Artificial Intelligence Research Group - Robotics

Enabling the humanoid robot Myon to learn reoccurring movements of arbitrary length online

Manuel Gellfart
Matr. 4594870

A thesis submitted in partial fulfillment for the
degree of Master of Computer Science

Supervisor: Prof. Dr. Raúl Rojas
Assisting Supervisor: Prof. Dr. Manfred Hild

Institute of Computer Science, Freie Universität Berlin, Germany
June 9th, 2016

Eidesstaatliche Erklärung

Ich versichere hiermit an Eides Statt, dass diese Arbeit von niemand anderem als meiner Person verfasst worden ist. Alle verwendeten Hilfsmittel wie Berichte, Bücher, Internetseiten oder ähnliches sind im Literaturverzeichnis angegeben. Zitate aus fremden Arbeiten sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungskommission vorgelegt und auch nicht veröffentlicht.

Berlin, June 9th, 2016 _____

(Manuel Gellfart)

Abstract

Motif discovery describes finding reoccurring sequences in time series. It has been used frequently in different domains such as bioinformatics, medicine, and robotics. A lot of different work exists towards approaches for motif detection. This thesis is based on previous work, transforming time series to discrete representations by applying the *Symbolic Aggregate approXimation* (SAX). Afterwards, a hierarchical grammar structure is build by the greedy string compression algorithm *Sequitur*. The resulting grammar structure aids motif detection in linear execution time with linear space requirements. This thesis presents the approach and adapts it to find reoccurring movements in the motion data of the humanoid robot *Myon*. After implementing and evaluating this approach, adaptions for real-time execution are developed. Finally, motion detection is implemented directly on the robot and two techniques to learn motifs are introduced, enabling users to teach, label, and discard movement motifs.

Contents

Chapter 1

Introduction

In this chapter we will first introduce the project in which context this thesis takes place. Then the problem tackled by this thesis is presented.

1.1 Robot Myon

The background of this thesis is the *Myon* project. Myon is a humanoid robot which has been designed, built and programmed by the *Neurorobotics Research Laboratory* (NRL) of the *Beuth Hochschule für Technik* in Berlin. Myon has the unique property of being decentralized. Every body part of the robot is autonomous and can be run on its own [1]. This offers the research method of first implementing a new behavior on a small subset of limbs before extending it to the whole robot.

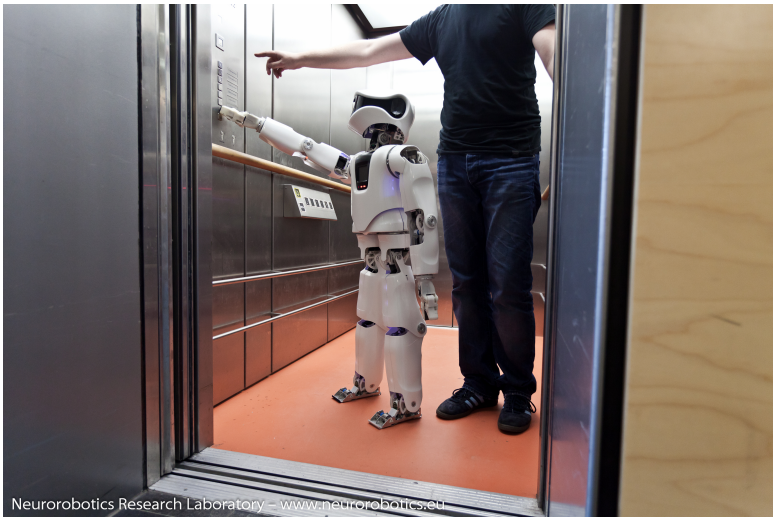

FIGURE 1.1: The robot Myon.

The purpose of constructing a humanoid robot is to rebuild human design, behavior and awareness. Therefore the robot is the size of an 8 years old child and has joints similar to human limbs. To communicate with its environment, the robot has microphones as ears, and a speaker where the mouth would be located. The one eye hides a camera which is used to detect optical flows, colors and faces. One goal of Myon is to interact with its environment and learn concepts from it like a child. The robot not only listens to noises and watches its surroundings, but people interact with it by moving and manipulating it.
The motivation of this thesis within the project is to enable Myon to learn, remember and recognize reoccurring movements. Achieving this would lead to new ways for the robot to interact with its environment. It could react to known actions as they occur and connect these with acoustic and visual perceptions.

1.2 Problem

Detecting and learning motifs in time series is an important problem in data-mining.[1] Motif discovery can be applied in many scientific domains such as bioinformatics, medicine and robotics. For this work it is used to learn and recognize robot movements which can have arbitrary length and are not known a priori. The implemented algorithms should be suited for online execution directly on the robot.[2]
Achieving this goal involves solving the following challenge: Detecting reoccurring motifs in time series. The motif detection algorithm needs to discover unknown motifs of arbitrary length. These motifs need to be stored efficiently. The algorithms should be able to operate without supervision in order to assert autonomous behavior. All algorithms and models require to be efficient in time and space due to the robot's limitations and in order to be executed online. After evaluating and adapting the used techniques a feedback loop will be developed as a proof of concept offering means to teach motifs to Myon and enabling the user to relabel or discard these motifs.
The main algorithms and methods used to find and store reoccurring motifs are not novel and will be presented in chapter 2. Although these techniques have been used on a wide range of domains, there exists only little work using the algorithms to detect motifs in motion data [2]. Furthermore, the previously conducted work only focuses on offline motif detection to find reoccurring patterns in previously captured time series.

[1] Motifs are reoccurring subsequences of a time series.
[2] Online execution processes each sequence in real time utilizing the input stream's history.

Chapter 2

Background

In the past 20 years a lot of research has been conducted on motif detection in time series. During this time period, a lot of different approaches have been developed or further improved. This chapter gives an overview of some different approaches and discusses their viability for our specific problem.
A lot of work exists on the detection of known patterns in time series [3–5]. These approaches are not viable for our use case, because we do not have prior knowledge about the motifs we want to find. We will therefore not consider any of these methods in this chapter.

2.1 Definitions

Before we dive into techniques and methods for motif detection, we will cover some basic definitions and terminology in this section. This is necessary to be able to fully describe and understand the underlying problems and the approaches solving them. The following definitions are widely used in literature covering motif detection in time series [6–11].

Definition 1. *Time Series*: A time series $T = t_1, \cdots, t_n$ is an ordered data set of real valued data points.

The exact size of a time series is not important as most algorithms work on subsections of time series called subsequences. In our case, the algorithm is supposed to run online, operating on an infinite stream of time series data.

Definition 2. *Subsequence*: A subsequence $S = t_j, \cdots, t_m$ is a fixed size subset containing contiguous data from a time series T.

All approaches covered in previous work or this thesis operate on subsequences of time series. These subsequences can influence the performance and characteristics like the minimum motif size depending on the used methods for motif detection. Without subsequences, it would be impossible to perform motif detection in time series online because every subsequence is processed independently from future subsequences. Most algorithms work with time series by shifting subsequences using a sliding window.

Definition 3. *Sliding Window*: A sliding window w defines the interval which is used to scroll over a time series $T = t_1, \cdots, t_n$. For every pair of succeeding subsequences $S = t_i, \cdots, t_{i+m}$ and $S' = t_j, \cdots, t_{j+m}$ the sliding window w is the distance $j - i$.

Usually sliding windows have a size $w \leq m$, where $m = |S|$ is the size of the subsequences. In consequence, $m - w$ data points of two contiguous subsequences will overlap. While smaller window sizes slow down the processing, several motif discovery techniques require normalization of data and most methods discover motifs by somehow comparing subsequences with each other. Thus, overlapping subsequences will result in more discovered motifs. Finding a good trade-off is important because some algorithms will yield trivial matches when the sliding window is chosen too small.

Definition 4. *Match*: Two subsequences $S, S', S \neq S'$ are considered a match. If $d(S, S') \leq R$, with distance function d and range R.

Matches are simply subsequences which are similar to each other within a range R, respective to some distance function d. Some motif discovery algorithms do not use distance measures and thus have different definitions for matches [2, 9, 11]. Although the above definition is intuitive, the problem of trivial matches persists.

Definition 5. *Trivial Match*: A trivial match is a match $S = t_i, \cdots, t_{i+m}, S' = t_j, \cdots, t_{j+m}$ with $|i - j| < q$ for some minimum distance q.

Trivial matches are matches which are too close to each other. For instance, shifting a subsequence by merely one position will, generally, create a match. We are never interested in trivial matches and thus there are always mechanisms to avoid these such as a minimum gap between two occurrences between matches[10].

Definition 6. *Motif*: Given a time series T with length n and subsequence length m A motif M is a set of similar sequences in T, where the length of each sequence in M is bound by minimum length m and maximum length n.

For distance based algorithms a motif is a set of subsequences matching each other. Other algorithms define similar sequences differently. Their motifs can even consist of

sequences with different lengths.

BruteForce(T) $\rightarrow$ (L_1, L_2)
Data: T: Time Series
Result: L_1, L_2: Locations for the best motif.

```
1 best-match ← ∞;
2 for i = 1, ··· , m do
3 |   for j = i + 1, ··· , m do
4 |   |   if d(T_i, T_j) < best-match then
5 |   |   |   best-match ← d(T_i, T_j);
6 |   |   |   L_1 ← i, L_2 ← j;
7 |   |   end
8 |   end
9 end
```

Algorithm 1: Brute-Force algorithm to find the best motif in a time-series.

Some work covers improvements to the runtime of the brute-force algorithm [6, 10]. This algorithm stores the distance of the current best match. It then iterates over each possible pair of subsequences, comparing their distance with the best so far. When a better motif has been found, the best match - as well as the locations for the best motif - are updated and returned in the end. The resulting runtime is $\frac{m(m-1)}{2} \in \mathcal{O}(m^2)$.

2.2 Previously Conducted Work

2.2.1 PROJECTION Algorithm

The *PROJECTION* algorithm was introduced by Buhler and Tompa [12] to address the Planted (l, d)-Motif Problem written by Pevzner and Sze [13]:

Definition 7. *Planted* (l, d)*-Motif Problem*: Given a sample of t sequences of length n, find an unknown motif M of length l. This motif is planted once in each sequence, where each occurrence differs from M in d random positions.

The *PROJECTION* algorithm follows a probabilistic approach: It iterates over each of the t sequences. In each iteration the next subsequences of length l are processed and hashed using *random projections* by choosing k of the l positions at random. Then each subsequence x of length l is hashed to $h(x)$, where $h(x)$ simply projects x to the concatenated k selected positions of x. In other words we ignore the $l-k$ positions which were not chosen. The more subsequences hash to the same bucket, the more likely it is, that they belong to the planted motif M.

2.2.1.1 Adaptations to Time Series

First the time series has to be discretized into a DNA-like string structure to apply the *PROJECTION* algorithm to them. A paper of Jessica Lin from 2002 accomplishes this by applying Symbolic Aggregate approXimation (SAX) to the time series to transform the data into a discrete symbolic representation [11]. Details on the SAX transformation will follow in subsection 3.2.1. In 2003, Chio improved this approach to discover motifs more efficiently by using the transformed time series as input for the *PROJECTION* algorithm [7]. In each iteration, each subsequence string of length l is hashed into a collision matrix by randomly selecting a fixed number k of their positions.

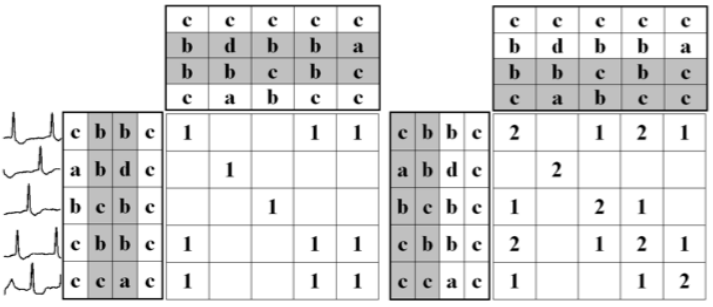

FIGURE 2.1: *Left:* Collision matrix after the first iteration of the *PROJECTION* algorithm. Positions one and four where selected for the hash function and the resulting cells increased by one. *Right:* The resulting matrix after the second iteration (Positions three and four selected) [2]

In Figure 2.1, two iterations are illustrated where each subsequence string has length $l = 4$. In each iteration, $k = 2$ positions of each sequence are selected for the hash function and the matching cells are increased. Candidates for motifs are discovered by choosing the highest valued cells in the resulting matrix. Experiments showed that this method offered an execution time linear in the time series length while being robust to noisy data. This approach has been further improved and adjusted [2], but the basic ideas remained the same.

2.2.2 Grammar Based Motif Discovery

A newer approach is to transform time series into a context-free grammar representation to utilize its hierarchical structure for motif discovery. The idea is that, in context-free grammars, each non terminal symbol represents a repeated pattern. Thus, motifs in time series can efficiently be inferred from their grammar. In 2014, *Jessica Lin et al.* published an algorithm that finds variable-length motifs in time series using grammar induction [9]. She uses the discrete symbolized representation of time series [11] as input for the *Sequitur* algorithm. *Sequitur* is a greedy text-compression algorithm which computes a

context-free grammar from a sequence of discrete symbols [14]. More information on *Sequitur* follows in subsection 3.2.2. After transforming the time series into its grammar representation, Lin's work offers a visualization tool to navigate through all rules in the grammar, highlighting corresponding sequences in the time series.

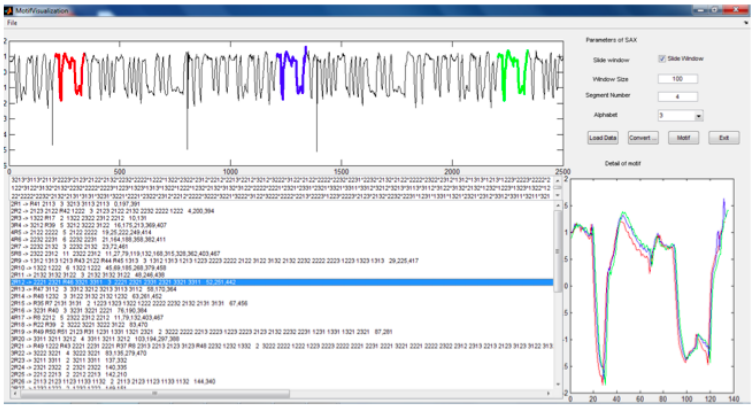

FIGURE 2.2: Screenshot of the Grammar/Motif Vizualitation Tool. *From Upper Left to Lower Right:* It shows a plot of the time series with highlighted rule sequences, the current parametrization of the algorithm, the computed grammar with selectable rules, and a plot of all occurrences of the currently selected rule. [9]

In the inducted grammar, not all of the rules derive to important sequences in the time series. Therefore, the paper proposes that a *rule interest* is necessary to rank the rules. This rule ranking is deferred to future work and simply rule length and number of occurrences are used as indicators for a rule's interest.
In her work, Lin shows that a simple greedy grammar induction algorithm yields good results for motif detection while running in linear time.

2.2.3 Anomaly Detection

A strongly related discipline to finding motifs is to discover time series discords or anomalies. Instead of searching for similar sequences, these algorithms reveal sequences in a time series, differing the most from other sequences. Even though this tackles a different problem, the used techniques are the same. Thus, the trend in anomaly detection is the same as in motif discovery. The most work on this topic uses SAX to transform the time series to discrete data. Different algorithms are then applied to the data:
The first work mentioning anomaly detection defines a recursive algorithm using the compression-based dissimilarity measure between two sequences x and y [15]:

$$CDM(x,y) = \frac{C(xy)}{C(x) + C(y)} \tag{2.1}$$

In the above equation $C(x)$ is defined as the size of the compression of x, using a given compression algorithm. So $CDM(x, y)$ is close to 1 for two unrelated sequences x and y, and close to zero for strongly related x and y. This measure is used in a recursive algorithm which gets the discrete symbolized time series as input. In each recursive step, it divides the input sequence x into two subsequences y and z and uses them to compute $CDM(y, x)$ and $CDM(z, x)$. The algorithm continues with the subsequence resulting in higher dissimilarity as input because it is most likely to contain the best anomaly.

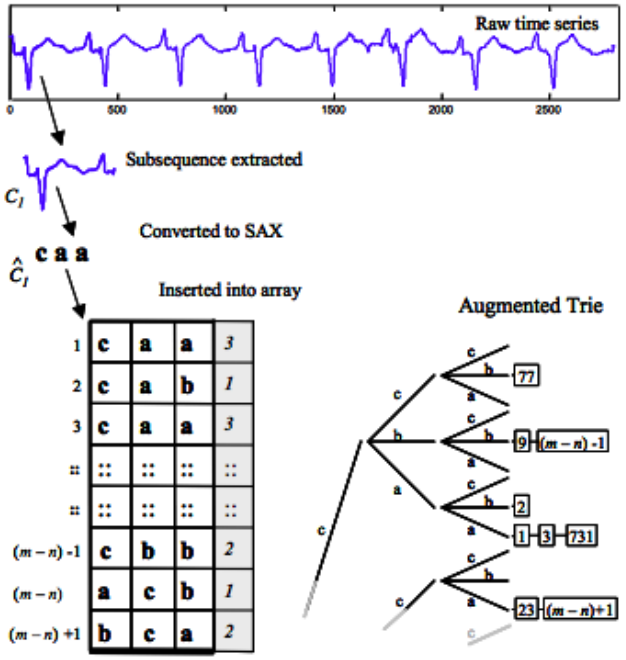

FIGURE 2.3: Visualization of the processing of a time series to approximate the perfect order of elements. This shows how every SAX word is stored in an array together with their occurrence count while a trie is build to store each position. [6]

Another work by *Lin et al.* [6] improves the trivial brute force algorithm for finding time series discords. This algorithm is similar to algorithm 1 but it searches for the pair of subsequences with the biggest distance to each other. This is done by the following two observations: First of all, the inner loop of the brute force algorithm can be abandoned as soon as one pair has a smaller distance than the current best discord. Secondly, if the data is arranged perfectly, the runtime will be linear as the first execution of the inner loop will find the best discord and all other executions will be suspended after the first step. This perfect order is then approximated by applying SAX to each subsequence and counting the occurrences of each SAX word while storing their positions in the original time series. This follows the idea that SAX words which occur frequently are very unlikely to contain the best time series discord. That is why these are visited last by the outer loop. For the inner loop, the stored occurrences are processed first and then all other sequences are processed in random order. This is done because sequences

sharing the same SAX word are more likely to have a short distance and therefore more likely to cause the inner loop to break.

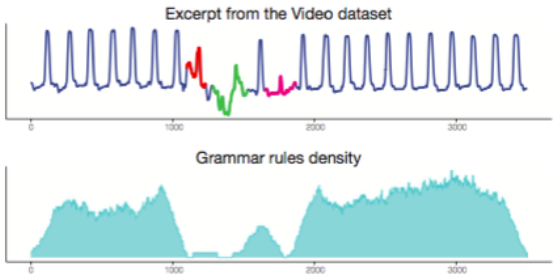

FIGURE 2.4: The upper plot shows a time series with its best anomalies colored. The curve illustrated below is the corresponding rule density curve. [8]

The most recent paper addressing anomaly detection uses a grammar based approach [8]. It also uses SAX and Sequitur to acquire a hierarchical grammar from time series data. They define a *rule density curve* which assigns the number of matching rules to each point of a time series. Then anomalies are extracted by selecting sequences with low rule density. A small example for this approach is shown in Figure 2.4.

Chapter 3

Approach

This chapter first gives an overview of the contributions which are made in this following work. It will then enlighten the chosen approach to solve the problem of finding reoccurring movements of arbitrary length in a robot's movement.

3.1 Contribution

The following list contains the contributions made in the succeeding chapters. These contain combine existing methods with novel ideas for the application of motif discovery. They are arranged in order of apparition in this work.

1. Algorithms for Offline Motif Discovery
    - Symbolic Aggregate ApproXimation
    - Sequitur
2. Adaptions to Robotics
3. Efficient Offline Implementation
4. Evaluation and Experiments
    - Qualitative Evaluation of Discovered Motifs
    - Verification of Performance
5. Online Implementation
    - Implementing Motif Discovery on Robot Myon
    - Methods to Teach Labeled Motifs to Myon
6. Summarizing Ideas for Future Work

3.2 Basic Techniques

The approach is based on Lin's work using SAX and Sequitur to store time series in a hierarchical grammar structure.

3.2.1 Symbolic Aggregate ApproXimation

In this section, we use an algorithm to transform our input into discrete data. The Symbolic Aggregate ApproXimation (SAX) transforms a subsequence into its symbolic representation [16]. By doing so, SAX achieves discretization and dimensionality reduction. Additionally, the symbolic representation offers the possibility to apply text-mining, compression, and many bioinformatics algorithms to the resulting data.
Before applying SAX to any subsequence, its data has to be normalized in order to have mean zero and standard derivation one. The reason behind this will be adressed in subsubsection 3.2.1.2. The SAX algorithm is given a subsequence $S = t_1, \cdots, t_n$ with $t_i \in \mathbb{R}$ and two parameters: the alphabet Σ with size $a = |\Sigma|$ and the word size w. The resulting output $S^* = t_1^*, \cdots t_w^*$ has length w with $t_i^* \in \Sigma$.
SAX consists of two steps: First, the time series is transformed using Piecewise Aggregate Approximation. Secondly, the PAA representation is symbolized into a discrete string.

3.2.1.1 Piecewise Aggregate Approximation

This step achieves dimensionality reduction by transforming any subsequence S of length n into its Piecewise Aggregate Approximation $S' = t'_1, \cdots, t'_w$ consisting of w segments. Each segment is computed by this formula:

$$t'_i = \frac{w}{n} \sum_{j=(i-1)\cdot\frac{n}{w}+1}^{i\cdot\frac{n}{w}} t_j$$

The resulting S' is an approximation of S with reduced dimensionality $w \leq n$. Each segment t'_i is approximated by computing the mean value of its corresponding $\frac{n}{w}$ values of the original sequence S.

Figure 3.1 shows an example sequence with its PAA representation. The thin line is the normalized time series S and each thick horizontal line is a segement of S'. Here the Piecewise Aggregation Approximation reduces the sequence's 125 dimensions to only 25 dimensions. This can be achieved in a single iteration over S with linear runtime

$\mathcal{O}(n)$. In order to receive useful output for further applications, finding a suitable word size is important. The latter strongly depends on the use case and input data. A good parametrization for detecting movements will be evaluated later.

3.2.1.2 Symbolization

Utilizing the PAA can drastically reduce the dimensionality of the input data. However, to work efficiently with the data, it has to be discrete. Therefore the next step assigns a symbol to each segment S' of the computed PAA. In order to receive significant symbols, they are required to occur equiprobably. To achieve this, the normalization of the original time series data to mean zero and standard derivation one is made use of. This normalized data is similar to a Gaussian Distribution. Therefore, Gaussian breakpoints can be computed or taken from a lookup table as shown in Figure 3.2. These breakpoints split a Gaussian curve into a equiprobable areas. To assert a symbol to each segment, the breakpoints $\beta_0, \beta_1, \cdots, \beta_{a-1}\beta_a$ are chosen for the alphabet size a, where the breakpoints $\beta_0 = -\infty$ and $\beta_a = +\infty$ are always fixed.

For each t'_i in S' a symbol $t^*_i \in \Sigma := \sigma_1, \cdots, \sigma_a$ is selected by

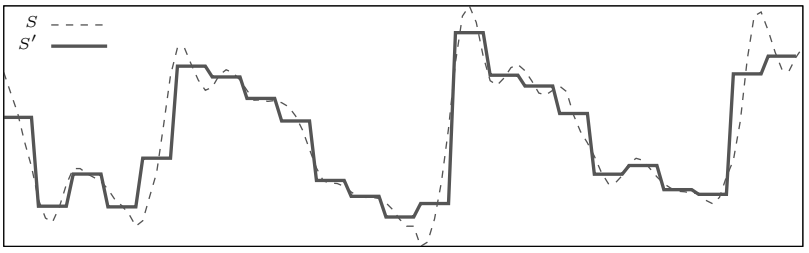

FIGURE 3.1: Plot of a sequence with its Piecewise Aggregate Approximation as overlay.

β_i \ a	3	4	5	6	7	8	9	10
β_1	-0.43	-0.67	-0.84	-0.97	-1.07	-1.15	-1.22	-1.28
β_2	0.43	0	-0.25	-0.43	-0.57	-0.67	-0.76	-0.84
β_3		0.67	0.25	0	-0.18	-0.32	-0.43	-0.52
β_4			0.84	0.43	0.18	0	-0.14	-0.25
β_5				0.97	0.57	0.32	0.14	0
β_6					1.07	0.67	0.43	0.25
β_7						1.15	0.76	0.52
β_8							1.22	0.84
β_9								1.28

FIGURE 3.2: This table shows for alphabet size a up to 10 the corresponding breakpoints β_i to split a Gaussian distribution into a equiprobable regions. [16]

$$t_i^* = \sigma_j \iff \beta_{j-1} \leq t_i' < \beta_j$$

This formula assigns every equiprobable region between two consecutive breakpoints β_{j-1}, β_j to the symbol σ_j. For each segment t_i', we then analyze in which region the segment's value lies and assign the corresponding symbol. This symbolization method asserts equiprobable symbols and yields a discrete symbol string for further algorithms. Due to the fixed alphabet and word size a and w, the symbolization can be accomplished in constant time $\mathcal{O}(a \cdot w)$.

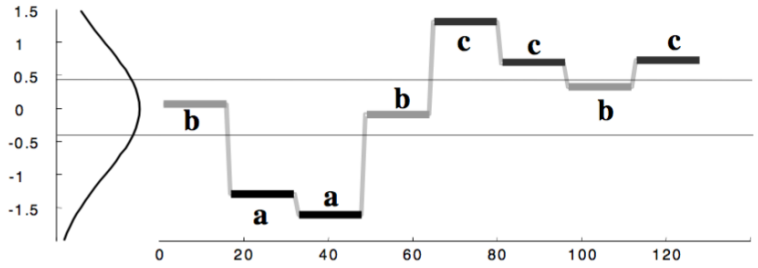

FIGURE 3.3: Plot of an time series with its SAX as overlay. [16]

The alphabet size a and the word size w have high influence on the quality of our approach. We have figured out that $a \in [2, 4]$ and $w \in [2, 6]$ lead to sophisticated results.

3.2.2 Sequitur

The *Sequitur* algorithm is a string compression algorithm. It creates a context free, hierarchical grammar structure consisting of terminal and non-terminal symbols. Sequitur will use the resulting SAX words $T_1^*, \cdots, T_m^*$ as input. Every SAX word from the input string is treated as a terminal and the non-terminals are used to substitute other grammar-symbols during the process. To understand the idea of this method, we first have to introduce digrams. A digram consists of two succeeding grammar-symbols. The input is processed by keeping track of its digrams in a digram dictionary. This digram dictionary ensures the linear runtime for *Sequitur* and always contains every digram in the current grammar.

The input sequence is processed greedy.[1] Every new input word is processed immediately and independently of following input. For this reason, it is a perfect fit for online execution. Each word is stored in a terminal grammar-symbol and then appended to the start rule $R0$. This produces the trivial non-hierarchical grammar $R0 \rightarrow T_1^* \cdots T_m^*$.

[1] Greedy algorithms make a locally optimal decision for each input.

To enforce a meaningful hierarchical grammar, two basic principles are always followed: The *Digram Uniqueness* and the *Rule Utility*.

3.2.2.1 Digram Uniqueness

This principle is responsible for our grammar to be hierarchical and context-free. It states that every digram can appear only once in the whole grammar structure. Thus, the digram dictionary can never contain the same digram more than once. Ensuring digram uniqueness forces us to detect collisions in the digram dictionary every time we try to insert a digram. When a collision is detected, we have to take one of two possible measures: either a new rule is introduced which produces the duplicate digram, and both digrams are substituted with non-terminals or, if such a rule already exists, this rule will be reused to substitute the digram containing the newly added symbol. In both cases, the digram dictionary has to be updated afterwards to contain the latest digrams. This digram update can lead to another collision in the digram dictionary causing chain-substitutions.

Table 3.1: Example for applied digram uniqueness.

	input string	sequitur grammar	digram dictionary
(0)	bbb abc cba bbb	0 → bbb abc cba bbb	bbb abc, abc cba, cba bbb
(1)	bbb abc cba **bbb abc**	0 → **1** cba **1** **1** → **bbb abc**	bbb abc, **1 cba,** **cba 1**
(2)	bbb abc cba bbb abc **cbb bbb**	0 → 1 cba 1 **cbb bbb** 1 → bbb abc	bbb abc, 1 cba, cba 1, **1 cbb,** **cbb bbb**
(3)	bbb abc cba bbb abc cbb **bbb abc**	0 → 1 cba 1 cbb **1** 1 → bbb abc	bbb abc, 1 cba, cba 1, 1 cbb, **cbb 1**

In Table 3.1 a small example for two applications of digram uniqueness is illustrated. All changes caused by the latest added digram are written in bold letters. The first column shows the full input string which leads to the grammar shown in the second column. The digram dictionary after processed input is shown in the last column.
In line (1) a new rule is introduced because the digram **bbb abc** already exists in the digram dictionary and there exists no previous rule producing this digram. In line (3) we reuse the rule created in (1).

3.2.2.2 Rule Utility

When grammar-symbols are substituted some rules remain which are just referenced by one non-terminal symbol in the whole grammar. In this case, the second principle *Rule Utility* applies. It states that every rule of the grammar has to be referenced by at least two non-terminals. This is important to achieve a reduction of grammar size by every rule. When a grammar-rule loses its utility, we delete the rule and replace its remaining non-terminal symbol by the grammar-symbols which have been produced by the deleted rule. After this operation, the digram dictionary has to be maintained to contain the up-to-date digrams again.

TABLE 3.2: Minimal example for applied rule utility.

	input string	sequitur grammar	digram dictionary
(0)	bbb abc cba bbb	0 → bbb abc cba bbb	bbb abc, abc cba, cba bbb
(1)	bbb abc cba **bbb abc**	0 → **1** cba **1** **1** → **bbb abc**	bbb abc, **1 cba, cba 1**
(2)	bbb abc cba bbb **abc cba**	0 → **2 2** 1 → bbb abc **2** → **1 cba**	bbb abc, 1 cba, **2 2**
(3)	bbb abc cba bbb abc cba	0 → 2 2 ~~1 → bbb abc~~ 2 → **bbb abc** cba	bbb abc, 2 2, **abc cba**

In Table 3.2, a minimal example for rule utility is given. Until line (1), it is equal to our previous example. However, in row (2), the digram uniqueness has been violated again because the sequence **1 cba** occurred twice. Therefore, the rule 2 is created and both occurrences of rule 1 are substituted. This causes a violation of the rule utility for rule 1. In (3) the rule utility is reestablished by deleting rule 1 and replacing its remaining occurrence within rule 2 with its derivation **bbb abc**.

The sequitur grammar created in Table 3.3 shows how multiple operations can be performed for a single new SAX word input. In row (1) a new rule is created to ensure digram uniqueness. When the next input is processed in (2), rule 2 is created because the digram **1 cba** occurs twice in the grammar. Due to the substitutions of both non-terminals 1 in the start rule, the rule utility is violated. We therefore delete rule 1 and replace its remaining non-terminal in rule 2 with its derivation **abc cbb**. Line (3) is

another simple operation: the first digram of rule 2 reoccurs leading to another rule creation. The last row results from three chained operations caused by violations of digram uniqueness and rule utility. We start by reusing rule 2 when the digram **3 cba** is processed. This violates the rule utility as rule 3 is utilized just once in rule 2. Thus, the non-terminal is again replaced by its derivation. Additionally the digram **bbb 2** occurs twice in the grammar, infringing the digram uniqueness leading to the additional rule **4** → **bbb 2**.

TABLE 3.3: More complex sequitur example

	input string	sequitur grammar	digram dictionary
(0)	bbb abc cbb cba	0 → bbb abc cbb cba	bbb abc, abc cbb, cbb cba
(1)	bbb abc cbb cba **abc cbb**	0 → bbb **1** cba **1** **1** → **abc cbb**	**bbb 1**, abc cbb, **1 cba**
(2)	bbb abc cbb cba abc cbb **cba**	0 → bbb **2 2** ~~1 → abc cbb~~ **2** → **abc cbb cba**	**bbb 2**, abc cbb, **cbb cba**, **2 2**
(3)	bbb abc cbb cba abc cbb cba **bbb abc cbb**	0 → bbb 2 2 **bbb 3** 2 → **3** cba **3** → **abc cbb**	bbb 2, abc cbb, **3 cba**, 2 2, **2 bbb, bbb 3**
(4)	bbb abc cbb cba abc cbb cba bbb abc cbb **cba**	0 → **4** 2 **4** 2 → **abc cbb** cba ~~3 → abc cbb~~ **4** → **bbb 2**	bbb 2, abc cbb, **cbb cba**, **4 2**, **2 4**

3.2.2.3 Complexity

This subsection will explain the idea of the proof for linear time complexity of *Sequitur*. For additional information, the detailed proof can be found in [14]. The proof to show the linear runtime of this algorithm uses the amortized costs. Instead of analyzing the operations for each processed symbol, the possible operations for the whole sequence are analyzed. Sequitur consists of five operations; two of which are simple and three of which are complex: introducing a new grammar-rule, reusing an existing rule and deleting a rule. Additional to these complex operations, there exist two simple operations with constant runtime which are executed exactly once for each new SAX word leading to a trivial linear runtime: the first step is to select a new input word and appending it to the start-rule. The second step is to check the digram dictionary for collisions.
However, it is difficult to predict how often the complex operations are executed because they can be triggered more than once for a single input word. In fact, they can

be triggered $\mathcal{O}(\sqrt{m})$ times in the worst case when the preceding $\mathcal{O}(\sqrt{m})$ symbols invoke no complex operations. The argumentation is based on the fact that the reduction of the grammar's size is bound by its linear input-size. In combination, both deleting a rule and reusing an existing rule reduce the grammar's size by one, while creating a new rule results in the same grammar-size. The proof bounds the number of rule creations to obtain linear amortized cost for the execution of *Sequitur*.
In summary, *Sequitur* promises to operate quickly due to its linear runtime. Additional, it is a greedy algorithm. This qualifies our approach for online execution because decisions can be made for each new input without knowing the future time series. The space efficiency is hard to predict as it is dependent on the input data.

3.2.3 Adaptions

Before focusing on finding good motifs, we observe the grammar data and its SAX input words. To achieve this, we used a software called *Stickman* to replay recorded robot datasets while displaying a stick figure of the robot's limbs and adding the robot's visuals. The data exploration focused on finding good recordings containing interesting robot movements to use as input for the algorithms. But, on the other hand, we also detected idle phases containing no movement within time series.

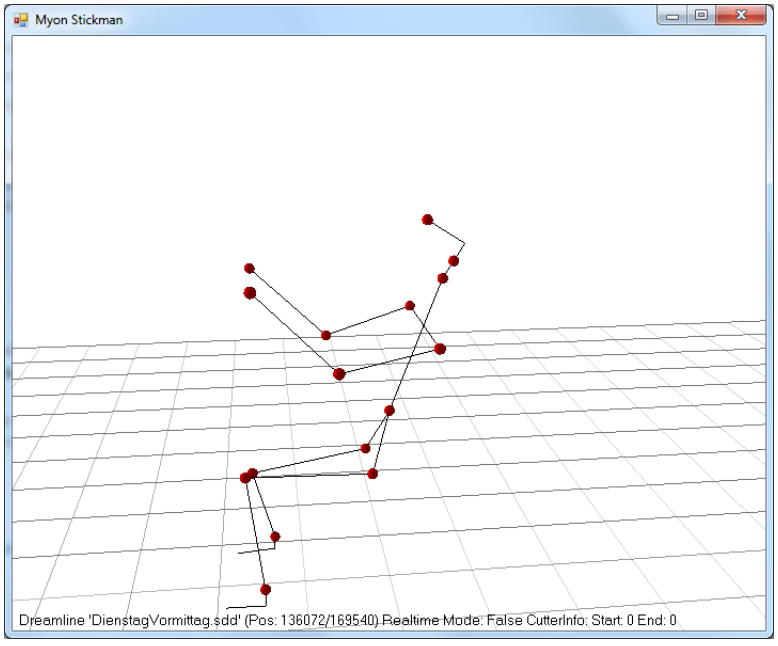

FIGURE 3.4: Screenshot of *Stickman* Software. This Software can replay Myon's limb position from *Dreamlines*

These idle sequences in the data create an important issue: idle sequences usually results in SAX words consisting of b^*; this will create grammar structures like those in Table 3.4. Especially long idle phases create huge and deep grammar structures making it difficult

to distinguish between grammar for complex movements and grammar for idle phases. The solution for this problem is to ignore repeated SAX words. This solution has also been used in [9] to remove trivial matches and achieve better space efficiency.

TABLE 3.4: Sequitur grammar during idle phase.

input string	sequitur grammar
$(bbb)^{31}$	$0 \rightarrow 3\ 3\ 3\ 2\ 1$ bbb $1 \rightarrow$ bbb bbb $2 \rightarrow 1\ 1$ $3 \rightarrow 2\ 2$

This measurement not only has the benefit of avoiding idle grammar structures, it also enables the matching of slower movements with fast movements as shown in Table 3.5. Therefore, the matching of movements is not as strict and it enables us to match similar movements of different lengths. The exact effect of this measurement will be evaluated in chapter 5.

TABLE 3.5: Slow vs fast movement with ignored repeated words.

	input string
fast movement	bbb abc cba bbb abc cba bbb
slow movement	bbb abc ~~abc~~ cba ~~cba~~ bbb abc ~~abc~~ cba ~~cba~~ bbb

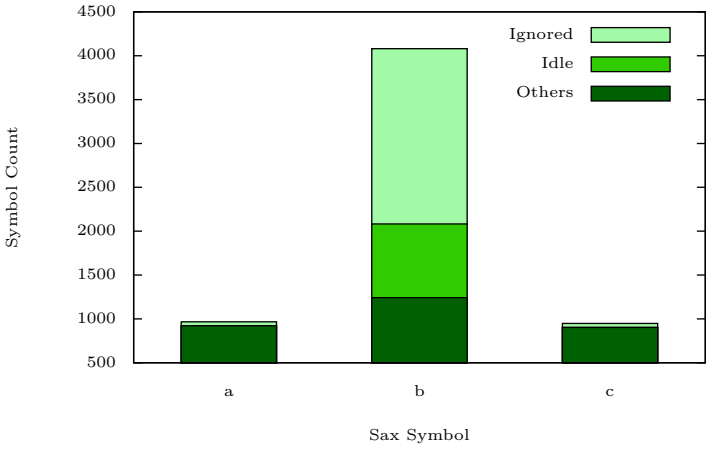

FIGURE 3.5: This plot is generated using a dataset recorded during a rehearsal session for an opera, where Myon participated. SAX was parametrized with $a = 3, w = 3n = 24$.

In Figure 3.5, the distribution of symbols in one example dataset is plotted. The different shades of green separate the symbols between three groups. Light green symbols occur in

ignored SAX words, green symbols in remaining idle words *bbb*, and dark green symbols are not ignored and not idle. The plot shows the desired effect of ignoring words to reduce idle structures. While less than 5% of symbol a and c are ignored, we ignore 50% of all b's. The remaining idle words (green) cannot create grammar structures like those in Table 3.4 because their words do not repeat and should cause no problems. Additionally, the idle symbols separate between movements and making it wrong to ignore every idle symbol.
Another problem of the robot's data is that it contains small fluctuations due to the precision of our float values and very small unintended movements. To gain reliable data, the last five bits of each data point are set to zero because these are imprecise. The unintended motion in the limbs can be tackled by either a low-pass filter [17, 18] or using a standard derivation threshold. Currently, the second option is implemented: the normalization is extended by setting the whole sequence to zero if the standard derivation is below a fixed value. By doing this, imprecise values as well as too small unintended movements do not influence the outcome.

3.2.4 Interesting rules

To enable Myon to learn motifs, we first have to extract these motifs from our hierarchical sequitur grammar structure. In a first step, instead of extracting complete motifs, interesting grammar rules are extracted. This is accomplished by considering every grammar rule when it is reused or inserted. The decision whether a rule is extracted is made by computing a rule interest. As proposed in [9], a rule interest is computed whenever a grammar rule is substituted. This rule interest uses weights for different features of grammar rules:

- The number of rule utilization of the rule in the whole grammar. This feature only reveals little information about the rule interest and thus will be weighted quite low.
- The length of the given rule. This is the number of terminals within the original sequence of a rule. Longer rules are usually more interesting than short rules.
- The rule complexity of the given rule which counts the number of non-terminals in the given rule and its child-rules. Rules consisting of many other rules are more complex, and thus more interesting.

A minimum rule interest is defined as a threshold to identify interesting rules. This enables the algorithm to identify and suggest important rules autonomously.

Chapter 4

Implementation

While implementing the algorithms, at first a simulation system was implemented to be able to test and especially debug the code more easily.[1] It is important to note that we never use fake data at any point to assure a correct performance of our methods. The software is written in C without external libraries. In the next sections we will discuss design decisions and deliver a detailed documentation for our code.
Our program consists of four main parts: the main logic collecting data and controlling the main flow of the algorithms. A small statistics module which offers methods to normalize data. The SAX module implementing all functions necessary to transform sequences into their SAX representation, and a Sequitur module which contains methods to compute the Sequitur grammar. In the simulator, we implemented an additional file input and output module holding functions to read recorded Dream-data and to generate plot files. We use the free gnuplot library to for our plots.[2] The Sequitur module contains a higher complexity compared to the other modules, therefore most of this chapter will cover its implementation.

4.1 SAX

All algorithms use the same configuration containing the following parameters:

- sequence length n: Defines the length of each sequence as the number of data points.
- alphabet size a: Defines the size of the SAX alphabet and thus controls the dimensionality reduction.

[1] The simulation software is copied onto the DVD attached to this thesis.
[2] Gnuplot can be downloaded from `http://www.gnuplot.info/`

- number of words w: Defines the number of words each sequence is transformed to.
- rule interest threshold r: Defines the rule interest threshold. The resulting plots are generated for all rules with an interest higher than r.

At the beginning of the the SAX transformation function we ensure a valid SAX configuration. A configuration is valid if $\frac{n}{w} \in \mathbb{N}$ and $a \in [2, 5]$. The algorithm needs to be able to divide each sequence into w equal sized segments for the Piecewise Aggregate Approximation, and therefore the sequence length n has to be a multiple of the number of words w. The breakpoints are read from a breakpoint table which we defined up to alphabet size five. Alphabet sizes of less than two make no sense because the resulting SAX transformation would consist of only one repeated symbol. The sliding window is not a parameter because we fixed it to the half of the sequence length.

SAX$(S, \Sigma, \beta) \rightarrow (S^*)$
Data:
$S := s_1, \cdots, s_n$: Next Subsequence.
$\Sigma := \sigma_1, \cdots, \sigma_a$: Alphabet
$\beta := \beta_0, \cdots, \beta_a$: Breakpoints
Result: $S^* := s^*_1, \cdots, s^*_w$: Resulting SAX word.

```
1  wordsize ← n/w;
2  for i = 1, ··· , w do
3  |   paa_sum ← 0;
4  |   for j = 1, ··· , wordsize do
5  |   |   paa_sum ← paa_sum + s_j;
6  |   end
7  |   paa_segment ← paa_sum/wordsize;
8  |   for k = 0, ··· , a do
9  |   |   if paa_segment < β_k then
10 |   |   |   s*_i ← σ_k;
11 |   |   end
12 |   end
13 end
```

Algorithm 2: Pseudo Code of the SAX algorithm.

After the validation of the SAX configuration, the SAX transformation is computed. The function expects two arguments: a normalized sequence of size n, and an already allocated character array of size w containing the resulting SAX representation after execution. The SAX transformation is implemented with one outer and two inner loops. The outer loop is executed w times, each run determines $s*_i$ the next symbol. The

first inner loop builds a sum over the data points in the segment and divides it by the segment's size for the PAA value (lines 4-7). Then the second inner loop searches for the symbol corresponding to the PAA value by looping over the breakpoints (lines 8-12). Each transformation can be accomplished in

$$\mathcal{O}(w \cdot (\frac{n}{w} + a)) = \mathcal{O}(n + w \cdot a) = \mathcal{O}(n)$$

because w and a are constants.

4.2 Sequitur

The main challenge for the implementation of Sequitur is to find suitable data-structures. It is important to have an outline of all possible scenarios to design the program structure. Therefore, we take another look at the basic principles of Sequitur resulting in five operations. These required operations have already been introduced in subsection 3.2.2:

- Appending a new terminal to the start rule
- Checking the digram dictionary for collisions
- Introducing a new grammar rule
- Reusing an existing grammar rule
- Deleting a grammar rule

Our implementation is roughly structured by these five operations. However, some methods execute multiple of these operations.

4.2.1 Data Structure

Sequitur is implemented using five structs, two of which are used for the digram dictionary, and three of which are used for the Sequitur grammar. The digram dictionary consists of the digram table and several digram lists. The table holds its size, the current element count, and a pointer to its buckets. The size controls the number of buckets in this table. Each bucket contains one digram list which is a one direction linked list consisting of one pointer to a digram and another pointer to the next element of the list. The Sequitur grammar consists of three data-structures: symbol, rule, and grammar. The symbol structure is similar to a linked list working in both directions. Every symbol

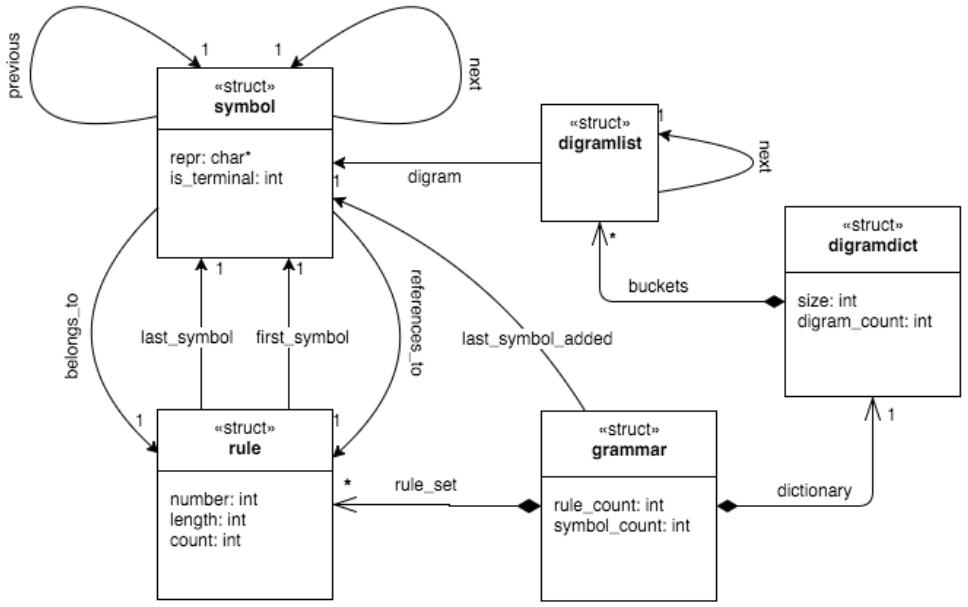

FIGURE 4.1: Diagram showing the structs and its connections for the Sequitur implementation

has a pointer to its previous and next symbol. Additionally, every symbol can either have a pointer to a character array holding its SAX word, or hold a pointer to its referenced grammar rule. Depending on which of both fields is set, the symbol is either a terminal or non-terminal symbol. Some symbols represent guards. These are the first or the last symbol of a rule. They hold a pointer to the rule they belong to. This guard structure is important for some operations to be executed in constant time. The reserved storage for symbols and digrams can be controlled by constants.

A Rule holds three attributes: the rule's length, invocation count, and its unique rule number. Additionally, each rule has two pointers: one to its first symbol, and another to its last symbol. The grammar struct acts as a main structure. Besides the grammar and symbol count, it holds an array of all the grammar rules and a reference to the digram dictionary. In the current implementation, the grammar rules are limited to 4096. The grammar also has a pointer to the last symbol added.

On startup of the robot, this structure is initialized. First of all, every bucket of the digram dictionary is created containing an empty list and stored in the dictionary. The empty start rule with number 0 is then created and introduced to the grammar.

4.2.2 Logic

Sequitur(s);
Data: s: New terminal symbol.

```
1  if not s equals grammar.last_symbol_added then
2  |  grammar.last_symbol_added ← s;
3  |  append symbol s to start rule R_0;
4  |  if digram_dictionary contains (s.previous, s) which does not overlap then
5  |  |  old_digram ← digram_dictionary((s.previous, s));
6  |  |  merge_digrams(old_digram, s.previous);
7  |  else
8  |  |  insert (s.previous, s) into digram_dictionary;
9  |  end
10 end
```

Algorithm 3: Pseudo code of Sequitur's main routine.

Sequitur requires one character array parameter containing the next SAX word as input and returns no value. It merely changes the current Sequitur grammar. The main logic for Sequitur is spread into four functions not counting small helper methods for e.g. dictionary operations and digram comparison.
The main sequitur function (see algorithm 3) fulfills simple tasks which have to be executed for every new SAX word. At first, it compares the new symbol with the previously added symbol. If they are equal, Sequitur stops because repeated symbols are ignored. Otherwise, we continue by adding the new symbol to the grammar's start rule. Finally, Sequitur attempts to add the new digram starting at the new symbols' previous symbol to the digram dictionary. In case the digram already exists in the dictionary, a method responsible for solving digram conflicts is invoked.

Merge_Digrams(old, new**)**;
Data:
old: First symbol of old digram
new: First symbol of the digram causing a conflict

```
1  if old and old.next are guards then
2  |  R_old ← old.belongs_to;
3  |  non_terminal ← create_non_terminal(R_old);
4  |  substitute(new, non_terminal);
5  else
6  |  R_new ← create_rule();
7  |  non_terminal ← create_non_terminal(R_new);
8  |  substitute(old, non_terminal);
9  |  non_terminal ← create_non_terminal(R_new);
10 |  substitute(new, non_terminal);
11 end
12 foreach non-terminal symbol n which has been substituted do
13 |  reduce_references(n)
14 end
```

Algorithm 4: Pseudo code of the merge digrams subroutine.

The function *merge_digrams* reestablishes the digram uniqueness. It expects two parameters: a pointer to the old digram, and another pointer to the newly added digram. The old digram is stored in the digram dictionary. The new digram is equal to the old one and therefore it's insertion violated the digram uniqueness. After the execution of this function, the conflict is resolved by either introducing a new rule to the grammar (lines 2-4) or by reusing an existing rule (lines 6-10). To decide whether a new rule has to be introduced, we check if both symbols of the old digram are guards. This means that the rule consists only of the symbols of the digram. If this is the case, the rule will be reused to substitute the new digram. If at least one of the symbols in the old digram is not a guard, a new rule has to be created. The old digram's symbols are added to the new rule and then we first substitute the old digram by a non-terminal symbol referencing the new rule and then the new digram by another non-terminal symbol referencing the new rule. After both, rule creation and rule-reusage, we reduce the rule invocation count for each non-terminal symbol in any substituted digrams.

Substitute($s, non\_terminal$);
Data:
s: first symbol of digram which has to be substituted
$non\_terminal$: substitution for the digram
1 remove outdated digrams from the dictionary;
2 update guard pointers to point to $non\_terminal$ if s or $s.next$ are guards;
3 replace $(s, s.next)$ by $non\_terminal$ in grammar structure;
4 $non\_terminal.references\_to.count++$;
5 **foreach** new created digram s' **do**
6 **if** $digram\_dictionary$ **contains** $(s'.previous, s')$ which does not overlap **then**
7 $old\_digram \leftarrow digram\_dictionary((s'.previous, s'))$;
8 $merge\_digrams(old\_digram, s'.previous)$;
9 **else**
10 insert $(s'.previous, s')$ into $digram\_dictionary$;
11 **end**
12 **end**

Algorithm 5: Pseudo code of the substitute subroutine.

The *substitute* function expects two symbols. The first parameter is the first symbol s of the digram which should be substituted by the non-terminal symbol $non\_terminal$, passed as second parameter. At first, existing outdated digrams are removed from the dictionary. The possible outdated digrams are the preceding and succeeding digrams of symbol s. If any of these exist, they are deleted out of the digram dictionary. The old digram starting at s is substituted, by unchaining it from the grammar's structure and redirecting all pointers directed to s to point to $non\_terminal$. The next step is to consider the guards; if any of the two symbols in the substituted digram have been guards, the non-terminal symbol inherits all guard pointers. Additionally the *first_symbol* and *next_symbol* pointers of the guarded grammar rule are redirected to the non-terminal, if any of those were directed to a symbol in the old digram. The substitution is finished by inserting all newly created digrams in the digram dictionary. If any of these inserts cause another conflict, the *merge_digrams* procedure is invoked again. (lines 5-11).

Reduce References(*non_terminal*);
Data: *non_terminal*: non-terminal symbol which has been replaced

```
1  R ← non_terminal.references_to;
2  R.count − −;
3  if R.count == 1 then
4  |  Remove old digrams from dictionary;
5  |  Remove guard pointers from R.first_symbol and R.last_symbol;
6  |  R.first_symbol.previous ← non_terminal.previous;
7  |  R.last_symbol.next ← non_terminal.next;
8  |  non_terminal.previous.next ← R.first_symbol;
9  |  non_terminal.next.previous ← R.last_symbol;
10 |  if non_terminal is a guard then
11 |  |  update guard pointer to point to either R.first_symbol or R.last_symbol
12 |  end
13 |  foreach new created digram s' do
14 |  |  if digram_dictionary contains (s'.previous, s') which does not overlap then
15 |  |  |  old_digram ← digram_dictionary((s'.previous, s'));
16 |  |  |  merge_digrams(old_digram, s'.previous);
17 |  |  else
18 |  |  |  insert (s'.previous, s') into digram_dictionary;
19 |  |  end
20 |  end
21 end
```

Algorithm 6: Pseudo code of the *reduce_references* subroutine.

Reducing the invocation or reference counter of a grammar rule is implemented in the *reduce_references* method. It expects a non-terminal symbol as input. After termination of the function, the rule invocation counter of the rule referenced by the non-terminal input symbol is reduced by one. If this violates the rule utility constraint of Sequitur, the rule is deleted and the remaining non-terminal symbol is replaced by the symbols, the rule derives to. The rule utility is violated if and only if the rule invocation counter is reduced to exactly one. In case this constraint is not infringed, the process returns after reducing the counter. Otherwise, to rehabilitate the rule utility at first the guard pointers are removed. Then the preceding and succeeding digrams of the non-terminal are removed from the dictionary. Now the non-terminal is removed from the grammar by chaining together the previous symbol of the non-terminal with the first symbol of the grammar rule and the last symbol of the grammar rule with he next symbol of the non-terminal (lines 6-9). If the non-terminal symbol has been a guard, its guard pointers

are copied and redirected to either the first or last symbol of the deleted grammar rule. Finally, all new created digrams are inserted into the dictionary, resolving possible additional conflicts (lines 13-20).

4.3 Creating Plots from Rules

Whenever the *merge_digrams* function is executed it computes the rule interest for the newly inserted or re-used rules. If the rule interest exceeds the defined threshold the program automatically generates plot files. On the one hand, a plot for the full time series with every high interest rule marked. On the other hand one plotfile for each high interest rule, plotting all its occurrences. These files will be used for our experiments in the following chapter.
All in all, we build a simulation program to transform recorded data from Myon into hierarchical grammar structures. The resulting files can be used to quickly plot all interesting rule occurrences and the complete time series with marked interesting rule sequences. These files will be stored in a subfolder called *plotfiles*. For each parametrization a new subfolder named by the chosen parameters will be created.

Chapter 5

Evaluation and Experiments

The evaluation works with recorded data of the robot.[1] This data has the same format as the data used when executing the algorithms directly on the robot. Yet, the data points of the recorded time series can have a varying distance to their neighboring points. The gaps between data increase if the robot performs blocking tasks. Therefore, the data has been rescaled to ensure a gap of 40 milliseconds between each neighboring pair of values. This can be done because each recorded data point also stores the current uptime of the robot.

5.1 Expectations

The goal of the experiments is to either verify expectations or to analyze why certain expectations are not met. The anticipated results are formulated in the following list.

- The length of time series should not influence its results. Short input data with lesser motifs and data with multiple motifs should both be processed efficiently and yield good results.
- Although the used algorithms only work with one dimension and most movements occur in multiple limbs, the quality of detected motifs is expected to be good. This is founded in the belief that the movements in a single limb are still unique for many motifs if the limb participates.
- The results of our approach should contain few false-positives.
- The intensity of the movements should not have an influence on motif detection as the data is first normalized.

[1] The used datasets are stored on the attached DVD.

- The used methods should be able to detect different kinds of movements.
- Correct parametrization of our approach has immediate effect on the quality of the outcome. Therefore, we want to find a well suited parametrization for motif detection in motion time series.
- The performance of this approach should have linear execution time and, for long datasets with many repetitions, the resulting grammar's size should follow a saturation curve.

5.2 Experiments

The following sections contain experimental results. The datasets will increase in size and complexity. The focus of these experiments are the exploration of the detected rules to first validate our expectations and to later gain information about the relationship between sequitur rules and motifs. The outcome of these experiments is substantial to build the online motif detection directly on the robot later on. All plots are directly generated by our offline implementation.[2] This implementation generates plot-files for each interesting rule and one plot for the whole time series.

5.2.1 Simple Dataset

The first data set is a very short time series. At the beginning of the recording, the robot's left arm is hanging down. While recording the arm is moved up by about 90 degrees, tilted left to right and released into its starting position. This is repeated four times. This experiment tests that little repetition of a motif is enough to detect it, and it helps finding a good first parametrization since the resulting grammar can still be manually reproduced.

TABLE 5.1: Sequitur Grammar created from simple dataset using Parametrization $a = 2, w = 3, n = 24$

$0 \rightarrow$ 7 7 12 10 12 aaab 13 abba baaa bbbb
$7 \rightarrow$ baaa abaa bbaa bbba 10
$10 \rightarrow$ baab 13 bbaa
$12 \rightarrow$ baaa aabb bbaa
$13 \rightarrow$ aabb abbb

[2]The Simulator program is delivered on the attached DVD.

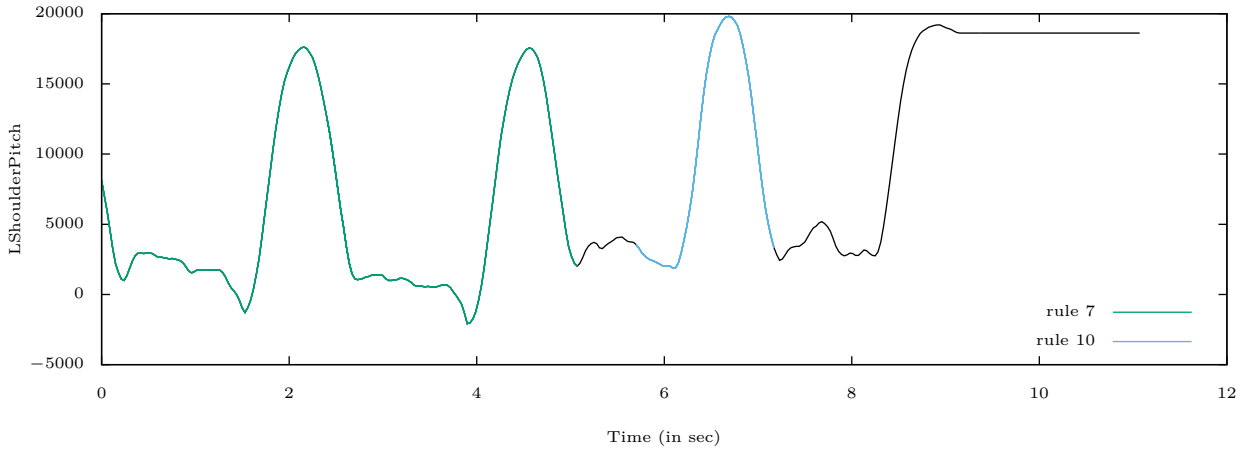

FIGURE 5.1: Motifs detected in simple experiment.
Parametrization: $a = 2, w = 4, n = 20$.

The results of the SAX and Sequitur algorithms can be seen in Figure 5.1. The plot illustrates the movement of the robot's shoulder pitch on the y-axis over the execution time on the x-axis. The black line is the raw input data, and the colored lines are subsets of this data corresponding to the detected most interesting rules. The matching Sequitur grammar is shown in Table 5.1. The gaps in the rule numbering is caused by rule-deletions. The first six rules have been used to build up rule 7. Rule 8 and rule 9 were deleted and replaced by rule 10. This result matches our expectation of achieving good motif detection even in small datasets. Our planted motif was detected and found in three spots. While rule 7 only occurs twice, rule 10 occurs three times: twice within rule 7 and once in the starting rule. Thus, rule 7 and 10 match occurrences of our motif. This data set is too small for statements about false positives because almost the whole time series belongs to the same motif.
Alphabet size two, sequence length 20, and four SAX symbols for each sequence have been chosen for the above outcome.

5.2.2 Rehearsal Dataset

The next time series has been recorded during a rehearsal for an opera. The conductor of the orchestra showed Myon how to conduct by moving the robot's arm. This time series is longer and contains more motifs. The expectation of the resulting grammar is to have motifs matching arm movements during the conducting phase of the time series. All parts in which the robot simply waits should be skipped and should not influence the grammar.

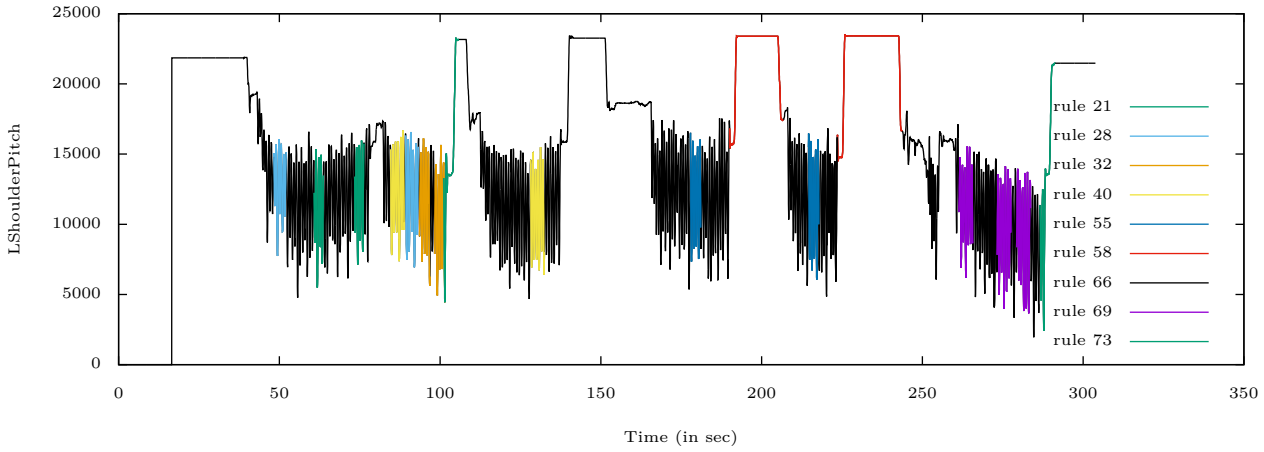

FIGURE 5.2: Motifs detected in longer time series recorded during a rehearsal session. Parametrization: $a = 2, w = 3, n = 24$.

The resulting sequitur grammar's most interesting rules illustrate how well Sequitur works. Although the alphabet only consist of two symbols and every SAX word has only three letters, our approach is able to match multiple rules during the conducting phase. In Figure 5.2, the full time series is plotted and the most interesting rules it has detected are colored. Observing the patterns which the rules derive to yields that most of the rules match conducting movements. Their curves look sinus-like with varying amplitudes. To inspect single rules, multi-plots are generated by drawing each occurrence of the rule in the main-plot on the upper side and it's child rules' occurrences below that.

In Figure 5.3, a interesting rule (rule 28) is observed. The rule matches sequences which occur while the robot is performing conducting movements. It spans over four seconds and is a sinus-like curve over four and a half wave periods. The amplitudes vary between the two occurrences of the rule. Due to the normalization, Sequitur is still able to match both into the same rules. This behavior can be observed multiple times in the child rules too. This only works because the wave-length of the curves are the same. This result is very important because in general, intensities are most of the times not important to identify the type of a movement.

One rule even matches the breaks between arm movements. Rule 58 is one of these pause movements. It occurs twice, once during a 12 second break and once during a 19 second break. This can be accomplished because repeated words are skipped. Although the rule itself does not contain much movement, identifying it is handy to split a recorded Dream into multiple parts. For our conducting purpose, we could use rules like this to split our data into multiple songs. The conductor could tell Myon when a song starts

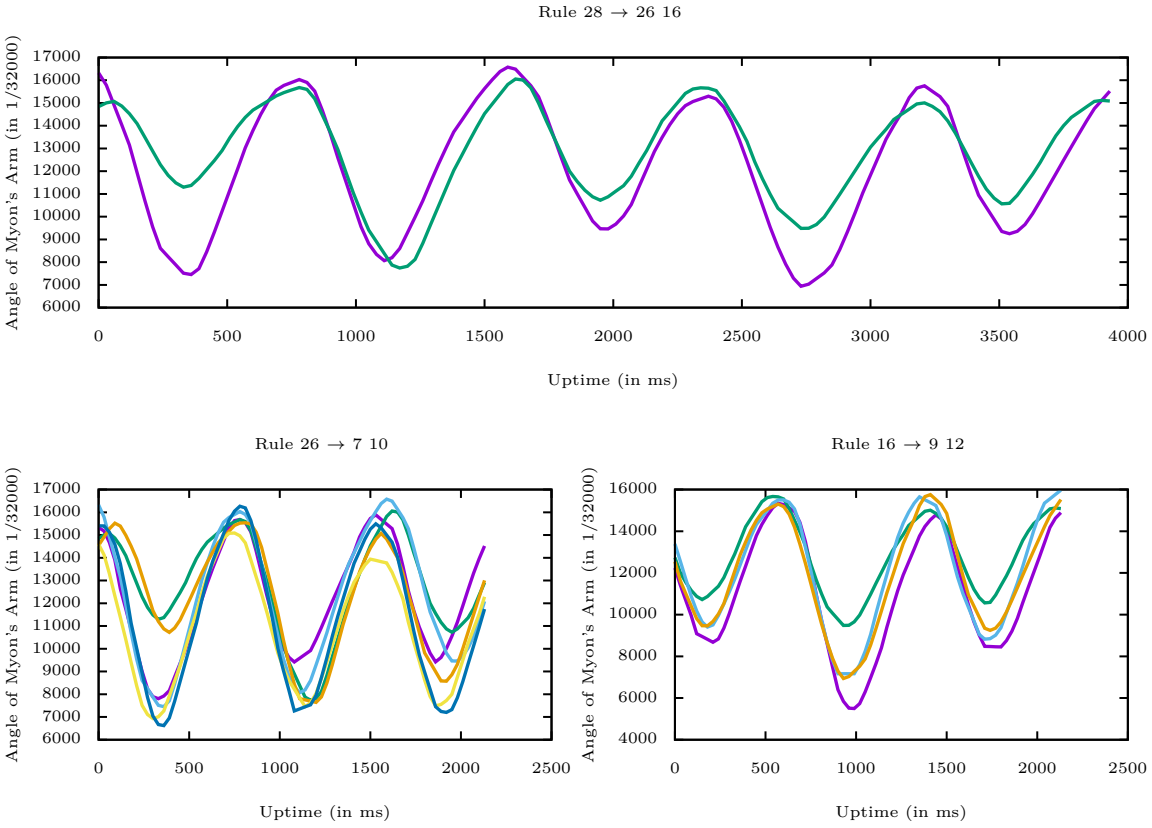

FIGURE 5.3: Occurrences of a conducting rule and its child rules.

and how it is called, conduct it with the robot's arm and Myon could identify the end of the song by an occurrence of a rule matching a pause motif.

5.2.3 Conducting Dataset

The robot participated in the aforementioned opera. During the premiere of the play, some data was recorded. One recording was cut to fit one scene which is interesting for arm movements. At first, the conductor of the orchestra conducts while Myon just sits next to him. Then the conductor takes the robot's arm and conducts with him. In the end of the scene, the robot conducts by himself. Here we hope to be able to find rules, matching the guided with the autonomous conducting.

We achieved the best results with a parametrization of $a = 2$, $w = 4$ and $n = 24$. In Figure 5.5, the plot for the described time series is shown. The two main conducting parts occur at 250 seconds until 290 seconds and at 310 seconds until 370 seconds. The algorithm detects most of its interesting rules within these intervals, especially the second interval is almost completely covered by sequences matching rules.

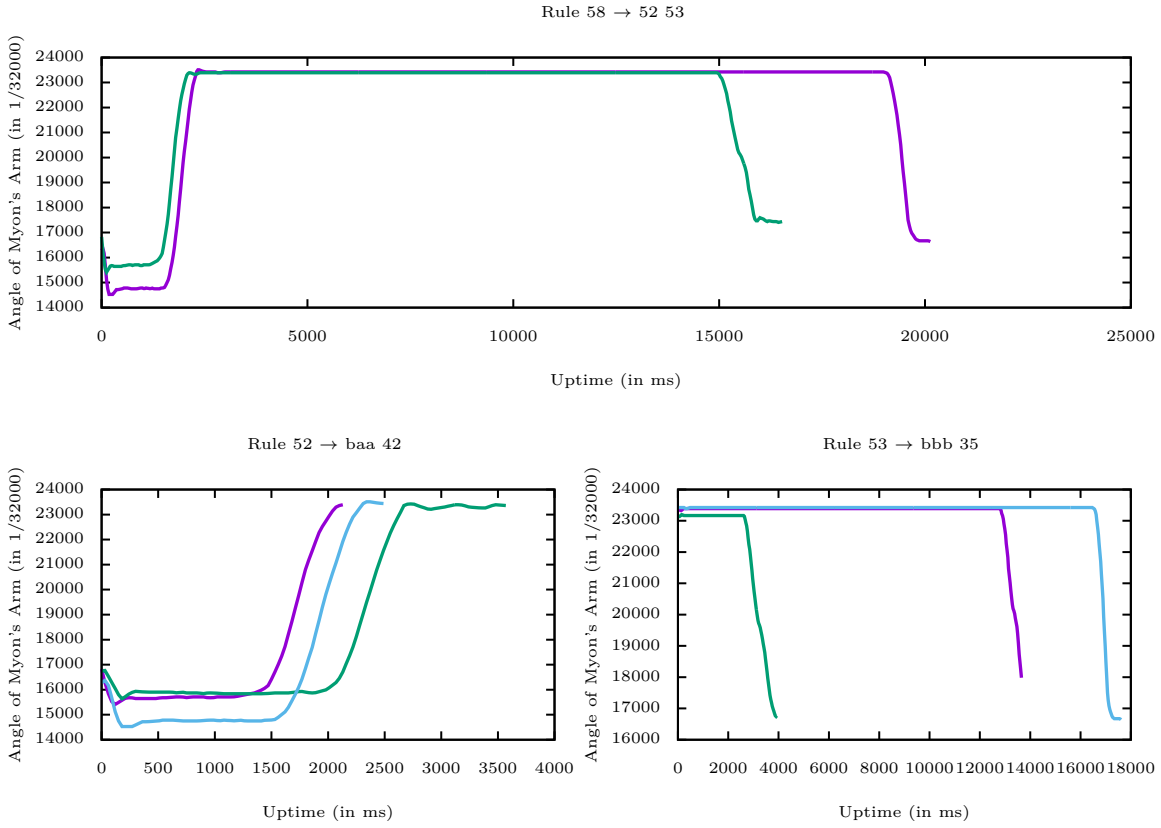

FIGURE 5.4: Occurrences of a pause rule and its child rules.

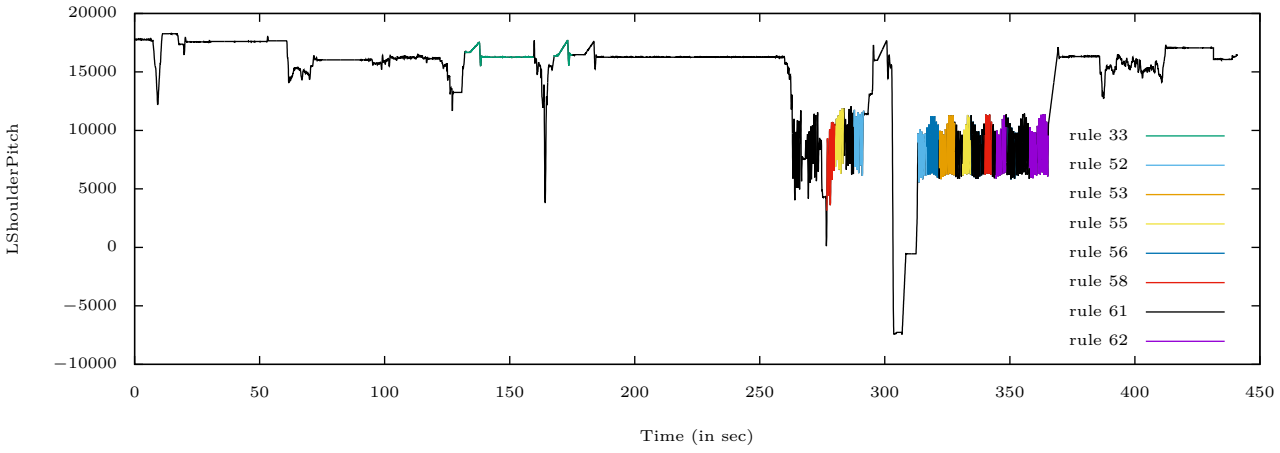

FIGURE 5.5: Motifs detected in longer time series recorded during a scene of the opera play.
Parametrization: $a = 2, w = 4, n = 24$.

The plots of the rule occurrences show that there is only one high interest rule matching sequences from the guided-conducting phase with sequences from the second conducting

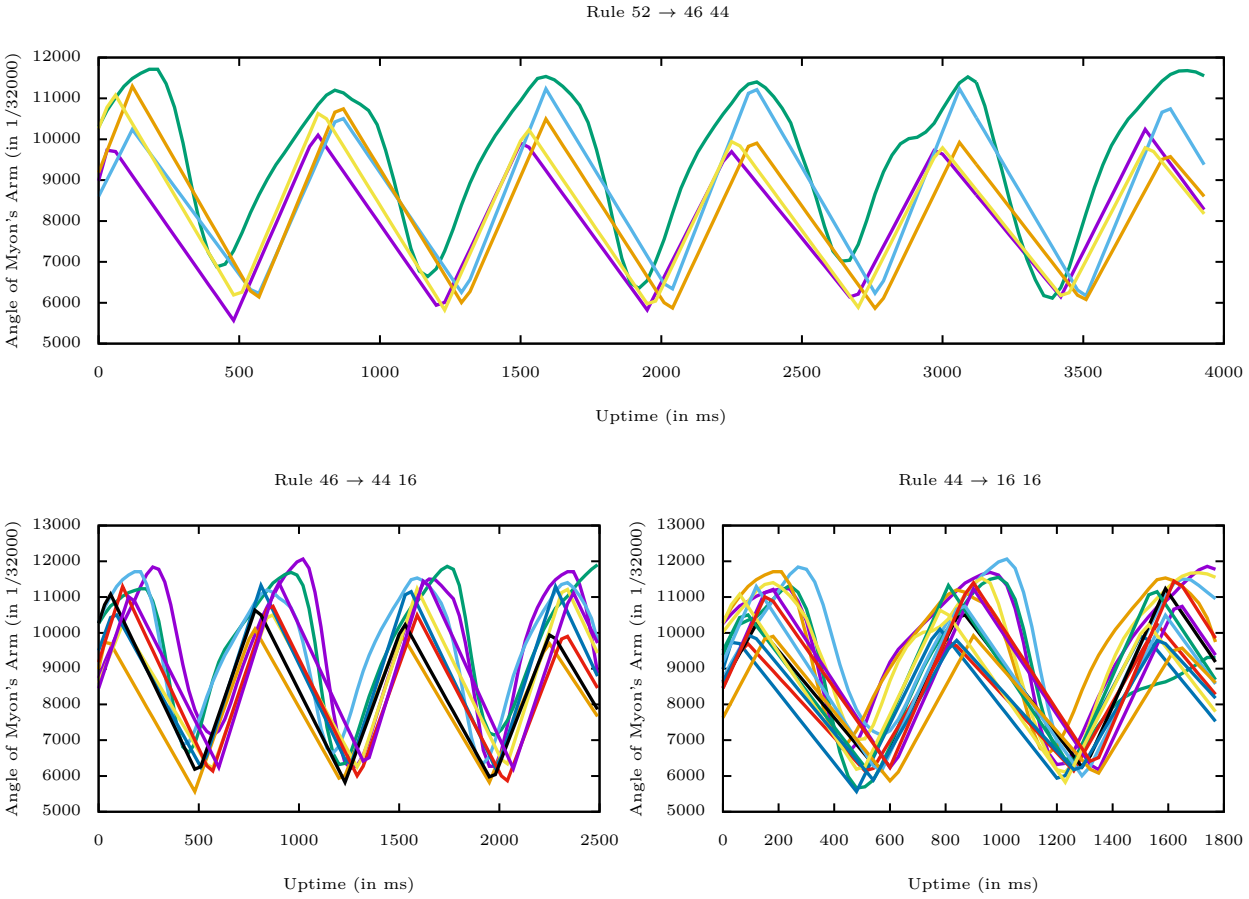

FIGURE 5.6: This shows a detailed plot of all occurrences of rule 52 and its child rules.

phase. However, some child-rules also share this property. Using this observation, rule-sets can be created to match motifs. To do this, a subset of the grammar consisting of the interesting rules and all its descendant rules are examined. The resulting grammar is shown in Table A.1. Then sets are created for all rules which share common descendants.

These rule-sets are illustrated in Figure 5.7. The circles of the Venn diagram represent the detected interesting rules. Overlapping rules share common descendants. The more common descendants exist, the more surface of the circles overlaps. This visualization shows that some rules contain other interesting rules. Rule 61, for instance, derives to rule 56, which contains rule 52. All of these three rules invoke rules 42, 46 and 16. Thus, they are strongly connected and serve as good candidates for a motif. Rule 55 should also be included into this motif-set as it also shares three descendants with them, while rule 33 only shares rule 16 as descendant. By continuing to build up sets like this with the other rules, we could divide our interesting rules into three sets:

$$S_1 := \{R61, R56, R55, R52\} S_2 := \{R62, R58, R53\} S_3 := \{R33\}$$

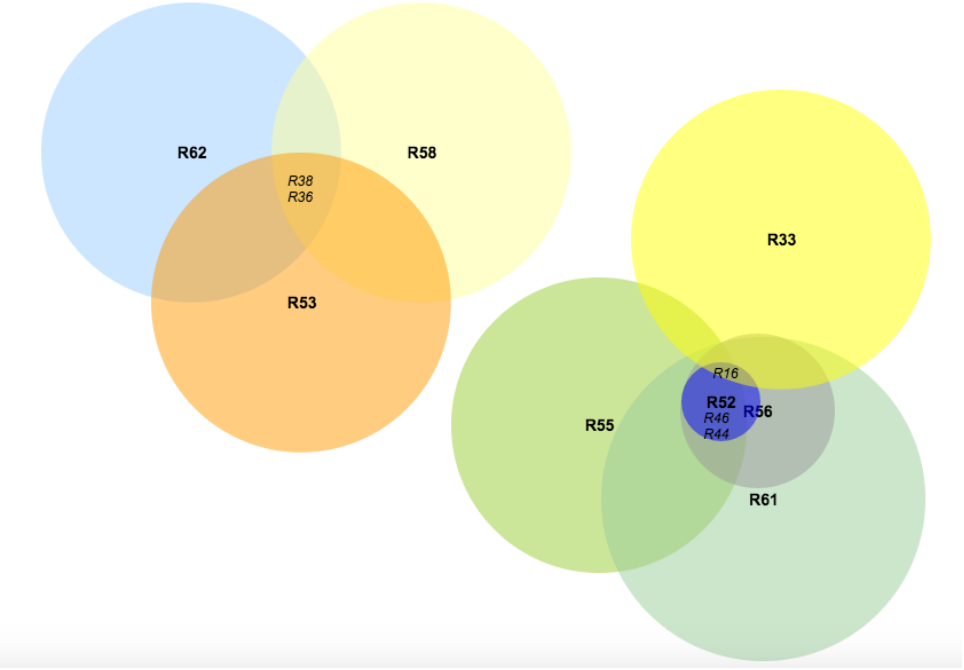

FIGURE 5.7: Venn diagram visualizing all interesting rules with common descendants. The circles with bold labels represent interesting rules. The italic labels represent the common descendants of an overlap.

Inspecting the detailed rule plots of these sets yields that these sets are well suited to match motifs. The plots are shown in section B.1

5.2.4 Combined Data Set

The dataset for this experiment is a time series resulting from concatenating the three previously analyzed time series together with another sequence recorded during a rehearsal session. The main goal of this setup is to ensure a low number of false-positives amongst rules found interesting.
The resulting grammar consists of roundabout 1000 symbols spread among 170 rules, 42 of which had a rule interest higher than the threshold 10. All of these 42 rules have been checked for false-positives. We define false-positives as rules with at least one intuitively wrong occurrence. All in all, only four rules could be labeled as false positives. Even for those cases, the occurrences are not completely different but diverge just in small parts.

In Figure 5.8, one false positive rule is displayed. While the orange and blue lines match well, the yellow line varies after one second. All three occurrences are matched into this rule because they share the sinus-like characteristic lasting over three wave-periods. The only difference is that the last period of the yellow line is significant longer than the first two. Due to the skipping of repeated words, this last part is transformed into the same SAX words and thus matches the same sequitur rule. There exists one more false

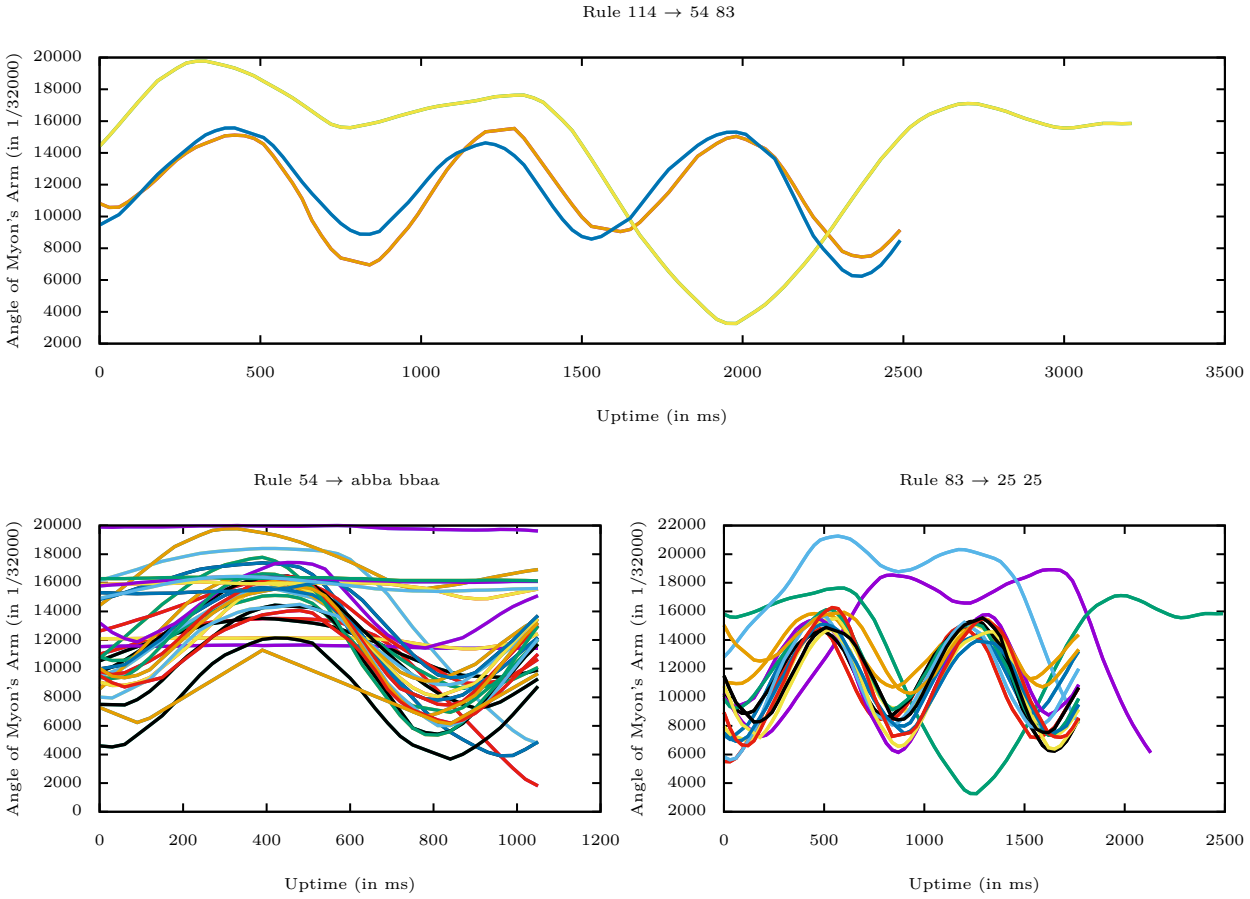

FIGURE 5.8: This plot shows one false positive caused by the "stretch-effect"

positive caused by this stretch-effect. It is arguable whether similar movements executed slower are false positives or intended matches. Making this decision depends on the use case of motif detection. Provided that these false positives should be avoided, repeated words should not be skipped anymore or should be limited to words representing idle-phases.

Furthermore, some false-positives are caused by normalization. While normalization usually benefits the results of this approach, sometimes it leads to huge differences between the original data and the normalized versions. This can cause sequences with almost no movement to match sequences with strong movements which just happen to have the same wave-length. One example from this experiment's results is shown in Figure 5.9. The blue line is a movement with a peak-to-peak amplitude of about 8000, while the orange line's amplitude is around 200. Yet, these movements are matched because their peaks occur in the same interval and the normalization enhances low amplitudes while shrinking high amplitudes. Many of these false positives are already being avoided by filtering sequences with too low standard derivation and simply setting

their data to zero values. When too many false positives caused by normalization remain, the minimum standard derivation should be incremented.

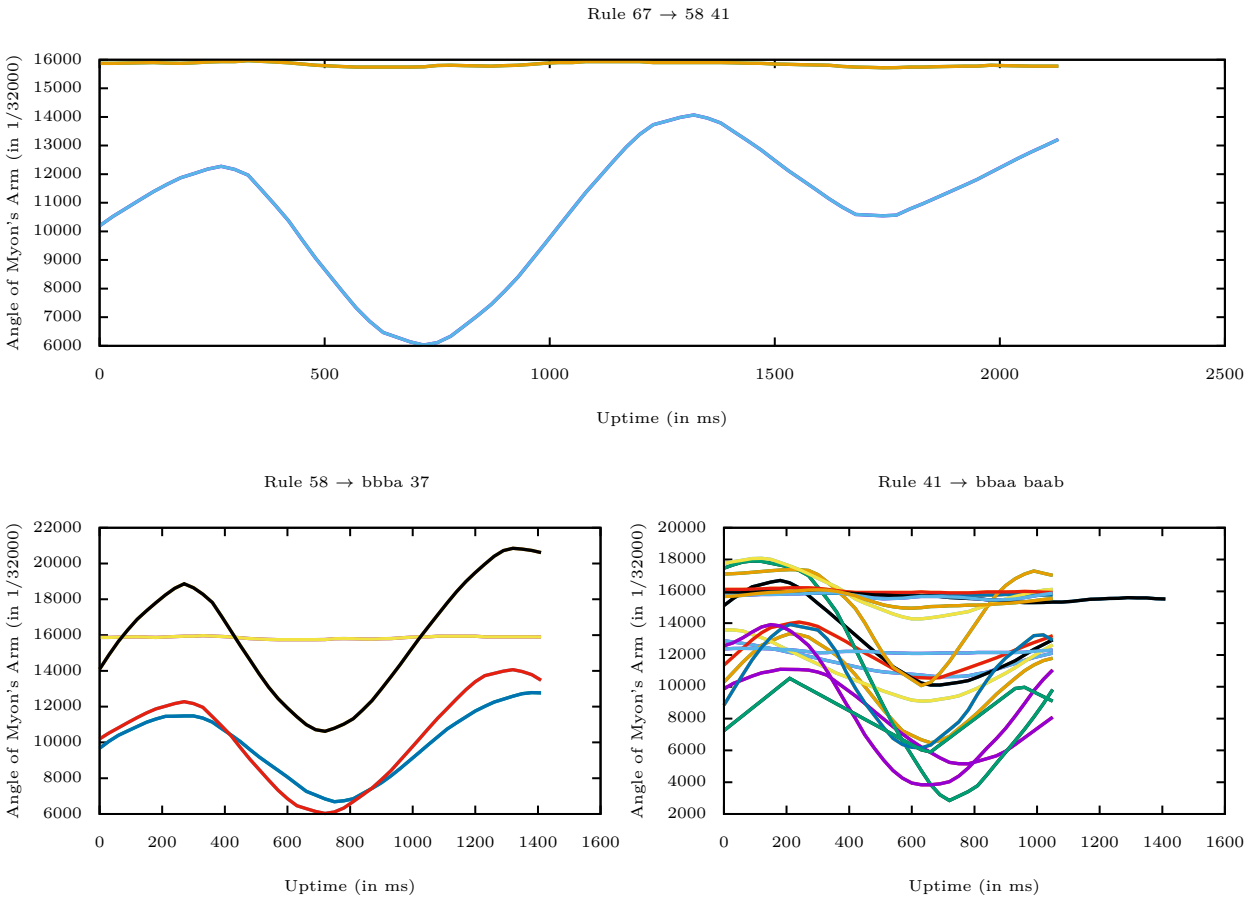

FIGURE 5.9: This plot shows one false positive caused by extreme normalization.

The results of this experiment are very promising since less than 10 percent of our results are false-positives and the other rules' occurrences match well.

5.2.5 Walking Dataset

The last data set in this episode of experiments is a time series which consists of multiple sequences of the robot walking. The curves of walking movements are different from the previously examined time series. The data captured in the left knee of the robot is used as input. This time series with its interesting rules can be seen in Figure 5.10.

This result shows that we are able to find motifs in different kinds of movements. The algorithm only found three high interest motifs, but the interesting motifs contain no false-positives. In fact, the extracted rules match very similar sequences occurring during walking. The parametrization has been adapted to match the new movements,

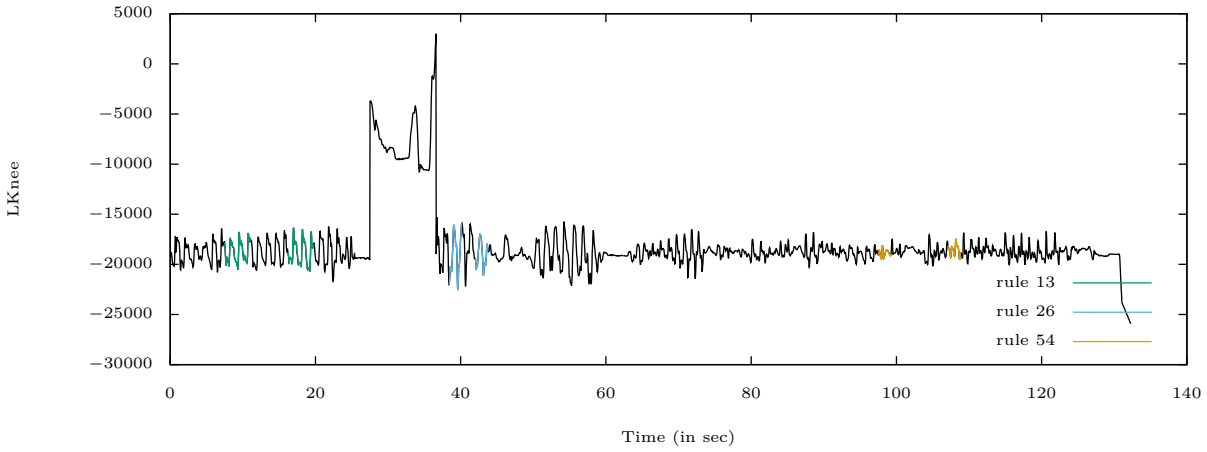

FIGURE 5.10: Time series recorded while Myon has been walking. The occurrences of the detected high interest rules are highlighted.
Parametrization: $a = 2, w = 4, n = 20$.

however, the old parametrization also yielded quite good results. This shows that the parametrization sometimes has to be changed to detect better motifs in different data sources. For motif detection on the robot, this could be done by mapping each limb to its own parametrization.

These experimental results verify most of our expectations. The lengths of time series did not influence the quality of detected interesting rules. The one dimensionality of our input does not pose a problem. On the contrary, the curves of different movement motifs matched into different rules.
The normalization of the input data supports finding good matches most of the time. But normalization can also be an obstacle which can not be resolved easily. Sometimes, the normalization emphasizes sequences which do not really contain movements, leading to false positives. As described above, this problem can be resolved by adjusting the preprocessing.
Good results strongly depend on a good parametrization of the algorithms. This involves the parametrization for SAX as well as choosing an appropriate rule interest threshold. SAX and Sequitur performed well with a sequence length $n \in [20, 24]$, alphabet size $a \in [2, 3]$ and word length $w \in [3, 4]$. This sequence length corresponds to a length between 0.8 and 0.96 seconds.
For the rule interest threshold, the most reliable results were achieved with a threshold higher than 10, leading to very few false positives. Depending on the use case, however, it could still be better to set it lower. For instance, if we want to increase the recall of detected motifs and do not care about false-positives, a low threshold should be picked.

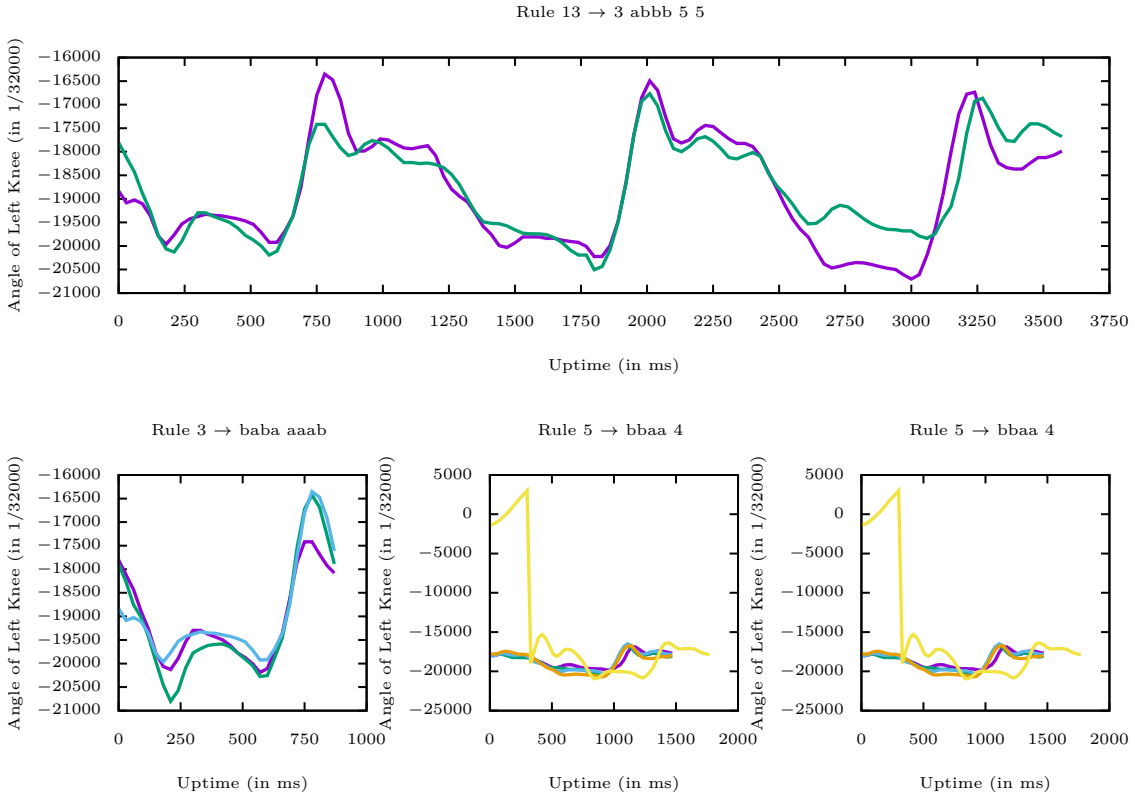

FIGURE 5.11: Motifs detected in time series consisting of multiple walking sequences.

5.3 Performance

To measure the performance of this approach, a sufficiently large dataset was missing. Therefore, two time series were created by combining existing time series together. One *repeated dataset* and another *mixed dataset* has been created. The repeated dataset contains the time series shown in Figure 5.5 repeated 20 times. The mixed dataset also contains that time series 20 times but the small time series plotted in Figure 5.1 is randomly mixed in-between. The algorithms are executed on a MacBook Air with an 1.3GHz Intel Core i5 processor and 8GB RAM.

The conducting time series was chosen for its lack of long pauses containing no movement. These interludes would falsify our results as they always lead to fast execution times and small grammars due to many skipped words. Two datasets are used to validate the hypothesis that a time series consisting of a repeated sequence leads to a fast saturation when Sequitur is executed. The mixed data should lead to a slower saturation because the irregular occurrences of the smaller time series mixed in-between avoid the creation of big grammar rules.

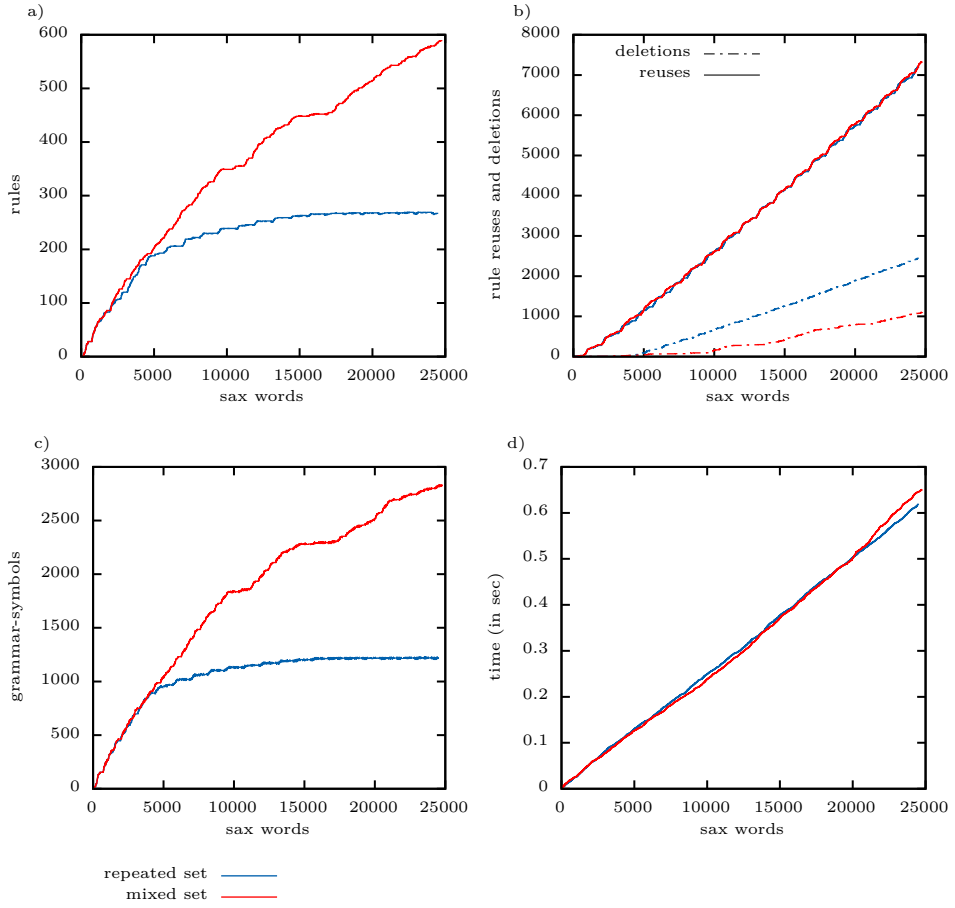

FIGURE 5.12: Performance results for SAX and Sequitur executed offline using two datasets. Both results were achieved with parametrization $a = 2$, $w = 4$, $n = 24$.

In Figure 5.12, we always applied the number of SAX words to the x-axis with different measures at the y-axis. Figure 5.12a shows how many rules were used during the execution of Sequitur. Deleted rules are not taken into account. Figure 5.12b plots the rule-reuses and rule-deletions over the processed symbols. A rule-reuse only applies to symbols being replaced by a non-terminal symbol of a rule which already existed when the digram uniqueness was violated. This means neither of the two substitutions for newly created rules are contained here. Rule-deletions are counted whenever the rule utility constraint of Sequitur is violated. This plot is really interesting as only rule-reusage and rule-deletion reduce the grammar's size. The development of the grammar's size is conducted in plot Figure 5.12c. The size of the grammar is measured in number of grammar symbols contained in the grammar structure. This includes all terminal and non-terminal symbols. The last panel shows the execution time which was measured by comparing the process' CPU time during the execution of the algorithms.

Observing the blue plots corresponding to our *repeated dataset* yields the expected results: at first, the number of grammar rules grows linear while knowledge about the

occurring movements is build up. After about one fifth of the data, which is four repetitions, the grammar is saturated, causing known patterns to collapse into already existing rules. This causes the number of rules to grow extremely slower after about 5000 SAX words and developing further into a constant number of rules after 15000 read words. This has direct impact on the total number of grammar symbols, resulting in a similar development of the curve (see Figure 5.12c). The reasons for the decline in grammar growth are given by the rule-reusage and deletion plot. While the number of rule-reuses is almost linear with a small increase in the slope, the rule-deletions start to significantly rise after the first fifth of data. This causes grammar rules to derive to longer subsequences and causes the observed reduction of the grammar's size. This observation confirms our expectation of a fast rule saturation because the same sequence is repeated multiple times in this dataset. The linear execution time of our approach is confirmed in the last panel. 25000 SAX words have been processed in less than 0.8 seconds. This time measurement includes reading the input data, transforming it to SAX, and the Sequitur processing including the determination of the rule interest for each rule.

The red curves measured using the *mixed dataset* differ in some metrics from the blue graphs. First of all, the plots for the execution time are almost identical. This is to be expected as the input data should not have influence on the algorithm's linear runtime. Yet, the development of the grammar's size and number of rules varies from our *repeated dataset*. When using the mixed data, the saturation only happens locally and then the line falls back into linear growth. Two of these saturations happen around 10000 and 15000 words. Within the unsaturated parts, the slope is linear with a small decrease over time. The reason for this difference can be observed in the rule-reuse and deletions panel. The rule-reuse line of both plots is almost identical, but the rule-deletions occurring while processing our mixed dataset happen less frequent. This is caused by the small different datasets mixed within the repetitions. Every rule-deletion reduces the grammar's symbol count by one. Comparing the rule-deletions yields that the *repeated dataset* has about 1500 more rule-deletions than the *mixed dataset*. This difference can also be found by examining the discrepancy between the plots for the number of symbols.

All in all, this evaluation shows that our approach is very quick while yielding good results across different datasets. The saturation of the Sequitur grammar is not as strong as we hoped for but the potential for this algorithm can still be observed in the repeated dataset.

Chapter 6

Online Motif Detection

Using the knowledge achieved from our experimental results, the online motif detection is build for Myon.[1] This chapter covers decisions made to implement motif detection online. This involves data processing and transformation as already introduced in chapter 3. The approach is extended to enable the robot to extract motifs from the grammar. At last, two methods of learning and training motifs are introduced.

6.1 SAX and Sequitur

The heart of this implementation is SAX and Sequitur. For these algorithms, the offline implementation is taken as it proved to achieve good results. The biggest difference between the offline and online implementation of our approach is the data acquisition and preprocessing.

It is complicated and impracticable to read the recorded data while executing the algorithms directly on the robot. Therefore, a better solution to collect and process the data is used. The structure of the robot's implementation is important for this task. While the robot is running, the main procedure is running in a loop. But, many tasks are executed during interrupts, suspending the main process. Different interrupts exist for several events. There is one spinal cord interrupt which is called whenever the data of the robot's spinal cord is updated. This spinal cord data is the same data written into the recorded datasets. Therefore, this interrupt provides a perfect entry point to introduce our algorithms into the existing Myon codebase. This interrupt has the advantage that it is called at a rate of 100 HZ. Thus, we do not need to rescale the data to ensure that each pair of neighboring datapoints has the same interval.

This interrupt is extended to call our motif detection procedure. This procedure reads

[1] The implementation is stored in the SVN of the Neurorobotics Research Laboratory.

the next value and appends it to a sequence array of length n. Whenever this array is full, this data is preprocessed and fed to our SAX and Sequitur algorithm. The latter half of the sequence array is then shifted to the beginning as it contains the first half of the next sliding window's data. For the sequence length, a good outcome has been observed during the experiments with values between 20 and 24. For good online results we have to adjust this parameter to fit the new input data interval. This can be accomplished by rescaling the interval to sequence length $[80, 96]$. All other parameters remain the same leading to the parametrization $a = 2$, $w = 4$, $n = 92$.
The data is still normalized and set to zero for too low standard derivations during procession. The threshold for the standard derivation is set to 70. This value has been chosen by trying to hold the robot's arm still while observing the standard derivation. Then, a threshold value close to the highest observed values has been chosen.
No further changes have been made to either data preprocessing, SAX or Sequitur.

6.2 Motif extraction

So far, the Sequitur grammar has been used to highlight specific rules exceeding our rule interest threshold. The resulting grammar rules are equivalent to the previously defined matches. These rule matches share the following properties controlled by parametrization:

- Its occurrences' lengths are lower bounded by the length of a digram $n + \frac{n}{2}$ with sequence size n and window size $\frac{n}{2}$.
- The maximum size of a rule occurrence is only limited by the input size.
- Two matching sequences can overlap in at most $\frac{n}{2}$ points.

The rule utility ensures that every grammar rule matches at least two sequences sharing the same SAX transformation. Evaluation showed, that even for simple datasets, a single rule will most likely not match all occurrences of one motif. Therefore, it is not sufficient to limit motifs to sequences matching the same interesting grammar rules.
For this reason, annotations are introduced, enabling terminal and non-terminal grammar symbols to be annotated. These annotations currently consist of numbers identifying different motifs. Whenever two symbols are annotated with the same number, they belong to the same motif. This idea is inspired by the results presented in Figure 5.7 where multiple rules were combined into sets by drawing a Venn diagram.
The Sequitur algorithm was adjusted enabling rules to inherit annotations from their

descendants. A rule matches a motif if both guards share the same annotation. Whenever a substitution happens during the execution of Sequitur, the resulting non-terminal is annotated if the referenced rule matches a motif. This builds up knowledge about motifs bottom-up and ensures that a motif will spread whenever a rule matching a motif is reused.

TABLE 6.1: Example illustrating the inheritance of grammar annotations. The subscripted texts are the motifs annotated to the grammar symbol.

	input string	sequitur grammar
(0)	ab ba	$0 \rightarrow \text{ab}_{\text{M1}}\ \text{ba}_{\text{M1}}$
(1)	ab ba ab ba	$0 \rightarrow 1_{\text{M1}}\ 1_{\text{M1}}$ $1 \rightarrow \text{ab}_{\text{M1}}\ \text{ba}_{\text{M1}}$
(2)	ab ba ab ba ab ba ab ba	$0 \rightarrow 2_{\text{M1}}\ 2_{\text{M1}}$ $2 \rightarrow 1_{\text{M1}}\ 1_{\text{M1}}$ $1 \rightarrow \text{ab}_{\text{M1}}\ \text{ba}_{\text{M1}}$
(3)	ab ba ab ba ab ba ab ba ab ba bb ba	$0 \rightarrow 2_{\text{M1}}\ 2_{\text{M1}}\ 1_{\text{M1}}$ bb ba $2 \rightarrow 1_{\text{M1}}\ 1_{\text{M1}}$ $1 \rightarrow \text{ab}_{\text{M1}}\ \text{ba}_{\text{M1}}$
(4)	ab ba ab ba ab ba ab ba ab ba bb ba ab ba bb ba	$0 \rightarrow 2_{\text{M1}}\ 2_{\text{M1}}\ 3$ $3 \rightarrow 1_{\text{M1}}$ bb ba $2 \rightarrow 1_{\text{M1}}\ 1_{\text{M1}}$ $1 \rightarrow \text{ab}_{\text{M1}}\ \text{ba}_{\text{M1}}$

A small example illustrates this behavior in Table 6.1. The grammar starts with one digram **ab ba** which is annotated with motif $M1$. In the second line, the digram is repeated resulting in the first rule. Both non-terminals of this rule inherit the annotation of the first digram because both annotated symbols of the digram guard the rule. In the next line, this motif grows to eight repetitions, leading to another rule inheriting the annotation. In the last two lines, another rule is created different from the motif $M1$. Rule 3 does not inherit the annotation because only one of its guards is annotated.
These annotations make it possible to mark motifs in the Sequitur grammar independently from its rules. Reoccurring motifs can be detected and extracted while they occur by combining the rule interest with annotations of grammar symbols. Whenever a rule is used and its interest exceeds the threshold, the guards of the rule are checked for their annotations. If the annotations match the same motif, an event is appended to the robot's event queue indicating that a motif has been detected.

6.3 Learning Motifs

Thus far, the idea of annotations and how they spread has been covered. Initially, there are no symbols in the grammar and thus, neither are there annotations. The problem of deciding when to introduce new annotations will be addressed in this section. Two approaches, which can also be combined, will be presented.
Both techniques are limited by the robot's currently existing means of communication: the robot is able to say a limited amount of words and numbers. Users can communicate with the robot by using either barcodes or beeper. The beeper is a small device creating sequences of beep-sounds in different frequencies. These are then converted to a number depending on the sequence.
Both of the following methods will be using barcodes to enable the user to communicate with the robot while Myon uses text-to-speech to answer and formulate its demands. Verbs such as signaling, communicating or answering are used interchangeably to describe the user-robot communication with barcodes.

6.3.1 Curiosity Driven Learning

The first learning strategy is called Curiosity Driven Learning. The underlying idea is that the robot should be curious about its environment. This approach is autonomously triggered by Myon. Whenever a reoccurring movement builds up enough curiosity, it grabs the robot's attention resulting in a request for the motif of this movement.
This is done by listening for motif events in the robot's event queue. There are two possible event types: the first is triggered when an interesting rule matching a known motif occurs, the second event signals that an interesting rule without annotation has been found. The robot will alert these events by either saying the number of the detected motif or by saying the word *erkannt* (engl. detected) to inform about yet unknown movements.
Both words initiate the feedback loop expecting a response from an user. The feedback loop works the same for either event type. A response time of 10 seconds is given for a response to the robot's request for feedback. While the robot is waiting for a response, it will not enter another feedback loop. Thus, concurrent interesting rules are ignored. If no response is given within the response time, the current annotations of the grammar's symbols remain unchanged.
Similar to the two event types, there exist two possible responses: at first, the user can signal that no motif occurred and should be discarded. Secondly, a reserved range of barcodes is used to indicate different motifs.
If the robot is told to discard the last motif, all annotations of the grammar symbols

which belong to the matched rule are removed. This includes the non-terminal symbols as well as the guards of this rule. If the robot did not detect a known motif and is told to discard the last motif, it ends the feedback loop but does not change any annotations. When a motif is identified by the user, the robot adds these annotations to the corresponding grammar symbols.

6.3.2 Human-triggered Learning

This method is initiated by the human user and introduces motifs to the robot by teacher-child interactions. The user initiates the learning process by signaling Myon which motif is going to be taught. This sets Myon into a recording state during which it annotates all newly created grammar symbols with the previously shown motif id. The robot indicates entering this state by saying the word *Klappe* (a German synonym for clapperboard). This recording state lasts until the user tells the robot to stop. During this phase, the Curiosity Driven Learning is deactivated.
This way of teaching motifs has the advantage of being independent of the rule interest. Thus, small rules and even terminal symbols which occur only in the start rule are annotated. This improves the chance of a motif to spread through the grammar by the mechanics illustrated in Table 6.1. On the one hand, this method provides faster results in comparison to the Curiosity Driven Learning because less communication is required. On the other hand, it yields a higher potential to produce false positives.

6.3.3 Combined Effects

Both learning techniques are implemented and active at the same time. they yield the best results when combined. The Human-triggered Learning is well suited to introduce new motifs and quickly generate a baseline of annotated grammar symbols. The Curiosity Driven Learning can then be used to assert correct annotation for the introduced motifs and to make corrections if necessary.
To verify the taught motifs, the user can just perform them again, once the recording phase of the Human-triggered Learning has ended. If the motifs are annotated correctly, the Curiosity Driven Learning method will cause the robot to signal a detected motif and enter the feedback loop. In case the recognized motif is correct, the user is assured that Myon learned the introduced movement. Otherwise, the feedback loop offers the possibility of an immediate correction.

Chapter 7

Summary and Future Work

The results of this work offer a lot of possible extensions and future work. This chapter gives ideas for further improvements and raises unresolved issues. Finally, the results of this thesis are summarized.

7.1 Remaining Problems

Most obstacles we stumpled upon could be resolved by small changes to our approach. However, some issues still remain in the current implementation. This section covers problems which were raised during implementation of the presented approach or discovered by testing the motif detection on the robot.

7.1.1 Usability

Teaching motifs to Myon involves human-robot interaction. In this work we used the existing means of communication to realize this interaction. Aside from manipulating the robot's limbs, giving input to Myon involves heavy use of barcodes. This results in a lot of breaks between the performance of a motif, especially when using the Curiosity Driven Learning which involves a lot of communication. Using barcodes is problematic because the user has to stop the current motion to pick up the correct barcode and wait until the robot scanned it. Therefore, the human-robot interaction should be replaced by a more intuitive communication channel like speech recognition as soon as it is available. Speech recognition also has the advantage of having extendable grammar annotations to contain wordst or phrases instead of numbers for each motif.

This problem can be avoided by using the Human-triggered Learning which only involves signaling the beginning and end of a motif. We particularly advice this teaching technique for long and complex motifs.

7.1.2 Storage Management

The evaluation of Sequitur showed that it is bounded by linear space requirements. However, this requirement is already reduced by skipping repeated grammar symbols and ignoring sequences with little movement. The size of the grammar structure still depends on the data and can grow linear. The storage on the robot is limited and will be full at some point. In this case, the current implementation will stop Sequitur and requires a data reset. This problem has to be addressed before extending this prototype implementation to multiple limbs and other data sources. Increasing the data size for our grammar postpones the problem instead of solving it. Solving this challenge requires a storage management to rearrange the grammar.
One possible mechanism to resolve this issue could be forgetting uninteresting grammar symbols and rules similar to a cache. The algorithm could be extended by a forgetting phase which is triggered whenever the storage is almost full. One possible way to clear the storage is to delete all grammar symbols and rules except the interesting rules and their descendants. Afterwards, the amount of fragmentation should be reduced by a defragmentation step to free the storage.

7.2 Further Work

This work showed that movement motifs can efficiently be detected by transforming time series to hierarchical grammar structures. There are however some more possibilities to extend the algorithm.
Currently, the motif detection is limited to one limb of the robot. There could be more Sequitur grammars for each additional limb. Without further adaption of the motif detection algorithm, Myon should be able to detect motifs in each of the resulting grammars online. Furthermore, this can be combined with multi- and subdimensional motif detection [19, 20]. Subdimensional motif detection would enable the robot to find motifs which always occur across specific dimensions. It could differentiate between conducting with one arm and conducting with both arms for instance. Although the existing methods for subdimensional motif detection use the PROJECTION algorithm, the basic idea should be able to be translated to grammar-based motif discovery.
The approach presented in this thesis can also be used for more domains than movement motifs. We could add input data from the robot's camera to detect reoccurring shapes [2]

or process sound data with SAX and Sequitur. Combining this with the subdimensional motif discovery would enable Myon to detect reoccurring movements and connect these with its acoustic and visual perceptions. For instance, it could recognize a conducting movement while listening to a known sequence of sound.

7.3 Conclusion

During this thesis, we presented and explained an approach to find motifs of arbitrary length in time series. These methods transform a time series into its discrete SAX representation, achieving dimensionality reduction. The resulting SAX words are used as input for Sequitur to build and maintain a context-free, hierarchical grammar structure. A software was implemented which builds Sequitur grammars from recorded robot data. The program creates files - which can be plotted with gnuplot - visualizing the most interesting grammar rules in a given time series. The rules are extracted by computing a rule interest for each rule, and then selecting all rules above a rule interest threshold. Experiments validated that this approach is well fitted to detect motifs in a robot's motion data. We were able to match several motifs and received few false-positives. The linear execution time could be verified and a strong decline of grammar growth was observed in data repeating the same sequences, while data with different movements and resulted in linear space requirements. The implementation was adapted to be executed in real-time, directly on the robot, and introduced to the robot's codebase. Finally, the novel idea of rule annotations was introduced to enable Myon to match grammar symbols to motifs. Based on these annotations, two methods of teaching motifs involving human-robot interaction were presented, enabling the user to teach, label, and discard motifs.

Appendix A

Grammars

65 → abba 41

62 → 59 38

61 → **56** 16

59 → 38 36

58 → 65 42

56 → **52** 16

55 → 43 46

53 → 38 38

52 → 46 44

46 → 44 16

44 → 16 16

43 → 20 aabb

42 → baab 38

41 → aaab abba

38 → 36 36

36 → abba baab

33 → 28 16 19

28 → 21 bbba

21 → bbbb aabb

20 → aabb bbaa

19 → abbb bbbb

16 → bbaa aabb

TABLE A.1: Excerpt from grammar consisting of all interesting rules and their descendant rules.
The most interesting rules are bold.

Appendix B

Interesting Grammar Rules

B.1 Conducting Dataset

FIGURE B.1: This plot shows all occurrences of the rule set S_1. This set contains four rules. Each rule's occurrences are plotted into a single panel. The x-axis contains the uptime in milliseconds. The y-axis corresponds to the angle of Myon's left shoulder.

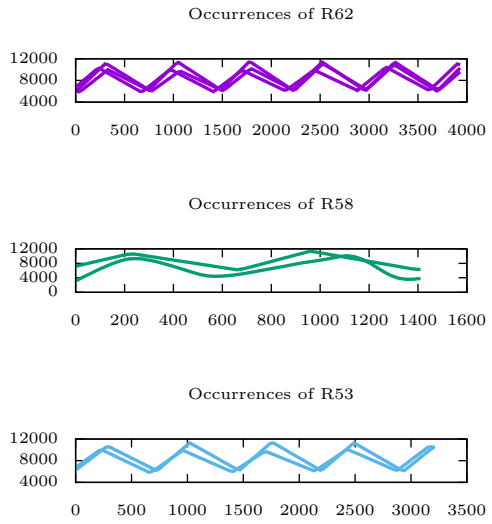

FIGURE B.2: This plot shows all occurrences of the rule set S_2. This set contains three rules. Each rule's occurrences are plotted into a single panel. The x-axis contains the Uptime in milliseconds. The y-axis corresponds to the angle of Myon's left Shoulder.

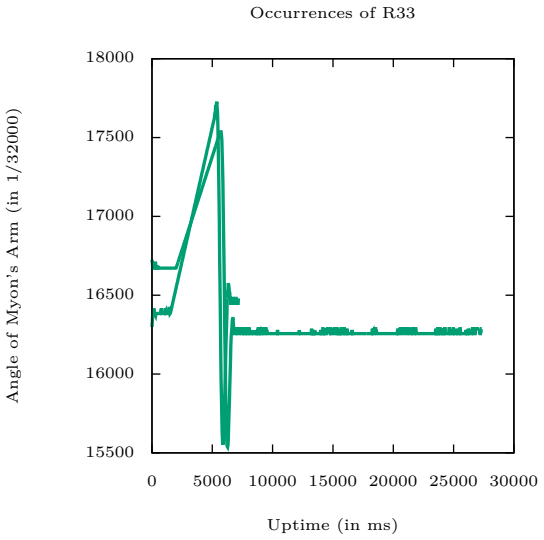

FIGURE B.3: This plot shows the rule in the rule set S_2 which contains a single rule.

B.2 Combined Dataset

All plots show interesting rules resulting from the execution of Sequitur using the combined data set.

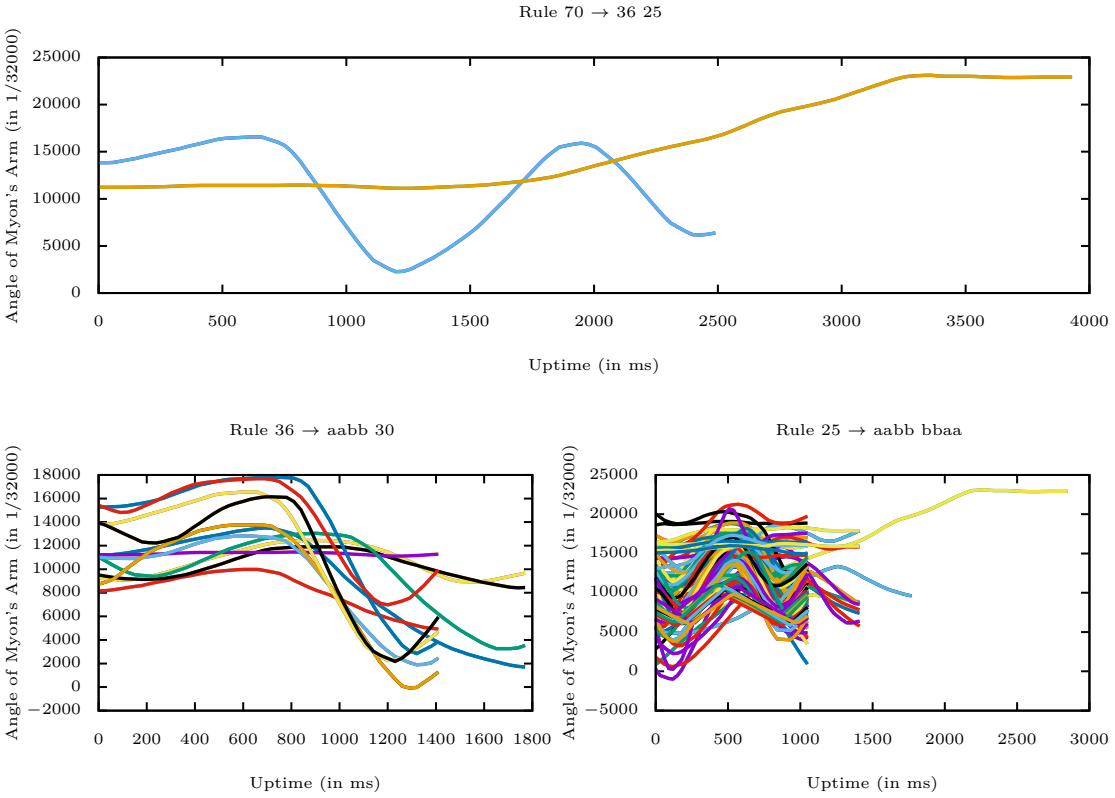

FIGURE B.4: This plot shows a false positive rule resulting from the combined data set. It is caused by the "stretch-effect" and normalization.

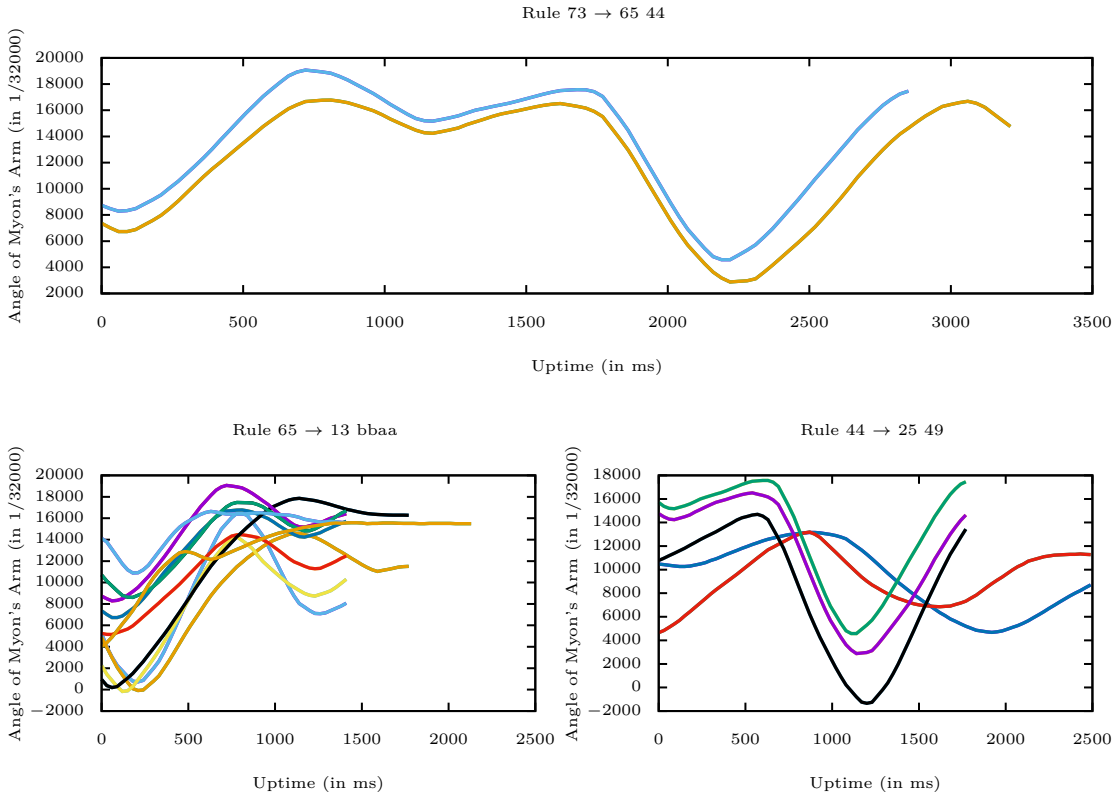

FIGURE B.5: Rule occurrences of rule 73. This rule was detected by our approach as interesting rule and matches two very similar sequences.
Parametrization: $a = 2, w = 4, n = 24$.

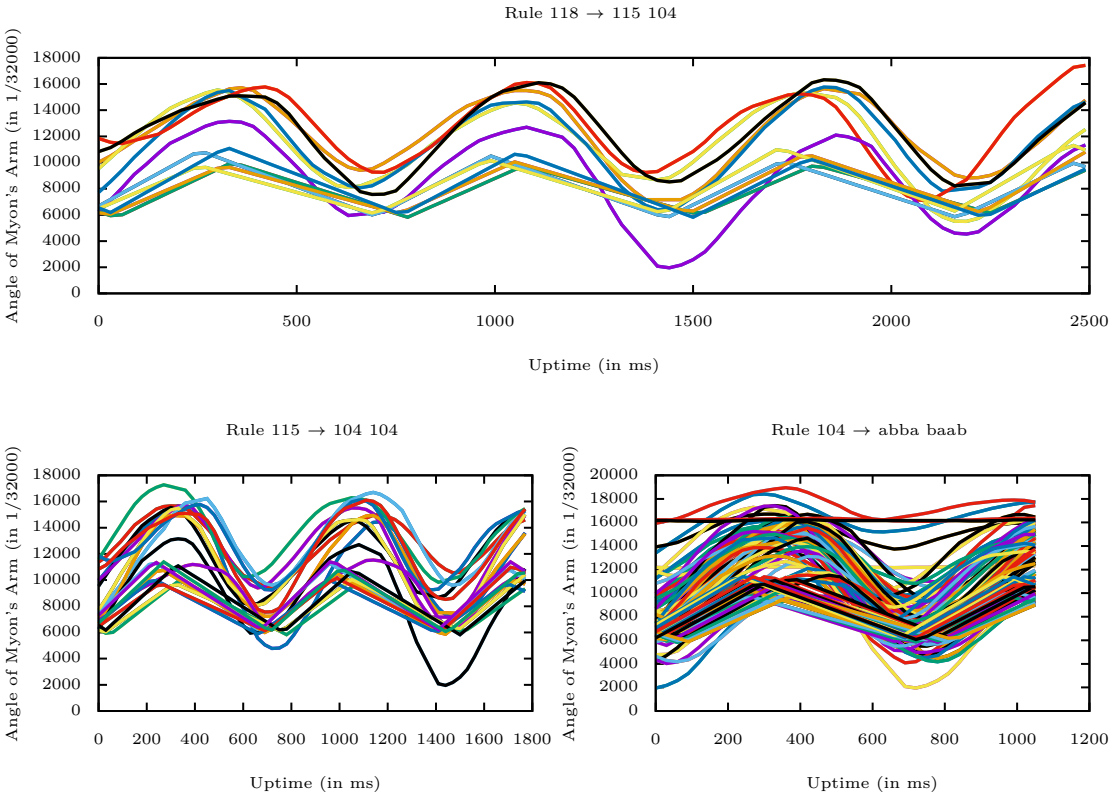

FIGURE B.6: Rule 118 matches many sequences occurring in different positions across different time-series. Parametrization: $a = 2, w = 4, n = 24$.

Bibliography

[1] Manfred Hild, Torsten Siedel, Christian Benckendorff, Matthias Kubisch, and Christian Thiele. Myon: Concepts and Design of a Modular Humanoid Robot Which Can Be Reassembled During Runtime. In Proceedings of the 14th International Conference on Climbing and Walking Robots and the Support Technologies for Mobile Machines, Paris, France, September 2011.

[2] Dragomir Yankov, Eamonn Keogh, Jose Medina, Bill Chiu, and Victor Zordan. Detecting time series motifs under uniform scaling. In Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '07, pages 844–853, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-609-7. doi: 10.1145/1281192.1281282. URL http://doi.acm.org/10.1145/1281192.1281282.

[3] Rakesh Agrawal, King-Ip Lin, Harpreet S. Sawhney, and Kyuseok Shim. Fast similarity search in the presence of noise, scaling, and translation in time-series databases. In Proceedings of the 21th International Conference on Very Large Data Bases, VLDB '95, pages 490–501, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc. ISBN 1-55860-379-4. URL http://dl.acm.org/citation.cfm?id=645921.673155.

[4] Byoung-Kee Yi, H. V. Jagadish, and Christos Faloutsos. Efficient retrieval of similar time sequences under time warping. In Proceedings of the Fourteenth International Conference on Data Engineering, ICDE '98, pages 201–208, Washington, DC, USA, 1998. IEEE Computer Society. ISBN 0-8186-8289-2. URL http://dl.acm.org/citation.cfm?id=645483.653609.

[5] Xianping Ge and Padhraic Smyth. Deformable markov model templates for time-series pattern matching. In Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '00, pages 81–90, New York, NY, USA, 2000. ACM. ISBN 1-58113-233-6. doi: 10.1145/347090.347109. URL http://doi.acm.org/10.1145/347090.347109.

[6] Eamonn Keogh, Jessica Lin, and Ada Fu. Hot sax: Efficiently finding the most unusual time series subsequence. In Proceedings of the Fifth IEEE International Conference on Data Mining, ICDM '05, pages 226–233, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7695-2278-5. doi: 10.1109/ICDM.2005.79. URL http://dx.doi.org/10.1109/ICDM.2005.79.

[7] Bill Chiu, Eamonn Keogh, and Stefano Lonardi. Probabilistic discovery of time series motifs. In Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '03, pages 493–498, New York, NY, USA, 2003. ACM. ISBN 1-58113-737-0. doi: 10.1145/956750.956808. URL http://doi.acm.org/10.1145/956750.956808.

[8] Pavel Senin, Jessica Lin, Xing Wang, Tim Oates, Sunil Gandhi, Arnold P. Boedihardjo, Crystal Chen, and Susan Frankenstein. Time series anomaly discovery with grammar-based compression. In Proceedings of the 18th International Conference on Extending Database Technology, EDBT 2015, Brussels, Belgium, March 23-27, 2015., pages 481–492, 2015. doi: 10.5441/002/edbt.2015.42. URL http://dx.doi.org/10.5441/002/edbt.2015.42.

[9] Yuan Li, Jessica Lin, and Tim Oates. Visualizing variable-length time series motifs. In Proceedings of the Twelfth SIAM International Conference on Data Mining, Anaheim, California, USA, April 26-28, 2012., pages 895–906, 2012. doi: 10.1137/1.9781611972825.77. URL http://dx.doi.org/10.1137/1.9781611972825.77.

[10] Abdullah Mueen, Eamonn J. Keogh, Qiang Zhu, Sydney Cash, and M. Brandon Westover. Exact discovery of time series motifs. In Proceedings of the SIAM International Conference on Data Mining, SDM 2009, April 30 - May 2, 2009, Sparks, Nevada, USA, pages 473–484, 2009. doi: 10.1137/1.9781611972795.41. URL http://dx.doi.org/10.1137/1.9781611972795.41.

[11] Jessica Lin, Eamonn Keogh, Stefano Lonardi, and Pranav Patel. Finding motifs in time series. In Proceedings of the Second Workshop on Temporal Data Mining, Edmonton, Alberta, Canada, July 2002. URL http://citeseer.ist.psu.edu/lin02finding.html.

[12] Jeremy Buhler and Martin Tompa. Finding motifs using random projections. In Proceedings of the Fifth Annual International Conference on Computational Biology, RECOMB '01, pages 69–76, New York, NY, USA, 2001. ACM. ISBN 1-58113-353-7. doi: 10.1145/369133.369172. URL http://doi.acm.org/10.1145/369133.369172.

[13] Pavel A. Pevzner and Sing-Hoi Sze. Combinatorial approaches to finding subtle signals in dna sequences. In Proceedings of the Eighth International Conference on

Intelligent Systems for Molecular Biology, pages 269–278. AAAI Press, 2000. ISBN 1-57735-115-0. URL http://dl.acm.org/citation.cfm?id=645635.660985.

[14] Craig G. Nevill-Manning and Ian H. Witten. Identifying hierarchical structure in sequences: A linear-time algorithm. J. Artif. Int. Res., 7(1):67–82, September 1997. ISSN 1076-9757. URL http://dl.acm.org/citation.cfm?id=1622776.1622780.

[15] Eamonn Keogh, Stefano Lonardi, and Chotirat Ann Ratanamahatana. Towards parameter-free data mining. In Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '04, pages 206–215, New York, NY, USA, 2004. ACM. ISBN 1-58113-888-1. doi: 10.1145/1014052.1014077. URL http://doi.acm.org/10.1145/1014052.1014077.

[16] Jessica Lin, Eamonn Keogh, Stefano Lonardi, and Bill Chiu. A symbolic representation of time series, with implications for streaming algorithms. In Proceedings of the 8th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, DMKD '03, pages 2–11, New York, NY, USA, 2003. ACM. doi: 10.1145/882082.882086. URL http://doi.acm.org/10.1145/882082.882086.

[17] Haemwaan Sivaraks and Chotirat Ann Ratanamahatana. Robust and accurate anomaly detection in ecg artifacts using time series motif discovery. Comp. Math. Methods in Medicine, 2015:453214:1–453214:20, 2015. URL http://dblp.uni-trier.de/db/journals/cmmm/cmmm2015.html#SivaraksR15.

[18] Arvind Balasubramanian, Jun Wang, and B Prabhakaran. Multidimensional motif discovery in physiological and biomedical time series data, 2013.

[19] D. Minnen, C. Isbell, I. Essa, and T. Starner. Detecting subdimensional motifs: An efficient algorithm for generalized multivariate pattern discovery. In Data Mining, 2007. ICDM 2007. Seventh IEEE International Conference on, pages 601–606, Oct 2007. doi: 10.1109/ICDM.2007.52.

[20] Alireza Vahdatpour, Navid Amini, and Majid Sarrafzadeh. Toward unsupervised activity discovery using multi-dimensional motif detection in time series. In Proceedings of the 21st International Jont Conference on Artifical Intelligence, IJCAI'09, pages 1261–1266, San Francisco, CA, USA, 2009. Morgan Kaufmann Publishers Inc. URL http://dl.acm.org/citation.cfm?id=1661445.1661647.