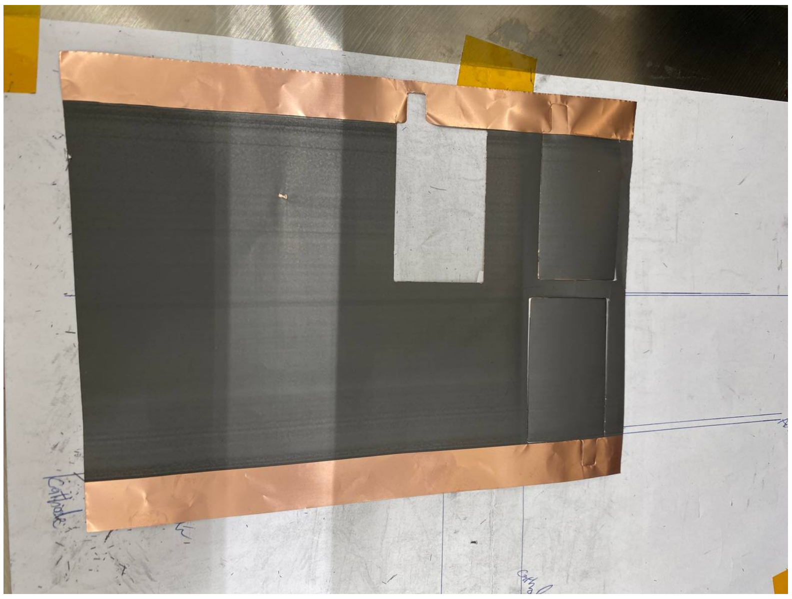


Figure 5.3: Punching of electrode sheets to uniform size.

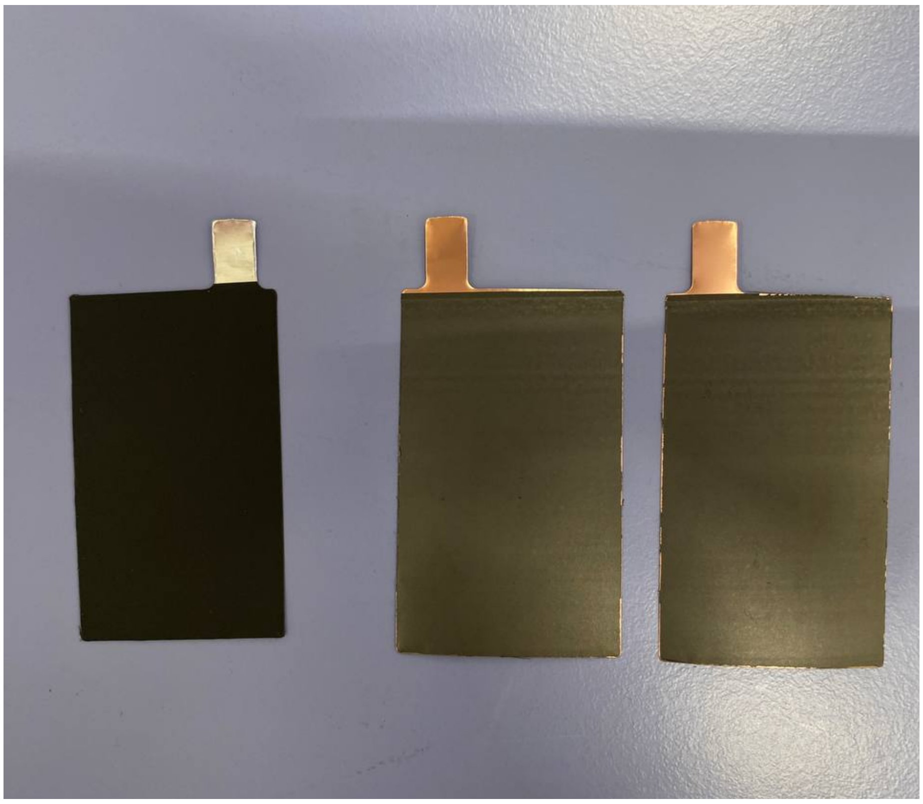


Figure 5.4: Finished punched electrodes: cathode (left) and anodes (right).

5.2.3 Tab Placement and Welding

Following the punching of the electrode sheets, current collector tabs were attached to both the anode and cathode electrodes. The purpose of these tabs is to enable electrical connection between the electrode stack and the external circuitry once the pouch is sealed. Two tabs were welded to the electrodes using a mechanical tab-pressing tool inside the dry room.

For the cathode, aluminum tabs were used, while nickel tabs were chosen for the anodes. This selection is based on the electrochemical compatibility of each material:

- **Aluminum** is commonly used for cathode current collectors due to its excellent conductivity and electrochemical stability at high positive potentials.

- **Nickel** is preferred for anode connections because it resists corrosion and provides stable contact at the lower (negative) potential range.

The two anode sheets were connected to a single nickel tab by overlapping the edges and applying pressure using the automated tool. This ensured reliable electrical contact and mechanical fixation. The same procedure was followed for the cathode electrode and its aluminum tab. The result is shown in Figure 5.5.

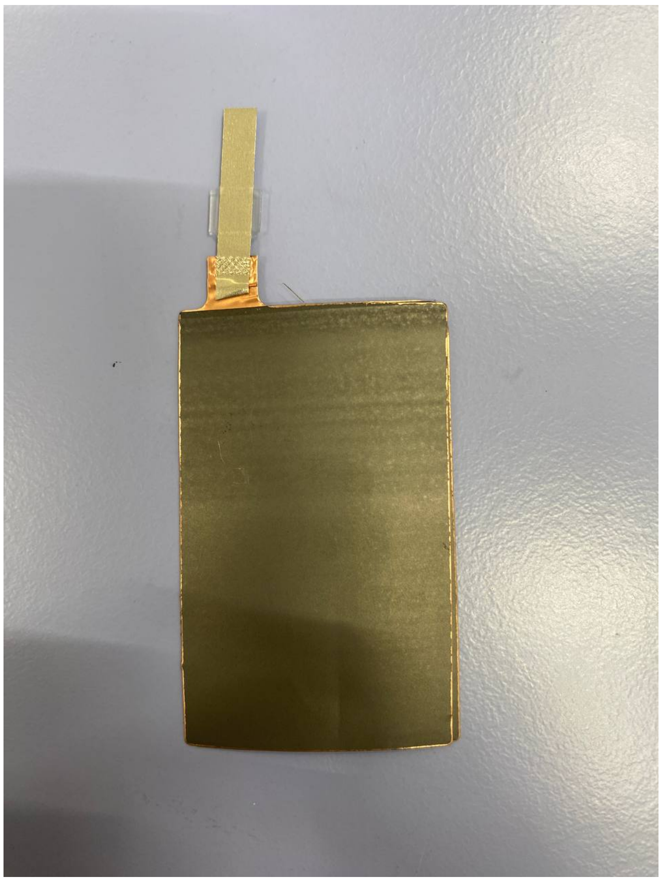


Figure 5.5: Placement of nickel tab on the anode electrodes.

5.2.4 Separator Placement and Stack Assembly

After welding the tabs to the electrodes, the separator was introduced to electrically isolate the anode and cathode layers. A single porous separator sheet was used per pouch cell. Instead of individually punching or cutting two separate pieces, the separator was simply folded around the cathode layer, creating a sandwich structure where the cathode is encapsulated between two separator surfaces, as shown is Figure 5.6. This method simplifies handling and ensures consistent overlap, reducing the risk of edge short circuits during cycling.

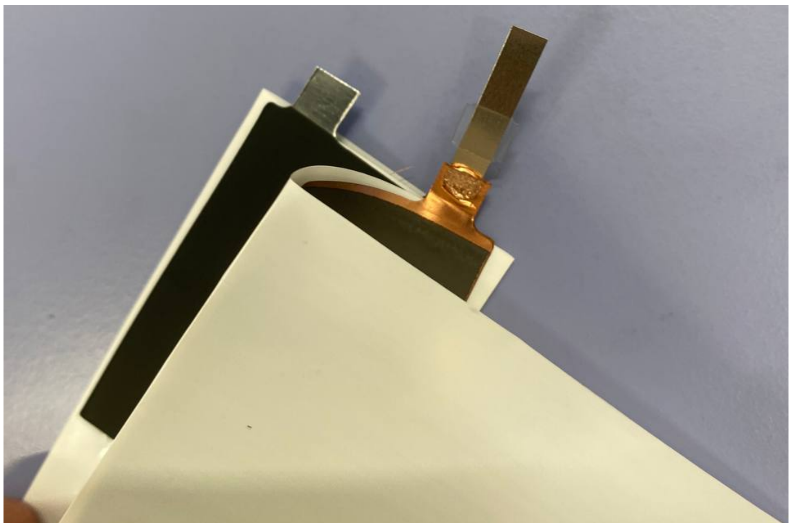


Figure 5.6: Close-up of the electrode stack showing the folded separator wrapping around the cathode and positioned between two anode layers.

The anode layers were then aligned on either side of the folded separator-cathode assembly. To maintain precise alignment and mechanical stability during sealing, the full stack was temporarily fixed using high-temperature resistant green polyimide tape. This tape holds the multilayer assembly together without interfering with the sealing process or chemically reacting with the electrolyte.

An illustration of the aligned stack just before pouch insertion is shown in Figure 5.7.

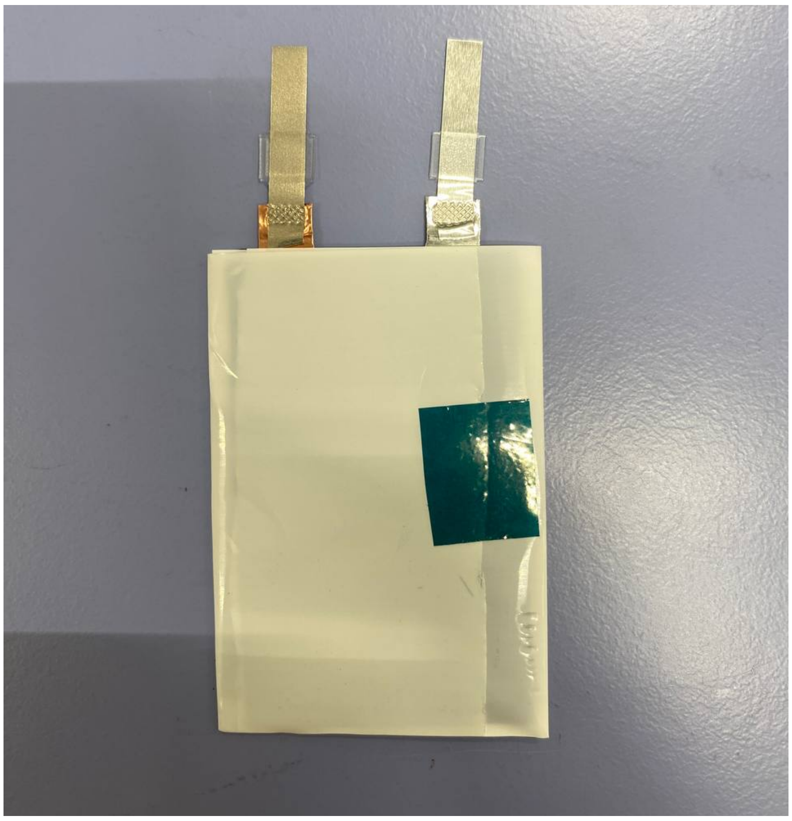


Figure 5.7: Completed electrode stack with polyimide tape holding the layers in position.

5.2.5 Pouch Preparation and Stack Insertion

To create the pouch housing, a rectangular sheet of aluminum-laminated pouch foil was folded in half. The bottom edge of this fold was immediately sealed using the heat sealer tool as shown in Figure 5.8.

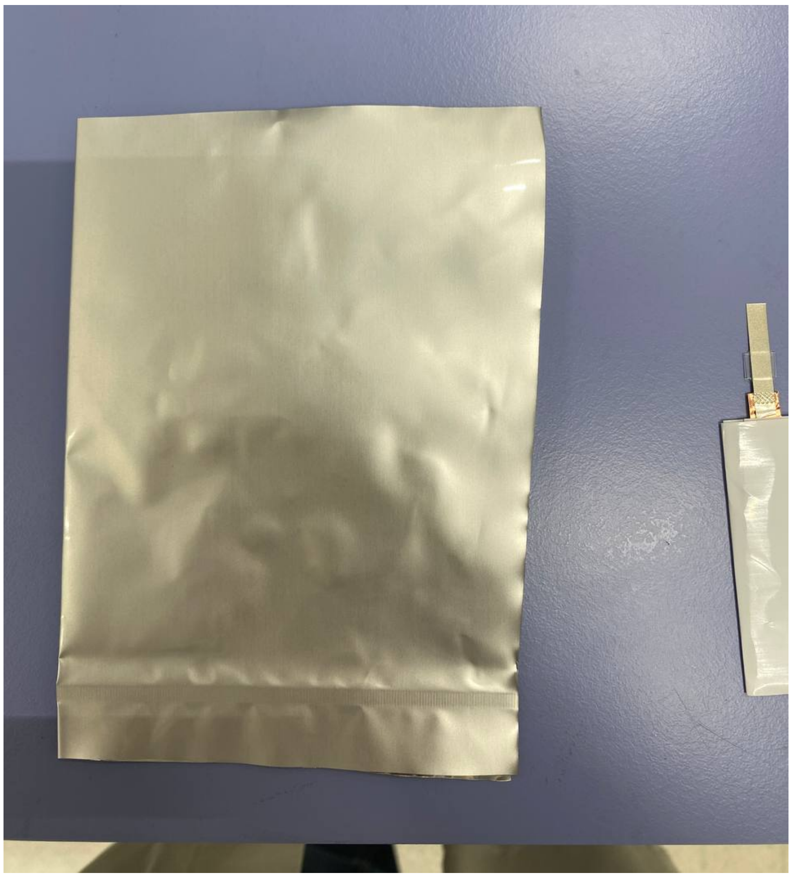


Figure 5.8: Sealing the bottom edge of the folded pouch foil to form the base of the cell.

The assembled electrode stack was then placed into the pouch (see Figure 5.9). It was positioned carefully in the corner formed by the fold and the sealed bottom edge.

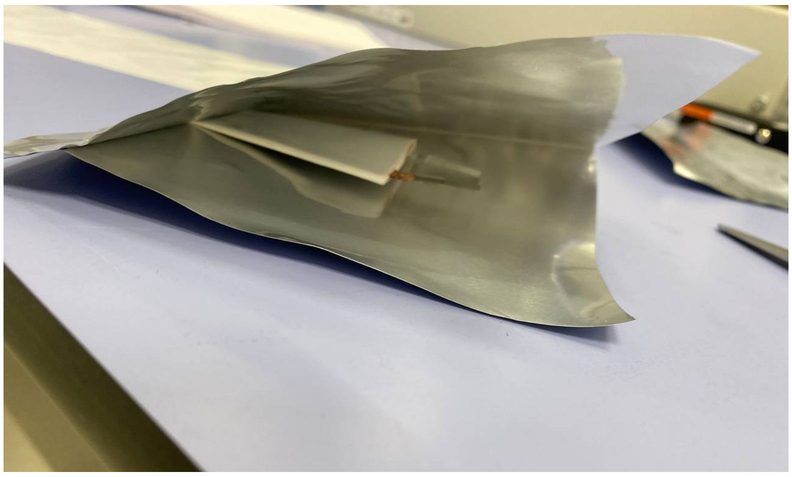


Figure 5.9: Electrode stack inserted into the corner of the folded pouch.

Afterwards, the top side was cut so that the tabs extend outwards for later connection. Finally, the top side of the pouch was sealed. This step required careful alignment of the tabs with the adhesive glue strips inside the pouch. These strips help fix the tabs in place and ensure a reliable seal around them. Figure 5.10 shows the result after sealing both the top and bottom edges.

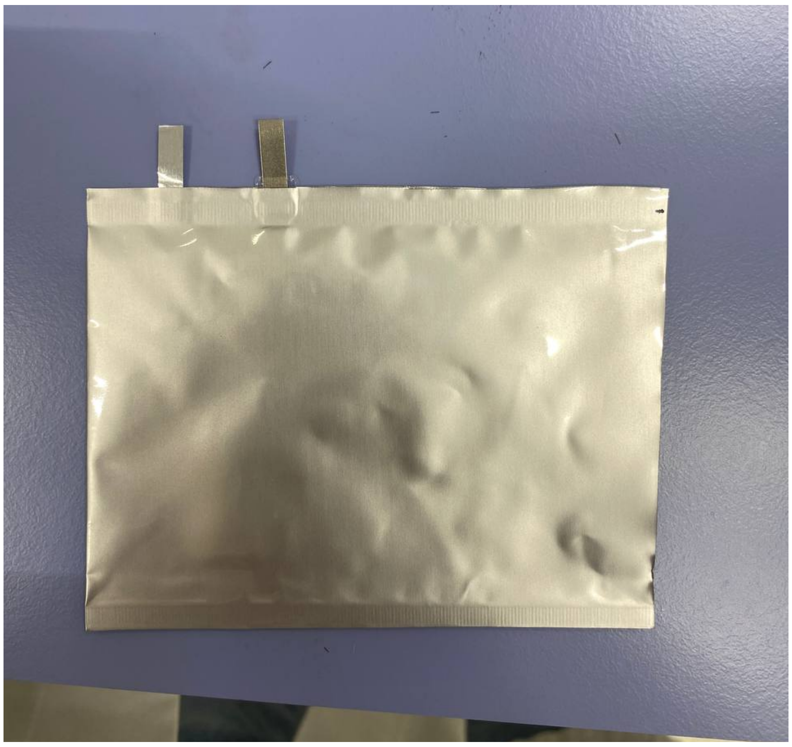


Figure 5.10: Top sealing of the pouch with the tabs aligned on the glue strips to ensure proper sealing.

5.2.6 Electrolyte Filling and Final Sealing

After partial sealing, the cell was prepared for electrolyte injection. A commercially available lithium-ion electrolyte solution, shown in Figure 5.11, was used. This electrolyte is a mixture of ethylene carbonate (EC) and ethyl methyl carbonate (EMC), with a total lithium salt concentration of 2.0 mol/L. This solvent combination is widely used in lithium-ion batteries due to its balance between ionic conductivity, film-forming properties, and thermal stability [40]. Using a calibrated volumetric pipette, a controlled amount of electrolyte was dropped directly onto the exposed electrode stack (see Figure 5.12). This ensures uniform wetting of both the separator and active materials, while avoiding overfilling that could interfere with the sealing process.

Figure 5.11: Electrolyte consisting of a binary solvent mixture of EC and EMC with a total lithium salt concentration of 2.0 mol/L.

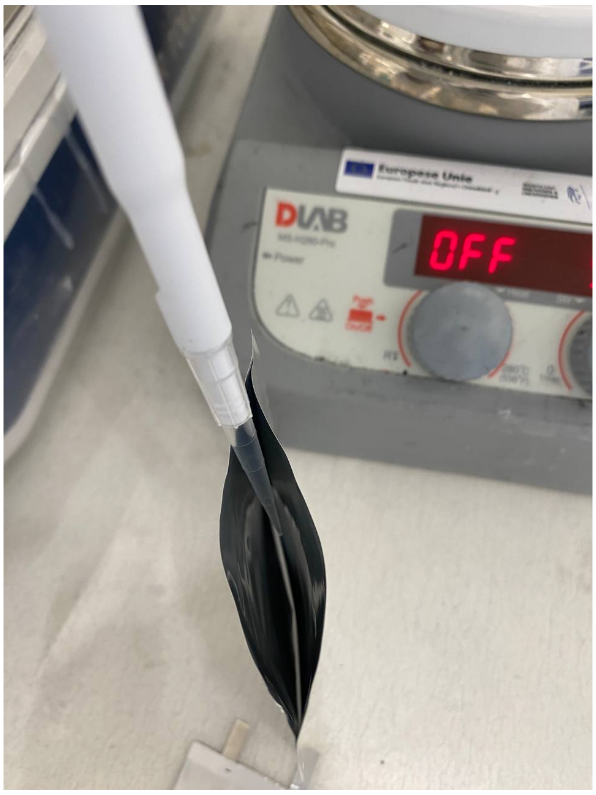


Figure 5.12: Dropping of electrolyte onto the electrode stack using a volumetric pipette.

Immediately after electrolyte addition, the cell was transferred to a vacuum chamber equipped with a heat-sealing tool. This step allows remaining air to be extracted before the pouch is fully sealed, thereby minimizing trapped gases and ensuring long-term stability (Figure 5.13).

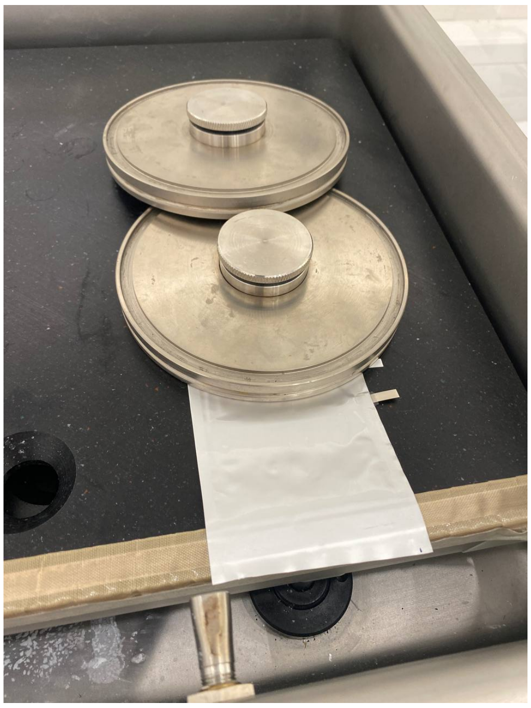


Figure 5.13: Pouch cell placed into a vacuum heat-sealer to evacuate air and close the final edge.

Once the chamber reached the target vacuum level, the final open side of the pouch was sealed. The resulting pouch cell, now fully enclosed, is shown in Figure 5.14. The cell was then inspected visually and gently pressed to verify mechanical sealing and to detect any potential leakage of the electrolyte.

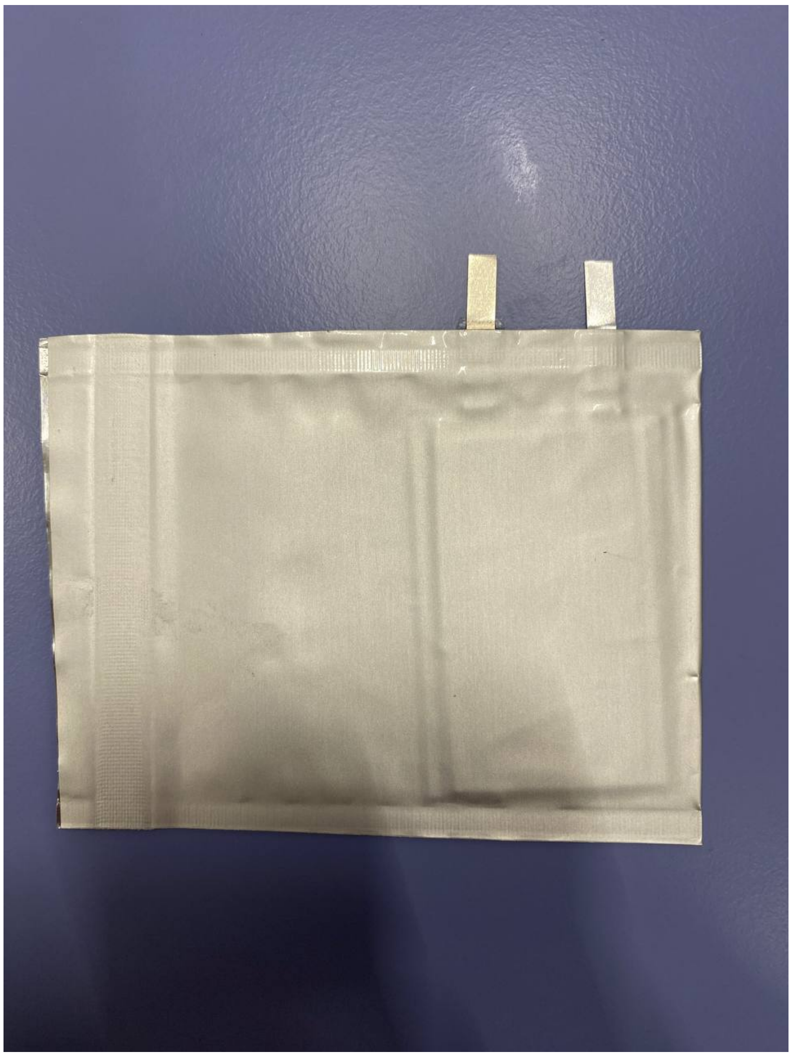


Figure 5.14: Fully sealed pouch cell after vacuuming.

As seen in Figure 5.14, the initial pouch foil was deliberately cut oversized relative to the electrode stack. This design choice simplifies vacuum sealing, as the extra material allows easier positioning in the vacuum chamber and ensures full edge contact with the heating elements.

After the initial vacuum seal, a second heat seal was applied on the same open side, this time positioned closer to the electrode stack (see Figure 5.15). This created a tighter enclosure that matches the footprint of the internal components, ensuring compactness as the pouch cell has to fit in the designed enclosure.

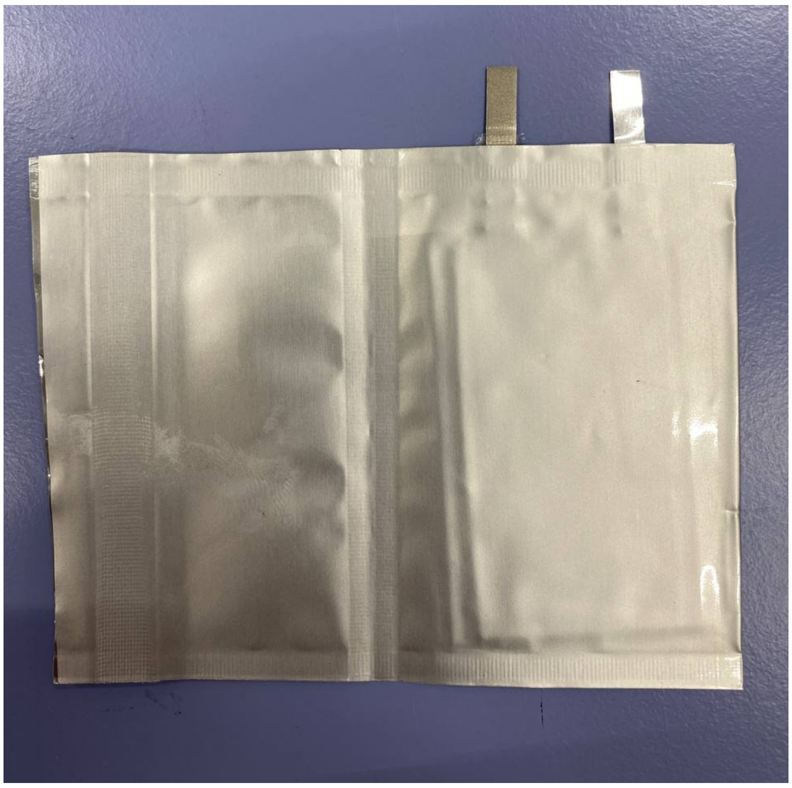


Figure 5.15: Second heat seal applied closer to the electrodes, reducing excess internal volume.

Finally, the excess pouch foil beyond the second seal was trimmed off using scissors, resulting in a finished pouch with dimensions closely matching the electrode geometry (Figure 5.16). This final shape improves cell handling, facilitates fixture mounting, and reduces gas volume inside the pouch that could otherwise dilute evolved species.

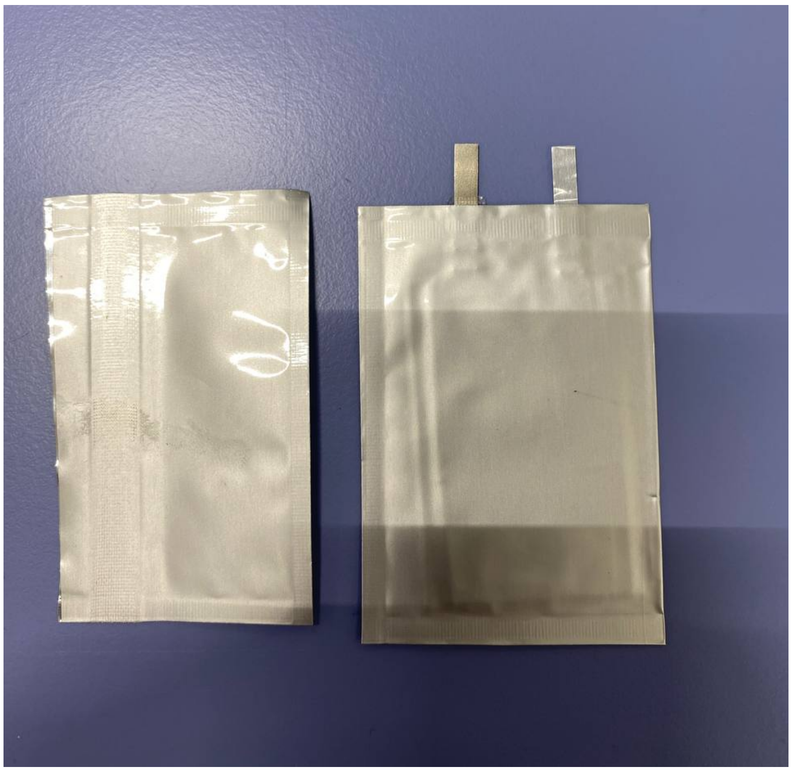


Figure 5.16: Excess pouch foil trimmed after final sealing.

5.3 Common Issues and Failure Modes During Fabrication

Despite careful handling, several failure modes can occur during pouch cell fabrication, particularly due to the manual nature of lab-scale assembly. Figure 5.17 shows three representative examples of such issues.

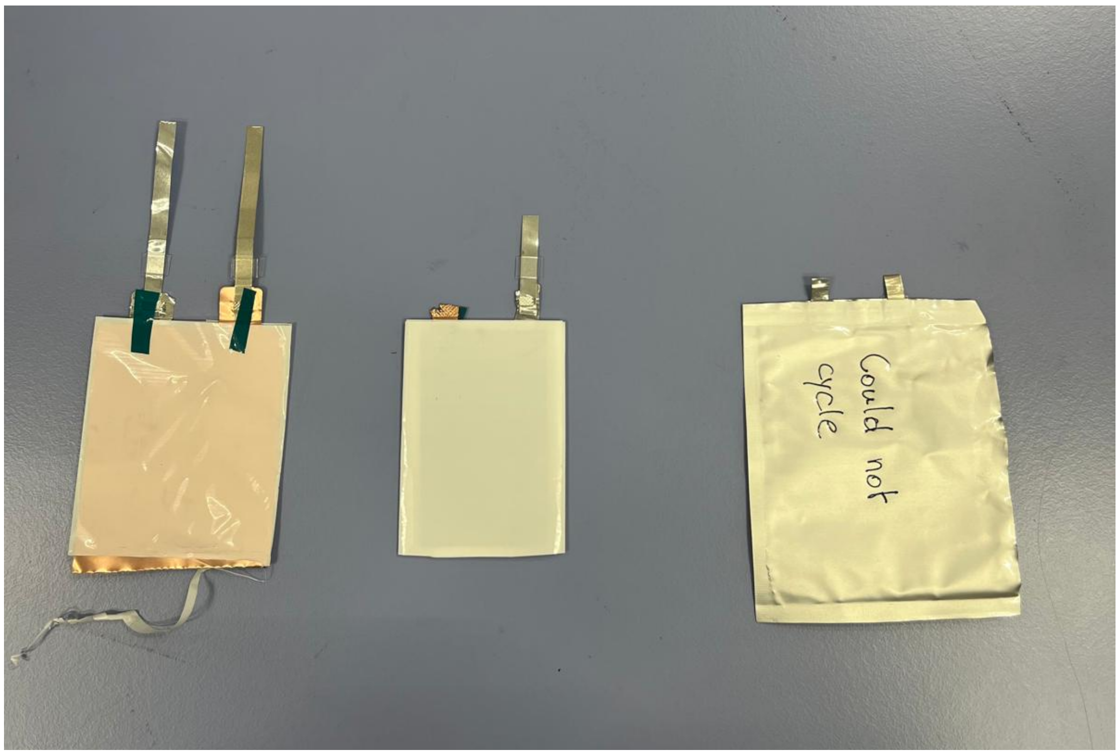


Figure 5.17: Examples of typical fabrication errors. From left to right: (1) separator too short, causing electrode contact; (2) broken tab due to improper welding or mechanical stress; (3) improperly vacuum-sealed pouch resulting in failed cycling.

On the left, the separator sheet was cut too short, which led to direct contact between the anode and cathode layers—causing internal short circuits and immediate cell failure. In the middle, a current collector tab has broken off, likely due to incorrect welding tool parameters or mechanical damage during handling. Such failures are common when dealing with thin electrode foils and require careful adjustment of welding force and duration. On the right, the pouch cell was not vacuum-sealed properly. Residual air inside the pouch interfered with electrolyte wetting and internal pressure regulation, ultimately preventing the cell from functioning correctly during cycling.

These examples highlight the importance of precision, consistency, and environmental control in pouch cell fabrication. Even small errors in dimensions, alignment, or sealing can significantly affect the electrochemical performance and safety of the final device.

Chapter 6

Experimental Validation and Measurements

6.1 Experimental Goals and Scope

The first objective of this chapter is to assess how different sampling strategies influence the observed gas-evolution signatures during cycling. In particular, we aim to determine the correlation between the timing of charge/discharge events and the sensitivity, resolution, and quantitative reliability of gas detection under three distinct measurement protocols:

(i) **Fixed-interval accumulation.** Gases are sampled for a defined period (two hours) at regular intervals, after which the sampling line is switched to the mass spectrometer for analysis. By integrating the gas produced over a long collection window, this protocol increases the total analyte amount and thus the analytical sensitivity, at the expense of temporal specificity.

(ii) **Continuous operando monitoring.** A constant flow of argon carries evolved gases directly into the mass spectrometer throughout the entire charge–discharge sequence. This method maximizes temporal resolution but risks low species concentration in the gas stream, potentially reducing signal-to-noise ratio when the instantaneous gas-generation rate is small.

(iii) **End-of-experiment bulk measurement.** After many complete cycles, the gas lines are opened and all accumulated gas is analyzed in one batch. This approach provides an integrated measure of total gas production over an extended period, allowing direct comparison of cumulative gas yields, but sacrifices any information on the cycle-to-cycle dynamics.

Throughout all experiments, the pouch cell was cycled between 2.5 V and 3.65 V with a 12 h rest period prior to the first cycle to ensure homogeneous electrolyte wetting. By systematically comparing these three sampling strategies under identical cycling conditions, we will elucidate the trade-offs between time resolution, sensitivity, and cumulative accuracy, and thereby recommend the optimal protocol for future operando gas-analysis studies.

6.2 System Functionality and Baseline Testing

6.2.1 Validation of Electrical Connection

Figure 6.1 displays the applied potential (E_{we}) versus time over approximately 75 hours, during which ten complete charge–discharge cycles were executed between 2.5 V and 3.65 V. The smooth, repeatable plateaus at 3.65 V and valleys at 2.5 V in each cycle confirm that the potentiostat maintained stable control over the cell voltage.

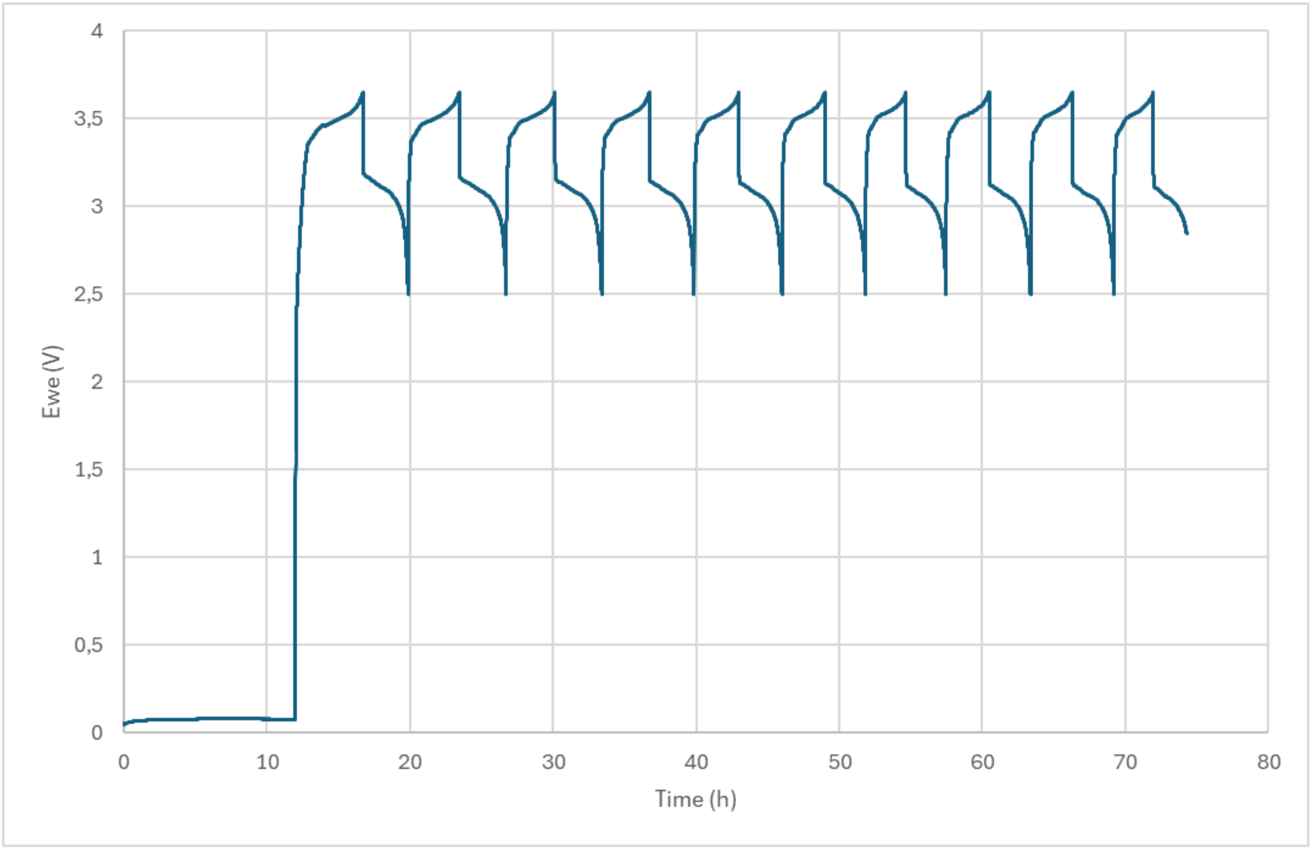


Figure 6.1: Cell voltage (E_{we}) versus time, showing ten consecutive charge–discharge cycles over 75 h.

The absence of irregular voltage drops or noise spikes indicates that the internal electrical feedthroughs and tab-connector interface provided reliable, low-resistance contact throughout prolonged cycling. Consequently, we conclude that the electrical connection scheme is robust and suitable for extended operando gas-analysis experiments.

6.2.2 Fabrication Troubleshooting and Successful Cycling

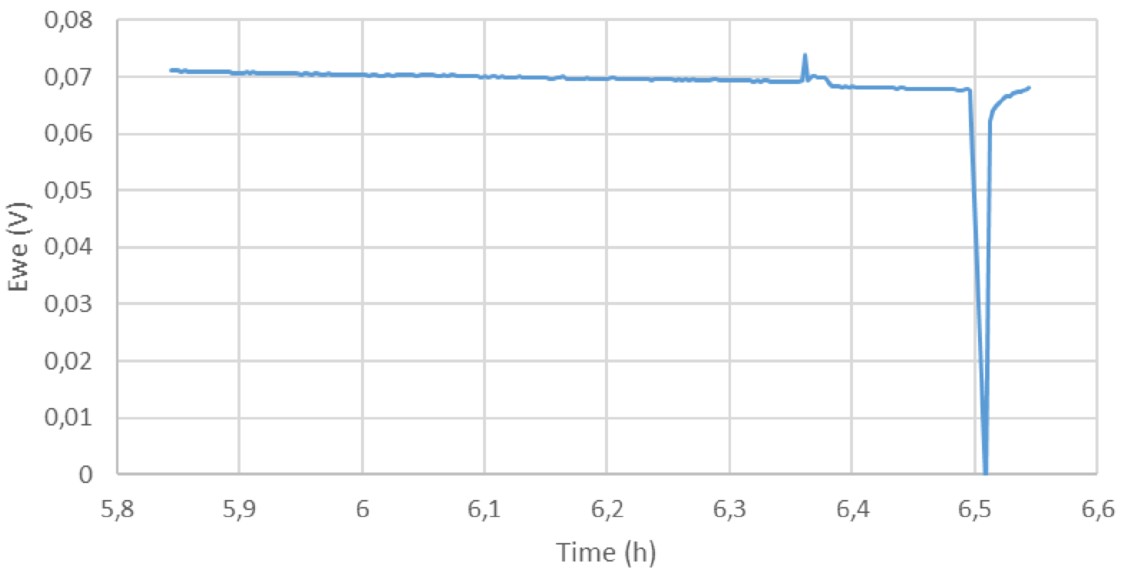
Prior to obtaining a stable ten-cycle voltage profile (Figure 6.4) we encountered multiple fabrication issues, which we systematically diagnosed by testing both assembled cells and loose pouches.

Failed in-cell cycling attempts Three pouch cells fabricated in our custom housing were cycled under varying conditions of electrolyte loading and constant current rate, yet none completed a full charge–discharge cycle, despite verified electrical contact. Figure 6.2 shows the first cell, which stalled during initial charging, Figure 6.1 depicts the second cell, which reached the upper voltage limit but failed to discharge, and Figure 6.3 illustrates the third cell, which exhibited erratic voltage behavior before cutoff.

Loose-pouch verification To determine whether the failures originated in our housing or in the pouch fabrication, two pouches were cycled outside of the custom cell body under identical electrochemical parameters. As shown in Figure 6.3 and Figure 6.4, both loose pouches were unable to undergo full cycling, confirming that the root cause lay in the pouch assembly rather than the operando setup.

Identification of sealing and electrolyte distribution issue Examination of the pouch fabrication process revealed that omission of a proper vacuum-sealing step led to inadequate electrolyte wetting, since we had introduced a gas-release port but had not ensured uniform electrolyte dispersion under vacuum. Once vacuum sealing was applied correctly, thereby forcing electrolyte into all electrode areas, subsequent cell assemblies reliably cycled.

Successful cycling A representative successful cycle is presented in Figure 6.4. The cell charges smoothly to 3.65 V and discharges to 2.50 V, with reproducible plateaus and no anomalous voltage dips or noise spikes. This result validates that proper vacuum sealing and electrolyte distribution are critical to achieving stable electrochemical performance in our operando gas-analysis configuration.



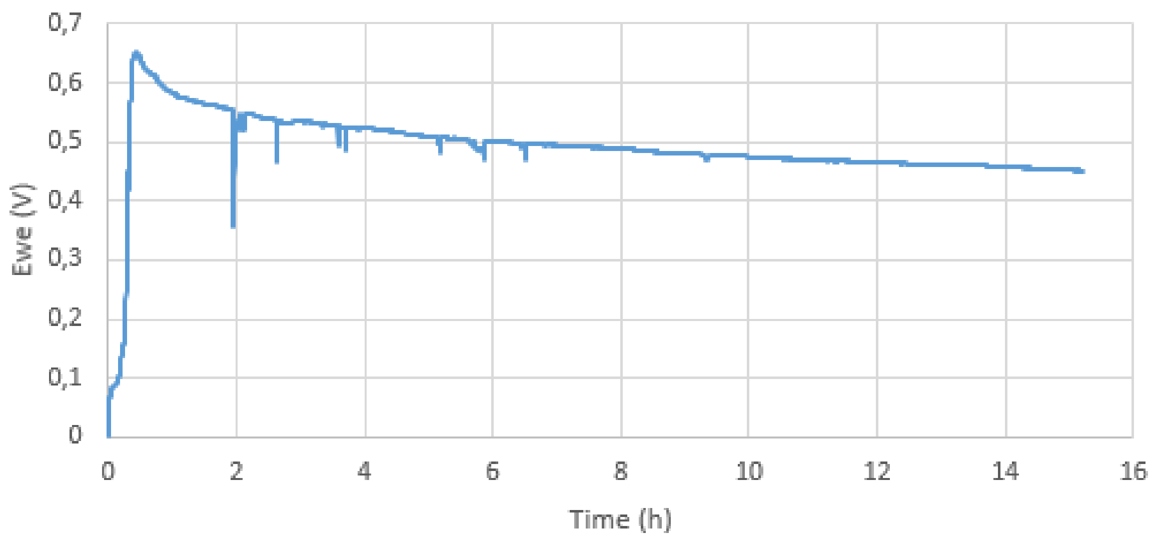
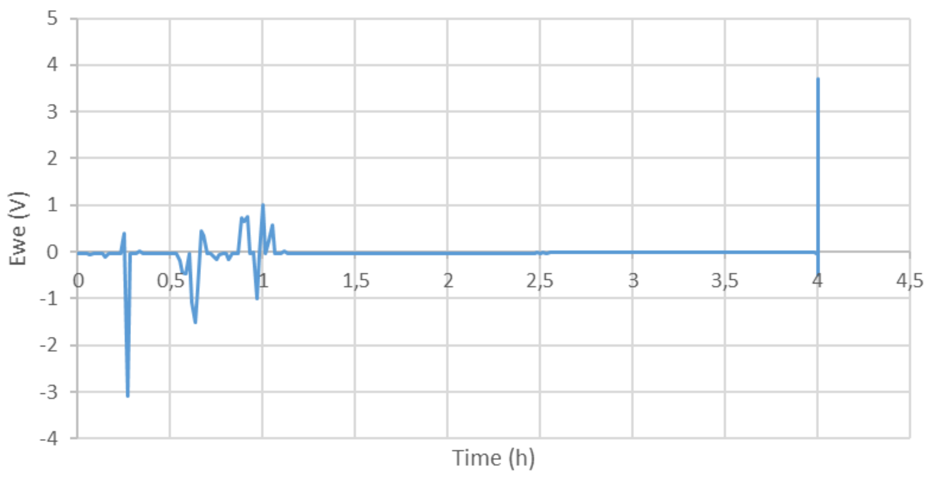


Figure 6.2: Voltage profiles of the first two failed in-cell cycling attempts, showing inability to complete charge–discharge loops.



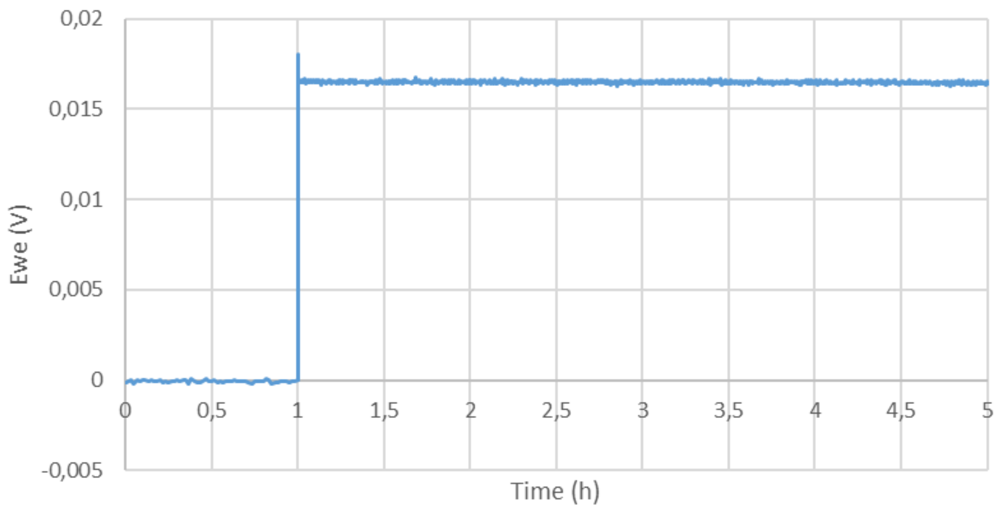
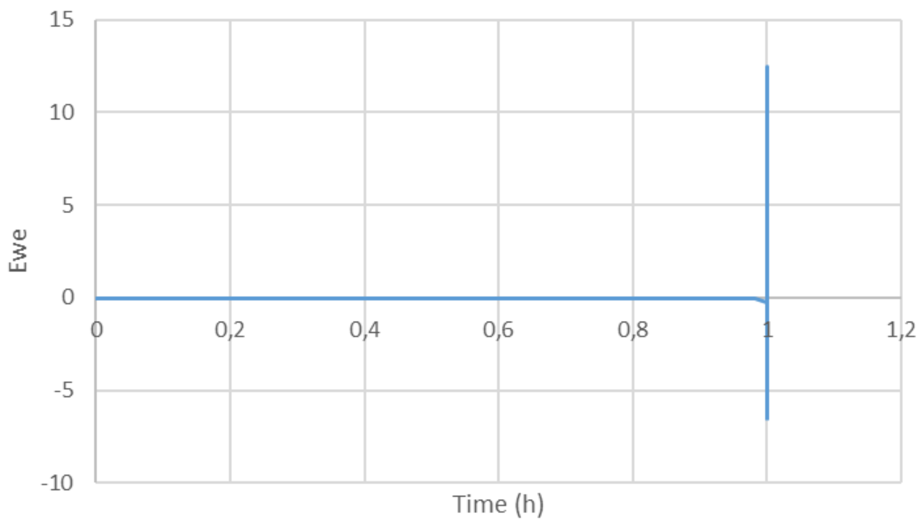


Figure 6.3: Top: third failed in-cell cycling trial with erratic voltage response. Bottom: first failed loose pouch test outside of the cell housing.



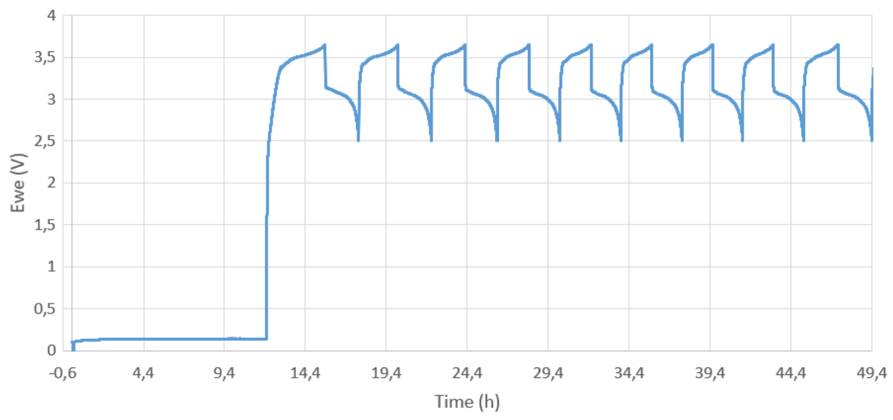


Figure 6.4: Top: second failed loose pouch trial confirming fabrication issue. Bottom: successful ten-cycle voltage profile after implementing proper vacuum sealing and electrolyte distribution.

6.2.3 Leak Testing and Purge Dynamics

After sealing the cell, when residual ambient air remains trapped inside a controlled argon purge at 80 mL/min was initiated to displace nearly all of the remaining air. Figure 6.5 presents the full time series: the pink trace shows the air count-rate, and the black trace shows the argon count-rate. Every "shark-tooth" spike corresponds to a valve transition when the flow path switches from bypass to the cell line. Although the cell remains effectively sealed, very brief air-leakage peaks appear at each switch, likely due to minute amounts of ambient air momentarily entering during valve actuation. These spikes are extremely small and decay immediately under the continuous argon flow.

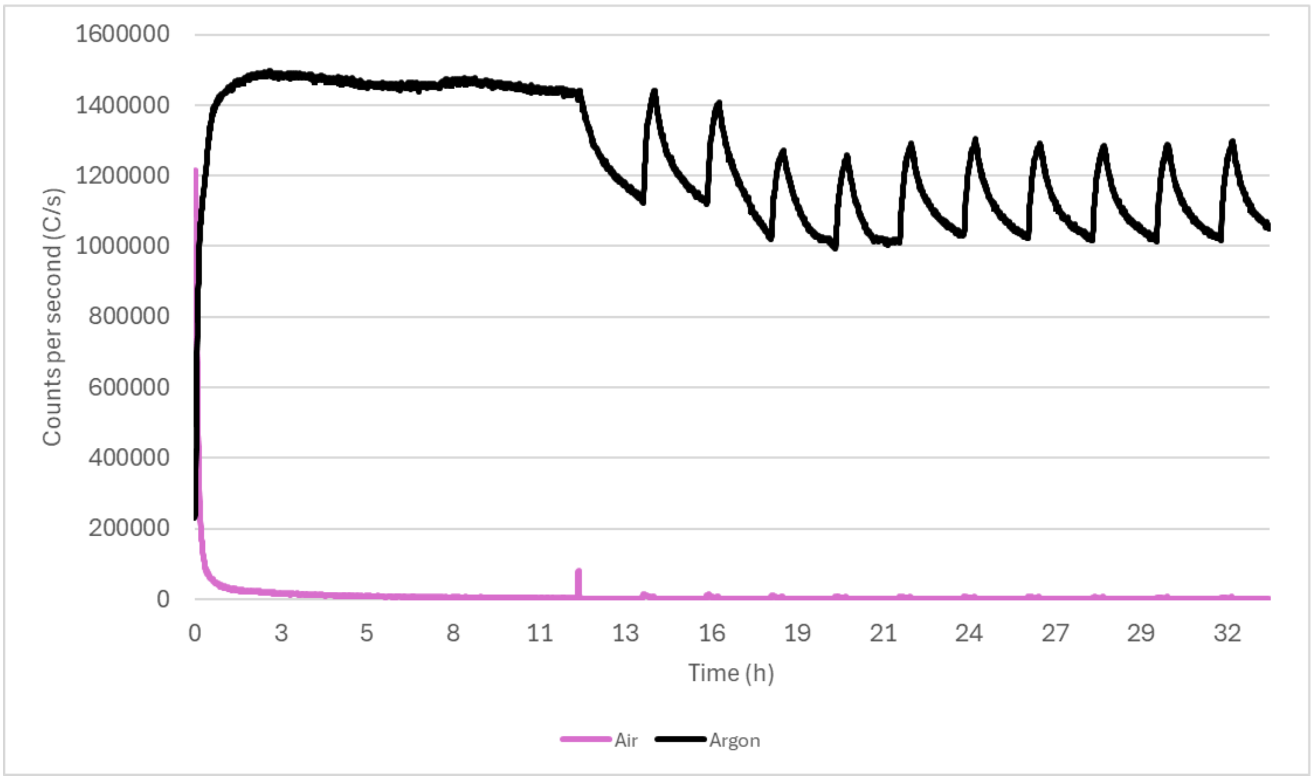


Figure 6.5: Air (pink) and argon (black) count-rates during valve switching. The initial high air count shows trapped ambient air, subsequent spikes correspond to valve operations.

After switching to argon purge, the air signal decays rapidly toward a low steady level. Figure 6.6 provides a zoomed view of this stabilization: following the abrupt drop, the pink trace settles at roughly 800 counts/s.

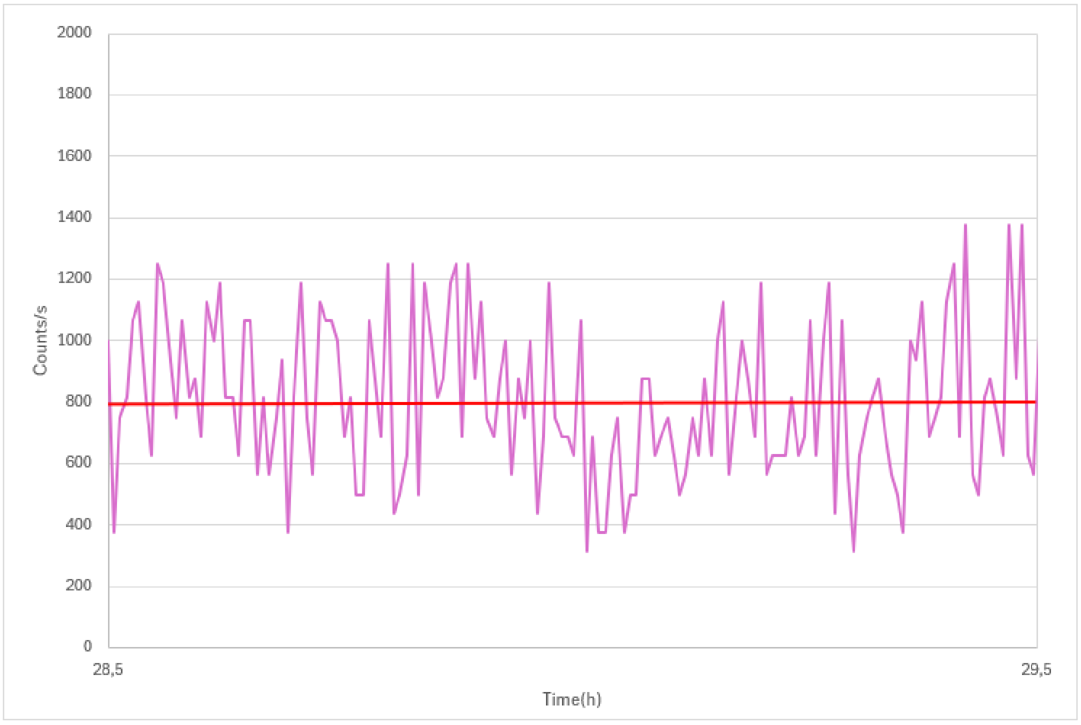


Figure 6.6: Zoom on the air signal during argon purge, showing stabilization at approximately 800 counts/s.

This residual 800 counts/s represents only

$$\frac{800}{1.11 \times 10^6} \times 100\% \approx 0.072\%$$

of the initial trapped-air signal. The rapid decay and low steady-state baseline demonstrate that the argon purge effectively removes nearly all ambient air, verifying the cell's gas-tight integrity under test conditions.

6.3 Exploratory Measurements and Data Presentation

6.3.1 Fixed-Interval Accumulation

Figures 6.7 and 6.8 present the hydrogen, methane, ethylene, and carbon dioxide signals recorded during each 20-minute sampling window taken every two hours. The underlying charge–discharge protocol cycled the pouch cell between 2.5 V and 3.65 V; each gas measurement spans a complete window that may overlap both charge and discharge steps.

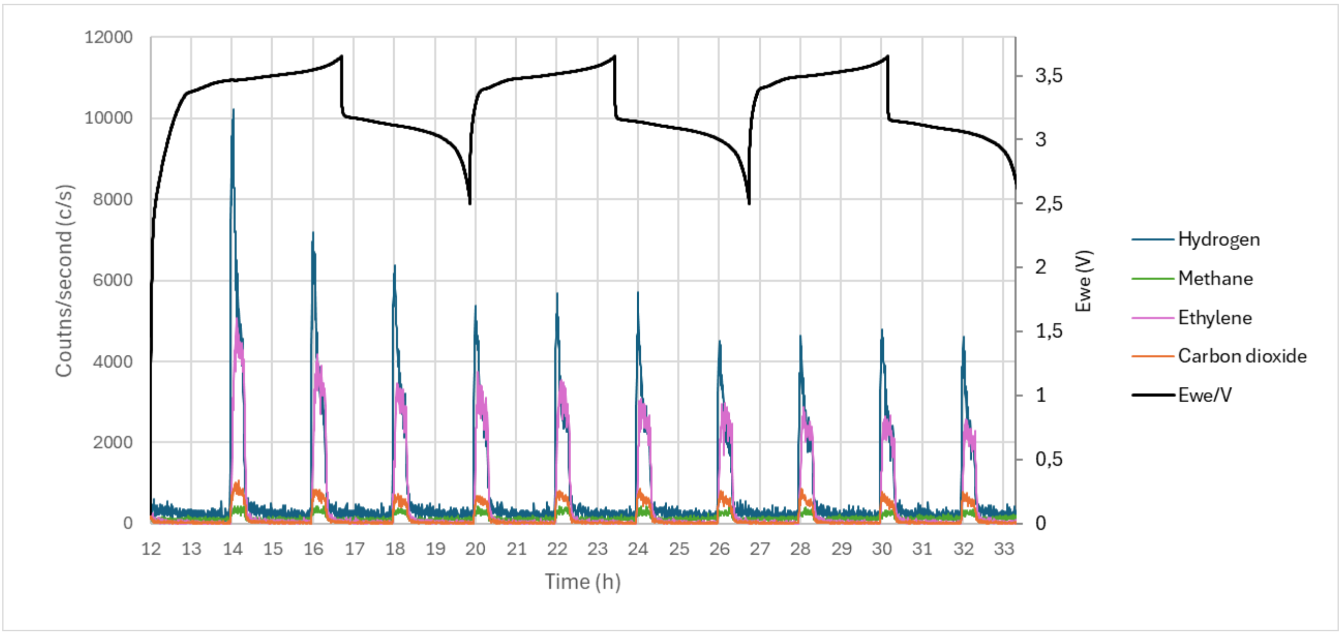


Figure 6.7: Raw counts-per-second traces for H_2, CH_4, C_2H_4, and CO_2 during ten 20-minute sampling intervals spaced every two hours. Each blue, orange, magenta, and green spike corresponds to hydrogen, methane, ethylene, and carbon dioxide, respectively.

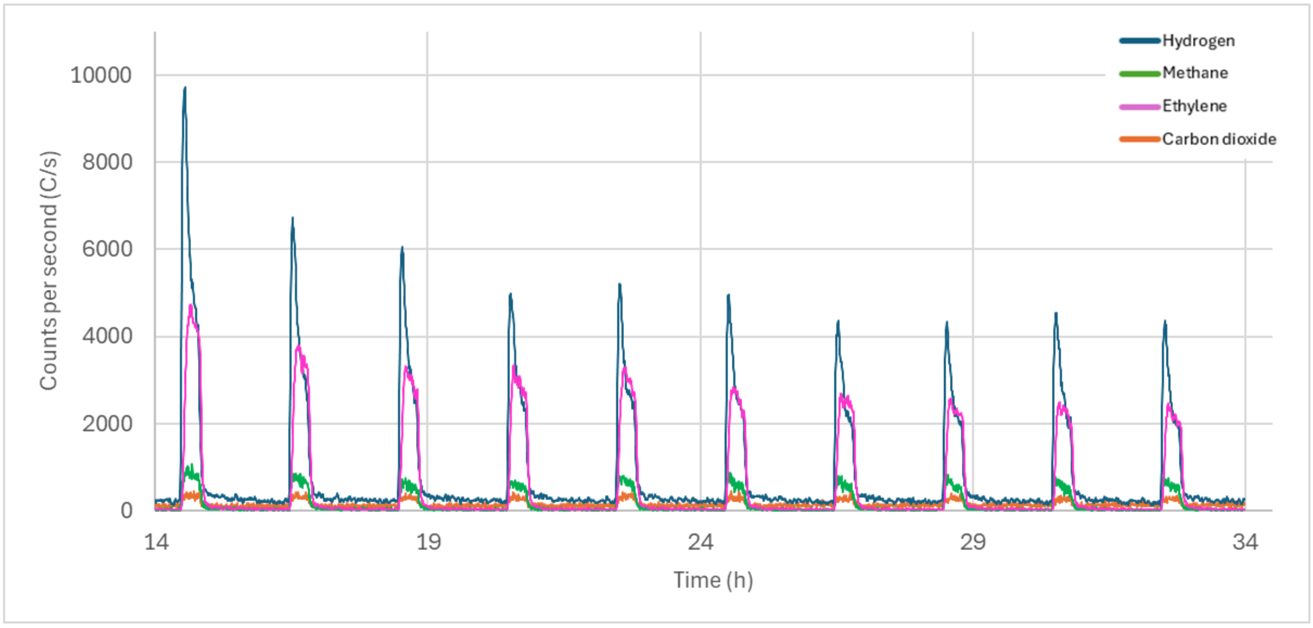


Figure 6.8: Five-point moving average of the data in Figure 6.7, reducing high-frequency noise to clarify overall emission profiles for each 20-minute window.

Observations:

- In both figures, hydrogen (blue) exhibits the largest peak amplitude in each sampling window, followed by ethylene (magenta), methane (green), and carbon dioxide (orange).
- The timing of the peaks is consistent: each 20-minute window begins with a sharp rise in all four gas signals, reaching maxima within the first few minutes, then decaying more gradually over the remainder of the sampling period.
- Peak heights decrease modestly from the first to the third sampling window, then stabilize for subsequent windows.
- No secondary peaks or shoulders appear within any single 20-minute window, suggesting each sampling interval captures a single dominant gas-evolution event.

6.3.2 Bulk Gas Collection After Ten Cycles

In this protocol the pouch cell was subjected to 20 complete charge–discharge cycles between 2.5 V and 3.65 V, after which all accumulated gases were collected in a single 20-minute sampling window under an 80 mL/min argon purge, Figure 6.9 shows the full time series of counts per second for hydrogen, methane, ethylene and carbon dioxide during this bulk measurement

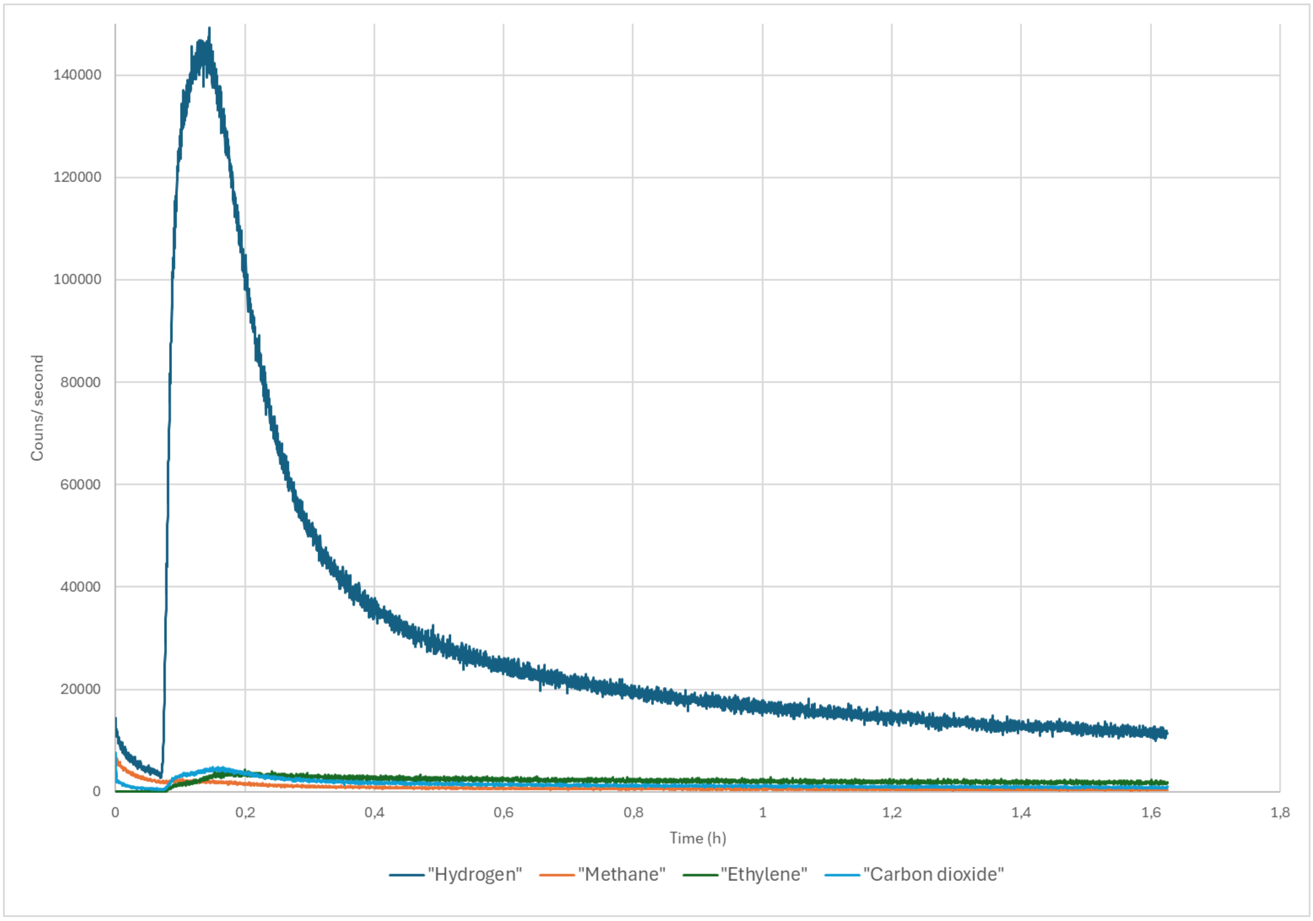


Figure 6.9: Raw emission profiles for H_2 (blue), CH_4 (orange), C_2H_4 (green) and CO_2 (cyan) during the 20-minute bulk sampling following ten cycles

Initial Observations Hydrogen exhibits by far the largest peak reaching approximately 148 000 counts/s, methane and ethylene reach peaks of around 2 200 counts/s and 3 500 counts/s respectively, carbon dioxide peaks at roughly 4 500 counts/s, each species rises sharply within minutes of opening the sampling line then decays over the remainder of the 1.7 h window, no secondary emission events are visible indicating a single dominant release

Zoomed Decay Comparison Figure 6.10 presents a close-up of the decay tails for methane, ethylene and carbon dioxide from 0.15 h onward, methane (orange) falls below 1 000 counts/s by 0.35 h, carbon dioxide (blue) and ethylene (green) decay more slowly reaching approximately 1000 counts/s and 1 800 counts/s respectively by 1.7 h

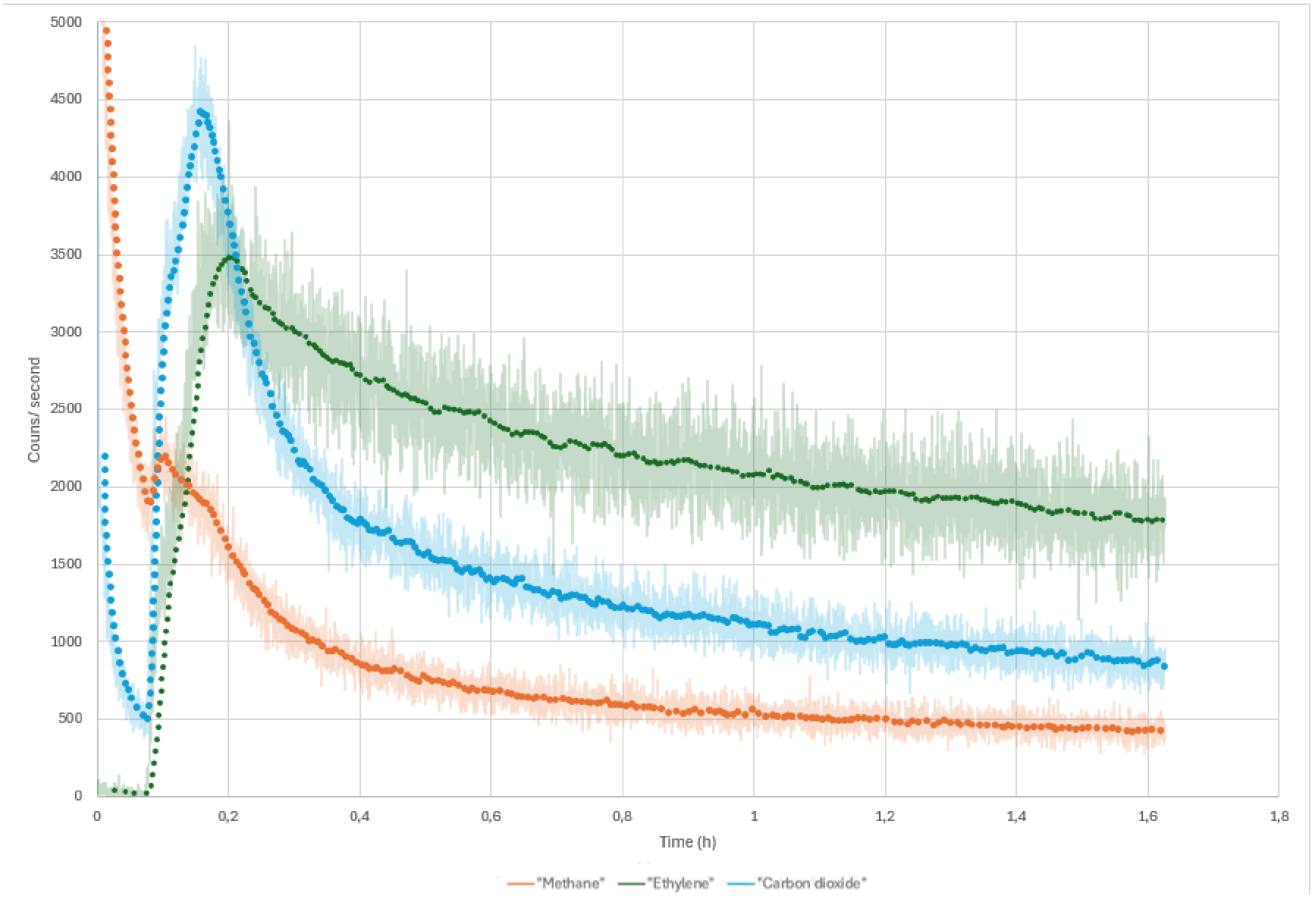


Figure 6.10: Zoomed decay profiles of CH_4, C_2H_4 and CO_2 following the initial peak

Decay Observations Carbon dioxide's rapid decline suggests efficient purging under continuous argon flow, methane and ethylene's more gradual decay indicates stronger interactions with cell components or slower diffusion, the persistent hydrogen tail implies ongoing low-rate evolution or desorption, signal noise remains minimal throughout confirming stable measurement conditions

6.3.3 Full Operando Monitoring

In the full operando experiment the pouch cell was cycled from 2.50 V up to 3.65 V and back while the mass spectrometer sampled continuously at one-second intervals. Figure 6.11 shows the complete gas count-rate traces for hydrogen (blue), methane (green), ethylene (magenta) and carbon dioxide (orange), together with the cell voltage (black).

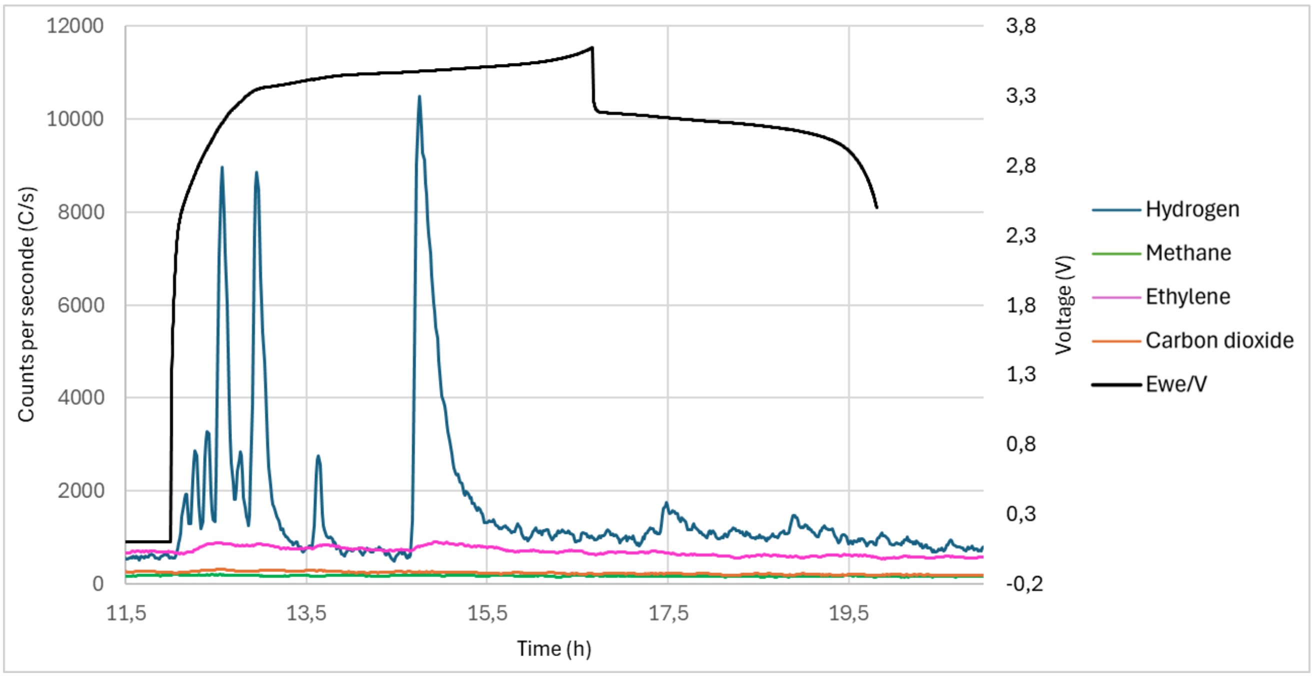


Figure 6.11: Continuous operando gas monitoring during one complete charge–discharge cycle. Black trace: cell voltage (right axis, V); colored traces: 5-point moving average of gas count–rates (left axis, counts s^{-1}).

Observations Two distinct hydrogen peaks emerge during the charging ramp, the first coinciding with the initial activation of the electrode around 12.5 h with a peak of 8900 c/s and the second appearing during the steep voltage rise near 14.7 h with a peak of 11250 , indicating that hydrogen evolution is most intense at these load-driven stages. Ethylene shows a similar but smaller double-peak pattern at the same time points, confirming concurrent electrolyte decomposition. Methane and carbon dioxide signals remain near their low baselines throughout charging, with only slight elevations during the hydrogen peaks. This muted response reflects their lower overall yield and the continuous argon purge that attenuates transient concentrations.

Hydrogen as primary indicator Earlier fixed-interval and bulk measurements demonstrated that hydrogen is the predominant evolved gas, therefore its count–rate peaks here serve as the clearest markers of gas generation events. The double-peak structure aligns directly with the two voltage inflection points, underscoring the strong coupling between cell overpotential and hydrogen evolution.

Secondary gases under continuous purge Continuous sampling through line 1 maintains a steady argon flow, which improves temporal resolution yet dilutes minor gas species. Consequently methane, ethylene and carbon dioxide traces appear nearly flat save for small bumps synchronous with the hydrogen peaks. This behavior confirms that while these decomposition products are generated, their concentrations are comparatively low and their detection is more sensitive to sampling mode.

These observations validate the operando setup's ability to resolve gas evolution in direct correlation with electrochemical driving forces, highlighting hydrogen as a robust indicator of degradation onset and progression during charging. Further quantitative conversion of count–rate to volumetric gas yield will be presented in Section 6.4.

6.4 Data Discussion

In this section we interpret and compare the three measurement protocols in terms of time resolution, sensitivity and cycle-to-cycle behavior.

6.4.1 Cycle-to-Cycle Trends

Figure 6.8 shows a gradual decline of the H_2 peak height from 9 730 counts/s in the first window to 6 000 counts/s in the third window, thereafter stabilizing around 4 500 counts/s in later windows. Quantitatively, this corresponds to a decrease of approximately 38 percent from the first to the third window, and by the fifth window the H_2 peak has fallen by over 50 percent relative to the initial value. Beyond the seventh window the gas evolution rate remains near 4500 counts/s, less than half of the first-cycle peak.

These data demonstrate that the first cycle produces the greatest amount of hydrogen, and that each subsequent cycle yields progressively less gas. This progressive attenuation is consistent with the rapid formation of a passivating interphase on the anode surface during initial cycling, which suppresses further electrolyte decomposition and side-reaction pathways in later cycles.

Although the fixed-interval showed that the first cycle produces the most total gas, the full operando data (Figure 6.11) reveal exactly when within that cycle the gas appears. Two clear hydrogen peaks align with the voltage rise during charge. As soon as the cell switches to discharge and the voltage falls, the hydrogen signal drops back toward baseline. Ethylene follows the same pattern but with smaller peaks, while methane and carbon dioxide stay close to their baselines during discharge.

These results show that gas production is not uniform throughout the cycle but is concentrated in the charging step, especially during the initial activation and the steep voltage increase near the charge limit. Once discharge begins, gas evolution quickly subsides. This confirms that the main gas-forming reactions occur under anodic overpotential rather than during discharge.

In conclusion, the most gas is released during the charging portion of the first cycle, with very little generated during discharge or in later cycles. This highlights the importance of the initial charge step for electrolyte breakdown and gas evolution in lithium-ion pouch cells.

6.4.2 Temporal Resolution versus Sensitivity

Fixed-interval sampling yields the largest absolute gas volumes per window, but by integrating over 20 min every 2 h it blurs the fine structure of individual charge/discharge events. In contrast, continuous operando monitoring detects sharp, voltage-synchronous spikes of gas evolution (see Fig. 6.11), but the constant argon purge dilutes the transient species concentration and thus reduces the instantaneous signal-to-noise ratio for minor gases.

6.4.3 Voltage Correlation

All major hydrogen emission spikes align precisely with the onset of the charge plateau at 3.65 V (see Fig. 6.11). This temporal correlation strongly implicates over-potential–driven electrolyte decomposition as the dominant gas-generation mechanism rather than discharge-driven side reactions.

6.4.4 Method Recommendation

For mechanistic investigations that require pinpointing the exact timing of gas- formation events, continuous operando monitoring is clearly superior. For absolute quantification of total gas yields and cross-protocol comparison, bulk (end-of-experiment) sampling provides the highest sensitivity and simplest mass balance. The fixed-interval approach represents an intermediate compromise, offering both temporal context and enhanced signal by integrating over longer windows.

Chapter 7

Conclusion

This thesis presents the development, implementation, and validation of the fully automated operando gas-analysis system explicitly designed for lithium-ion pouch cells, that represents a significant advancement beyond the existing coin-cell–based setup. The system was designed to overcome limitations in existing experimental approaches, such as manual valve operation, poor temporal control of gas sampling, and insufficient sealing that can compromise the quality of gas analysis.

7.1 Major Technical Achievements and Findings

The platform's technical innovations are best illustrated by four core achievements.

First, a hermetic enclosure was engineered and produced out of stainless-steel with precision O-ring seals, yielding leak rates below 800 counts$\cdot$s^{-1} in continuous mass-spectrometry tests. Moreover, seamless electrochemical integration was realized through custom copper-rod feedthroughs, which preserved potentiostat connectivity without compromising enclosure integrity; stable cycling over ten charge–discharge sequences (75 h) confirmed both electrical and mechanical robustness.

Third, a microcontroller-driven solenoid valve network was implemented and controlled via a custom made Python-based GUI, enabling automated sampling triggers based on time intervals, voltage thresholds and cycle or rest triggers.

Finally, a systematic comparison of sampling methods, (fixed-interval draws, end-of-test bulk collection, and continuous operando monitoring) demonstrated that hydrogen evolution events reliably precede capacity fade, while detection of minor species (e.g., C_2H_4, CO_2) is strongly influenced by the chosen sampling strategy and its timing.

7.2 System Limitations and Integration Challenges

Despite its strengths, the current setup exhibits several constraints.

Throughput is limited by its single-cell configuration; although the control software is already designed to manage up to four parallel lines, additional solenoid valves are required to scale.

Residual dead volume, reduced by 65 % through shorter tubing, still attenuates rapid gas transients, indicating the need for internal enclosure volume miniaturization to improve temporal resolution.

Moreover, the lack of pressure measurement and control over pouch-cell compression precludes systematic studies of pressure-dependent behaviour; integrating force sensors and adjustable actuators would address this gap.

Finally, a systematic evaluation of the three sampling modalities—fixed-interval accumulation, end-of-test bulk collection, and continuous operando monitoring—yielded two principal conclusions. Notably, the initial fixed-interval window (every 2 hours) consistently produced the largest cumulative gas volume, indicating that gas evolution is most pronounced during the first cycle. Furthermore, continuous operando measurements unequivocally demonstrate that gas-evolution events are concentrated during the charging segments of each cycle.

7.3 Research Applications Enabled by Automated Sampling Strategies

The automation framework underpinning this platform unlocks a spectrum of experimental designs.

Fully operando analysis captures transient gas events in real time, offering uninterrupted insight into dynamic degradation processes.

Stepwise triggered sampling, based on user-defined time, voltage, or cycle-state criteria, permits targeted investigations of specific phases, for example, during the plateau phase of the cycling.

Furthermore, hybrid protocols that run multiple trigger types concurrently deliver both cumulative gas-yield data and high-resolution snapshots of fast events within a single experiment, eliminating the need for manual intervention.

7.4 Planned System Enhancements

To extend the platform's utility, three key enhancements are proposed.

First, a multiplexed sampling infrastructure will be realized by expanding the solenoid-valve network and retrofitting the existing coin-cell setup with the same automated valves, thereby enabling parallel analysis of multiple cell formats and boosting time efficiency.

Second, an oven-compatible redesign of the containment and sampling assemblies will facilitate controlled thermal ramp and hold experiments within a standard laboratory oven.

Third, integration of a pressure sensor or a regulated pressure apply system will permit active monitoring and control of applied compression, opening systematic studies into the interplay between mechanical stress and electrochemical performance or gas evolution.

7.5 Future Research Directions

Building on these enhancements, future studies can explore temperature-dependent kinetics by quantifying how varying oven conditions affect SEI growth rates and gas-generation pathways, thus informing thermal-management strategies.

Pressure-effect mapping will leverage the new compression control to correlate applied stress with interfacial impedance changes and gas-emission profiles, guiding mechanical design optimization.

Cross-format comparative analysis, running automated coin-cell and multiple pouch cell experiments in parallel, will directly compare degradation kinetics and gas-evolution signatures across cell geometries, maximizing throughput and time efficiency.

7.6 Final Reflections

This operando gas-analysis platform for pouch cells offers a combination of airtight integrity, automation, and flexible sampling control. By delivering synchronized gas and electrochemical measurements under controlled thermal and mechanical conditions, it establishes a comprehensive framework for advancing battery-degradation studies, safety assessments, and material innovation.

In conclusion, these capabilities set the stage for real-time diagnostic tools and data-driven strategies that will significantly enhance the research, development, and management of next-generation lithium-ion batteries.

Reference List

[1] D. et al., "Study on gas production characteristics of lithium iron phosphate battery module under high and low temperature co2," *2022 China International Conference on Electricity Distribution (CICED)*, pp. 151–155, 2022.

[2] C. D. M. et al., "Gas analysis by mass spectrometry," *ASM International*, vol. 10, pp. 144–151, 2019.

[3] J.-P. S. et al., "On the effect of cathode porosity in lithium-ion batteries," *Journal of The Electrochemical Society*, vol. 167, no. 6, p. 604509, 2020.

[4] N. T. M. Balakrishnan, A. Das, N. S. Jishnu, L. R. Raphael, J. D. Joyner, J.-H. Ahn, M. J. J. Fatima, and R. Prasanth, "The great history of lithium-ion batteries and an overview on energy storage devices," in *Electrospinning for Advanced Energy Storage Applications* (N. T. M. Balakrishnan and R. Prasanth, eds.), Materials Horizons: From Nature to Nanomaterials, pp. 1–20, Springer Nature Singapore, 2021.

[5] T. Kim, W. Song, D.-Y. Son, L. K. Ono, and Y. Qi, "Lithium-ion batteries: Outlook on present, future, and hybridized technologies," *Journal of Materials Chemistry A*, vol. 7, pp. 2942–2964, 2019.

[6] H. Sharma, S. Sharma, and P. K. Mishra, "A critical review of recent progress on lithium ion batteries: Challenges, applications, and future prospects," *Microchemical Journal*, vol. 212, p. 113494, 2025.

[7] G. Zubi, R. Dufo-López, M. Carvalho, and G. Pasaoglu, "The lithium-ion battery: State of the art and future perspectives," *Renewable and Sustainable Energy Reviews*, vol. 89, pp. 292–308, 2018.

[8] K. Wong and W. Chow, "Principle for the working of the lithium-ion battery," *Journal of Modern Physics*, vol. 11, pp. 1743–1750, 2020.

[9] C. A. R. Junior, E. R. Sanseverino, P. Gallo, M. M. Amaral, D. Koch, Y. Kotak, S. Diez, G. Walter, H.-G. Schweiger, and H. Zanin, "Exploring lithium-ion battery degradation: A concise review of mechanisms and mitigation strategies," *Batteries*, vol. 10, no. 7, p. 220, 2024.

[10] J. S. E. et al, "Lithium ion battery degradation: what you need to know," *Phys. Chem. Chem. Phys*, vol. 23, pp. 8200–8221, 2021.

[11] e. a. Y. Merla, B. Wu, "Novel application of differential thermal voltammetry as an in-depth state-of-health diagnosis method for lithium-ion batteries," *J. Power Sources*, vol. 307, p. 308–319, 2016.

[12] Y. Li, Y. Chen, Z. Liu, Z. Wang, and Y. Tang, “Lithium plating mechanism, detection, and mitigation in lithium-ion batteries: A review,” *Progress in Energy and Combustion Science*, vol. 85, p. 100896, 2021.

[13] Y. Zhang, J. Wang, L. Chen, and et al., “Onboard early detection and mitigation of lithium plating in fast-charging batteries,” *Nature Communications*, vol. 13, p. 5678, 2022.

[14] M. Poluektov *et al.*, “Revisiting mechanism of silicon degradation in li-ion batteries,” *The Journal of Physical Chemistry Letters*, vol. 16, no. 12, pp. 1234–1240, 2025.

[15] T. Vorauer *et al.*, “Impact of solid-electrolyte interphase reformation on capacity loss in silicon-based lithium-ion batteries,” *Communications Materials*, vol. 4, p. 44, 2023.

[16] B. S. Instruments, “Determination of the diffusion coefficient of an inserted species in a host electrode with eis, pitt and gitt techniques,” tech. rep., Application Note 70, June 2024.

[17] K. Liu, Y. Liu, D. Lin, A. Pei, and Y. Cui, “Materials for lithium-ion battery safety,” *Science Advances*, vol. 4, no. 6, p. eaas9820, 2018.

[18] Y. Chen *et al.*, “A review of lithium-ion battery safety concerns: The issues, strategies, and testing standards,” *Journal of Energy Chemistry*, vol. 59, pp. 83–99, Aug. 2021.

[19] F. A. Administration, “Evaluation of lithium battery thermal runaway vent gas analysis,” tech. rep., Technical Report TC22-12, 2022.

[20] W. Yao, M. Toubes-Rodrigo, and C. Chen, “Integration of mass spectrometry to characterize the gas evolution of batteries in thermal runaways using adiabatic calorimetry,” tech. rep., H.E.L Group Application Note, Mar. 2025.

[21] M. Dubarry *et al.*, “A comprehensive review on the characteristics and modelling of lithium-ion battery ageing,” tech. rep., TU Delft Repository, 2023.

[22] H. Wang, S. Wu, C. Shao, W. Luan, and H. Chen, “Thermal runaway and gas generation dynamics in aged lithium-ion batteries under low temperatures,” *Journal of Energy Storage*, vol. 124, p. 116852, 2025.

[23] T. Zheng, M. Muneeswara, H. Bao, J. Huang, L. Zhang, D. S. Hall, S. T. Boles, and W. Jin, “Gas evolution in li-ion rechargeable batteries: A review on operando sensing technologies, gassing mechanisms, and emerging trends,” *ChemElectroChem*, vol. 11, no. 2, p. e202400065, 2024.

[24] B. Mercier-Guyon, J.-F. Colin, I. Profatilova, and S. Martinet, “A quantitative and qualitative study of gas generation observed in lifepo4-tinb2o7 cells,” *Journal of Applied Electrochemistry*, vol. 54, p. 731–738, 2024.

[25] J. Scharf *et al.*, “Gas evolution in large-format automotive lithium-ion battery during formation,” tech. rep., DIVA Portal, 2023.

[26] Thermo Fisher Scientific, “Advanced SEM solutions for lithium battery research,” 2020. Accessed: February 26, 2025.

[27] S. Sasse, “Characterization of active material from lithium-ion batteries by xrd and sem,” Aug. 2024.

[28] S. Hildebrand, F. Ferrario, and N. Lebedeva, "Comparative overview of methods for the detection of airborne electrolyte components released from lithium-ion batteries," *Energy Technology*, vol. 12, no. 1, p. 2300647, 2024.

[29] D. Sturk, L. Rosell, P. Blomqvist, and A. A. Tidblad, "Analysis of Li-ion battery gases vented in an inert atmosphere thermal test chamber," *Batteries*, vol. 5, no. 3, p. 61, 2019.

[30] Y. Fernandes, A. Bry, and S. de Persis, "Identification and quantification of gases emitted during abuse tests by overcharge of a commercial Li-ion battery," *Journal of Power Sources*, vol. 389, pp. 106–119, 2018.

[31] E. J. K. Nilsson and A. A. Tidblad, "Gas emissions from lithium-ion batteries: A review of experimental results and methodologies," *Batteries*, vol. 10, no. 12, p. 443, 2024.

[32] R. Winkler *et al.*, "Mass spectrometric methods for monitoring redox processes in electrochemistry," *PMC*, vol. 14, pp. 27863–27889, Dec. 2013.

[33] J. Wang *et al.*, "Recent development and applications of differential electrochemical mass spectrometry in emerging energy conversion and storage solutions," *Chemical Society Reviews*, vol. 53, pp. 5199–5252, July 2024.

[34] A. B. Wong *et al.*, "Decoding the puzzle: recent breakthroughs in understanding degradation mechanisms of li-ion batteries," *Dalton Transactions*, vol. 52, pp. 16786–16817, Nov. 2023.

[35] T. F. Scientific, "Analysis of electrolyte components of lithium-ion batteries using gas chromatography-mass spectrometry," tech. rep., Application Note AN000394, 2023.

[36] Gotron Belgium, "Atmega328 nano ontwikkelbord," 2024. Accessed: 21-05-2025.

[37] RS Components, "Rs pro 24 v dc solenoid valve, 3/2 nc, g 1/8," 2024. Accessed: 21-05-2025.

[38] Gotron Belgium, "Transistor irfz44n n-ch 55v 49a 94w 0.0175r to220ab," 2024. Accessed: 23-05-2025.

[39] B. Instrument, *Installation and Operating Manual – Brooks 0152/0154 Microprocessor Control & Read-Out Unit*, 2002. Accessed: 21-05-2025.

[40] K. Xu, "Nonaqueous liquid electrolytes for lithium-based rechargeable batteries," *Chemical Reviews*, vol. 104, no. 10, pp. 4303–4418, 2004.

Appendix A

Appendix - Software Code in Python

main.py

```python
# main.py

import automaticLOGIC
import manualLOGIC
from InterfaceMAIN import InterfaceApp

if __name__ == "__main__":
    app = InterfaceApp()
    app.mainloop()

```

ECdata.py

```python
import pandas as pd
import os
def load_ec_data(filepath):

    if not os.path.isfile(filepath):
        print(f"[ERROR] Bestand niet gevonden: {filepath}")
        return None

    column_names = [
        "mode", "ox", "time/s", "Ewe/V", "I/mA", "cycle number"
    ]

    # Lees het bestand vanaf regel 95 met vaste kolomnamen
    df = pd.read_csv(filepath, sep=';', decimal=',', usecols=column_names)

    # Vervang decimale komma's → punten en zet types om
    df = df.map(lambda x: str(x).replace(',', '.'))

    # Conversie naar numerieke types

    df["Ewe/V"] = pd.to_numeric(df["Ewe/V"], errors='coerce')
    df["ox"] = pd.to_numeric(df["ox"], downcast="integer", errors='coerce')

    return df

```

inputs.py

```python
# input.py
# inputs.py
import serial
import json
import time

CONFIG_PATH = "config.json"

try:
    with open(CONFIG_PATH, "r") as f:
        CONFIG = json.load(f)
except FileNotFoundError:
    CONFIG = {"COM_PORT": "", "file_paths": {}}
```

```python

USER_PARAMETERS = {1: {}, 2: {}, 3: {}, 4: {}}
FILE_PATHS = CONFIG.get("file_paths", {})

arduino = None

def reconnect_arduino():
    global arduino
    try:
        if arduino and arduino.is_open:
            arduino.close()
            print(" Bestaande verbinding gesloten.")

        port = CONFIG["com_port"]
        arduino = serial.Serial(port, 9600, timeout=2)
        time.sleep(2)  # Arduino heeft opstarttijd nodig

        # Wacht op eerste bericht (zoals 'READY')
        if arduino.in_waiting:
            startup_msg = arduino.readline().decode("utf-8").strip()
            print(f" Arduino startup bericht: '{startup_msg}'")

        # Stuur PING
        arduino.write(b"PING\n")
        response = arduino.readline().decode("utf-8").strip()
        print(f" Arduino antwoord: '{response}'")

        if any(x in response.lower() for x in ["ack", "ready", "pong"]):
            print(f" Arduino verbonden op {port}")
            return True
        else:
            print(f" Geen geldig antwoord van Arduino op {port}")
            arduino.close()
            arduino = None
            return False

    except Exception as e:
        print(f" Fout bij verbinden met Arduino: {e}")
        arduino = None
        return False

def close_serial():
    global arduino
    if arduino and arduino.is_open:
        try:
            arduino.close()
            print(" Arduino verbinding succesvol gesloten.")
        except Exception as e:
            print(f" Fout bij sluiten van Arduino verbinding: {e}")

# Arduino pinout mapping (logische naam → pin)
VALVE_PINS = {
    "V0": 10,  # Purgeklep (L0)
    "V1-A": 2,  # Lijn 1 - A
    "V1-B": 3,
    "V2-A": 4,
    "V2-B": 5,
    "V3-A": 6,
    "V3-B": 7,
    "V4-A": 8,
    "V4-B": 9,
}

# Flowmeter pinnen
FLOWMETER_ANALOG_PIN = "A0"
FLOWMETER_PWM_PIN = 11  # optioneel

def set_flow_rate(line: int):
    rate = CONFIG["flow_rates"].get(str(line), 0)
    voltage = (rate / 30.0) * 5.0
    cmd = f"FLOW {line} {voltage:.2f}\\n"
    serial.write(cmd.encode())

# Arduino COM-poort (voor klepaansturing)
COM_PORT = "COM7"

# Datapad voor logging
DATA_FOLDER = "./data_logs/"

# Modus: 'automatisch' of 'manueel'
MODUS = "automatisch"

# Gebruikersparameters per lijn
USER_PARAMETERS = {
    1: {
        "sample": "",
        "purge_after": 0,
        "purge_for": 0,
        "flow_duration": 0,

        # Triggers
        "use_voltage_trigger": True,
        "trigger_voltage_step": 0.1,  # V

        "use_time_trigger": True,
        "trigger_time_interval": 30,  # s

        "use_halfcycle_trigger": True,  # op einde van laad/ontlaad

        "line_active": False

    },
    2: {
        "sample": "",
        "purge_after": 0,
        "purge_for": 0,
        "flow_duration": 0,

        # Triggers
        "use_voltage_trigger": True,
        "trigger_voltage_step": 0.1,  # V

        "use_time_trigger": True,
        "trigger_time_interval": 30,  # s

        "use_halfcycle_trigger": True,  # op einde van laad/ontlaad
```

```

            "line_active": False
        },
        3: {
            "sample": "",
            "purge_after": 0,
            "purge_for": 0,
            "flow_duration": 0,

            # Triggers
            "use_voltage_trigger": True,
            "trigger_voltage_step": 0.1,   # V

            "use_time_trigger": True,
            "trigger_time_interval": 30,   # s

            "use_halfcycle_trigger": True,  # op einde van laad/ontlaad

            "line_active": False
        },
        4: {
            "sample": "",
            "purge_after": 0,
            "purge_for": 0,
            "flow_duration": 0,

            # Triggers
            "use_voltage_trigger": True,
            "trigger_voltage_step": 0.1,   # V

            "use_time_trigger": True,
            "trigger_time_interval": 30,   # s

            "use_halfcycle_trigger": True,  # op einde van laad/ontlaad

            "line_active": False
        },

}
```

outputs.py

```python
# outputs.py
import serial
import inputs


# Klepstatussen
VALVE_STATUS = {
    "V0": "open",
    "V1-A": "closed",
    "V1-B": "closed",
    "V2-A": "closed",
    "V2-B": "closed",
    "V3-A": "closed",
    "V3-B": "closed",
    "V4-A": "closed",
    "V4-B": "closed"
}

def set_valve_state(valve_id, state: str):
    if valve_id not in VALVE_STATUS:
        print(f" Ongeldige klep-ID: '{valve_id}' (line_number klopt wel, maar klep-ID bestaat niet!)")
        return

    assert state in ["open", "closed", "inactive"]
    VALVE_STATUS[valve_id] = state

    if not inputs.arduino:
        print(f" Arduino niet beschikbaar, commando niet verzonden: {valve_id}:{state}")
        return

    if state in ["open", "closed"]:
        try:
            cmd = f"{valve_id}:{state}\n"
            inputs.arduino.write(cmd.encode('utf-8'))
            print(f" Commando verzonden: {valve_id}:{state}")
        except Exception as e:
            print(f" Fout bij schrijven naar Arduino: {e}")

    if valve_id is None:
        print(" Waarschuwing: valve_id is None (mogelijk fout in line_number)")


def get_valve_state(valve_id: str) -> str:
    return VALVE_STATUS.get(valve_id, "inactive")

def valve_is_open(valve_id: str) -> bool:
    return VALVE_STATUS.get(valve_id, "closed") == "open"

# Live meetdata per lijn
LIVE_DATA = {
    1: {"voltage": 0.0, "time": 0.0, "halfcycle": False, "flow": 0.0},
    2: {"voltage": 0.0, "time": 0.0, "halfcycle": False, "flow": 0.0},
    3: {"voltage": 0.0, "time": 0.0, "halfcycle": False, "flow": 0.0},
    4: {"voltage": 0.0, "time": 0.0, "halfcycle": False, "flow": 0.0},
}

def update_live_data(line: int, voltage=None, time=None, halfcycle=None, flow=None):
    if voltage is not None:
        LIVE_DATA[line]["voltage"] = voltage
    if time is not None:
        LIVE_DATA[line]["time"] = time
    if halfcycle is not None:
        LIVE_DATA[line]["halfcycle"] = halfcycle
    if flow is not None:
        LIVE_DATA[line]["flow"] = flow

# Laatste meetmoment per lijn (voor triggers)
LOG_STATE = {
    1: {"last_voltage": 0.0, "last_time": 0.0},
    2: {"last_voltage": 0.0, "last_time": 0.0},
    3: {"last_voltage": 0.0, "last_time": 0.0},
    4: {"last_voltage": 0.0, "last_time": 0.0},
}

# Excel logging bestanden per lijn (optioneel)
EXCEL_FILES = {
    1: None,
    2: None,
    3: None,
    4: None
}
```

manualLOGIC.py

```python
from outputs import VALVE_STATUS, set_valve_state

def toggle_valve_state(valve_id):
    current = VALVE_STATUS[valve_id]
    new_state = "closed" if current == "open" else "open"
    set_valve_state(valve_id, new_state)
```

automaticLOGIC.py

```python
import threading
from inputs import USER_PARAMETERS, FILE_PATHS, CONFIG
from outputs import set_valve_state
import pandas as pd
from ECdata import load_ec_data
import queue
import time
import os
import csv
from datetime import datetime, timedelta

measurement_lock = threading.Lock()
operation_queue = queue.Queue()
MONITOR_INTERVAL = 1  #sec
LOG_START_TIME = None

# Status per lijn
line_states = {
    1: {"active": False, "purged": False},
    2: {"active": False, "purged": False},
    3: {"active": False, "purged": False},
    4: {"active": False, "purged": False},
}

def start_automatic_process(line_number):
    """
    Wordt getriggerd vanuit de interface bij het drukken op 'Start' voor een lijn.
    """
    global LOG_START_TIME
    if LOG_START_TIME is None:
        LOG_START_TIME = datetime.now()
        print(f" Logging-timer gestart vanaf: {LOG_START_TIME.strftime('%H:%M:%S')}")

    if line_states[line_number]["active"]:
        print(f"Lijn {line_number} is al actief.")
        return

    print(f" Starten van automatische cyclus voor lijn {line_number}...")
    line_states[line_number]["active"] = True

    line_states[line_number]["purged"] = False

    try:
        purge_after = float(USER_PARAMETERS[line_number]["purge_after"])
    except ValueError:
        print(f" Ongeldige waarde voor purge_after bij lijn {line_number}")
        return

    threading.Timer(purge_after * 60, lambda: _do_purge(line_number)).start()

def _do_purge(line_number):
    def task():
        if not line_states[line_number]["active"]:
            print(f" Purgen geannuleerd: lijn {line_number} is gestopt.")
            return
        with measurement_lock:
            if not line_states[line_number]["active"]:
                return

            print(f" Purge gestart voor lijn {line_number}")
            _close_valve("V0")
            _open_valves(line_number)

            try:
                purge_for = float(USER_PARAMETERS[line_number]["purge_for"])
            except ValueError:
                print(f" Ongeldige waarde voor purge_for bij lijn {line_number}")
                return

            time.sleep(purge_for * 60)

            print(f" Purge afgerond voor lijn {line_number}")
            _close_valves(line_number)
            _open_valve("V0")
            line_states[line_number]["purged"] = True

            # triggers starten
            params = USER_PARAMETERS[line_number]
            if params.get("use_time_trigger", False):
                _schedule_next_measurement(line_number)
            if params.get("use_voltage_trigger", False):
                start_voltage_monitoring(line_number)
            if params.get("use_halfcycle_trigger", False):
                start_halfcycle_monitoring(line_number)
            if params.get("use_cycle_trigger", False):
                start_cycle_monitoring(line_number)
            if params.get("use_rest_trigger", False):
                start_rest_monitoring(line_number)

        # Na purge: verwerk eerstvolgende wachtrijtaak
        threading.Timer(10, _process_next_queue_item).start()

    # Prioriteit 0 = hoogste prioriteit
    _queue_or_execute(priority=0, task=task)

def _end_purge(line_number):
    if not line_states[line_number]["active"]:
        return

    print(f" Purge afgerond voor lijn {line_number}")
    _close_valves(line_number)
    _open_valve("V0")
    line_states[line_number]["purged"] = True

    params = USER_PARAMETERS[line_number]

    if params.get("use_time_trigger", False):
        _schedule_next_measurement(line_number)

    if params.get("use_voltage_trigger", False):
        start_voltage_monitoring(line_number)

    if params.get("use_halfcycle_trigger", False):
        start_halfcycle_monitoring(line_number)

    if params.get("use_cycle_trigger", False):
        start_cycle_monitoring(line_number)

    if params.get("use_rest_trigger", False):
        start_rest_monitoring(line_number)


def _schedule_next_measurement(line_number):
    if not line_states[line_number]["active"]:
        return

    try:
        interval = float(USER_PARAMETERS[line_number]["trigger_time_interval"])
    except ValueError:
        print(f" Ongeldige waarde voor trigger_time_interval bij lijn {line_number}")
        return

    def delayed_measurement():
        try:
            df = load_ec_data(FILE_PATHS[str(line_number)])
            last_row = df.iloc[-1]
        except Exception as e:
            print(f" Fout bij inlezen EC-data voor tijdtrigger: {e}")
            last_row = None

        _perform_measurement(line_number, trigger_source="time", last_row=last_row)

        #  Herplan opnieuw!
        _schedule_next_measurement(line_number)

    threading.Timer(interval, delayed_measurement).start()

def _perform_measurement(line_number, trigger_source="unknown", last_row=None):
    def task(trigger_source=trigger_source, last_row=last_row):
        if not line_states[line_number]["active"]:
            print(f" Meting geannuleerd: lijn {line_number} is gestopt.")
            return

        with measurement_lock:
            print(f" Meting gestart voor lijn {line_number}")
            #set_flow_rate(line_number, arduino)
            _close_valve("V0")
            _open_valves(line_number)
            log_measurement(line_number, trigger_source, last_row)

            try:
                duration = float(USER_PARAMETERS[line_number]["flow_duration"]) * 60
            except ValueError:
                print(f" Ongeldige waarde voor flow_duration bij lijn {line_number}")
                return

            time.sleep(duration)

            print(f" Meting afgerond voor lijn {line_number}")
            _close_valves(line_number)
            _open_valve("V0")


        # Na afronding → verwerk volgende taak
        threading.Timer(10, _process_next_queue_item).start()

    # Prioriteit 10 = laag, want meting
    _queue_or_execute(priority=10, task=task)


def _end_measurement(line_number):
    if not line_states[line_number]["active"]:
        return

    print(f" Meting afgerond voor lijn {line_number}")
    _close_valves(line_number)
    _open_valve("V0")

    #  Herplan enkel bij tijdtrigger
    if USER_PARAMETERS[line_number].get("use_time_trigger", False):
        _schedule_next_measurement(line_number)

def _open_valves(line_number):
    if not line_states[line_number]["active"]:
        print(f" Poging tot openen van kleppen voor gestopte lijn {line_number}
        ↪  genegeerd.")
        return
    set_valve_state(f"V{line_number}-A", "open")
    set_valve_state(f"V{line_number}-B", "open")

def _close_valves(line_number):
    set_valve_state(f"V{line_number}-A", "closed")
    set_valve_state(f"V{line_number}-B", "closed")

def _open_valve(valve_id):
    set_valve_state(valve_id, "open")

def _close_valve(valve_id):
```

```python
    set_valve_state(valve_id, "closed")

def stop_line(line_number):
    print(f" Lijn {line_number} gestopt.")
    line_states[line_number]["active"] = False
    _close_valves(line_number)
    _open_valve("V0")
def stop_automatic_process(line_number):
    last_voltage_trigger.pop(line_number, None)
    last_halfcycle.pop(line_number, None)
    last_cycle.pop(line_number, None)

    if not line_states[line_number]["active"]:
        print(f"Lijn {line_number} is al gestopt.")
        return

    print(f" Stoppen van lijn {line_number}...")
    line_states[line_number]["active"] = False

    # Sluit kleppen
    _close_valves(line_number)
    _close_valve("V0")

    print(f" Lijn {line_number} volledig gestopt.")

# Voor voltage-trigger
last_voltage_trigger = {}

def start_voltage_monitoring(line_number, trigger_source="voltage", last_row=None):
    filepath = FILE_PATHS[str(line_number)]
    voltage_step = float(USER_PARAMETERS[line_number]["trigger_voltage_step"])
    min_v = float(USER_PARAMETERS[line_number].get("trigger_voltage_min", 0.0))
    max_v = float(USER_PARAMETERS[line_number].get("trigger_voltage_max", 10.0))
    print(f" Voltage-trigger monitoring gestart voor lijn {line_number}...")

    #  Initieer met de eerste beschikbare spanning als startreferentie
    try:
        df_init = load_ec_data(filepath)
        first_voltage = float(df_init.iloc[0]["Ewe/V"])
        last_voltage_trigger[line_number] = first_voltage
        print(f"[Lijn {line_number}] Initiele spanning ingesteld op
         ↪  {first_voltage:.3f} V")
    except Exception as e:
        print(f" Fout bij initialiseren van startspanning: {e}")
        return  # Stop als we geen initwaarde kunnen zetten

    def monitor():
        if not line_states[line_number]["active"]:
            return

        try:
            df = load_ec_data(filepath)
            last_row = df.iloc[-1]
            voltage = float(last_row["Ewe/V"])
            print(f"[Lijn {line_number}] Laatste rij: {last_row.to_dict()}")
        except Exception as e:
            print(f" Fout bij lezen voltage-trigger bestand: {e}")
            threading.Timer(MONITOR_INTERVAL, monitor).start()
            return

        prev = last_voltage_trigger.get(line_number)
        delta = abs(voltage - prev)

        if delta >= voltage_step:

            if min_v <= voltage <= max_v:
                print(
                    f" Voltage-trigger geactiveerd bij V={delta:.3f} V binnen limiet
                    ↪  ({min_v}{{max_v}) (lijn {line_number})")
                _perform_measurement(line_number, trigger_source="voltage",
                ↪  last_row=last_row)
                last_voltage_trigger[line_number] = voltage  # Enkel updaten na
                ↪  meting
            else:
                print(
                    f" V voldoende ({delta:.3f} V), maar voltage buiten limieten:
                    ↪  {voltage:.2f} V (lijn {line_number})")

        threading.Timer(MONITOR_INTERVAL, monitor).start()

    monitor()

# Voor half-cycle-trigger
last_halfcycle = {}
def start_halfcycle_monitoring(line_number):
    filepath = FILE_PATHS[str(line_number)]
    print(f" Half-cycle monitoring gestart voor lijn {line_number}...")

    def monitor():
        if not line_states[line_number]["active"]:
            return

        try:
            df = load_ec_data(filepath)
            hc = int(df["ox"].iloc[-1])
            last_row = df.iloc[-1]
            print(f"[Lijn {line_number}] ox = {hc}")
        except:
            threading.Timer(MONITOR_INTERVAL, monitor).start()
            return

        prev = last_halfcycle.get(line_number)
        if prev is None:
            last_halfcycle[line_number] = hc
            print(f"[Lijn {line_number}] Initiele ox opgeslagen: {hc}")
            threading.Timer(MONITOR_INTERVAL, monitor).start()
            return

        if hc != prev:
            last_halfcycle[line_number] = hc
            print(f" Half-cycle trigger geactiveerd bij ox-overgang {prev} → {hc}
            ↪  (lijn {line_number})")
            _perform_measurement(line_number, trigger_source="half cycle",
            ↪  last_row=last_row)

        threading.Timer(MONITOR_INTERVAL, monitor).start()

    monitor()

# Voor cycle-trigger
last_cycle = {}

def start_cycle_monitoring(line_number, trigger_source="voltage", last_row=None):
    filepath = FILE_PATHS[str(line_number)]
    print(f" Cycle-trigger monitoring gestart voor lijn {line_number}...")

    first_after = int(USER_PARAMETERS[line_number].get("trigger_cycle_first_after",
    ↪  0))
    then_every = int(USER_PARAMETERS[line_number].get("trigger_cycle_then_every", 1))

    def monitor():
        if not line_states[line_number]["active"]:
            return

        try:
            df = load_ec_data(filepath)
            current_cycle = int(float(df["cycle number"].iloc[-1]))
            print(f"[Lijn {line_number}] Cycle: {current_cycle}")
        except:
            threading.Timer(MONITOR_INTERVAL, monitor).start()
            return

        prev_cycle = last_cycle.get(line_number, -1)
        if current_cycle != prev_cycle:
            last_cycle[line_number] = current_cycle

            if current_cycle >= first_after and (current_cycle - first_after) %
            ↪  then_every == 0:
                print(f" Cycle-trigger geactiveerd bij cycle {current_cycle} (lijn
                ↪  {line_number})")
                try:
                    last_row = df.iloc[-1]
                except:
                    last_row = None

                _perform_measurement(line_number, trigger_source="full cycle",
                ↪  last_row=last_row)

        threading.Timer(MONITOR_INTERVAL, monitor).start()

    monitor()
#rest trigger
last_mode = {}

def start_rest_monitoring(line_number):
    filepath = FILE_PATHS[str(line_number)]
    print(f" Rest-trigger monitoring gestart voor lijn {line_number}...")

    def monitor():
        if not line_states[line_number]["active"]:
            return

        try:
            df = load_ec_data(filepath)
            mode = int(df["mode"].iloc[-1])
            last_row = df.iloc[-1]
            print(f"[Lijn {line_number}] mode = {mode}")
        except:
            threading.Timer(MONITOR_INTERVAL, monitor).start()
            return

        prev_mode = last_mode.get(line_number)
        if prev_mode is None:
            last_mode[line_number] = mode
            threading.Timer(MONITOR_INTERVAL, monitor).start()
            return

        if mode == 3 and prev_mode != 3:
            print(f" Rest-trigger geactiveerd: mode 3 (lijn {line_number})")
            _perform_measurement(line_number, trigger_source="rest",
            ↪  last_row=last_row)

        last_mode[line_number] = mode
        threading.Timer(MONITOR_INTERVAL, monitor).start()

    monitor()


def _process_next_queue_item():
    if not operation_queue.empty() and not measurement_lock.locked():
        item = operation_queue.get()
        if isinstance(item, tuple) and callable(item[1]):
            _, task = item
            threading.Thread(target=task).start()
        else:
            print(" Fout: ongeldige taak in wachtrij:", item)

def _queue_or_execute(priority, task):
    if measurement_lock.locked():
        operation_queue.put((priority, task))
        print(f" Taak met prioriteit {priority} toegevoegd aan wachtrij.")
    else:
        threading.Thread(target=task).start()

def log_measurement(line_number, trigger=None, last_row=None):
    if last_row is None:
        print(f" Meting niet gelogd: geen last_row beschikbaar (lijn {line_number},
        ↪  trigger: {trigger})")
        return

    # Als het een pandas.Series is, omzetten
    if hasattr(last_row, "to_dict"):
        last_row = last_row.to_dict()

    # Als het nog steeds geen geldig dict is: skippen
    if not isinstance(last_row, dict) or not last_row.get("Ewe/V"):
        print(f" Meting niet gelogd: last_row onvolledig of ongeldig (lijn
        ↪  {line_number})")
        return

    # Logbestand genereren per dag
    date_str = datetime.now().strftime("%Y-%m-%d")
    logfile = os.path.join(os.path.dirname(__file__), f"log_automatic_{date_str}.csv")

    now = datetime.now()
    if LOG_START_TIME is not None:
        elapsed = now - LOG_START_TIME
        elapsed_str = str(timedelta(seconds=int(elapsed.total_seconds())))
    else:
        elapsed_str = ""

    base = {
        "timestamp": now.strftime("%Y-%m-%d %H:%M:%S"),
        "elapsed": elapsed_str,
        "line": line_number,
        "trigger": trigger if trigger else "",
    }

    row = {**base, **last_row}

    file_exists = os.path.isfile(logfile)
    with open(logfile, "a", newline="") as f:
        writer = csv.DictWriter(f, fieldnames=row.keys())
        if not file_exists or os.path.getsize(logfile) == 0:
            writer.writeheader()
        writer.writerow(row)

def set_flow_rate(line_number, serial_port):
    try:
        rate = float(CONFIG.get("flow_rates", {}).get(str(line_number), 0.0))
        voltage = (rate / 30.0) * 5.0
        serial_port.write(f"FLOW {line_number} {voltage:.2f}\\n".encode())
        print(f" Flow rate voor lijn {line_number}: {rate} ml/min → {voltage:.2f}V")
    except Exception as e:
        print(f" Flow rate instellen mislukt voor lijn {line_number}: {e}")

```

InterfaceMAIN.py

```
import customtkinter as ctk
from automaticUI import LineFrame
from manualUI import build_manual_tab
from visualUI import VisualUI
from tkinter import messagebox
from settings_popup import SettingsWindow


ctk.set_appearance_mode("dark")
ctk.set_default_color_theme("blue")


class InterfaceApp(ctk.CTk):


    def __init__(self):
        super().__init__()
        self.geometry("1300x600")
        self.title("Mass Spec Controller")
        self._build_tabs()
```

```python
        self._last_tab = self.tab_view.get()
        self._poll_tab_change()  # start check op tab-wissels

        settings_btn = ctk.CTkButton(self, text=" Settings", command=self._open_settings_window, width=80)
        settings_btn.place(relx=1.0, rely=0.0, anchor="ne", x=-20, y=20)  # rechtsboven

    def _build_tabs(self):
        self.tab_view = ctk.CTkTabview(self, width=950, height=750)
        self.tab_view.pack(padx=10, pady=10)

        self.auto_tab = self.tab_view.add("Automatic")
        self.manual_tab = self.tab_view.add("Manual")
        self.visual_tab = self.tab_view.add("Visualisation")

        self._build_auto_tab()
        build_manual_tab(self.manual_tab)
        self._build_visual_tab()

    def _build_auto_tab(self):
        scroll_frame = ctk.CTkScrollableFrame(self.auto_tab, width=1200, height=700)
        scroll_frame.grid(row=0, column=0, padx=10, pady=10, sticky="nsew")

        for i in range(1, 5):
            lf = LineFrame(scroll_frame, line_number=i)
            row = (i - 1) // 2
            col = (i - 1) % 2
            lf.grid(row=row, column=col, padx=10, pady=10, sticky="nsew")

        scroll_frame.grid_columnconfigure((0, 1), weight=1)
        scroll_frame.grid_rowconfigure((0, 1), weight=1)

    def _build_visual_tab(self):
        visual_ui = VisualUI(self.visual_tab)
        visual_ui.pack(fill="both", expand=True, padx=20, pady=20)

    def _open_settings_window(self):
        SettingsWindow(self)

    def _poll_tab_change(self):
        current_tab = self.tab_view.get()
        if current_tab != self._last_tab:

            # Als we naar Manual gaan: waarschuwing
            if current_tab == "Manual":
                response = messagebox.askyesno(
                    title=" Manual Control",
                    message="Manual control can override automated operations.\nAre you sure you want to continue?"
                )
                if not response:
                    self.tab_view.set("Automatic")


            elif current_tab == "Visualisation":
                self._refresh_visual_tab()

            self._last_tab = self.tab_view.get()

        self.after(200, self._poll_tab_change)

    def _refresh_visual_tab(self):
        for widget in self.visual_tab.winfo_children():
            widget.destroy()
        self._build_visual_tab()


if __name__ == "__main__":
    app = InterfaceApp()
    app.mainloop()
```

manualUI.py

```
import customtkinter as ctk
from manualLOGIC import toggle_valve_state
from outputs import VALVE_STATUS

# Voeg alle lijnen toe
VALVES = [
    "V0",
    "V1-A", "V1-B",
    "V2-A", "V2-B",
    "V3-A", "V3-B",
    "V4-A", "V4-B"
]

def build_manual_tab(tab):
    positions = {
        "V0": (0, 0),
        "V1-A": (0, 1), "V1-B": (1, 1),
        "V2-A": (0, 2), "V2-B": (1, 2),
        "V3-A": (0, 3), "V3-B": (1, 3),
        "V4-A": (0, 4), "V4-B": (1, 4),
    }

    widgets = {}

    for valve in VALVES:
        row, col = positions[valve]
        color = "green" if VALVE_STATUS[valve] == "open" else "red"

        btn = ctk.CTkButton(tab,
                            text=valve,
                            fg_color=color,
                            width=80,
                            height=50,
                            corner_radius=5,
                            command=lambda v=valve: update_valve(v, widgets))
        btn.grid(row=row, column=col, padx=20, pady=20)
        widgets[valve] = btn

def update_valve(valve_id, widgets):
    toggle_valve_state(valve_id)
    new_color = "green" if VALVE_STATUS[valve_id] == "open" else "red"
```

```python
    widgets[valve_id].configure(fg_color=new_color)
```

automaticUI.py

```python
import customtkinter as ctk
from tkinter import BooleanVar
from outputs import valve_is_open
from inputs import USER_PARAMETERS
from automaticLOGIC import start_automatic_process, stop_automatic_process  # <- hier toegevoegd

class LineFrame(ctk.CTkFrame):
    def __init__(self, master, line_number):
        super().__init__(master)
        self.line_number = line_number
        self.active_var = BooleanVar(value=False)  # standaard uit
        self.configure(fg_color="#2b2b2b", corner_radius=10)
        self._build_widgets()
        self._start_valve_polling()
        self._toggle()  # <- voeg dit toe

    def _start_valve_polling(self):
        self._update_valve_visuals()
        self.after(1000, self._start_valve_polling)

    def _update_valve_visuals(self):
        va = valve_is_open(f"V{self.line_number}-A")
        vb = valve_is_open(f"V{self.line_number}-B")
        self.valve_a.configure(fg_color="green" if va else "red")
        self.valve_b.configure(fg_color="green" if vb else "red")

    def _toggle_entry(self, bool_var, entry_widget, label_widget):
        if bool_var.get():
            label_widget.grid()
            entry_widget.grid()
        else:
            label_widget.grid_remove()
            entry_widget.grid_remove()

    def _build_widgets(self):
        self.columnconfigure((0, 1, 2, 3), weight=1)

        toggle_frame = ctk.CTkFrame(self)
        toggle_frame.grid(row=0, column=0, columnspan=4, sticky="w", padx=10, pady=5)
        self.toggle = ctk.CTkSwitch(toggle_frame, text=f"Activate Line {self.line_number}",
                                    variable=self.active_var, command=self._toggle)
        self.toggle.pack(anchor="w")

        ctk.CTkLabel(self, text=f"Line {self.line_number}", font=("Arial", 18)).grid(row=1, column=0, columnspan=4, pady=5)

        self.entries = {}
        fields = ["Sample", "Purge after (min)", "For (min)", "Flow duration (min)"]
        for i, field in enumerate(fields):
            ctk.CTkLabel(self, text=field + ":").grid(row=2+i, column=0, sticky="w", padx=10, pady=5)
            self.entries[field] = ctk.CTkEntry(self, width=120)
            self.entries[field].grid(row=2+i, column=1, sticky="w", pady=5)

        self.time_trigger = BooleanVar()
        self.voltage_trigger = BooleanVar()
        self.halfcycle_trigger = BooleanVar()
        self.cycle_trigger = BooleanVar()
        self.rest_trigger = BooleanVar()

        trigger_frame = ctk.CTkFrame(self)
        trigger_frame.grid(row=8, column=0, columnspan=6, padx=10, pady=10,
         ↪  sticky="ew")
        trigger_frame.grid_columnconfigure((0, 1, 2), weight=1)

        ctk.CTkLabel(trigger_frame, text="Trigger Options", font=("Arial",
         ↪  14)).grid(row=0, column=0, columnspan=3,
                                                                                      ↪  sticky="w",
                                                                                      ↪  padx=10,
                                                                                      ↪  pady=(5,
                                                                                      ↪  10))

        # Time trigger
        self.trigger_time_entry = ctk.CTkEntry(trigger_frame, width=100)
        self.trigger_time_entry.insert(0, "30")
        self.time_label = ctk.CTkLabel(trigger_frame, text="Interval (s):")

        time_cb = ctk.CTkCheckBox(trigger_frame, text="Enable time trigger",
         ↪  variable=self.time_trigger,
                                  command=lambda:
                                  ↪  self._toggle_entry(self.time_trigger,
                                                                      ↪  self.trigger_time_entry,
                                                                     self.time_label))
        time_cb.grid(row=1, column=0, sticky="w", padx=10, pady=5)
        self.time_label.grid(row=1, column=1, padx=10, pady=5)
        self.trigger_time_entry.grid(row=1, column=2, pady=5)
        self.time_label.grid_remove()
        self.trigger_time_entry.grid_remove()

        # Voltage trigger
        self.trigger_voltage_entry = ctk.CTkEntry(trigger_frame, width=100)
        self.trigger_voltage_entry.insert(0, "0.2")
        self.voltage_label = ctk.CTkLabel(trigger_frame, text="Voltage (V):")
        self.min_voltage_entry = ctk.CTkEntry(trigger_frame, width=100)
        self.min_voltage_entry.insert(0, "2.0")
        self.min_voltage_label = ctk.CTkLabel(trigger_frame, text="Min. voltage (V):")

        self.max_voltage_entry = ctk.CTkEntry(trigger_frame, width=100)
        self.max_voltage_entry.insert(0, "4.0")
        self.max_voltage_label = ctk.CTkLabel(trigger_frame, text="Max. voltage (V):")
        volt_cb = ctk.CTkCheckBox(trigger_frame, text="Enable voltage trigger",
         ↪  variable=self.voltage_trigger,
                                  command=lambda: [
                                      self._toggle_entry(self.voltage_trigger,

                                                                      ↪  self.trigger_voltage_entry,
                                                                      ↪  self.voltage_label),
                                       ↪  self._toggle_entry(self.voltage_trigger,self.min_voltage_e
                                                                      ↪  self.min_voltage_label),
                                       ↪  self._toggle_entry(self.voltage_trigger,self.max_voltage_e
                                                                      ↪  self.max_voltage_label)])
        volt_cb.grid(row=2, column=0, sticky="w", padx=10, pady=5)
        self.voltage_label.grid(row=2, column=1, padx=10, pady=5)
        self.trigger_voltage_entry.grid(row=2, column=2, pady=5)
        self.min_voltage_label.grid(row=3, column=1, padx=10, pady=5)
        self.min_voltage_entry.grid(row=3, column=2, pady=5)
        self.max_voltage_label.grid(row=4, column=1, padx=10, pady=5)
        self.max_voltage_entry.grid(row=4, column=2, pady=5)
        self.voltage_label.grid_remove()
        self.trigger_voltage_entry.grid_remove()
        self.min_voltage_label.grid_remove()
        self.min_voltage_entry.grid_remove()
        self.max_voltage_label.grid_remove()
        self.max_voltage_entry.grid_remove()

        # Half-cycle (geen inputveld)
        ctk.CTkCheckBox(trigger_frame, text="Enable half-cycle trigger",
        ↪  variable=self.halfcycle_trigger).grid(row=3,




        # Cycle trigger
        self.trigger_first_cycle_entry = ctk.CTkEntry(trigger_frame, width=100)
        self.trigger_first_cycle_entry.insert(0, "2")
        self.cycle_first_label = ctk.CTkLabel(trigger_frame, text="First after:")

        self.trigger_every_cycle_entry = ctk.CTkEntry(trigger_frame, width=100)
        self.trigger_every_cycle_entry.insert(0, "10")
        self.cycle_every_label = ctk.CTkLabel(trigger_frame, text="Then every:")

        cycle_cb = ctk.CTkCheckBox(trigger_frame, text="Enable cycle trigger",
        ↪  variable=self.cycle_trigger,
                                   command=lambda: [
                                       self._toggle_entry(self.cycle_trigger,
                                       ↪  self.trigger_first_cycle_entry,
                                                          self.cycle_first_label),
                                       self._toggle_entry(self.cycle_trigger,
                                       ↪  self.trigger_every_cycle_entry,
                                                          self.cycle_every_label)])
        cycle_cb.grid(row=4, column=0, sticky="w", padx=10, pady=5)

        self.cycle_first_label.grid(row=5, column=1, padx=10, pady=5)
        self.trigger_first_cycle_entry.grid(row=5, column=2, pady=5)

        self.cycle_every_label.grid(row=6, column=1, padx=10, pady=5)
        self.trigger_every_cycle_entry.grid(row=6, column=2, pady=5)

        self.cycle_first_label.grid_remove()
        self.trigger_first_cycle_entry.grid_remove()
        self.cycle_every_label.grid_remove()
        self.trigger_every_cycle_entry.grid_remove()

        rest_cb = ctk.CTkCheckBox(trigger_frame, text="Enable rest trigger",
        ↪  variable=self.rest_trigger)
        rest_cb.grid(row=5, column=0, sticky="w", padx=10, pady=5)

        status_frame = ctk.CTkFrame(self)
        status_frame.grid(row=2, column=2, rowspan=3, padx=10, pady=5)

        self.led = ctk.CTkLabel(status_frame, text="", font=("Arial", 32),
        ↪  text_color="red")
        self.led.pack(pady=5)

        self.valve_a = ctk.CTkLabel(status_frame, text=f"V{self.line_number}-A",
        ↪  width=60, height=30, fg_color="red", corner_radius=6)
        self.valve_b = ctk.CTkLabel(status_frame, text=f"V{self.line_number}-B",
        ↪  width=60, height=30, fg_color="red", corner_radius=6)
        self.valve_a.pack(pady=2)
        self.valve_b.pack(pady=2)

        self.warning = ctk.CTkLabel(self, text=" Please fill in all required fields.",
        ↪  text_color="red")
        self.warning.grid(row=11, column=0, columnspan=2, pady=5)
        self.warning.grid_remove()

        # Start en Stop knoppen
        ctk.CTkButton(self, text="Start Line", command=self._start).grid(row=12,
        ↪  column=0, columnspan=4, pady=10,
                                      sticky="ew",
                                      ↪  padx=150)
        ctk.CTkButton(self, text="End Line", command=self._stop,
        ↪  fg_color="red").grid(row=13, column=0, columnspan=4,

                                             ↪  pady=5,
                                             ↪  sticky="ew
                                             ↪  padx=150)

    def _toggle(self):
        state = "normal" if self.active_var.get() else "disabled"
        color = "#2b2b2b" if self.active_var.get() else "#3a3a3a"
        self.configure(fg_color=color)
        for widget in self.winfo_children():
            if widget not in [self.toggle.master, self.toggle]:
                try:
                    widget.configure(state=state)
                except:
                    pass

        USER_PARAMETERS[self.line_number]["line_active"] = self.active_var.get()

    def _start(self):
        # Eerst triggers ophalen
        use_time = self.time_trigger.get()
        use_voltage = self.voltage_trigger.get()
        use_halfcycle = self.halfcycle_trigger.get()
        use_cycle = self.cycle_trigger.get()
        use_rest = self.rest_trigger.get()

        # Controleer of minstens één trigger aangevinkt is
        if not (use_time or use_voltage or use_halfcycle or use_cycle or use_rest):
            self.warning.configure(text=" Please select at least one trigger.")
            self.warning.grid()
            self.led.configure(text_color="red")
            return

        # Bepaal dynamisch welke velden verplicht zijn
        required_fields = ["Sample", "Purge after (min)", "For (min)", "Flow duration
        ↪  (min)"]

        # Controleer of al die velden effectief ingevuld zijn
        all_filled = all(self.entries[field].get() for field in required_fields)
        if not all_filled:
            self.warning.configure(text=" Please fill in all required fields.")
            self.warning.grid()
            self.led.configure(text_color="red")
            return

        # Alles is OK → parameters opslaan en lijn starten
        self.warning.grid_remove()
        self.led.configure(text_color="green")

        params = USER_PARAMETERS[self.line_number]
        params["sample"] = self.entries["Sample"].get()
        params["purge_after"] = float(self.entries["Purge after (min)"].get())
        params["purge_for"] = float(self.entries["For (min)"].get())
        params["flow_duration"] = float(self.entries["Flow duration (min)"].get())
        params["use_rest_trigger"] = use_rest

        # Enkel als gebruikt → ophalen
        if use_time:
            params["trigger_time_interval"] = float(self.trigger_time_entry.get())
        if use_voltage:
            params["trigger_voltage_step"] = float(self.trigger_voltage_entry.get())
            params["trigger_voltage_min"] = float(self.min_voltage_entry.get())
            params["trigger_voltage_max"] = float(self.max_voltage_entry.get())

        params["use_time_trigger"] = use_time
        params["use_voltage_trigger"] = use_voltage
        params["use_halfcycle_trigger"] = use_halfcycle
        params["use_cycle_trigger"] = use_cycle
        if use_cycle:
            params["trigger_cycle_first_after"] =
            ↪  int(self.trigger_first_cycle_entry.get())
            params["trigger_cycle_then_every"] =
            ↪  int(self.trigger_every_cycle_entry.get())

        start_automatic_process(self.line_number)

    def _stop(self):
        # Stop de automatische logica
        stop_automatic_process(self.line_number)

        # Zet LED terug op rood
        self.led.configure(text_color="red")

        # Reset alle parameters voor deze lijn
        USER_PARAMETERS[self.line_number] = {}

        # Wis alle invoervelden
        for entry in self.entries.values():
            entry.delete(0, "end")

        # Zet alle checkboxes uit
        self.time_trigger.set(False)
        self.voltage_trigger.set(False)
        self.halfcycle_trigger.set(False)
        self.cycle_trigger.set(False)

        # Eventuele waarschuwingen verbergen
        self.warning.grid_remove()

```

visualUI.py

```
import customtkinter as ctk
from outputs import VALVE_STATUS
from inputs import USER_PARAMETERS

class VisualUI(ctk.CTkFrame):
    def __init__(self, master):
        super().__init__(master)
        self.configure(fg_color="transparent")
        self._build_layout()
        self._update_colors()

    def _build_layout(self):
        self.grid_columnconfigure((0, 1, 2, 3, 4), weight=1)
        self.grid_rowconfigure((0, 1), weight=1)

        self.valves = {}

        # Bypass altijd tonen
        self.valves["V0"] = self._make_valve(0, 0, "V0")

        # Dynamisch toevoegen per actieve lijn
        col_offset = 1
        for line in range(1, 5):
            if USER_PARAMETERS[line].get("line_active", False):
                self.valves[f"V{line}-A"] = self._make_valve(0, col_offset,
                ↪  f"V{line}-A")
                self.valves[f"V{line}-B"] = self._make_valve(1, col_offset,
                ↪  f"V{line}-B")
                col_offset += 1

    def _make_valve(self, row, column, name):
        valve = ctk.CTkLabel(
            self,
            text=name,
            width=80,
            height=40,
            corner_radius=6,
            fg_color="grey",
            text_color="white"
        )
        valve.grid(row=row, column=column, padx=30, pady=30)
        return valve

    def _update_colors(self):
        for valve, widget in self.valves.items():
            state = VALVE_STATUS.get(valve, "inactive")
            if state == "open":
                widget.configure(fg_color="green")
            elif state == "closed":
                widget.configure(fg_color="red")
            else:
                widget.configure(fg_color="grey")

        # Herhaal elke seconde
        self.after(1000, self._update_colors)
```

settings$_{p}$*opup.py*

```
import customtkinter as ctk
import serial.tools.list_ports
import json
from inputs import CONFIG, reconnect_arduino, close_serial

class SettingsWindow(ctk.CTkToplevel):
    def __init__(self, master=None):
        super().__init__(master)
        self.lift()
        self.focus_force()
        self.grab_set()
        self.title("Instellingen")
        self.geometry("500x900")
        self.resizable(False, False)

        ctk.CTkLabel(self, text="Arduino COM-port:", font=("Arial", 14)).pack(pady=10)
        self.com_var = ctk.StringVar()
        ports = [p.device for p in serial.tools.list_ports.comports()]
        self.com_box = ctk.CTkComboBox(self, values=ports, variable=self.com_var,
        ↪  width=300)
        self.com_box.pack()

        ctk.CTkButton(self, text=" Reconnect Arduino",
        ↪  command=self._reconnect_arduino).pack(pady=10)
        self.status_label = ctk.CTkLabel(self, text="", font=("Arial", 12))
        self.status_label.pack(pady=2)

        self.file_vars = {}
        self.flow_vars = {}

        for i in range(1, 5):
```

```python
            ctk.CTkLabel(self, text=f"File path Line {i}:", font=("Arial", 12)).pack(pady=(15 if i==1 else 8, 2))
            file_var = ctk.StringVar()
            ctk.CTkEntry(self, textvariable=file_var, width=400).pack()
            self.file_vars[i] = file_var

            ctk.CTkLabel(self, text=f"Flow Meter Rate Line {i} (ml/min):", font=("Arial", 12)).pack(pady=(8, 2))
            flow_var = ctk.StringVar()
            ctk.CTkEntry(self, textvariable=flow_var, width=100).pack()
            self.flow_vars[i] = flow_var

        ctk.CTkButton(self, text="Save Settings", command=self._save).pack(pady=20)
        self._load_defaults()

    def _load_defaults(self):
        try:
            with open("config.json", "r") as f:
                config = json.load(f)
            self.com_var.set(config.get("com_port", ""))

            for i in range(1, 5):
                self.file_vars[i].set(config.get("file_paths", {}).get(str(i), ""))
                self.flow_vars[i].set(str(config.get("flow_rates", {}).get(str(i), "")))
        except FileNotFoundError:
            pass

    def _save(self):
        config = {
            "com_port": self.com_var.get(),
            "file_paths": {
                str(i): self.file_vars[i].get().strip().strip('"').strip("'")
                for i in range(1, 5)
            },
            "flow_rates": {
                str(i): float(self.flow_vars[i].get()) if self.flow_vars[i].get().strip() else 0.0
                for i in range(1, 5)
            }
        }

        with open("config.json", "w") as f:
            json.dump(config, f, indent=2)

        CONFIG.update(config)
        self.destroy()

    def _reconnect_arduino(self):
        close_serial()
        CONFIG["com_port"] = self.com_var.get()
        result = reconnect_arduino()
        if result:
            self.status_label.configure(text=" Verbinding geslaagd!", text_color="green")
        else:
            self.status_label.configure(text=" Geen verbinding mogelijk.", text_color="red")
```