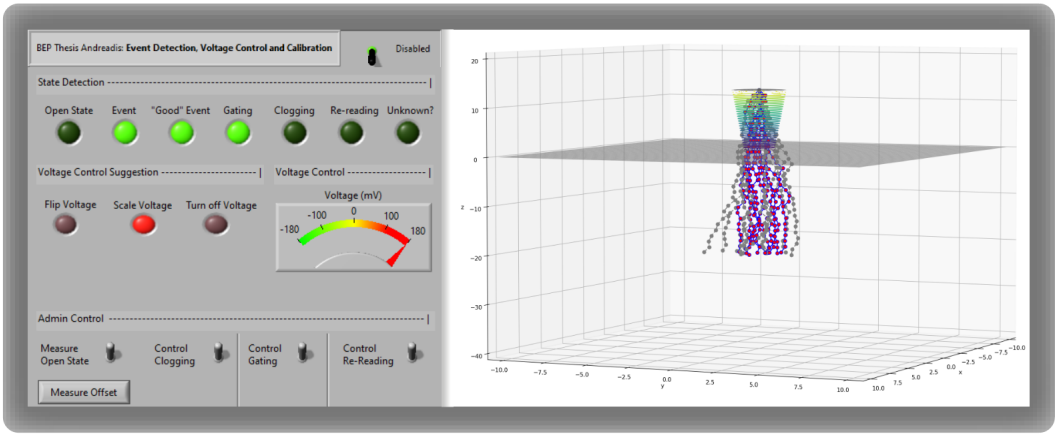
Single-molecule protein sequencing using biological nanopores.

Konstantinos Andreadis

(4999967)



End Project Thesis of the Bachelor of Science (BSC) in Applied Physics
Research group: Cees Dekker Lab
Department: Bionanoscience
Faculty of Applied Sciences (TNW)
Technical University of Delft

Date of Submission: Delft, 23. 06. 2022
To be defended publicly on the 7^{th} of July at 14:00

Thesis supervisor: Cees Dekker
Daily supervisors: Ian Nova and Justas Ritmejeris
Special thanks to: Henry Brinkerhoff

TU Delft

Abstract

The last few years have shown promising developments in single-molecule protein sequencing using a biological nanopore. Using only one membrane pore in a salt buffer, an applied potential creates a measurable ion current through the pore's constriction. If a small molecule passes through the pore, this disruption is visible as a characteristic current pattern based on the molecular structure of the molecule. Recent developments by Henry in 2021 show that single amino acid substitution in short proteins can be detected using this method. Unfortunately, due to the complexity of the reading, de novo sequencing does not yet apply to these single-molecule reads. Nevertheless, detecting different variants of a protein is possible. In current work this method is being developed to detect phosphorylation sites on a peptide of biological significance. For the latter, an immunopeptide called IRS2 is chosen to be once and twice phosphorylated. This phosphorylation causes the peptide to be a cancer biomarker. If this low-cost method of post-translational modification detection is successful, it would be an improvement compared to the standard mass spectrometry sequencing approach.

In this BEP research, problems faced before and after sequencing the IRS2 peptide are approached from two different perspectives. The first aim is to adapt the data acquisition software to automate the workflow. The second aim is to adapt and model the Freely Jointed Chain (FJC) Model to a heterogeneously charged peptide. Both aims are defined to be applicable to other peptides as well.

For the first aim, a LabVIEW plugin was developed using insights from data processing in MATLAB. This tool detects the state of the nanopore reading, responds using voltage control in a closed feedback loop and frequently allows for calibration checks. All features were tested with training sequences from real data and show promising results. However, more testing in the lab is required to determine its accuracy compared to a human operator.

For the second aim, the FJC model was analytically adapted to the IRS2 peptide inside the nanopore's electric field. The energetically most favourable configuration was then sought with a Metropolis Algorithm. As a result, the electrostatic potential was calculated and implemented into the Metropolis Algorithm. Despite the simplification of this method, it is still expandable in a modular way to incorporate additional potential such as charge-charge interactions or springs to simulate backbone flexibility. Finally, improving this model further would eventually lead to the relative positions of all chain elements inside the pore to develop an understanding of the (phosphorylated) IRS2 readings.

Table of Content

1 Introduction

In Biomedicine, accurately determining the sequence of proteins is vital in improving disease diagnosis, treatment, and prevention. While DNA sequencing is widely available and efficient, protein sequencing still is mainly dependent on inconvenient methods with complex instrumentation. These include mass spectrometry, which directly measures short peptides. Even if mass spectrometry is accurate for small peptide readings, it requires expensive instrumentation and large copy numbers. Furthermore, while DNA or RNA sequencing can determine a protein's coded sequence, mutations after biosynthesis or post-translational modifications cannot be detected using only DNA sequencing techniques. Therefore, nanopore sequencing has emerged as a new form of sequencing single molecules and is currently being adapted to protein reads.

At first, nanopore sequencing was used for reading DNA sequences (Laszlo, 2014 [1] and Nova, 2017 [2]) by the following principle: A membrane in a salt buffer contains a single pore protein. After applying a voltage, ions flowing through the pore produce a measurable current. If single-stranded DNA (ssDNA) is added and passes through the nanopore, the measured ion-current is reduced. These characteristic changes in ion current are then related to the sequence of the DNA. However, as free ssDNA will move through the pore too quickly to resolve, a motor enzyme is added to "walk" on the probe. As DNA is negatively charged, it automatically "threads" into the pore due to the electric force "pulling" it downwards. As soon as the motor enzyme attached to the end of the DNA reaches the pore protein, it starts to translocate the DNA step-wise, pulling the DNA up out through the pore and allowing the bases to be sequenced as they proceed sequentially. With each enzyme step, multiple DNA bases simultaneously influence the measured current. The latest methods even use a hectometre map (with an average of 6 bases at a time) as a reference. With the steps described above, DNA sequencing is now possible and commercially available. A more detailed explanation of the sequencing process will follow in chapter 2. Next to DNA sequencing, protein sequencing using the same biological nanopore and process is now in development. As it does not need to be negatively charged, it requires a different approach and complex resolving methods to determine its sequence.

In order to assess the DNA methods potential for sequencing proteins, it is essential to distinguish between three types of sequencing: "De Novo," "Referencing," and "Fingerprinting." De Novo relates to reading a sequence without prior knowledge, the 'ideal' case. Referencing detects changes from known sequences, e.g., mutations. Last, Fingerprinting aims to identify a protein from a mixture. In the case of the DNA nanopore sequencing method, de novo sequencing is now possible and commercially available. However, its complexity (4^4 bases) is far below that of proteins (20^8 amino acid combinations with amino acids being half the size of bases). Therefore, de novo is not yet possible for protein sequencing using nanopores. Nevertheless, referencing is possible for short proteins. Based on the research of Henry Brinkerhoff et al. in 2021 [3] (and Manrao, 2012 [4]), a DNA and peptide conjugate can be used to detect single amino acid substitutions in short proteins. For this research, it is applied to the imunopeptide IRS2, a biomarker for cancer if its S amino acid is phosphorylated once (pIRS2) or twice (p2IRS2) (Zarling, 2014 [5]). If this method is successful, it will present an opportunity for early cancer diagnosis.

However, challenges faced at the moment are not limited to the development of the method: As single-molecule sequencing is very sensitive and prone to error, the experiment runs often require the dedication of a lab worker. This work includes continuously responding to "bad" signals with voltage control and reading off characteristic current values. Automated workflow is therefore critical to efficiency as meaningful data acquisition requires multiple traces without any "errors".

The post-experimental data analysis provides several difficulties as well. Even though a nanopore has never read this immunopeptide before, it depicts an apparent phosphorylation pattern difference. However, its biophysical interpretation is rather difficult due to the complexity of such small-scale systems. In addition, one of the most significant differences with the known DNA nanopore sequencing method is the charge. As DNA is poly-negatively charged, a protein with negative, neutral or positive charge is an unprecedented molecule to be read. As all (electric) forces inside the pore depend on the charge profile, it essentially creates a new field of research. While Molecular Dynamics Simulations (MD) could help gain insight into interactions between parts of the system, a more qualitative understanding of the heterogeneity of the probe's charge in this nanopore confinement is necessary. The latter could aid in explaining the readings obtained by the lab or create the building blocks for these types of complex biophysics simulations.

This BEP research approached the two problems from two different perspectives. First, the acquisition software (LabVIEW) was adapted to automate the workflow of the measuring process in the lab. Specifically, based on MATLAB classification, a set of rules for system state detection and voltage control were added to the existing LabVIEW acquisition code, enabling automated feedback control without user input. In addition, calibration features were added for ease of use. For the second problem, a model known as the Freely Jointed Chain Model (FJC) was adapted to the heterogeneously charged probe inside the nanopore's electric field. In order to solve for the most favourable state in a given setting, it was adapted analytically and optimised with the Metropolis Algorithm. Based on this approach, an explanation of the phosphorylated peptide reading was sought. To conclude, the research aims are stated as follows:

1. Adapting the data acquisition software to automate the workflow.

2. Adapting the Freely Jointed Chain Model to a heterogeneously charged probe in an electric field, confined to a nanopore, and optimising it with a Metropolis Algorithm."

2

Theory

This chapter explains the relevant theory to understand the approach taken. First, the peptide to be sequenced is introduced, combined with a description of the full sequencing workflow. This includes an analysis of a typical reading process and its common problems. Finally, the FJC model is adapted analytically to the nanopore sequencing setup and protein used.

2.1. Immunopeptides and phosphorylations: The IRS2 peptide

In this section, the significance of the immunopeptide IRS2 is explained. As developments by Henry in 2021 [3] showed that single amino acid substitution in short proteins can be detected using nanopore sequencing with biological nanopores, an application of this principle is the immunopeptide IRS2. IRS2 is of biological significance as its phosphorylation indicates a form of cancer, acting therefore as biomarker. This phosphorylation essentially refers to its S amino acid being phosphorylated once (pIRS2) or twice (p2IRS2) as explained by Zarling in 2014 [5]. In this work, an IRS2 protein is therefore pre-ordered with Post-Translational Modifications (PTMs) at its S amino acid. These PTMs are of unprecedented value as they cannot be detected with DNA sequencing methods.

2.2. Nanopore protein sequencing in the lab

2.2.1. Experimental setup

In this section, the experimental procedure for nanopore sequencing is described. After preparation, a KCl buffer solution is added to the U-shaped tube (depicted in figure 2.1). Next, a voltage is applied, and the K+ kations and Cl- anions are attracted to their opposite charge side. Next, the lipid membrane is created and rearranged on the (-) side (seen right in figure 2.1) using air bubbles to form one homogeneous layer. Finally, the MspA is added (marked green in figure 2.1). As it attaches automatically to the membrane, a pore is opened, letting ions pass through. This "open state" current read by the ampere meter, based on experience, should be approximately 200 pA (for a given temperature) for one single MspA and an applied voltage of 180 mV. In order to remove the excess MspA, a buffer is added and extracted at the same time. As soon as one MspA is "captured," the Helicase, ATP and Magnesium are added. The setup is now complete.

2.2.2. Peptide-oligo conjugate as solution for protein sequencing challenges

In this section, the challenges of protein vs. DNA sequencing are explained, followed by an explanation of all capturing, reading etc. processes involved after a completed experimental setup.

One of the vast challenges facing protein sequencing using nanopores is their net charge. As DNA is negatively charged (the backbone having -1 as charge), peptides have heterogeneous charge. The 20 amino acids are positively, negatively, or neutrally charged. This presents a difficulty to the "threading" process, which refers to the DNA's negatively charged 'tail' being drawn to the positive side of the membrane. Furthermore, all known motor enzymes do not walk on (denatured) proteins and can therefore not ensure a slower stepping rate. A peptide-oligo conjugate (POC) is used to avoid the latter. This combination of a known single-stranded DNA (ssDNA) template, peptide, threading ssDNA, and click-attachment chemistry is shown below and in figure A.5.

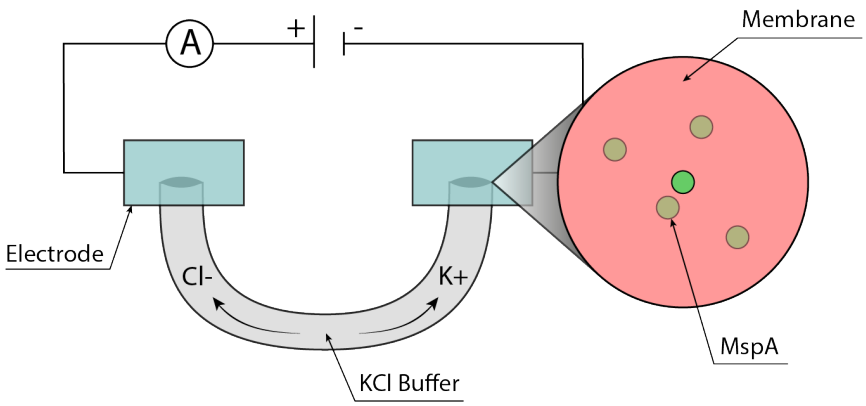


Figure 2.1: Illustration of the experimental setup of nanopore sequencing. Marked in red, a lipid membrane is applied on the negative side of a U-shaped buffer solution in a electric potential. The MspA, marked in green, attaches to said membrane. As soon as only one MspA remains, the Helicase, ATP and Magnesium are added. The setup is now complete. (Own figure created with Adobe Illustrator.)

Template DNA + DBCO + azide + YLDSGIHSGAC + mal- threading DNA

2.2.3. MspA as biological nanopore and Hel308 as motor enzyme

Next, the Mycobacterium smegmatis porin A (MspA) is used as the biological nanopore (with its key characteristics described in Laszlo, 2016 [6] and Crnkovic, 2021 [7]). In figure 2.2, it is shown inside the lipid membrane (the top view (left) and enlarged cross-section view (right)). The two main areas of interest are the vestibule and constriction. As particles can be "captured" inside the vestibule, the constriction defines the ion flow and is essentially the "sensor" of the nanopore.

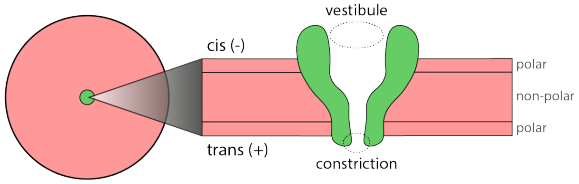


Figure 2.2: Illustration of a Mycobacterium smegmatis porin A (MspA) inside a lipid membrane indicating its vestibule and constriction areas. (Own figure created with Adobe Illustrator.)

Finally, the motor enzyme helicase (specifically: Hel308) is chosen due to its natural function of unwinding double-stranded DNA. All three POC, MspA and Hel308 enable the "capture process".

2.2.4. Capture process of a peptide-oligo conjugate in a nanopore

First, a single current value is read from the ion current flow (figure 2.3 a). Next, the POC tail threads into the pore (b) with the cholesterol binding directly to the membrane (not shown in figure), and the complementary ssDNA is "peeled" off the template ssDNA (c). As soon as the helicase rests on the MspA, the POC is "captured." The helicase now "walks" on the probe, reversing the translocation direction. (d). This complete process is equivalent to a sudden drop in current.

2.2.5. Reading process of a peptide-oligo conjugate in a nanopore

Next, as shown in figure 2.4, the template ssDNA passes through the constriction at the start of the reading process. After this, the template ssDNA is read (a), resembling a step-like current pattern. However, once the linker passes through (b), it suddenly switches to the peptide reading (c) for unknown reasons. Finally, it re-reads the second linker (d) because the helicase falls off the first linker. The latter is visible as a periodic pattern.

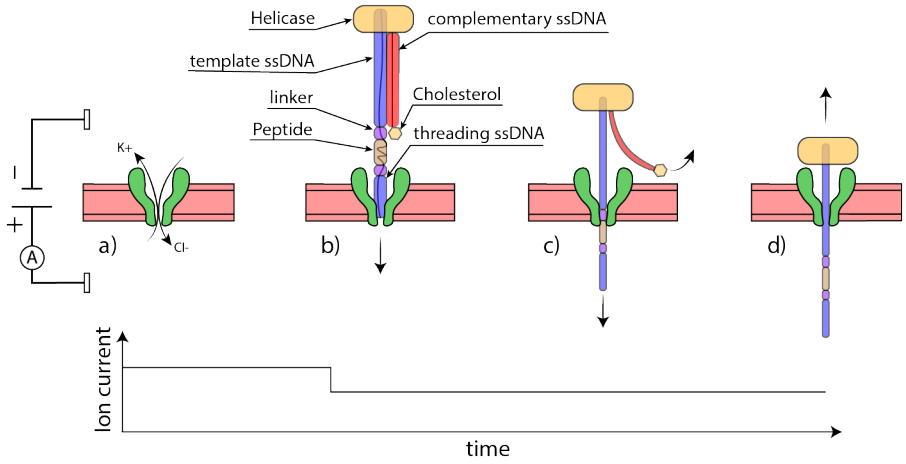


Figure 2.3: Illustration of the probe capturing process: First, a single current value is read from the ion current flow (a). Next, the POC tail threads into the pore (b), and the complementary ssDNA is "peeled" off the template ssDNA (c). As soon as the helicase rests on the MspA, the POC is "captured." The helicase now "walks" on the probe, reversing the translocation direction. (d) This complete process is equivalent to a sudden drop in current. (Own figure created with Adobe Illustrator.)

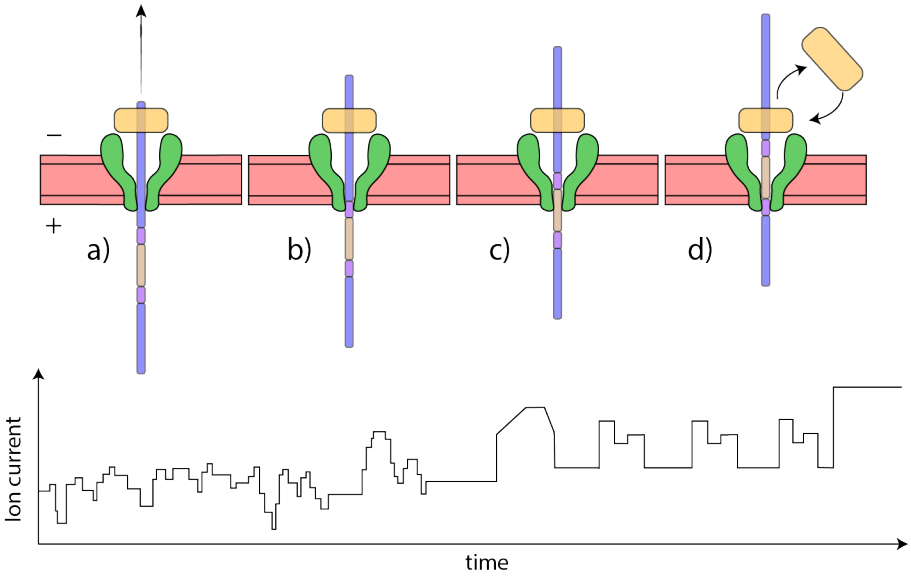


Figure 2.4: Illustration of the probe reading process: At first, the template ssDNA is read (a), resembling a step-like current pattern. However, once the linker passes through (b), it suddenly switches to the peptide reading (c) for unknown reasons. Finally, it re-reads the second linker (d) because the helicase falls off the first linker. The latter is visible as a periodic pattern. (Own figure created with Adobe Illustrator.)

2.3. LabVIEW automation: Problems during nanopore reads in the lab

A real capturing and reading process, however, is prone to various problems. First, the open state value is only stable in theory. This value is defined by the ion current if the nanopore is unoccupied, therefore "open". As single DNA strands pass through frequently, this current pattern is rather very noisy. Second, particles blocking the constriction of the MspA "clog" the pore. As depicted in figure 2.5 (a), this corresponds to a drop to a value with few/no steps. Hereafter, a typical reading is depicted with a sudden shift to a lower value. This process is called "gating" (b) and is thought to be caused by the rear ends of the MspA "folding" inside. The last error is already mentioned before: "re-reading" (c). This repeated process also hinders the nanopore in its further reads, as it would normally continue on endlessly if the user in the lab does not intervene.

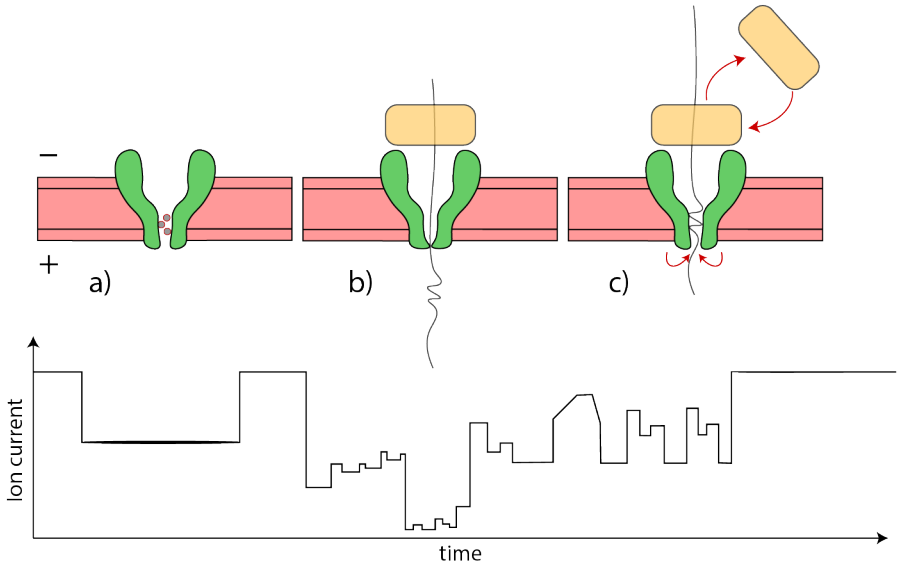


Figure 2.5: Illustration of potential errors encountered during nanopore reads. Particles blocking the constriction of the MspA "clog" its pore (a), this corresponds to a drop to a value with few/no steps. Hereafter, a typical reading is depicted with a sudden shift to a lower value. This process is called "gating" (b) and is caused by the rear ends of the MspA "folding" inside. Finally, "re-reading" (c) refers to the figure 2.4 (d). (Own figure created with Adobe Illustrator.)

With this, a collection of states is defined to describe the system during a read:

- Open State
- Event
  - "Bad" event: Clog
  - "Good" event:
    - Gating
    - Re-reading

A 'bad' event is less favourable and should be avoided, whereas a 'good' event represents a reading of the POC as a whole during a typical data acquisition. In order to manage those events, the voltage can be flipped (essentially "kicking" out the probe by repelling the DNA's negative charge) or scaled (to "reset" the system by lowering the voltage to a lower level).

In addition, two other types of calibration errors occur. First, an open state value drift is measured. This increase over time is determined by the heating process of the equipment in the lab. With higher temperature and therefore thermal fluctuations, a higher flux of ions through the pore's constriction slowly increases said value. Second, an intrinsic offset of the measurement system does always apply. This would for example not allow for zero current if no voltage is applied. Both the increased open state and offset values should be measured frequently to improve measurement.

Once traces are recorded in the lab, their interpretation still requires an improved model to describe the POC inside this nanopore confinement. For example, figure 2.4 completely disregards the chain-like character of the POC and its arrangement in space.

2.4. Python adaptation and optimisation: The Freely Jointed Chain model

An existing model for polymers is adapted to the situation at hand: The Freely Jointed Chain model (FJC). It is widely used for different types of polymers with a force applied at the end (Noakes, 2019 [8]). It describes a chain of connected elements with their respective angles and distances to one other (see figure 2.6 (a)), assuming it cannot be stretched. It is fixed at one end (the helicase) and embedded in an electric field. However, not just one force is applied at the polymer's end. Instead, it is a heterogeneously charged polymer in which each element responds differently to the potential at its position. Furthermore, each element is not equivalent to one amino acid or base. It is instead a collection, as the persistence length between elements determines the least amount of length not significantly affected by the curvature. In order to model the boundary conditions of the nanopore confinement, figure 2.6 (b) depicts the 3D areas "prohibited" for chain elements to pass through. The membrane is modelled as a flat surface at z = 0 with pore radius R, the MspA as a cone shifted and cut at both sides and the helicase as a flat surface on top of the MspA.

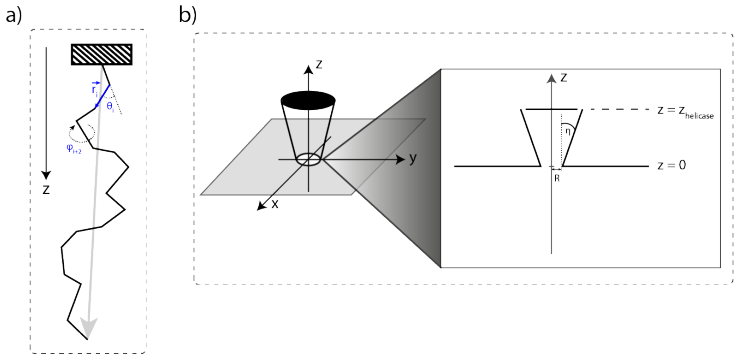


Figure 2.6: Illustration of the Freely Jointed Chain Model (a) and the simplified boundary conditions for the analytical approach taken (b).
(Own figure created with Adobe Illustrator.)

With this, the following variables are defined:

- i = chain element index
- n = total chain element count
- θ,ϕ = list of chain element relative angles with $\theta \in [0,\pi]$, $\phi \in [0,2\pi]$
- R = constriction radius
- η = MspA cone angle

The boundary condition forcing the chain to "avoid" certain regions is described by adding a potential barrier V = ∞ or indicator function $\mathbb{1} = 0$ if

- $z_i < \frac{\sqrt{x_i^2+y_i^2}}{\tan(\eta)}$ for $z \in [0, z_{hel}]$ ("avoid MspA")
- $z_i = 0 \rightarrow \sqrt{x_i^2+y_i^2}$>R ("stay inside constriction and avoid membrane")
- $(x_1, y_1, z_1) = (0,0,z_{hel})$ ("start at helicase")
- $z_n < 0$ ("end of the chain should be in trans area (below membrane) and not above (cis)")

As the FJC model is now adapted mathematically, its physical behaviour must be re-defined for this heterogeneously charged polymer and electric field. From Statistical Physics, a useful concept is the "partition function" denoted by $\mathbb{Z}$. As a chain can arrange itself in multiple ways in 3D, an ensemble of states/configurations must be analysed. For example, the derive the probability of obtaining one state is $P_{state} = \frac{1}{Z} e^{-\beta E_{state}}$ with $\beta = 1/(k_B T)$. However, i requires all states' Boltzmann distributions to be summed to form the partition function shown in equation 2.1. It is important to note that the state chosen here is a collection of relative angles and a fixed starting point in xyz.

$$\mathbb{Z} = \sum_{states} e^{-\beta E_{states}} \tag{2.1}$$

Next, the electrostatic energy of one state is defined with $V(\vec{r}_i)$ being the electric potential at location $\vec{r}_i$:

$$E_{state} = \sum_i q_i V(\vec{r}_i) \tag{2.2}$$

However, the electric potential must first be obtained by solving the Poisson equation for the given boundary conditions. Using E_{state} and taking all spherical surface elements of the angles into account according to simonensemble.github [9], the final partition function is now as follows:

$$\mathbb{Z} = \prod_i \mathbb{1}(i) \int_0^{2\pi} \int_0^{\pi} e^{-\beta U_i} \sin(\theta_i) d\theta_i d\phi_i \tag{2.3}$$

with 𝟙(i) chosen as:

$$\mathbb{1}(i) = \begin{cases} 0 : \left(z_i < \frac{\sqrt{x_i^2+y_i^2}-R}{\tan\eta} \wedge z_i \in [0, z_{hel}]\right) \vee \left(z_i = 0 \wedge \sqrt{x_i^2+y_i^2} > R\right) \\ 1 : else \end{cases} \tag{2.4}$$

The last step is the calculation of the Helmholtz Free Energy:

$$F = -\frac{1}{\beta} \ln(Z) = -\frac{1}{\beta} \ln\left(\prod_i \int_0^{2\pi} \int_0^{\pi} e^{-\beta(U_i + g(i))} \sin(\theta_i) d\theta_i d\phi_i\right) \tag{2.5}$$

If E_{state} and F are minimised, the system would theoretically be in the most energetically favourable state. However, solving this problem only analytically is not possible. Therefore, a Metropolis-Hastings algorithm is implemented. It aims to derive the target distribution of all states by using its proportionality to the partition function. Using one initial state, it chooses whether to add a new random state perturbation based on statistic transition probabilities using the aforementioned $\mathbb{Z}$. Due to its proximity to the partition function, it stays relatively close to the energetically most favourable states. In theory, if the algorithm runs for an infinite amount of iterations, it should converge to the "actual" configuration space. A more detailed algorithm is shown later (1).

3

Experimental Procedure

This chapter explains the experimental procedure for three major steps: Data processing with MATLAB, automation in LabVIEW and model adaptations in Python.

3.1. MATLAB Data Processing: Classification of reading events

The first objective of this thesis's research was to develop an automated data acquisition controller that could take the role of a technician's real-time operations during an experiment. A technician actively manages the applied voltage during an experiment based on the observed ion-current patterns. Prerecorded data was analysed to establish the categories of recorder signals and related user input for these types of signals in order to define the rules for this automated controller.

This chapter describes the data processing done in MATLAB to obtain insight for automation in LabVIEW. The goal was to classify and visualise categories of 'bad' and 'good' events. The dataset analysed in this research was the unphosphorylated immunopeptide (IRS2) reading. Recorded in LabVIEW, it contained all measured or controlled variables at a high sampling rate (50 kHz). The "event classifier" script, previously written in MATLAB, then lets the user visually classify pre-processed traces of events using two tags: The "category" and the "quality" of the event shown. In this research, six categories were visually defined based on the dataset provided and documented numerically. The categories chosen are listed as follows:

1. moderate (< 10 as standard deviation, with few exceptions) or noisy clog without steps
2. moderate clog with single or few steps
3. moderate or noisy clog with multiple steps
4. moderate or noisy clog without steps intervened
5. moderate clog with few spikes
6. gating

An example of a category 1 event is shown in figure 3.1 (a), depicting a "clog" and category 6 event of "gating" during reading (b). As qualities, three degrees of a 'success' of a reading were defined:

1. A repeating pattern is visible
2. A template ssDNA reading is visible
3. A template ssDNA and peptide reading are visible

The third quality, with an example trace shown in figure 3.2, was the "target" trace. It can be compared to the predicted sequence of the template shown in figure 3.3, where a characteristic end-of-reading collection of two peaks is visible. This type of reading was to be aimed for while recording in the lab. In order to gain insights from both categories and qualities, a MATLAB LiveScript was written. This script (Appendix A.4.2) imports data and their assigned categories and qualities. Using interactive sliders, events can be filtered by category or quality to create histograms and extract relevant data. This data included but was not limited to event duration, mean and standard deviation. For the IRS2 dataset, all insights derived via MATLAB are shown in tables A.1, A.2 and A.3. The sorted events by category and quality are shown in figure A.1. The next section will define LabVIEW state recognition logic with these characteristic values.

Finally, an additional feature in the MATLAB code was added for testing purposes. A training sequence could be created using only an array containing the indices of wanted events. The sequence also included an increase in the open state value over time to simulate thermal fluctuations and added typical open state current patterns in between events.

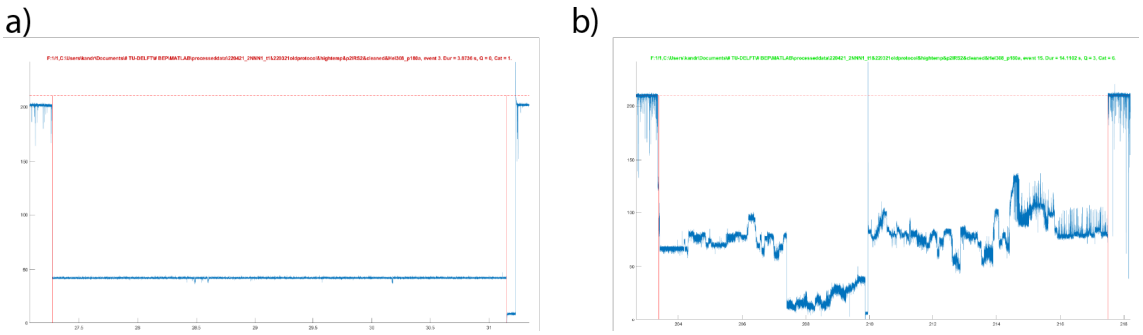


Figure 3.1: Example of a category type 1 event classified in MATLAB ("clog") (a) and category type 6 event ("gating") during a reading (b). (Own figure created with MATLAB.)

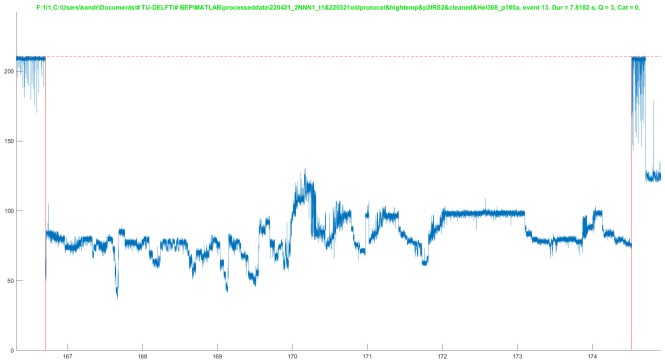


Figure 3.2: Example of a quality 3 event of a successful template, peptide and linker re-reading in MATLAB. (Own figure created with MATLAB.)

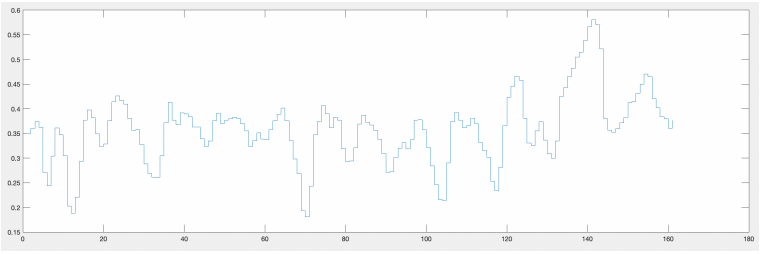


Figure 3.3: Prediction by MATLAB of the ssDNA template reading of the IRS2 probe. (Figure used with permission from Ian Nova, created with MATLAB.

3.2. LabVIEW Automation: Design and testing of the plugin

This chapter describes the steps taken to automate the data acquisition workflow in LabVIEW. As this software is to be used primarily by researchers in the lab, their needs are thoroughly documented and converted to a logic scheme for LabVIEW.

The demands are listed as follows:

- It should detect states using **moving standard deviation** and **ratio current value / open state value**.
  - Open state → relative value is above a threshold
  - Event→relative value is below a threshold
    - ◇ "Bad" Event → Standard Deviation (StdDev) is below threshold
      - · Clogging
    - ◇ "Good" Event → StdDev is above a threshold
      - · Gating →relative value is below a threshold
      - · Re-reading → typical reading duration from the start of trace reading has elapsed
- It should provide feedback on voltage control and apply it to the voltage generator.
  - If "Clogging", it should flip the voltage for short and long duration
  - If "Gating", it should scale the voltage to 1/5
  - If "Re-reading", it should flip the voltage
  - It should have safety mechanisms to ensure a smooth run
- It should implement closed-loop feedback control to check for success and adapt accordingly.
- It should periodically update the open state value if in "Open State".
- If in "Open State", it should enable turning off the voltage to measure offset current value.
- Everything should be optional (with a possibility to turn the plugin off as a whole).

Based on these demands, a flow-chart concept design is shown in figure 3.4.

To develop said automated lab "helper", an initial sandbox environment was made. It is a closed testing environment with an artificial current signal made by the user. Its feature include, for example, a turning knob to simulate a standard deviation in the current reading. The state detection, voltage control, and calibration codes were developed in this environment. The detected state is shown using indicators, and the voltage value "controlled" is depicted on the graph.

All three algorithms can be found in Appendix algorithm 2 to 4.

However, as it required real data to determine the state accurately, training sequences from MATLAB were then implemented. Figure A.3 shows an example of such a training process. All current values were read value-by-value, and the reading speed can be adjusted or a forward/reset button pressed. Up until this point, all programming was done in a different LabVIEW code. However, as the original "main.vi" code (the original front panel thereof is depicted in figure A.2) was adapted to a Data Acquisition System (DAQ), several changes had to be made when integrating code. These challenges included working with wave-forms (collection of data points with timestamps) and sampling issues. Also, all direct control of voltage must be connected to the plugin and timing issues must be addressed. The latter emerges from the target "main.vi" code uses multiple complex loops, and LabVIEW timing is unfortunately not 100% predictable.

Finally, it was tested in the lab with the real DAQ attached to it. Parallel to this, a second version was created for testing purposes. The latter simulates using a training sequence (described in algorithm 5) and saves its recommendations and/or decisions to a file for documentation purposes. From this, two final sets of codes were defined with their complete codes in Appendix A.4.2.

- main_final_andreadis.vi" containing the plugin inside the data acquisition software
- main_simulation_final_andreadis.vi containing the plugin inside a simulation environment

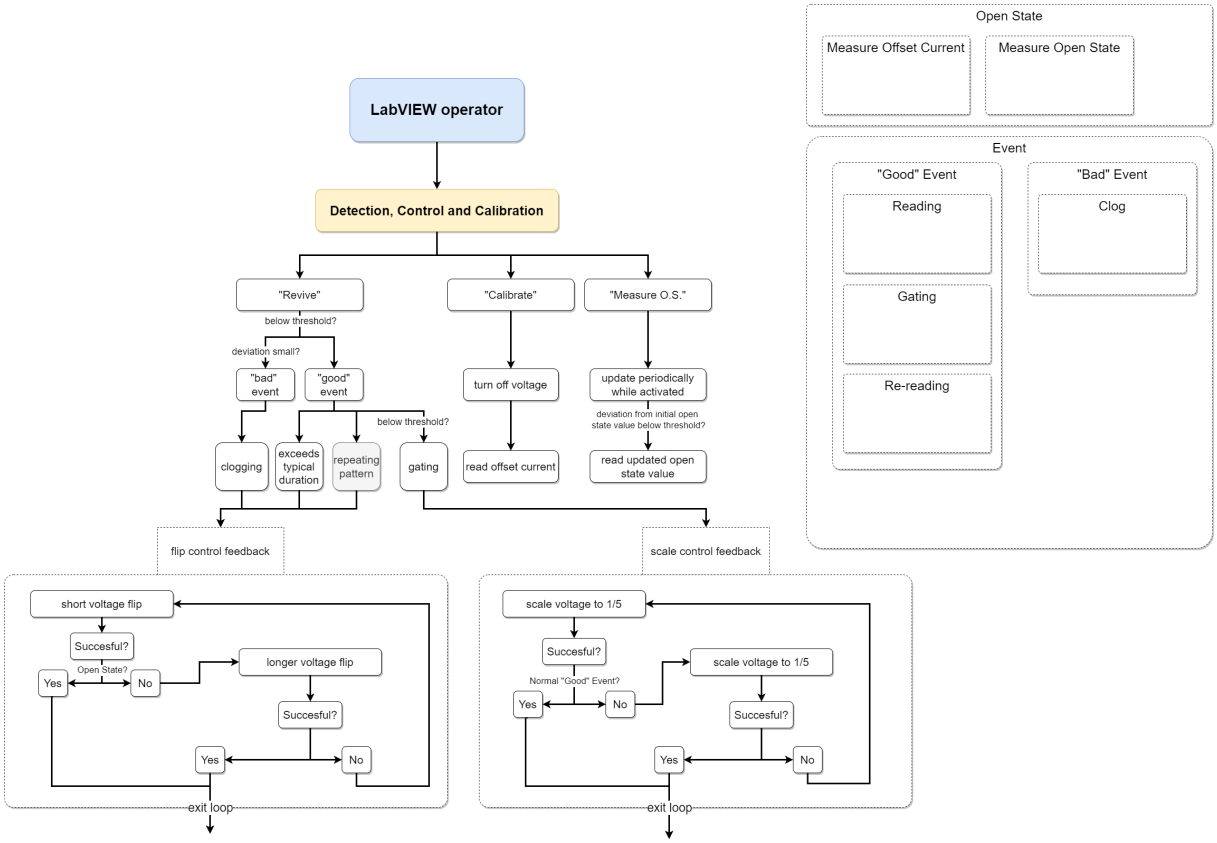


Figure 3.4: LabVIEW operator diagram flowchart describing its tasks (left) and possible states of the system (right). (Own figure created with Draw.io)

3.3. Python Model: Boundary conditions, Laplace and Metropolis

This chapter deals with the methods used in the biophysics Python model. First, the environment described in the theoretical chapter was converted to a set of planes visualised in figure 3.5. Second, an algorithm checked the validity according to figure 3.6 (a) and algorithm 8 to avoid chain-environment crossings. In figure 3.6 (b) this check is demonstrated using a random population of chains with non-valid chains marked grey.

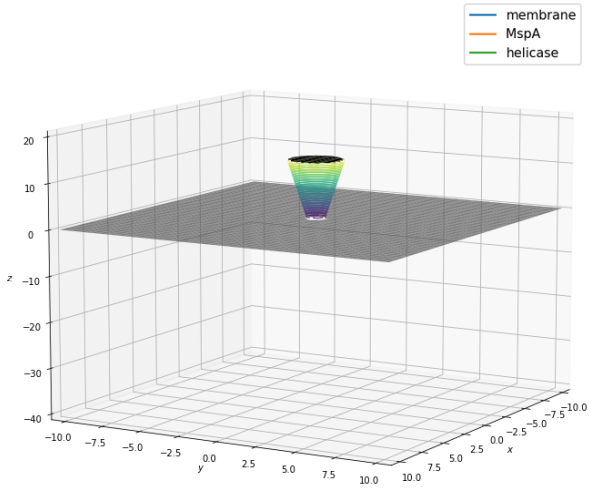


Figure 3.5: 3D boundary conditions applied with a plane as membrane, shifted and cut cone as MspA and plane as helicase. (Own figure created with Python.)

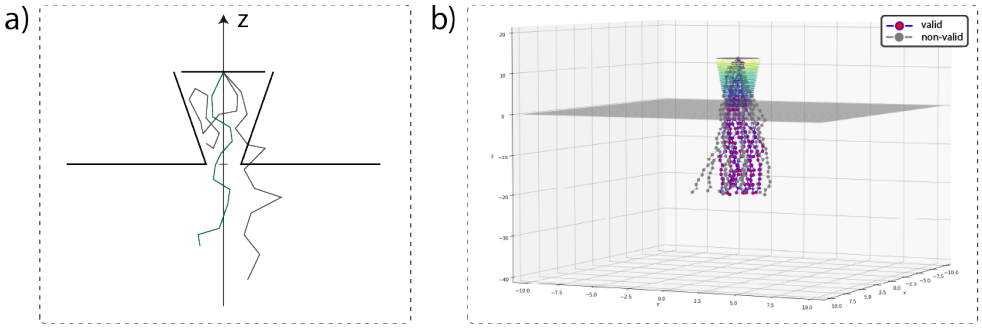


Figure 3.6: Illustration of theoretical (im-)possible configurations (a) and the validity check implementation in Python using a random sample of chains with all non-valid ones marked grey (b). (Own figure created with Adobe Illustrator (a) and Python (b))

Next, the electric field is to be calculated for the pore. A radial symmetric field is assumed to revolve around the z-axis perpendicular to the membrane, centred in the MspA. This coordinate transformation corresponds to a conversion from cylindrical (s,ϕ,z) to Cartesian (x,y,z) coordinates. Figure 3.7 shows the boundary condition, while an additional voltage drop is applied as V/2 and -V/2 at z = $+\infty$ and z=$-\infty$, respectively. In order to solve for the remaining (s,z) coordinates, the Poisson Equation must now be solved.

Even though this 2D cylindrical approach avoids extra computational time for 3D, it required some adaptations to the Poisson equation. Furthermore, no charges were assumed to be present, neglecting surface charges of the membrane, MspA and Helicase, and the charges of the probe. The latter could still be implemented by solving for each charge configuration individually. However, it is only relevant for charge-charge interactions and beyond the time scope of this Thesis. The Poisson equation, now also called the "Laplace" equation, can be rewritten as stated in equation 3.1:

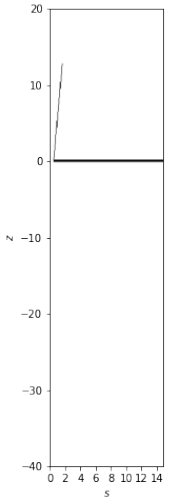


Figure 3.7: 2D plot of the boundary conditions applied for the Laplace electric potential solver. At these points in space, the electric potential should be 0. (Own figure created with Python.)

$$-\frac{\rho\left(s,\phi,z\right)}{\epsilon}=0=\nabla^2\phi=\frac{1}{s}\frac{\partial}{\partial s}\left(s\frac{\partial\phi}{\partial s}\right)+\frac{1}{s^2}\frac{\partial^2\phi}{\partial\phi^2}+\frac{\partial^2\phi}{\partial z^2}=\frac{1}{s}\frac{\partial\phi}{\partial s}+\frac{\partial^2\phi}{\partial s^2}+\frac{\partial^2\phi}{\partial z^2} \tag{3.1}$$

Due to rotational symmetry, $\frac{\partial^2\phi}{\partial\phi^2}=0$ and $\phi\left(s,\phi,z\right)=\phi\left(s,z\right)$. The fixed charges present are denoted by $\rho\left(s,\phi,z\right)$ and ϕ is the electrostatic potential of interest. However, solving this for boundary and voltage conditions requires a computational approach. For this, the Finite Difference Method is implemented, simplifying to equation 3.2:

$$\nabla^2\phi\approx\frac{1}{s}\frac{\phi\left(s+a,z\right)-\phi\left(s,z\right)}{a}+\frac{\phi\left(s+a,z\right)+\phi\left(s-a,z\right)+\phi\left(s,z+a\right)+\phi\left(s,z-a\right)-4\phi\left(s,z\right)}{a^2} \tag{3.2}$$

taking $a=1$ leads to equation 3.3:

$$\phi_{N+1}\left(s,z\right)=\frac{\phi_N\left(s+a,z\right)+\phi_N\left(s-a,z\right)+\phi_N\left(s,z+a\right)+\phi_N\left(s,z-a\right)+\frac{1}{s}\phi_N\left(s+a,z\right)}{4+\frac{1}{s}} \tag{3.3}$$

This procedural relation can be solved using the Jacobi method for solving linear partial differential equations (PDEs). Using an initial guess and many iterations, it converged to the solution. However, it is not time-efficient and is often improved by the Gauss-Seidel method, which uses already calculated values for ϕ for faster convergence. At last, Successive over-relaxation (SOR) increases Gauss-Seidel's efficiency even more by using $\phi_{new}\left(x,y,z\right)=\phi_{old}\left(x,y,z\right)+\omega\delta\left(x,y,z\right)$. ω is the SOR parameter between 1 and 2, while $\delta\left(x,y,z\right)$ is the difference between the newly calculated potential and the one before. All steps are described in detail in algorithm A.4.2.

Now that the electrostatic potential was calculated, a chain must be created in 3D with algorithm 7. The latter requires a spherical to Cartesian coordinate transformation described in algorithm 6. For completeness, each chain is a matrix with the following five columns: The x, y, and z positions and relative angles θ and ϕ. Each row represents one element of the chain. The collection of chains would then be an array containing individual matrices. Next, the charge distribution was defined by dividing the chain into segments with their respective charges. Figure 3.8 depicts these charge profiles for the two different phosphorylations (see algorithm 7. Note that this division is not to scale and certainly simplifies the charges of the probe, neglecting net charge effects.

To calculate the electrostatic energy of a state, all left to do is use the afore-calculated ϕ in algorithm 12 However, the difficulty lies within the discrete nature of finite element analyses. Therefore, a very dense grid is

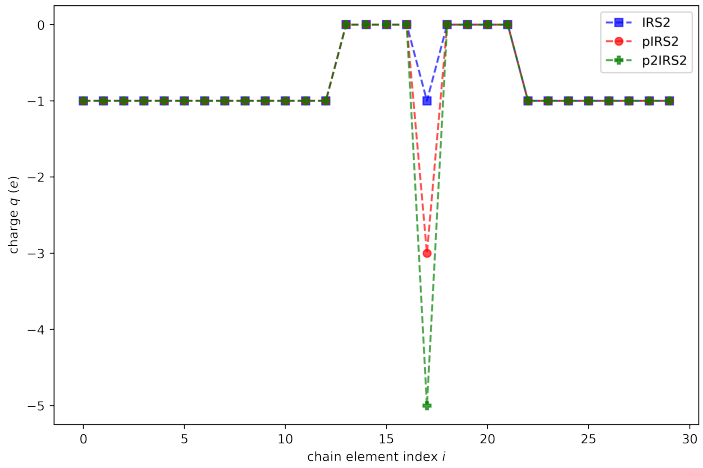


Figure 3.8: Plot of charge vs. chain element index for three degrees of phosphorylation of a IRS2 immunopeptide. (Own figure created with Python.)

chosen (200x200 points), and the position of each element is "snapped" to a grid point. With this algorithm 9, each electrostatic potential can be obtained and multiplied by the respective charge.

Finally, a Metropolis Algorithm is implemented as described in augustinus.kristia.de [10] and statlect.com [11]. The initial state is a collection of zeros for the θ and ϕ angles, thus a vertical line. With each iteration, an angle perturbation is drawn from a Normal($\mu = 0$, $\sigma = 1$) distribution. Based on a comparison between a Uniform(0,1) distribution sample and the new-to-old ratio of the Helmholtz Free Energy, the new state is accepted. The latter also requires the new state to be a valid state based on algorithm 8. These steps are shown in algorithm 1. Overall the whole progress is saved as a .gif animation for visualisation purposes (to be shown at the Thesis Defence).

Algorithm 1: Metropolis Algorithm

input:

- N_iterations (-)
- initial [] and [] (both arrays in radians)
- k_B (J/K)
- T (K)
- check_valid (function)
- E (function)

output:

- accepted/refused states (array of arrays)

$\beta = 1/(k_B * T)$;
for i in range(N_iterations) **do**
    $\Delta\theta$,$\Delta\phi$ = random.normal();
    P_accept = exp(β*[E(θ + $\Delta\theta$,ϕ + $\Delta\phi$) - E(θ,ϕ)]) * Π_i (sin(θ[i] + $\Delta\theta$[i])/sin(θ_i));
    sample = random.uniform(0,1);
    if (sample < min(P_accept,1) and (check_valid(chain($\theta + \Delta\theta$,$\phi + \Delta\phi$)) == True) **then**
        θ+ = $\Delta\theta$;
        ϕ+ = $\Delta\phi$;
    end
end

4

Results and Discussion

This chapter describes the results from both LabVIEW automation and Python Modelling, as well as discusses their relevance and/or potential shortcomings.

4.1. LabVIEW Automation: Developed plugin and training results

First, the results obtained from automating the data acquisition software in LabVIEW were analysed. Figure 4.1 shows the developed plugin, with variables to be controlled, state detection indicators, voltage control suggestions and admin control features. The simulation environment, depicted in figure A.4, recorded the main results. It is to be noted that the 're-reading' detection only works for the given dataset, as its precise calculation would require a full Hel308 reaction scheme analysis to determine the enzyme stepping rate in the given conditions. The latter would, however, exceed the scope of this BEP project. Figures 4.2(and Appendix A.6 and A.7) show the three training sequences used as well as a precise state detection denoted by the state index, a voltage control suggestion and an Open State value. Overall, it shows a successful implementation of the requirements set by the user. As marked with text boxes, the open state was detected, followed by a "good" reading event. As soon as the drop was detected, a "gating" state was marked, followed by the scaling of the voltage. After a typical duration had elapsed, "re-reading" was marked, ended with a return to the open state. As its open state value had now increased, the update was indeed successful. The "bad" event then triggered the flipping of the voltage in a periodic manner until the open state was reached again.

Three critical insights were gained from these training sets: At every state change an initial time lag was required until the response was taken. This is determined by the standard deviation point-by-point calculation requiring a certain amount of samples to continue. Furthermore, frequent open state value updates improved the initial guess correctly. Finally, voltage control mimicked the actions of a lab worker for flipping or scaling the voltage.

When testing in the lab, multiple insights were gained as well. First, the Open State update should only be accepted if the relative deviation from the previous one is not too significant (by a deviation of $\approx 20\%$). Second, all 'wait time' functions need to be converted to 'elapsed time' functions to avoid timing issues. Finally, the standard deviation should be 0 if the state is in an "Open State" or if it is too large due to voltage control actions.

This feedback was implemented, and the final code A.4.2 and algorithms A.4.1 were made completely modular for future changes and available in the Appendix. However, more lab tests had not been possible due to time constraints and difficulties in sample preparation. For future improvements, more safety features must also be implemented to avoid a direct voltage flip when a "good" reading is about to occur. Furthermore, the standard deviation calculation must be adapted for better compatibility with different sampling frequencies. Finally, a type of confidence metric should be determined by recording actual lab responses and suggestions of the software to compare its accuracy.

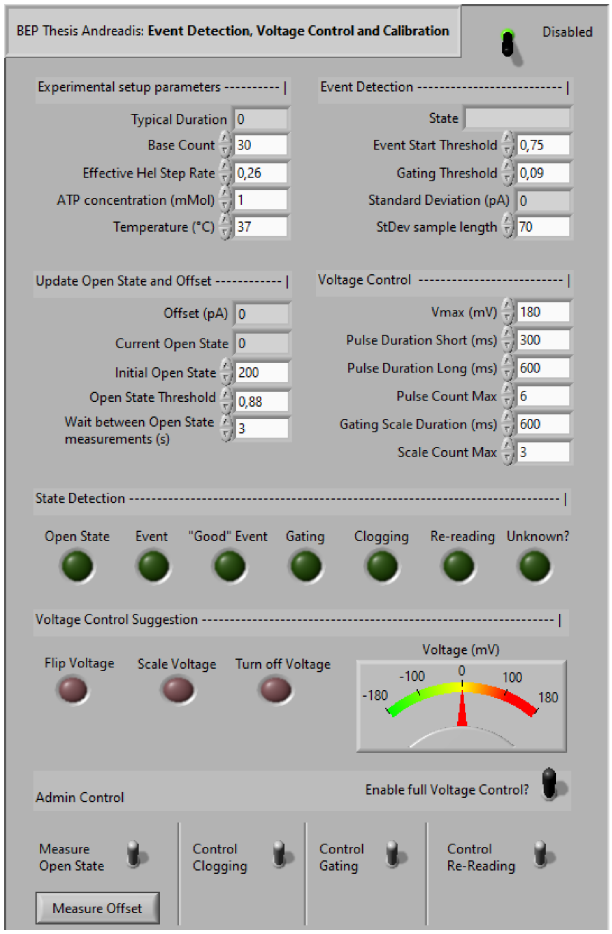


Figure 4.1: LabVIEW plugin front panel view. (Own figure export from LabVIEW)

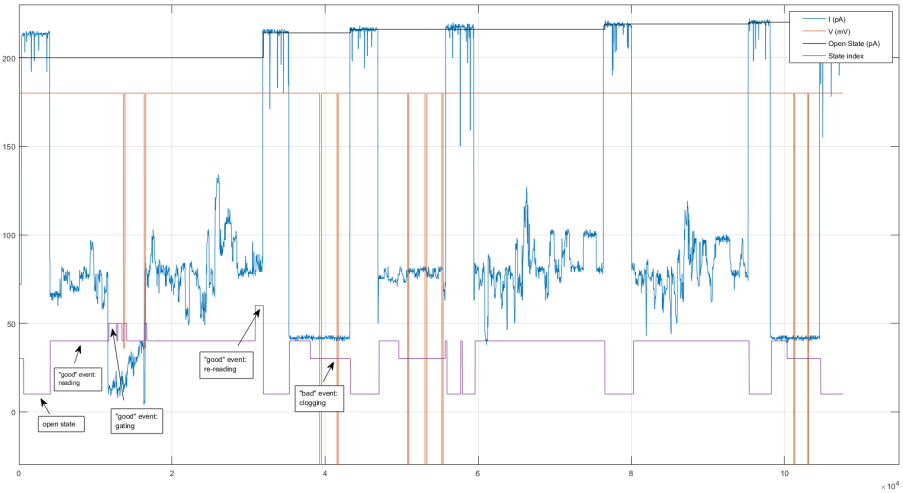


Figure 4.2: Training result of the state detection and voltage control features of the LabVIEW code when given a training sequence generated from real data in MATLAB. The different plotted lines denote the measured current I (pA), controlled voltage V (mV), detected Open State (pA) and State Index. (Own figure created with MATLAB.)

4.2. Python Model: Model adaptation and optimisation results

This chapter describes the result of the Python model adaptation and simulation. First, the pore configuration's electrostatic potential was solved, as visible in figure 4.3. Furthermore, the validity check of the chain inside the boundary conditions was proven to be very rigid for various amounts of random testing populations. However, most importantly, the Metropolis algorithm for the probe IRS2 (so without any phosphorylation) showed promising results. First, as depicted in figure 4.4 (a), the states it accepted are all valid. Second, shown in figure 4.4 (b), the electrostatic energy depicts a decreasing trend with each new iteration. Finally, as the state with minimal energy, figure 4.5 depicts such a configuration. It is plotted for the IRS2 probe (a), once (b) and twice (c) phosphorylated according to the charge profile shown in 3.8. A full .gif movie animation of the optimisation process will be shown at the thesis defence. It is fascinating to see the outcomes of the algorithm and its arrangement with respect to the electrostatic potential. One possible interpretation of a state with more horizontal elongation is its preference for the x-y plane. As the potential increases with lower z, aiming for the same z value would decrease the energy. However, this would need more extensive testing and more accurate parameters are needed, especially for the persistence length defining the chain element spacing. In this case, a guess of 0.6nm as pore diameter, 5 degrees as pore angle with the vertical, a height of 8nm and chain spacing of 1.2 nm are given for the situation. Nonetheless, it did prove the method developed for the FJC model yields a chain θ and ϕ space which can be optimised. Finally, it is to be noted that while this approach may seem to oversimplify all conditions of the actual situation, it is expandable in a modular way. These extensions include: First, charge-charge interactions can be added by solving the Laplace equation for each charge configuration. Second, a spring potential can be added to simulate backbone flexibility and thus creating an Extensible FJC model. Third, electro-osmotic potentials and surface charge–probe interactions could be added. All these additional potentials are merely added to the exponential inside the partition function. Combined with these improvements, a more extensive test run must be conducted to understand the readings differing in three degrees of phosphorylation. The latter includes analysing the relative positions of all chain elements with respect to the MspA's constriction and solving these for different charge profiles as well as chain lengths. Finally, the last point of discussion would be related to the fixed system in 3D, as the MspA can also translocate in the xy plane, adding a force applied by the constriction during movement.

The final code in Jupyter Notebook format A.4.2 and algorithms A.4.2 were all made available in the Appendix.

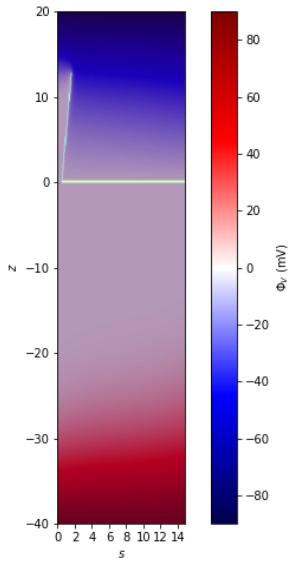


Figure 4.3: Electrostatic potential derived by solving the Laplace equation with successive over relaxation in Python, approached with Finite Difference Methods. It clearly responds to the boundary conditions imposed.
(Own figure created with Python.)

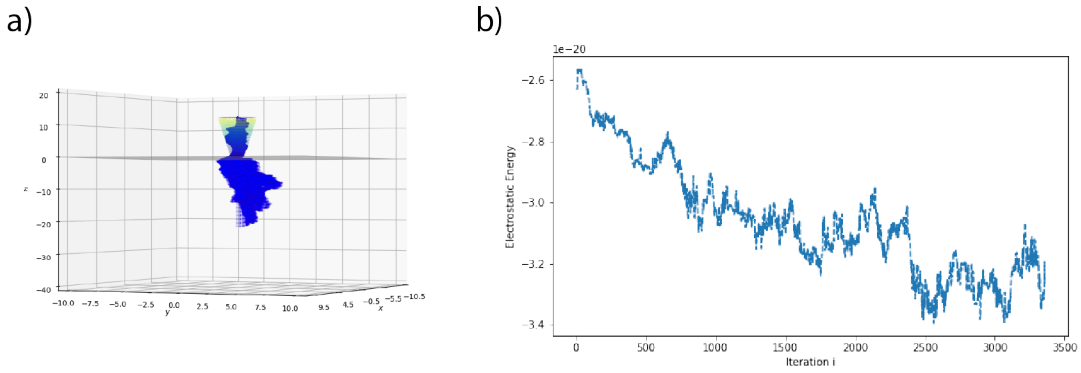


Figure 4.4: The states accepted by an iteration of 80000 steps are all drawn together to visualise the validity condition always being met(a). Second, shown in figure (b), the electrostatic energy depicts a decreasing trend with each new iteration. (Own figure created with Python.)

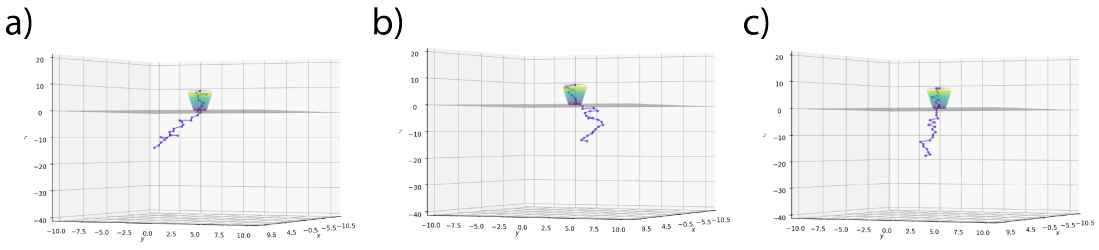


Figure 4.5: 3D plot of the three lowest energy states derived with the help of the Metropolis algorithm for the immunopeptide IRS2 (a) being once (b) and twice (c) phosphorylated. (Own figure created with Python.)

5

Conclusions

In this BEP research, problems faced before and after sequencing the IRS2 peptide were approached from two different perspectives. The first aim was to adapt the data acquisition software to automate the workflow. The second aim was to adapt and model the Freely Jointed Chain (FJC) Model to a heterogeneously charged probe.

At first, the acquisition software in LabVIEW was adapted with MATLAB data processing insights. Then, tested with training sequences, it accurately determined the system's state and responded to it. Next, its calibration features, such as the offset measurement and open state value update, showed promising results in actual lab test runs. However, more lab tests were needed as well as safety features to avoid e.g. a direct voltage flip when a "good" reading is about to occur. Furthermore, the standard deviation calculation must be adapted for better compatibility with different sampling frequencies. Finally, a type of confidence metric should be determined by recording actual lab responses and suggestions of the software to compare its accuracy.

For the second problem, the Freely Jointed Chain Model model was analytically adapted to the IRS2 probe inside the nanopore's electric field. The energetically most favourable configuration was then sought with a Metropolis Algorithm. As a result, the electrostatic potential was calculated and implemented into the Metropolis algorithm. Despite the simplification of this method, it is still expandable in a modular way to incorporate additional potential such as charge-charge interactions by solving the Laplace equation for each charge configuration, a spring potential to simulate backbone flexibility, electro-osmotic potentials and surface charge–probe interactions. These additional potentials are merely added to the exponential inside the partition function. Finally, improving this model in the feature would eventually lead to the relative positions of all chain elements with respect to the MspA's constriction to develop an understanding of the (phosphorylated) IRS2 readings.

Bibliography

1. Laszlo, A. H. et al. Decoding long nanopore sequencing reads of natural DNA. Nat Biotechnol **32,** 829–33. ISSN: 1546-1696. https://www.ncbi.nlm.nih.gov/pubmed/24964173 (2014).
2. Nova, I. C. et al. Investigating asymmetric salt profiles for nanopore DNA sequencing with biological porin MspA. PloS one **12,** e0181599–e0181599. ISSN: 1932-6203. https://pubmed.ncbi.nlm.nih.gov/28749972%20https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5531483/ (2017).
3. Brinkerhoff, H., Kang Albert, S. W., Liu, J., Aksimentiev, A. & Dekker, C. Multiple rereads of single proteins at single–amino acid resolution using nanopores. Science **374,** 1509–1513. https://doi.org/10.1126/science.abl4381 (2021).
4. Manrao, E. A. et al. Reading DNA at single-nucleotide resolution with a mutant MspA nanopore and phi29 DNA polymerase. Nat Biotechnol **30,** 349–53. ISSN: 1546-1696. https://www.ncbi.nlm.nih.gov/pubmed/22446694 (2012).
5. Zarling, A. L. et al. MHC-Restricted Phosphopeptides from Insulin Receptor Substrate-2 and CDC25b Offer Broad-Based Immunotherapeutic Agents for Cancer. Cancer Research **74,** 6784–6795. ISSN: 0008-5472. https://doi.org/10.1158/0008-5472.CAN-14-0043 (2014).
6. Laszlo, A. H., Derrington, I. M. & Gundlach, J. H. MspA nanopore as a single-molecule tool: From sequencing to SPRNT. Methods **105,** 75–89. ISSN: 1046-2023. https://www.sciencedirect.com/science/article/pii/S1046202316300615 (2016).
7. Crnkovic, A., Srnko, M. & Anderluh, G. Biological Nanopores: Engineering on Demand. Life **11,** 27 (2021).
8. Noakes, M. T. et al. Increasing the accuracy of nanopore DNA sequencing using a time-varying cross membrane voltage. Nature biotechnology **37,** 651–656. ISSN: 1546-1696 1087-0156. https://pubmed.ncbi.nlm.nih.gov/31011178%20https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6658736/ (2019).
9. Ensemble, T. S. The freely jointed chain (FJC) model https://simonensemble.github.io/2017-10/freelyjointedchain.
10. 2022. https://agustinus.kristia.de/techblog/2015/10/17/metropolis-hastings/.
11. Taboga, M. Metropolis-hastings algorithm https://www.statlect.com/fundamentals-of-statistics/Metropolis-Hastings-algorithm.

A

Appendix

A.1. Additional LabVIEW and MATLAB automation material

A.1.1. MATLAB event classification results

Table A.1: Results of category classification (of IRS2 events) in MATLAB .

Category	Description	Event Nr.	Mean	St. Dev.	Open State	±	Level 1	Level 2	Level 3	Level 4	Steps?	Spike?	Duration (s)
1	moderate more noisy clog without steps	5	38.9	3.0	202	1	39	-	-	-	0	0	2.356
1	moderate clog without steps	3	42.2	1.0	202	1	42	-	-	-	0	0	2.424
1	noisy clog without steps	12	78.5	10.6	208	1	77	-	-	-	0	0	2.298
1	moderate clog without steps	17	77.9	3.5	212	1	77	-	-	-	0	0	5.768
1	moderate clog without steps	21	68.0	2.1	213	2	68	-	-	-	0	0	2.9772
2	moderate clog with single step	4	80.1	9.0	202	1	61	84	-	-	1	0	3.872
2	moderate clog with single step	7	64.9	7.5	203	2	82	62	-	-	1	0	2.058
2	moderate clog with few initial steps	19	84.6	4.8	212	1	85		-	-	few	0	2.706
2	moderate clog with few steps	20	66.8	7.2	212	2	74	64	80	60	few	0	4.306
2	moderate clog with single step	30	72.0	9.2	215	1	66	87	-	-	1	0	3.538
3	moderate clog with small multiple steps	9	77.8	3.5	206	1	78		-	-	multiple	0	4.344
3	noisy clog with multiple steps	25	63.5	15.4	214	2	80	59	75	53	multiple	0	2.929
3	noisy clog with multiple steps	27	58.0	17.1	214	1	52	99	-	-	multiple	0	3.06
4	noisy clog without steps intervened	6	45.9	26.3	203	1	50	-	-	-	0	0	3.65
4	moderate clog without steps intervened	8	67.9	21.5	206	1	70.5	-	-	-	0	0	5.368
4	moderate clog without steps intervened	26	54.9	20.9	214	2	53	-	-	-	0	0	2.914
5	moderate clog with two spikes	10	52.9	9.5	207	1	51	-	-	-	0	2	7.408
5	moderate clog with initial step and spike	23	59.5	8.9	213	2	84	56	106	-	1	1	4.306
6	gating	11	82.7	9.2	208	-	90	49	-	-	-	0	5.738
6	gating	15	69.3	26.8	210.5	-	79	14	-	-	-	0	14.11

Table A.2: Classification of IRS2 MATLAB "gating"-type events.

Quality	Type	Event Nr.	Duration (s)	St. Dev.	Lower Threshold I/I0	Open State	Mean	St. Dev.	Level 1	Level 2
1	Repeating Pattern	11	5.738	9.1873	0.236	208	82.7172	9.1873	90	49
3	DNA + peptide read	15	14.11	26.8495	0.067	210.5	69.2967	26.8495	79	14
	Typical duration (s):	7.636								
	Lower gating threshold:	0.236								

Table A.3: Results of quality classification (of IRS2 events) in MATLAB .

Quality	Description	Event Nr.	Duration (s)	St. Dev.
1	Repeating Pattern	33	4.2546	11.8848
1	Repeating Pattern	49	6.266	13.4151
1	Repeating Pattern	11	5.738	9.1873
2	DNA read	35	2.5408	11.3213
2	DNA read	16	3.298	15.232
2	DNA read	51	4.572	13.5364
2	DNA read	38	2.849	10.013
2	DNA read	18	3.386	10.3257
3	DNA + peptide read	14	7.23	13.7489
3	DNA + peptide read	13	7.818	12.793
3	DNA + peptide read	50	7.86	14.7071
3	DNA + peptide read	28	8.21	13.5463
3	DNA + peptide read	15	14.11	26.8495

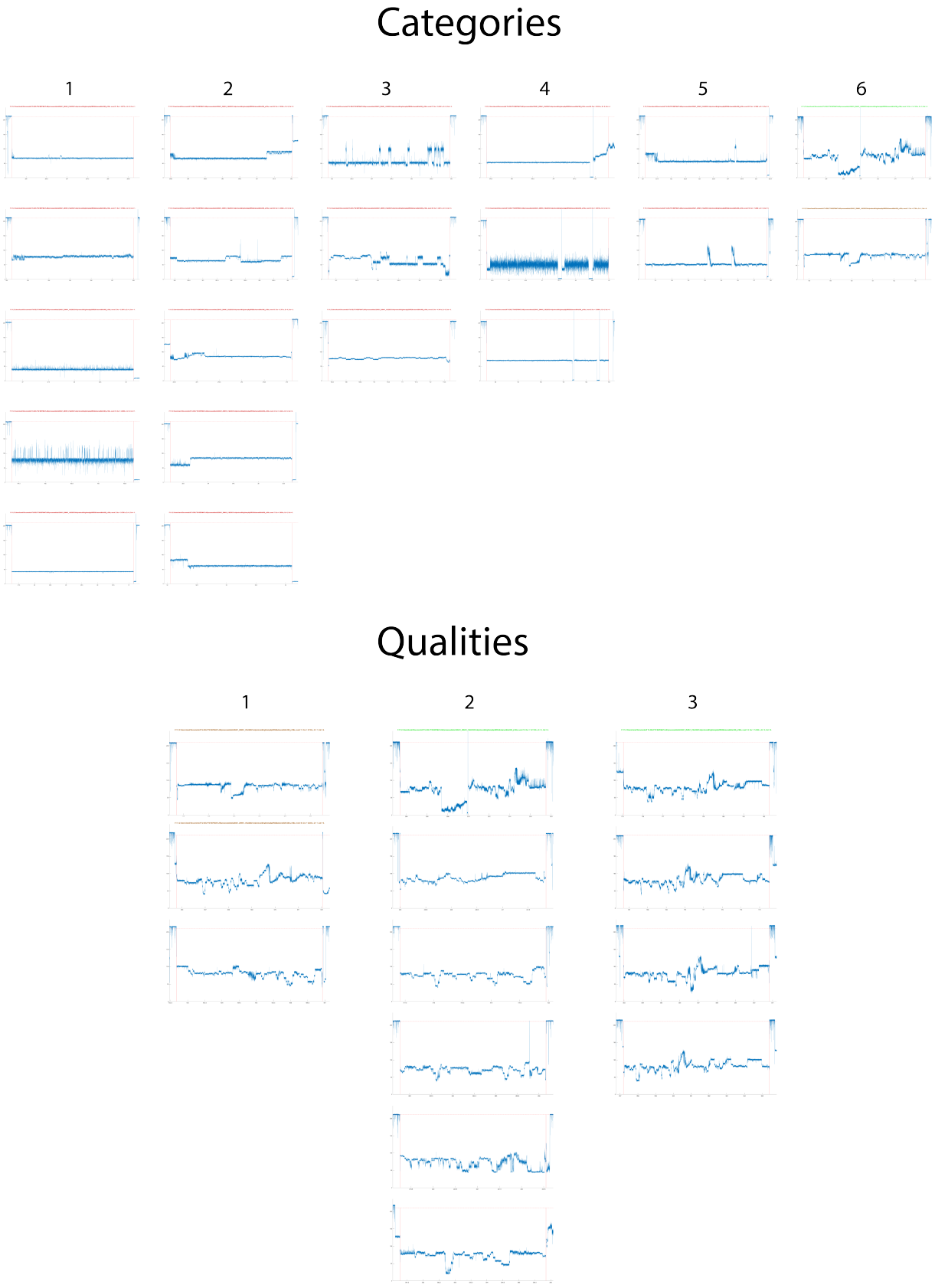


Figure A.1: All MATLAB classified events sorted by 6 categories and 3 qualities. (Own figure created with MATLAB.)

A.1.2. Original LabVIEW front panel

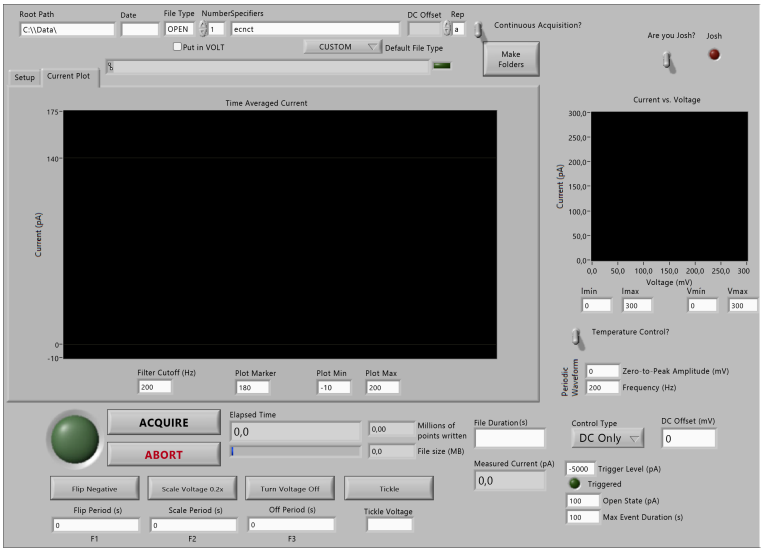


Figure A.2: Original LabVIEW front panel view "main.vi".
(Own exported figure of LabVIEW.)

A.1.3. Intermediate testing recording snapshot

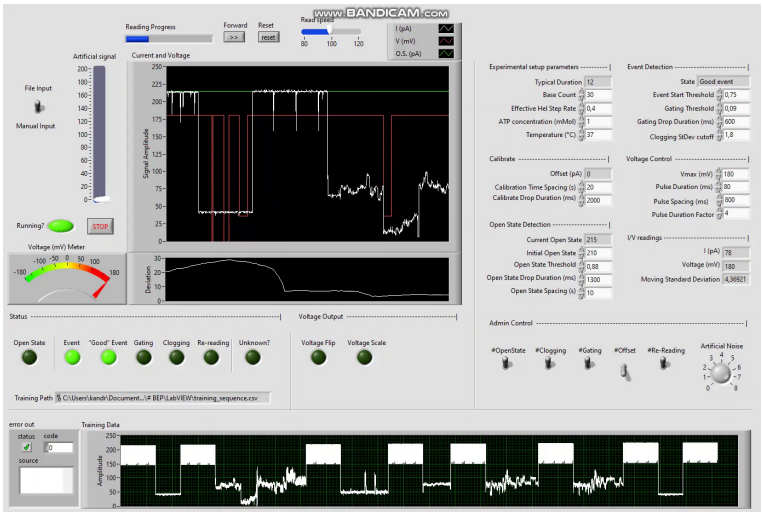


Figure A.3: Snapshot of training sequence simulation in LabVIEW.
(Own exported figure of LabVIEW.)

A.1.4. Final simulated testing front panel

Figure A.4: Final simulation front panel view in LabVIEW.
(Own exported figure of LabVIEW.)

A.2. Chemical drawing of IRS2

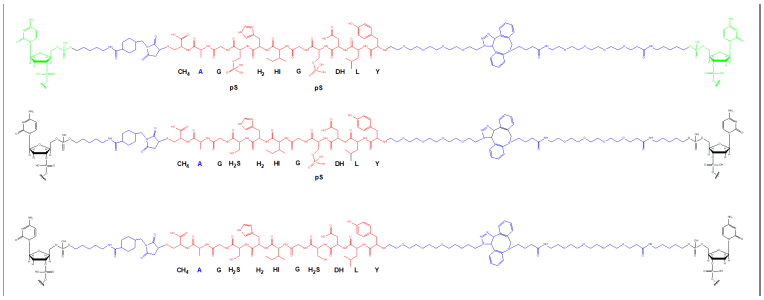


Figure A.5: Chemical Drawing of peptide-oligo conjugate IRS2 used to allow for peptide sequencing with previous DNA nanopore sequencing methods. (Figure used with permission from Ian Nova using ChemDraw.)

A.3. Additional training sequences results

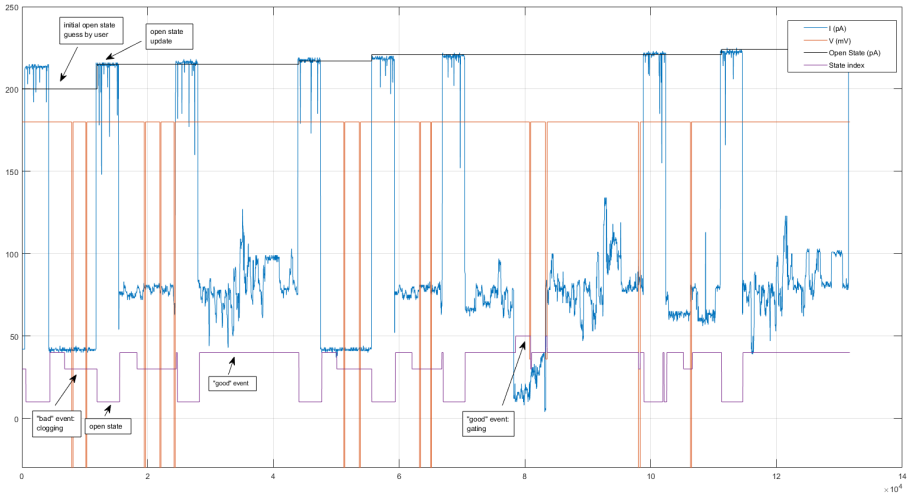


Figure A.6: Training result 2 of the state detection and voltage control features of the LabVIEW code when given a training sequence generated from real data in MATLAB. The different plotted lines denote the measured current I (pA), controlled voltage V (mV), detected Open State (pA) and State Index. (Own figure created with MATLAB.)

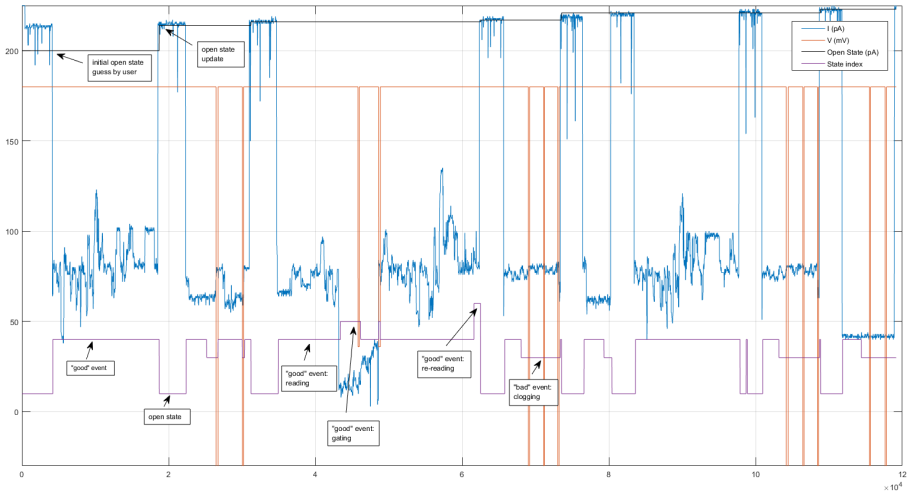


Figure A.7: Training result 3 of the state detection and voltage control features of the LabVIEW code when given a training sequence generated from real data in MATLAB. The different plotted lines denote the measured current I (pA), controlled voltage V (mV), detected Open State (pA) and State Index. (Own figure created with MATLAB.)

A.4. All developed algorithms

A.4.1. LabVIEW algorithms

Algorithm 2: State detection algorithm

input:

- activate operator (True/False)
- standard deviation (pA)
- “bad” event stdev cutoff (pA)
- open state threshold (-)
- current open state (pA)
- measured current (pA)
- event start threshold (-)
- gating threshold (-)
- typical duration (s)

output:

- state (“”)
- state_index (-)
- boolean (led) indicators (True/False):unknown, open state, event, clogging, good event, gating, re-reading

```
while activate operator == True do
    current ratio = measured current (pA) / current open state (pA);
    if standard deviation > stdev voltage control cutoff then
        standard deviation = 0;
    end
    if current ratio > open state threshold then
        state = “open state”;
        open state = True;
        state_index = 1;
    end
    if not open state and current ratio < event start threshold then
        state = “event”;
        event = True;
        state_index = 2;
    end
    if event and standard deviation < “bad” event stdev cutoff then
        state = ”clog”;
        clogging = True;
        state_index = 3;
    else
        state = “good event”;
        good event = True;
        state_index = 4;
    end
    if good event then
        if current ratio < gating threshold then
            state = ”gating”;
            gating = True;
            state_index = 5;
        end
        if elapsed time > typical duration then
            state = ”re-reading” ;
            re-reading = True ;
            state_index = 6 ;
        end
    end
    if (open state, event, clogging, good event, gating, re-reading) == False then
        state = “unknown” ;
        unknown = True;
        state_index = 0;
    end
end
```

Algorithm 3: LabVIEW Voltage control algorithm

input:

- activate operator (True/False)
- state ("")
- gating scale duration (ms)
- vmax (mV)
- pulse duration long (ms)
- pulse duration short (ms)
- scale count max (-)
- pulse count max (-)
- boolean controls (True/False):control clogging, gating and re-reading

output:

- voltage (mV)
- pulse count (-)
- scale count (-)
- boolean led indicators (True/False): flip, scale and turn off voltage

```
while activate operator == True do
    if state == "good event" then
        scale voltage, flip voltage = False;
        voltage = vmax;
    end
    if state == "clog" then
        scale voltage = False;
        if control clogging and not gating then
            if pulse count > pulse count max / 2 then
                time to wait = pulse duration short;
            else
                time to wait = pulse duration long;
            end
            if not flip voltage and time to wait then
                pulse count += 1;
                voltage = -vmax;
                flip voltage = True;
            else
                if pulse count == pulse count max then
                    pulse count = 0;
                end
                voltage = vmax;
                flip voltage = False;
            end
        end
    end
    if state == "gating" and control gating then
        [ additional code is left out, however visible in LabVIEW block diagram ] if elapsed time > gating
        scale duration then
            voltage = 0.2 * vmax;
            scale voltage = True;
        else
            voltage = vmax;
            scale voltage = False;
        end
    end
    if state == "re-reading" and control re-reading then
        if control clogging and not gating then
            if pulse count > pulse count max / 2 then
                time to wait = pulse duration short;
            else
                time to wait = pulse duration long;
            end
            if not flip voltage and elapsed() == time to wait then
                pulse count += 1;
                voltage = -vmax;
                flip voltage = True;
            else
                if pulse count == pulse count max then
                    pulse count = 0;
                end
                voltage = vmax;
                flip voltage = False;
            end
        end
    end
end
```

Algorithm 4: labview Calibration algorithm

input:

- activate operator (True/False)
- state ("")
- wait between open state updates (s)
- initial open state (pA)
- measured current (pA)
- measure open state (True/False)
- measure offset (True/False)

output:

- current open state (pA)
- offset (pA)
- boolean led indicators (True/False)
  - flip voltage
  - scale voltage
  - turn off voltage
  - open state update

```
while activate operator == True do
    flip voltage = False;
    if measure open state and elapsed time > wait between open state updates then
        open state update = True;
        if |measured current - initial open state| / initial open state < 0.2 then
            current open state = measured current;
        else
            current open state = initial open state;
        end
    end
    if measure offset then
        turn off voltage = True;
        voltage = 0;
        if measured current < 5 then
            offset = measured current;
        end
    else
        turn off voltage = False;
        voltage = vmax;
    end
end
```

Algorithm 5: LabVIEW Simulation Training algorithm

input:

- training data (.csv spreadsheet separated by "," delimiter)
- read speed (-)
- restart file reading (True/False)
- voltage (mV)
- current open state (pA)
- state index

output:

- record (.csv spreadsheet separated by ";" delimiter for columns)

```
while not LabVIEW_stop do
    if i >= size(training data) or restart file reading then
        i = 0;
    else
        i += 1;
    end
    measured current = training data [ range (i, size(training data) ];
    write_delimited_spreadsheet(measured current, voltage, current open state, state index);
end
```

A.4.2. Python Model algorithms

Algorithm 6: sph2cart:Spherical to Cartesian coordinate converter

input:

- r (-)
- θ,ϕ (rad)

output:

- x,y,z (-)

```
x = r*sin(theta)*cos(phi);
y = r*sin(theta)*sin(phi);
z = r*cos(theta);
```

Algorithm 7: create_chain: Create an ideal 3D chain

input:

- θ,ϕ (arrays in rad)
- chain_spacing (-)

output:

- chain (matrix):
  - 1st column: x coordinates
  - 2nd column: y coordinates
  - 3rd column: z coordinates
  - 4th column: theta angles
  - 5th column: phi angles

```
chain = zeros((N_chain, 5)) chain[0,0] = 0;
chain[0,1] = 0;
chain[0,2] = z_hel;
for i in range(1,N_chain) do
    chain[i,0] = chain[i-1,0] + sph2cart(chain_spacing, theta[i], phi[i])[0];
    chain[i,1] = chain[i-1,1] + sph2cart(chain_spacing, theta[i], phi[i])[1];
    chain[i,2] = chain[i-1,2] - sph2cart(chain_spacing, theta[i], phi[i])[2];
end
chain[:,3] = theta;
chain[:,4] = phi;
```

Algorithm 8: check_valid: test validity with boundary conditions

input:

- _chain_ (matrix)

output:

- valid (boolean true/false)

```
valid = True;
for i in range(len(_chain_)) do
    chain_x = _chain_[i,0];
    chain_y = _chain_[i,1];
    chain_z = _chain_[i,2];
    chain_s = sqrt(chain_x**2 + chain_y**2);
    if ((chain_z < (chain_s-R)/tan(angle_pore)) and (chain_z >= 0)) then
        valid = False;
    end
    if ((-0.1 < chain_z < 0.1) and (chain_s > R)) then
        valid = False;
    end
    if (chain_z > z_hel) then
        valid = False;
    end
    if _chain_[-1,2] > 0 then
        valid = False;
    end
end
```

Algorithm 9: snap_xyz: Snap coordinate to nearest meshgrid point

input:

- xyz (array)

output:

- snap_index (-)

```
error = 0.2;
xyz = round(xyz,3);
val = round(val,3);
snap_index = where((xyz > (val - error)) and (xyz < (val + error)));
```

Algorithm 10: create_charge_chain: Map charges to chain array

input:

- Charge q (in terms of e) for:
  - template ssDNA
  - linker
  - peptide
  - phosphorylated amino acid
  - threading ssDNA
- e (C)

output:

- charges (array)

```
charges = ones((N_chain))*10;
temp = round((11/23)*N_chain);
link = round((1/23)*N_chain);
pep = round((2/23)*N_chain);
p_pep = round((2/23)*N_chain);
thread = round((4/23)*N_chain);
charges[0:temp-1] = q_tem * e;
charges[temp-1:temp+link-1] = q_lin * e;
charges[temp+link-1:temp+link+pep-1] = q_pep * e;
charges[temp+link+pep-1:temp+link+pep+p_pep-1] = q_phos * e;
charges[temp+link+pep+p_pep-1:temp+link+2*pep+p_pep-1] = q_pep * e;
charges[temp+link+2*pep+p_pep-1:temp+link+2*pep+p_pep+link-1] = q_lin * e;
charges[temp+link+2*pep+p_pep+link-1:] = q_thread * e;
```

Algorithm 12: E_elecstat: Calculate electro-static energy of chain

input:

- _chain_ (matrix)
- e (C)
- V_phi (matrix)
- charges (array)

output:

- E (-)

```
q_template = -1 ;
q_threading = q_template ;
q_linker = 0 ;
q_peptide = 0 ;
q_phospho = -1 or 3 for pIRS2 and -5 for p2IRS2 ;
charges = create_charge_chain(q_template, q_linker, q_peptide, q_phospho,q_threading);
E = 0;
for i in range(len(_chain_)) do
    E += charges[i] * V_phi[snap_xyz(z,_chain_[i,2]),;
    snap_xyz(s,sqrt(_chain_[i,0]**2+_chain_[i,1]**2))];
end
```

A.5. Full Code

A.5.1. MATLAB LiveScript code

Event Classification - Kostas Andreadis (BEP)

Import processed data folders

```
% Import processed data folders
A='C:\Users\kandr\Documents\# TU-DELFT\# BEP\MATLAB\processeddata';
folders={};
A_folders = dir(A);
filesep = '\';

% Insert string to be contained in data folder
search_folder = "IRS2";

% Filter folder list by search term
for xx=1:length(A_folders)
    if xx>=3 & contains(A_folders(xx).name,search_folder)
        name = A_folders(xx).name;
        folders=[folders,append(A,filesep,name)];
    end
end
```

Classify events using categories and qualities

```
% Classify events using categories and qualities
eventclassifier(folders,'downsamplerate',1);
```

```
Loading and downsampling folder 1/1. Please wait.
```

```

% Import event.mat, reduced.mat
events_dir = [folders{1} filesep 'event.mat'];
reduced_dir = [folders{1} filesep 'reduced.mat'];
load(events_dir, 'event');
load(reduced_dir, 'reduced');

disp("Done with classification.")
```

```
Done with classification.
```

Filter events by category

```
% Insert category of interest
search_category = 1;
disp("Category of interest: " + search_category+ ". Please wait...");
```

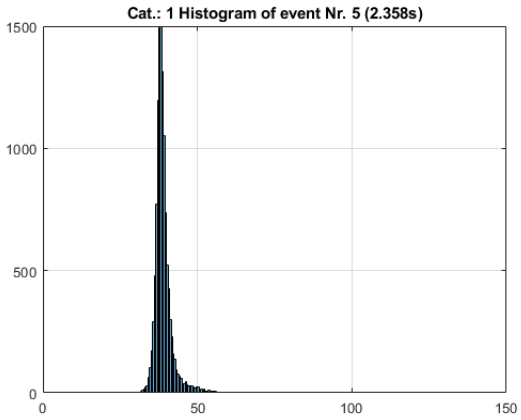
```
Category of interest: 1. Please wait...
```

```

% Pre-allocate variables
cat_indeces = [];
cat_sizes = [];
cat_count = 0;

% Count, index and measure length of events matching search term
for xx=1:length(event.category)
    if event.category(xx) == search_category
        cat_count = cat_count + 1;
        cat_indeces = [cat_indeces,xx];
        length_event = size(reduced.data(event.eventStartNdx(xx):event.eventEndNdx(xx)));
        cat_sizes = [cat_sizes,length_event(2)];
    end
end

% Pre-allocate matrix for higher efficiency
cat_cropped_data = NaN(cat_count,max(cat_sizes));

% Fill matrix with cropped event data
for xx=1:cat_count
    doi = reduced.data(event.eventStartNdx(cat_indeces(xx)):event.eventEndNdx(cat_indeces(xx)));
    for yy=1: cat_sizes(xx)
        cat_cropped_data(xx,yy) = doi(yy);
    end
end

disp("Done! Amount of events filtered: " + cat_count + " for category "+ search_category);
```

```
Done! Amount of events filtered: 5 for category 1
```

Plot events filtered by category

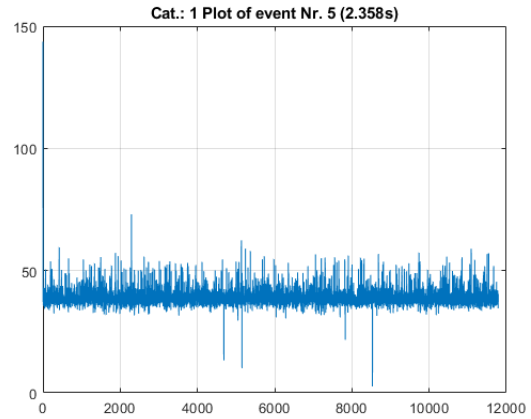
```
% plot(reduced.vdata)
% ylim([-200,200]);
% xlim([0,1e6]);

% Plot Histogram of Data
cat_hist_select = 2;
cat_hist = cat_cropped_data(cat_hist_select,:);
figure();
histogram(cat_hist);
grid on;
title("Cat.: "+search_category+" Histogram of event Nr. "+cat_indeces(cat_hist_select)+" ("+round(event.eventDur(cat_indeces(cat_hist
xlim([0,150]);
```




```
cat_mu = nanmean(cat_hist); % Mean after removing NaN values
cat_sigma = nanstd(cat_hist); % Standard Deviation after removing NaN values

% disp(cat_mu);
% disp(cat_sigma);

disp("Event Nr."+cat_indeces(cat_hist_select)+" | Mean: " + cat_mu + ", StdDev: " + cat_sigma);
```

```
Event Nr.5 | Mean: 38.9191, StdDev: 3.0278
```

```
% Plot Graph of Data
cat_plot_select = 2;
figure();
plot(cat_cropped_data(cat_plot_select,:));
grid on;
title("Cat.: "+search_category+" Plot of event Nr. "+cat_indeces(cat_plot_select)+" ("+round(event.eventDur(cat_indeces(cat_plot_sele
```



Filter events by quality

```
% Insert quality of interest
search_quality = 3;
disp("Quality of interest: " + search_quality+ ". Please wait...");
```

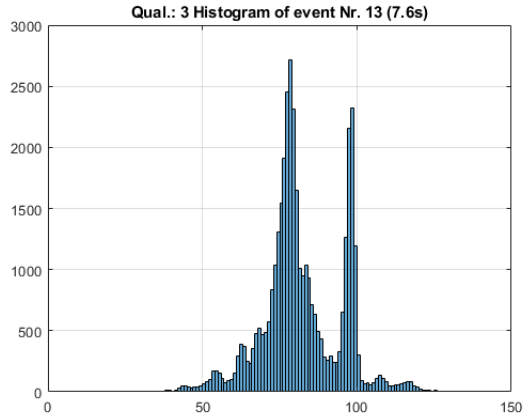
```
Quality of interest: 3. Please wait...
```

```
% Pre-allocate variables
qual_indeces = [];
qual_sizes = [];
qual_count = 0;

% Count, index and measure length of events matching search term
for xx=1:length(event.quality)
    if event.quality(xx) == search_quality
        qual_count = qual_count + 1;
        qual_indeces = [qual_indeces,xx];
        length_event = size(reduced.data(event.eventStartNdx(xx):event.eventEndNdx(xx)));
        qual_sizes = [qual_sizes,length_event(2)];
    end
end

% Pre-allocate matrix for higher efficiency
qual_cropped_data = NaN(qual_count,max(qual_sizes));

% Fill matrix with cropped event data
for xx=1:qual_count
    doi = reduced.data(event.eventStartNdx(qual_indeces(xx)):event.eventEndNdx(qual_indeces(xx)));
    for yy=1: qual_sizes(xx)
        qual_cropped_data(xx,yy) = doi(yy);
    end
end

disp("Done! Amount of events filtered: " + qual_count+ " for quality " + search_quality);
```

```
Done! Amount of events filtered: 5 for quality 3
```

Plot events filtered by quality

```
% plot(reduced.vdata)
% ylim([-200,200]);
% xlim([0,1e6]);

% Plot Histogram of Data
qual_hist_select =1;
qual_hist = qual_cropped_data(qual_hist_select,:);

figure();
histogram(qual_hist);
grid on;
title("Qual.: "+search_quality+" Histogram of event Nr. "+qual_indeces(qual_hist_select)+" ("+round(event.eventDur(qual_indeces(qual_
xlim([0,150]);
```




```
qual_mu = nanmean(qual_hist); % Mean after removing NaN values
qual_sigma = nanstd(qual_hist); % Standard Deviation after removing NaN values

disp("Event Nr."+qual_indeces(qual_hist_select)+" | Mean: " + qual_mu + ", StdDev: " + qual_sigma);
```

```
Event Nr.13 | Mean: 82.271, StdDev: 12.793
```

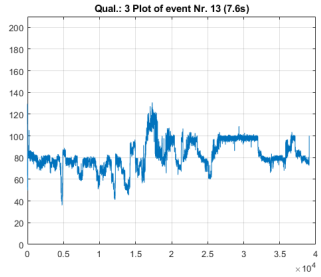
```
disp(qual_mu);
```

```
    82.2710
```

```
disp(qual_sigma);
```

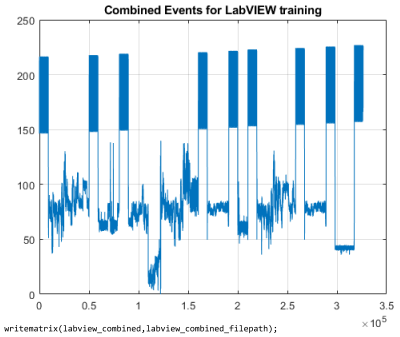
```
    12.7930
```

```
% Plot Graph of Data
qual_plot_select = 1;
figure();
plot(qual_cropped_data(qual_plot_select,:));
grid on;
ylim([0,210])
title("Qual.: "+search_quality+" Plot of event Nr. "+qual_indeces(qual_plot_select)+" ("+round(event.eventDur(qual_indeces(qual_plot_
```



Save Events of Interest for LabVIEW training

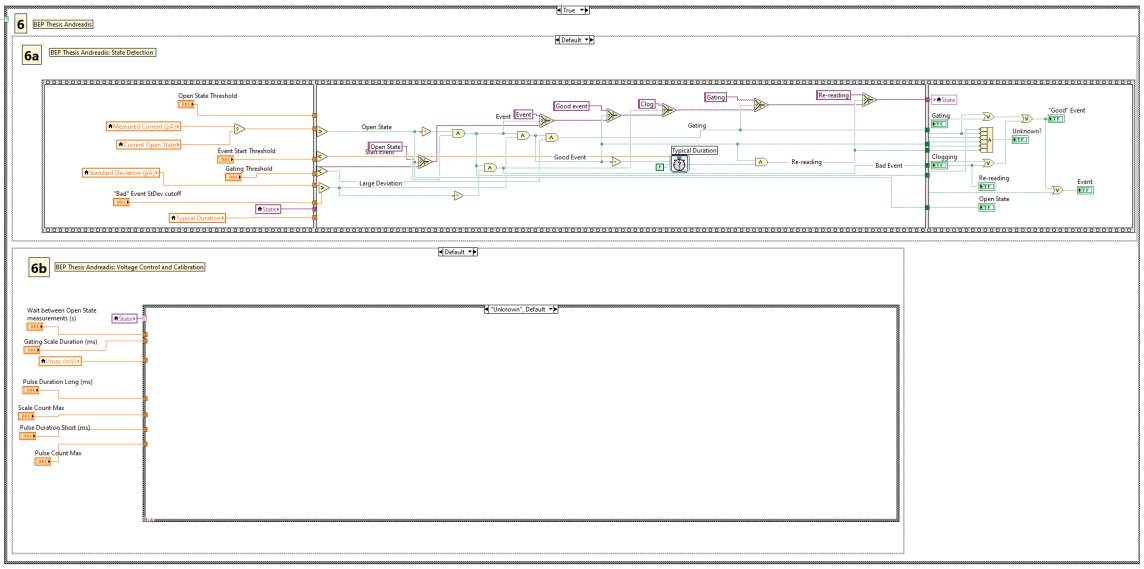
```
% Events of interest
event_select_index = fliplr([3,9,13,7,9,15,20,28]);
labview_combined_filepath = 'C:\Users\kandr\Documents\# TU-DELFT\# BEP\LabVIEW\training_sequence03.csv';

% Extracted and repeated open state sequence
openstate_extracted = reduced.data(event.eventEndNdx(25)+2500:event.eventEndNdx(25)+3400);
open_state_repeat = 10;
openstate = repmat(openstate_extracted,1,open_state_repeat);
open_state_increase = 1.3;

% Initialise Combined Data Array with Open State
labview_combined = [openstate];
% Plot extracted open state
% figure();
% plot(openstate_extracted);
% xlim([0,length(openstate_extracted)]);
% title("Extracted Open State");

% Create Combined Array
for xx=1:length(event_select_index)
    % Find Event of interest
    event_select = abs(reduced.data(event.eventStartNdx(event_select_index(xx)):event.eventEndNdx(event_select_index(xx))));

    % Exclude interventions by lab technician
    fip_indices = find(event_select>150);
    event_select(fip_indices) = [];

    % Add Event of interest
    labview_combined = [labview_combined, event_select];

    % Add Open State
    labview_combined = [labview_combined, openstate+xx*open_state_increase];
end

figure();
plot(labview_combined);
grid on;
% ylim([0,250]);
title("Combined Events for LabVIEW training");
```



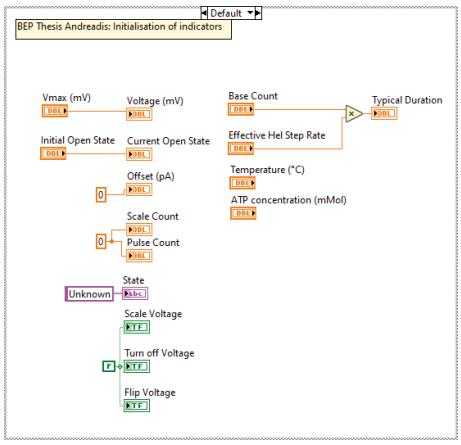

```
writematrix(labview_combined,labview_combined_filepath);
```

A.5.2. LabVIEW block diagram code

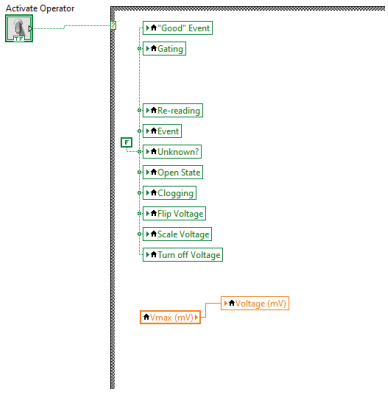
General code overview:



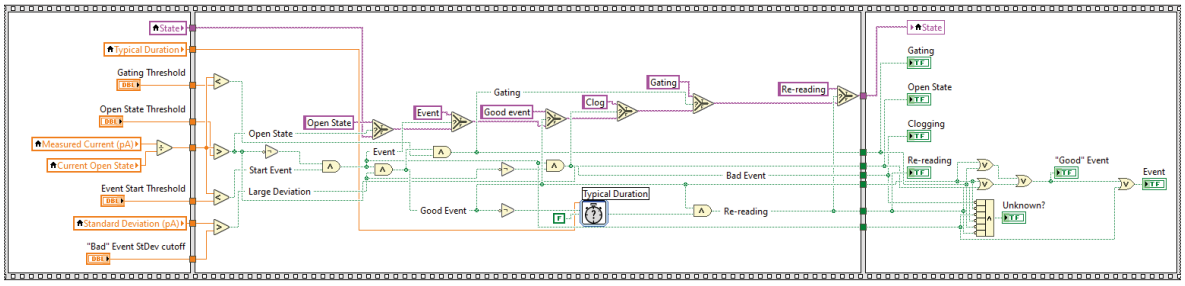
Initialisation of parameters:



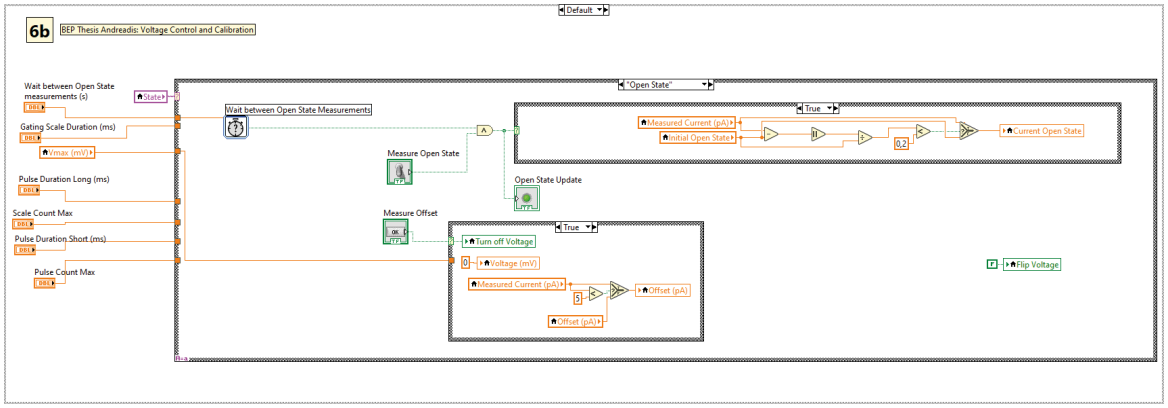
Initialisation of state and voltage values:

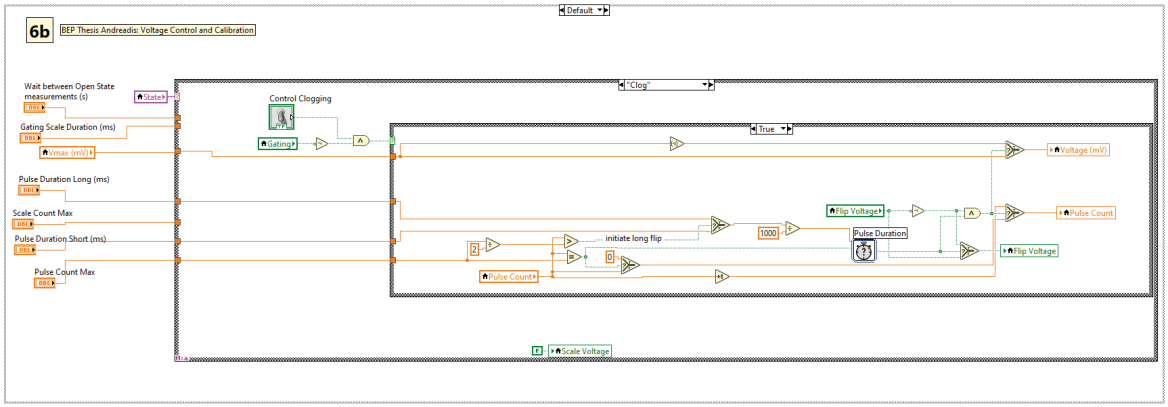


State detection machine:

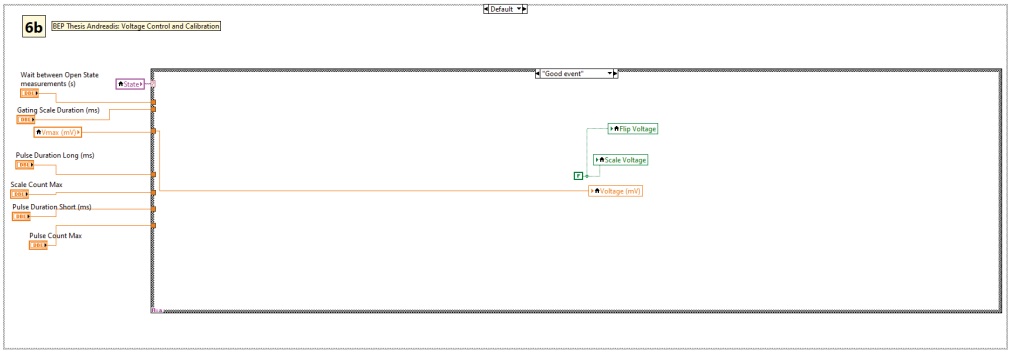


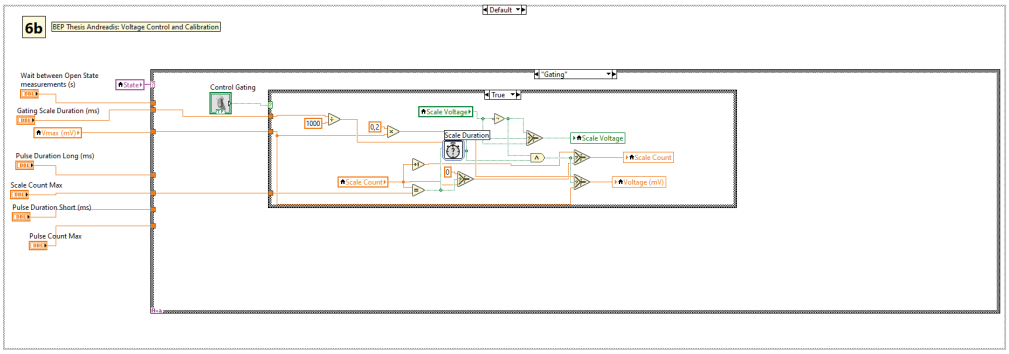
“Open Sate” and “clogging” voltage control:



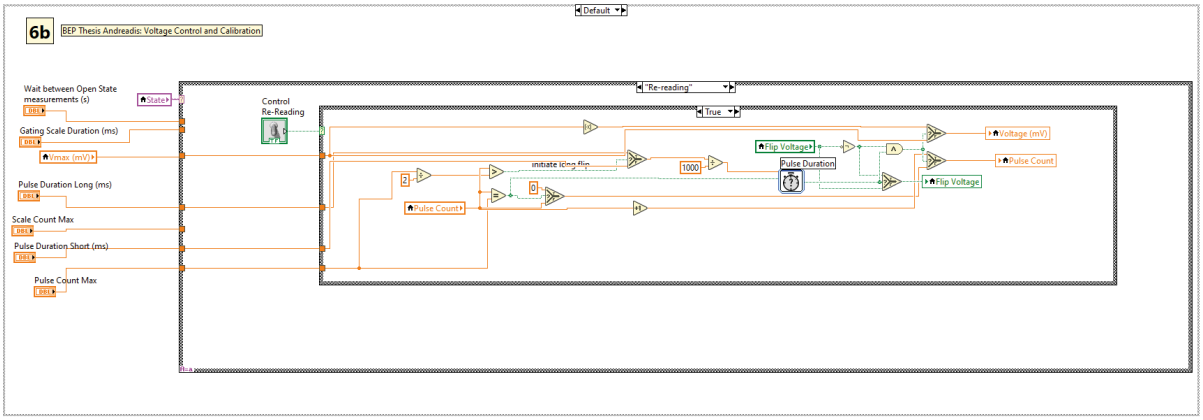


“Good” event and “gating” voltage control:

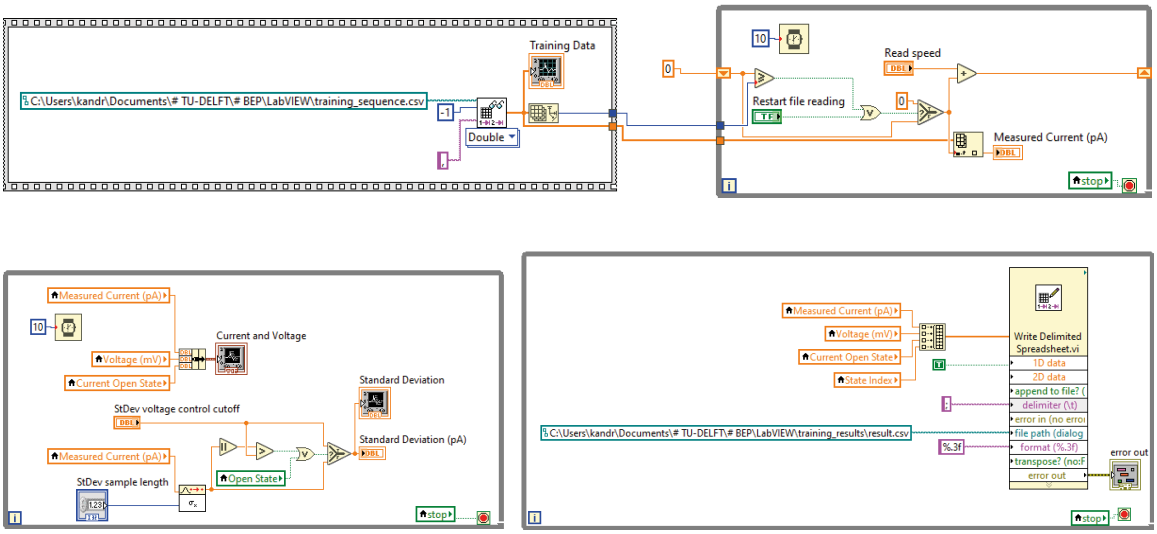




“Re-reading” voltage control:



Additional simulation testing environment code:



A.5.3. Python code

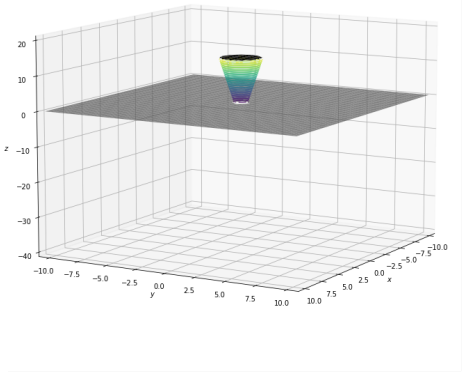
BEP Andreadis 2022 | Simulation of a (non-) charged probe

Import required libraries

In [1]:
```
# Import Numerical and Plotting Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d
from mpl_toolkits.mplot3d import Axes3D
import warnings
import time
from matplotlib import animation, rc
from IPython.display import HTML
warnings.filterwarnings("ignore", category=UserWarning)

# %matplotlib notebook  # if interactive 3D plot is desired
# Set resolution to 200 dpi for thesis media creation
plt.rcParams['figure.dpi'] = 200
```

Define and visualise 3D boundary conditions

In [2]:
```
# Geometric Parametes
R = 0.5                         # radius of pore
angle_pore = np.radians(5)  # angle of MspA in degrees
z_hel = 13                      # 'height' of helicase above membrane = fixed start
 point of chain

# 3D plotting parameters
N = 200 # amount of points to plot
xmin = -7*1.5
xmax =  7*1.5
ymin = xmin
ymax = xmax
zmin = -20*2
zmax =  20

# Define x and y range of points in space
x_range    = np.linspace(xmin,xmax,N)
y_range    = np.linspace(ymin,ymax,N)
X,Y        = np.meshgrid(x_range,y_range)


# Define boundary conditions of mspa, membrane and helicase
Z_mspa     = (np.sqrt(X**2 + Y**2) - R)/np.tan(angle_pore)
Z_mspa[Z_mspa >= z_hel] = None # crop cone above helicase
Z_mspa[Z_mspa < 0]     = None # crop cone below membrane

Z_membrane = np.zeros((N,N))
Z_membrane[R > np.sqrt(X**2+Y**2)] = None # crop pore out of membrane

Z_helicase = np.ones((N,N))*z_hel
Z_helicase[ (z_hel*np.tan(angle_pore) + R) < np.sqrt(X**2+Y**2)] = None # simple
plane above mspa

# Plot findings
fig = plt.figure(figsize=(12,12))
ax = plt.gca(projection = "3d")

ax.plot_surface(X,Y, Z_membrane,color="0.4",alpha=0.7, label='membrane')
ax.contour3D(X,Y, Z_mspa,25,alpha=0.7, label='MspA')
ax.plot_surface(X,Y, Z_helicase,color="k",alpha=1, label='Helicase')
ax.set_xlim(xmin,xmax)
ax.set_ylim(ymin,ymax)
ax.set_zlim(zmin,zmax)
ax.set_xlabel("$x$")
ax.set_ylabel("$y$")
ax.set_zlabel("$z$")
ax.view_init(10, 30)
# ax.legend()
plt.savefig('figures/boundary_conditions.png')
plt.show()

# plt.figure(figsize=(8,8))
# plt.plot(range(10),label='membrane')
# plt.plot(range(10),label='MspA')
# plt.plot(range(10),label='helicase')
# plt.legend()
# plt.savefig('figures/boundary_conditions_legend.png')
# plt.show()
```



Solving the Laplace Equation in cylindrical coordinates

$$-\frac{\rho(s,\phi,z)}{\epsilon_0} = \nabla^2\phi = \frac{1}{s}\frac{\partial}{\partial s}(s\frac{\partial\phi}{\partial s}) + \frac{1}{s^2}\frac{\partial^2\phi}{\partial\phi^2} + \frac{\partial^2\phi}{\partial z^2} = \frac{1}{s}\frac{\partial\phi}{\partial s} + \frac{\partial^2\phi}{\partial s^2} + \frac{\partial^2\phi}{\partial z^2}$$

Due to rotational symmetry, $\frac{\partial^2\phi}{\partial\phi^2} = 0$ and $\phi(s,\phi,z) = \phi(s,z)$

$\rho(s,\phi,z)$ being the fixed charges and ϕ the electro-static potential

Finite Difference Method (Central)

$$\nabla^2\phi \approx \frac{1}{s}\frac{\phi(s+a,z)-\phi(s,z)}{a} + \frac{\phi(s+a,z)+\phi(s-a,z)+\phi(s,z+a)+\phi(s,z-a)-4\phi(s,z)}{a^2}$$

taking $a = 1$ leads to

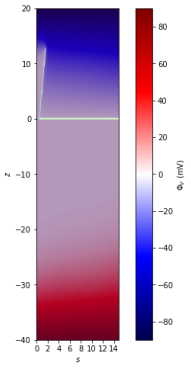
$$\phi_{N+1}(s,z) = \frac{\phi_N(s+a,z)+\phi_N(s-a,z)+\phi_N(s,z+a)+\phi_N(s,z-a)+\frac{1}{s}\phi_N(s+a,z)}{4+\frac{1}{s}}$$

- Jacobi method for solving linear partial differential equations (PDEs), uses initial guess and converges to solution
- Gauss-Seidel method speeds up Jacobi method by using already calculated values for ϕ
- Successive over-relaxation (SOR) increases Gauss-Seidel even more by using $\phi_{new}(x,y,z) = \phi_{old}(x,y,z) + \omega\delta(x,y,z)$
- ω being the S.O.R. parameter, speeds up if above 1, however does not converge any more if bigger than 2
- $\delta(x,y,z)$ being the difference between the newly calculated potential and the one before

In [3]:
```
'''
input:
    N_plot_points (-)
    xmin,xmax,ymin,ymax,zmin,zmax (-)
    V (mV)
    target_accuracy (-)
    SOR_parameter (-)

output:
    Electric potential V_phi (matrix: N_plot_points x N_plot_points)
'''

# Amount of plotting points
N = 200

# Cylindrical and cartesian coordinates
smin = 0
smax = np.sqrt(xmax**2+ymax**2)
s = np.linspace(smin,smax,N)
x = np.linspace(xmin,xmax,N)
y = np.linspace(ymin,ymax,N)
z = np.linspace(zmin,zmax,N)

# Voltage applied in V
V = 0.180

# Desired accuaracy of simulation
target_accuracy = 1e-4

# Initialise electric potential matrix
V_phi = np.zeros([N,N])
delta = V_phi.copy()

# Initialise fixed boundary conditions
boundary_membrane_mspa = np.zeros((N,N))
membrane_z_N  = int(N*(abs(zmin)/(zmax + abs(zmin))))
boundary_membrane_mspa[membrane_z_N,7:] = 1
for zi in range(N):
    for si in range(N):
        if ((round(z[zi]) == round((s[si] - R)/np.tan(angle_pore))) and (z[zi] <
= z_hel) and (z[zi] > 0)):
            boundary_membrane_mspa[zi,si] = 1
boundary_membrane_mspa = np.flipud(boundary_membrane_mspa) #flip matrix (needed
 due to abnormal errors of matplotlib)

# Initialise phi difference (for convergence check)
delta_max = 1
delta_max_list = []

SOR_parameter = 1.8 # successive over-relaxation parameter for faster convergenc
e, determined by trial-and-error
print("Using SOR parameter ω =",round(SOR_parameter,2))
start = time.time()
while delta_max > target_accuracy: # Laplace electric potential solver
    # this loop may take a while depending on target_accuracy and the dimension
 N
    for i in range(N):
        for j in range(N):
            # Add voltage drop across membrane
            if   i == 0:   V_phi[i,j] = -V/2
            elif i == N-1: V_phi[i,j] =  V/2
            # if s = 0 is reached, reflect and use 2nd value instead
            elif j == 0:
                V_phi[i,j]  = V_phi[i,1]
            # if s = smax is reached, reflect again
            elif j == N-1:
                delta[i,j] = (V_phi[i+1,j]+V_phi[i-1,j]+V_phi[i,j-1]+V_phi[i,j-1
] + V_phi[i,j-1]/s[j])/(4+1/s[j]) - V_phi[i,j]
                V_phi[i,j]  += SOR_parameter*delta[i,j]
            # if boundary is reached, set potential to 0
            else:
                if boundary_membrane_mspa[i,j]:
                    V_phi[i,j] = 0
                else:
                    delta[i,j] = (V_phi[i+1,j]+V_phi[i-1,j]+V_phi[i,j+1]+V_phi[i
,j-1] + V_phi[i,j+1]/s[j])/(4+1/s[j]) - V_phi[i,j]
                    V_phi[i,j]  += SOR_parameter*delta[i,j]
    # calculate max phi difference to test convergence
    delta_max = np.max(np.abs(delta))
    delta_max_list.append(delta_max)
    print("N_iter %d delta_max %e\r" % (len(delta_max_list), delta_max), end='')
end = time.time()
print()
print("Finished! Elapsed time:",str(round(end-start,1))+"s")
```

```
Using SOR parameter ω = 1.8
N_iter 168 delta_max 9.943225e-05
Finished! Elapsed time: 27.7s
```

In [5]:
```
# Plot phi_V
plt.figure(figsize=(8, 8))
plt.imshow(V_phi*1000,extent = [smin,smax,zmin,zmax],cmap='seismic')
plt.colorbar(label='$\Phi_{V}$ (mV)')
plt.imshow(boundary_membrane_mspa,extent = [smin,smax,zmin,zmax],alpha=0.4)
# plt.imshow(boundary_membrane_mspa !=0,cmap="binary",extent = [smin,smax,zmin,z
max],alpha=1)
plt.xticks(np.arange(smin,smax+1,2))
plt.xlabel("$s$")
plt.ylabel("$z$")
# plt.savefig('figures\laplace_s-z_boundary.png')
plt.savefig('figures\laplace_s-z.png')
plt.show()
```

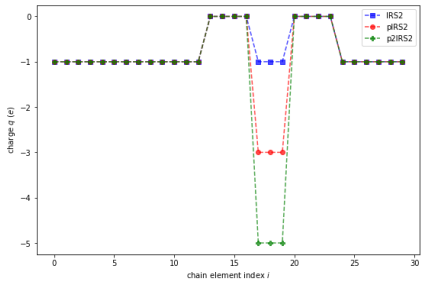


(charge) chain creation, validity check and energy calculation

In [6]:
```
# Define relevant variables
e = 1.60217663e-19  # charge of an electron in C
k_B = 1.380649e-23  # Boltzmann costant in J/K
T   = 273 + 30      # room temperature in K
beta = 1/(k_B*T)

N_chain = 30        # number of chain elements
chain_spacing = 1.2 # space betwen chain elements

def sph2cart(r, theta, phi): # spherical to cartesian coordinate transformation
    '''
    input:
        r (-)
        theta (radians)
        phi (radians)
    output:
        x,y,z
    '''
    return [r * np.sin(theta) * np.cos(phi),r * np.sin(theta) * np.sin(phi),r *
np.cos(theta)]

def create_chain(theta, phi): # Create ideal 3D chain
    '''
    input:
        theta (array in radians)
        phi (array in radians)
    output:
        chain (matrix):
        1st column: x coordinates
        2nd column: y coordinates
        3rd column: z coordinates
        4th column: theta angles
        5th column: phi angles
    '''
    chain = np.zeros((N_chain, 5))
    chain[0,0] = 0
    chain[0,1] = 0
    chain[0,2] = z_hel
    for i in range(1,N_chain):
        chain[i,0] = chain[i-1,0] + sph2cart(chain_spacing, theta[i], phi[i])[0]
        chain[i,1] = chain[i-1,1] + sph2cart(chain_spacing, theta[i], phi[i])[1]
        chain[i,2] = chain[i-1,2] - sph2cart(chain_spacing, theta[i], phi[i])[2]
    chain[:,3] = theta
    chain[:,4] = phi
    return chain

def check_valid(_chain_): # test validity with boundary conditions
    '''
    input:
        _chain_ (matrix)
    output:
        valid (boolean true/false)
    '''
    valid = True
    for i in range(len(_chain_)):
        chain_x = _chain_[i,0]
        chain_y = _chain_[i,1]
        chain_z = _chain_[i,2]
        chain_s = np.sqrt(chain_x**2 + chain_y**2)
        if ((chain_z < (chain_s-R)/np.tan(angle_pore)) and (chain_z >= 0)):
            valid = False
        if ((-0.1 < chain_z < 0.1) and (chain_s > R)):
            valid = False
        if (chain_z > z_hel):
```

```
            valid = False
    if _chain_[-1,2] > 0:
        valid = False
    return valid

def snap_xyz(xyz,val): # Snap coordinate to nearest meshgrid point
    '''
    input:
        xyz (array)
    output:
        snap_index (-)
    '''
    error = 0.2
    xyz = np.round_(xyz,3)
    val = round(val,3)
    snap_index = np.where((xyz > (val - error)) & (xyz < (val + error)))
#     print(val,xyz)
    return snap_index[0][0]

def create_charge_chain(q_tem, q_lin, q_pep, q_phos,q_thread): # Map charges to
 chain array
    '''
    input:
        Charge q (in terms of e) for:
        template ssDNA
        linker
        peptide
        phosphorylated amino acid
        threading ssDNA
        e (C)
    output:
        charges (array)
    '''
    charges  = np.ones((N_chain))*10

    temp     = round((11/23)*N_chain)
    link     = round((1/23)*N_chain)
    pep      = round((2/23)*N_chain)
    p_pep    = round((2/23)*N_chain)
    thread   = round((4/23)*N_chain)
    charges[0:temp-1]                                                = q_tem * e
    charges[temp-1:temp+link-1]                                      = q_lin * e
    charges[temp+link-1:temp+link+pep-1]                             = q_pep * e
    charges[temp+link+pep-1:temp+link+pep+p_pep-1]                   = q_phos * e
    charges[temp+link+pep+p_pep-1:temp+link+2*pep+p_pep-1]           = q_pep * e
    charges[temp+link+2*pep+p_pep-1:temp+link+2*pep+p_pep+link-1]    = q_lin * e
    charges[temp+link+2*pep+p_pep+link-1:]                           = q_thread *
e
    return charges

q_template   = -1
q_threading  = q_template
q_linker     = 0
q_peptide    = 0
q_phospho    = -1 #-3 for pIRS2 and -5 for p2IRS2

charges    = create_charge_chain(q_template, q_linker, q_peptide, q_phospho,q_th
reading)
charges_p  = create_charge_chain(q_template, q_linker, q_peptide, -3,q_threading
)
charges_p2 = create_charge_chain(q_template, q_linker, q_peptide, -5,q_threading
)

plt.figure(figsize=(9,6))
```

```
plt.plot(charges/e, "s--",color='b',label="IRS2",alpha=0.7)
plt.plot(charges_p/e, "o--",color='r',label="pIRS2",alpha=0.7)
plt.plot(charges_p2/e, "P--",color='g',label="p2IRS2",alpha=0.7)
plt.xlabel("chain element index $i$")
plt.ylabel("charge $q$ ($e$)")
plt.legend()
plt.savefig("figures\charges.png")
plt.show()

def E_elecstat(_chain_):
    # Calculate electro-static energy of chain
    '''
    input:
        _chain_ (matrix)
        e (C)
        V_phi (matrix)
        charges (array)
    output:
        E (-)
    '''
    E = 0
    for i in range(len(_chain_)):
        E += charges[i]*V_phi[snap_xyz(z,_chain_[i,2]), snap_xyz(s,np.sqrt(_chai
n_[i,0]**2+_chain_[i,1]**2))]
    return E
```

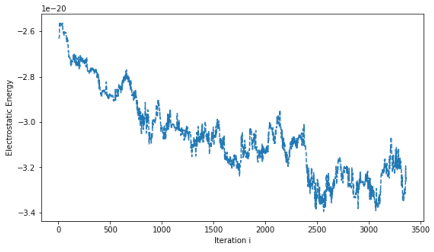


Metropolis Algorithm

In [7]:
```
'''
input:
    N_iterations (-)
    initial [θ] and [φ] (both arrays in radians)
    kB (J/K)
    T (K)
    check_valid (function)
    E (function)
output:
    accepted/refused states (array of arrays)
'''
theta_record_all = []
phi_record_all = []
chain_all = []
energy_all = []

np.random.seed(0)

theta_record = np.zeros(N_chain)
phi_record = np.zeros(N_chain)

N_iter_tot = 5000
A = 1

for N_iter in range(N_iter_tot):
    delta_theta_record = np.radians(A*np.random.normal(size = N_chain))
    delta_phi_record   = np.radians(A*np.random.normal(size = N_chain))
    chain_original = create_chain(theta_record, phi_record)
    chain_shifted  = create_chain(theta_record+delta_theta_record, phi_record+de
lta_phi_record)
    E_original = E_elecstat(chain_original)
    E_shifted = E_elecstat(chain_shifted)
    P_accept = np.exp(beta*(E_shifted - E_original))

    for i in range(N_chain):
        if theta_record[i] == 0 or theta_record[i] == 0:
            P_accept *= 1 + (np.cos(theta_record[i] + delta_theta_record[i]))/np
.cos(theta_record[i]) #taylor expansion around theta = 0
        else:
            P_accept *= (np.sin(theta_record[i] + delta_theta_record[i]))/np.sin
(theta_record[i])

    sample_test = np.random.uniform(0,1)
    Pt = min(P_accept,1)
    if sample_test < Pt and check_valid(chain_shifted): #accept
        theta_record += delta_theta_record
        phi_record   += delta_phi_record
        theta_record_all.append(theta_record)
        phi_record_all.append(phi_record)
        chain_all.append(chain_shifted)
        energy_all.append(E_shifted)
    else:
        pass

    print("Progress: {}%".format(round(100*N_iter/N_iter_tot)), end='\r')

theta_record_all = np.asarray(theta_record_all)
phi_record_all = np.asarray(phi_record_all)
chain_all = np.asarray(chain_all)
optimal_chain = chain_all[-1]
energy_all = np.asarray(energy_all)
print()
print("Finished!")
```

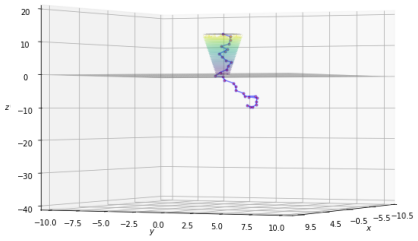
```
Progress: 100%
Finished!
```

In [8]:
```
plt.figure(figsize=(9,5))
plt.plot(energy_all,'--')
plt.xlabel("Iteration i")
plt.ylabel("Electrostatic Energy")
plt.savefig('figures\energy_vs_iteration.png')
plt.show()
```



Visualise converged states

In [9]:
```
fig = plt.figure(figsize=(10,10))
ax = plt.gca(projection = "3d")

ax.plot_surface(X,Y, Z_membrane,color="0.6",alpha=0.6)

plot_selected = True

if plot_selected:
#     chain_select = chain_all[-1]
    chain_select = chain_all[np.argmin(energy_all)]

    ax.plot(chain_select[:,0],chain_select[:,1],chain_select[:,2],color='b',alph
a=0.5,marker='o',markersize=3,mfc='r')
#     ax.scatter3D(chain_select[:,0],chain_select[:,1],chain_select[:,2],s=10,al
pha=0.5,c = charges, cmap='Spectral')
else:
    for i in range(len(chain_all)):
        ax.plot(chain_all[i][:,0],chain_all[i][:,1],chain_all[i][:,2],color='b',
alpha=0.1,marker='o',markersize=1,mfc='r')
#         ax.scatter3D(chain_all[i][:,0],chain_all[i][:,1],chain_all[i][:,2],s=
5,alpha=0.5,c = charges, cmap='Spectral')

ax.contour3D(X,Y, Z_mspa,30,alpha=0.4)
ax.plot_surface(X,Y, Z_helicase,color="0.4",alpha=0.6)

ax.set_xlim(xmin,xmax)
ax.set_ylim(ymin,ymax)
ax.set_zlim(zmin,zmax)
ax.set_xticks(np.arange(xmin,xmax+1,5))
ax.set_xlabel("$x$")
ax.set_ylabel("$y$")
ax.set_zlabel("$z$")
ax.view_init(0, 25)
if plot_selected:
    plt.savefig('figures/plot_selected.png')
else:
    plt.savefig('figures/all_state_visited.png')
plt.show()
```



Save Metropolis algorithm progress as .gif

In [10]:
```
plt.rcParams['figure.dpi'] = 100
fig = plt.figure(frameon=False)
fig.set_size_inches(10,8)
ax = fig.add_subplot(projection='3d')
line, = ax.plot([],[],[],marker='o',markersize=3,mfc='r',label='0')
ax.plot_surface(X,Y, Z_membrane,color="0.6",alpha=0.6)
ax.contour3D(X,Y, Z_mspa,30,alpha=0.2)
ax.plot_surface(X,Y, Z_helicase,color="0.4",alpha=0.6)

ax.set_xlim(xmin,xmax)
ax.set_ylim(ymin,ymax)
ax.set_zlim(zmin,zmax)
ax.set_xlabel("$x$")
ax.set_ylabel("$y$")
ax.set_zlabel("$z$")
ax.view_init(1, 25)
fig.subplots_adjust(left=0, bottom=0, right=1, top=1, wspace=None, hspace=None)

def init():
    line.set_data(chain_all[0][:,0],chain_all[0][:,1])
    line.set_3d_properties(chain_all[0][:,2])
    return line

frame_duration = 10 # in s
gif_duration_total = 1 # in s
total_frame_count = int(gif_duration_total/(frame_duration/1000))
print(total_frame_count)

def animate(i):
    i = int(i*len(chain_all)/total_frame_count)
    line.set_data(chain_all[i][:,0],chain_all[i][:,1])
    line.set_3d_properties(chain_all[i][:,2])
    line.set_label("Iteration i = {}".format(i))
    ax.legend(loc='upper center')
    return line

anim = animation.FuncAnimation(fig, animate, init_func=init,frames=total_frame_c
ount, interval=frame_duration)
anim.save('figures/metropolis_progress.gif',progress_callback=lambda i, n: print
("Progress: {}%".format(round(100*i/n)), end='\r'))
print()
print("Finsihed saving .gif animation!")
plt.close(fig)
plt.rcParams['figure.dpi'] = 200
```

```
MovieWriter ffmpeg unavailable; using Pillow instead.
```

```
100
Progress: 99%
Finsihed saving .gif animation!
```

A.6. BEP Thesis hour log

§ BEP Andreadis 2022 - Hours Log

Description	No. hours	Category	Date	Remarks	Status
Full Schedule for all 10 weeks	2.0	BEP planning	before begin	-	Done
Research Question formulation and Thesis LaTeX setup	2.5	Setup	before begin	-	Done
Initial Online Search & skim for/of relevant papers	5.0	Literature Study	before begin	-	Done
MATLAB refreshing courses online	2.0	BEP planning	before begin	-	Done
LabVIEW refreshing with study course material	3.0	BEP planning	before begin	-	Done
General Lab Introduction	1.0	Introduction	19/4/2022	-	Done
Ian explanation on experimental setup	2.0	Literature Study	19/4/2022	-	Done
Ian explanation on experimental setup	4.0	Literature Study	20/4/2022	-	Done
Ian explanation on theory	2.0	Literature Study	20/4/2022	-	Done
LabServant & co. campus card access rights	1.0	Setup	20/4/2022	-	Done
Reading 4 papers on theory and experiments	1.0	Literature Study	20/4/2022	-	Done
Cees Dekker Protein Sequencing Meeting	1.0	Meeting	21/4/2022	-	Done
Reading 4 papers on theory and experiments	2.0	Literature Study	21/4/2022	-	Done
Ian explanation on MATLAB workflow	2.0	Literature Study	21/4/2022	-	Done
MATLAB, LabVIEW, GitLab and EndNote setup	2.0	Setup	21/4/2022	-	Done
Reading 3 papers on extra theory (E.O. flow)	3.0	Literature Study	22/4/2022	-	Done
Initial MATLAB quality classification	1.0	Classification	22/4/2022	-	Done
MATLAB quality and category classification	4.0	Classification	25/4/2022	-	Done
MATLAB quality and category classification	5.0	Classification	28/4/2022	-	Done
Cees Dekker Lab Meeting	2.5	Meeting	28/4/2022	-	Done
MATLAB quality and category classification	3.0	Classification	29/4/2022	-	Done
LabVIEW initial test environment setup	2.0	Setup	29/4/2022	-	Done
LabVIEW operator design with flowcharts	1.0	Automation	2/5/2022	-	Done
LabVIEW Event detection and voltage control	3.0	Automation	2/5/2022	-	Done
LabVIEW Calibration	3.5	Automation	3/5/2022	-	Done
LabVIEW Event detection, control and calibration	7.0	Automation	4/5/2022	-	Done
Cees Dekker Protein Sequencing Meeting	1.0	Meeting	4/5/2022	-	Done
LabVIEW training with real data extracted and combined from MATLAB classification	7.0	Test	6/5/2022	-	Done
Henri original LabVIEW code explanation and cross-VI integration tips	1.0	Meeting	6/5/2022	he would like a modular state detection diagram for own (neural network based detection) research	Done
LabVIEW rewriting with modular structure and seperate loops	5.0	Automation	7/5/2022	-	Done
LabVIEW rewriting with modular structure and seperate loops	2.0	Automation	9/5/2022	-	Done
Setup DAQ simulator for LabVIEW compatibility	2.0	Setup	9/5/2022	-	Done
LabVIEW operator code integration into main.vi code	4.0	Automation	11/5/2022	-	Done
Initial test with DAQ and experimental set up	0.5	Test	11/5/2022	-	Done
Advanced MATLAB Classification	2.5	Classification	11/5/2022	-	Done
Start of Thesis Report Structuring	2.0	Thesis Report Writing	12/5/2022	-	Done
Test State Detection with real experimental DAQ set-up	2.0	Test	12/5/2022	membrane broke, however initial success for proof of concept	Done
Test State Detection and calibration with real experimental DAQ set-up	1.5	Test	12/5/2022	adaptions to code made live	Done
Update LabVIEW code after initial lab test run to improve amongst else timing issues	4.0	Automation	13/5/2022	open state update fixed, change all time wait functions to elapsed time functions, change StDev calculatior	Done
Update operator design diagram and other work done	1.0	Thesis Report Writing	17/5/2022	-	Done
Test State Detection and calibration with real experimental DAQ set-up	0.5	Test	17/5/2022	without experiment attached, however very effective and functional!	Done
Thesis Report Content Structuring	2.0	Thesis Report Writing	18/5/2022	-	Done
Thesis Report Theory Figure creation	1.0	Thesis Report Writing	18/5/2022	-	Done
Meeting with Henry in lab to discuss code	0.5	Meeting	18/5/2022	pay attention to StDev calculation --> exclude spikes!	Done
Thesis Report Theory Figure creation	2.5	Thesis Report Writing	19/5/2022	-	Done
Update LabVIEW code with stdev calculation, overall (visual) improvement and indicators	5.0	Automation	19/5/2022	-	Done
Meeting Cees Dekker Protein Sequencing meeting	1.0	Meeting	19/5/2022	-	Done
LabVIEW creation of simulation file input alternative	2.5	Automation	20/5/2022	made simulation using test file, optimised StDev calculation and file reading	Done
LabVIEW saving of state and suggestions	0.5	Automation	20/5/2022		Done
Ian meeting for next part of thesis	0.5	Meeting	20/5/2022	decided to replace noise analysis by charge and spring simulation, coming week perhaps labview tests in la	Done
Literature study on A.C. sequencing, other types of DNA/peptide force interactions	4.0	Literature Study	23/5/2022	-	Done
Python Simulation initial set up with simplifications	1.0	Force analysis on (charged) probe	23/5/2022	-	Done
Python Simulation langevin function modeling	3.5	Force analysis on (charged) probe	23/5/2022	-	Done
LabVIEW code simulation and operator finalization and commenting	4.0	Automation	24/5/2022	-	Done
MATLAB figure export of events of interest	1.5	Classification	24/5/2022	-	Done
create 3 training sequences and record LabVIEW response	0.5	Test	24/5/2022	-	Done
create 3 training sequences and record LabVIEW response	2.5	Test	25/5/2022	-	Done
Literature study on chain models for ssDNA and peptide	2.5	Force analysis on (charged) probe	25/5/2022	-	Done
Cees Dekker Lab Meeting	1.5	Meeting	25/5/2022	-	Done
Initial simplified model set up	1.5	Setup	25/5/2022	-	Done
Google Sheet and Python model development	2.0	Force analysis on (charged) probe	26/5/2022	-	Done
Thesis content writing	3.0	Thesis Report Writing	27/5/2022	-	Done
Electric force simulations in Python	3.0	Force analysis on (charged) probe	30/5/2022	introduced radial electric field in addition to linear field, vector arrow representation	Done
Thesis content writing	4.5	Thesis Report Writing	30/5/2022	wrote algorithms of labview code	Done
Electric and -osmotic force simulations in Python	7.0	Force analysis on (charged) probe	31/5/2022	addition of vestibule	Done
LabVIEW Lab Test with IRS2	0.5	Test	31/5/2022	open state value should adopt last value if too large and not initial open state value initial open state value should be live control clogging too fast, durations larger + begin waiting time measure offset only in open state, otherwise scale voltage == FALSE !!! StDev is sometimes large (because of noisy clog)	Done
Meeting Henry on Simulation	2.0	Meeting	31/5/2022	very helpful, simplified problem with analytical solution + metropolis algorithm	Done
Literature Study and Developement of formula's and boundary conditions (on paper and in	7.5	Force analysis on (charged) probe	1/6/2022	analytical approach: formula's and boundary conditions and computational set up with visualisation of bou	Done
Implementation of Henry Feedback on formula's and initial laplace simulations	7.0	Force analysis on (charged) probe	2/6/2022	laplace electrostatic potential calculation: where is potential??	Done
simulation of chain	10.0	Force analysis on (charged) probe	2/6/2022		Done
simulation of chain	3.0	Force analysis on (charged) probe	3/6/2022		Done
Cees Dekker Protein Sequencing Meeting	1.0	Meeting	3/6/2022	Henry Feedback: Implement first potential due to applied voltage without chain, and then add charges (thi	Done
Simulation of Chain with elecstat. Potential energy calculation	7.0	Force analysis on (charged) probe	4/6/2022	only solution for Laplace initial voltage, charge list created, lapalce solving adaptation to boundaries instea	Done
Simulation of Chain with elecstat. Potential energy calculation	2.0	Force analysis on (charged) probe	7/6/2022	visualised state with lowest potential energy from random sample collection with charges as markers	Done
Helmholtz calculation and Voltage boundary conditions	4.0	Force analysis on (charged) probe	8/6/2022		Done
Thesis figure Creation	3.5	Thesis Report Writing	9/6/2022		Done
Thesis figure Creation	2.5	Thesis Report Writing	9/6/2022		Done
Thesis report writing	4.0	Thesis Report Writing	10/6/2022		Done
Meeting Henry on Simulation	1.0	Force analysis on (charged) probe	10/6/2022		Done
Implementation of Henry's feedback on Laplace solving and Metropolis algorithm	4.0	Force analysis on (charged) probe	10/6/2022		Done
Implementation of Henry's feedback on Laplace solving and Metropolis algorithm	3.0	Force analysis on (charged) probe	11/6/2022	A changes plots significantly, thetha and phi initial guess is now 0,0	Done
Thesis report writing	4.0	Force analysis on (charged) probe	12/6/2022		Done
Metropolis solving and gif documentation thereof	5.5	Force analysis on (charged) probe	13/6/2022		Done
Thesis report writing	3.0	Thesis Report Writing	13/6/2022		Done
Thesis report writing	4.0	Thesis Report Writing	14/6/2022		Done
Thesis report writing	12.5	Thesis Report Writing	14/6/2022		Done
Thesis report writing	16.0	Thesis Report Writing	15/6/2022		Done
Thesis report writing	4.0	Thesis Report Writing	16/6/2022		Done
Thesis report writing	8.0	Thesis Report Writing	20/6/2022		Done
Thesis report writing	7.5	Thesis Report Writing	21/6/2022		Done
					Planned
					Planned
					Planned
					Planned
					Planned