**BABEŞ-BOLYAI UNIVERSITY CLUJ-NAPOCA**
**FACULTY OF MATHEMATICS AND COMPUTER SCIENCE**
**SPECIALIZATION Informatică Engleză**

# DIPLOMA THESIS

# Representationally sane user interfaces on the dual spaces of quasicrystals

**Supervisor**
**Benţa Kuderna-Iulian**

*Author*
*Bodică Septimiu-Călin*

2022

ABSTRACT

The field of human computer interaction (HCI) can be dated to the 1970s and 1980s, when researchers studied user interface design in the newly emerging field of personal computers. Even now, despite a great advance in the capabilities of automated systems, computer systems require human input to be configured, maintained, and analyzed, as well as interpreted. Sometimes ease of use is sacrificed so that to make a system verifiable, reduce the number of potential bugs, issues and prevent undefined behavior. Even with intelligent systems like AI, which use images for result retrieval or accept voice queries, there is a long gap to bridge between a result that is valid and one that is human-interpretable.

This thesis aims to analyze the topic of user interfaces (UI) for the purpose of identifying separable components that are invariant with respect to the user's perception of them. A framework is provided to collect data and study the user's perception of a UI with the objective of reducing mental fatigue and converging to universally viable UI designs, meanwhile providing a coherent and mathematical justification for these choices.

# Contents

# Chapter 1

# Introduction

## 1.1 Background and motivation

Computer software are sometimes complex, requiring an elaborate specification and code to perform even simple tasks. Depending on the programmer's preferences and biases, different approaches may be taken when solving a problem. A programmer's understanding of another one's solution may be different than the original author's expectation, and an author's own work may seem alien to themselves when the work is taken out of context. The process of peer review is based on sharing different perspectives, and over time can generate consensus in the form of standards and conventions. One such example is the Python Enhancement Proposals (PEP), which are founded on principles recorded in the "PEP 20" document. These standards evolve slowly over time as the community gains experience with using the tools and come to the same conclusions by sharing their experiences.

Code naturally goes through a process involving compilers, because for a human it is much easier to work with representations resembling natural language. The compiler accepts this natural language and produces machine-interpretable code automatically. A result of compilation is that regardless of subjective differences in implementation, the same outcome is achieved, by forcing the users to respect key constraints in the language specification. The constraints are enforced by introducing compilation errors, warnings, suggestions.

Linters are extensions that programmers can install in their IDE to statically analyze code style and improve the chances of a correct outcome. They work independently from the compiler but can still improve the programmer's work, while at the same time the linter being maintained independently of the compiler, which is good because compilers are very complex tools that change slowly. A linter can be updated regularly with the help of community driven feedback. There are many such linters for the language Python, which enforce the PEP standards.

Compilers can reduce multiple solutions to machine code approaching the same final representation, through optimization and rewriting processes. This abstract final representation is hard for humans to interpret. To circumvent this, small and well understood separable components are composed to form large objects or data types.

A fully fledged application can have many interacting components, which need to be configured properly so that no unpredictable behaviour can occur. Sometimes, however, these individual components have large similarities and even shared logic between them.

**In this work**, a framework for profiling user actions is described. The framework attempts to generalize the process of harnessing multimodal signals from a user interface, separating aspects of any potential GUI into a maximal set of independent parameters. In the first part, a mathematical basis for discussing the general aspects of user interfaces is proposed.

The second part is a discussion about performing tasks such as correlating externally collected data about the user, as biometric data, sensor fusion of biometric modalities, muscle tracking, and others, and using this plentitude of data to form valid, useful and sane hypotheses about the experience of the user, for the purpose of designing UIs that satisfy the largest number of possible users.

Of particular interest are the representation of the application's components, including the data structures employed. The goal is to identify what dual problems exist across seemingly separate aspects of the UI with the goal of achieving a minimal representation of UI actions. The framework proposes a way of matching the user input to the UI's response, as well as the user's response to events visible on the GUI.

The final goal of this framework is to draw conclusions about a possible correlation between the underlying mathematical description of the perceived environment of a user, and the user's conscious and unconscious ability to perceive the rich environment, or the user's failure thereof.

Human-computer interaction (HCI) can be further discussed as well as neural network's possible interpretation of a UI. By contrasting the human brain with neural networks, the objective is to gain insights into what elements can be eliminated and what alternative representations there are that achieve equally good results to classical UI design patterns.

The benefit of generalizing HCI is the ability to translate the conclusions into already existing user interfaces. One advantage in this pursuit is that a UI has an underlying *specification* that is often provided as a nested hierarchy, a graph, or an XML. Finding a correlation between a user's profile and a specification language can then be used to adapt UI behaviour to the optimal choice. If this solution is provably

sub-optimal, the parameters can be slowly updated so that learning to transition to the desired outcome is as comfortable as possible, as seen by performing analysis and prediction on the collected UI interaction metadata.

## 1.2 Human-computer interaction

Human-computer interaction (HCI) is concerned with the similarities and dissimilarities between the human and the computer. The two are viewed as systems, with special attention to the human brain, which can be seen as the counterpart of the computer. Lately, neural networks and AI systems have grown significantly. This work is written with the consideration that the human brain and a neural network can both use a UI, and might be used interchangably.

The data gathered about the usage of the UI can be processed and interpreted in order to gain knowledge about specific usage patterns, inefficiencies, and design and improve systems. It is closely related with the fields of neuroscience, computer science, physiology, network dynamics, and others. An overlap may also be seen between HCI and BCI, the emerging fields of virtual reality, augumented reality and extended reality (AR, XR), and the Web 3.0. The reason for this is that increasingly large quantities of data are being viable for inference by edge-devices that are part of and form larger cloud-native structures. The user interfaces may see a transition to a more human embodiment, while more complex queries may be formed by combining recommendation engines with other input modalities such as brain waves, voice input, body tracking, eye tracking, and personalized insights.

Combining multiple streams of data from different sensing domains of a system can improve the quality of each individual signal (e.g. Kalman filters), increase predictability of a system, and decrease the susceptibility to subsystem failure. Sensor fusion is concerned with such multimodal analyses. Common methods include voting mechanisms or combining multiple agent's sensing to achieve coordinated swarm behavior. [1] User feedback can be considered such a complex swarm behavior.

In biological systems we face high complexity, as there are many measurements that can be taken, and the explanations for the change in a signal often involve interactions of a large number of subsystems, which is difficult to simulate. Often but not always the amount of data that *can* be collected is much larger than what is needed to achieve good results. UIs generally follow principles of simple designs that are easy to understand, and as a result the data that can be collected about using a UI is as simple. Sometimes, additional measurements can be made, or one wants to consider how the usage patterns of a large number of users might be interpreted. With such considerations, the complexity increases, but can still be useful. As will be

shown in the discussions, forming hypotheses about the reasons for which random variables or joint probabilities might be correlated is very complex. Sometimes, automated theorem provers, or decision support systems, or proof assistants, are the best tools to navigate the complexity of the human body's interconnected measurable signals, as human language will probably not be useful.

Measurements that have use potentials include heart rate, blood pressure, electrodermal response, and EEG, which could all be used to analyze the body's reaction to various tasks.

In artistically oriented tasks, such as drawing, sculpting, writing, EEG is of particular interest, with varying spatial and temporal correlations depending on the nature of the task [2]. Recent progress in the field of virtual reality (VR), show that eye tracking might be an easy source of valuable signals. The headsets in the future could incorporate localized EEG, which is also discussed in the following chapters.

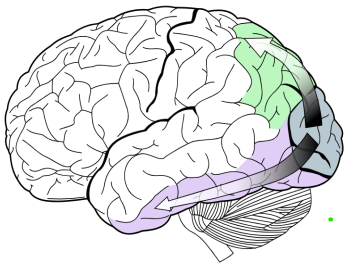
Electrocorticography approaches the ideal scale for measuring the highly complex human brain. For a long time only optical imaging or magnetic sensing was achievable, but projections show that research groups such as Neuralink can increase the resolution with advanced materials. While EEG is not as spatially precise as electrocortigraphy (ECG), choosing the right number of electrodes and sampling frequency can provide enough statistical evidence for localizing a significant amount of brain activity. If the task has a high compatibility with the scalp measurements, the cross correlations between the channels can be used as a brain-computer interface (BCI). The computer systems are the medium to achieve real-time interpretation of the brain signals. Computational models using statistical techniques for feature extraction together with machine learning models are the methods used.

Recently BCI have been used with more success for communicating with disabled patients, where neurofeedback systems co-adapt with a learning patient in order to use raw brain activity guided by sound to produce natural language text. Because of the structured nature of brain connectivity, electric activity measures can be interpreted differently based on the location where they occur. Some theories about how signals propagate through the brain are compatible with the concept of receptive fields from the study and design of artificial neural networks (ANN). In the case of visual perception, the raw optical input arrives in the hindbrain, or occipital area, and gets diffused along the temporal and central axes, eventually having the potential of reaching the frontal lobe.

For the receptive fields of the visual cortex, one could expect the electrodes T3, T4, T5, T6 to activate for handwriting because they are located close to the temporal lobe, and more predominant C3, CZ, C4 activity for drawing faces, with the most raw input being found in the occipital region (Fig. 1.1). The apparent justification for such a hypothesis is that the cortical homunculus associated to the facial nerves has

been identified to be closer to the central region of the electrode map. [3] At the same time, both of these regions are in proximity to regions associated with high order visual areas. Thus the signals are dependent on the sensory stimuli. The signal's localization can also depend on the patient's profile, and personal characteristics. The problem is that the space of hypotheses to test with respect to the neural signals is very large. There could be problems with associating signals from the temporal lobe to those on the central lobe because there might be hidden variables such as obstructed or unmeasured regions on which the signals depend. For this reason a decision support system or proof assistant may be better suited to probe the realm of possibilities. ...*cite ontological studies

Figure 1.1: Ventral-dorsal stream



In the second part of study the framework presented is used to determine if data from a hand drawing task to can make a prediction about the user. The evolution of the user's state should then be used to predict future behavioral patterns, and if possible to guide the user into a favorable state by changing the parameters of the UI. It is observed that there is a similarity factor between hand drawing tasks and using a cursor to interact with a 2-dimensional GUI. For this reason, hand-drawing concepts and UI interaction concepts may be used interchangeably.

The approach is building a framework that is able to use data that about UI activity in order to personalize the user experience (UX). The identified parameters are the main tools for this end goal. It is not enough to have the parameters alone, but the choice of parameters must be justified and based in mathematically provable facts. In the methods section a description of the mathematical model shows that a complexity measure is a valuable tool for identifying the features of a UI. A large body of mathematics focused on representation theory is referenced. By connecting representation theory to user interfaces, an enriched discussion of user interfaces, feature extractors, gesture detection and finally data structures is presented.

# Chapter 2

# Related work

The reader is safe to assume from the references that familiarity with the usual software engineering concepts, data structures, artificial neural network glossary are useful. Most of the work presented in the framework aims to provide the background for discussing UI concepts, and ideas to improve designs. To do this, mathematical language is used with a background in linear algebra, group theory, geometry. Because the environment in which the application runs is a Turing machine built on *von Neumann* architecture, there is an implicit mathematical toolset for discussing the complexity and computability of the application. The problem arises when making the transition to the highly nonlinear and complex human brain, and even some larger neural network models. Even the mathematical models described in Chapter 3 should only outline a problem that is intrinsically very complex.

To illustrate a potential bridge between the simple mathematical description and the rich data structures computed, and the complicated physics involved in HCI, consider the classical problem of the brain-computer interface (BCI), which has been a subject of research that dates back as far as the 1930s. A BCI has to solve a multitude of tasks: abnormality detection, signal classification, pattern detection and predictions on a certain window, generating synthetic signals and interpreting the data with respect to other sensor modalities, such as heart rate or muscle movement. To solve all of these tasks at once, the analog signals must be converted to digital ones and stored. Using a computer with an operating system has the advantage that the solution provided can be event-driven, and human-readable data structures can be used in specifying the program's behavior. The events and data structures have a nested or hierarchical representations, because some of the tasks are inherently subordinate to others. This tends to form a so-called "Pipeline", where the raw signal is transformed successively and branched into parallel processing steps. For example, to classify a multi-channel EEG signal, one would take the raw signal, obtain an interpolated (compressed) digital equivalent at a sampling rate $\omega$, obtain the STFT (Short-time Fourier transform) which is a spectrogram, feed the image into a com-

puter vision machine learning model, and output the confidence of each prediction, given by the softmax function:

$$\sigma(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}},$$

which is a linear activation function.

This example is shown to highlight the fact that a neural network consists of linear activation functions as well as nonlinear ones. In the exemplified pipeline, most transformations can be considered to be linear operations, with the exception of the fully connected neural layers of the ANN. Chaining multiple layers together is considered highly nonlinear and difficult to predict. Furthermore, the human agent can be considered a part in this pipeline, or part of a more complex human-in-the-loop system. The only way to cope with the complexity of nonlinear systems is to add constraints to either their input or the behavior of the system instead. By doing this some properties can be derived and they can even be simulated to a certain (often small) degree.

Other scenarios exist using different techniques and are detailed in the next subsection. To conclude the introduction to this chapter, the purpose of the thesis is reiterated: to obtain a correspondence between highly deterministic aspects of user interfaces, and highly complex and nonlinear domains such as the human brain.

## 2.1 Signal processing

To detect abnormalities, the signals are analyzed in sequences of batches of a certain length of time. Statistics are extracted at the granular level of individual time windows, or the correlations between them can be considered instead. The field of digital signal processing (DSP) is very broad, and includes fields such as spectral analysis, Fourier analysis, and the study of nonlinear systems. Distinct from digital signals, analog signals are continuous in the time domain. An analog signal can be very precise up to a certain level where the noise is inseparable from the signal changes. It is not of interest to elaborate on the different measures of the quality of the signals for the scope of this work, but note, without naming them, that there are such measures.

Sometimes specialized hardware is used to convert the analog signal which is obtained from the mechanical or electrical sensor, into a digital signal, without delegating this responsibility to the computer, using an analog-to-digital converter (ADC). The inverse task is performed with a DCA, but in the processing pipeline we are interested in the digital signal. If the instrument possesses an ADC, then the signal that is obtained from the instrument is already split into discrete values at a

certain sampling rate $\omega$ measured in Hz. The discrete set of values can be reconstructed from the samples using numerical approximations. The reader is refered to the following tools used throughout the paper: STFT (Short-time Fourier transform), Time-frequency analysis (e.g. wavelet transform). The wavelet transform can be used to define a Hilbert basis, and is of particular interest because of its ability to compress a large quantity of EEG signals into orthornomal wavelets, which makes it easier to visualize as well.

$$\langle \theta \mid \pi \rangle \tag{2.1}$$

For predicting signals, there is a problem of choosing the right time window, and correlating the environmental factors with the signal to obtain a probability of correctness associated with the prediction. Imagine that the raw signal is used as input in a prediction pipeline, the prediction of the next 5 second interval might not only depend on the previous 5 seconds, but also on macro-statistical indicators that are correlated with the perceived environment. As habituation occurs the signals could change as a function of biological factors or personal traits that vary across individuals too, so there is the dimension of cross-subject statistical indicators. Since this unfolds in the time domain, it might be considered a highly complex dynamical system. Even a pipeline of hand-crafted features could be full of error outside a very narrow set of constraints in which it can be used. A small neural network has been shown to produce results similar to differential equation solver, so one could assume that using a complex multi-layer ANN can produce a good prediction of the signal, in a similar fashion to the manually crafted feature extraction pipeline. The problem now is understanding the justification of the prediction, which is likely to be an abstract list of neurons weights with no interpretable meaning outside of the neural network itself. The following section discusses the problems raised by using layered neural networks for signal processing tasks.

## 2.2 Stroke embeddings

Compositional stroke embeddings (CoSE) are a way of representing hand drawings as a series of pen strokes, and interpret them using transformers and self-attention based neural networks. [4] These paradigms of neural network architecture design are succintly explained in the following subsection.

The discrete nature of the events represented by brush stokes enable RNNs to make predictions about the next set of strokes. It has been shown in other language-related RNN studies, that the possibility of breaking a problem down from word level to character level can yield better representations of the latent space, as well as

allow phonetic structures to be considered. This has offered an advantage to some cross-lingual neural machine translation (NMT) tasks. **cite**

CoSE uses a dataset of diagrams to make a prediction about the next components of the diagrams. As such it is a generative model. It is a complement to the other architectures that solve tasks related to handwriting, language and text. The latent space of the model is closer to the bottleneck section of the model, which is what gives it the ability to generalize toward previously unseen data. In the study, one of the aims is to discuss the potential of optimizing this latent space by using the body of mathematics known as representation theory. As seen in the introduction, the proposition is that the way of generating periodic and aperiodic tilings is not straightforward, and serves as a good starting point for understanding complexity, information entropy, and generalizability. In fact some problems as the Domino Problem (see Wang dominoes) are closely related to decidability and undecidability problems, and the halting problem of Turing machines.

## 2.3 Artificial neural network (ANN) models

Neural network models can achieve better results than hand-crafted pipelines by being able to internally represent long term temporal dynamics. RNN and LSTM architectures can be used to predict a future time window. An RNN or recurrent neural network is a layered neural network which can accept sequential input and is capable of integrating data about the positioning and context of individual items from the input sequence. This is very popular for natural language tasks and has had some success with time series data. LSTM or long short-term memory is a take on RNNs that improve their contextualizing abilities by using a specific wiring between the layers called the attention architecture that improves language understanding. One problem with using such models on time series data is that the inputs can be extremely long. The data on which the RNN and LSTM models are usually trained are not very large bodies of text, unless it is an experimental model like GPT-3 which can handle large texts but has a much more complicated architecture and takes a lot of computing resources to train. Consider a 5000 word book. At a sampling rate of 256Hz, it would take only 19 seconds or 0.3 minutes to gather a 5000 sample signal. GPT-2 and GPT-3 accept somewhere around 2048 words, and have billions of parameters that need to be trained. As a result of the model complexity, it is able to solve many sub-tasks such as named entity recognition and translation. The weights propagate from layer to layer through the network when training, so it is possible that the individual units that solve these sub-tasks improve the performance on the other tasks as well.

This raises the question of localizing neural activations in a neural network.

There are tools to achieve this since every aspect of each neuron is completely known by the computer. For the human brain it is more difficult, with instruments such as fMRI, BOLD imaging, and EEG. LORETA is a standardized way of localizing electrical activity in the brain from electrical signals. It is possible to reason about event related potentials (ERP) by understanding the functional connectivity of the brain regions, using a tractography. [5] As mentioned in the introduction, forming a hypothesis about what is correlated and what is not can be nearly impossible with natural language. Techniques such as probabilistic graphical models (PGM), combined with proof assistant systems should be used. These systems take as input a set of constraints and use the measurements to produce a theorem based on those constraints, by techniques such as linear programming. Anumachipalli et al. have done research on using electrocortigram (ECoG) signals in the ventral sensorimotor cortex (vSMC), superior temporal gyrus (STG) and inferior frontal gyrus (IFG) with RNNs to translate imagined speech. [6] The sensorymotor cortex encodes the muscle activations of face at the syllable level in different locations. The ECoG sensors have enough spatial resolution to distinguish the coding of the imagined speech. ECoG have a higher spatial resolution than EEG for localizing brain activity, but they are a very invasive method.

Convolutional neural network (CNN) architectures are among the oldest and most widely known. CNNs have been used with high success for predicting epilepsy or depression. [7, 8] These architectures take a simple 1-dimensional tensor as input, but it is also possible to consider classifying spectrograms of the signals, maintaining a high evaluation score but applying the convolutional network on an n-dimensional time-frequency representation. [9] RNNs and CNNs still have a very large number of parameters, and have problems with small datasets and generalizing.

Other neural network models include generative models, which solve the task of generating synthetic signals of a specific class that could closely resemble real recordings. A generative adversarial neural network (GAN) is trained by using a so-called discriminator. The model takes as input an image from a dataset, and during training tries to match the output with the input. During inference, only vague information about the desired image is provided, and the network uses the learned aspects of the dataset to synthesize new, unseen outputs, that are indistinguishable from real samples. Because signals are just 1-dimensional samples, they can be formatted as an image, or instead their spectral representation can be provided. The GAN architectures are closely related to the encoder-decoder paradigm of neural network architecture design. They can do tasks similar to GANs, but instead of generating a signal they some use the learned features for accurate classification. An experienced reader might notice that the representation of some models are just the

inverted topology of other ones, or sub-modules connected together. The GAN has a dual bottleneck architecture. Their representation could prove insightful in the later discussions about representation theory of this study. [10, 11, 12] These architectures have higher potential of being able to generalize away from the limitations of a dataset while still maintaining valid results.

Autoencoder based architectures can learn deep features from a signal. A powerful set of emerging neural network architectures named graph neural networks allow deeper insights into what kind of features the networks learn. They allow better human interpretability of the learned representation, as explained in the EEG-GNN work. [13]

# Chapter 3

# Methods

This chapter is structured in 3 parts. First, the environment and tools are described. Then, a mathematical description of the problem is given. Then, the mathematical description is referenced to make arguments about software-related aspects. The gap between neural networks and the human brain is outlined with remarks about the measures of complexity of the user interface using representation theory. The concept of latent space of a neural network is very useful in relating the two worlds. Finally some extensions are suggested, and some results are provided about using feature extractors and neural networks to aid this pursuit. The framework presented should be thought of as a human-in-the-loop ecosystem that has no intention of being complete but should be a tool based in mathematically sound descriptions that can be extended.

## 3.1 Software

### 3.1.1 Overview

The main piece of the application is the drawing board. This chosen because of the fundamental way in which drawing captures kinematic data. The user's unique way of representing objects through hand drawing has been already studied, and different user profiles such as ones based on geographic region have already been documented. There is ample data to support building hypothesis about this topic. First of all, handwriting data itself is a kind of hand drawing task. Then, languages are already localized in geographic clusters. It was hinted already in Chapter 2 that drawing of diagrams is also related to handwriting in its own particular way. A neural network's representation of a dataset of hand drawn diagrams is very similar to handwriting because individual strokes compose larger constructs. What is omitted is the phonetic aspects of speaking a language, which goes to show that humans represent concepts such as languages by integrating separate perspectives,

like speaking, writing words, writing sentences, reading sentences into one whole. This is even more interesting when using languages composed of syllabic scripts mixed with multiple writing systems with numerous homographs, like Japanese and Kanji.

A drawing is composed of strokes, which is the interval of time between the moment when the pen is activated, and deactivated. Basic kinematic data comes with this such as velocity and acceleration. The shape drawn in a single stroke may be simple, such as a line, or complex, such as a set of curves or splines. Other dimensions of the stroke are the colour used, the thickness and shape of the brush, the number of cursors, the orientations of the cursors, the symmetry axes of the cursors and their relative positions.

Let there also be another category of information about the strokes that is called second-order characteristics. These are as follows: order of the strokes (FS), family of slopes (SS), time of initiation of a stroke (TS), and the relationship of a stroke to past and future strokes (RS). RS is expanded on within our framework by using a recommender system for a future stroke. The recommender comes with different heuristics for proposing a new stroke, with special interest on forming self-similar patterns, or following a probability distribution based on the length of a sequence of recommendation, for example.

### 3.1.2 Mathematical description

First the data structures of importance throughout the discussion are specified.

The objective of the task is the tiling, composed of edges:

$$L = \{(p_i, p_j) \mid i, j \in \mathbb{N}, p \in P_c\}. \tag{3.1}$$

Denote the related set $P_c$ of points within the tiling edges:

$$P_c = \{(s_x, s_y) \mid s \in \mathbb{R}\}, \tag{3.2}$$

such that

$$\forall l = (p_i, p_j) \in L, p_{ij} \in P_c. \tag{3.3}$$

A distinct set from $P_c$ is $P_s$ which is all points that can be sampled on the canvas.

$$P_s = \{(s_x, s_y) \mid s \in \mathbb{R}\}, \tag{3.4}$$

with $P_c \subset P_s$.

The current task is denoted as:

$$D = \{\forall s \in P_{c,s}\} \tag{3.5}$$

It should be noted that the input of two distinct entities or game actors are involved in the final result: the user and the game engine. While the resulting state is a composite of $P_c$ and $P_s$, both need to be stored separately by the game engine, for a few reasons. First of all, inputs could overlap, in fact it is the user's objective in certain game modes to perfectly draw over the game's input. To measure the similarity of the player's input and the optimal game engine's suggestion, the sources of the two sample points need to be distinguished.

This consideration is important when generalizing to multiple agents or decomposing the game engine's input into multiple recommendation styles.

Note that more or less, the points $s \in D$ correspond to pixels or paint strokes on the whiteboard.

In equation (3.4) we defined $P_s$ which is a very general space of all points that can be sampled. We expand this into the set of strokes $H_s$ that can be collected, where H stands for *history*, which ultimately belongs in the state machine. Let

$$H_s = \{\forall s = \{p_i, ..., p_j \mid p \in P_s\}\}, \tag{3.6}$$

with each stroke $s \in H_s$ and define the map,

$$\theta : \mathbb{N} \longrightarrow \mathbb{R}^2, \tag{3.7}$$

which maps all points from the stroke onto an interpolation function $f$:

$$\theta(p_{\overline{i,j}}) = f(x) \mid i < j \in \mathbb{N}, \quad p \in s, \quad s \in H_s \tag{3.8}$$

s.t.,

$$\forall p_i \in s, \quad f(i) = p \tag{3.9}$$

The following discussion is structured as follows: we describe the naturally occurring order structure induced by the temporal domain of the measurements. We then define a Push-Down Automata (PDA) and a set of symbols belonging to its associated grammar $G = \{N, \Sigma, P, S\}$, which describes the Finite State Machine (FSM) of the system.

First of all, let $t : X \longrightarrow \mathbb{R}$ be the time function, having $L, P, H \subset X$, we then have the natural order structure:

$$t(s_i) <= t(s_j), \tag{3.10}$$

a unique property given by the ability of a state to have multiple cursors $nC$ is given by the potential equality,

$$t(s_i) = t(s_j) \iff nC > 1. \tag{3.11}$$

### 3.1.3 The state machine correspondence

The system is modelled by a push-down automata (PDA). Let $M$ be a PDA:

$$M = \{Q, \Sigma, \Gamma, \delta, q0, H0, F\}, \tag{3.12}$$

with a set of states

$$Q = \{Init_i, Gen_i, Pred_i, Free_i \mid i \in \mathbb{N}\}. \tag{3.13}$$

These give the general classes of states but need not be complete in this description, because each individual state could be further decomposed and classified bringing little value to the discussion. Here, $Init_i$ is a class of initialization states when starting the program, $Gen_i$ is a set of states used for generating the tiling objective, generating a new prediction or set of predictions, $Free_i$ is a cluster of states where the main mode of interacting is hand-drawing.

The PDA, M, is capable of switching game modes by parsing the input and triggering switches using the stack(s). The complexity of the inputs has no theoretical upper bound, and it will be shown that gestures can be integrated into the UI to eliminate buttons, which supports the study's pursuit of compressed representations. A further "complexification" is to include an image retrieval engine that uses features of a gesture to change the interface. Due to time constraints we don't fully implement the game mechanics of all game modes such as that of UI simulation, but only detail on the currently existing hierarchical representation of UIs in *other sections*. Hypothetically, a $Sim_i$ class of states would be introduced to account for this game mode.

To model all possible events within the PDA, as well as signals, the alphabet for the input is the following:

$$\Sigma = \{Point_{\delta xy}, K_{\updownarrow}, K_{\leftrightarrow}\}. \tag{3.14}$$

The associated set of rules $\delta$ of $M$ are a set,

$$\delta : Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \longrightarrow Q \times \Gamma, \tag{3.15}$$

which makes use of the stack to implement switching logic as between game modes $q_i \in Q$, store the history of inputs $H$, and adjust the random variables of the probabilistic sampling $\chi_i$ such as when changing the number of predicted strokes and the frequency of the predicted strokes, or using gestures.

The final states $F$ are also determined by gestures, or the probability of successfully completing a current task based on previous input, which is a final state specific to the current game mode only $F' \subset F$.

Consider drawing the edges of a tiling as game mode (1) and clicking buttons on a UI the game mode (2). The events involved in both are $Point_{\delta xy}$. The rules $\delta$, depending on the state $q \in Q$, will either record a stroke when holding the mouse button $K_{LMBdn}$ or releasing it $K_{LMBup}$. Ideally the system would capture data about all previous movements of the cursor, but when applying this on real hardware, the system would quickly run out of memory, especially when the cursor moves in n dimensions, or equivalently $Point_{\delta x_{\overline{1,n}}}, \langle x_i \rangle \in \mathbb{R}^n$.

Storing all history is not necessary when we record only the intervals delimited by the $K_{LMB}$ inputs. The resulting state modifiers are recorded by the stack, and the inputs are inserted in the set H. Mathematically we denote a map between the grammar and the problem sets:

$H_s, H_e \in H$

$\phi : Gamma \longrightarrow H_s$

$\phi(x_k, x_{k+1+n}) = (p_k, ..., p_{k+1+n}, K_{\updownarrow}), p \in P_s, K \in \Sigma$ if $k_{+2+n} is K_{\updownarrow}$

A stroke will always begin with a vertex and terminate with a vertex. The dual problems of the tiling drawing and clicking on the UI elements is then as follows:

$\phi(x_k, x_{k+1}) = (p_{k-1}, K_{\uparrow}), p \in P_s, K \in \Sigma$ while still maintaining the ability to interpolate two points

$$\delta : H_s^2 \longrightarrow H_s$$

$$\delta(s_i, s_j) = \begin{cases} y - y_{s_i} - m * (x - x_{s_i}) = 0 \\ m = \frac{y_2 - y_1}{x_2 - x_1} \end{cases} \tag{3.16}$$

Consider $l \in L$ a possible prediction with $L \subset P_c \times P_c$. We would like to impose an order structure $\tau_1 : l_1, l_2, ..., l_n$ s.t. $\Psi : L \times L \longrightarrow \{T/F\}, \Psi(l_1, l_2) = m_{l_1} > m_{l_2}$, where

$$m_l = \frac{y_{p2} - y_{p1}}{x_{p2} - x_{p1}}, \quad p_{1,2} \in l$$

and want to look for alternatives of $\Psi$ where the comparison operator itself is a function $g : L \times L \longrightarrow \{T/F\}, \Psi(l_1, l_2) = g(l_1, l_2)$.

For the inputs $K_\uparrow$ and $K_\rightarrow$ the state machine transitions into a batch prediction mode where multiple predictions are made. A prediction is a function of the order structure together with 2 random variables that describe the number k of strokes to predict, and the batches in relation to the time in which they will be provided.

$$P(\chi) = \begin{cases} 1, & \iff q = q_{full}, q \in Q \\ \frac{\lambda^k * e^{-k}}{k!}, & \iff q = q_{timed}, q \in Q \end{cases} \tag{3.17}$$

where $q_{timed}$ has a chance to give a prediction batch in a set of time intervals, but will always roll one of those intervals.

Remark that the number of cursors $nC$, the order structure corresponding to their orientations $\tau_\upsilon$ and the map of their relative positions $\delta_\upsilon$ are all characteristics of the next prediction synthesized by the program. This information is calculated on the moment of providing this prediction as-needed, and not at the initialization phase. Therefore there is a map $\phi : Q \times \Gamma \longrightarrow P \times \tau$ where the tuple-set $\tau = \{(nC, \tau_\upsilon, \delta_\upsilon, l) \mid l \in L\}$ is the set of possible current predictions mapped by $\phi$. Alternatively one should choose a mapping co-domain of $\phi$ s.t. additional information that was left out is included. For example a coloring of each vertex by relative position could result in a more complex sampling of the current prediction: $\phi : Q \times \Gamma \longrightarrow P \times \tau \times \gamma$ where $\gamma : N \longrightarrow \mathbb{N}^3, \gamma(x) =< i, j, k >$ s.t. $i, j, k < 255$.

### 3.1.4 Sequences of predictions

Recall the mapping $\phi$ (3.18) and let this generate the set $S = \{(nC, ..., l), l \in L\}$. Take a sequence of predictions from this set $s_1, s_2, ..., s_n, s \in S$. We have already defined an order structure of the orientations $\tau_\upsilon$, for each individual stroke of this prediction, formally this means $(\tau_\upsilon, l) \subset s, s \in S$.

Further we can derive additional transformations through subsequent mappings. Let

$$\phi^1 = \{s_i \mid \overline{xy}_{j,s_{i-1}} = \overline{xy}_{i,s_i}, \forall i \in \mathbb{N}, x_{s_i} \in S \cap L\} \tag{3.18}$$

be the set resulting from mapping the set of all predictions by imposing the constraint that the endpoints of the lines be contiguous.

Theoretically there could be any number of such transformations. A single additional mapping that might be of interest is provided, $\phi^2$, in which the longest common subsequence (LCS) algorithm is used to introduce two additional tunable parameters: *k* and *r*, which are the chosen subsequence length and the number of repetitions.

$$\begin{aligned} \phi^2(k,r) &= f(g(S^1)) \\ &= f(s_i \mid s_i \in \phi^1(S)) \\ &= \{s_i \mid m_{l_{s_i}} = m_{l_{s_{i+r}}}\} \end{aligned} \tag{3.19}$$

having $\| S^3 \| = m$, where $m_i$ is the slope of the line.

Then, the set of constraints C have a set of mappings $\phi^i$, where C is:

$$C = \begin{cases} c_1, & \text{The endpoints are connected.} \\ c_2, & \text{They are formed by repetitions of the LCS.} \end{cases} \tag{3.20}$$

### 3.1.5 Complexity of the tilings

Let $Sp(X)$ be an operator acting on a set, named the spectrum of that set, and consider $L \supset P_c$. Then $Sp(L)$ is our measure of complexity, in the following way:

$$\begin{aligned} Sp(X) &= \langle m_i \rangle, \forall i \in \mathbb{N}, \\ &= \langle m_i' \mid \{-1, 1\} \rangle \end{aligned} \tag{3.21}$$

with $m_i' = |m_i|$ is the absolute value of the slope $m_i$. This goes to show that the number of reflection, rotation and transformation operations that produce the tilings can still generate finite entropy of information. The inner product of the linear space of slopes with the set $\{-1, 1\}$ produces the negatives of each slope, but this is only relevant when having to consider orientation, as we did in the mapping $\tau_v$.

In the case of periodic or regular tilings, $Sp(X)$ would be enough, but for aperiodic tilings like Penrose tilings, quasicrystals, and quasi-periodic crystallographic groups, the following is proposed:

$$\max_{k>=2, r>0} |\phi^2(k,r)|_{k,r} \tag{3.22}$$

which is to say find the maximum k and r of the LCS function. Since we can't make claims about which of k and r is more important, the reader might either consider them separately or come up with a function $g : \mathbb{N}^2 \longrightarrow \mathbb{R}$ to best suit their needs.

Note that one might also see a similarity between this measure and the span of a set $span(X)$, which might prove to be similar, but if one considers span of quasistructures, the previously defined $Sp$ would be modified to include the LCS complexity measure. An idea to further refine it is to determine the probability $\chi$ of finding a value close to the $|\phi^2|_{k,r}$ in less than $r'$ neighbors.

## 3.2 User interface and game window

*P1: tilings, algebras, representation, graph rewriting, XML, knowldege mining*

The data collection app is presented as a game where the task is to complete the current pattern. Patterns viable for the game are plane-filling tilings of shapes. The choice of space-filling tilings and curves is based on the pre-existing wealth of literature describing the representation of these geometric structures. The algebras underlying their representations can not only be used to identify and isolate stimuli but also to generalize user interfaces and prove similarities between transformations of the same object. Tilings can be described by finite groups, or finite reflection groups to be more specific. * If a system can be proven to have a sufficiently close graph representation, for example by using a specification language that can describe hierarchies of objects, then by using rewriting rules acting on the graph the resulting graphs might be objects that belong to the same group. Since the hypotheses of the framework's knowledge mining system are based on algebraic descriptions of the fundamental components of the system, we can look for patterns even in very complex systems by rewriting them to an equivalent but simpler one. For a detailed analysis refer to the Tiling (Chapter X, which includes discussion about diagrams)

Figure 3.1: Caption

The game loop has two modes to best capture both the tasks of kinematic data collection and generalized user interfaces: a mode where the task is to draw a set of tilings based on sequences of strokes, and a task that simulates a user interface and the transitions between the activities meanwhile capturing the cursor's position.

To help achieve the goal of drawing the tiling, the game engine recommends a series of next strokes, highlighted on the screen, based on complex logic. The player then draws over these recommendations.

Deviating from the recommendation is not a problem, but a valuable stream of information. The game should detect if the user's initial intention is one of the best possible recommendations in the first place. If it is not, the game can adapt the next recommendations to better fit the current user's inputs and adapt while transitioning to a good outcome. The latter should be considered a human-in-the-loop (HIL)

feedback system, because while the system adapts, the user is also learning and becoming habituated. In the case of the HIL system, the learned behavioral patterns should match one of the best possible combinations, as well as be interpretable by other users.

The player has the choice to reset the game or finish the current tiling. The game engine decides when the input is sufficiently close to the goal so that the task is considered complete and after which the player has the option of transitioning to the next task.

The game is allowed to choose the next task based on the previous tasks, but the detail of introspection of the previous tasks is limited to general statistical indicators. For example, the current task should not depend on fine-grained data of the previous task but should instead depend on the previous task choices and some indicators of the complexity of the previous task.

The main window, as seen in Fig. X, has no buttons. The way to navigate is by drawing a gesture, which is fed into the shape recognizer artificial neural network (ANN). ~~This component is detailed in chapter X, because it has other uses as well.~~ (detecting patterns when using a UI by inferring over a time window, ...)

## 3.3 Tilings

### 3.3.1 Background

The representation of tilings is a complex topic. Usually there is a diagrammatic language or a sequence of symbols to denote a tiling. We refer to the sequences as notations, which are further described.

Figure 3.2: Picture of the diagram notation

The fundamental building block is a polygon, or sometimes the vertices and edges of these polygons. They can be generalized to n dimensions but for this discussion we refer to the Euclidean plane. Regular tilings involve regular polygons as the building blocks. There are specific operations done iteratively to form a composite of the fundamental blocks. These can be rotation and reflection operations, for example.

Figure 3.3: Picture of some fundamental tilings

It is possible that some notations are too restrictive to produce every possible case. Cundy  Rollet's notation has some problems regarding the ability to generate unique tesselations*dagger** because of ambiguity. [**GomJau-Hogg**]

There are many classifications of the possible ways of tiling the Euclidean plane, and this has been a subject of scientific debate for decades, before 1987 when Grünbaum and Shephard published their work. [**GrunbaumShephard**]

To every tiling corresponds its dual representation, and possibly coloring together with other properties and particularities. Considering the dual graph where the centroid of each polygon is a vertex, and these nodes are connected. When the resulting tiling is identical to the original one, the object is called self-dual.

Figure 3.4: Picture of tilings together with their dual and tables of tilings

Closely related figures are ones produced by the plane crystallographic groups, plane symmetry groups, or wallpaper groups.

### 3.3.2 Mathematical background

A Coxeter group is an abstract group that admits a formal description in terms of reflections.

The group has the following presentation: $\langle r_1, r_2, ..., r_n \mid (r_i r_j)^{m_{ij}} = 1 \rangle$ where $m_{ii} = 1$ and $m_{ij} >= 2$ for $i \neq j$. A presentation is a way of specifying a group from a set of generators and a set of relations. The relations in this descriptions are of the form $(r_i r_j)^{m_{ij}}$.

Ultimately one obtains a correspondence between a Coxeter diagram and a Coxeter matrix. For the formal details refer to Coxeter 1935. [**coxeter1935**]

Figure 3.5: Picture of some Coxeter groups with their diagrams

The Schläfli symbol has a notation of the form $\{p, q, r, ...\}$ and can define regular polytopes and tesselations. For example, $\{3\}$ is an equilateral triangle, $\{4\}$ a square, and so on.

By using a fraction one can also describe pentagrams $\{\frac{5}{2}\}$, pentagons $\{\frac{5}{1}\}$, which are given by the number of vertices and their turning number. The notation has some useful properties, for example finding the dual of polytopes. If a polytope of dimension $n >= 2$ has Schläfli symbol $\{p_1, p_2, ..., p_{n-1}\}$ then its dual has the symbol $\{p_{n-1}, p_{n-2}, ..., p_1\}$.

Figure 3.6: Some Schläfli symbols with the generated result

### 3.3.3 Motivation

Although the framework does not generate the tilings by itself, but instead uses feature extractors together with an Support-Vector-Graphics (SVG) parser, the mathematical context is considered very important to the study. It might prove valuable to attempt to compress a hierarchical description of a UI to its generator. Usually the UI is described by a tree structure such as an XML, XAML, or equivalent representation. It might be worth investigating if using symmetry groups and compositions of functions that do similar transformations can produce similar results to classical specification methods. It could also be investigated if the tree hierarchy of an UI, or more generally the scene graph of any virtual environment, can be decomposed into pieces that can be described by such special groups, and the graph might be annotated with this information.

## 3.4 Database

Microsoft SQL Server 2019 is deployed on an Intel i7 machine with a 256GB SSD to collect metadata about the game as well as sensor data. The database was tested with fast time series data, of a bandwidth of at least approaching 64kbps. The database ran without performance bottlenecks for at least 180 seconds per game session. Indexing was attempted with respect to the timestamp column of the table, but there were some issues when doing this on 50000 records. A better DB alternative such as Redis, which is well suited for time-series data, was left unexplored.

The database is structured in tables that are mainly used for logging. A log of strokes exist at runtime that is replicated in the DB. Additional environment metadata such as game mode selections exists. The gestures are logged in a separate table. The strokes themselves constitute a time series, and every metadata is timestamped, but additional tables can be included for time series data. As previously mentioned, the *MS SQL Server 2019* is capable of handling a few minutes of synchronous time series queries including insertions. For a discussion on processing the time series tables review Chapter 4, as more details are presented in a comparison with image processing and video processing with machine learning techniques.

The ETL Pipelines consists of stored procedures that are triggered synchronously by game events and run asynchronously on *MS SQL Server 2019*'s engine. These are transformations applied to tables and could be thought of as functional programming designs. The source table is not modified but an output table is generated with the results, which are then updated on re-running the query, making the stored procedures pure functions. The output tables are manually specified into the configuration of the application and are queried by any of the other soft-

ware components. This level of decoupling introduced by SQL Server's modularity and asynchronous processing is worth the processing power even for a small application. Stored procedures include methods for querying past predictions with constraints, past user input, optionally sequence matching algorithms, and lastly generating human-readable statistics about the previous sessions.

## 3.5 Dependencies and structure

The game is written in Python 2.7 and Python 3.9+ which is separated into at least 5 classes of packages/modules. For more explanation on the reasoning behind this choice review the "Software alternatives" section. The whiteboard depends on a few external python packages, available from the PyPi registry: pygame, pyodbc, numpy.

A configuration file can be used to define some global parameters about the game logic, the UI, logging, and game modes. Here, the random variables $\chi_i$ specified in Section 3.2, 3.17, are defined as a list of possible values that can be sampled. The game may of course choose to alter the joint probability distribution which affects $P(\chi_i)$, but the values that can be sampled still respect the configuration. For example, when recommending a batch of length *k* of edges, between each render on the screen the program may wait 100, 200 or 300 milliseconds (ms). This means that for stroke number $k_n$, it will take a minimum of $t = n * 100$ ms, and a maximum of $t = n * 300$ ms. The more often the sampling $P(\chi_i)$ occurs, the higher the probability of having a non-minimal time is, within *n* rolls.

The main class is the *Program* class, which captures events from the system using *pygame* utilities, and passes them down to the member classes. Among the member classes there are the following: Whiteboard, Tiling, Paintbrush. They respectively use lower level concepts such as custom classes including *Stroke*, *Line*, *Path*, and *pygame* classes including *Rect*, *PixelArray*, *Surface*.

Note that these game objects are fundamental enough for even a simple feature extractor to be used to make a relation between other game environments and user interfaces to this framework's abstract GUI formulation. Convolution layers in convolutional neural networks act as feature extractors themselves. Deep layers of a neural network especially encoder-decoder architectures have high level features connected to the labels of the data in order to synthesize new previously unseen data. If acting on a set of simple features like lines, curves, polynomials, the networks weights might be more easy to interpret when using tools for visualizing them.

The framework also contains a *shape recognizer* neural network model, also written in Python, that can use an interval of strokes $s_i \in S$ to recognize specific shapes

which are in turn detected by the game by periodically querying the DB for events or receiving an event notification through a web socket.

Another decoupled software module is the *point cloud extractor* which provides a tiling to the game. The point cloud extractor is written in Python, and parses an SVG file to produce a set of edges formed by points in the tiling. The SVG contains absolute and relative coordinates of the endpoints of edges of vectors. The tilings used are preiodic or quasi-periodic regular tilings, made of fundamental regular polygons, like triangles, pentagons, squares, etc. Even though SVG supports rendering of curves, there is no tiling containing curves used by the framework. The reason is that for quasi-periodic tilings, curves might be given by a mapping of walks on the edges using a weighted average. A similar case was discussed in 3.2 when defining the LCS algorithm. Consider also the interpolation function $\theta(p_{\overline{i,j}}$. Now the average $\phi^3(s_i) = \frac{\overline{xy}_{s_i1}+\overline{xy}_{s_i2}}{2}$ is given to obtain $\theta(\phi^3)$, the interpolated averages, which are curves. When applying this to quasi-periodic tilings, one has to start from a point $p$ and see what is $|spec(\phi^2)|_{r'}$ where $d(p, r') = \sqrt{\sum (p_{\overline{xy}} - r'_{\overline{xy}})^2}$. The curves in the spectrum set is another case of the exemplified $Sp(X)$, which could be an indicator of the complexity of a tiling or of a stroke history which has an intersection with a tiling. This will allow for the *feature extractor* of the gestures to be connected with the complexity analyses, which won't be detailed in this study but can be expanded upon. One can imagine that UI transitions can be changed using the parametrized components to obtain a set of curves that is as varied as possible, in order to not accidentally perform the wrong sequence of clicks and fail to reach the desired transition.

## 3.6 Motivation for software choices and alternatives

The software was tested with external hardware such as a Wacom pen tablet on Windows 10. This is not a requirement but was in interesting case study because Windows provides integration using Windows Ink, which is a service that connects the operating system to the driver. This allows pressure to be forwarded to an application using the SDK and various gestures to be performed when using the OS itself. The most valuable data were described to be the changes in complexity associated with the transitions, and possibly kinematic data. Because the sampled points are timestamped, one can measure the velocity of the drawing strokes, but can also omit them when not relevant. A pen only changes the kinematics of the user, and not the drawing necessarily. If whole body tracking was included then more complex gestures could be performed, but it remains to be determined if the data would be relevant to the Euclidean embedding of regular tilings. It might be possible to investigate whether hyperbolic tilings work better in 3-dimensional space,

or if whole-body kinematic can correspond to higher-dimensional tilings. Since the higher-dimensional tilings have a 2-dimensional analog the results could still be correlated with the findings of this study, as some 3-dimensional space filling tilings are obtained by mere projections, rotations and reflections of their two-dimensional building blocks.

To keep development integrated in the Windows ecosystem, it would have been preferrable to use C, with .NET's legacy classes for drawing. The *System.Drawing* namespace can be used with legacy windows forms apps. Newer WPF apps, in which the UI is specified by XAML, have the *InkCanvas* control. It was not possible to choose this for the framework because there is no control over each individually sampled point over which the cursor hovers. The .NET WPF app would also interpolate the points with their own methods, instead of the chosen Bresenham's line algorithm. There is no also way to control the sampling speed and therefore the number of points chosen for interpolation. The framework's Python 2.7 solution which allows for pixel-level manipulation was chosen despite losing a signal modality, that of the pen's pressure, which was easier to integrate in the Windows ecosystem.

An alternative for the drawing board that was considered is also a JavaScript native environment. D3.js allows for manipulation of an SVG element to produce animations shapes and drawings. The problem was that accessing the logic behind the sampling and interpolation was again difficult. The SVG element is a HTML native control, and the JavaScript low level routines for pixel-by-pixel access were not visible. Also, when using React together with the previous, there would be a conflict of access over the base SVG element that corresponded to the canvas. Still, SVG would have had some nice properties. First of all, it is based on XML, and there is a 1-to-1 correspondence between the HTML SVG tag and an SVG image. A minus is the complexity of the internals and high number of particularities to various operating systems and browsers. An hierarchical representation such as XML and SVG can still be one of the best ways, because they can be modelled as nodes of a tree or graph. The nodes can be annotated with custom attributes and also geometrically embedded and reorganized. Many classical problems in Graph theory are concerned with these aspects and are closely related to the mathematical discussion of the correspondence between the tilings and the notion of complexity with respect to their representations.

# Chapter 4

# Human-computer interaction

TBD

## 4.1 Emotion processing in the brain

To date there is no clear localization of emotion in the brain. Paul Ekman and Klaus Scherer provide an elaborate discussion of this topic in their works. Valence and arousal are commonly used to refer to the subjective appearance of emotions and feelings. The theory of appraisal is an important driver of the processing of an emotion episode. [14, 15]

Russel's valence arousal model is another very powerful model in conjunction with EEG measurements. Maruyama et al. have performed an independent component analysis with a source localization method, to obtain distinct neural activities from EEG signals. They propose that neural activity can correlate with specific emotional states, depending on the frequency and location of the signal. [16]

## 4.2 Cortical homunculi, receptive fields during drawing

Good communication between brain regions involved in specific tasks is important to consider when recording neural activity. Thompson et. al have shown using diffusion tractography that impaired communication between the motor and somatosensory homunculus is associated with poor manual dexterity. [17]

Kropf et al. have conducted a review about the involvement of the somatosensory cortex in processing emotion and empathy. The primary somatosensory cortex (SI) and the secondary somatosensory cortex (SII) are both found to be involved in tactile attention. [18]

Studies from immersive environments such as virtual reality (VR) [19] have suggested that the cortical homunculus is capable of adapting to novel bodies. Homuncular flexibility (HF) is a paradigm in which physical motions are transformed by remapping degrees of freedom from tracked movements onto an avatar. VR interventions for phantom limb pain (PLP) has been shown to give significant reduction in symptoms. [20] The mechanism by which altering the visual input of a patient suffering from PLP reduces pain has been proposed to be a cortical reorganization, enabled by neuroplasticity.

## 4.3 Data and software

MNE is a Python framework for working with EEG data. [21]. The neural networks are trained using Tensorflow. Data availability is limited by the specificity of the problem we are trying to solve, and also EEG data availability. We employ an already existing EEG dataset on which we train our classifiers. We then augument this data with our own using a generative model that predicts EEG signal features using only tablet data, for which we use our own EEG measurements taken during the drawing tasks.

The drawing data is a fusion of the CoSE, extended with 2 additional dimensions representing pen angle and pen pressure. Using the data as a series of strokes has many advantages and allows us to specify a strict time window for working on the EEG data as well as using RNN models.

The pen strokes are used to generate a set of most likely EEG patterns to follow, in a multi stage learning pipeline, which detects the likelihood and frequency of spikes or ERPs, predicts a likelihood of abnormalities, estimates fatigue and cognitive performance, micro expressions or micro affective states. Depending on the type of end to end system that is expected, the architecture of the multi stage model could be normalized, or split into smaller independent models.

## 4.4 Predicting signals: TimeGAN

The architecture generates the most likely EEG signal given a pen stroke, which is represented by the compositional model. We use a derrived model of TimeGAN, a generative model that can learn high dimensional relationships of the signal dynamics to generate realistic signals. [22] This architecture has been adapted in the past for privacy-preserving medical data generation, as well as EEG signal generation. [23, 24]

The network is fine-tuned to make better use of the latent space, or embedding

space, by using features of the tablet recordings as input. Additionally, the features of the tablet are extended with a set of parameters that can control the visual cueues delivered to the user. The user does not have any control over this set of parameters, however the simulated EEG signal is conditionally dependent on them. The visual signals are adjusted for maximizing the likelihood of an improved generated EEG pattern, as seen in Fig. **??**.

Additionally we extend the model to have the possibility to use other recordings that may also be taken as signals, for example ECG.

Images of spectrograms

# Chapter 5

# Discussion

### 5.0.1 Feature extractors

Images of feature representations: Hough

# Chapter 6

# Conclusions

# Bibliography


[1] Matlab / Mathworks website. "Sensor Fusion and Tracking Toolbox". URL: https://www.mathworks.com/products/sensor-fusion-and-tracking.html.

[2] Eva Ose Askvik, F. R. (Ruud) van der Weel, and Audrey L. H. van der Meer. "The Importance of Cursive Handwriting Over Typewriting for Learning in the Classroom: A High-Density EEG Study of 12-Year-Old Children and Young Adults". In: *Frontiers in Psychology* 11 (2020), p. 1810. ISSN: 1664-1078. DOI: 10.3389/fpsyg.2020.01810. URL: https://www.frontiersin.org/article/10.3389/fpsyg.2020.01810.

[3] Boring Matthew J. Robinson Amanda K. Venkatesh Praveen. "Very high density EEG elucidates spatiotemporal aspects of early visual processing". In: *Scientific Reports* 7 (2017). ISSN: 2045-2322. DOI: 10.1038/s41598-017-16377-3. URL: https://doi.org/10.1038/s41598-017-16377-3.

[4] Emre Aksan et al. *CoSE: Compositional Stroke Embeddings*. 2020. arXiv: 2006.09930 [cs.LG].

[5] Roberto D. Pascual-Marqui. "Standardized low-resolution brain electromagnetic tomography (sLORETA): technical details." In: *Methods and findings in experimental and clinical pharmacology* 24 Suppl D (2002), pp. 5–12.

[6] Gopala Krishna Anumanchipalli, Josh Chartier, and Edward F. Chang. "Speech synthesis from neural decoding of spoken sentences". In: *Nature* 568 (2019), pp. 493–498.

[7] Haidar Khan et al. "Focal onset seizure prediction using convolutional networks". In: *IEEE Transactions on Biomedical Engineering* 65.9 (2017), pp. 2109–2118.

[8] Yuki Hagiwara U. Rajendra Acharya Shu Lih Oh et al. "Automated EEG-based screening of depression using deep convolutional neural network". In: *Computer methods and programs in biomedicine* 161 (2018), pp. 103–113.

[9] Nadia Mammone, Cosimo Ieracitano, and Francesco Carlo Morabito. "A deep CNN approach to decode motor preparation of upper limbs from time-frequency maps of EEG signals at source level". In: *Neural networks : the official journal of the International Neural Network Society* 124 (2020), pp. 357–372.

[10] Francesco Rundo et al. "An Innovative Deep Learning Algorithm for Drowsiness Detection from EEG Signal". In: *Computation* 7.1 (2019). ISSN: 2079-3197. DOI: 10.3390/computation7010013. URL: https://www.mdpi.com/2079-3197/7/1/13.

[11] Hirokazu Takahashi et al. "Convolutional neural network with autoencoder-assisted multiclass labelling for seizure detection based on scalp electroencephalography". In: *Computers in Biology and Medicine* 125 (2020), p. 104016. ISSN: 0010-4825. DOI: https://doi.org/10.1016/j.compbiomed.2020.104016. URL: https://www.sciencedirect.com/science/article/pii/S0010482520303474.

[12] Garrett Honke et al. "Representation learning for improved interpretability and classification accuracy of clinical factors from {EEG}". In: *International Conference on Learning Representations*. 2021. URL: https://openreview.net/forum?id=TVjLza1t4hI.

[13] Andac Demir et al. "EEG-GNN: Graph Neural Networks for Classification of Electroencephalogram (EEG) Signals". In: *CoRR* abs/2106.09135 (2021). arXiv: 2106.09135. URL: https://arxiv.org/abs/2106.09135.

[14] Klaus R. Scherer. "What are emotions? And how can they be measured?" In: *Social Science Information* 44.4 (2005), pp. 695–729. DOI: 10.1177/0539018405058216. eprint: https://doi.org/10.1177/0539018405058216. URL: https://doi.org/10.1177/0539018405058216.

[15] Paul Ekman. "An argument for basic emotions". In: *Cognition and Emotion* 6.3-4 (1992), pp. 169–200. DOI: 10.1080/02699939208411068. eprint: https://doi.org/10.1080/02699939208411068. URL: https://doi.org/10.1080/02699939208411068.

[16] Yasuhisa Maruyama et al. "Independent Components of EEG Activity Correlating with Emotional State". In: *Brain Sciences* 10 (Sept. 2020), p. 669. DOI: 10.3390/brainsci10100669.

[17] Abigail Thompson et al. "Impaired Communication Between the Motor and Somatosensory Homunculus Is Associated With Poor Manual Dexterity in Autism Spectrum Disorder". In: *Biological Psychiatry* 81 (July 2016). DOI: 10.1016/j.biopsych.2016.06.020.

[18] Erika Kropf et al. "From anatomy to function: the role of the somatosensory cortex in emotional regulation". In: *Revista Brasileira de Psiquiatria* 41 (2019), pp. 261–269.

[19] Andrea Stevenson Won et al. "Homuncular Flexibility in Virtual Reality". In: *Journal of Computer-Mediated Communication* 20.3 (Jan. 2015), pp. 241–259. ISSN: 1083-6101. DOI: 10.1111/jcc4.12107. eprint: https://academic.oup.com/jcmc/article-pdf/20/3/241/19492300/jjcmcom0241.pdf. URL: https://doi.org/10.1111/jcc4.12107.

[20] Thomas Rutledge et al. "A Virtual Reality Intervention for the Treatment of Phantom Limb Pain: Development and Feasibility Results". In: *Pain Medicine* 20.10 (June 2019), pp. 2051–2059. ISSN: 1526-2375. DOI: 10.1093/pm/pnz121. eprint: https://academic.oup.com/painmedicine/article-pdf/20/10/2051/30101742/pnz121.pdf. URL: https://doi.org/10.1093/pm/pnz121.

[21] Alexandre Gramfort et al. "MEG and EEG Data Analysis with MNE-Python". In: *Frontiers in Neuroscience* 7.267 (2013), pp. 1–13. DOI: 10.3389/fnins.2013.00267.

[22] Jinsung Yoon, Daniel Jarrett, and Mihaela van der Schaar. "Time-series Generative Adversarial Networks". In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach et al. Vol. 32. Curran Associates, Inc., 2019. URL: https://proceedings.neurips.cc/paper/2019/file/c9efe5f26cd17ba6216bPaper.pdf.

[23] Cristóbal Esteban, Stephanie L. Hyland, and Gunnar Rätsch. *Real-valued (Medical) Time Series Generation with Recurrent Conditional GANs*. 2017. arXiv: 1706.02633 [stat.ML].

[24] Kay Gregor Hartmann, Robin Tibor Schirrmeister, and Tonio Ball. *EEG-GAN: Generative adversarial networks for electroencephalograhic (EEG) brain signals*. 2018. arXiv: 1806.01875 [eess.SP].