Comparison of Simulated Extracellular Spikes from Pyramidal Neurons and Interneurons

by

Daniel Marelius Bjørnstad

Thesis

for the Degree of

Master of Science

Faculty of Mathematics and Natural Sciences
August 15th, 2016

Abstract

Duration of extracellular spikes has long been an indicator of whether the measured neuron is a pyramidal neuron or an interneuron. Using computational simulation of neural activity results from the article Pettersen & Einevoll (2008) was replicated. To investigate how spike amplitude and width can classify pyramidal neurons and interneurons, measurements were done in 260 biophysical models from the Blue Brain Project. 1000 electrodes were simulated for each model and positioned randomly around the soma to mimic experimental procedures. Probability distributions of the measured widths and amplitudes were compared using AOC curves and a overlap measure. The spike width definitions were found to considerably effect how well classification can be done with the peak-to-peak spike width definition providing the best results. When spike amplitude was combined with width measurements the separation between interneuron and pyramidal neurons increased compared to using spike width alone. This suggests that areas in the spike amplitude width space can be attributed to certain kinds of cell types.

Contents

1 | Introduction

Neurons electrically excitable cells which are responsible for all brain functions. There are an estimated 100 billion nerve cells in the brain that connect to each other in large networks. Neurons pass information between each other by sending action potentials through the nerve cell along thin structures called axons. These action potentials are sharp electrical impulses that can be measured by sharp electrodes positioned near the cell body (soma). The extracellular action potentials (EAPs, also called spikes) are generated by the currents leaving and entering cell membrane.

Recording electrodes can pick up the signal from several neurons at the same time, this requires spike sorting algorithms to isolate the spikes from each neuron. Spike sorting algorithms use the spike shape as a tool to differentiate neurons as all neurons tend to have slightly different spike shapes. Here we pose the question if additional information can be extracted from these spike shapes.

There are several types of neurons in the brain that serve different functions. Some have the ability to excite other neurons, while others can inhibit neurons thus preventing them from firing. The neurons that excite other neurons tend to have longer shapes than those that inhibit. Excitatory neurons of this kind are called pyramidal neurons from the characteristic shape of the soma which is comparable to a cone or pyramid. The shape of inhibitory neurons were found to be smaller and branching out less then their pyramidal counterparts. The name interneuron was then adopted for these kinds of neurons. The terms pyramidal neurons and interneurons reflect the shape (morphology) of the neurons, but has as such also been associated with the function of the neuron (Freund & Kali (2008) and Spruston (2009))

Early research showed that interneurons that had high firing rates showed thin action potentials with short durations. These action potentials could be distinguished from pyramidal cells with longer action potentials and showed a more regular spiking pattern (Mountcastle et al. (1969)). Pyramidal cells exhibiting long and regular spike patterns has since been referred to as regular spiking.

Here we have attempted to look closer at the differences in the spike shapes of pyramidal neurons and interneurons using biophysically accurate models from the Blue Brain Project (Ramaswamy et al. (2015)). Using only the spike width for identifying pyramidal neurons from interneurons has recently been debated as this black and white distinction seems not always to hold. To contribute to this debate we try to focus on the amplitude of the spikes in additional to the spike duration. We also try to look at how the definition of the spike duration and amplitude effect identification. Lastly a filter is applied to the spikes and gauge how this effects the results. The filter is applied because in many cases the raw signal from measurement electrodes must

be filtered to remove unwanted noise.

This thesis is organized by: (Chapter 2) going through a short theory introducing key knowledge of the problem at hand. (Chapter 3) The methods used gather and analyze the results. While working on this project a new python package was created to simplify the creation of the simulations. This package was created with the idea of modular simulations in mind, which might prove innovative. (Chapter 4) A validation of the simulation environment is done by attempting to replicate results from Pettersen & Einevoll (2008). Then the main simulation are presented using biophysical models from the Blue Brain Project. (Chapter 5) Finally the results are discussed.

2 | Theory

2.1 The Neuron

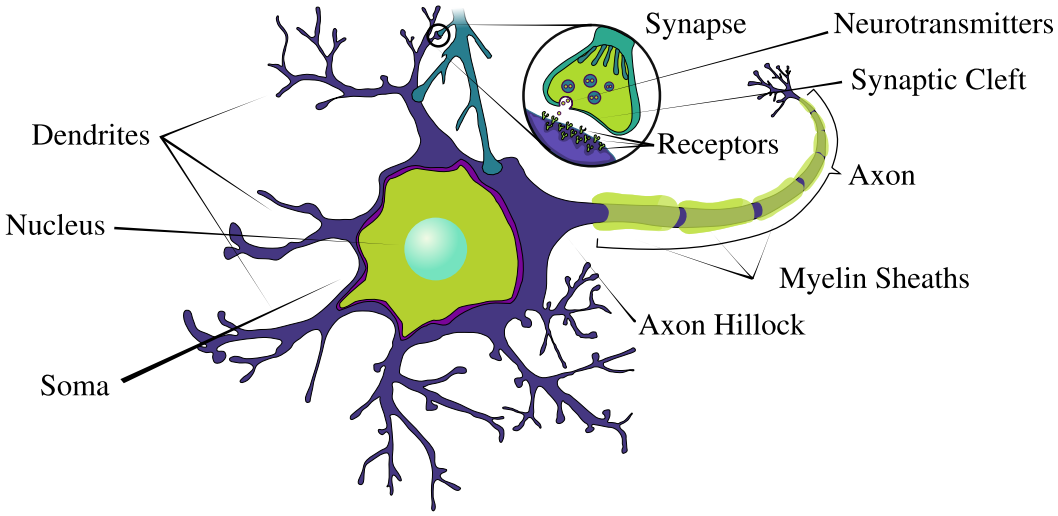

Figure 2.1: The anatomy of a neuron cell.

Neurons are electrically excitable cells that are a fundamental part of all brain functions. Other names include nerve cells, neurone or more colloquially brain cells. Neurons form in big networks which process information, and in the human brain there is an estimated 10^{11} neurons.

Special proteins in the cell membrane enables the neuron to fire action potentials when it is electrically excited. These action potentials are sharp voltage changes that propagates through

the full structure of the neuron. The same properties that makes the neuron able to fire makes the action potential regenerative, meaning it will propagate without decay.

The cell body of the neuron, the soma, has dendrites and the axon attached to it. The dendrites and the axon are very thin branching structures with a width usually in the order of $1\,\mu\mathrm{m}$. While neurons often have many dendrites directly attached to the soma there is only one axon attached to the soma at the axon hillock. The axon can branch several times before it ends and usually connects to the dendrites of other neurons via synapes.

The synapes are electrically sensitive which allows information, in the form of voltage fluctuations, to pass between neurons. Though the majority of all synapes are axo-dendritic (axon to dendrite), other junctions are also possible. Other junctions include but are not limited to, dendrite to dendrite, axon to axon and axon to blood vessel. When an action potential reaches a synapse it will activate the synapse and pass information to the connect neuron. The information that is passed along depends on the type of synapse, and if it is of a chemical or electrical type. (Sterratt et al. (2011))

2.2 The Cell Membrane

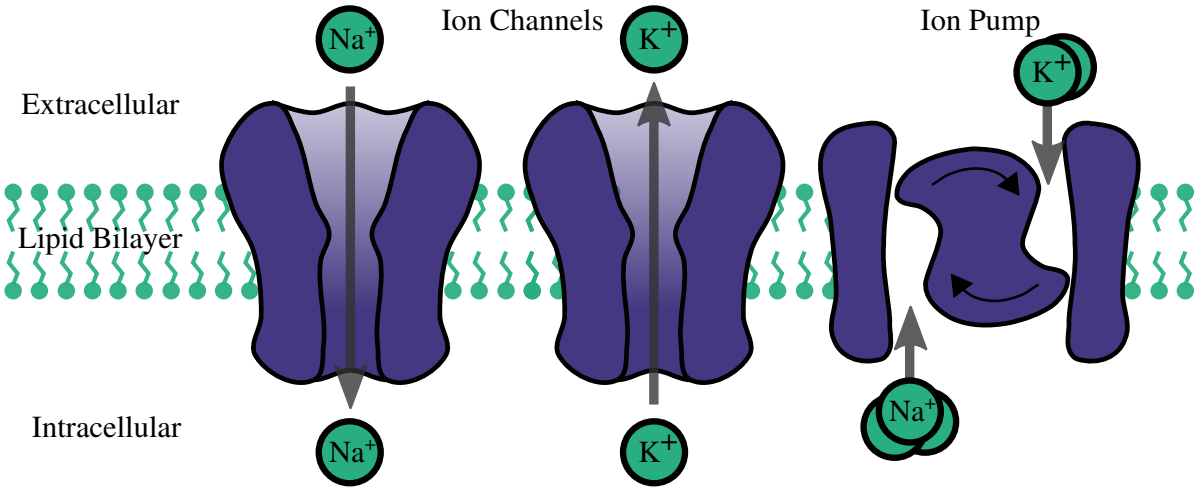

Figure 2.2: Common portrayal of ion channels and pumps in the neuron cell membrane. They are responsible for creating a potential gradient across the membrane. Ion channels have selective permeability of ions. Ion pumps activly transport ions through the membrane, often against the potential gradient. Image modified from Sterratt et al. (2011).

The potential difference between the inside and the outside of the neurons are caused by different concentrations of ions in the extracellular and intracellular medium. The ions cannot pass through the cell membrane as it consists of a $5\,\mathrm{nm}$ lipid bilayer which is mostly impenetrable to ions.

In the membrane sits ion channels and ion pumps which can have selective permeability to ions. Collectively they are referred to as ion transporters. These are created by proteins in complicated shapes. There and there are over 100 families of electrically sensitive ion channels (Sterratt et al. (2011)). Together they create a potential gradient across the membrane. The most

significant ions in this process are Sodium (Na^+), Potassium (K^+), Calcium (Ca^{2+}), Magnecium (Mg^{2+}) and Chloride (Cl^-). Ion channels are divided between passive channels and active channels where the active channels can change permeability under certain conditions while passive channels have a constant permeability. Figure 2.2 shows a common portrayal of ion channels and pumps.

The ion pumps differ from the channels by actively transporting certain ions through the membrane. For instance, the Sodium-Potassium exchanger pushes two K^+ ions out of the cell for every three Na^+ it pushes into the cell. Doing this creates a net loss of charge inside the cell and the pump is therefore electrogenic. Not all pumps are electrogenic, the Sodium-Hydrogen exchanger transports H^+ and Na^+ without effecting the net charge. For each H^+ ion out of the cell the pump pushes one Na^+ into the cell.

To further understand the electrical activity of neurons it is useful to view the neuron as an electronic circuit where the ion channels, ion pumps and the membrane serve as different electronic components. Figure 2.3 shows the electronic circuit used by Hodkin and Huxley in their original description of a neuron membrane. Hodgkin & Huxley (1952) and Sterratt et al. (2011)

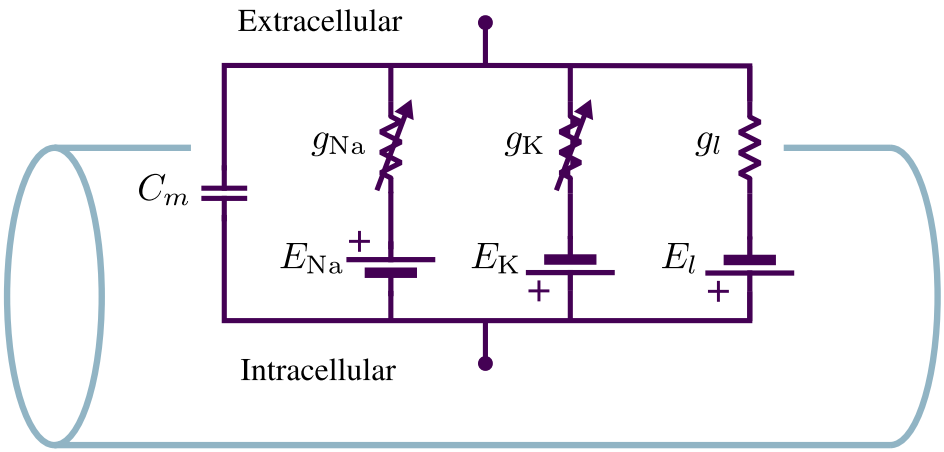

Figure 2.3: Hodgkin and Huxley electrical circuit. Shows a circuit model of the neuron membrane, each ion channel sets up a potantial like a battery with the selective permiability of ions represented by a variable resistor.

2.3 Circuit Model

Hodgkin and Huxley originally tried to understand the electrical properties of neuron by studying squid giant axons. These axons are among the largest in the animal kingdom and permitted more easy access for the electrodes of their time. They isolated the conductance of each of the channels and created mathematical models for them. The model was published 1952 and earned a Nobel prize in 1963.

The electrical circuit equivalent to a small piece of membrane has a capacitor in parallel with a resistor and battery for each type of ion channel. This kind of circuit is similar to a RC

circuit and shares many features. The ionic currents passing (I_{Na}, I_{K}, I_{l}) through the membrane will charge the membrane capacitor. The electromotive force from the ion channels will thus charge the membrane and resulting potential difference is called the resting potenial, V_{rest}. The resting potential of neurons are generally around $-70\,\text{mV}$.

The current equation of this neuronal equivalent circuit is

$$I = C_m \frac{dV}{dt} + I_{\text{Na}} + I_{\text{K}}. \tag{2.1}$$

Where the current through each channel is

$$I_{\text{Na}} = g_{\text{Na}} \cdot (V - E_{\text{Na}}) \quad I_{\text{K}} = g_{\text{K}} \cdot (V - E_{\text{K}}) \tag{2.2}$$

The heart of the Hodgkin and Huxley model is how the active conductances change with the membrane potential and will not be covered here. Given specific functions for the active conductances g_{Na} and g_{K}, this circuit was enough to describe an important feature of neurons called action potentials. Other than reproducing the action potential, the Hodgkin and Huxley model offered insight into the underlying mechanisms which experiments were not able to do.

2.4 Action Potential

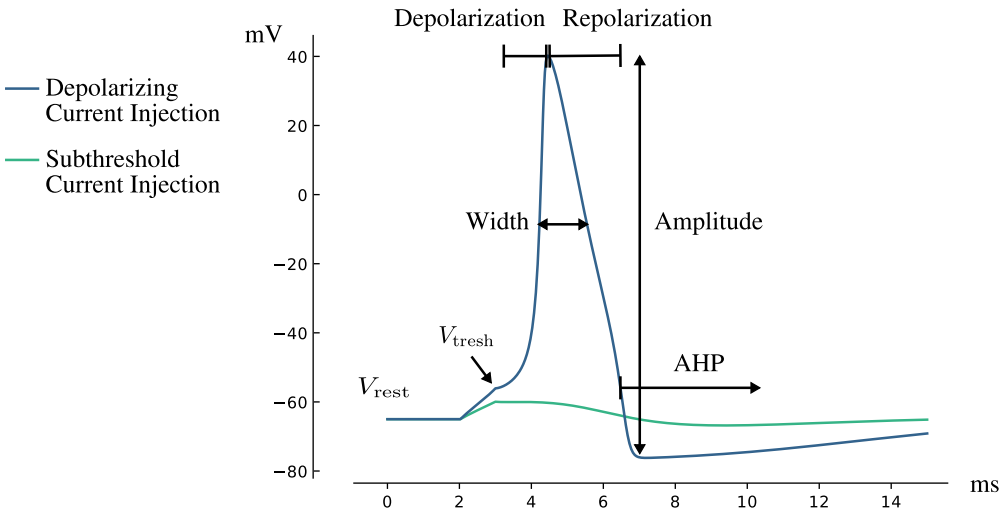

Figure 2.4: Anatomy of an action potential.

Action potentials are sharp increases in the membrane potential followed by a less sharp decrease towards the resting potential. The action potentials propegate along the cell membrane and into the axon. They are generally accepted as the information carriers between neurons and as such has been a big focus of neuroscience since its discovery.

Figure 2.4 shows the Hodgkin and Huxley action potential and some common definitions. Action potentials are initiated when the membrane potential is increase beyond a threshold, V_{tresh}. If the membrane potential does not reach the threshold it will steadily decrease to the neuron's resting potential. The source of current input are are mainly synapes that react to incoming action potentials from other neurons. A different source could be excitement through

a current electrode. When the threshold is reached the membrane potential will rapidly increase and an action potential is fired.

The increase in potential is referred to as depolarization until it reaches its peak magnitude. The potential then decreases towards the cells resting potential call the repolarization phase. As the cell repolarizes the potential will often decrease beyond the resting potential called the afterhyperpolarization.

The shape of the action potentials are directly related to the types and densities of the ion channels present in the membrane. This makes the action potential from neurons look slightly different, though they all have common features. Because neurons have slightly different action potentials it has been proposed that they can be used to differentiate neuron cell types, which is the basis of this project.

2.5 Multi-Compartmental Models

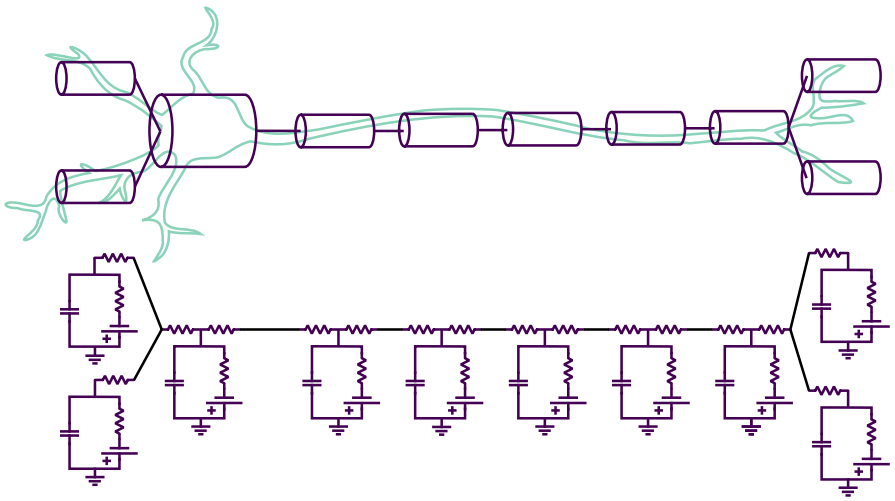

Figure 2.5: Multi-compartmental model.

The circuit model represents a patch of a neuronal membrane where the potential across the membrane is the same. If we want to study the voltage change across a longer piece of membrane we need to divide the membrane into parts and have a model for each of them. This is the basis for creating bigger models, where multiple pieces of membrane equivalent circuits are combined into a multi-compartmental model (fig. 2.5). Each of the compartments use a Thevinin equivalent circuit of the for generality. The size of the compartments has to be small enough so that the membrane potential can be considered the same at all places (Sterratt et al. (2011))

At any point the sum of the axial currents are equal to the capacitive, ionic and electrode currents in the compartment. The currents that go through the membrane are called transmembrane currents which is important for the calculation of the extracellular potential.

The calculation of the currents that pass into the adjacent compartments can be done using cable theory. These kind of calculations are already available in multi-comparmental simulation software such as NEURON.

2.6 Electrodes

The membrane potential is often desired when measuring neurons in physical experiments. Though recording the membrane potental is not always feasable as neurons are small (soma around $30\,\mu\mathrm{m}$) and the measurement electrode has to be in contact with the membrane. To measure neuron activity electrodes can more simply be placed in the vicinity of neurons and pick up the extracellular potentials from the transmembrane currents. Most knowledge about the physiological function of the brain is based on the recordings from such single point electrodes.

There are several types of electrodes and are always improving in quality. Earlier studies were often limited by the precision of the instruments. The type of electrodes used depends on several factors. An often used type of electrode are tetrodes consisting of four very small electrodes bundled together. The size of each electrode is generally around $30\,\mu\mathrm{m}$ in diameter.

2.7 Calculating Extracellular Potential

The extracellular potential is the electric potential generated from the transmembrane currents in the neurons. It can be calculated using Maxwell's equations and the continuity equation if the spatial distribution along the morphology of transmembrane currents and the extracellular conductivity is known. When a neuron fires this can be seen from the extracellular potential which will have a spike with similar timing to the intracellular spike (Lindén et al. (2013), Nunez & Srinivasan (2006), Pettersen & Einevoll (2008), and Gold et al. (2009)).

By modelling the neuron as compartments and approximating each compartment as a current point source at a distance r, the potential at from a single compartment will be,

$$\phi(r,t) = \frac{1}{4\pi\sigma}\frac{I(t)}{r}. \tag{2.3}$$

Where $I(t)$ is the transmembrane currents at time t and σ is the extracellular conductivity. This is the potential of a simple point charge in a isotropic medium, also known as Colomb's Law. As the compartments are positioned in discrete points in space and the sources add linearly, the equation can be expanded to include all N compartments of the morphology.

$$\phi(r,t) = \sum_{n=1}^{N}\frac{1}{4\pi\sigma}\frac{I_n(t)}{r_n} \tag{2.4}$$

To further improve the extracellular calculation the compartments can be approximated as line sources instead of point sources. For a linear current source of length Δs_n, the potential is given by,

$$\phi(r,t) = \frac{1}{4\pi\sigma} \sum_{n=1}^{N} \int_{-\Delta s_n}^{0} \frac{I_n(t)\, ds_n}{\Delta s_n \sqrt{r_n^2 + (h_n - s_n)^2}} \tag{2.5}$$

$$\phi(r,t) = \frac{1}{4\pi\sigma} \sum_{n=1}^{N} I_n(t) \frac{1}{\Delta s_n} \log \left| \frac{\sqrt{h_n^2 + r_n^2} - h_n}{\sqrt{l_n^2 + r_n^2} - l_n} \right|. \tag{2.6}$$

Where h_n is the longitudial distance from the end of the line to compartment n and $l_n = \Delta s_n + h_n$ is the distance from the start of the line (Gold et al. (2009)).

2.8 Extracellular Action Potentials

Extracellular action potentials (EAPs) are quite different than their intracellular correlates. The shape that is most often associated with an EAP is shown in fig. 2.6 (model L5_TTPC1_cADpyr232_1 from the Blue Brain Project using a grid simulation class). This shape is similar to the first derivative of the intracellular action potential and has been attributed to the resistive and capacitive properties of the neuron membrane (Brooks & Eccles (1947) and Henze et al. (2000)).

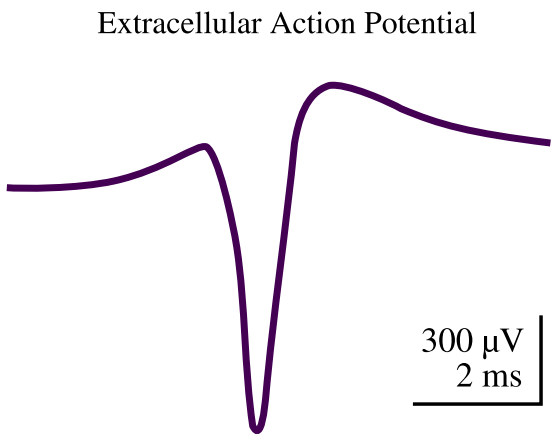

Figure 2.6: Portrayal of an extracellular action potential.

The fact is that EAPs can vary greatly in shape is apparent when looking at a simulation of the extracelluar potential in fig. 2.7. Though the transmembrane currents dissipate quickly the further away the compartments are from the soma. This results in EAPs relatively far away from soma but close to dendrites are practically indetectable as the amplitude is generally lower than the noise of the instrument. The EAP that can be measured usually has a negative polarity.

2.9 Neuron & LFPy

NEURON is a neuron simulation environment based upon multi-compartmental modeling. It was first released in 1997 and has become a mainstream simulation tool for multi-compartmental simulations since then.

NEURON supports custom mechanisms for the neuron membrane which can be inserted in any neuron compartments. Originally the software used a scripting language called HOC to set up and run simulations. HOC is much used for NEURON simulations today, but has long since died out in all other softwares. NEURON also features a python interface which can be used instead of scripting in HOC (*Documentation NEURON* (2016)).

LFPy is a Python package that uses the aforementioned equations to calculate the extracellular potential. It is built around NEURON such that existing models easily can be used width LFPy. NEURON uses cylindrical compartments, so by default LFPy uses line current sources equation for calculations. This setting can be changed to use point sources instead of line sources (Lindén et al. (2013)).

Python has a greater flexibility than the scripting language NEURON was originally used. LFPy also tries to improve how the programmer interacts with NEURON. It does this by creating classes for items such as the cell model or electrodes and this adds a more pythonic "flavor" to the simulations.

2.10 Cell Type Classification

The shape of action potentials is a big focus of this project as they are a direct product of the types and densities of the ion channels present in neurons. This is again based on the genetical makeup of the neurons. This makes the action potential a candidate for a classification parameter of different types of neurons.

It was early observed that the shape of action potentials are different for individual neurons. Mountcastle et al. (1969) discovered what they called regular spiking and thin spikes.

Results are based on spike width and amplitude. The basis for all spike shapes are the types and concentrations of ion channels. This is the central factor that decides if the neuron has short or long action potentials.

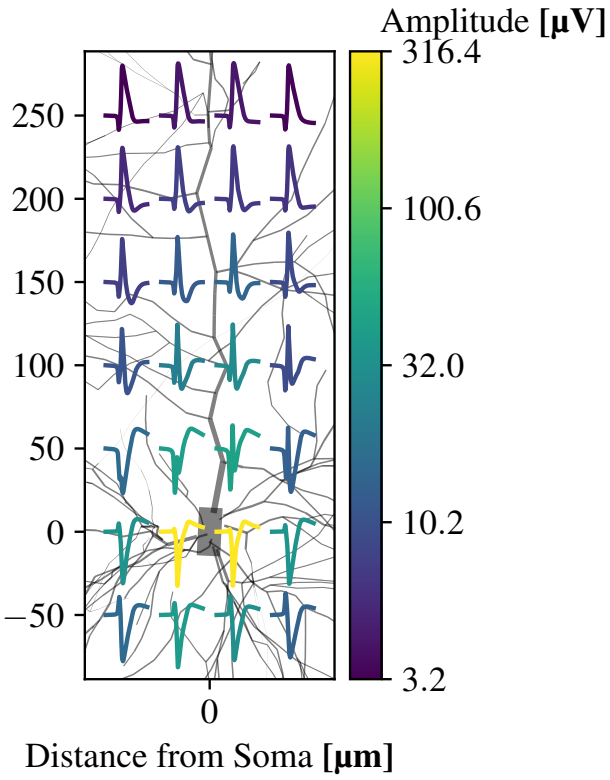

Figure 2.7: Extracellular spikes (EAPs) simulated around the soma. EAPs have a large diversity in shapes, though the most prominent ones are usually around soma.

3 | Methods

This section embodies methods that I have implemented into the simulations myself and most of which did not have finished solutions I could use. In general I have tried to use programming packages to do lower level calculations and tie them together to create a bigger software which I have used to set up and run simulations in. The product of this is the python package I have created which is called LFPy_util, which is ment to be an extension to the packge LFPy.

3.1 Spike Width Measurement

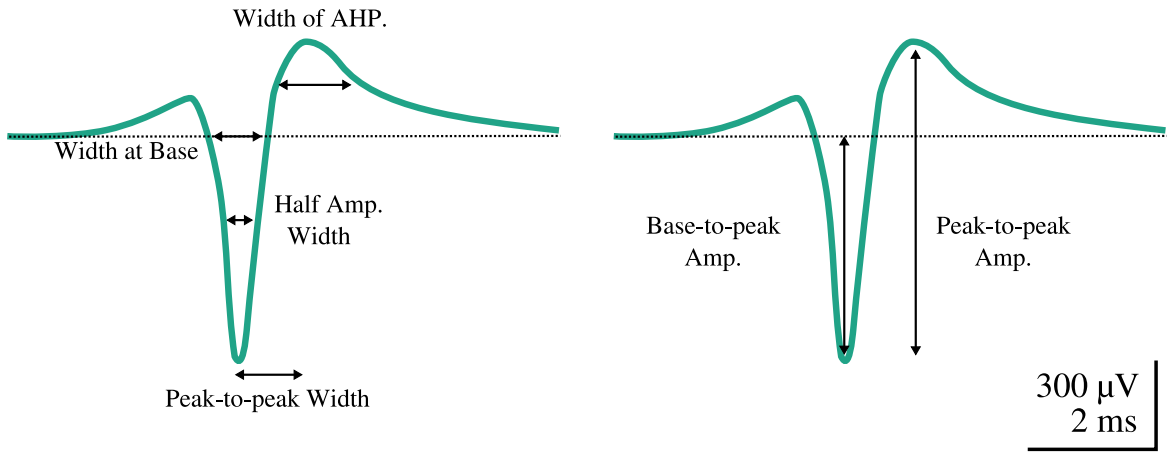

Figure 3.1: Depiction of an extracellular action potential with different width and amplitude definitions. As there are multiple definitions used in the literature it is not self-evident which ones to use.

There are several definitions of spike width used in neuroscience. In some cases the definition of the spike width is not mentioned and simply addressed as the spike duration. The most

commonly used spike width definition is the spike width at half amplitude (Bean (2007)).

Often the choice of spike width definition is chosen by seeing which gives the best bimodal distribution. Of some spike width definitions encountered in the literature were: width at half amplitude, peak-to-peak width, width at base and width of afterhyporalization phase. Each of the width and amplitude definitions evaluated here can bee seen in fig. 3.1. The implementation of each of these measurements are found in the appendix at section A.1.

Peak-to-peak Width: Refered to as type I spike width in the code. The width is measured as the time from the minimum potential to the maximum. This is a rough measure of the time from the polarization phase to the afterhyperpolarization phase.

The definition can be implemented by measuring the width from minimum to maximum value, but in cases where the spike is flipped the definition must also change. As such the implementation was done by defining the positive axis along the maximum absolute value and calculate the time from the maximum value to the preceding minimum value.

Half Amplitude Width: Width is measured as the duration the spike is below half amplitude of the signal measured from the baseline. The baseline is commonly set as the value of the start of the signal, though this is not always well defined. In many cases the start of the signal is chosen trivialy and if exciting a neuron with a square current the membrane potential will rise gradually. A more accurate description would be to set the baseline at the firing threshold of the intracellular spike. This is not possible in the case of extracellular spikes, but here the baseline can simply be set as 0.

Width at Base: The width at the baseline is commenly used in experiments. It can be difficult to get a good resolution as spikes are very short and measured width at baseline gives a longer duration than width at half amplitude.

Calculating width at baseline can pose problems when doing computer simulations of extracellular spikes as there is no firing threshold. If the baseline is set at 0 the duration of spike has a tendency to become unnaturally long due to the slight potential created by the charging of the membrane before firing. Experimentally this is not a problem as there is always noise in the signal. When attempting to measure the width at baseline for extracellular simulations the baseline can either be set to some fraction of max amplitude or set as a constant value. Both of these solutions have drawbacks however because it makes the threshold be chosen trivially.

Width of Afterhyporalization (AHP): Certain features of an action potential can be connected to the activation of some ion channels. That is why in some cases the duration of the afterhyperpolarization desired. The AHP phase is only defined as for the intracelluar action potential, but the definition can be expanded to EAPs by defining it as the point the potential comes back to $0\,\mu\mathrm{V}$ from the negative peak.

3.2 Spike Amplitude Measurement

Amplitude is easier to calculate than the width of a spike, but there are still a few different ways to define the amplitude. Figure 3.1 shows the two amplitude definitons inspected here.

Base-to-peak Amplitude: In most cases the amplitude is defined as the distance from a baseline, in a similar manner as a sinusoid. As with some of the width definitions the baseline is not always accuratly defined. For the EAP, shown in fig. 3.1, the baseline can bet set to 0. For the intracellular AP the baseline must be further defined as the start of the signal or at the firing threshold.

Peak-to-peak Amplitude: Amplitude is defined as the difference of the minimum and maximum value. This amplitude definition is the easiest one to implement and offers no ambiguousness in EAPs or intracellular APs.

3.3 Histogram Similarity Measure

To asses width and amplitude definitions on how well they classify interneuron from pyramidal neurons one needs a way to analyze the resulting data. To asses how well a variables can be seperated into groups, such as the width from pyramidal neurons or interneurons, is a classification problem. Classification is a big field in itself and a high quality classification algorithm is beyond the scope of this project. But if the separation between two variables are clear and bimodal there should still be possible to declare that this variable can be used as a classification parameter.

If we assume electrodes are placed randomly around a neuron and calculate the amplitude and width one can create a probability distribution of that measure. This is done by creating a histogram of the samples where the height of the bars are equal to the number of samples within that bin. By normalizing this distribution we get the measured probability distribution of that variable.

Comparing histograms is of fundamental importance in pattern classification, clustering and information retrival problems. From this field we can borrow a measure of the difference betweeen histograms. This measure is frequently refered to as the histogram distance. As histograms can be viewed as points in multidimensional space the histogram distance are also commonly refered to as a metric.

Choosing an aproptiate metric should be fit to the spesific data and results desired. To keep things simple two metrics have been choosen for this project, the Receiver Operator Characteristic Area under Curve (ROC AUC) and a simple intersection metric.

Receiver Operator Characteristic Area under Curve: Receiver Operator Characteristic (ROC) curves gives a measure of the performance of binary classifiers. The ROC curve compares two probability distributions where the true-positive (TP) rate is plotted against the false-positive (FP) rate. This can be plotted by calculating the area under the probability distribution above a criteria while moving the criteria across the sample space (fig. 3.2). If the sample space of the two distributions are disjoint the true-positive rate will go to 1 while the false-positive rate will stay at 0. This is considered a perfect classification and indicates that a threshold can be used to classy the two distributions from each other. (*Receiver operating characteristic* (2016)).

The area under the ROC curve (AUROC or AUC) has commonly been used as a way to measure model comparison. The AUROC goes from 0.5 where classification is as good as guessing, to 1.0 with a perfect classification. ROC curves and AUROC are often used in machine learning algorithms and evaulation of diagnostic tests. (Bradley (1997), Park et al. (2004), and *Receiver operating characteristic* (2016)).

Simple Overlap: This metric is also called the coefficient of overlap and is simply defined as the overlap of two histograms. It can be defined similar to the intersection divided by the union of sets, the Jaccard Index.

If $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$ and $\boldsymbol{y} = (y_1, y_2, \ldots, y_n)$ are the vectors of two histograms the overlap O is defined as,

$$O(x, y) = \frac{\sum_i \min(x_i, y_i)}{\sum_i \max(x_i, y_i)},$$

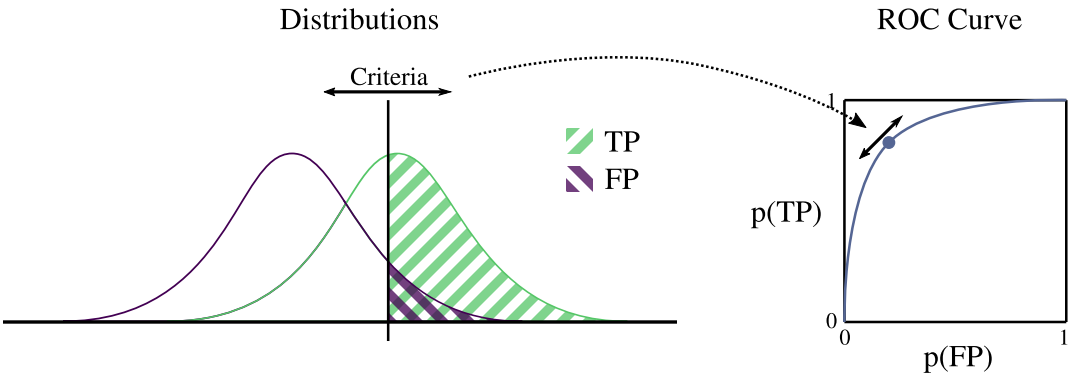

Figure 3.2: The calculation of two densities into a ROC curve. In terms of probabiity a ROC curve measures the True Positive (TP) rate over the False Positive (FP) rate. The TP and FP values are calculated at multiple criterias, where each criteria value gives one point on the ROC curve. The area under a ROC curve (AUROC) is used as a measure of how well two distributions can be seperated from each other. If two identical distributions or histograms are compared the TP value is equal to the FP value at all thresholds. The ROC cureve would then be a line from $(0, 0)$ to $(1, 1)$ and the AUC would be 0.5 which indicated that catagorization could be done no better than chance.

This metric was chosen because AUROC is only defined for one dimenensional data and because of its simplicity. The metric measures 0 when the two variables share no data in common and 1 if compared to itself. A 2d histogram metric was needed because the width and amplitude from interneurons and pyramidal neurons are being compared simultaniously.

3.4 Blue Brain Models

Obtaining a reconstruction of the morphology of neurons is an arduous task which has limited the number of fully reconstructed neuron models.

The Human Brain Project Neocortical Microciruit Collarboration Portal (`bbp.epfl.ch/nmc-portal/welcome`, Ramaswamy et al. (2015)) released multiple neuron models from the hind limb somatosensory cortex of 2-week-old Wistar Han rats. The neuron models are based on the classification criteria set by the Blue Brain team. The models have been classified into both morphological and electrophysiological types with full morphological reconstruction. These were named m-types and e-types, and in combination they created neuron models with me-types. The was 55 m-types and 11 e-types found, and in total 207 me-types. The me-type reflects a unique neuron class. Each of the me-types has 5 example models than can be downloaded from their website.

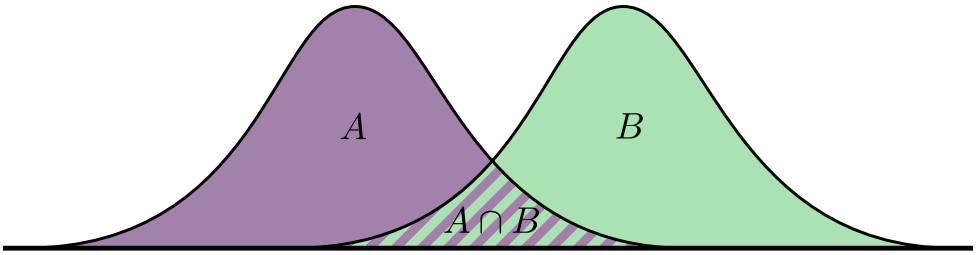

Figure 3.3: An intersection metric as a way to measure the similarity between two distributions. ROC curves are only defined for 1d densities so this metric was chosen as it can be used for 2d histograms.

3.5 LFPy_util

3.5.1 About

LFPy_util is a python package that was created for this project with the purpose to simplify the simulation pipeline for multiple neurons and creating an easy to use interface when developing new simulations. LFPy_util extends and uses the package LFPy to accomplish this. Simulations can be run in parallel and data from simulations can automatically be saved and loaded to avoid unnecessary processing time. LFPy_util has 9000 lines of code, documentation can be found at `https://github.com/lastis/LFPy_util`.

LFPy is a Python package created to calculate extracellular potentials. Another major feature is wrapping the cell model and electrodes from the NEURON simulation environment into Python objects, such as the `LFPy.Cell` and `LFPy.StimIntElectrode` classes. This makes working with NEURON more pythonic as NEURON is based on a scripting language without such functionality. That is, even though the Python interface of NEURON uses objects, the objects are bound to a global state and cannot be instantiated.

While NEURON has support for parallel processing of single simulations it does not have any inherit support for running multiple independent simulations. For "embarrassingly parallel" situations like this, users have had to resort to creating their own methods to start each simulation independently. For instance, one solution would be to use a Python script to run other scripts through the command line, effectively starting new processes. In addition there is no function to reset the simulation environment which can make previous simulations affect later ones.

LFPy_util uses Python's multiprocessing package to run independent simulations and thus overcomes some of the shortcomings of NEURON and LFPy.

3.5.2 Minimal Working Examples

These examples show how to create a new custom simulation from scratch and how to use them with LFPy_util. A basic understanding of object-oriented programming and Python is required.

To run a simulation LFPy_util must first have a `LFPy.Cell` object it can use to interact with the model. The cell object gives access to functions such as `Cell.simulate()` which starts the NEURON simulation. A template of such a function can be seen in listing 1. A fully working example of such a function can be seen in the appendix (section A.3).

Listing 1: load_model_simple.py

```
import LFPy

def get_cell(neuron_name):
    """
    Load a spesific model based on the input string and return
    a LFPy Cell object

    :param string neuron_name:
        String to identify the neuron model that will be loaded.
    :returns:
        Cell object from LFPy.
    """

    cell = LFPy.Cell( some cell parameters ... )

    return cell
```

LFPy_util use subclasses of the class `LFPy\_util.sims.Simulation`. to organize simulations. Listing 2 shows a very minimal example of such a subclass. If the functions `simulate`, `process_data` and `plot` are not overrided, a `NotImplementedError` will be raised.

Listing 2: new_simulation_class_simple.py

```
from LFPy_util.sims import Simulation

class CustomSimulation(Simulation):
    def __init__(self):
        # Inherit the LFPyUtil simulation class.
        Simulation.__init__(self)
        # These values are used by the super class to save and load data.
        self.set_name("custom_sim")

    def simulate(self, cell):
        pass

    def process_data(self):
        pass

    def plot(self, dir_plot):
        pass
```

The LFPy_util simulation class has been created to reflect four parts of a typical neuron simulation. (1) Initilization, (2) simulation/gather data, (3) processing data and (4) plotting the data. These four parts are retained in the initilization of the object, the `__init__()` function, a simulation function `simulate(cell)`, a process function `process_data()` and a plotting function `plot()`. Run parameters, plot parameters and data are stored in dictionaries in the simulation class and the variables are named `run_param`, `plot_param` and `data` respectively. The LFPy_util simulation class has additional properties that among other things enable saving and loading data and naming those files. Because of this a new simulation class should inherit LFPy_util's simulation class.

Listing 3 defines a simulation class that inherits the LFPy_util simulation class, but adds some more functionality. The function `simulate` inserts a stimulus electrode and applies a current to soma with parameters defined in `__init__`. The membrane potential is then stored in the `data` dictionary. The function `process_data` creates a normalized version of the membrane potental and saves it. The function `plot` plots the membrane potential. To exemplify the use of `plot_param`, a conditional statement is used to decide whether or not to plot the normalized version.

Listing 3: new_simulation_class.py

```
import LFPy
import LFPy_util
import matplotlib.pyplot as plt
from LFPy_util.sims import Simulation

class CustomSimulation(Simulation):
    def __init__(self):
        """
        Typical initialization function, called when a new instance
        is created.
        """
        # Inherit the LFPyUtil simulation class.
        Simulation.__init__(self)
        # These values are used by the super class to save and load data.
        self.set_name("custom_sim")

        # Create some parameters that are used by the simulate method.
        self.run_param['delay'] = 100       # ms.
        self.run_param['duration'] = 300    # ms.
        self.run_param['amp'] = 1.0         # nA.

        # Create a parameters used by the plotting function.
        self.plot_param['plot_norm'] = True

    def simulate(self, cell):
        """
        Setup and starts a simulation, then gathers data.

        :param LFPy.Cell cell:
            Cell object from LFPy.
        """
        # Create an electrode with LFPy.
        soma_clamp_params = {
            'idx': cell.somaidx,
            'amp': self.run_param['amp'],
            'dur': self.run_param['duration'],
            'delay': self.run_param['delay'],
            'pptype': 'IClamp'
        }
        stim = LFPy.StimIntElectrode(cell, **soma_clamp_params)

        cell.simulate()

        # Store the data .
        self.data['soma_v'] = cell.somav
        self.data['soma_t'] = cell.tvec

    def process_data(self):
        """
        Process data from the simulate function, usually to prepare
        the data for plotting. This function creates a normalized
        version of the membrane potential.
        """
        soma_v_norm = self.data['soma_v'].copy()
        soma_v_norm -= soma_v_norm[0]
        soma_v_norm /= soma_v_norm.max()
        self.data['soma_v_norm'] = soma_v_norm

    def plot(self, dir_plot):
        """
        Plot data from the simulate and process_data function.
        This functions plots the membrane potential and the
        normalized version.

        :param string dir_plot:
            Path to the directory where plots should be saved.
        """
        plt.plot(self.data['soma_t'], self.data['soma_v'])
        # Save the plot to input directory with the name "custom_sim_mem".
        LFPy_util.plot.save_plt(plt, "custom_sim_mem", dir_plot)
        # plot_param can be used to affect the plotting.
        if self.plot_param['plot_norm']:
            plt.plot(self.data['soma_t'], self.data['soma_v_norm'])
            # Save the plot to input directory.
            LFPy_util.plot.save_plt(plt, "custom_sim_mem_norm", dir_plot)
```

The newly created simulation class can now be used to run a complete simulation as seen in listing 4.

Listing 4: first_simulation.py

```
# Import the get_cell function defined previously.
from load_model import get_cell

# Import the newly defined simulation class.
from new_simulation_class import CustomSimulation

# Load the model, using a string to identify which model will be returned.
cell = get_cell("pyramidal_1")

# Create an instance of the custom simulation class.
sim_custom = CustomSimulation()

sim_custom.simulate(cell)
sim_custom.process_data()
# Plots are stored in a folder called first_simulation.
sim_custom.plot("first_simulation")
```

If one attempted to run the simulation function in listing 3 more than once in the same program, we would experience that subsequent simulations would be affected by previous simulations. This is because a new electrode will be applied once for each call to the simulation function. With NEURON or LFPy there are no easy way to reset the environment to prevent this. One workaround is to use Python's multiprocessing package to create multiple independent processes. Doing this for every simulation will require excessive amounts of code. LFPy_util can simplify this with tools that requires less code and are compatible with classes like the one defined in listing 3.

The class `LFPy\_util.Simulator` accepts one or more objects that inherit LFPy_util's simulation class and can run them either in serial or paralell and either in a new independent processes or in the same process. Listing 5 shows how the newly created simulation class can be used with `LFPy\_util.Simulator` to run multiple simulations in parallel and in independt processes.

Listing 5: multiple_simulations.py

```
import LFPy_util
from load_model import get_cell
from new_simulation_class import CustomSimulation

sim = LFPy_util.Simulator()
sim.set_cell_load_func(get_cell)
# The string is passed to the get_cell function.
sim.set_neuron_name("pyramidal_1")

sim_custom_1 = CustomSimulation()
sim_custom_1.run_param['amp'] = 1.25 # nA

sim_custom_2 = CustomSimulation()
sim_custom_2.run_param['amp'] = 0.75 # nA
# Avoid sim_custom_2 overwriting the data from sim_custom_1.
sim_custom_2.set_name("custom_sim_2")

# Add the simulations to a list we want to run.
sim.push(sim_custom_1)
sim.push(sim_custom_2)

sim.simulate()
sim.plot()
```

The function `Simulator.push` adds the simulation objects to a list. When the function `Simulator.simulate()` is ran, the `simulate()` function of those objects will be

called in parallel and in independt processes. The `Simulator.plot()` function will call `process_data()` and `plot(dir_plot)` functions of those objects, and the parameter `dir_plot` will be created based on the name of the simulation class and the name of the neuron name. In this case the plots are stored in the directories `./pyramidal_1/plot/custom_sim/` and `./pyramidal_1/plot/custom_sim_2/`.

The `Simulator` class utilizes save and load features of the simulation class when the `Simulator.simulate()` function is called. This makes the dictionaries `data` and `run_param` be saved to file. If `Simulator.plot()` is ran without `Simulator.simulate()` being called first, the program will notice that the simulations are missing the `data` dictionary. It will then attempt to load the `data` and `run_param` from file. After listing 5 has been run once line 22 can thus be commented out and the data will be loaded instead of running the simulation.

LFPy_util comes with some predefined simulations that have been used for results in this article. Listing 6 shows an example of how to use the predefined simulations.

Listing 6: predefined_simulations.py

```
1  import LFPy_util
2  from load_model import get_cell
3
4  sim = LFPy_util.Simulator()
5  sim.set_cell_load_func(get_cell)
6  sim.set_output_dir("predefined_simulations")
7
8  sim.set_neuron_name("pyramidal_1")
9
10 sim_electrode = LFPy_util.sims.MultiSpike()
11 sim_electrode.run_param['spikes'] = 3
12
13 sim_intra = LFPy_util.sims.Intracellular()
14
15 # Add the simulations to a list we want to run.
16 sim.push(sim_electrode, False)
17 sim.push(sim_intra)
18
19 sim.simulate()
20 sim.plot()
```

The class `MultiSpike` searches for the input current that will result in 3 spikes and then applies that electrode. It also plots some figures, like the input current and the spikes. `Intracellular` simulates and plots statistics about some intracellular recordings. The details of the predefined simulations can be found in section A.2 List of Simulation Classes.

Note the extra condition, `False`, in line 16. This tells the simulator that this simulation should not be run in an independent process. When the `MultiSpike` simulation is finished, the electrode it used to generate 3 spikes is still loaded in NEURON. When the simulation function of `Intracellular` runs, the electrode will excite the neuron and generate 3 spikes. This feature can be used to link simulations and makes the simulations modular.

The previous examples use only one neuron model. To be able to run many different models it is usefull to define a function such as the one in listing 7.

Listing 7: simulator.py

```
import LFPy_util
from load_model import get_cell

def get_simulator(neuron_name):
    sim = LFPy_util.Simulator()
    sim.set_cell_load_func(get_cell)
    sim.set_output_dir("multiple_neurons")
    sim.set_neuron_name(neuron_name)

    sim_electrode = LFPy_util.sims.MultiSpike()
    sim_electrode.run_param['spikes'] = 3

    sim_intra = LFPy_util.sims.Intracellular()

    sim.push(sim_electrode, False)
    sim.push(sim_intra)

    return sim
```

To do the same simulation as in listing 6, one could write:

Listing 8: run_simulator.py

```
from simulator import get_simulator

sim = get_simulator("pyramidal_1")
sim.simulate()
sim.plot()
```

Listing 9 runs the two predefined simulations with two different neuron models. The class `LFPy\_util.SimulatorManager` takes a list of neuron names and passes them to the `get_simulator(neuron_name)` function in listing 7. The parameter `neuron_name` are the elements of the list of neuron names. After running the code once, `simm.simulate()` can be commented out and the simulations will load data instead of running the simulations.

The final simulation uses three files: listing 1, listing 7 and listing 9 and 2 predefined simulation classes.

Listing 9: multiple_neurons.py

```
from simulator import get_simulator

neurons = ["pyramidal_1", "pyramidal_2"]

simm = LFPy_util.SimulatorManager()
# Number of LFPy_util.Simulator objects  that will run in parallel.
simm.concurrent_neurons = 8
simm.set_neuron_names(neurons)
simm.set_sim_load_func(get_simulator)

simm.simulate()
simm.plot()
```

4 | Results

4.1 Model Validation

To verify that the simulation environment could be trusted some results from Pettersen & Einevoll (2008) were replicated. Specifically the spike width and amplitude dependency in relation to the distance from soma was compared to current results.

4.1.1 Setup

Model: The Mainen & Sejnowski (1996) morphology was used with a passive model, which is the same model used in Pettersen & Einevoll (2008). The cell was rotated using PCA (principal component analysis) on the compartment positions. This calculates three orthogonal vectors such that the positions of the compartments has the greatest variance along the first principal component, second highest along the second and third most along the third. The first principal component was made parallel to the y-axis which puts the apical dendrites along this axis (the dendrites that emerge from the apex of the cell) (fig. 4.1).

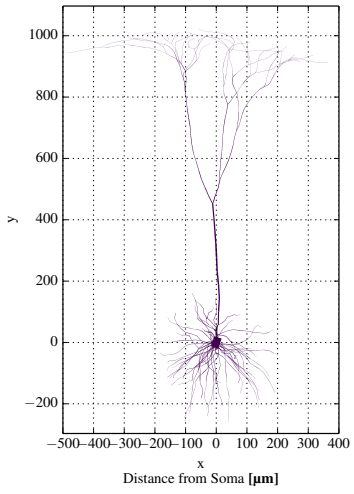

Figure 4.1: Morphology of Mainen & Sejnowski (1996) cell. The apical dendrites are located along the y-axis after rotation with PCA.

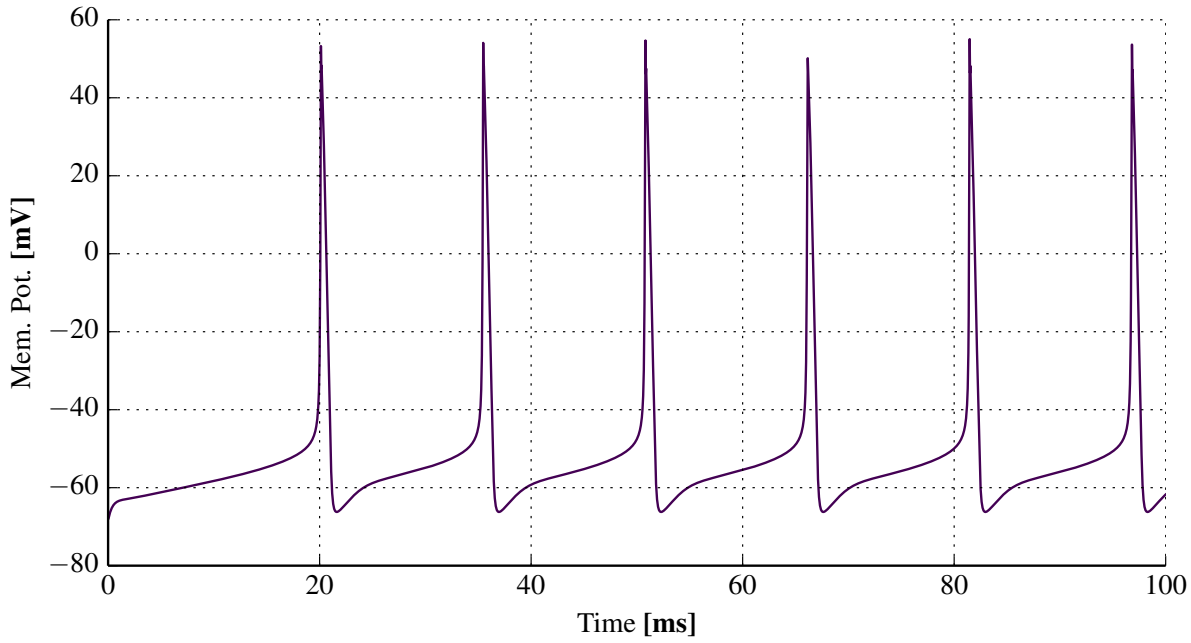

Figure 4.2: Simulation of the Connor-Stevens model using parameters from Dayan & Abbott (2001). A similar graph is shown in fig. 6.1 (B) in their book.

Spike Generation: To recreate the action potential used in Pettersen & Einevoll (2008) a spike was generated using the Connor-Stevens model (Connor & Stevens (1971) and Connor, Walter, et al. (1977)) using the same parameters as Dayan & Abbott (2001) and Pettersen & Einevoll (2008). In fig. 4.2 the Connor-Stevens simulation is shown where the second peak was used for further analysis. This spike had an amplitude of $119.49\,\mathrm{mV}$ from baseline. The baseline was estimated to $-53.26\,\mathrm{mV}$ and the peak at $53.26\,\mathrm{mV}$. These values matches Dayan & Abbott (2001), but not the spike used in Pettersen & Einevoll (2008) which had an amplitude of $83\,\mathrm{mV}$ from baseline. Pettersen & Einevoll (2008) does not go into further detail about the creation of the action potential other than stating the action potential were similar to Dayan & Abbott (2001). The difference might be explained by the fact that action potentials from pyramidal neurons often peaks at $20\,\mathrm{mV}$, and that this was achived by scaling the original signal from the Connor-Stevens model. To compensate for the difference the action potental used in further simulations were scaled to $83\,\mathrm{mV}$ (fig. 4.3).

The input current was set to $12.6\,\mu\mathrm{A\,cm^{-2}}$. and was very carefully adjusted to make the magnitude spectrum (fig. 4.5) similar to Pettersen & Einevoll (2008) figure 3. Without adjustment the magnitude spectrum tended to have a different inital value, from 6 to $8\,\mathrm{mV}$, and was not as smooth.

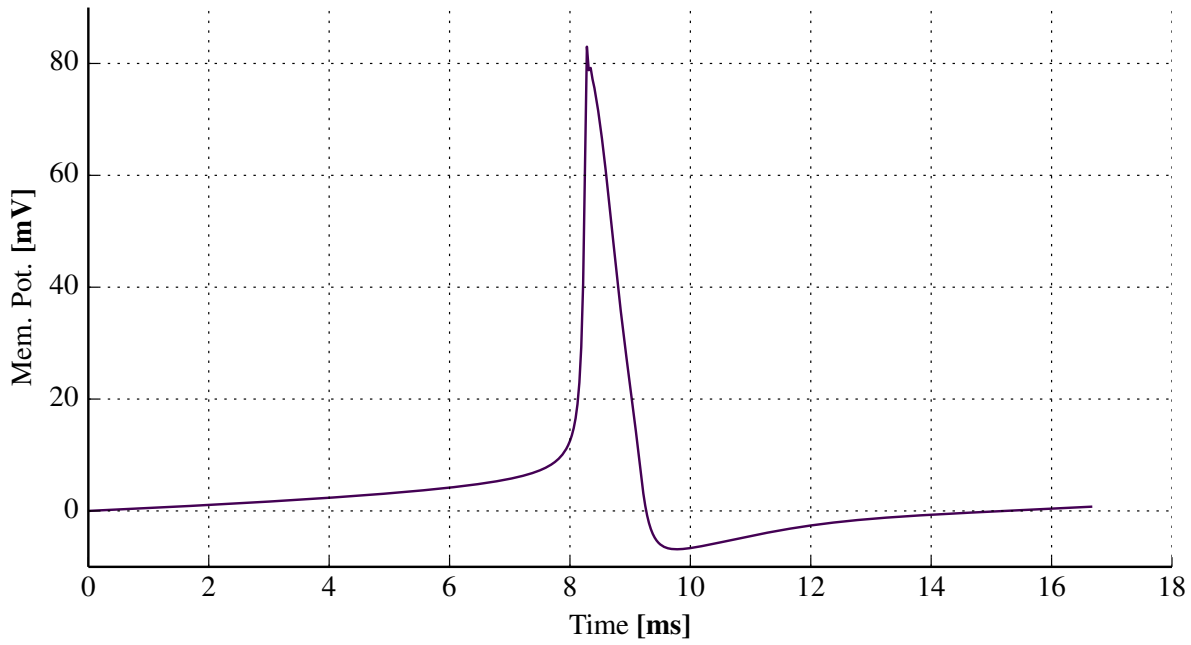

Figure 4.3: The second peak in fig. 4.2 scaled to $83\,\mathrm{mV}$ to match the action potential used in Pettersen & Einevoll (2008).

Parameters: Parameters for the Neuron simulation were the same as Pettersen & Einevoll (2008) and the aforementioned action potential was used as a boundary condition in soma. This was accomplished by setting the membrane potential equal to the action potential in all soma sections using the `.play` vector function in Neuron. This "excites" the neuron even though there are no ion channels in the model.

According to the model specifications we used a membrane resistance of $R_m = 30\,\mathrm{k\Omega}$, membrane capacitance $C_m = 1\,\mu\mathrm{F}$, axial resistance $R_a = 150\,\Omega$, time resolution $dt = 2^{-5}\,\mathrm{ms}$. The reversal potential was set to zero.

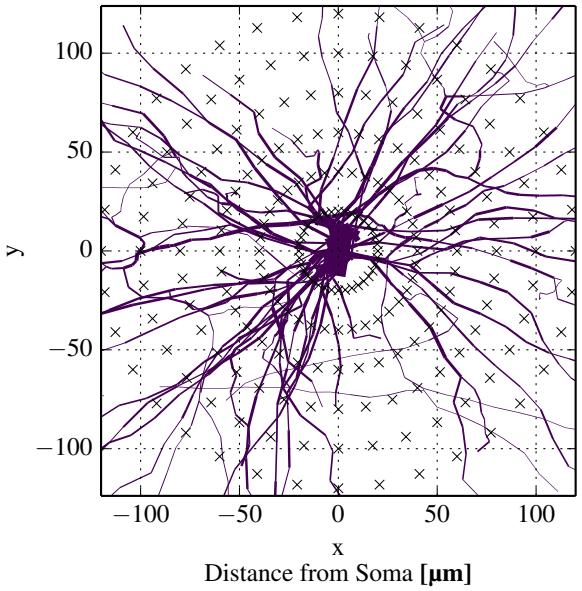

Figure 4.4: Electrode positions placed in a plane around soma perpendicular to the axis along the apical dendrites.

Electrode Positions: Recording sites were placed in the xz-plane at 11 linearly spaced positions along 36 lines with equal angular spacing (fig. 4.4). Pettersen & Einevoll (2008) states the recording positions were in the plane perpendicular to the apical dendrites, this is ensured by the rotation done with PCA and putting the electrodes in the xz-plane.

Spike Width & Amplitude: A baseline was set as the value at the start of the signal. Amplitude

was calculated as the difference between the maximum value and the baseline. The spike width was defined as the duration the potential was above 25% of the maximal absolute amplitude. This was accomplished by using the half-amp. width python function but with the threshold set at 25%.

At $dt = 2^{-5}\,\mathrm{ms}$, the spike width from the Connor-Steven model was $0.5625\,\mathrm{ms}$. This is similar to the reported spike width from Pettersen & Einevoll (2008) which was was $0.55\,\mathrm{ms}$.

4.1.2 Results

The action potental that was used in Pettersen & Einevoll (2008) is similar to the one used here. The amplitude of the fourier transform is displayed in fig. 4.5, which is in close resemblance to the action potential in fig. 3 in the paper.

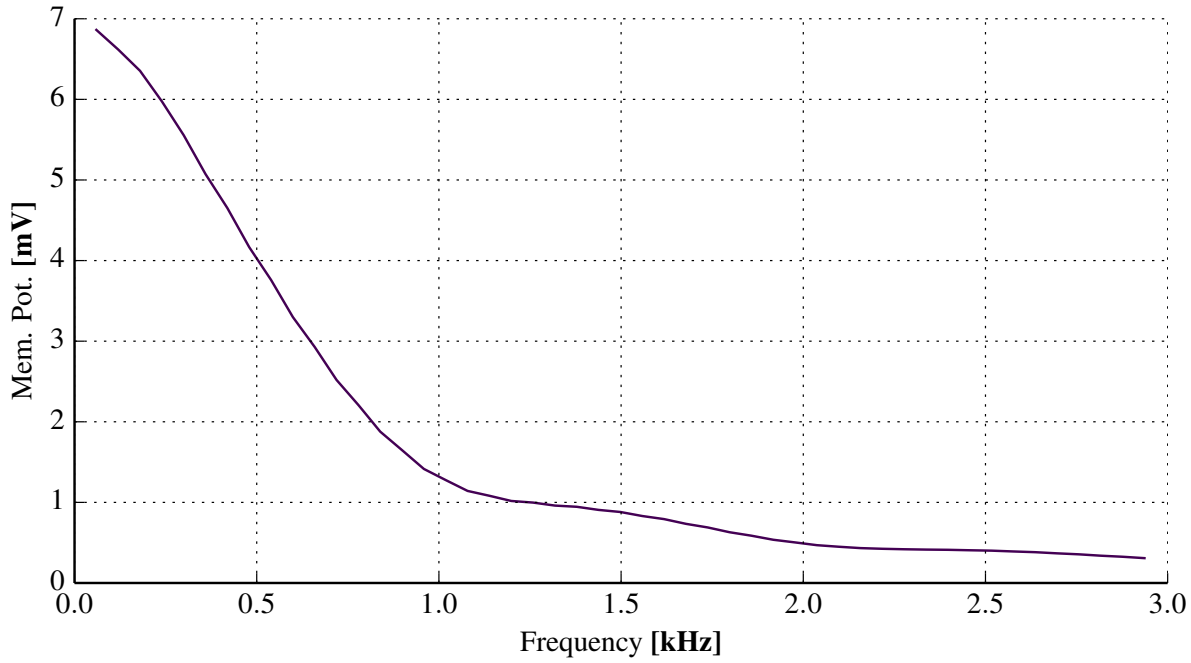

Figure 4.5: Magnitude specter of simulated somatic membrane potential.

The spike width increases with the distance from soma as seen in fig. 4.6. These results from Pettersen & Einevoll (2008) show an initial spike width of about $0.5\,\mathrm{ms}$ at $20\,\mu\mathrm{m}$ to $0.8\,\mathrm{ms}$ at $120\,\mu\mathrm{m}$. Current results are higher than the reported widths by about $0.1\,\mathrm{ms}$ at every distance. The spike width is defined as width of the negative phase at 25% of the maximum amplitude. If the spike width is adjusted to 35% of the maximum amplitude the results match nearly perfectly, though the importance of this is unclear.

Sudden changes in spike width was experienced with increased distance from soma. Above $200\mu V$ most of the spikes shapes were not well defined. This was also reported in Pettersen & Einevoll (2008).

Figure 4.7 shows the spike amplitude with logarithmic axes. The exact results from Pettersen & Einevoll (2008) are not available but the approximate value can be seen from their plots and the exponential decay $1/r^n$ was reported as $n \sim 2$ at $20\,\mu\mathrm{m}$ and $n \sim 2.5$ at $120\,\mu\mathrm{m}$. Current results are not identical to those findings and have an exponent of $n = 2$ at $20\,\mu\mathrm{m}$

and $n = 2.8$ at $120\,\mu m$. The value of the amplitude was about $350\,\mu V$ in Pettersen & Einevoll (2008), but the current model only gives an amplitude of $120\,\mu V$ at $20\,\mu m$.

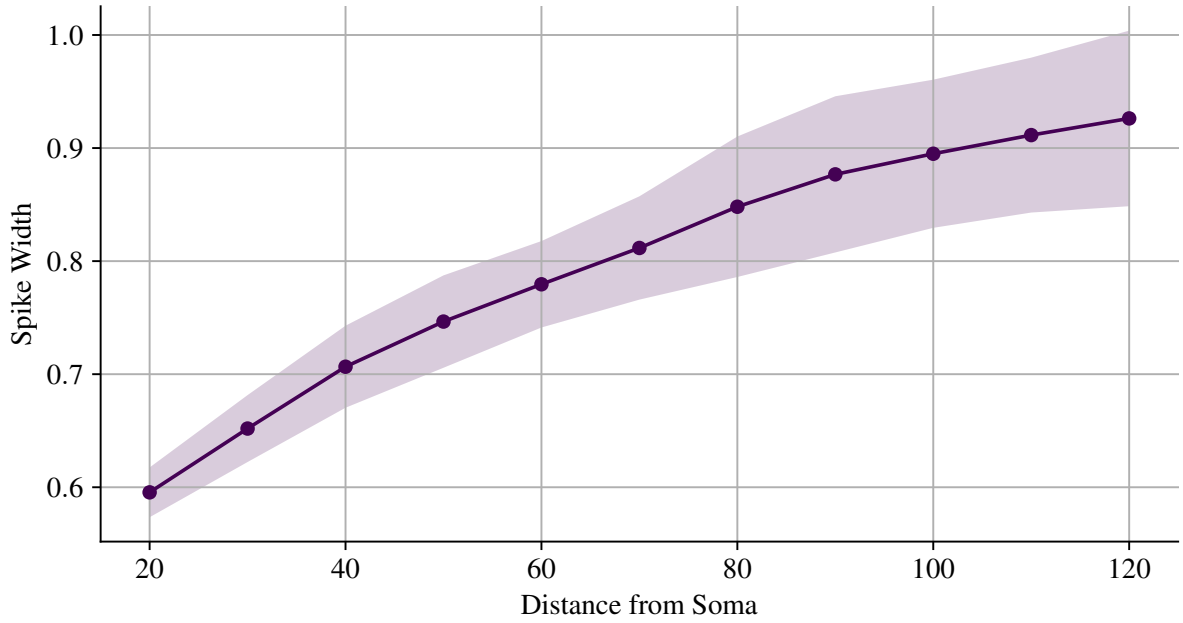

Figure 4.6: Spike width over distance. Mean +/- 1 std.

4.1.3 Conclusion

The current results are qualitativly similar to the results in Pettersen & Einevoll (2008) however there are quantitative differences. These differences are probably due to differences in methodology we have not been able to identify since we have been unable to obtain the source code.

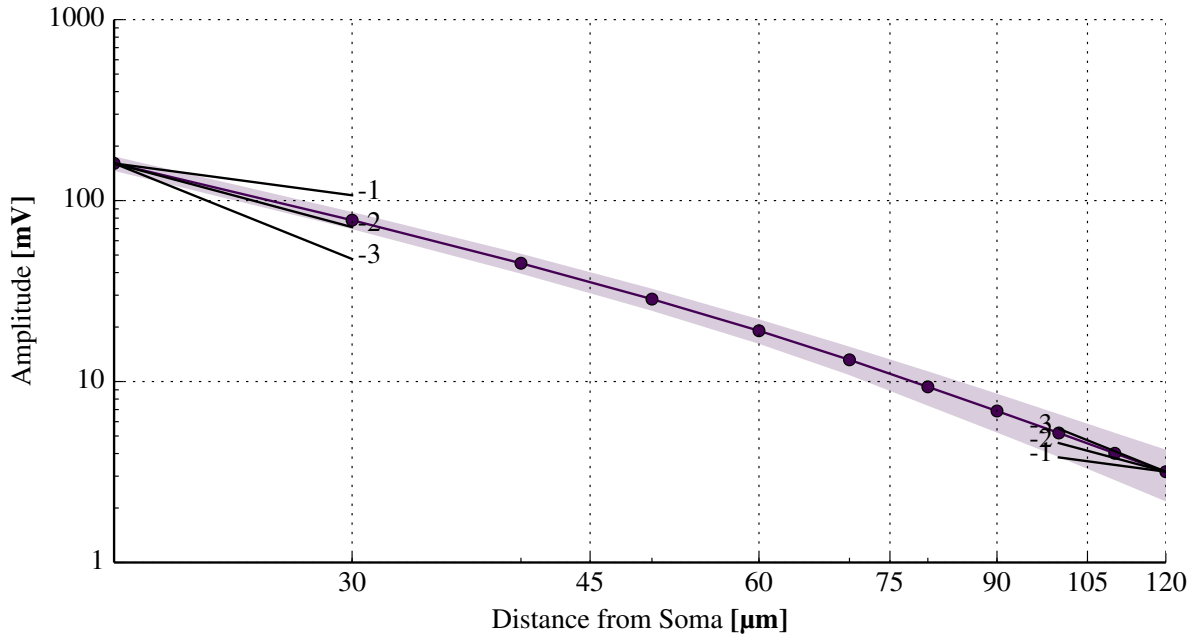

Figure 4.7: Spike amplitude over distance. Mean +/- 1 std. The power law decays $1/r$, $1/r^2$ and $1/r^3$ are shown at the leftmost and rightmost data points.

4.2 Blue Brain Simulations

The number of previous simulations of extracellular action potentials are few. The publicly avaiable models from the Blue Brain (section 3.4) include neuron models with full reconstruction of the morphology. The neuron models are ment to cover the whole diversity of neuronal types encounted in the mouse somatosensory cortex. This is one of the first simulations where the extracellular action potentials of a large number of different neuronal types are being compared.

Spike width and amplitude are known to be markers for pyramidal neurons and interneurons.

Many different definitions of spike width has been used to differentiate neurons and it is not clear which spike definitions are better suited for classification. Here models from the Blue Brain Project has been used to investigate if some spike width definition prove better than others for classification.

4.2.1 Setup

Models: Each simulation uses all available models in the L5 area. The pyramidal models consisted of models from 4 me-types with the morphologies TTPC1, TTPC2, STPC, and UTPC and the same e-type. The interneuron models consisted of the remainding 48 me-types. Each me-type had 5 models which in total summed to 20 pyramidal models and 240 interneuron models.

In this interneuron group many of the me-types represent only a small portion of the number of neurons found in L5. The findings from Markram et al. (2015) suggest that overall the ratio between pyramidal neurons and interneurons is around $87\% \pm 1\%$. And of those interneurons 50% were classified as baskets cells (LBC and NBC). The interneurons from L5 were seperated

into 9 m-types (morphological) and 10 e-types (electrophysiological) which gave a total number of 48 me-types.

Each of the 5 models in a me-type have a slightly different morphological shape and/or firing pattern. Though the degree of difference is not made clear in the model documentation.

All neuron models were rotated using PCA before simulations were instigated, identical to the setup in section 4.1.1.

The extracellular conductivity was set to $\sigma = 0.3\,\Omega^{-1}$ based upon data from experimental measurements.

Spike Generation: Spikes were created using the simulation class MultiSpike (section A.2.2). For each model 3 spikes were provoked during a simulation of $1000\,\mathrm{ms}$ with a square current pulse of equal duration.

All stimulus electrodes uses the `LFPy.StimIntElectrode` with a custom made electrode named ISyn. With the default stimulus, `IClamp`, the sum of the transmembrane transmembrane currents are equal to the input current. With ISyn the currents are correctly summed to 0.

Electrode placement: In most experiments when electrodes are placed in the brain the distance from the elcetrode to the neurons are usually unknow. The electrodes positions are usually adjusted until they pick up a signal from nearby neurons. To simulate this type of positioning, the electrodes were placed at random locations around the soma.

Electrodes were placed using the simulation class SphereRand (section A.2.1). 1000 electrodes were placed around each soma within a distance of $60\,\mu\mathrm{m}$. This max distance were chosen because even models with the strongest extracellular amplitude were no higher than $50\,\mu\mathrm{V}$ at a distance of $60\,\mu\mathrm{m}$.

As the electrodes were randomly placed in euclidian space the number of electrodes per distance r increases as a function of r^2. Electrodes closer than $15\,\mu\mathrm{m}$ have been ignored since not all models had any electrodes within this distance.

4.2.2 Choosing the Optimal Width Definition

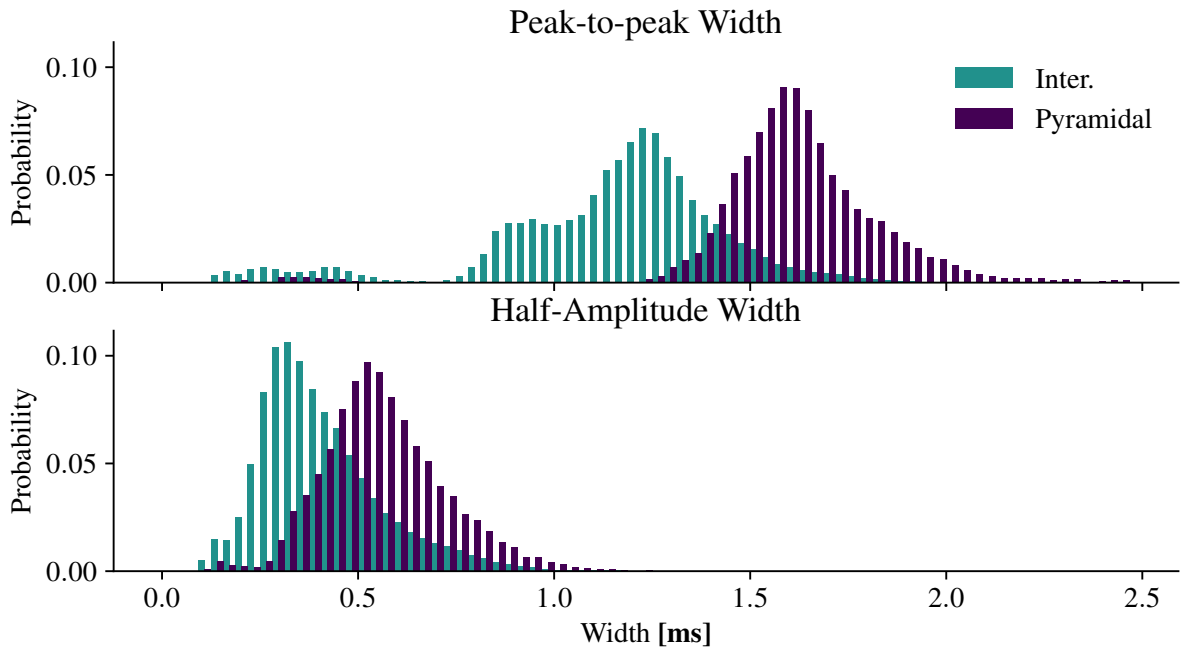

Figure 4.8: Spike widths of all L5 interneuron and pyramidal models from the simulations. Top is peak-to-peak spike width; bottom is width at half amplitude. Width measurements has been binned according to the size of the simulation timestep dt. Width of the bars are half dt. The histograms represent a probability distribution of measuring a width given a neuronal type.

Some investigation has previously gone into which features of the action potential are best suited for classifying neurons. Barthó et al. (2004) evaluated several spike features and concluded that the spike duration most reliably gave the best separation between pyramidal neurons and interneurons. Moreover they suggested that the peak-to-peak width definition of the unfiltered trace reliably gave the better bimodal distributions than the half-amplitude width. Their experiments were carried out in the somatosensory cortex and prefrontal cortex of both anethesiased and drug-free mice.

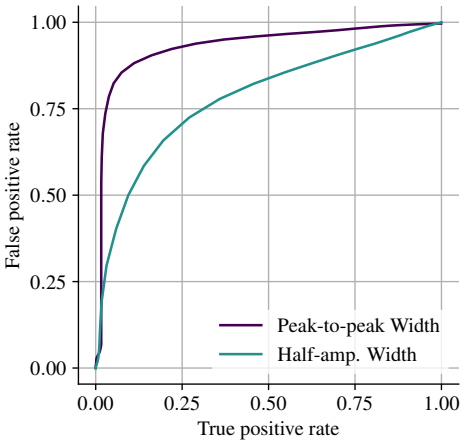

Figure 4.9: Comparisons between each group of models using ROC curves of the width samples of pyramidal neurons and interneurons. Graphs closer to $(0, 1)$ indicate a better separation of the models. The peak-to-peak area under curve is 0.94 while area under curve of half-amplitude width is 0.78.

Width Distribution: A good width definition is recoqnized by having a better separation between interneurons and pyramidal neurons. The extracellular spike width was recorded from each of the electrodes and binned according to the simulation timestep. The resulting histogram was then used for the basis of mea-

suring the separation between neuron models (fig. 4.8).

For both deifinitions the spike width of interneurons are smaller than the width of pyramidal neurons. These findings are in line with previously established research (McCormick et al. (1985), Barthó et al. (2004), and Pettersen & Einevoll (2008)).

The histograms represent a probability distribution of measuring a certain spike width given the neuronal type.

Little overlap of the neurons models signify a good ability to seperate the models into two classes. It is useful to have a metric to measure the separation between models. A proper distance metric for histogram data is already an important component for some machine learning tasks.

Though the classification this data probably would be better suited a machine learning algorithm it is still useful to have a number on the separation between the neuron models. A common measure of the perfomance of a binary classifiers are ROC curves. The area under curve of the ROC has been used as a measure of the accuracy of the classifyer. Figure fig. 4.9 shows the ROC curves of the two width definitions. The AUC was 0.94 for the peak-to-peak definition and 0.78 for the half-amplitude definition. In this case the peak-to-peak definition serves as a better classification criteria then half-amplitude.

Width by Distance from Soma:

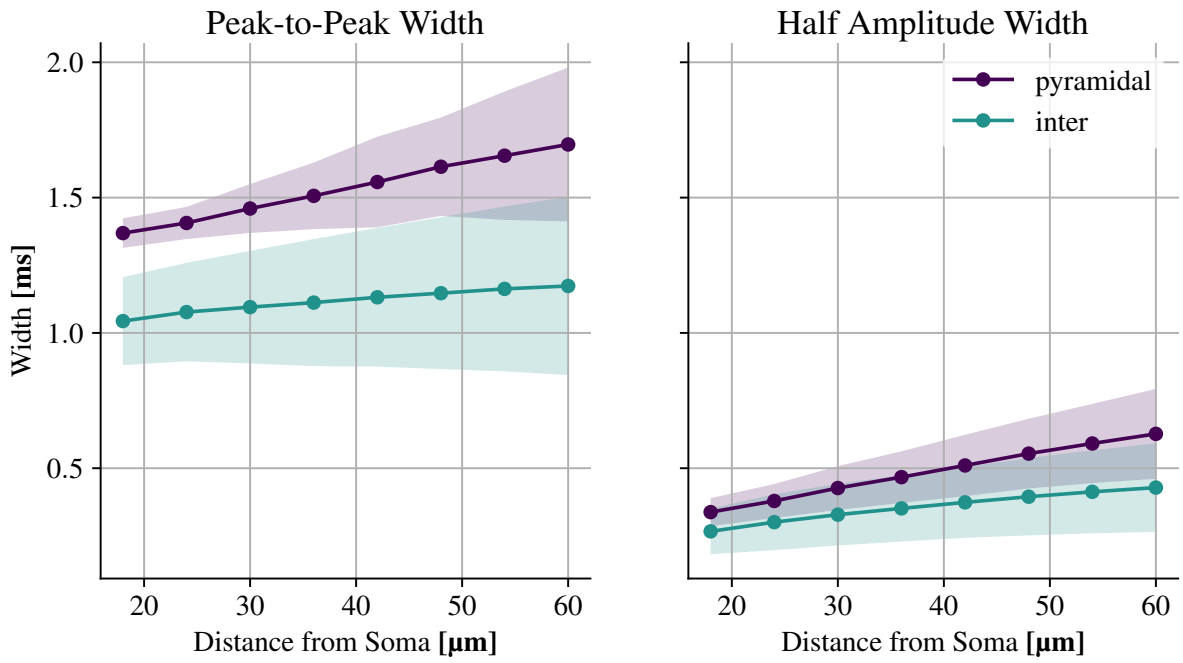

Figure 4.10: Comparison of the spike width by distance from soma using the peak-to-peak and the half-amplitude definition. Mean +/- 1 std. Each measurement from the electrodes have been binned into 8 bins and the mean and standard deviation was calculated. Half-width width shows a higher overlap at all distances than peak-to-peak definition.

The two spike widths were recorded and binned according to their distance from soma (fig. 4.10). The lower and upper bounds of each graph is 1 s.t.d. It is immediately clear that the

separation is higher for the peak-to-peak width.

Variance: To evaluate the precision of the two width definitions the coefficient of variation, c_v, was used rather than the variance.

$$c_v = \frac{\sigma}{\mu} \tag{4.1}$$

Because the variance is of similar magnitude for both width definitions, the overall greater mean of the peak-to-peak width results in a lesser c_v at all distances (fig. 4.11). This suggests that the peak-to-peak width is more accurate than the half-amplitude width. The peak-to-peak width is also generally has longer durations which makes it easier to measure for real electrodes.

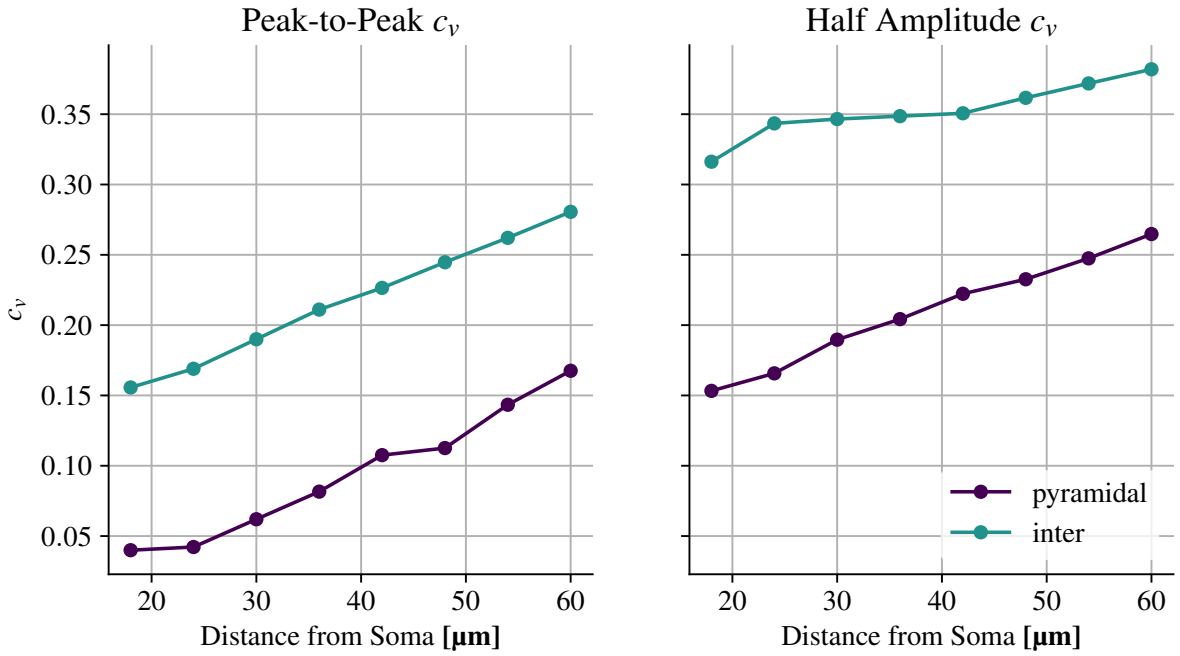

Figure 4.11: Coefficient of variation, c_v, calculated from fig. 4.10. A lower coefficient of variation entail a more precise measurement. c_v is lower at all distances for the peak-to-peak definition compared to the hal-amplitude definitions for both pyramidal neurons and interneurons.

4.2.3 Choosing the Optimal Amplitude Definition

The two amplitude definitions investigated are the amplitude from baseline and the peak-to-peak amplitude definition.

Figure 4.12 shows the amplitude over distance from soma. Though there is a clear separation between pyramidal neurons and interneurons at a given distance, the distance is usually unknown during experiments. The resulting histogram (result not shown) when ignoring distance does not show a clear biomodal distribution and as such is not suited as a classification parameter alone.

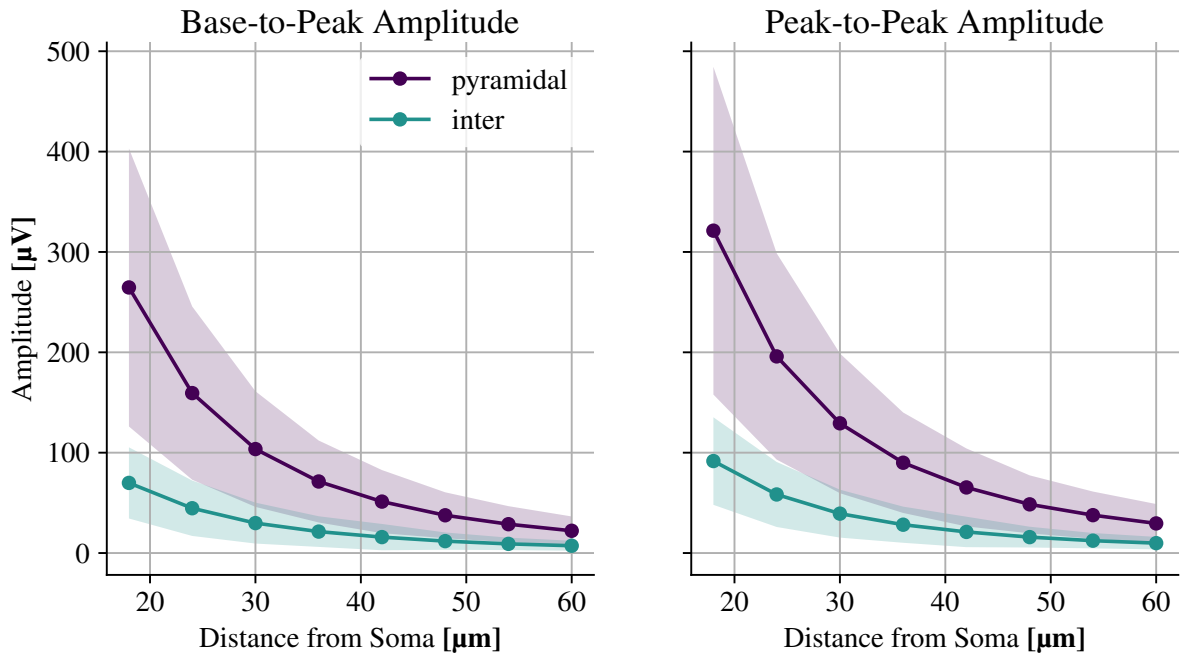

Figure 4.12: Amplitude measurements by distance from soma. Mean +/- 1 std. Analysis is done identical to spike width from soma (fig. 4.10). Based on separation between neuron types neither definition shows a clear advantage over the other as the overlaps are similar.

There seems not to be a clear distinction of which amplitude definition is better. As with the width definition we can calculate the coefficient of variation to compare the accuracy of the definitions. The coefficient of variation (results not shown) is lower for the peak-to-peak width definition at all distances. This can be attributed to that the variance for both definitions are similar while the mean value of peak-to-peak amplitude is always higher.

4.2.4 Combining Spike Width and Amplitude

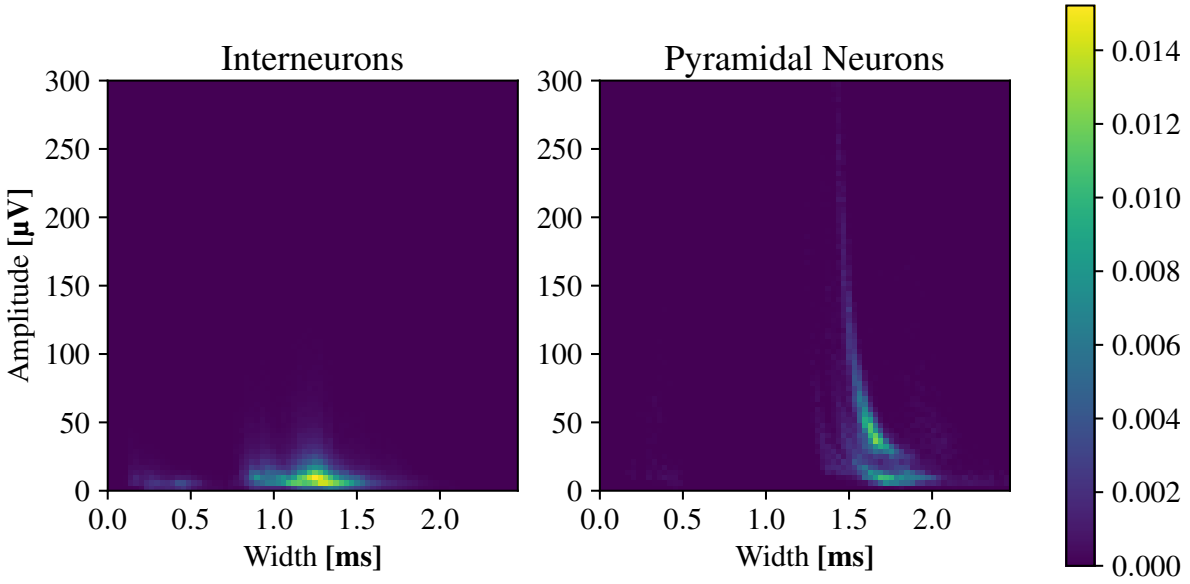

Figure 4.13: Spike width and amplitude histograms of interneurons and pyramidal neurons. The overlap of these two distributions is 5.10%. The overlap of the spike width distributions alone is 12.15%.

As long spike widths from interneurons accompany low amplitudes and short spike widths from pyramidal neurons accompany high amplitudes it is reasonable to conclude that the amplitude could be used as an addition parameter for classification. From this point on the peak-to-peak width and amplitude definitions will be used.

The spike width and amplitude can be combined into 2d histograms as seen in the right column of figure fig. 4.13. The left column shows the histogram only based on spike width. The histograms are made from the spike width and amplitude data from all pyramidal (20) and interneuron models (250) from L5.

When combining spike amplitude and spike width it is no longer possible to apply the AUC similarity metric as was done in Choosing the Optimal Width Definition (section 4.2.2). To compare pyramidal neurons against interneurons the histograms were compared using a similarity metric defined as the histogram intersection divided by the union Histogram Similarity Measure (section 3.3). This metric can be interpreted as the overlap between the two histograms and is a valid metric for both 1d and 2d histograms.

A considerable difference was seen when comparing the overlap between the histograms. When only using spike width the overlap between interneurons and pyramidal neurons was 12.15%. In the case of spike width and amplitude the overlap was 5.10%. This suggests that using spike amplitude as an additional parameter for classification will give a more accurate result. The spike width and amp. 2d histogram of the pyramidal models has a multimodal distribution that could suggest this group could be split into subclasses (fig. 4.13). This could be a result of a low diversity among the pyramidal models as the pyramidal data is based on 20 models versus 240 interneuron models. Though there is a much higher number of interneuron

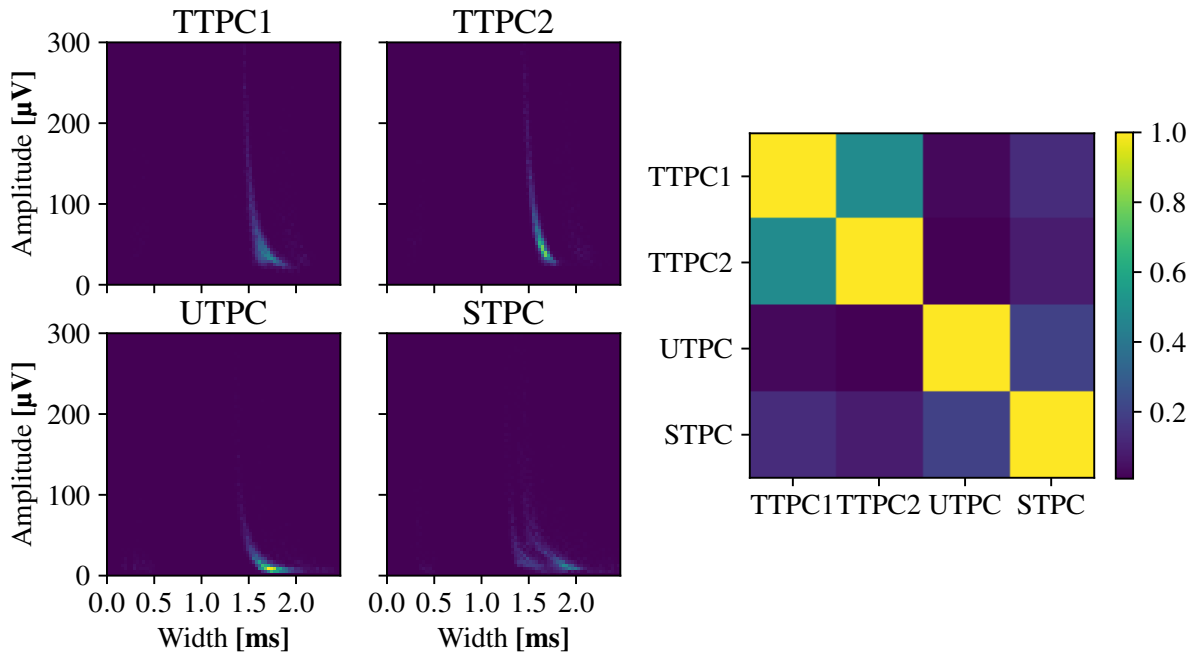

Figure 4.14: Comparison of the pyramidal types. Each group is a differnt morphological type classified but share the electrophysiological type. Right side shows the overlap of each histogram. The thick thufted morphological types (TTPC1, TTPC2) have more in commen than the untufted (UTPC) and slender-tufted (STPC). This suggests that pyramidal cells can be split into subclasses using spike width and amplitude alone.

models, the number of models were based on the diversity of neurons encountered in the mouse somatosensory cortex. If the diversity in the pyramidal models correctly represent the diversity in the mouse brain it will be possible to classify some subclasses of pyramidal neurons based only on spike width and amplitude.

Figure 4.14 shows the histograms of the different classes of pyramidal neurons where each class has 5 models. Each of these classes are me-types but also represents the morphological class, the m-type, as the pyramidal neurons only have one e-type. Since the firing patterns are similar the difference of the histograms show that the morphology has a big impact on the extracellular spikes.

Figure 4.14 also shows the overlap between each of the classes. The two groups TTPC1 and TTPC2 has a high overlap of 50% and as such they cannot easily be distingushed from another. The group UTPC has little overlap with both TTPC1 and TTPC2, but overlaps STPC with 30%. This suggests that the pyramidal cells can be distinguished from each other in at least 2 groups. Furthermore the main reason they can be seperated from each other is their morphology.

The interneurons does not show an as easily distinguishable histogram.

4.2.5 Effects of Filtering

As filtering is a common manipulation of the signal to remove noise it is interesting to see if the separation between interneurons and pyramidal neurons persist when the signals are filtered. Figure 4.15 shows the same as fig. 4.13 but with the signals filtered with a bandpass butterworth filter of rank 1 between 300 Hz and 6.7 kHz using the function `scipy.signal.lfilter`. These values were chosen as they are reported as common values for bandpass filtering.

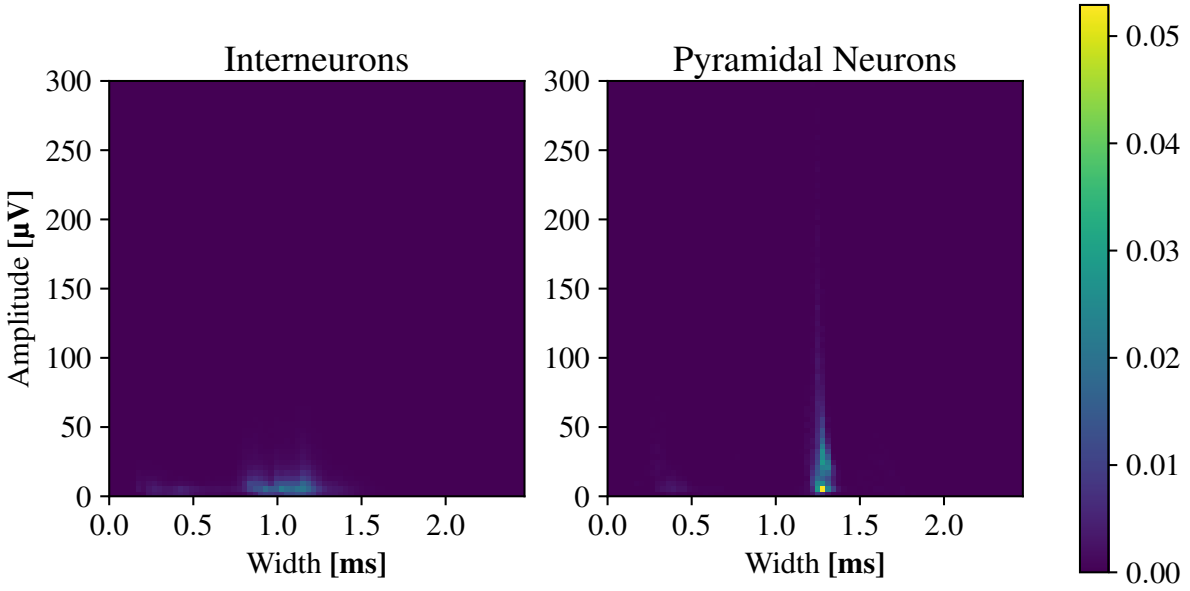

Figure 4.15: Amplitude and width histograms using filtered signals. The shape of the histograms change drastically compared to fig. 4.13, though a separation between interneurons and pyramidal neurons can still be observed.

Though the histograms have changed drastically, the distance metric shows that the models can be separated from each other. The overlap in the filtered amplitude width histograms are 9.90%, an increase from the unfiltered signal that was 5.10%.

In the filtered data the different pyramidal models are closer together than the unfiltered data, though they still retain the separation between them similar as before (fig. 4.16).

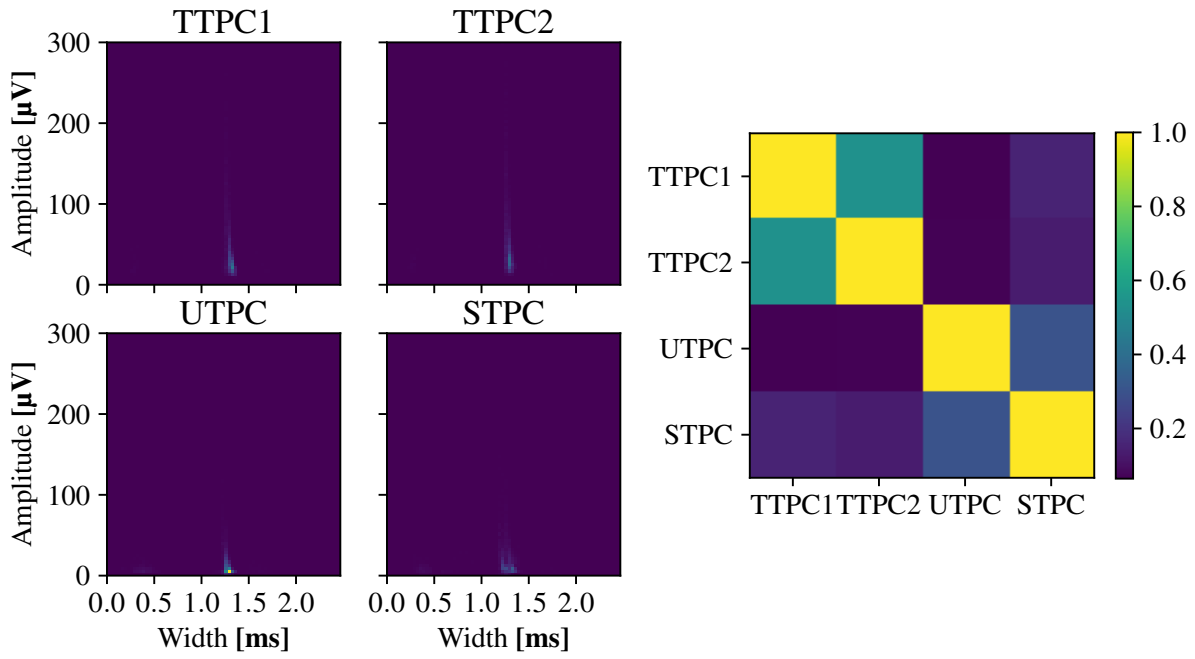

Figure 4.16: Comparison of the individual pyramidal types using the filtered signals. Overlap between the histograms are still maintained.

4.2.6 Comparing Results

Results have been compared to some online sources and a number of articles.

Neuroelectro: The page *NeuroElectro* (2016) (`http://neuroelectro.org/ephys_prop/23/`) has a collection electrophysiological data from a large number of published articles. They store data about several features such as spike width, resting potential, memrbane time constants and other spike data. The data is only intracellular measurements though it gives an insight in the variance of the action potentials in the literature. The duration of the intracellular action potential is significant because it heavily effects the extracellular action potential.

The closest set of data in NeuroElectro to the somatosensory cortex is neocortex layer 5-6 pyramidal cells. The half-amplitude width of these cells were $(1.20 \pm 0.53)\,\mathrm{ms}$. This is a very large variance as the as the intracellular spike widths for current simulations were $(0.70 \pm 0.10)\,\mathrm{ms}$. The sources of this variation is hard to approximate. It is likely due to a mixture of reasons including different species of animals, measuring equipment, preperation procedures, different brain areas, etc. This large variance in spike widths suggests that comparing data must be done in very similar conditions as the models are based on.

Anastassiou et. al: To further compare results the models were used to replicate some results from the paper Anastassiou et al. (2015). The focus of this article was comparing extracelluar measurements to internal measurements and the brain area investigated was also the somatosensory cortex.

They measured the extracellular and intracellular action potential and looked at how the shape changes as the firing rate of the neuron increases. They reported in general that the spike width of neurons increased as the firing rate increases.

Fig. 4c in Anastassiou et al. (2015). shows spike width and amplitude intracellulary and extracellular for a pyramidal cell. The same simulation was carried out for one pyramidal neuron (TTPC1_cADpyr232_1), results shown in fig. 4.17.

The shapes of functions seen in fig. 4.17 has little in common with Anastassiou et al. (2015), and does not increase the way that was reported by them.

The article also gives an estimate of the extracellular conductivity which was estimated by fitting the spike amplitude data to the function of the potential of a point source. $V_e = \frac{I}{\sigma}\frac{1}{4\pi r}$. The values given for the conductivity σ in L5 was $(0.41 \pm 0.24)\,\Omega^{-1}\,\mathrm{m}^{-1}$. The high uncertainty was attributed to the estimation of the membrane capacity. The difference in amplitude between these results and them could be explained by the conductance, as the amplitude is proportional to this variable. Though the shape of the curves are not similar.

Bartho et al.: Barthó et al. (2004) did a comparison of half amplitude spike width and peak-to-peak spike width of multiple neurons in areas including the L5 somatosensory cortex of rats so results should be comparable.

For putative pyramidal neurons the spike widths ranged from $0.50\,\mathrm{ms}$ to about $1.50\,\mathrm{ms}$ for the peak-to-peak duration and $0.20\,\mathrm{ms}$ to about $0.40\,\mathrm{ms}$ for the half amplitude duration. These are quite different from current results where most peak-to-peak pyramidal spike widths lie above $1.5\,\mathrm{ms}$, and the half-amplitude spikes are mostly above $0.5\,\mathrm{ms}$ (fig. 4.8).

For putative interneurons the spike widths are also shorter. All interneuron peak-to-peak widths found to be less than $0.5\,\mathrm{ms}$ which is significantly lower than current results.

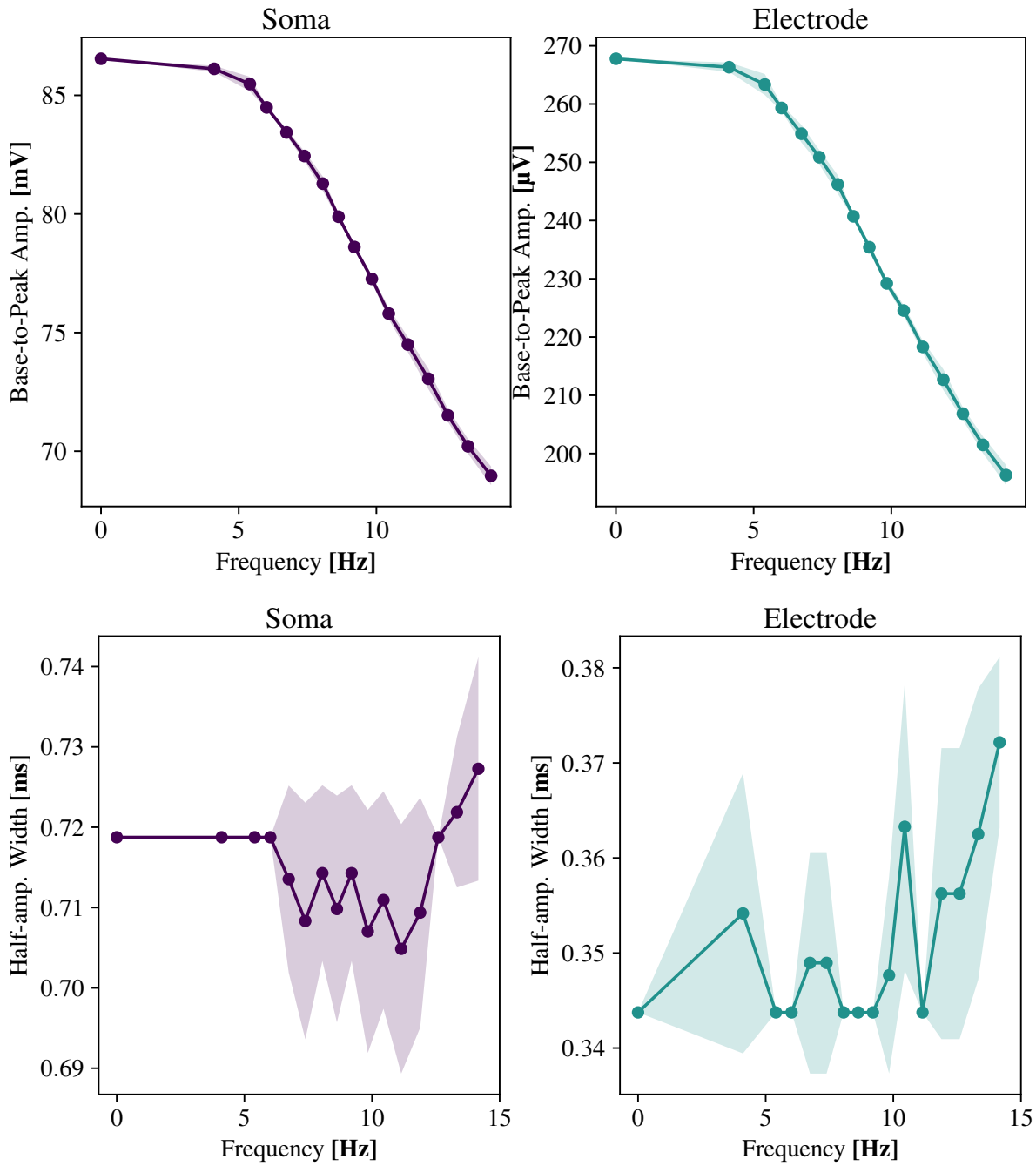

Figure 4.17: Reconstruction of experiments carried out by Anastassiou et al. (2015), fig 4c in their article. Simulation are from a single pyramidal neuron with an electrode placed at a distance of 20 µm.

5 | Discussion

Replication of results from Pettersen & Einevoll (2008) revealed quantitative differences though the general shape of the graphs were similar.

The spikes from interneurons and pyramidal neurons was compared using models from the Blue Brain Project. The results showed that there are differences in the duration of the EAPs which also has been reported in the literature. This led to the conclusion that peak-to-peak spike width definition was superior to the half-amplitude definition. There were also differences in the amplitude of pyramidal and interneurons which was used to further separate the two classes.

When a filter was applied to the simulation the separation between interneurons and pyramidal neurons remained. Indicating that identification can still be done using a filtered signal.

Some features of the spikes were then compared to a online source and two articles, revealing a large variance in the spikes of neuroscience community.

Spike Duration as an Identification Criteria: Mountcastle et al. (1969) initially reported that thin spikes could discerned from the spikes of what they called regular spiking pyramidal cells. The thin spikes were taken as in indication of cortical stellate neurons which was later validated by McCormick et al. (1985).

Recently the validity of using spike duration as an indicator for neuron types have come under question. Firstly, to measure spike durations the measurement equipment must have a good resolution and little noise. This is sometimes difficult, especially with the short duration of the half-amplitude width. It is also important to know what kind of filtering has been done too the signal, as spike shapes tend to change drastically following a band pass filter (Quian Quiroga (2009)).

Additionally Vigneswaran et al. (2011) reported that pyramidal cells in the primate neocortex may have much shorter action potentials than than that of non-primates. If neurons only can be identified for certain animals it will certainly reduce the significance of spike durations. Furthermore Robbins et al. (2013) reported that spike shapes measure near axons of pyramidal cells generate short duration waveform.

Though there has been uncertainty whether spike duration can be used as a classification parameter, current results clearly show that there is a difference between interneurons and pyramidal neurons (fig. 4.8). This rests on the assumption that the Blue Brain models are valid.

The Most Promising Width Definition: Several width definitions were evaluated before simulations were ran. Width definitions that included the baseline of the signal was abandoned because the signals resulted in deceptively long durations caused by the simulation environment (results not shown). This decision was based on two reasons: (1) the simulations does not

have noise so the duration from the start of the EAP to the time it takes the extracelluar potential to reach $0\,\mu V$ again could be long. (2) The initial phase of the EAP caused by capacitive currents are not always positive. In those cases the triphasic shape of EAP were compromised and the width calculation would start earlier than for other EAPs.

Ultimately two spike width definitions were compared against each other, the half-amplitude width and the peak-to-peak width. Using ROC comparison of the width measurements peak-to-peak width showed the best results for identification. This is in agreement with results from Barthó et al. (2004) that suggested the peak-to-peak width gave improved bimodal distributions.

The coefficient of variation also revealed that the peak-to-peak definition had a higher precision than half-amplitude width.

Choice of Amplitude Definition: Two amplitude definitions were investigated. Of the base-to-peak definition and the peak-to-peak definition, the peak-to-peak amplitude was found to have a better precision then the base-to-peak amplitude in addition to having a simple implementation. The peak-to-peak amplitude also avoided having to define the baseline of the signal.

Spike Amplitude Improves Identification: Spike width and amplitude was collected to create two 2d histograms of the interneurons and pyramidal neurons. When comparing the 2d histograms to the 1d histograms, based only on spike width, the separation between interneurons and pyramidal neurons increased. The separation was measure by calculating the overlap between the histograms, which decreased from 12.15% to 5.10%.

The increase in separation when using the amplitude as an additional parameter suggests that amplitude is a valuable feature of the EAPs.

Simulations revealed that pyramidal neurons had in general longer durations and higher amplitude than interneurons. As the distance from soma increases the spike durations became longer for both width definitions, this was also seen in Pettersen & Einevoll (2008).

If only spike width is used for identification, the long interneuron spikes will have similar values as short duration pyramidal spikes. This in turn makes the histograms overlap and identification more difficult. Though because long interneuron spikes tend to have low amplitude while short pyramidal cell spikes tend to have high amplitude makes this area of overlap smaller. This is probably why the amplitude is a valuable feature for classification.

It was seen that the group pyramidal neurons showed an additional bimodal distribution in the amplitude and width histogram. This suggests that subclassification could be a possibility.

Identification still Feasible after Filtering: Quian Quiroga (2009) showed that filtering EAPs can change the shape of EAPs considerably. This was also seen when applying a butterworth filter to the simulations. Therefor care must be taken if the shape of the signal is important. Despite this a butterworth bandpass filter was applied to the simulation to gauge the effects on the separation between interneurons and pyramidal neurons.

The results showed that classification was still possible using the filtered signal. The overlap changed to 9.9% from the previous 5.10%, which is still a reasonable separation.

LFPy_util as a Suggestion for Structuring Simulations: LFPy_util is created to be versatile when it comes to creating new simulations. The main idea of LFPy_util was to be able to split

apart simulations into smaller modules that can be turned on and off. This enables quicker development times as the simulation modules can be tested separately and it enables simple parallel simulations.

If the package I created will not be used by future developers I think the idea behind LFPy_util should still be considered a good way to structure simulations. Usually simulations are created using scripts that are ment to only run one time. The addition of simulation classes I think will make simulations easier to share among programmers and make code more open and understandable.

Future Work Classification of neurons only based on EAPs has proved dificult, but with the advent of neuron models with reconstructed morphology this might change. The grand wish of this work could be to create a database of the different spikes from neuronal classes with an algorithm classify them. This could mean that one could simply input you spikes and maybe other parameters such as brain region and get the type of neuron you are recording from. Though this might be an impossibility, future work might employ tools such as LFPy and LFP_util to identify neuron classes even more accurately.

6 | Acknowledgement

A big thank you to my supervisor Gaute T. Einvoll and Torbjørn Vefferstad Ness for their guidance throughout my master thesis. Their support never faltered and were always available to any questions.

Bibliography

Anastassiou, Costas A. et al. (2015). "Cell type- and activity-dependent extracellular correlates of intracellular spiking." en. In: *Journal of Neurophysiology* 114.1, pp. 608–623. ISSN: 0022-3077, 1522-1598. (Visited on 05/04/2016).

Barthó, Peter et al. (2004). "Characterization of neocortical principal cells and interneurons by network interactions and extracellular features." eng. In: *Journal of Neurophysiology* 92.1, pp. 600–608. ISSN: 0022-3077.

Bean, Bruce P. (2007). "The action potential in mammalian central neurons." en. In: *Nature Reviews Neuroscience* 8.6, pp. 451–465. ISSN: 1471-003X. (Visited on 04/25/2016).

Bradley, Andrew P. (1997). "The use of the area under the ROC curve in the evaluation of machine learning algorithms." en. In: *Pattern Recognition* 30.7, pp. 1145–1159. ISSN: 00313203. (Visited on 08/15/2016).

Brooks, C. McC & J. C. Eccles (1947). "Electrical Investigation of the Monosynaptic Pathway Through the Spinal Cord." en. In: *Journal of Neurophysiology* 10.4, pp. 251–273. ISSN: 0022-3077, 1522-1598. (Visited on 08/10/2016).

Connor, J. A. & C. F. Stevens (1971). "Prediction of repetitive firing behaviour from voltage clamp data on an isolated neurone soma." In: *The Journal of Physiology* 213.1, pp. 31–53.

Connor, J. A., D. Walter, & R. McKown (1977). "Neural repetitive firing: modifications of the Hodgkin-Huxley axon suggested by experimental results from crustacean axons." In: *Biophysical Journal* 18.1, pp. 81–102. ISSN: 0006-3495. (Visited on 11/13/2015).

Dayan, Peter & Laurence F. Abbott (2001). *Theoretical neuroscience*. Vol. 806. Cambridge, MA: MIT Press. (Visited on 11/12/2015).

Documentation NEURON (2016). URL: `https://www.neuron.yale.edu/neuron/docs` (visited on 03/17/2016).

Freund, Tamas & Szabolcs Kali (2008). "Interneurons." en. In: *Scholarpedia* 3.9, p. 4720. ISSN: 1941-6016. (Visited on 08/15/2016).

Gold, Carl et al. (2009). "High-Amplitude Positive Spikes Recorded Extracellularly in Cat Visual Cortex." In: *Journal of Neurophysiology* 102.6, pp. 3340–3351. ISSN: 0022-3077. (Visited on 05/02/2016).

Henze, Darrell A. et al. (2000). "Intracellular Features Predicted by Extracellular Recordings in the Hippocampus In Vivo." en. In: *Journal of Neurophysiology* 84.1, pp. 390–400. ISSN: 0022-3077, 1522-1598. (Visited on 07/31/2016).

Hodgkin, Alan L. & Andrew F. Huxley (1952). "A quantitative description of membrane current and its application to conduction and excitation in nerve." In: *The Journal of physiology* 117.4, pp. 500–544.

Lindén, Henrik et al. (2013). "LFPy: a tool for biophysical simulation of extracellular potentials generated by detailed model neurons." In: *Frontiers in neuroinformatics* 7.

Mainen, Zachary F. & Terrence J. Sejnowski (1996). "Influence of dendritic structure on firing pattern in model neocortical neurons." In: *Nature* 382.6589, pp. 363–366. (Visited on 11/13/2015).

Markram, Henry et al. (2015). "Reconstruction and simulation of neocortical microcircuitry." In: *Cell* 163.2, pp. 456–492.

McCormick, D. A. et al. (1985). "Comparative electrophysiology of pyramidal and sparsely spiny stellate neurons of the neocortex." en. In: *Journal of Neurophysiology* 54.4, pp. 782–806. ISSN: 0022-3077, 1522-1598. (Visited on 06/09/2016).

Mountcastle, Vernon B. et al. (1969). "Cortical neuronal mechanisms in flutter-vibration studied in unanesthetized monkeys: Neuronal periodicity and frequency discrimination." In: *Journal of neurophysiology*.

NeuroElectro (2016). URL: http://neuroelectro.org/ephys_prop/23/ (visited on 08/07/2016).

Nunez, Paul L. & Ramesh Srinivasan (2006). "Electric fields of the brain: the neurophysics of EEG." In:

Park, Seong Ho, Jin Mo Goo, & Chan-Hee Jo (2004). "Receiver Operating Characteristic (ROC) Curve: Practical Review for Radiologists." In: *Korean Journal of Radiology* 5.1, pp. 11–18. ISSN: 1229-6929. (Visited on 08/11/2016).

Pettersen, Klas H. & Gaute T. Einevoll (2008). "Amplitude variability and extracellular low-pass filtering of neuronal spikes." In: *Biophysical journal* 94.3, pp. 784–802.

Quian Quiroga, R. (2009). "What is the real shape of extracellular spikes?" In: *Journal of Neuroscience Methods* 177.1, pp. 194–198. ISSN: 0165-0270. (Visited on 01/28/2016).

Ramaswamy, Srikanth et al. (2015). "The neocortical microcircuit collaboration portal: a resource for rat somatosensory cortex." In: *Frontiers in Neural Circuits*, p. 44. (Visited on 08/01/2016).

Receiver operating characteristic (2016). en. URL: https://en.wikipedia.org/w/index.php?title=Receiver_operating_characteristic&oldid=734576784 (visited on 08/15/2016).

Robbins, Ashlee A. et al. (2013). "Short duration waveforms recorded extracellularly from freely moving rats are representative of axonal activity." eng. In: *Frontiers in Neural Circuits* 7, p. 181. ISSN: 1662-5110.

Spruston, Nelson (2009). "Pyramidal neuron." en. In: *Scholarpedia* 4.5, p. 6130. ISSN: 1941-6016. (Visited on 08/15/2016).

Sterratt, David et al. (2011). *Principles of computational modelling in neuroscience*.

Vigneswaran, G., A. Kraskov, & R. N. Lemon (2011). "Large Identified Pyramidal Cells in Macaque Motor and Premotor Cortex Exhibit "Thin Spikes": Implications for Cell Type Classification." en. In: *Journal of Neuroscience* 31.40, pp. 14235–14242. ISSN: 0270-6474, 1529-2401. (Visited on 11/12/2015).

A | Appendix

A.1 Frequently Used Functions

The simulations use the same functions for detecting spikes, calculating spike width and amplitude. These functions are used frequently and their full implementation is included here. The functions are located in the module `LFPyutil.data_extraction`. Other helper functions are also included in this module.

Peak-to-Peak Spike Width: Spike width is calculated from the absolute maximum value to the follow opposite minimal value. If the absolute value of the minimum is higher than the maximum value the signal will be flipped. The function receives a matrix of spikes, where each row is the values of the potential over time. The returned values are an array of spike times and a matrix with the spike traces. The shape of the spike trace matrix is equal to the input matrix and allows easy plotting.

```
def find_wave_width_type_I(matrix, dt=1):
    """
    Wave width defined as time from minimum to maximum.
    """

    matrix = np.array(matrix)
    if len(matrix.shape) == 1:
        matrix = np.reshape(matrix, (1, -1))
    widths = np.zeros(matrix.shape[0])
    trace = np.empty(matrix.shape)
    trace[:] = np.NAN
    for row in xrange(matrix.shape[0]):
        signal = matrix[row].copy()
        offset = signal[0]
        signal -= offset
        # Assume that maximum abs. value is the "spiking" direction.
        if signal.max() < -signal.min():
            signal = -signal
        idx_1 = np.argmax(signal)
        idx_2 = idx_1 + np.argmin(signal[idx_1:])
        widths[row] = idx_2 - idx_1
        trace[row, idx_1:idx_2] = matrix[row, idx_1:].max() * 1.05
    return widths * dt, trace
```

Spike Width at Threshold: Spike width is calculated a threshold of maximum value, with a default value of 0.5. The argument `amp_option` specifies if which side is considered the spiking direction. It can either be `'neg'`, `'pos'` or `'both'`. In the case of `'both'` the maximum value will be considered the spiking direction, evaluated for each spike. Other input and output variables are the same as peak-to-peak spike width.

```
def find_wave_width_type_II(matrix, threshold=0.5, dt=1, amp_option='both'):
    """
    Compute wave width at some fraction of max amplitude. Counts the number
    of indices above threshold and multiplies by **dt**.
    """
    matrix = np.array(matrix)
    if len(matrix.shape) == 1:
        matrix = np.reshape(matrix, (1, -1))
    widths = np.zeros(matrix.shape[0])
    trace = np.empty(matrix.shape)
    trace[:] = np.NAN
    for row in xrange(matrix.shape[0]):
        signal = matrix[row].copy()
        signal -= signal[0]
        # Flip the signal if the negative side should be used.
        if amp_option == 'neg':
            signal = -signal
        elif amp_option == 'both' and signal.max() < -signal.min():
            signal = -signal
        thresh_abs = signal.max() * np.fabs(threshold)
        signal_bool = signal > thresh_abs
        signal_index = np.where(signal_bool)[0]
        widths[row] = np.sum(signal_bool)
        trace[row, signal_index] = matrix[row, signal_index[-1]]
    return widths * dt, trace
```

Base-to-Peak Amplitude: Amplitude defined as the difference between the maximum value and baseline. Baseline simply set at the start of the signal. Inputs a matrix of spike signals, identical to the previous spike width functions.

```
def find_amplitude_type_I(matrix, amp_option='both'):
    """
    Finds the amplitude of signals defined as the difference from the
    maximum value to the start of the signal.
    """

    matrix = np.array(matrix)
    if matrix.ndim == 1:
        matrix = matrix.reshape([1, -1])
    amp_low = np.zeros(matrix.shape[0])
    if amp_option == 'both' or amp_option == 'neg':
        for row in xrange(matrix.shape[0]):
            amp_low[row] = np.abs(np.min(matrix[row] - matrix[row,0]))
    amp_high = np.zeros(matrix.shape[0])
    if amp_option == 'both' or amp_option == 'pos':
        for row in xrange(matrix.shape[0]):
            amp_high[row] = np.abs(np.max(matrix[row] - matrix[row,0]))
    amp = np.maximum(amp_low, amp_high)
    return amp
```

Peak-to-Peak Amplitude: Amplitude defined as the duration from maximum value to the follow minimal value. The maximum value is considered the spiking direction. Inputs and outputs the same as the previous functions.

```
def find_amplitude_type_II(matrix):
    """
    Finds the amplitude of signals from minimum to maximum.
    """
    matrix = np.array(matrix)
    if matrix.ndim == 1:
        matrix = matrix.reshape([1, -1])
    amps = np.zeros(matrix.shape[0])
    for row in xrange(matrix.shape[0]):
        signal = matrix[row]
        offset = signal[0]
        signal -= offset

        amp_1 = signal.max()
        amp_2 = signal.min()
        amps[row] = np.fabs(amp_2 - amp_1)
    return amps
```

Extract Spikes: For each spike found in a signal, this function cuts out an area around the spikes and returns each of them in a row in the returned matrix. The function was created so the returned matrix can be inputed to any of the previous functions.

`t_vec` must be a 1d array of the simulation time in milli seconds. Spikes will be extracted from the 1d array `v_vec`, that could for instance be from a single electrode or a membrane potential.

`pre_dur` and `post_dur` specifies the desired duration on each side of the maximum value of the spike. If a spike at the end or the start of the signal does not fit with the `pre_dur` and `post_dur` padding, the spike will be ignored. A related function `data_extraction.find_spikes` more simply counts the spikes and can also ignore spikes using `pre_dur` and `post_dur`.

Spikes are detected by local maxima above the `threshold` which is measured in standard deviations. `threshold_abs` can be used to set an absolute threshold for spike detection. This is useful in cases where there are no spikes in the signal or the signal has a "rebound" after the hyperpolarization phase.

```
def extract_spikes(t_vec,
                   v_vec,
                   pre_dur=16.7*0.5,
                   post_dur=16.7*0.5,
                   threshold=4,
                   amp_option='both',
                   threshold_abs=None):
    """
    Get a new matrix of all spikes of the input v_vec.
    """
    t_vec = np.array(t_vec)
    v_vec = np.array(v_vec)
    if len(t_vec) != len(v_vec):
        raise ValueError("t_vec and v_vec have unequal lengths.")

    threshold = np.fabs(threshold)
    dt = t_vec[1] - t_vec[0]
    pre_idx = int(pre_dur / float(dt))
    post_idx = int(post_dur / float(dt))
    if pre_idx + post_idx > len(t_vec):
        # The desired durations before and after spike are too long.
        raise ValueError("pre_dur + post_dur are longer than the time vector.")
    if pre_idx == 0 and post_idx == 0:
        raise ValueError("pre_dur and post_dur cannot both be 0.")

    v_vec_unmod = np.copy(v_vec)
    v_vec = zscore(v_vec)
    if amp_option == 'pos':
        pass
    elif amp_option == 'neg':
        v_vec = - v_vec
    elif amp_option == 'both':
        if -v_vec.min() > v_vec.max():
            v_vec = -v_vec
            v_vec_unmod = -v_vec_unmod

    # Find local maxima (or minima).
    max_idx = argrelextrema(v_vec, np.greater)[0]

    # Return if no spikes were found.
    if len(max_idx) == 0:
        warnings.warn("No local maxima found.", RuntimeWarning)
        return np.array([]), np.array([]), np.array([])
    # Only count local maxima over threshold as spikes.
    v_max = v_vec[max_idx]
    v_max_unmod = v_vec_unmod[max_idx]
    length = len(v_max) - 1
    for i in xrange(length, -1, -1):
        # Remove local maxima that is not above threshold or if the spike
        # shape cannot fit inside pre_dur and post_dur
        check_1 = v_max[i] < threshold
        check_2 = max_idx[i] < pre_idx
        check_3 = max_idx[i] + post_idx > len(t_vec)
        check_4 = v_max_unmod[i] < threshold_abs
        if (check_1 or check_2 or check_3 or check_4):
            v_max = np.delete(v_max, i)
            max_idx = np.delete(max_idx, i)

    spike_cnt = len(max_idx)
    # Return if no spikes were found.
    if spike_cnt == 0:
        warnings.warn("No maxima above threshold.", RuntimeWarning)
        return np.array([]), np.array([]), np.array([])
    n = pre_idx + post_idx

    spikes = np.zeros([spike_cnt, n])
    I = np.zeros([spike_cnt, 2], dtype=np.int)

    for i in xrange(spike_cnt):
        start_idx = max_idx[i] - pre_idx
        end_idx = max_idx[i] + post_idx
        I[i, 0] = start_idx
        I[i, 1] = end_idx
        spikes[i, :] = v_vec_unmod[start_idx:end_idx]
    t_vec_new = np.arange(spikes.shape[1]) * dt
    return spikes, t_vec_new, I
```

A.2 List of Simulation Classes

These are summaries of the predifined simulation classes included in LFPyutil. The constructor of each class is shown which include run parameter, processing parameters and plot parameters. The plot functions are not detailed here as many of them are insignificant for this project.

How to access data from the simulations, documenation and source code can be found at `https://github.com/lastis/LFPy_util`.

A.2.1 SphereRand

The simulation function places electrodes in uniformly distributied locations around the soma within a default radius of 50 µm (`run_param['R']`). Spikes are detected by thresholding the soma membrane potential using the function `extract_spikes` (section A.1). That timing is applied to all electrodes such that all electrodes measure the same part of the simulation. If the signal has several spikes the spike index must be supplied, the default setting uses the first spike.

Many simulations has a run parameter called `ext_method` and decides if neuronal compartments will be considered point sources or line sources in LFPy. The default parameter is always `'som_as_point'` which represents the soma compartment as a point source and all other compartments as line sources.

By default the setting `assert_width` is `False`, but can be used to ignore spikes using thresholds.

When the simulation is finished the the raw signal, the spike widths and amplitudes and distance from soma of every electrode is stored in the `data` dictionary.

Listing 10: SphereRand Parameters

```
class SphereRand(Simulation):
    def __init__(self):
        Simulation.__init__(self)
        self.set_name("sphere")

        self.debug = False
        self.run_param['N'] = 1000
        self.run_param['R'] = 50 # um
        self.run_param['sigma'] = 0.3
        self.run_param['ext_method'] = 'som_as_point'
        self.run_param['seed'] = 1234

        self.process_param['amp_option'] = 'both'
        self.process_param['pre_dur'] = 16.7 * 0.5
        self.process_param['post_dur'] = 16.7 * 0.5
        self.process_param['threshold'] = 3
        self.process_param['width_half_thresh'] = 0.5
        self.process_param['bins'] = 11
        self.process_param['spike_to_measure'] = 0
        self.process_param['assert_width'] = False
        self.process_param['assert_width_I_low'] = 0.2 # ms
        self.process_param['assert_width_I_high'] = 3 # ms
        self.process_param['assert_width_II_low'] = 0.1 # ms
        self.process_param['assert_width_II_high'] = 2.0 # ms
        self.plot_param['elec_to_plot'] = []
        self.plot_param['use_tex'] = True
```

A.2.2 MultiSpike

The simulation function inserts an electrode in the soma compartment until the desired number of spikes are reached. The default parameters will apply square current pulses of a durtion of $300\,\mathrm{ms}$ with an initial amplitude of $100\,\mathrm{pA}$. The desired number of spikes are found using the bisection method. If the number spikes are less then the desired number the amplitude will be increased by 25%. The `pptype` is a string defining the type of measurement electrode used. The string is sent to LFPy and supports custom types of electrodes. For further information about electrodes see LFPy documentation ((http://lfpy.github.io/classes.html)).

When the desires spikes are found the electrode is applied to the simulation environment. This can be exploited to chain simulation classes, as in the example listing 7.

Before a simulation is being run, this class will search for a previous run of the neuron model and attempt to apply the same current found there. This is to prevent unnecessary processing time.

Listing 11: MultiSpike Parameters

```
class MultiSpike(Simulation):
    def __init__(self):
        Simulation.__init__(self)
        self.set_name('mspike')

        # Used by the custom simulate and plot function.
        self.run_param['threshold'] = 4
        # Absolute threshold for a spike intracellular.
        self.run_param['threshold_abs'] = 0 # mV
        self.run_param['pptype'] = 'IClamp'
        self.run_param['delay'] = 100
        self.run_param['duration'] = 300
        self.run_param['init_amp'] = 0.1
        self.run_param['pre_dur'] = 16.7 * 0.5
        self.run_param['post_dur'] = 16.7 * 0.5
        self.run_param['spikes'] = 3
        self.plot_param['use_tex'] = True
        self.apply_electrode_at_finish = True
        self.only_apply_electrode = False
        self.verbose = False
        self._prev_data = None
```

A.3 Function to Load a Cell Object

```
import LFPy
import LFPy_util
import os
import numpy as np
import neuron
from glob import glob

# Location of the models.
DIR_FILE = os.path.dirname(os.path.realpath(__file__))
DIR_MODELS = os.path.join(DIR_FILE, 'res/')

def get_cell(neuron_name):
    """
    Load a spesific model based on a string and return a LFPy Cell object.

    :param string neuron_name:
        String to identify the neuron model that will be loaded.
    :return:
        Cell Object from LFPy.
    """
    original_cwd = os.getcwd()

    neuron.h.load_file('stdrun.hoc')
    neuron.h.load_file('import3d.hoc')

    # Use the neuron name to find the desired model.
    dir_nrn_model = os.path.join(DIR_MODELS, neuron_name)

    # Load mod files of the neuron.
    mechanism_mod_dir = os.path.join(dir_nrn_model, 'mechanisms')
    LFPy_util.other.nrnivmodl(mechanism_mod_dir)

    # The following .hoc files are spesific for the blue brain models.
    os.chdir(dir_nrn_model)
    #get the template name
    tmp_file = file("template.hoc", 'r')
    templatename = get_templatename(tmp_file)
    tmp_file.close()
    #get biophys template name
    tmp_file = file("biophysics.hoc", 'r')
    biophysics = get_templatename(tmp_file)
    tmp_file.close()
    #get morphology template name
    tmp_file = file("morphology.hoc", 'r')
    morphology = get_templatename(tmp_file)
    tmp_file.close()
    #get synapses template name
    tmp_file = file(os.path.join("synapses", "synapses.hoc"), 'r')
    synapses = get_templatename(tmp_file)
    tmp_file.close()
    neuron.h.load_file('constants.hoc')

    if not hasattr(neuron.h, templatename):
        # Load main cell template
        neuron.h.load_file(1, "template.hoc")
    if not hasattr(neuron.h, morphology):
        # Load morphology
        neuron.h.load_file(1, "morphology.hoc")
    if not hasattr(neuron.h, biophysics):
        # Load biophysics
        neuron.h.load_file(1, "biophysics.hoc")
    if not hasattr(neuron.h, synapses):
        # load synapses
        neuron.h.load_file(1, os.path.join('synapses', 'synapses.hoc'))


    for morphologyfile in glob('morphology/*'):
        # Instantiate the cell(s) using LFPy
        cell = LFPy.TemplateCell(
            morphology=morphologyfile,
            templatefile=os.path.join(neuron_name, 'template.hoc'),
            templatename=templatename,
            templateargs=0,
            tstartms=0,
            tstopms=300.,
            pt3d=True,
            timeres_NEURON=2 ** -5,
            timeres_python=2 ** -5,
            passive=False,
            v_init=-70,
            )
        # Reset back to the previous working directory.
        os.chdir(original_cwd)
        return cell
```

```
def get_templatename(file_template):
    '''
    Assess from hoc file the templatename being specified within
    Arguments
    ---------
    file_template : file, mode 'r'

    Returns
    -------
    templatename : str

    '''
    file_template = file("template.hoc", 'r')
    for line in file_template.readlines():
        if 'begintemplate' in line.split():
            templatename = line.split()[-1]
            continue

    return templatename
```