Trajectory Based Upper Body Gesture Recognition for an Assistive Robot

Gestenerkennung von Oberkörper-Trajektorien für einen Assistenzroboter
Bachelor-Thesis von Felix Divo aus Schwäbisch Hall
November 2019

TECHNISCHE UNIVERSITÄT DARMSTADT

Trajectory Based Upper Body Gesture Recognition for an Assistive Robot
Gestenerkennung von Oberkörper-Trajektorien für einen Assistenzroboter

Vorgelegte Bachelor-Thesis von Felix Divo aus Schwäbisch Hall

1. Gutachten: Prof. Dr. Jan Peters
2. Gutachten: Dorothea Koert, M. Sc.

Tag der Einreichung:

Erklärung zur Bachelor-Thesis

Erklärung zur Abschlussarbeit gemäß § 23 Abs. 7 APB der TU Darmstadt

Hiermit versichere ich, Felix Divo, die vorliegende Bachelor-Thesis ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Fall eines Plagiats (§ 38 Abs. 2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei der abgegebenen Thesis stimmen die schriftliche und die zur Archivierung eingereichte elektronische Fassung überein.

I herewith formally declare that I have written the submitted thesis independently. I did not use any outside support except for the quoted literature and other sources mentioned in the paper. I clearly marked and separately listed all of the literature and all of the other sources which I employed when producing this academic work, either literally or in content. This thesis has not been handed in or published before in the same or similar form.

In the submitted thesis the written copies and the electronic version are identical in content.

Datum / Date: ______________________ Unterschrift / Signature: ______________________

Abstract

The aging of the European and German population poses multiple challenges, in particular, the nursing sector potentially faces an increasing lack of caregivers to provide appropriate care for elderly people. Future robots might be able to assist in these scenarios by taking over repetitive or physical tasks of caregivers or by providing general assistance allowing seniors to longer stay at their own homes. A crucial task for such an assistive robotic system is the intuitive interaction between a robot and potentially multiple humans. Gestures, especially of the upper body and hand, are hereby a promising mean of communication.
This thesis will thus investigate how human gestures can be classified using machine learning techniques. The input data will be represented by sequences of upper body joint positions obtained from existing skeleton tracking software. Special focus will be given to Probabilistic Movement Primitives (ProMPs), as they have many desirable properties but have not yet been systematically evaluated for gesture classification tasks. As part of the evaluation, different preprocessing techniques such as Dynamic Time Warping and Local Optimization are investigated. Dimensionality reduction, using PCR and UMAP, is evaluated as both an alternative and an addition to the weight space transformation of ProMPs. For classification two methods, namely Gaussian Mixture models (GMMs) and Support Vector Machines (SVMs), are considered in offline and online settings.
The methods are evaluated on a newly recorded data set with over 1000 demonstrations of kitchen tasks as well as the widely adopted MSR Action3D data set. The results show that ProMPs can successfully classify gestures, and are able to compete with the SVMs used in this evaluation. The choice of the preprocessing methods is not able to improve on these results, and might even harm the classification score in some cases. However, both ProMPs and SVMs have numerical limitations when performing online learning and generally perform poorly on the higher-dimensional MSR Action3D data set. This motivates a reduction down to the classification of hand gestures, as these only consist of three spatial dimensions. Dimensionality reduction techniques were able to find suitable embeddings for the kitchen data set, but could not improve results on the MSR Action3D data set. Summing up, ProMPs can perform basic gesture classification, but further work is needed to improve the classification of more challenging actions.

Zusammenfassung

Das Altern der europäischen und deutschen Bevölkerung stellt viele Herausforderungen dar, und insbesondere im Pflegesektor könnte die angemessene Versorgung von Senior*innen vor dem Hintergrund eines sich abzeichnenden Personalmangels schwierig werden. Zukünftige Roboter könnten hier Pflegekräften helfen oder älteren Bürger*innen erlauben, länger in der eigenen Wohnung zu leben. Bei diesem Assistenzsystem ist die intuitive Interaktion zwischen dem Roboter und potentiell mehreren Menschen von zentraler Bedeutung. Gesten, insbesondere von Händen und dem Oberkörper, sind dabei ein vielversprechendes Kommunikationsmedium.
Diese Arbeit wird verschiedene Methoden des Maschinellen Lernens nutzen, um Gesten zu klassifizieren. Die Eingabedaten sind dabei Folgen der Positionen von Oberkörpergelenken, welche mithilfe vorhandener Skeleton-Tracker aus Videodaten extrahiert werden können. Besonderer Fokus liegt aufgrund ihrer nützlichen Eigenschaften auf Probabilistic Movement Primitives (ProMPs), welche noch nicht systematisch für Klassifikationsaufgaben getestet wurden. Als Teil der Evaluierung werden ebenfalls verschiedene Vorverarbeitungsschritte wie Dynamic Time Warping und Local Optimization verglichen. Sowohl als Alternative als auch als Zusatz zur Featuretransformation von ProMPs wird die Dimensionsreduktion mit PCA und UMAP betrachtet. Die Klassifikation selbst wird dann sowohl offline als auch online durchgeführt, und mit Gauss'schen Mischverteilungen (GMMs) und Support Vector Machines (SVMs) realisiert.
Die Methoden werden auf einem eigens aufgenommenen Datensatz von über 1000 Gestenausführungen aus einem Küchenszenario, sowie dem weit verbreiteten MSR Action3D Datensatz getestet. Die Ergebnisse zeigen, dass ProMPs erfolgreich Gesten klassifizieren können und dabei mit SVMs Schritt halten. Die gewählten Vorverarbeitungsschritte verbessern das Ergebnis dabei nur in wenigen Fällen und verschlechtern es sogar teilweise. Sowohl GMMs als auch SVMs erreichen beim Online-Lernen numerische Grenzen und liefern auf dem höherdimensionalen MSR Action3D Datensatz generell schlechtere Ergebnisse. Dies motiviert eine Einschränkung auf Handgesten, welche nur aus drei Raumdimensionen bestehen. Methoden zu Dimensionsreduktion waren im Küchenszenario erfolgreich, aber konnten die Ergebnisse auf dem MSR Action3D Datensatz nicht verbessern. Schlussendlich lässt sich folgern, dass ProMPs erfolgreich Gesten klassifizieren können, aber weitere Arbeit an der Verarbeitung komplexerer Gesten notwendig ist.

Acknowledgments

I want to warmly thank my supervisor Dorothea Koert for guiding me through this thesis. She helped me with all questions I had, while making sure that I completed the project myself. I further appreciate the weekly talks and discussions of the IAS group around and with Prof. Jan Peters, as they often inspired new approaches and views on topics being discussed.

I also want to thank the participants of the gesture recording sessions for their time, patience und at times insightful discussions.

Last but not the least, I would like to thank my family and friends who helped me throughout this semester.

Contents

Figures and Tables

List of Figures

List of Tables

List of Algorithms

Abbreviations, Symbols and Operators

List of Abbreviations

Notation	Description
AAL	active and assisted living
ADL	activities of daily living
AE-ProMP	ProMP using autoencoders, cf. *ProMP*
BFGS	Broyden-Fletcher-Goldfarb-Shanno optimization algorithm
COTS	commercial off-the-shelf
CV	cross validation
DBA	DTW barycenter averaging, cf. *DTW*
DP	dynamic programming
DR	dimensionality reduction
DTW	dynamic time warping
f.a.	for all
GMM	gaussian mixture model
HCI	human-computer interaction
IAS	*Intelligent Autonomous Systems Group* of the *Technische Universität Darmstadt*
kF-CS-CV	k-fold cross-subject cross validation
L-BFGS-B	limited-memory bound-constrained BFGS algorithm, cf. *BFGS*
LOO-CV	leave-one-out cross validation
MAE	mean absolute error
ML	machine learning

MP	movement primitive
MSE	mean squared error
MSR Action3D	gesture data set by Microsoft Research (MSR), also called *MSRA3D*
MSRA3D	cf. *MSR Action3D*
PCA	principal component analysis
PCR	principal component regression
PDF	probability density function
ProMP	probabilistic movement primitive
RBF	radial basis function
RGB	red, green, blue
RGB-D	red, green, blue, depth
SGD	stochastic gradient descent
s.t.	subject to
SVM	support vector machine
UMAP	uniform manifold approximation and projection
w.r.t.	with respect to

List of Symbols

Notation	**Description**
$\mathbf{\Sigma}$	the covariance matrix of some random variable
e	Euler's number, approx. 2.71828
$\mathbb{I}_{N\times N}, \mathbb{I}$	the $N \times N$ identity matrix; the dimensions may be omitted in cases where it is clear from the context
μ	the mean scalar of some univariate random variable
$\boldsymbol{\mu}$	the mean vector of some multivariate random variable
$\mathbf{\Phi}$	a RBF matrix, e.g. of a ProMP
T	the number of time steps in a trajectory

D	the number of trajectories for some label
$\boldsymbol{\tau}$	a single trajectory, i.e. a series of joint positions
$\boldsymbol{w}$	a weight vector of a ProMP

List of Operators

Notation	Description	Operator
Cov	the covariance matrix	$\mathrm{Cov}(\bullet)$
det	the determinant of a matrix	$\det(\bullet)$
exp	the exponential function	$\exp(\bullet)$
$\bar{\cdot}$	the (sample) mean of a random variable	$\bar{\bullet}$
ln	the natural logarithm	$\ln(\bullet)$
Var	the variance of some random variable; in the case of the sample variance it is unbiased	$\mathrm{Var}(\bullet)$

1 Introduction

> Notably, the share of those aged 15–64 is projected to decline from 67% to 56% by 2060. The share of those aged 65 and over is projected to rise from 17% to 30%. As a consequence, the EU would move from having four people of working-age to each person aged over 65 years to about two people of working-age.
>
> *European Commission, Department of Economic and Financial Affairs* (2012) [10]

1.1 Motivation

The lack of personnel will pose a significant problem on the elder care services in the European Union and Germany in the years to come [10]. Many possible solutions are being discussed and among them is the use of robots and other automated systems to assist elderly people as well as care personnel. The German Federal Ministry of Education and Research thus funds the *KoBo34 project*[1], which aims at developing a dual-arm humanoid robot that can intuitively interact with humans [11]. The *Intelligent Autonomous Systems Group* (IAS) of the *Technische Universität Darmstadt*, as well as the *Technical University of Munich* and the *Rosenheim University of Applied Sciences*, will be working from 2018 to 2021 on this project under the coordination of *Franka Emika GmbH*.

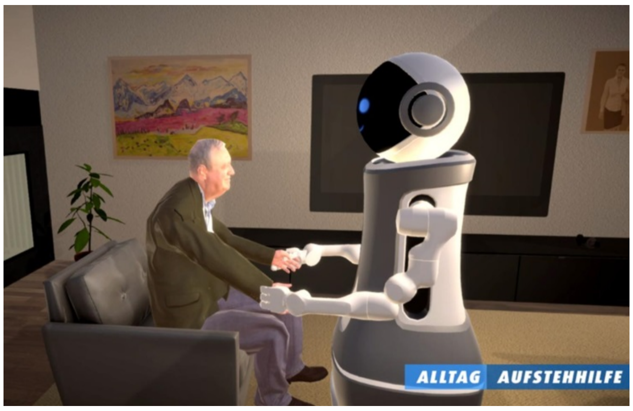


Figure 1.1: Symbolic image: *Kobo helps in everyday life* [1]. It shows a possible scenario where a mobile robot assists an elderly person with standing up from a sitting position by providing support to grab on to.

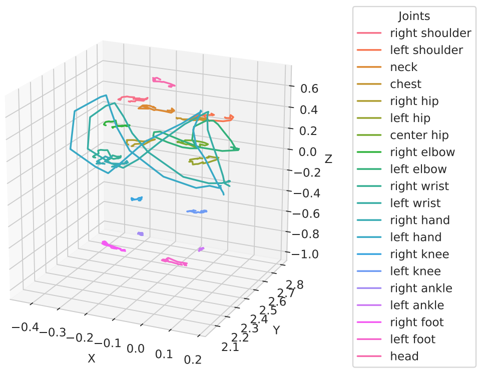


Figure 1.2: Representation of the *tennis* gesture of the *MSR Action3D* data set used for evaluation of the algorithms investigated in this thesis. The so-called skeletons consist of many joints, moving through space over time.

Generally, the contextualization, segmentation and classification of the human actions and gestures, as well as responding in a helpful way, are important steps for the interaction between humans and the assistive robot being developed. This thesis will work on the aspect of classification of *Activities of Daily Living* (ADLs) in the context of *Active and Assisted Living* (AAL). The interaction between the robot and humans will likely require a *multimodal* approach [12], i.e. including more than one sensing method, like integrating cameras, microphones and more simultaneously. It has been shown, however, that gestures and especially hand gestures play a vital role in human-to-human communication since they act "as the critical link between our conceptualizing capacities and our linguistic abilities" [13, p. 119]. This research will thus focus on the ability of the system to classify human gestures of the upper body, concentrating on hand gestures. That body region is also selected because many elderly people will likely be sitting when interacting with the robot because of their age and health, making the movement of the lower body parts less relevant for the interaction with the robot than the upper ones.

[1] The name is derived from "**Ko**operativer Assistenzro**Bo**ter für das **3**. und **4**. Lebensalter" (german) [11], which translates to *cooperative assistive robot for the 3rd and 4th ages of life*.

This thesis will investigate different supervised machine learning (ML) techniques to classify human gestures and actions based on skeleton data. Machine learning allows the system to generalize the algorithms being developed to many different scenarios, where the gestures to be recognized might change. This leads to another important criterion for the classification: It has to be able to generalize to *multiple subjects*, since it is likely that multiple people will be interacting with the same robot but would perform gestures differently. This will be evaluated experimentally by choosing the training and testing data accordingly. Going further, the system shall not only learn in one phase and classify in the second one, but rather learn on demand (*online*) when new gestures or a change in the execution of existing gestures is detected or the system is notified about it by a user. As a part of that, it must be able to recognize that some performed gesture is not known at all, i.e. it has to perform *classification against unknown*.
The use of skeleton data or trajectories of them, instead of for example direct visual data, is motivated by the vast reduction of the input data to only a few data points per time step, while conserving the essential information for this application.
Recognition and classification of human actions and gestures is not only important in the KoBo34 project, but in many different domains and applications as well, including: general human-computer interaction (HCI) [3, p. 130], semantic scene understanding [14, 15], content based video analysis [16], computational behavioral science [3, p. 130], ambient intelligence [17, p. 248], surveillance [3, p. 130], assistive technologies [3, p. 130], sign language understanding [3, p. 130] [18, p. 311], consumer behavior analysis [3, p. 130], entertainment/gaming [19, 20], animation [19, 16] and more.
Summing up, the general goal of this thesis is the development of a supervised machine learning system that shall be able to classify human gestures based on trajectories of joints. It has to be able to generalize to multiple subjects and allow for online learning.

1.2 Outline

This thesis is structured as follows:

Chapter 2 introduces the terminology being used as well as foundations on which this thesis builds upon. As part of that, the related literature is reviewed. Different time sequence alignment methods, probabilistic movement primitives, support vector machines, dimensionality reduction approaches, and available gesture data sets are presented.

Chapter 3 describes our approach at gesture classification. Starting with an exact problem statement, different preprocessing steps, dimensionality reduction methods, and classification possibilities are being discussed. Finally, descriptions of the newly recorded and existing data sets later used for evaluation are given.

Chapter 4 evaluates the methods presented previously, both qualitatively and quantitatively. It compares the different approaches and lays out the methodology used for testing the methods as well. Important characteristics of the implementation are given as part of this chapter.

Chapter 5 summarizes the approaches taken a gesture classification and the experimental results. Concluding from that, achievements and possible future directions of research, are discussed.

2 Foundations and Related Work

This section will provide the basic terminology and covers the foundations of this thesis. As part of that, the related literature is reviewed.

2.1 Terminology

Some sources differentiate between the terms *action* as a goal-driven motion of a body part and *gesture* as the desire motivating an action or series of actions [21, p. 129f] (or as a "visible action as utterance" [22, p. 2]). This thesis does not differentiate between these two concepts, since the distinction between those two is not in the scope of this thesis and might even be unfeasible without the integration of multimodal data. Treating both terms interchangeable follows the practice of many other authors in this area of research [3, 23, 18].
Gestures can be divided into several types of actions, of which each are relevant in this attempt to classify them:

- **Deictic gestures** draw attention to some spatial or temporal point (e.g. by pointing at something).
- **Conventional gestures** use some (culturally specific) convention of movement (e.g. nodding for approval).
- **Representational gestures** capture some concrete aspects of a concept or object, and can be divided into
  - **Iconic gestures**, which depict the concept or object being referred to directly (e.g. a hand twisting motion for opening a jar), and
  - **Metaphoric gestures**, which depict some transferred idea (e.g. being unsure by moving the hands like a pair of scales).

This generally follows the classification of *Cartmill & Goldin-Meadow* [21, p. 131], which was originally developed by *McNeill* in [24, p. 75ff] and discussed further by *Goldin-Meadow* [25, p. 6f].
In the context of this thesis, *recognition* describes the *classification* of gestures synonymously, following the terminology of the three aforementioned authors [3, 23, 18] as well as many others, since segmentation is not part of this approach.

2.2 Skeleton Data

A gesture can be recorded by various means, the most simple one being the use of cameras. These could either produce single or multiple normal RGB (*red, green, blue*) images, but also RGB-D (*red, green, blue, depth*) images, as are shown in figure 2.1 (a) and (b). Both exist as (commercial) off-the-shelf (COTS) products from various manufacturers [3, pp. 132f]. The resulting data representing a gesture is a (finite) sequence of frames $\boldsymbol{F}_1, \ldots, \boldsymbol{F}_n$, with each frame being a matrix $\boldsymbol{F}_i \in \mathbb{P}^{h \times w}$ of equal height h and width w. Each entry or pixel in the matrix is then a tuple $(r, g, b) \in \mathbb{P}$ or $(r, g, b, d) \in \mathbb{P}$, for RGB and RGB-D images respectively.

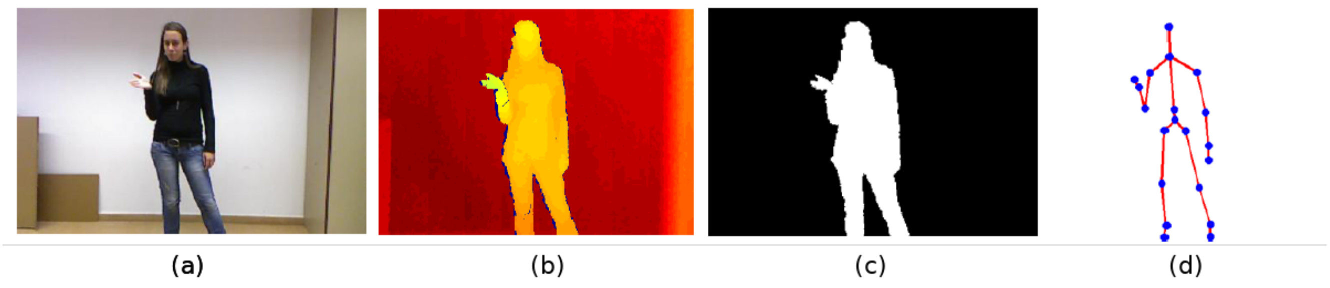


Figure 2.1: Different types of gesture representations: (a) RGB image; (b) color-coded depth image, with darker/more reddish areas being farer away; (c) segmentation map, where white indicates human body parts, and black the absence thereof; and (d) human skeleton data. Adapted from *Escalera et al.* [2].

Another approach is the use of so called *skeletons*, which represent the data on discrete points (see figure 2.1 (d)), which are intuitively chosen to be the joints of a human, like for example a shoulder, elbow or wrist. Skeletal data can

be obtained by either tracking markers at the joints, using an exoskeleton or by employing a special skeleton tracker software, which is also widely available [3, pp. 132f].
The selection of the joints to be used varies, and is a design choice illustrated by figure 2.2. Both the 15- and the 20-joint skeletons have been successfully used for classification [3, p. 131]. Furthermore, in principle both the translation (i.e. the position) of the joint points, as well as the angle between the inter-joint segments could be used to describe a pose. The joint positions will be used in this thesis, as done often in conjunction with the *ProMPs* introduced soon (cf. section 2.4). Each red point in the figure is a point $\boldsymbol{x}_i \in \mathbb{R}^3$ (or some lower dimensional space) and a pose can be expressed as a tuple of all points that were chosen, for example $\boldsymbol{P} = (x_1, \ldots, x_{15})$ for a 15-joint pose. A gesture is then simply a finite sequence $\boldsymbol{G} = (\boldsymbol{P}_1, \ldots, \boldsymbol{P}_n)$ of some length n.

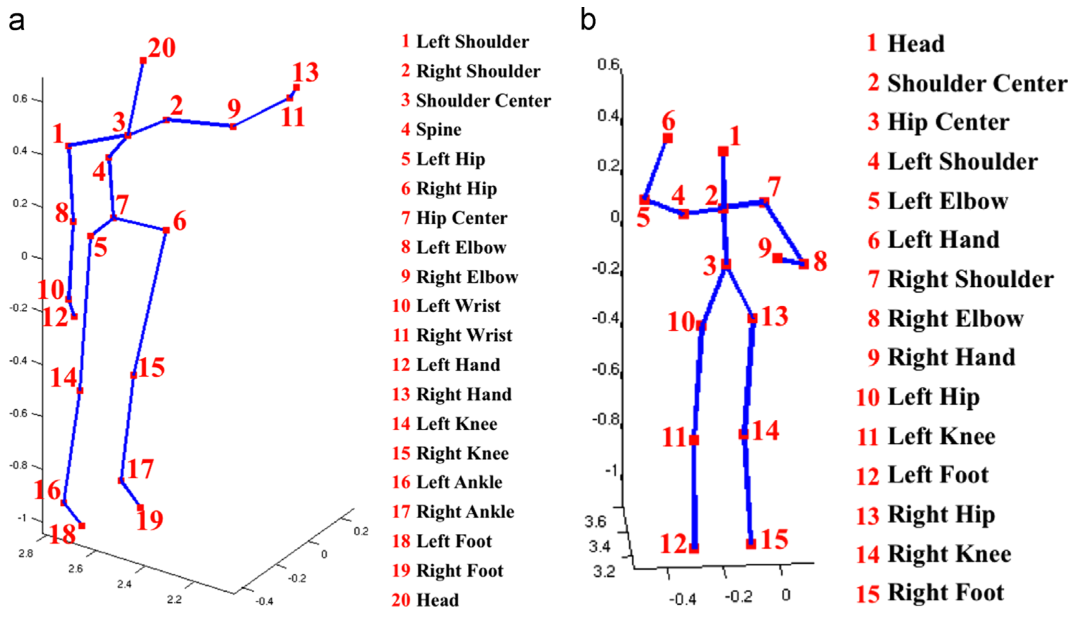


Figure 2.2: Different joints used for the skeleton representation, with (a) 20 joints and (b) 15 joints. Both are used in current data sets. The general idea is, to represent a human posture by only the spatial position of specific *joints*. These so-called joints may or may not correspond to the actual anatomy of humans. Taken from Lo Presti & La Cascia [3, p. 131].

2.3 Time Sequence Alignment with Dynamic Time Warping

In ML it is often required to align two time series which are similar except for some offset in time or different execution speed. That also includes waiting at the start or end of a recording, which is illustrated in figure 2.3. The *Dynamic Time Warping* (DTW) algorithm presented by *Hiroaki & Seibi* in 1978 employs a dynamic programming (DP) approach to solve this problem [26]. It is often used for calculating the similarity of two time sequences but can easily be extended to obtain the time warping required to interpolate one sequence as good as possible to the other one. The algorithm has since been generalized to multidimensional time sequences by *Shokoohi-Yekta et al.* [27] and heavily optimized DTW implementations exist as well, e.g. by *Salvador & Chan* [28]. This section describes the foundations of the original algorithm, with the now-used “symmetric” variant following the original paper of Hiroaki & Seibi.
The goal is to align two time sequences $\mathscr{A}$ of length A and $\mathscr{B}$ of length B. The time sequence $\mathscr{A}$ evaluated at some point a will be denoted as $\mathscr{A}(a)$, and as $\mathscr{B}(b)$, respectively for $\mathscr{B}$ evaluated at point b. The distance between the sequences at some points (a, b) will be defined as the Chebyshev distance, which is the simple absolute distance between the two functions evaluated at some point in time when the functions are simply real-valued as in this introduction: $d(a, b) = |\mathscr{A}(a) - \mathscr{B}(b)|$. We now construct a matrix $\boldsymbol{D}$ of size $A \times B$ to hold the intermediate distances of the DP algorithm, and use $\boldsymbol{D}[a, b]$ as a notion for the element in row a and column b. A backwards search at the end yields the warped path. The entire procedure is described in algorithm 2.1. The result is a discrete solution of the alignment of two functions by viewing them as discrete time sequences $\mathscr{A}$ and $\mathscr{B}$.

2.4 Probabilistic Movement Primitives (ProMPs)

One versatile approach at working with skeleton data are *probabilistic movement primitives* – or short ProMPs – first presented by *Paraschos et al.* in 2013 [29]. In general they have many desirable properties, like being able to smoothly

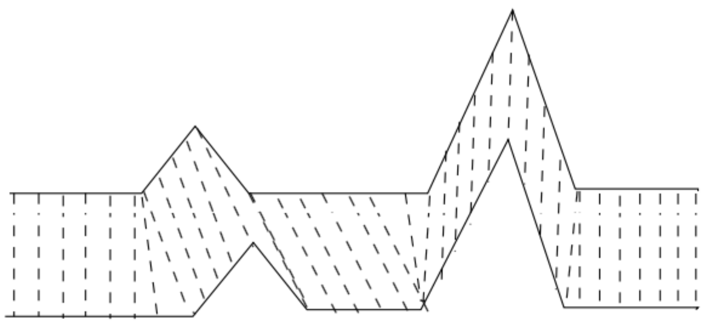


Figure 2.3: Illustration of the Dynamic Time Warping algorithm applied to a one dimensional function. The dashed lines show corresponding points in the time sequences obtained from applying the DTW algorithm. Taken from *Wikimedia* [4].

```
Algorithm: DTW path calculation without constraints
Data: time sequences $\mathscr{A}$ and $\mathscr{B}$ of length $A$ and $B$, some distance metric $d$ as defined previously
Result: a list $\boldsymbol{R}$ of pairs $(a, b)$ matching the sample points in the two given time sequences

Initialize the distance matrix
let $\boldsymbol{D} \in \mathbb{R}^{A \times B}$
$\boldsymbol{D}[1,1] \leftarrow 2 \cdot d(1,1)$

Initialize the backwards vector matrix; We store the origin of each iteration result
in order to make the final backwards search easier.
let $\boldsymbol{V} \in (\mathbb{R} \times \mathbb{R})^{A \times B}$
$\boldsymbol{V}[1,1] \leftarrow (0,0)$

Calculate distance matrix with DTW
for $a \leftarrow 1$ to $A$ do
    for $b \leftarrow 1$ to $B$ do
        $cost \leftarrow d(a,b)$
        if $a = 1$ then
            $\boldsymbol{D}[a,b] \leftarrow \boldsymbol{D}[a,b-1] + cost$
        else if $b = 1$ then
            $\boldsymbol{D}[a,b] \leftarrow \boldsymbol{D}[a-1,b] + cost$
        else
            $\boldsymbol{D}[a,b] \leftarrow \min \begin{cases} \boldsymbol{D}[a,b-1] + cost, & \mathscr{B} \text{ is faster} \\ \boldsymbol{D}[a-1,b-1] + 2 \cdot cost, & \text{both already match} \\ \boldsymbol{D}[a-1,b] + cost, & \mathscr{A} \text{ is faster} \end{cases}$
        $\boldsymbol{V} \leftarrow$ the index into $\boldsymbol{D}$ used to calculate $\boldsymbol{D}[a,b]$

Find lowest-cost path using backwards search
$\boldsymbol{R} \leftarrow []$ initialize with empty list
$a \leftarrow A$
$b \leftarrow B$
while $a > 0 \wedge b > 0$ do
    prepend $(a,b)$ to $\boldsymbol{R}$
    $a, b \leftarrow \boldsymbol{V}[a,b]$
return $\boldsymbol{R}$
```

Algorithm 2.1: Algorithm for finding the matching indices of two time sequences using DTW based on the original publication by Hiroaki & Seibi, but with a newly introduced helper matrix $\boldsymbol{V}$. In a first step, the distances resulting from each possible connection are calculated with a DP approach and their origin is stored. To obtain the actual path representing the optimal alignment, a backwards search is performed by following $\boldsymbol{V}$ back to the origin.

blend multiple gestures, conditioning to pass through a specific spacial and temporal point or provide the basis for a motor controller with coupled joints. This follows from the probabilistic approach, viewing the gestures as a distribution over trajectories.
The basic idea of ProMPs is that firstly, the sequence of joint positions is approximated with a series of weighted (gaussian) radial basis functions. Secondly, the distribution of those weights is learned and later used for probabilistic classification. The weights serve as a dimensionality reduction and the probabilistic formulation captures the inherent variance of human gestures by calculating both the mean and the full covariance matrix.
The following sections shall introduce the aspects of ProMPs which are relevant for this application. They are based on the original paper by Paraschos et al. and on the work of *Koert et al.* [9]. ProMPs are first introduced for gestures of a single joint and spatial dimension and generalized at the end. Both batch learning (offline) and incremental learning (online) is presented.

Transformation into Weight Space

A pose or position $\tau_t \in \mathbb{R}$ at time t is represented by a weight vector $\boldsymbol{w} \in \mathbb{R}^N$ with $N \geq 2$ weights as follows:

$$\tau_t = \boldsymbol{\Phi}_t \cdot \boldsymbol{w} = \begin{pmatrix} \phi_1(t) & \phi_2(t) & \dots & \phi_N(t) \end{pmatrix} \cdot \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_N \end{pmatrix} \tag{2.1}$$

A full trajectory sampled at T points $\boldsymbol{\tau} \in \mathbb{R}^T$ in time can then be approximated as:

$$\begin{pmatrix} \tau_1 \\ \tau_2 \\ \vdots \\ \tau_T \end{pmatrix} = \boldsymbol{\tau} = \boldsymbol{\Phi} \cdot \boldsymbol{w} = \begin{pmatrix} \phi_1(1) & \phi_2(1) & \dots & \phi_N(1) \\ \phi_1(2) & \phi_2(2) & \dots & \phi_N(2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_1(T) & \phi_2(T) & \dots & \phi_N(T) \end{pmatrix} \cdot \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_N \end{pmatrix}. \tag{2.2}$$

$\boldsymbol{\Phi}$ now is a matrix with a separate row for each point in time.
The motion of the actor $\boldsymbol{\tau}$ is approximated by the weighted sum of some number of basis functions $\phi_n(t)$. ϕ is chosen to be a *radial basis function* (RBF), like for example the gaussian function, resulting in

$$\begin{aligned} \phi(x) &= e^{-(\epsilon x)^2}, \text{ with } \epsilon = 2(N-1)\sqrt{-\ln h}, \text{ and} \\ \phi_n(t) &= \phi\left(\frac{t-1}{T-1} - \frac{n-1}{N-1}\right). \end{aligned} \tag{2.3}$$

In the formula for ϵ, $0 < h < 1$ is the height at which two neighboring ϕ_n and ϕ_{n+1} intersect, as is illustrated in figure 2.4. Values of h between 0.6 and 0.8 have given good results. It follows from the idea that ϕ_n should be h at the point halfway between to maximum of itself and of ϕ_{n+1}:

$$\begin{aligned} & & \phi\left(\frac{1}{2}\frac{1}{N-1}\right) &= h \\ &\iff & e^{-\epsilon^2\left(\frac{1}{2(N-1)}\right)^2} &= h \\ &\iff & -\epsilon^2 \frac{1}{(2(N-1))^2} &= \ln h \\ &\iff & \epsilon^2 &= (-\ln h)(2(N-1))^2 \\ &\iff & \epsilon &= 2(N-1)\sqrt{-\ln h} \end{aligned} \tag{2.4}$$

Since ϵ will be squared anyway, we can simply choose the positive solution in the second-last step.
The weights could then be calculated using linear regression: $\Phi^{-1}\boldsymbol{\tau} = \boldsymbol{w}$. But since that might be numerically unstable in the case of $\boldsymbol{\Phi}$ not being positive definite, *ridge regression* can be used to regularize it [30]. We first write it using the

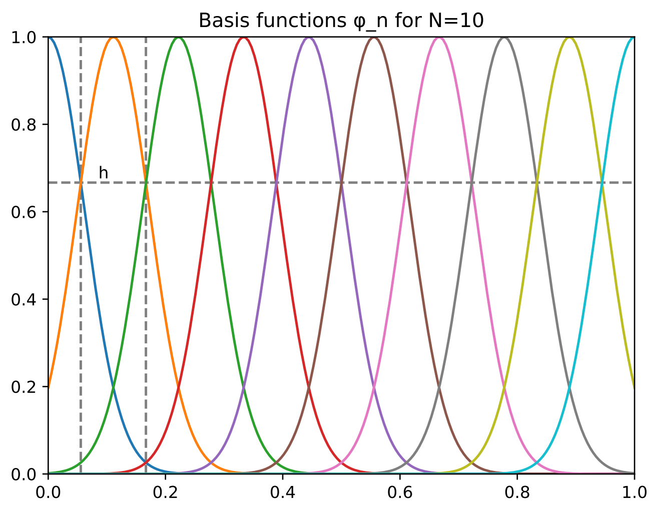


Figure 2.4: This shows $N = 10$ basis functions ϕ_n that can then be used to approximate a single trajectory by weighting the functions differently. The value of h was set to 2/3 and is shown by the horizontal dashed line. The vertical dashed line shows the two points at which the orange function ϕ_2 were was chosen to be h (cf. equation (2.4)).

pseudo inverse: $\left(\boldsymbol{\Phi}^T\boldsymbol{\Phi}\right)^{-1}\boldsymbol{\Phi}^T\boldsymbol{\tau} = \boldsymbol{w}$. We then add a small regularization to the diagonal of the matrix before calculating the inverse,

$$\left(\boldsymbol{\Phi}^T\boldsymbol{\Phi} + \lambda\,\mathbb{I}\right)^{-1}\boldsymbol{\Phi}^T\boldsymbol{\tau} = \boldsymbol{w}, \tag{2.5}$$

where $\lambda \in \mathbb{R}_+$ is a positive factor close to zero [31, p. 64].
The result of choosing different numbers of weights is evaluated in section 4.2.4. The approximation provided by ProMPs can be seen in figure 2.5.

Assembly of a ProMP (Batch Learning)

The goal of the learning phase is to calculate the D weight vectors $\left\{\boldsymbol{w}^d\right\}_{d=1...D}$ for the given trajectories $\left\{\boldsymbol{\tau}^d\right\}_{d=1...D}$ individually. The ProMP is then created by calculating the mean $\boldsymbol{\mu}_w^c$ and covariance matrix $\boldsymbol{\Sigma}_w^c$ of the weight vectors per class c. If the system consists of multiple joints and/or dimensions, the mean of each of them are stacked into a single vector and the covariance matrix $\boldsymbol{\Sigma}_w^c$ between all of them is calculated. Together they form the probabilistic movement primitive $\left(\boldsymbol{\mu}_w^c, \boldsymbol{\Sigma}_w^c\right)$ of a single class c. Some example ProMPs of drawing the symbol “9” in 2D are shown in section 4.2.3. The entire procedure of calculating ProMPs in batch/offline (using all training data at once) is shown in algorithm 2.2.

```
Algorithm: Batch calculation of ProMPs (offline)
Data: the classes of the traing data $c_1, c_2, \ldots c_C$ and the corresponding trajectories per class $\boldsymbol{\tau}_1^{c_i}, \boldsymbol{\tau}_2^{c_i}, \ldots, \boldsymbol{\tau}_D^{c_i}$ each
Result: a ProMP $(\boldsymbol{\mu}_w^c, \boldsymbol{\Sigma}_w^c)$ per each class $c$ in the training data
foreach class $c$ in the input data do
    foreach trajectory $\boldsymbol{\tau}_d^c$ of class $c$ do
        $\boldsymbol{w}_d^c \leftarrow \left(\boldsymbol{\Phi}^T\boldsymbol{\Phi} + \lambda\,\mathbb{I}\right)^{-1}\boldsymbol{\Phi}^T\boldsymbol{\tau}_d^c$
    $\boldsymbol{\mu}_w^c \leftarrow \mathbb{E}(\boldsymbol{w}_d^c,$ for $d = 1 \ldots D)$
    $\boldsymbol{\Sigma}_w^c \leftarrow \mathrm{Cov}(\boldsymbol{w}_d^c,$ for $d = 1 \ldots D)$
```

Algorithm 2.2: The algorithm used to calculate ProMPs in batch. If the trajectories consist of multiple dimensions, the weight vector must be stacked with all dimensions right after calculating the weights. A single trajectory then forms a single weight vector again.

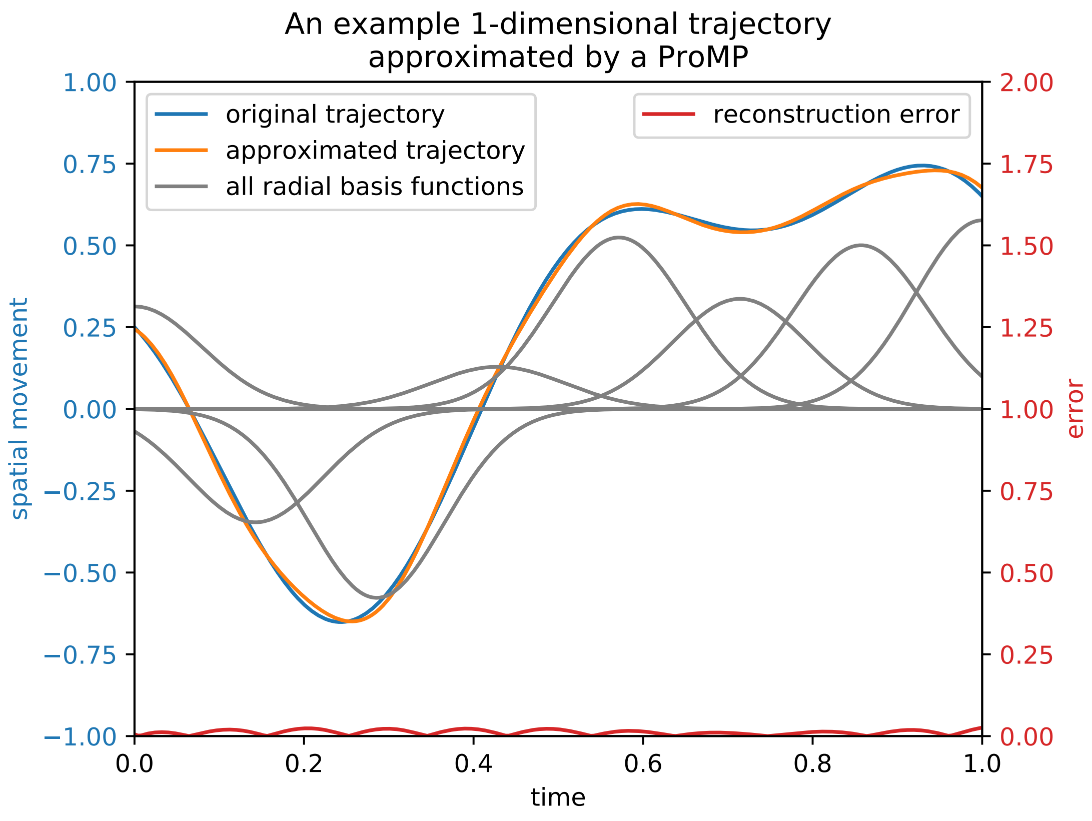


Figure 2.5: Illustration of approximating one-dimensional synthetic movement using a ProMP with $N = 8$ weights. The movement is generated by equation (4.1), which will be introduced later. The original trajectory to be approximated is shown in blue. The radial basis functions scaled by the corresponding weights are shown in grey. Summed up, they form the approximated trajectory shown in orange. Finally, the red plot on the bottom shows the absolute error at each point in time introduced by this transformation.

Classification

The result of the ProMP learning phase is a mean weight vector $\boldsymbol{\mu}_w^c$ and a weight covariance matrix $\boldsymbol{\Sigma}_w^c$ per each class c, forming a multivariate gaussian distribution each. Together they form a mixture distribution, in this case a *gaussian mixture model* (GMM). This can then be used for classification, as shown generally by *Bishop* [32, pp. 110–113, 430–432] and applied to ProMPs by Koert et al. [9].

We want to be able to calculate the probability density $\mathrm{p}(c \mid \boldsymbol{\tau})$ of a given trajectory $\boldsymbol{\tau}$ being of class c. Since we can transform trajectories into weight space, using equation (2.5), it is equivalent to look at the weights alone. Using *Bayes' Theorem* and a *uniform prior* $\mathrm{p}(c_j) = \mathrm{p}(c) = \frac{1}{C}$, we can calculate:

$$\begin{aligned}\mathrm{p}(c \mid \boldsymbol{\tau}) = \mathrm{p}(c \mid \boldsymbol{w})&\\ &= \frac{\mathrm{p}(\boldsymbol{w} \mid c)\,\mathrm{p}(c)}{\mathrm{p}(\boldsymbol{w})}\\ &= \frac{\mathrm{p}(\boldsymbol{w} \mid c)\,\mathrm{p}(c)}{\sum_{j=0}^{C}\left[\mathrm{p}(\boldsymbol{w} \mid c_j)\,\mathrm{p}(c_j)\right]}\\ &= \frac{\mathrm{p}(\boldsymbol{w} \mid c)\,\mathrm{p}(c)}{\left[\sum_{j=0}^{C}\mathrm{p}(\boldsymbol{w} \mid c_j)\right]\mathrm{p}(c)}\\ &= \frac{\mathrm{p}(\boldsymbol{w} \mid c)}{\sum_{j=0}^{C}\mathrm{p}(\boldsymbol{w} \mid c_j)}\end{aligned} \tag{2.6}$$

with some number of classes C. We can then calculate the probability density $\mathrm{p}(\boldsymbol{w} \mid c)$ that some given trajectory $\boldsymbol{\tau}$ with some weight vector $\boldsymbol{w}$ occurs given a class c with the ProMP $\left(\boldsymbol{\mu}_w^c, \boldsymbol{\Sigma}_w^c\right)$ by calculating

$$\begin{aligned}
\mathrm{p}(\boldsymbol{w} \mid c) &= \mathrm{p}\left(\boldsymbol{w} \mid \left(\boldsymbol{\mu}_w^c, \boldsymbol{\Sigma}_w^c\right)\right) \\
&= \mathcal{N}\left(\boldsymbol{w} \mid \boldsymbol{\mu}_w^c, \boldsymbol{\Sigma}_w^c\right) \\
&= \frac{1}{\sqrt{(2\pi)^N \det\left(\boldsymbol{\Sigma}_w^c\right)}} \exp\left(-{}^1\!/_2 \left(\boldsymbol{x} - \boldsymbol{\mu}_w^c\right)^T \left(\boldsymbol{\Sigma}_w^c\right)^{-1} \left(\boldsymbol{x} - \boldsymbol{\mu}_w^c\right)\right).
\end{aligned} \tag{2.7}$$

This amounts to evaluating the probability density function (PDF) of the multivariate normal distribution $\mathcal{N}$.
Using above formula, we can finally classify the given trajectory $\boldsymbol{\tau}$ with

$$c_{\text{predicted}} = \arg\max_c \mathrm{p}(c \mid \boldsymbol{\tau}). \tag{2.8}$$

Classification against unknown is then done simply by determining whether at least one class c yields an unnormalized probability density $\mathrm{p}(\boldsymbol{w} \mid c)$ exceeding some predefined threshold $p_{\min}$.

Incremental Learning of ProMPs

These ProMPs can also be learned incrementally (i.e. trajectory by trajectory), as demonstrated for ProMPs by Koert et al. [9, p. 602], based on the work of *Engel & Heinen* [33]. As we do not use a gating model with our labeled data like Koert et al. did, the *responsibilities* λ_{kn} are always 1 for the class of the training example and are 0 elsewhere. This simplifies the procedure and results in the algorithm 2.3.

```
Algorithm: Incremental calculation of ProMPs (online)
Data: trajectories $\boldsymbol{\tau}_1, \boldsymbol{\tau}_2, \ldots, \boldsymbol{\tau}_D$ and the corresponding classes $k_1, k_2, \ldots, k_C$
Result: a ProMP $(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$ per each class $k$ that was represented by at least one example in the input data
foreach pair $(\boldsymbol{\tau}, k)$ in the input data do
    calculate the weights
    $\boldsymbol{w} \leftarrow \left(\boldsymbol{\Phi}^T \boldsymbol{\Phi} + \lambda \mathbb{I}\right)^{-1} \boldsymbol{\Phi}^T \boldsymbol{\tau}$
    if class $k$ is seen for the first time then initialize ProMP
        $\boldsymbol{\mu}_k \leftarrow \boldsymbol{w}$
        $\boldsymbol{\Sigma}_k \leftarrow {}^1\!/_{10}\ \mathbb{I}_{N \times N}$
        $\mathrm{age}_k \leftarrow 1$
    else update ProMP
        update helper values
        $\mathrm{age}_k \leftarrow \mathrm{age}_k + 1$
        $\gamma \leftarrow \frac{1}{\mathrm{age}}$
        $\tilde{\gamma} \leftarrow \gamma + e^{-\mathrm{age}}$
        update result
        $\boldsymbol{\mu}_k^{prev} \leftarrow \boldsymbol{\mu}_k$
        $\boldsymbol{\mu}_k \leftarrow \boldsymbol{\mu}_k + \gamma(\boldsymbol{\tau} - \boldsymbol{\mu}_k)$
        $\boldsymbol{\Sigma}_k \leftarrow (1 - \tilde{\gamma})\boldsymbol{\Sigma}_k + \tilde{\gamma}(\boldsymbol{\tau} - \boldsymbol{\mu}_k)(\boldsymbol{\tau} - \boldsymbol{\mu}_k)^T - (\tilde{\gamma} - \gamma)(\boldsymbol{\tau} - \boldsymbol{\mu}_k^{prev})(\boldsymbol{\tau} - \boldsymbol{\mu}_k^{prev})^T$
```

Algorithm 2.3: The algorithm used to incrementally calculate ProMPs, adapted from Koert et al. [9, p. 602]. The weights representing the trajectory are calculated first. If the trajectory was not seen before, a new component of the mixture model is initialized, and else the existing one is updated. If the trajectories consist of multiple dimensions, the weight vector must be stacked with all dimensions right after calculating the weights and mean and covariance are then calculated over that stacked vector. A single trajectory then forms a single weight vector again, analogous to the batch calculation approach.

2.5 Sampling from Multivariate Normal Distributions

It is sometimes helpful to be able to sample from a probability distribution, like for example from ProMPs or the original trajectories in joint space. This enables the creation of gestures typical for a given ProMP in the weight space, which can

simply be transferred into the original joint space by multiplication with $\mathbf{\Phi}$ and be visualized. The following procedure is well suited for the sampling from (multivariate) normal distributions and is taken from *Gentle* [34, p. 307] and Bishop [32, p. 528].
Let $\boldsymbol{X} \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$ be some multivariate random variable we want to sample from. We first calculate a decomposition of the covariance matrix $\boldsymbol{D}\boldsymbol{D}^T = \Sigma$, for example by using the Cholesky decomposition. If $\boldsymbol{D}$ was determined numerically, it is often required to add some regularization afterwards, analogous to ridge regression: $\boldsymbol{D}' = \boldsymbol{D} + \lambda\,\mathbb{I}$. For example, a small $\lambda = 10^{-6}$ often works well. Then we generate some vector ξ, with each component being independently normal distributed: $\xi_i \sim \mathcal{N}(0, 1)$. Finally, we compute the desired sampled vector as

$$\boldsymbol{X} = \boldsymbol{D}'\xi + \boldsymbol{\mu}. \tag{2.9}$$

A visualization of trajectories and weights being sampled using this method can be found in figure 4.9.

2.6 Support Vector Machines (SVMs)

Support vector machines (SVMs) are a standard method in machine learning and can be used for both classification and regression [32, p. 325]. They can also be extended to multiclass classification, probabilities and online learning. SVMs are sparse classifiers in that they only need to store a small amount of training data points called *support vectors* once learning is over. They are also known for coping well with very high dimensional input data. Conceptually, SVM classifiers learn a hyperplane that separates the positive and negative examples. This is done in such a way that the distance of the so called *decisison border* to the closest data points (called support vectors) gets maximized, as is visualized in figure 2.6. Thus, they are also called *maximum margin classifiers*. The basic principles of two-class SVM classification shall be layed out in the following section based on Bishop [32, pp. 325–331]. Further improvements on SVMs are beyond the limits of this introduction.

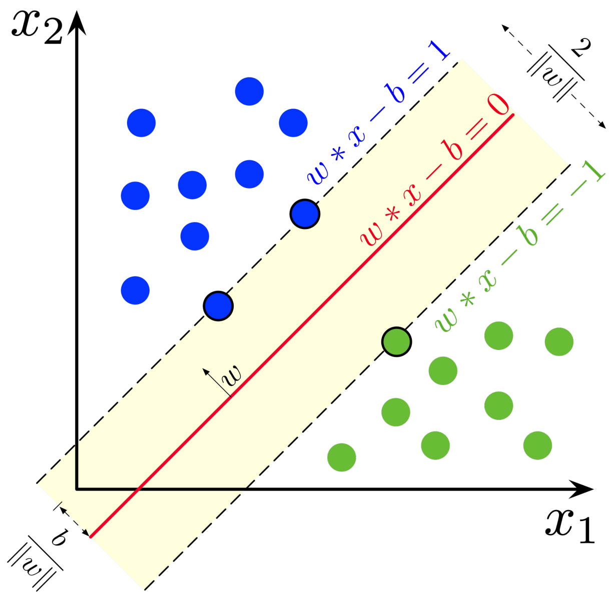


Figure 2.6: Illustration of the margin that separates two classes in a SVM. It shows how the hyperplane, shown in red, separates the two classes shown in blue and green. In this case, it is a line separating a plane into two areas. The border region is shown in yellow, and the distance between the red line and the support vectors in black contour is being maximized. Taken from *Wikimedia* [5].

2.6.1 General Introduction to SVMs

Let the $N > 0$ training data points be represented by the vectors $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$, each with a corresponding target value $t_n \in \{-1, +1\}$. The target value is $+1$ for positive examples and -1 for negative ones. A linear classifier can now be defined by classifying each value $\boldsymbol{x}$ by the sign of $y(\boldsymbol{x}) : \mathbb{R}^D \to \mathbb{R}$, with

$$y(\boldsymbol{x}) = \boldsymbol{w}^T\phi(\boldsymbol{x}) + b. \tag{2.10}$$

Let us further assume that the data is linearly separable, i.e. there exist some $\boldsymbol{w}$ and b, such that $\operatorname{sign}(y(\boldsymbol{x}_n)) = t_n$ for all $n = 1, \dots, N$. An equivalent notion of this condition is that $t_n \cdot \operatorname{sign}(y(\boldsymbol{x}_n)) > 0$ for all $n = 1, \dots, N$.
The hyperplane we are trying to find is defined by $y(\boldsymbol{x}) = 0$, and the distance of a point $\boldsymbol{x}_n$ to that plane is generally given by

$$\frac{|y(\boldsymbol{x}_n)|}{||\boldsymbol{w}||}. \tag{2.11}$$

As $t_n \cdot \operatorname{sign}(y(\boldsymbol{x}_n)) > 0$, we can equivalently formulate the distance as follows:

$$\frac{t_n y(\boldsymbol{x}_n)}{||\boldsymbol{w}||} = \frac{t_n(\boldsymbol{w}^T \phi(\boldsymbol{x}_n) + b)}{||\boldsymbol{w}||}. \tag{2.12}$$

The optimization problem of finding the maximum-margin hyperplane can subsequently be formalized by optimizing

$$\underset{\boldsymbol{w},b}{\arg\max} \left\{ \frac{1}{||\boldsymbol{w}||} \min_{n \in \{1,\dots,N\}} \left[t_n \left(\boldsymbol{w}^T \phi(\boldsymbol{x}_n) + b \right) \right] \right\}. \tag{2.13}$$

In order for this complex optimization problem to be solved it needs simplification. This can be achieved with a slight rewrite of the distance of the points to the hyperplane. We first note that some rescaling $y'(\boldsymbol{x})$ with $\boldsymbol{w} \to \kappa \boldsymbol{w}$ and $b \to \kappa b$ does not change the distance of any point $\boldsymbol{x}_n$ to the decision boundary:

$$\frac{t_n y'(\boldsymbol{x}_n)}{||\kappa \boldsymbol{w}||} = \frac{t_n(\kappa \boldsymbol{w}^T \phi(\boldsymbol{x}_n) + \kappa b)}{||\kappa \boldsymbol{w}||} = \frac{t_n \kappa (\boldsymbol{w}^T \phi(\boldsymbol{x}_n) + b)}{\kappa ||\boldsymbol{w}||} = \frac{t_n(\boldsymbol{w}^T \phi(\boldsymbol{x}_n) + b)}{||\boldsymbol{w}||} = \frac{t_n y(\boldsymbol{x}_n)}{||\boldsymbol{w}||}. \tag{2.14}$$

This allows us to to define

$$t_n \left(\boldsymbol{w}^T \phi(\boldsymbol{x}_n) + b \right) = 1 \tag{2.15}$$

for the point $\boldsymbol{x}_n$ closest to the hyperplane without changing the optimization problem. We now obtain that all data points $\boldsymbol{x}_n$ fulfill:

$$t_n \left(\boldsymbol{w}^T \phi(\boldsymbol{x}_n) + b \right) \geq 1, \qquad n = 1, \dots, N. \tag{2.16}$$

These are constraints to the optimization problem, which we now reduced to

$$\underset{\boldsymbol{w},b}{\arg\max} \frac{1}{||\boldsymbol{w}||} = \underset{\boldsymbol{w},b}{\arg\min} ||\boldsymbol{w}|| = \underset{\boldsymbol{w},b}{\arg\min} \frac{1}{2} ||\boldsymbol{w}||^2 \tag{2.17}$$

subject to the constraints given by equation (2.16) and slightly rewritten to make later observations easier.
It turns out that such a *quadratic programming* problem along with some linear inequality constraints can be tackled, using so called *Lagrange multipliers*. See Bishop [32, pp. 707–710] for a short introduction. The basic idea is to introduce some so called Lagrange multipliers $a_n \geq 0$ for $n = 1, \dots, N$, used for representing the constraints given in equation (2.16) together with the goal from equation (2.17), forming a single closed-form equation. Its minimization is equivalent to the above optimization problem and each local solution is a global one as well, since the problem is convex in the Lagrangian multiples [32, p. 325]. The Lagrangian function $L(\boldsymbol{w}, b, \boldsymbol{a})$ with the vector of multipliers $\boldsymbol{a} = (a_1, \dots, a_N)^T$ is given by

$$L(\boldsymbol{w}, b, \boldsymbol{a}) = \frac{1}{2} ||\boldsymbol{w}||^2 - \sum_{n=1}^{N} a_n \left[t_n \left(\boldsymbol{w}^T \phi(\boldsymbol{x}_n) + b \right) - 1 \right]. \tag{2.18}$$

As we need to find stationary points of that function, we require that $\frac{\partial L(\boldsymbol{w},b,\boldsymbol{a})}{\partial \boldsymbol{w}} = \boldsymbol{0}$ and $\frac{\partial L(\boldsymbol{w},b,\boldsymbol{a})}{\partial b} = 0$, arriving at the conditions

$$\begin{aligned} \boldsymbol{w} &= \sum_{n=1}^{N} a_n t_n \phi(\boldsymbol{x}_n) \\ 0 &= \sum_{n=1}^{N} a_n t_n. \end{aligned} \tag{2.19}$$

This can be used to transform the previous Lagrangian function from equation (2.18) into this equivalent form:

$$\boxed{\begin{aligned}\widetilde{L}(\boldsymbol{a}) &= \sum_{n=1}^{N} a_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} a_n a_m t_n t_m k(\boldsymbol{x}_n, \boldsymbol{x}_m),\\ &\text{s.t } a_n \geq 0\\ &\text{and } \sum_{n=1}^{N} a_n t_n = 0\end{aligned}} \tag{2.20}$$

in which we eliminated the direct dependance on $\boldsymbol{w}$ and b. This function is to be minimized. Also, the time required for a single evaluation of $\widetilde{L}(\boldsymbol{a})$ is now in $O(N^2)$, instead of $O(N)$ for equation (2.18). This is still acceptable, since we now do not need to be able to calculate $\phi(\boldsymbol{x})$ any more, but only the scalar product defined by the *kernel function* $k(\boldsymbol{x}_1, \boldsymbol{x}_2) = \phi(\boldsymbol{x}_1)^T \phi(\boldsymbol{x}_2)$. This is called the *kernel trick*. It also allows us to work on data sets that are not linearly separable in the space of $\boldsymbol{x}$ by using a non-linear kernel in the above formula.
It turns out that $a_n = 0$ for many n, as these are not the *active* vectors (also called support vectors) and do not contribute to the position of the hyperplane. Let $\boldsymbol{S} = \{n \in \{1, \ldots, N\} \mid a_n \neq 0\}$ be the set of all support vector indices. This allows the learned SVM to forget all data points $\boldsymbol{x}_n$ for $n \notin \boldsymbol{S}$. Classification can finally be done by again evaluating the sign of $y(\boldsymbol{x})$ from equation (2.10), which can now be formulated as

$$\boxed{y(\boldsymbol{x}) = \sum_{n \in \boldsymbol{S}} a_n t_n k(\boldsymbol{x}, \boldsymbol{x}_n) + b.} \tag{2.21}$$

The common value of b can be calculated numerically stable with

$$b = \frac{1}{|\boldsymbol{S}|}\sum_{n \in \boldsymbol{S}}\left(t_n - \sum_{m \in \boldsymbol{S}} a_m t_m k(\boldsymbol{x}_n, \boldsymbol{x}_m)\right). \tag{2.22}$$

The following section 2.6.2 gives a brief introduction into how to actually compute solutions to this problem of minimization $\widetilde{L}(\boldsymbol{a})$ over $\boldsymbol{a}$.

2.6.2 Online Learning of SVMs

SVMs can also be successfully learned online using for example *sequential gradient descent* (SGD, also called *stochastic gradient descent*), as shown by *Zhang* [35] (cf. `SGDClassifier` in `scikit-learn` [36]). First, we need to understand how the learning phase of a normal offline SVM might work. Generally, we have some real-valued function $E(\boldsymbol{a})$ we want to minimize, and which is the sum of some error function $E_n(\boldsymbol{a})$ per training data point $\boldsymbol{x}_n$:

$$E(\boldsymbol{a}) = \sum_{n=1}^{N} E_n(\boldsymbol{a}). \tag{2.23}$$

In the above definitions, $E(\boldsymbol{x})$ would correspond to $\widetilde{L}(\boldsymbol{a})$ from equation (2.20).
The *gradient descent* algorithm can be used to actually solve this problem, by starting with some initialization $\boldsymbol{a}^{(0)}$ and iteratively updating it until some convergence criterion is met:

$$\boldsymbol{a}^{(i+1)} = \boldsymbol{a}^{(i)} - \eta \nabla E\left(\boldsymbol{a}^{(i)}\right) = \boldsymbol{a}^{(i)} - \eta \nabla \left[\sum_{n=1}^{N} E_n\left(\boldsymbol{a}^{(i)}\right)\right]. \tag{2.24}$$

Here, ∇ denotes the gradient of a function, i.e. the vector of all partial derivatives of some real-values function, which intuitively points in the direction of the steepest increase in the function value. The hyperparameter $\eta \in \mathbb{R}_+$ is the learning rate, which can also be automatically adjusted, as *Xu* [37] has proposed.
Since optimizing over E as a whole would require N evaluations of E_n in each single step of the iteration, this can be reduced to only considering a single input data point per iteration. The resulting optimization is called *sequential gradient descent* and runs for exactly N iterations. It follows this modified update rule:

$$\boldsymbol{a}^{(i+1)} = \boldsymbol{a}^{(i)} - \eta \nabla E_i\left(\boldsymbol{a}^{(i)}\right). \tag{2.25}$$

As this formula does not require knowledge of the entire data set when appropriately selecting E_n, it can be learned online or in mini-batches.

2.7 Dimensionality Reduction

In general, dimensionality reduction (DR) is a method for transforming objects living in a so-called *ambient* space (i.e. the original space) to some lower-dimensional *latent* space. The underlying idea is that the data in the ambient space is not truly that high-dimensional, but rather spans some lower-dimensional sub-space – so-called *manifolds* – or can be approximated as such. Dimensionality reduction thus tries to reduce the amount of redundancy by projecting or reducing the data down from the ambient space into the latent space, while preserving certain properties. These might vary, and typically are a tradeoff between preserving local and global structure in the data [38, pp. 1–2].
This section shall present two practical approaches, namely nonprobabilistic *principal component analysis* (PCA), based on Bishop [32, pp. 561ff], and the newer *uniform manifold approximation and projection* (UMAP) from 2018, based on the original paper by *McInnes et al.* [38].

2.7.1 Principal Component Analysis

PCA can be formulated as a projection into a lower-dimensional space which minimizes the error introduced by that transformation. Let us consider N data points $\boldsymbol{x}_n \in \mathbb{R}^D$ in the D-dimensional ambient space. Introducing some complete orthonormal basis $\{\boldsymbol{u}_i\}$ for $i = 1, \dots, D$, we can represent the data points as linear combinations of that new rotated basis:

$$\boldsymbol{x}_n = \sum_{i=1}^{D} \alpha_{ni} \boldsymbol{u}_i. \tag{2.26}$$

As we want to reduce the number of dimensions down to the laten space dimensionality called M, we first split this representation into two parts:

$$\boldsymbol{x}_n = \sum_{i=1}^{M} z_{ni} \boldsymbol{u}_i + \sum_{i=M+1}^{D} b_{ni} \boldsymbol{u}_i. \tag{2.27}$$

Now setting $b_{ni} = b_i$ for all $i = M+1, \dots, D$, we obtain an approximation $\tilde{x}_n$ of $\boldsymbol{x}_n$:

$$\boldsymbol{x}_n \approx \tilde{x}_n = \sum_{i=1}^{M} z_{ni} \boldsymbol{u}_i + \sum_{i=M+1}^{D} b_i \boldsymbol{u}_i. \tag{2.28}$$

The goal of PCA is to reduce the mean squared error E of that transformation, given by

$$E = \frac{1}{N} \sum_{n=1}^{N} \|\boldsymbol{x}_n - \tilde{x}_n\|^2. \tag{2.29}$$

It can be shown, that the error E can equivalently be written using the covariance matrix $\boldsymbol{S}$ of the original data as follows [32, p. 564]:

$$\begin{aligned} E' &= \sum_{i=M+1}^{D} \boldsymbol{u}_i^T \boldsymbol{S} \boldsymbol{u}_i, \quad \text{s.t.} \\ \boldsymbol{u}_i &\neq \boldsymbol{0} \quad \text{f.a. } i = 1, \dots, D. \end{aligned} \tag{2.30}$$

The empirical covariance matrix is given by

$$\boldsymbol{S} = \frac{1}{N} \sum_{n=1}^{N} (\boldsymbol{x}_n - \bar{\boldsymbol{x}}_n)(\boldsymbol{x}_n - \bar{\boldsymbol{x}}_n)^T. \tag{2.31}$$

As it turns out, minimizing that loss function E' amounts to the calculation of the eigendecomposition $\boldsymbol{S}\boldsymbol{u}_i = \lambda_i \boldsymbol{u}_i$, and minimizing the eigenvalues λ_i defining the projection:

$$E'' = \sum_{i=M+1}^{D} \lambda_i. \tag{2.32}$$

Intuitively, this means minimizing the amount of distortion introduced by the projection, which in turn means that the variance inherent to the original data gets maximized in the new latent space. PCA can be computed very efficiently.

2.7.2 Uniform Manifold Approximation and Projection

UMAP builds upon firm and rather advanced mathematical foundations to justify the assumptions and design decision which went into the design of the procedure. McInnes et al. make extensive use of *Riemannian algeba* and *fuzzy simplical sets* to provide a rigorous and mathematically sound theory, exceeding what is comprehensible in a computer science bachelor's thesis. Thus, only the general intuition of the algorithm shall be briefly reproduced in this section.[1]
The first step is the generation of a neighboring graph, which aims at capturing what points are locally connected on the manifold we are trying to find. As the data is in general not uniformly distributed on the manifold, some normalization must be applied. To this end, a fuzzy logic based approach is taken, intuitively modeling the probability of all points being connected to some given point. The resulting neighbor graph then contains asymmetric weights, which are turned into a uniform measure by fuzzy union. That can be interpreted as the probability of the existence of at least one edge between two given points.
The next step is to embed that weighted undirected graph into the desired lower-dimensional representation. This is done by the minimization of the so-called cross entropy, which is given by a sum over all edges $\boldsymbol{E}$. The resulting loss function L can be seen as inducing a force directed graph layout, with the following components

$$L(\boldsymbol{E}) = \sum_{e \in \boldsymbol{E}} \left[\underbrace{w_h(e) \ln\left(\frac{w_h(e)}{w_l(e)}\right)}_{\text{attraction}} + \underbrace{(1-w_h(e)) \ln\left(\frac{1-w_h(e)}{1-w_l(e)}\right)}_{\text{repulsion}} \right]. \tag{2.33}$$

In this formula, $w_h(e)$ refers to the weight of some edge e in the higher dimensional ambient space, which is given by the normalized weights derived above. The weight $w_l(e)$ in the lower dimensional latent space is induced by the familiar Euclidean norm between the data points connected by e.
The actual implementation of the UMAP algorithm then makes use of various optimizations, like sequential gradient descent (cf. section 2.6.2) for more efficient optimization iterations, and by only considering the nearest neighbors in that graph optimization problem. The resulting method turns out to be very performant and scalable even for large numbers of data, ambient and latent dimensions, though not as fast as PCA.

2.8 Related Work

This section shall outline related work this thesis is build on. Firstly, a general discussion of different gesture classification systems is provided, secondly an alternative time warping approach is presented and lastly different available gesture data sets are reviewed.

2.8.1 Approaches to Gesture Classification

In 2015 *Lo Presti & La Cascia* published a survey on the use of skeleton data for gesture classification [3], where they develop a categorization of the various approaches which have been published and give an overview over the most successful ones of each type. They also give an overview over the data sets being used most often, which helped in composing the overview in section 2.8.3.
The taxonomy introduced in the paper is depicted in the figure 2.7. The major differentiation splits the data representations into three groups. The first group is called *joint-based representations*, which "extract alternative feature representations from the skeletons in order to capture the correlation of the body joints" [3, p. 134]. The second group consists of *mined joint based descriptors*, which aim at learning what subset of the joints is distinctive for a gesture. The third proposed category is the *dynamics-based descriptors*, which try to model the dynamics of the joint movement over time. The methods using a joint-based data representation can be further categorized into three sub-categories: *spatial descriptors*, *geometric descriptors*, and *key-pose based descriptors*. The idea of the first approach is that a gesture is sufficiently defined by either calculating the distance or the covariance matrix between all possible pairs of joints. The geometric descriptors use a sequence of transformation operations that either describe/approximate the movement of the skeleton in time or the relative positioning of subsets of joints to according supersets. The data representations in the third sub-category define characteristic key-poses, and a gesture is then approximated and identified by the sequence of the closest key poses.
Based on this, ProMPs can be classified as joint-based representations, and within them as a generative method using spatial descriptors.

[1] For readers of the paper lacking the deeper mathematical foundations required for the understanding of all technicalities, a more broadly comprehensible and rather visual description is given in the documentation of the reference implementation called `umap-learn` [39], which can be found here: `https://umap-learn.readthedocs.io/en/latest/how_umap_works.html`.

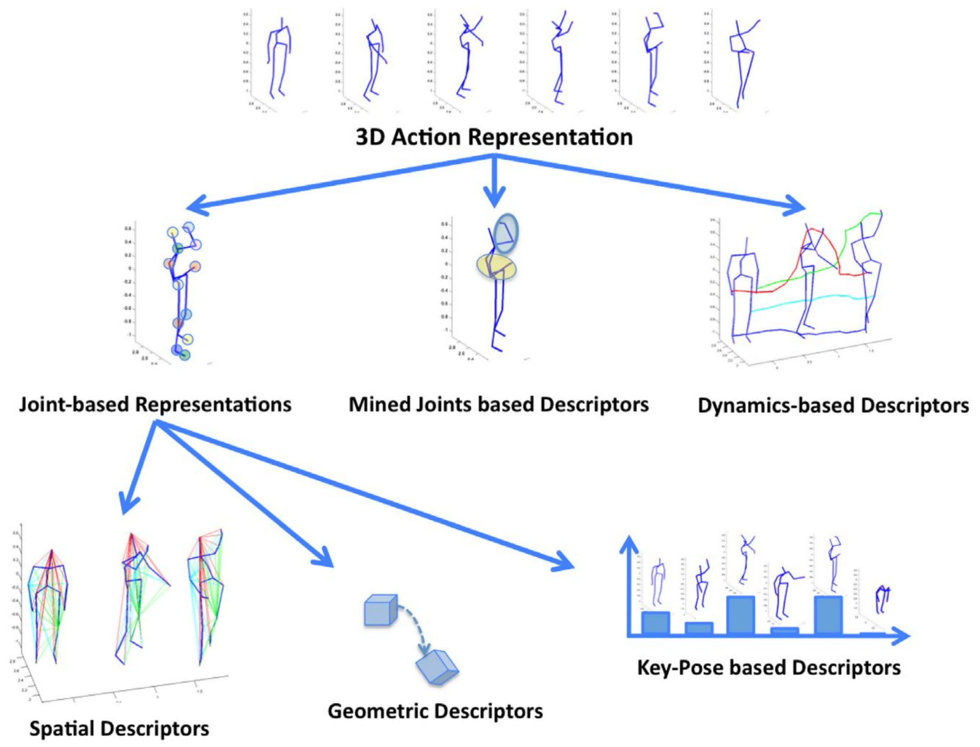

3D Action Representation
Joint-based Representations
Mined Joints based Descriptors
Dynamics-based Descriptors
Spatial Descriptors
Geometric Descriptors
Key-Pose based Descriptors

Figure 2.7: This image depicts different types of data representations used in skeleton-based gesture classification. The three main categories are given by *joint-based representations*, which directly extract features form the joints of a skeleton, *mined joint based descriptors*, which try to reduce the number of joints required for classification, and *dynamics-based descriptors*, which directly look at the change in the joint position, instead of special representations of static postures. The first type is further divided into *spatial*, *geometric* and *key-pose* based descriptors. Adapted from Lo Presti & La Cascia [3, p. 135].

Many different approaches have been proposed to perform gesture recognition, and many publications are available. This section will give a short overview over existing approaches following the survey of *Han et al.* from 2017 [40]. As motivated in section 2.8.3, special attention will be given to systems performing well on the *MSR Action3D* gesture data set (cf. section 2.8.3).

The best reported score on that data set is achieved by *Wang et al.* [41] at an accuracy of 96.9%. They improve on the previously used covariance matrices by using kernel matrices, showing that it performs superior to existing methods. The algorithm by *Cavazza et al.* [42] performs similarly well, and uses covariance matrices of the skeleton data too. Applying the kernel trick (cf. Bishop [32, p. 291f]) to this problem analytically allows them to also learn non-linear correlations between features. The algorithm of *Meshry et al.* [43] not only classifies gestures, but also detects whether some action is even taking place, while still being 96.1% accurate on the same data set. They represent gestures as bag-of-gesturelets (BoG), i.e. short parts of gestures and learn a histogram of their occurrence per class as the basis of their algorithm. They then introduce a novel method to search these descriptors efficiently in real-time, called Efficient Linear Search (ELS). The system proposed by *Du et al.* [44] uses hierarchical recurrent neural networks (RNNs) to classify gestures. The skeleton is first divided into five parts, and is then fed through a tree-like structure of neural networks, finally ending at a perceptron. The majority vote of the latter over all time steps is used to classify gestures. *Chaaraoui et al.* [45] propose the use of an evolutionary algorithm to determine the best sequence of key-poses used in classification of the gestures. There are also approaches to fuse skeleton data with depth data, as for example presented by *Shahroudy et al.* [46]. A key feature of their method is taking advantage of the sparsity of the feature space of body joints, as not all joints are relevant all the time. A different approach is developed by *Vemulapalli et al.* [47], where poses are viewed as elements of a Lie group, and gestures therefore can be considered being curves in that group. That insight is then used to form the input for more traditional classifiers like Fourier temporal pyramids and linear SVMs, achieving 92.5% accuracy on *MSR Action3D*. The first step of the algorithm of *Cippitelli et al.* [48] is a feature extraction, reducing the joint space to some structure-preserving lower dimensional space. Key postures are subsequently extracted and classification is done by a one-vs-one multiclass SVM. The work of *Jung & Hong* [49] builds on providing a novel alignment and similarity framework called Enhanced Sequence Matching (ESM). A gesture is then decomposed into multiple movement primitives (MPs), which serve as the basis of the classification. This paper reports the best accuracy score of 96.8% on the *MSRC-12* data set (cf. *Fothergill et al.* [50]) of all reviewed publications.

2.8.2 Time Sequence Alignment with Local Optimization

In 2014, *Maeda et al.* [51, p. 530] proposed an alternative to the previous time alignment method DTW (see section 2.3). An implementation together with a more detailed and corrected explanation is available from Maeda [52].
The general idea is to warp the time in such a way that the resulting trajectory stays smooth, and thus does not require tuning of a maximum slope parameter as DTW does. Here, $\boldsymbol{\tau}(t)$ is the trajectory interpolated/evaluated at time t, with t_r^j being the reference time (i.e. the unwarped time) and t_w^j being the warped time at index j. The warped time is obtained by minimizing the following cost function:

$$C(\boldsymbol{\theta}) = \sum_{j=1}^{T} \left| \boldsymbol{\tau}(t_r^j) - \boldsymbol{\tau}(t_w^j) \right| \tag{2.34}$$

The points in time t_r^j and t_w^j can be calculated as follows, assuming that the time is scaled to be in $[0, 1]$:

$$t_r^j = \frac{j-1}{T-1} \; (const.) \tag{2.35}$$

$$t_w^j = \theta_0 + s^j \cdot t_r^j, \text{ with } s^j = \sum_{n=1}^{N} \phi_n\left(t_r^j\right)\theta_n \tag{2.36}$$

The N basis functions $\phi_n(t)$ for $n = 1 \ldots N$ are chosen to be smooth, for example by using gaussian functions (RBFs) $\phi_n(t)$ as used by ProMPs (defined in equation (2.3)).
The solution of the optimization problem is then obtained by optimizing the parameters $\boldsymbol{\theta} = \{\theta_0, \boldsymbol{\theta}_{1:N}\}$ on $C(\boldsymbol{\theta})$. This can be calculated using gradient descent as proposed in the paper or using an existing general purpose function optimizer. The optimization is initialized with $\theta_0 = 0$ and $\theta_n = 1$, for $n = 1 \ldots N$.
The resulting warped/aligned trajectory $\boldsymbol{\tau}(\boldsymbol{t}_w)$ stays smooth, as it is only a multiplication and composition/chaining of smooth functions.

2.8.3 Gesture Data Sets

In machine learning, data sets of (in our case) labeled examples are often required for both the learning phase and the validation and testing of the system. There are several action and gesture classification and/or segmentation data sets in use today. They usually provide skeleton data, RGB(-D) video data or both. For a comprehensive overview over available data sets, please refer to the specialized surveys of *Zhang et al.* [53] and *Ruffieux et al.* [54]. The two gesture recognition surveys by Han et al. [40] and Lo Presti & La Cascia [3] also report on existing data sets. Additionally, the ML data set databases *PMLB* [55] and *OpenML* [56] contain various suitable data sets as well.[2]
A goal of this thesis is the quantitative comparison of the system under development to other methods, which have already been benchmarked. Thus, the wide adoption of the benchmarking data set used for the evaluation of the methods is of highest importance. The surveys of Lo Presti & La Cascia [3, p. 141] and *Han et al.* [40, p. 19] show that the *MSR Action3D* data set (short *MSRA 3D*) is widely adopted. The data set was first shown and used by *Wanqing et al.* in 2010 [58]. It already provides skeleton data which can be obtained in screen and world coordinates, of which we will use the latter. There are also typical validation protocols associated with the data set, namely:

HS-V Half subjects to test, the rest for training;

3F-1:2 3 folds cross-validation 1⁄3 of the data to test, 2⁄3 for training and

3F-2:1 3 folds cross-validation 2⁄3 of the data to test, 1⁄3 for training

[3, pp. 138f] (cf. [58, p. 12]).

We will be using the HS-V procedure, as this is a good – yet simple – method to evaluate the cross-subject learning capabilities of the system.

2 There also exist non-academic resources, like lists of ML data sets on *Wikipedia* [57].

3 Own Approach

This section presents our approach at gesture classification, starting out with stating the problem being tackled. The general pipeline used for going from a motion being recorded to a prediction of a gesture is presented next. The preprocessing and validation data sets being used are also described here.

3.1 Problem Statement and Goal

We are generally looking for gesture recognition systems or algorithms with the following properties. They should:

1. classify gestures based on trajectories of joints;
2. be able to perform multiclass classification (but not multilabel classification);
3. generalize well to multiple (unseen) subjects;
4. be able to recognize unknown gestures as such (classify against unknown), at best providing some certainty score or probability;
5. allow for online learning, in order to adjust over time;
6. have a low latency, considering the computational cost or complexity and
7. have a low latency, considering the number of frames that are required for a (preliminary) classification.

This thesis will focus on representation-based ML approaches, also called symbolic approaches. Connections approaches like neural networks have recently shown to be quite generally applicable and also successful in gesture classification [59]. However, drawbacks like the large computational cost and need for comparatively large amounts of training data cause representation-based approaches to still be prevalent in this field [60, p. 1].
The goal of this thesis is to use ProMPs for gesture classification, combining existing and novel approaches to achieve the best possible results in a quantitative benchmark on typical action recognition data sets. As presented in section 2.4, ProMPs are promising since they fulfill all of the aforementioned criteria 1.-7. They are superior to *Dynamic Movement Primitives* (DMPs) [61] – which model movement as dynamic systems using differential equations – in case the demonstrations are noisy [62, pp. 531, 534]. This is likely the case in our scenario, since the perceived gestures are not recorded in an idealized lab with markers being tracked, but rather in the field using comparatively error prone trackers for example build on RGB-D cameras (see section 2.2).
The three large surveys on gesture classification using symbolic approaches by Lo Presti & La Cascia (2015) [3], Sargano et al. (2017) [60] and Han et al. (2017) [40] do not mention ProMPs and no other sources presenting an evaluation on typical data sets could be found. A reason for this might be that they have not yet been been benchmarked using typical gesture classification data sets, since that was not the primary intent of their development [29, p. 1f]. Neither have any suggested improvements on ProMPs been evaluated like that, but always solely on special data sets [9, 63].
This thesis aims to achieve the best possible classification results using ProMPs/GMMs, and to compare them to other classification systems. Different preprocessing methods shall be systematically evaluated as well, namely the ones presented in the sections 3.3, 3.4 and 3.6. Additionally, dimensionality reduction methods shall be investigated and a comparison with SVMs be carried out.

3.2 The Proposed Pipeline

The different steps in the gesture classification system being developed are outlined in figure 3.1. Starting out with some action being performed by a human like a caregiver or elderly person, a video is recorded as a sequence of RGB or RGB-D images. It is then transformed into a sequence of skeletons using some off-the-shelf skeleton extraction method. Preprocessing methods as well as dimensionality reduction are performed, which produces features then given to a classifier. In the learning phase it fits to the arriving data in an offline or online scenario, and in the validation phase it classifies them, providing a prediction of what action/gesture the human has performed.

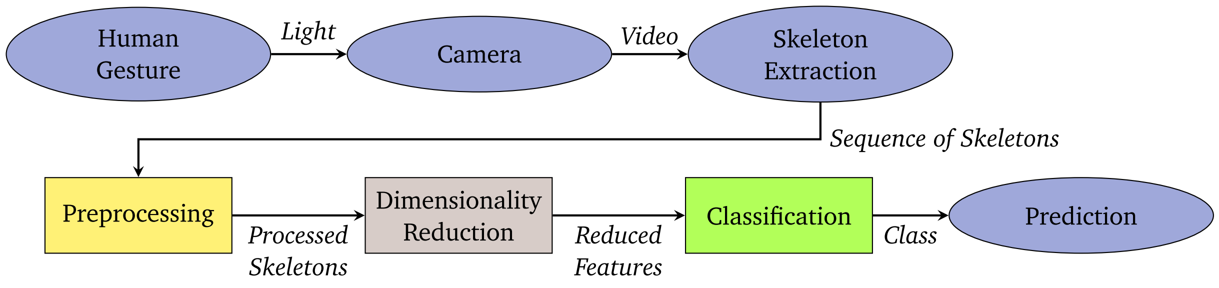


Figure 3.1: This figure depicts the general processing pipeline from the human performing a gesture until the classification result. Firstly, an RGB or RGB-D camera captures the movement of a human as a video. Some existing skeleton extraction method is used to obtain a sequence of poses, which are then preprocessed in some way. The result of that forms the input of some dimensionality reduction method and classification system, which will output some prediction.

The goal is the comparison of different preprocessing, dimensionality reduction and classification systems, which is depicted specifically in figure 3.2. Firstly, some normalization of the temporal and spatial scales is required, as described in section 3.4. Afterwards, the alignment of time sequences can be used as a preprocessing step, e.g. by first clipping away irrelevant parts of the gesture by using zero velocity clipping (section 3.5). Further alignment can be achieved with either spatial interpolation (section 3.6.1), DTW (section 2.3) or the local optimization approach (section 2.8.2 and section 3.6.4). These different methods and combinations thereof shall be evaluated, comparing it with no alignment being performed as a baseline. As a next step, dimensionality reduction can be achieved by transformation into weight space (section 2.4), followed by the probabilistic classification with GMMs presented in section 2.4. Alternatively, PCA and UMAP can be used as general purpose means of dimensionality reduction as well, and SVMs (section 2.6) pose an alternative to the probabilistic classification. The difference between these approaches shall be evaluated. These methods generally allow for both online and offline learning, and both schemes shall be evaluated and compared.

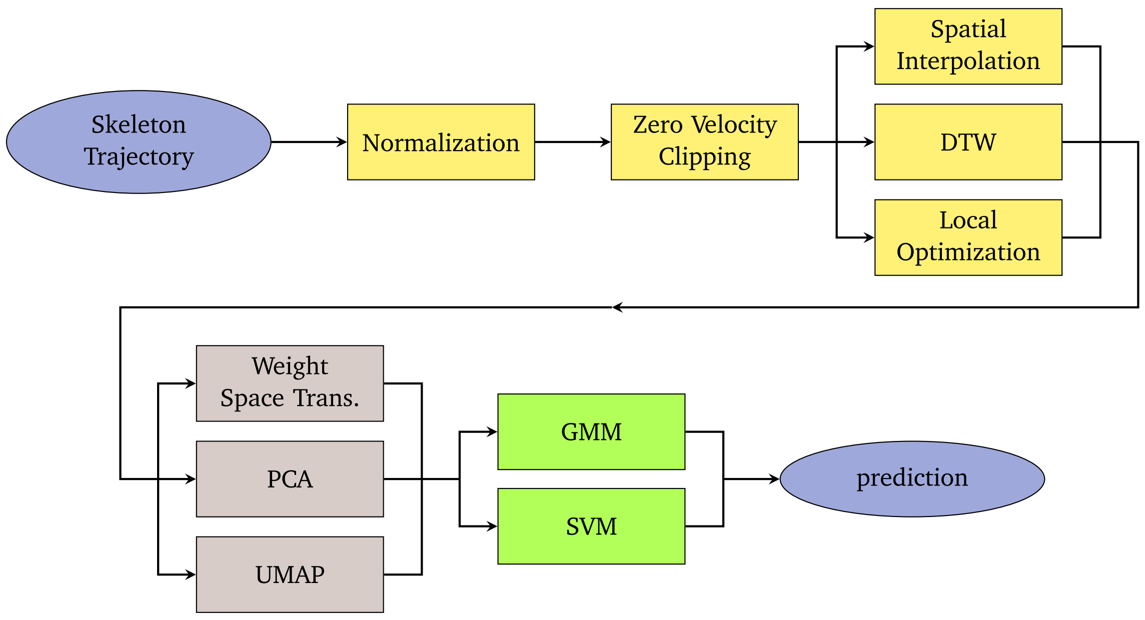


Figure 3.2: This graph shows the processing pipeline with details on the preprocessing and classification. First, some skeleton data is preprocessed by normalization and subsequent time sequence alignment. The alignment consists of zero velocity clipping followed by spatial interpolation, DTW or local optimization. Then, dimensionality reduction can be achieved by either the transformation into weight space as is typical for ProMPs, or by PCR or UMAP. Finally, an alternative to the probabilistic classification of GMMs is the use of SVMs, which can both be trained in offline and online fashion.

3.3 Dealing with Noise

Input data for the learner and/or classifier has to be preprocessed in various ways until it can be successfully used for training and later for classification.
Firstly, all real world data contains some degree of noise, which has to be dealt with. In our case, this is achieved by two means: For one part, the input data is smoothed by the transformation into weight space. This can be seen in the visual evaluation presented in section 4.2.3. ProMPs as such are very robust too, in that they compute the mean and primarily compare new data with it, taking the covariance into account. This reduces the effect of noise sufficiently, as no other means of smoothing are typically employed [9, 29].

3.4 Dealing with Different Sizes and Lengths of Recordings

The time is always normalized to lie within $t \in [0,1]$, and the spacial dimensions are normalized to $x, y, z \in [-1, +1]$. All trajectories with only a single joint start at the origin $(x, y, z)^T = (0, 0, 0)^T$ or the lower-dimensional equivalent.
If a new recording $\boldsymbol{\tau}'$ contains a gesture sampled at a different number of points in time T' (i.e. at a different frame rate) than some baseline recording $\boldsymbol{\tau}$ sampled at T points, both can still be transformed into the same weight space. This is achieved by (a) simply interpolating the data of the recording $\boldsymbol{\tau}'$ to T points, which can be done computationally independent for each dimension. The other approach (b) is to modify the transformation matrix $\boldsymbol{\Phi}$, which allows both transformations to result in a weight vector of the same length N. While $\boldsymbol{\Phi} \in \mathbb{R}^{T \times N}$ would be used for $\boldsymbol{\tau}$, the transformation $\boldsymbol{\Phi}' \in \mathbb{R}^{T' \times N}$ would be used for $\boldsymbol{\tau}'$. Both transformations would result in an equivalent weight vector of length N. The interpolation approach (a) was chosen because of its lower computational cost (cf. section 4.2.1).

3.5 Dealing with Surplus Recordings at the Start and End of Gestures

The idea of *zero velocity clipping* is to cut away all missing movement at the start and end of a recording such that only the relevant movement is retained. The implicit assumption of this method is that only larger movement is relevant for use in the ML task, i.e. in this case the classification of gestures.
The term *significant movement* is defined as the velocity being greater than some $\epsilon \in \mathbb{R}$. It can be calculated for a given trajectory $\boldsymbol{\tau}$ of length T and dimension N, represented as $\boldsymbol{\tau}_t = \begin{pmatrix} \tau_t^1 & \dots & \tau_t^N \end{pmatrix}^T \in \mathbb{R}^N$ for $T = 1, \dots, T$. We will now calculate the N-dimensional bounding box of the data points $\boldsymbol{\tau}_t$ as:

$$\boldsymbol{\tau}^{\min} = \begin{pmatrix} \min_{t \in 1,\dots,T} \tau_t^1 \\ \vdots \\ \min_{t \in 1,\dots,T} \tau_t^N \end{pmatrix}, \quad \boldsymbol{\tau}^{\max} = \begin{pmatrix} \max_{t \in 1,\dots,T} \tau_t^1 \\ \vdots \\ \max_{t \in 1,\dots,T} \tau_t^N \end{pmatrix}. \tag{3.1}$$

We further define some tolerance tol, which is the minimum velocity that is still relevant, measured in spatial difference per the entire trajectory, e.g. in meters per second if the trajectory was normalized to one second and measured in meters. From this, we can calculate ϵ as

$$\epsilon = \frac{tol \cdot \left\| \boldsymbol{\tau}^{\max} - \boldsymbol{\tau}^{\min} \right\|_2}{T} \tag{3.2}$$

and use it for clipping.

3.6 Dealing with Different Speeds of Execution

However, this does not solve the problem of the same gesture being executed a different speeds. A first subject might take 80 frames for picking up a bottle in a recording of length $T = 80$. A second subject might only take 60 frames for the same action after first waiting for 10 frames, but the entire recording would still be of same length T. The spatial interpolation method presented in the following section 3.6.1 can deal with such differences in a very basic form. Additionally, these speed variances and skews do not need to be constant in a single recording, but might change, especially for more complex gestures and actions. Thus, two methods have been presented to account for this, Dynamic Time Warping (DTW) in section 2.3 and Local Optimization in section 2.8.2. Note that in contrast to the equidistant interpolation method, these two are sequence alignment methods, and they align some trajectory to a reference. Equidistant interpolation simply transforms every trajectory independently into a more canonical representation.

3.6.1 Spatial Interpolation

Spatial interpolation transforms some trajectory $\boldsymbol{\tau} = \begin{pmatrix} \boldsymbol{\tau}_1 & \dots & \boldsymbol{\tau}_T \end{pmatrix}^T$ with dimension N and length T in such a way that the result is a trajectory where each point is equally spaced according to some distance function (cf. [9]). That distance is in this case based on the euclidean norm $\|\cdot\|_2$.
We start out by calculating the distances between all successive elements, by padding with a 0 for the first one:

$$\boldsymbol{\Delta}_t = \begin{cases} \|\boldsymbol{\tau}_t - \boldsymbol{\tau}_{t-1}\| & \text{if } t \leq 2 \\ 0 & \text{if } t = 1 \end{cases}, \text{ for } t = 1, \dots, T. \tag{3.3}$$

We then calculate the cumulated distances $\widetilde{\boldsymbol{\Delta}}$ as

$$\widetilde{\Delta}_t = \sum_{i=1}^{t} \Delta_t. \tag{3.4}$$

To interpolate, we view the calculated cumulative distances as the domain of a function $f : \mathbb{R} \to \mathbb{R}^N$, with $\boldsymbol{\tau}_t = f(\widetilde{\Delta}_t)$ for each $t = 1, \dots, T$. The interpolation is now obtained by interpolating that function to equally spaced temporal points, which results in a newly aligned trajectory $\boldsymbol{\tau}'$ reflecting the equal distances encoded in f:

$$\boldsymbol{\tau}'_t = f\left(\frac{t-1}{T-1}\right). \tag{3.5}$$

3.6.2 Continuous Dynamic Time Warping

DTW is a discrete solution of the problem of aligning time sequences. The result of the algorithm is a list of index pairs like $(1,0), (1,1), (1,2), (1,3), \dots$ matching sample points in the two possibly multidimensional sequences. This is not sufficient since we need continuos time sequences for the calculation of ProMPs.
Each index into the pair of sequences occurs at least a single time on each position in the tuples, but often more than once. We now reduce the list of matching pairs by eliminating all duplicate indices such that there are no two different matching pairs (a_1, b_1) and (a_2, b_2) with $a_1 = a_2$ or $b_1 = b_2$. In order for no systematic error to be introduced, the element closest to the center of the occurrences of the duplicates must be retained. Else the method does not produce good alignments and parts of the realigned sequence are sped up incorrectly for a short amount of time. We arbitrarily apply this filter for the first sequences indices prior to the second ones. We retain the elements in such a way that when some value occurs multiple times, we keep the element closest to the middle of the subsequence of duplicates. The final continuos trajectory is then obtained using interpolation, effectively resampling it to the original sample count.

3.6.3 Multidimensional Local Optimization

The approach presented in section 2.8.2 aligns the sequences by numerically solving an optimization problem. This approach is only presented for a 1-dimensional input (i.e. a single variable changing over time). In our case tough, we require the method to work on n-dimensional trajectories too, since even only a single joint in $\mathbb{R}^3$ already consists of three dimensions. To this end, the natural generalization of the cost equation (2.34) is used instead, which sums up the cost over all dimensions/components D. This results in the following formula, with $\boldsymbol{\tau}^k(t)$ being the k^{th} component of the trajectory τ evaluated at time t:

$$C'(\boldsymbol{\theta}) = \sum_{k=1}^{D} \sum_{j=1}^{T} \left| \boldsymbol{\tau}^k(t_r^j) - \boldsymbol{\tau}^k(t_w^j) \right| \tag{3.6}$$

While this generalization looks trivial in theory, it might have a big impact on the performance of the method. On the one hand it could result in larger computational costs. On the other hand it might also cause the method to fail to converge, or at least to greatly increase the number of iterations the solver needs to find a decent solution.

3.6.4 Proposed Local Optimization

That generalized local optimization did only run into local minima and was not able to correctly warp the time of some simple synthetic data used for a first evaluation. The problems are shown and discussed in detail in section 4.1.4. Thus, we came up with an an improvement on the calculation of the time warping $\boldsymbol{t}_w$. The idea is based on the idea that while the start of a sequence can be easily adjusted using θ_0, the end of the the warped sequence originally can not. When we are keeping the start of the warping constant, this is equivalent to adjusting the overall duration of the movement. The idea is to now simply introduce another scaling term θ_s which scales the entire movement uniformly, in addition to the existing s^j terms. This is in a way redundant, as the correct sequence of the parameters $\boldsymbol{\theta}_{1:N}$ could also represent that warping, but as with the introduction of θ_0 simplifies finding a solution of the optimization problem. The formula for calculating the time warping $\boldsymbol{t}_w$ then results in the following:

$$t_w^j = \theta_0 + \left(s_t \cdot t_r^j\right) * \theta_s \tag{3.7}$$

The solution of the optimization problem is obtained by now optimizing the parameters $\widetilde{\boldsymbol{\theta}} = \{\theta_0, \theta_s, \boldsymbol{\theta}_{1:N}\}$ on $C'(\boldsymbol{\theta})$ given in equation (3.6), resulting in the cost function $C''(\widetilde{\boldsymbol{\theta}})$. The new parameter θ_s is initialized to 1, indicating no scaling, neither faster nor slower.

Additionally, to aid with the convergence and to avoid any warping beyond a plausible point, we introduce bounds for the optimization problem, as is listed in table table 3.1 together with the initial values.

The evaluation of these proposed improvements is shown in section 4.1.4.

Parameter	Initial value	Lower bound	Upper bound
θ_0	0	0	0.9
θ_s	1	0.1	10
$\boldsymbol{\theta}_{1:N}$,	$(1,\ldots,1)$	$(0,\ldots,0)$	$(2,\ldots,2)$

Table 3.1: Optimization parameters $\boldsymbol{\theta}$ for time sequence alignment using local optimization, with initial values and upper and lower bounds. The parameters θ_0 and $\boldsymbol{\theta}_{1:N}$ are used in both the original and the proposed local optimization. The parameter θ_s is only used in the proposed optimization.

3.6.5 Framework for Time Sequence Alignment

The above time sequence alignment methods allow aligning one sequence to another. But what we actually require is a method to align all trajectories belonging to some class to some trajectory which represents that class well enough to allow for a meaningful alignment.

For offline learning, this is achieved by first calculating the barycenter of all trajectories of some label, and then aligning all trajectories to that barycenter. The barycenter is intuitively the average trajectory of that label, and does not have to be present in the actual input data. Soft-DTW by *Cuturi & Blondel* [7] was chosen as a start-of-the-art method for such barycenter calculation. It is often used for clustering and performs very well on the domain of time series, which is was developed for [7, pp. 894, 899]. An implementation is freely available in `tslearn` [8]. It better retains the shape than previous algorithms, as figure 3.3 shows. Future work could also build on the possibility of blending multiple trajectories using that method.

The most simple method for online learning would be to always remember the first example for each label an align all subsequent trajectories to than one. However, that first sequence might not be a good representative of the time sequence, and as such it could be updated incrementally, i.e. by using weighted Soft-DTW with decreasing weights for the newly added trajectories. For this thesis, the same procedure was chosen for the online learning as was done for the offline learning, in order to simplify the implementation.

For classification, we can now build on the fact that we learned a barycenter trajectory per class/label for both offline and online learning. Each demonstration we want to classify is now aligned to each barycenter, and the prediction with the probability is calculated. The prediction with the highest probability is finally selected. In the case of the classification with GMMs, this uses the non-normalized probability, which yielded better results in section 4.2.5.

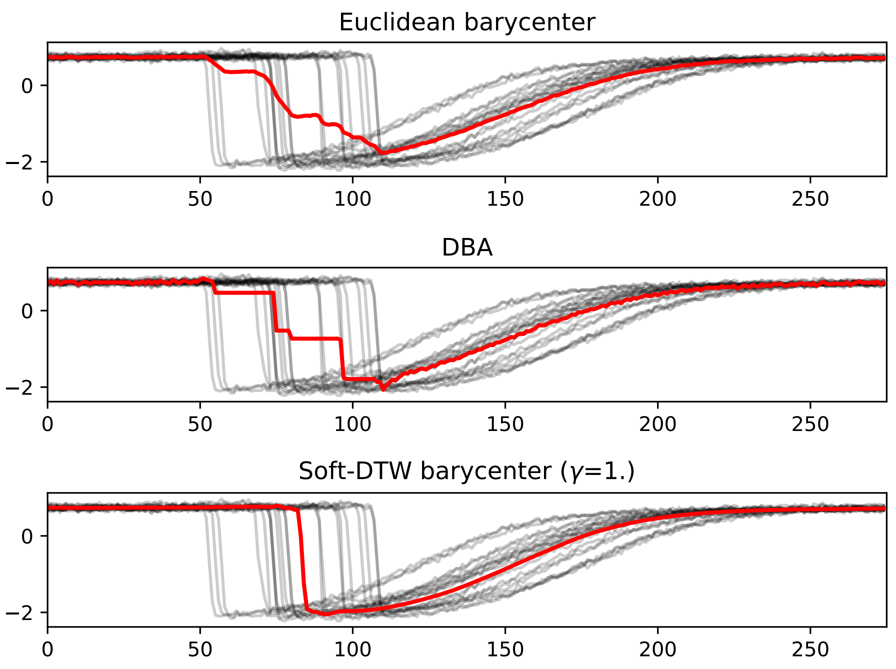


Figure 3.3: Illustration of different methods used for trajectory barycenter calculation. It compares Euclidean, DBA and Soft-DTW barycenter calculation. Euclidean is simply the arithmetic mean over all trajectories for each point in time. DBA refers to DTW Barycenter Averaging [6]. Soft-DTW [7] is the method used in this thesis. Adapted from `tslearn` [8].

3.7 Using other Learners in the Weight Space

An approach not yet explored by previous papers is the use of other supervised ML techniques on the weight space of ProMPS. The survey of Lo Presti & La Cascia (2015) concludes, “on this kind of data and for the problem of 3D skeleton-based action classification, discriminative approaches outperform the generative ones” [3, p. 134]. Thus, this thesis will explore the use of discriminative ML algorithms applied to the weight space. The weight space is chosen since there already were many attempts at using such systems on the skeleton data directly, and this approach should now benefit from the dimensionality reduction given by the transformation into weight space. Learning a SVM directly in the trajectory space will be performed as well in order to evaluate the benefit of the weight space transformation for SVMs. Other works have already been successful by using SVMs for gesture recognition, as summed up in section 2.8.1. We require an approach that allows for online learning, we will be using SVMs built on sequential gradient descent (SGD) as introduced in section 2.6.2.

The weight space transformation does reduce the dimensionality significantly. For example, representing a gesture with a skeleton consisting of 20 joints in three-dimensional (cartesian) space results in features at some point in time lying in $\mathbb{R}^{60}$. Transforming these trajectories into a weight space with $N = 10$ weights per dimension results in a weight space which is still of dimension $60 \cdot 10 = 600$. But compared to the number of dimensions of the raw trajectory – which is for example $60 \cdot 80 = 4800$ for 80 sample points in time – it is reduced by a lot. However, SVMs are known to be able to cope well with high-dimensional features [64, p. 1083], and it remains to the evaluation in section 4.3 to determine the benefit of the transformation.

3.8 Making use of Dimensionality Reduction

GMMs generally do not scale very well to higher dimensions, as firstly the calculation of the covariance matrix becomes increasingly unstable as we will see in section 4.2.2, and secondly because of the *curse of dimensionality*. That phenomenon describes the effect of uniformly random sampled vectors from a high dimensional space having a high probability of being nearly orthogonal [32, p. 33ff]. Fortunately, however, most data sets do not actually live in very high dimensions, as there is a lot of redundancy in the data points. Thus, we can employ dimensionality reduction methods as presented in section 2.7 in order to reduce the number of dimensions used by the classifier.

There are in principle two ways of reducing the dimensionality, independent from the actual method being applied:

1. The dimensionality reduction is used to reduce the representation of a single pose, e.g. to reduce a skeleton from being of dimension 60 down to for example 5. A trajectory results in a sequence in that new latent space.
2. The space of high dimensionality is considered to contain the entire trajectory, i.e. the dimensionality of skeleton (like 3 or 60) times the number of sample points or weights per each dimension. That entire space is then reduced into a single latent space vector per demonstration.

Both are in principle possible, and both can be combined with a weight space transformation, where the first approach would be applied either before or after the weights are calculated and the second one has to be applied afterward. Both shall be evaluated in chapter 4.

3.9 Gesture Data Sets used in the Evaluation

A main goal was the quantitative and comparable benchmarking of the implemented methods, which requires data sets of gestures and actions being performed. An overview over the data sets used in the evaluation is given in table 3.2.
To validate the basic implementation, a very simple data set *letters* of the symbols "1" and "9" each has been created by a single person using a standard pointing device (mouse) on a computer screen. The actions were performed by a single actor and the data contains x and y coordinates per each point in time.
To go beyond that tiny data set, a second data set called *salad* with 5 gestures and a total of 1050 demonstrations from a kitchen assistance scenario was recorded. The type of gestures are taken from Koert et al. [9], and were recorded in a lab using an *OptiTrack* triangulation system based on infrared markers. Figure 3.4 shows the markers while a subject is performing the "board" gesture. As of this writing, seven subjects have performed all five gestures for 30 times each. Some of the gestures were performed about one week after the first portion, to increase the variance introduced by the actors. For the same reason, the recordings were performed round-wise, i.e. an actor performed three times the first action, three times the second one and so forth. On the first session, 7 rounds were performed, and one week later 3 more rounds, totalling at $3 \cdot 5 \cdot (7+3) = 150$ actions performed in total per each of the seven subjects.
The *MSR Action3D* data set was introduced in section 2.8.3. It is a typical data set for skeleton based gesture classification that allows different methods to be compared as many authors report scores on that very data set. It is however very challenging, as the dimensionality far exceeds the range where ProMPs typically work well, as we will see later in the experimental evaluation in section 4.2.5. Thus, the data set manually reduced down to only the three spatial coordinates of the two wrists and the head was used as a fourth data set.

Name	# Demonst.	# Classes	# Subjects	# Dimensions	Source	Suggested Validation Method
letters	20	2	1	2	own	LOO-CV
salad	1050	5	7 (14)	3	own, scenarios from [9]	4F-CS-CV
MSR Action3D	547	20	10	$20 \cdot 3 = 60$	[65]	2F-CS-CV
MSR Action3D – reduced	547	20	10	$3 \cdot 3 = 9$	[65], processed	2F-CS-CV

Table 3.2: Data sets used for evaluating the classification methods. The number of demonstrations is the total number of trajectories in the data set, each of the given dimension. They are labeled with the given number of classes and were performed by the given number of actors. The first two data sets were recorded by us, with the scenarios for salad stemming from Koert et al. [9]. The salad actions were performed by 7 actors, but at two different days about a week apart, so they might show variance in the gestures more similar to 14 subjects. The third data set was very high dimensional, and got reduced to the two wrists and the head joints for speed of computation and numerical stability.

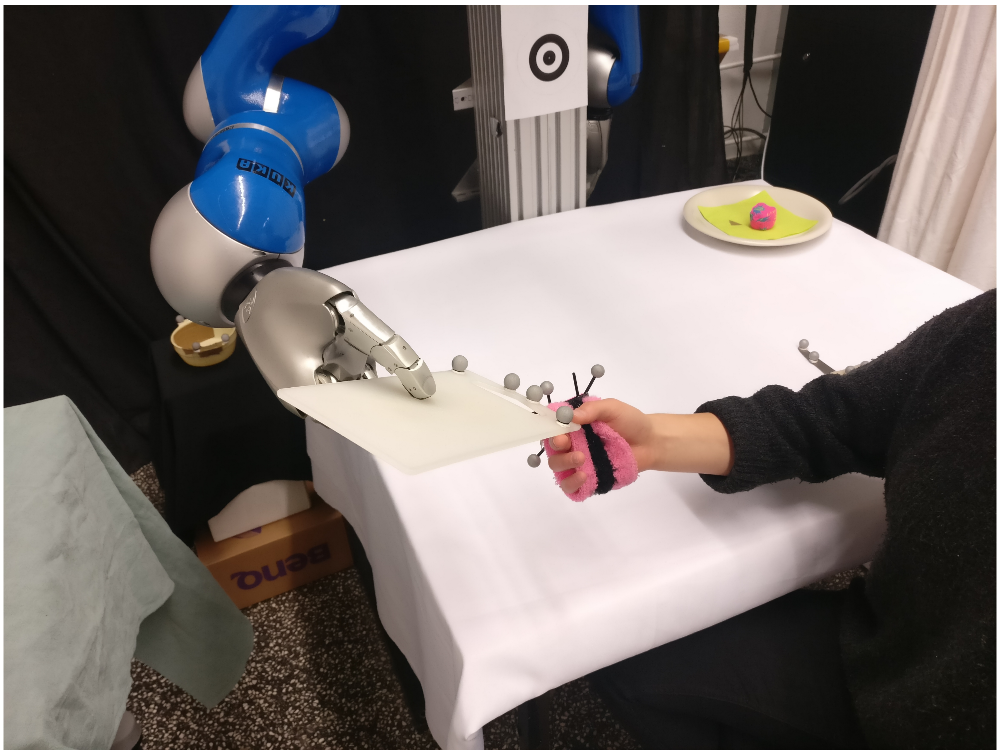


Figure 3.4: This photo was taken while a subject is performing the "board" gesture for recording the *salad* data set. The grey markers were used to track the position of the hand, and the robot was moved manually as it would in the real scenario in order to observe realistic behavior of the subjects.

4 Experiments

This section provides an empirical evaluation of the methods presented in the previous sections. Namely, we will first evaluate which time sequence alignment method provides the best preprocessing outcome. Then, different classification methods will be tested in combination with the most successful preprocessing methods. For this purpose, GMMs/ProMPs and SVMs are evaluated individually and compared at the end. Finally, dimensionality reduction using PCA and UMAP is investigated.
This section also includes details on the implementation such as software packages serving as a foundation for the system used in this evaluation. The implementation of all methods was accomplished in `Python 3` and makes heavy use of the `numpy` [66]. It also generally builds on the linear algebra routines of `scipy` [67] and the data handling framework `pandas` [68]. Wherever interpolation was required (e.g. for continuous DTW and local optimization), the third-order polynomial spline implementation in `scipy.interpolate.CubicSpline` was used.
Whenever performance metrics are given, they were obtained from running the benchmarks on a laptop with an *Intel Core i5-6300U* processor @ 2.40 GHz (two full cores, with hyper-threading) and 16 GB memory.

4.1 Time Sequence Alignment

The implementation and proposed improvements to time sequences alignment were first tested on a synthetic movement generated by the following formula on $t \in [0,1]$ (which was also used previously in figure 2.5):

$$x(t) = -\frac{1}{2}\left(\sin(2\pi t) - \frac{1}{2}\cos(4\pi t)\right) + 0.4t\ . \tag{4.1}$$

In order to test the re-alignment, the methods were benchmarked against two skewed versions of the function and the ability to realign it with the original $x(t)$ when sampled at 150 points in time was plotted. All three functions are shown in figure 4.1.

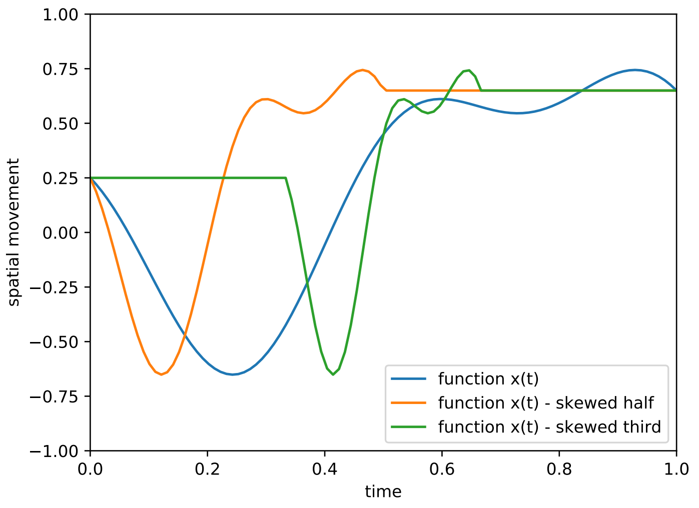


Figure 4.1: The synthetic movement generated by $x(t)$ from equation (4.1) on $t \in [0,1]$ and the two skewed versions "half" and "third". These functions sampled at equidistant points in time t will be used for a first evaluation of the time sequence alignment methods.

Afterwards, the time sequence alignment was each tested qualitatively on a trajectory of the *letters* and *salad* data set. This provided some visual insight into the performance. Finally, the methods are tested quantitatively against a batch of real data from the *salad* data set and evaluated using the mean squared error.

4.1.1 Zero Velocity Clipping

The simplest time sequence alignment method is zero velocity clipping as described in section 3.5. The results of this method, when applied on real data, can be seen in figure 4.2. We can see that the movement before the first little bump and after the following large bump was removed, and everything in between has been retained. This shows that it is successful in clipping away areas with no movement, i.e. with *zero velocity*. As we can see in this example, it can result in drastic transformations from input to output.

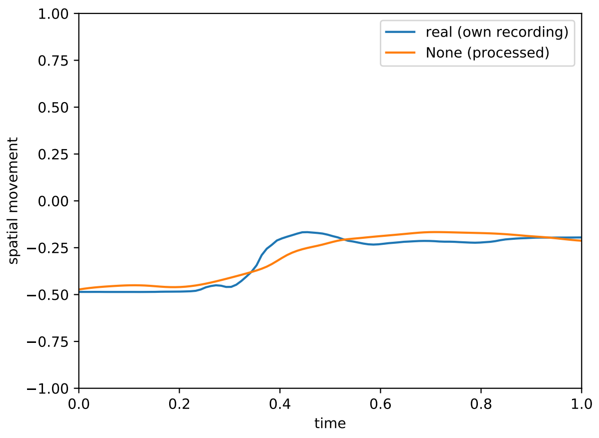


Figure 4.2: Visualization of the effect of zero velocity clipping on real recorded data from the *salad* data set. We can see that most of the start and end of the recording were removed as there was too little movement for it to be retained.

4.1.2 Spatial Interpolation

The effect of the spatial interpolation described in section 3.6.1 is shown in figure 4.3. We can visually validate the result and can confirm the expected behavior: Areas with no movement at the start and end disappear from the result and sections with faster movement get stretched out accordingly. The method works both on the synthetic data and real recorded data.

The eventual goal of the method is to align two different time sequences, such that demonstrations of the same gesture result in transformed features as similar as possible. Figure 4.4 shows the result of aligning skewed synthetic data as well as two different real recorded motions. As we can see, the algorithm is successful in that the resulting processed gestures are (visually) a lot more similar than the previous ones. As was to be expected and is now shown by the alignment of the synthetic data, the algorithm successfully clips away any preceding or following lack of movement.

4.1.3 Dynamic Time Warping

The original DTW algorithm presented in section 2.3 is a discrete solution, which already works well, as can be seen in figure 4.5a and 4.5b. The implementation of discrete DTW builds on the function `tslearn.metrics.dtw_path` from the `tslearn` package [8] without setting any constraints. It outputs a list of indices which map corresponding points on the functions/trajectories being sampled at discrete points in time. That is subsequently used to obtain the alignment. Continuous DTW then simply interpolates based on the obtained discrete alignment as described in section 3.6.2.

The results using continuous DTW are shown in figure 4.5c and 4.5d. It does improve significantly on the discrete output of DTW, as can be especially seen with the large stretching required for aligning the synthetic data. However, the algorithm does not preserve the overall shape of the trajectory, as can be observed on the real data. The height difference between the new demonstrations and the reference cannot be eliminated by DTW and as such, it tries to minimize the distance to the reference as far a possible. These highly non-linear time warpings are prevented by the local optimization approach being evaluated next. That very problem was also the motivation for the use of the local optimization approach by Meada et al.: “[It] not only avoids the tunning [*sic*] of [the DTW slope constraint] parameter but also preserves the overall shape of the trajectory” [51, p. 530] (cf. Sakoe & Chiba [26] for details on that constraint).

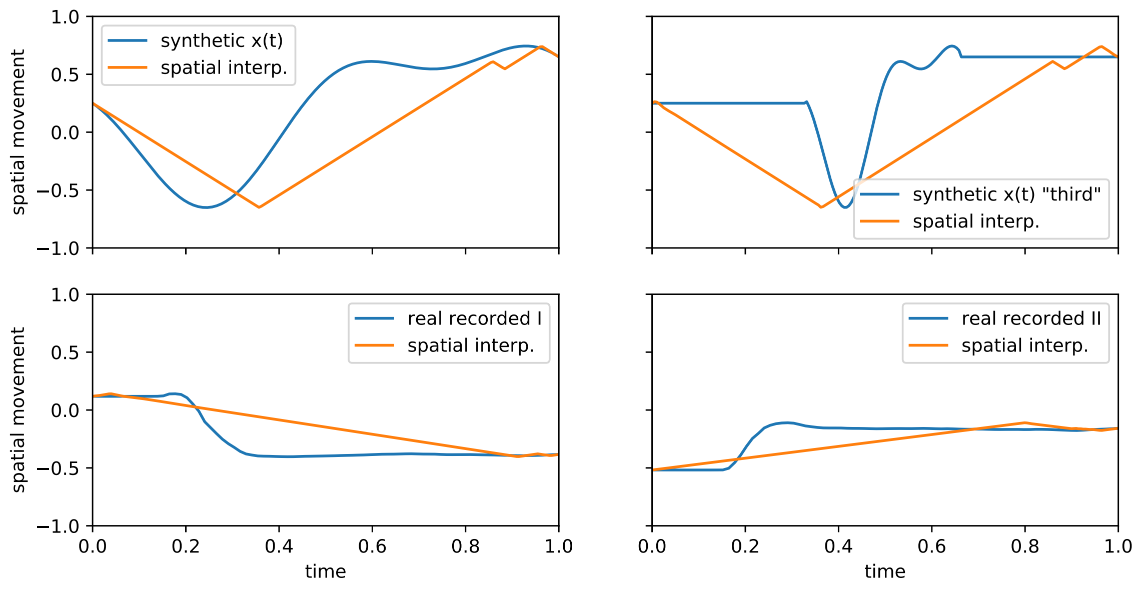
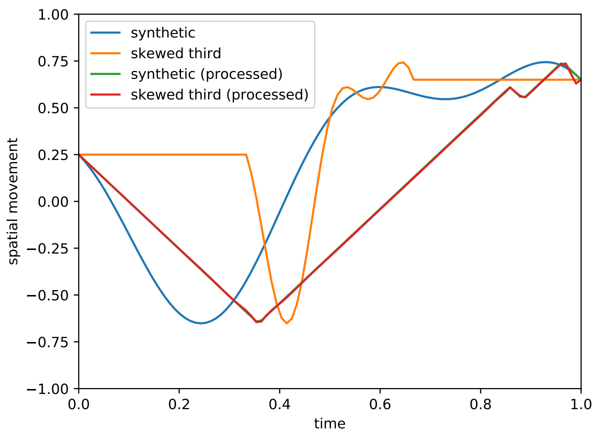
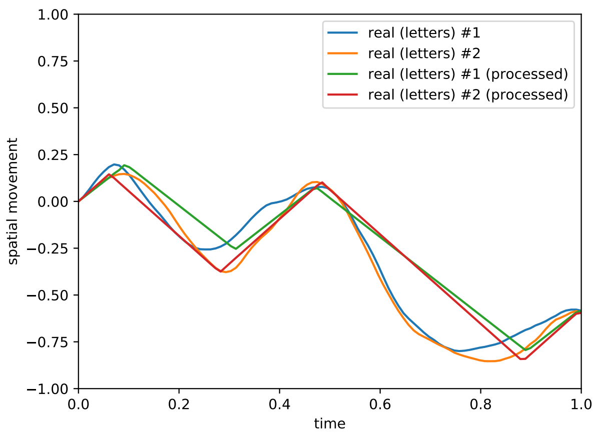


Figure 4.3: Illustration of the effect of the spatial interpolation on different trajectories. The top row shows the result of interpolating the synthetic function $x(t)$ from equation (4.1) and the skewed version "third" thereof. The bottom row shows the effect on real data of the *salad* data set, where *real recorded I* & *II* refer to the y- and z-axis of the same demonstration.

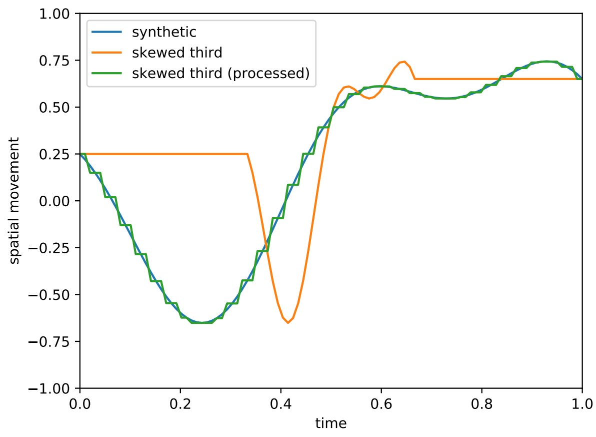


(a) Alignment of the synthetic skewed function "third" back to the original function. The alignment is so close that the original processed function is hidden beneath the newly aligned one.

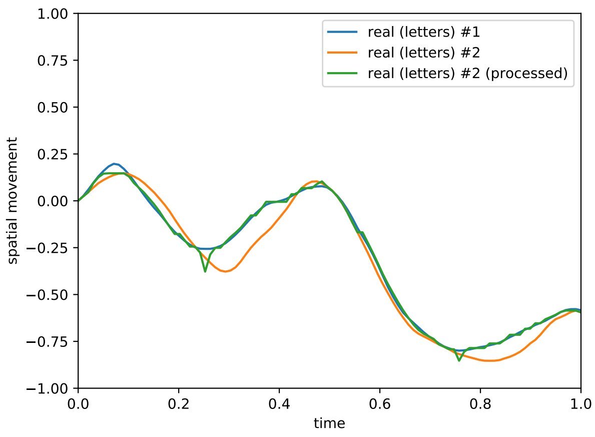


(b) Alignment of two different demonstrations of the "standup" gesture of the *salad* data set.

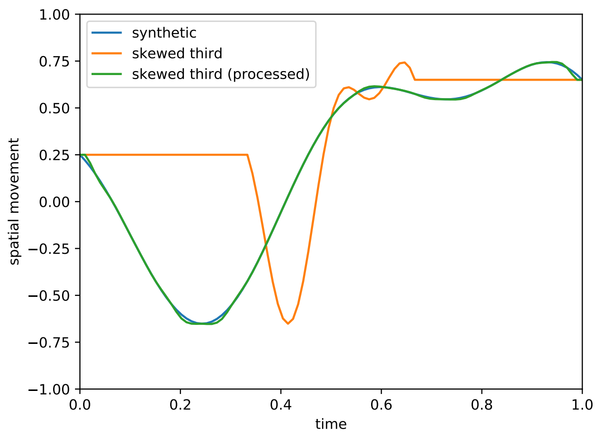
Figure 4.4: Alignment of synthetic data as well as two different demonstrations of a real recorded gesture using spatial interpolation. Both the original reference (blue) and the trajectory to be aligned (orange) have been processed, resulting in the green and red trajectories respectively.



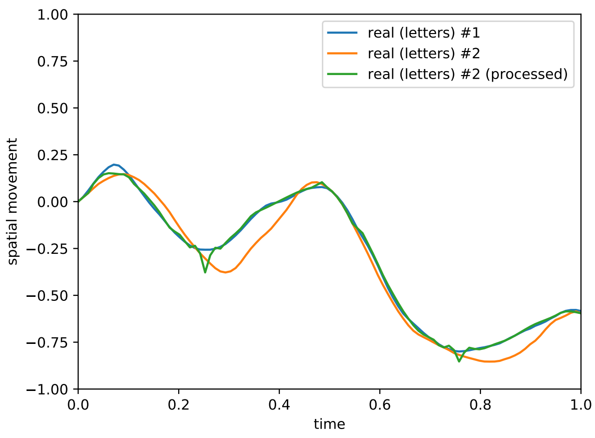
(a) Alignment of the synthetic skewed function “third” back to the original function using discrete DTW.



(b) Alignment of two different demonstrations of the “standup” gesture of the *salad* data set using discrete DTW.



(c) Alignment of the synthetic skewed function “third” back to the original function using continuous DTW. The alignment is so close that the original processed function is hidden beneath the newly aligned one.



(d) Alignment of two different demonstrations of the “standup” gesture of the *salad* data set using continuous DTW.

Figure 4.5: Alignment of synthetic data as well as two different demonstrations of a real recorded gesture using both discrete DTW in the first row as well as continuous DTW in the second row. The highly non-linear time warping can be nicely observed in the two plots on the right, resulting in spikes in the trajectory.

4.1.4 Local Optimization

Both the original local optimization presented in section 2.8.2 and the new optimization method proposed in section 3.6.4 require an optimization routine in order to minimize the cost function C (or the proposed changes to it) over $\boldsymbol{\theta}$. To this end, the general purpose routine `scipy.optimize.minimize` provided by `scipy` [67] was used. For the original local optimization, the `'BFGS'` method was used, which implements the *Broyden-Fletcher-Goldfarb-Shanno algorithm* (BFGS) described in more detail in *Nocedal & Wright* [69]. For the proposed method, bounds were introduced and thus the method `'L-BFGS-B'` was selected, which stands for ***l**imited-memory **b**ound-constrained BFGS*, first published by *Byrd et al.* [70] and implemented by *Zhu et al.* [71]. Both are the default algorithms for such scenarios in `scipy`.
A global optimization using `scipy.optimize.shgo` for both of the problems was attempted, but the time required for their convergence was so large that they could not be used in practice or even evaluated qualitatively. That global optimization method is an implementation of the algorithm of *Endres et al.* [72].
Results of the original optimization procedure is shown in figure 4.6a and 4.6b. It can be seen that it fails to converge to a good warping in both cases. In the case of the synthetic data, it manages to correctly warp the first half of the sequence, while in the case of the real data it fails to provide a good solution in most parts of the sequence. Especially the results of the synthetic data motivated the development of the proposed optimization method, which is shown in figure 4.6c and figure 4.6d. While it manages to nicely align the synthetic data, it completely fails to align the real trajectory. Thus, this short qualitative evaluation is inconclusive and subsequent quantitative analysis is required.

4.1.5 Quantitative Comparison

Figure 4.7 shows how different time sequence alignment methods compete against each other. The algorithms used in the comparison are described more detailed in the following paragraph:

none (baseline) corresponds to no alignment being done.

DTW (discrete) is the unmodified DTW as described in algorithm 2.1.

DTW is the continouos DTW algorithm described in section 3.6.2.

Local opt. (original) N=8 describes the original local optimization with the small generalization to multiple dimensions shown in section 3.6.3 with $N = 8$ weights being used for $\boldsymbol{\Phi}$.

Local opt. (proposed) N=8 is the modified local optimization proposed in section 3.6.4 with the same number of weights.

Spatial interpolation refers to the method presented in section 3.6.1.

Zero Velocity Clipping is the transformation explained in section 3.5.

Zero Velocity Clipping + Spatial interpolation combines zero velocity clipping and spatial interpolation.

Zero Velocity Clipping + DTW combines zero velocity clipping and continuous DTW.

Zero Velocity Clipping + Local opt. (original) N=8 combines zero velocity clipping and multidimensional local optimization.

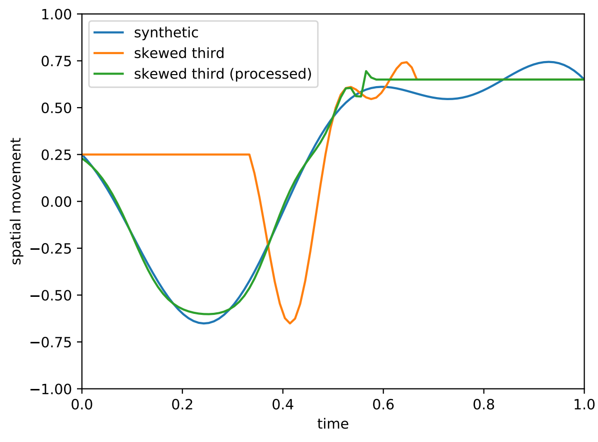
Zero Velocity Clipping + Local opt. (proposed) N=8 combines zero velocity clipping and the proposed local optimization.

Spatial interpolation + DTW combines spatial interpolation and continuous DTW.

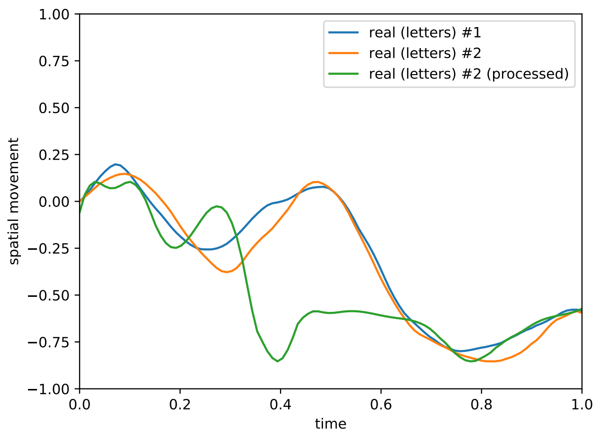
Spatial interpolation + Local opt. (original) N=8 combines spatial interpolation and multidimensional local optimization.

Spatial interpolation + Local opt. (proposed) N=8 combines spatial interpolation and the proposed local optimization.

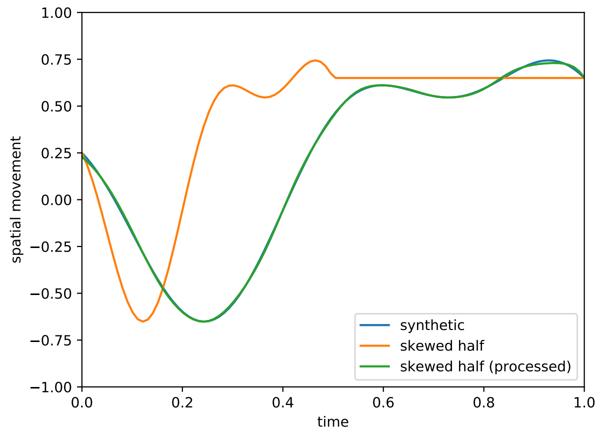
The results for each method were obtained from the *salad* data set, with 10 reference trajectories being selected as the first ones per label. All other trajectories of that label were then aligned to the reference and the resulting difference was used for the MSE shown in the plots. The trajectories were sampled at 100 points in time and are the three-dimensional



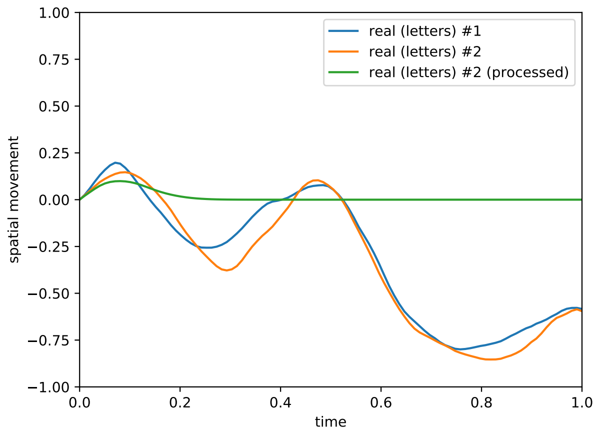
(a) Alignment of the synthetic skewed function “third” back to the original function using the original local optimization.



(b) Alignment of two different demonstrations of the “standup” gesture of the *salad* data set using the original local optimization.

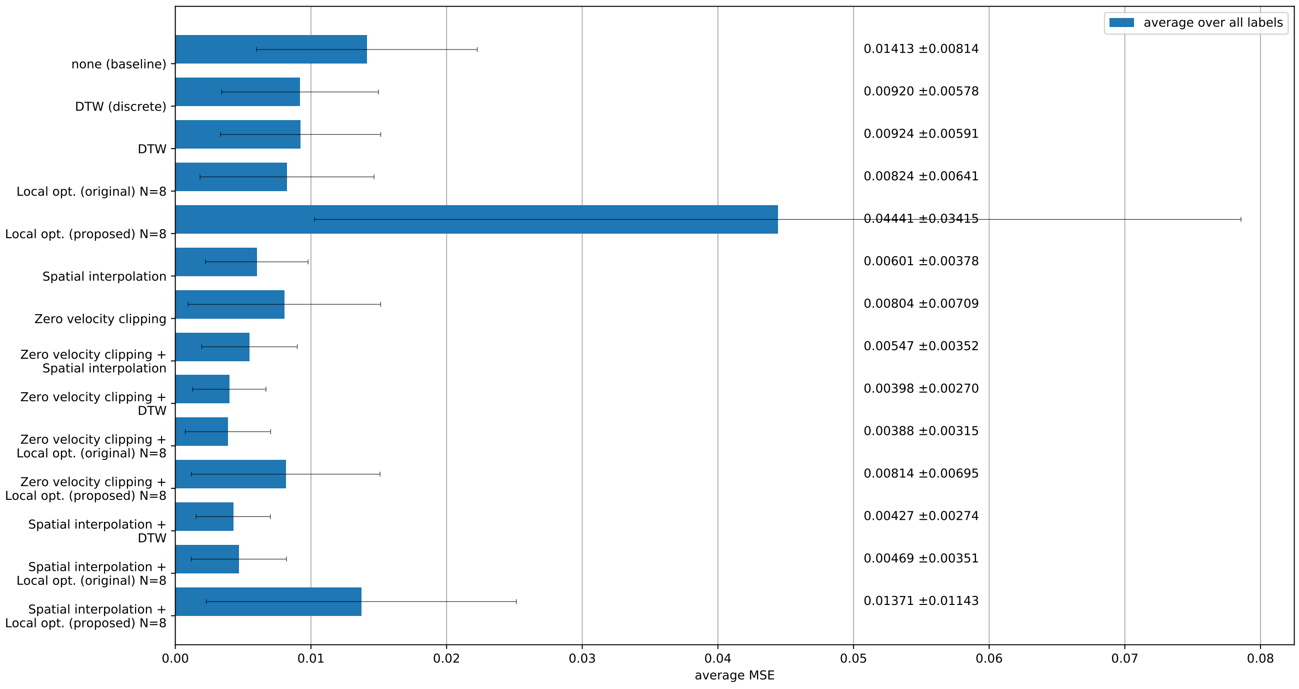


(c) Alignment of the synthetic skewed function “third” back to the original function using the proposed local optimization.

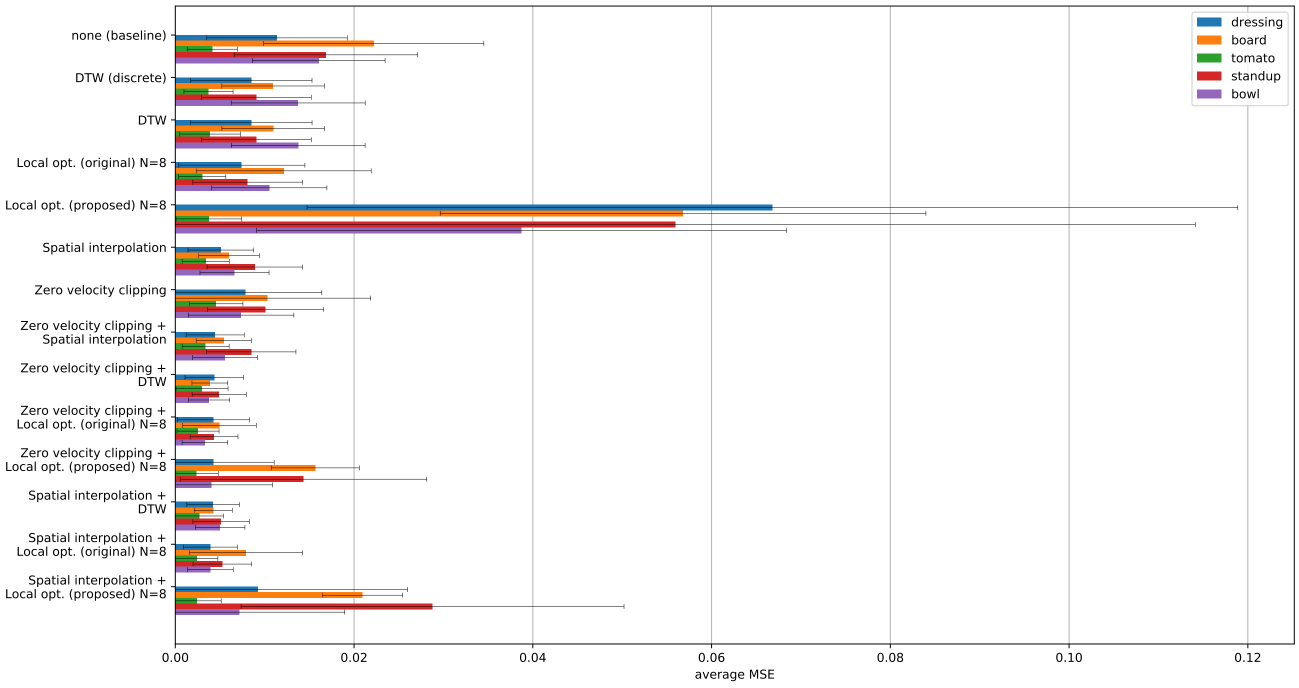


(d) Alignment of two different demonstrations of the “standup” gesture of the *salad* data set using the proposed local optimization.

Figure 4.6: Alignment of synthetic data as well as two different demonstrations of a real recorded gesture using both the original and the proposed local optimization. It can be seen that both methods work better on the synthetic data, which might be explained by the lack of noise in these functions. Furthermore, while the proposed alignment performs better on the synthetic data, it completely fails to align the real trajectory.



(a) Average MSE over all labels when aligning trajectories using the shown methods. The numbers show the average of the MSE and its standard deviation. Lower is better.



(b) Average MSE per label when aligning trajectories using the shown methods. Lower is better.

Figure 4.7: This plot shows the average difference after alignment between 10 trajectories per label and the rest of the trajectories of that label. The averaged MSE along with the standard deviation as error bars is provided. The sub-figure 4.7b shows the results per label, and 4.7a shows the score averaged over all labels. Details on the algorithms in use are given in section 4.1.5.

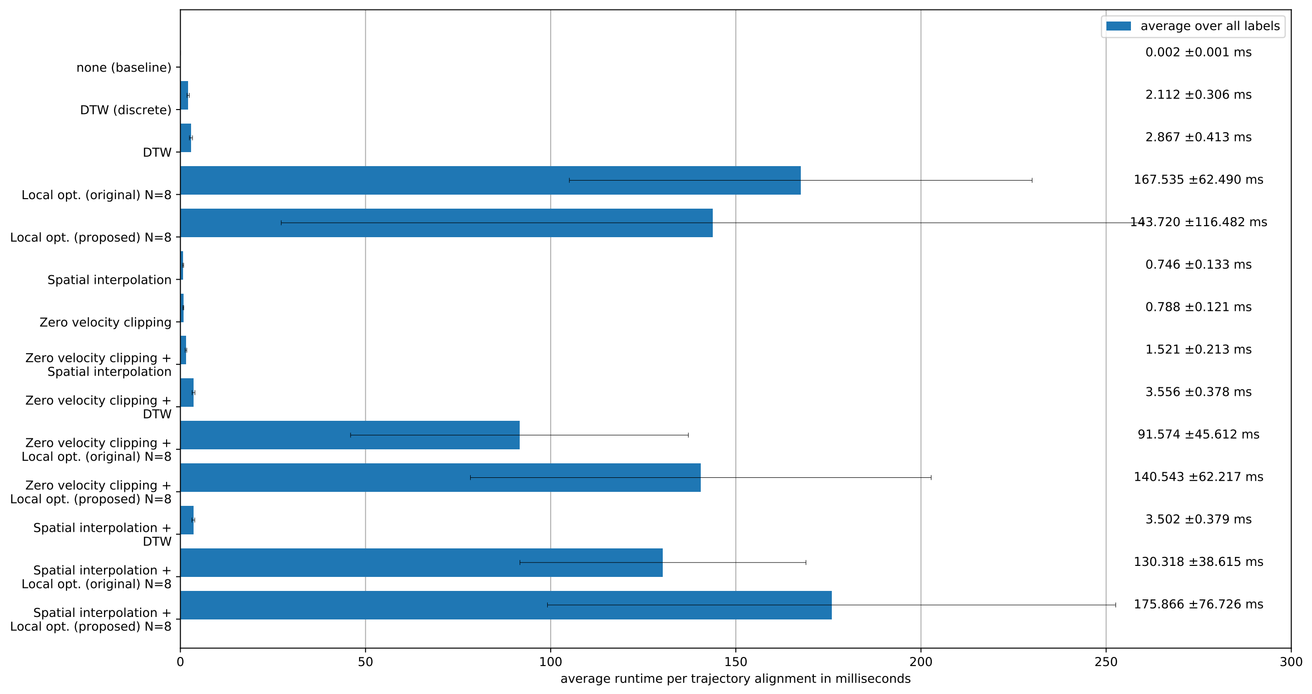
x, y & z coordinates of a wrist. The “none” method is no transformation happening at all and measures the inherent variance of the trajectories as a baseline to compare against. Percentages are given w.r.t. the baseline being 100%.
We can see that DTW does provide an overall improvement of the alignment, reducing the error to 65.25% (discrete) and 65.53% (continuous). Surprisingly, the difference between discrete and continuous DTW is according to this measurement with 0.28 percentage points negligible. It is not as significant as the visual excerpts in figure 4.5 suggested. It is to be noted that while the discrete result is slightly better, the continuous variant is used in the following tests because of the more visually plausible results obtained in the qualitative comparison (cf. section 4.1.3).
The original multidimensional local optimization approach is about 7.09 percentage points better than DTW at aligning the trajectories and achieves a score of 58.44% of the baseline. This is contrary to the bad performance seen previously in figure 4.6b, which led to the development of the proposed local optimization in the first place. That supposedly superior method now proved to be surprisingly bad, causing an outstanding 3.15-fold *increase* in the MSE. It completely fails at aligning the sequences and as such cannot be used. Looking at the error bars, we can also see that there was high variability in the results, suggesting that the method successfully aligned some of the sequences and provided even worse results on others. For example, the results on the label “tomato” are reasonably good, which might be explained by the rather straight movement similar to the warped trajectory see in figure 4.6d.
The spatial interpolation – despite its simplicity – performed the best among the pure methods, reaching a score of 42.62%. It consistently halved the MSE among all labels and has a comparatively low variance in the result. It should be noted that this method is not a pure time sequence alignment method, but significantly alters the shape of the trajectory as well, so in a way this compares apples and oranges. It is however successful in mapping similar trajectories to more similar ones.
Zero velocity clipping is with 54.11% also very successful in aligning sequences, although it not as good as spatial interpolation. It is, however, better than DTW and local optimization, despite being very basic.
Combining multiple methods yields even better results, for example, Zero velocity clipping followed by spatial interpolation gives results 3.12% percentage points better than only applying the latter one. These scores can be improved further by combining zero velocity clipping or spatial interpolation with continuous DTW and the original local optimization. The performance of the proposed local optimization is bad even when combined with other ones. Zero velocity clipping provided better results than spatial interpolation, reaching scores of 27.45% for DTW and 25.60% for local optimization, which is 2.83 and 7.66 percentage points better, respectively. The overall best performance can thus be obtained by first applying zero velocity clipping and subsequently the multidimensional local optimization.

4.1.6 Comparing the Runtime

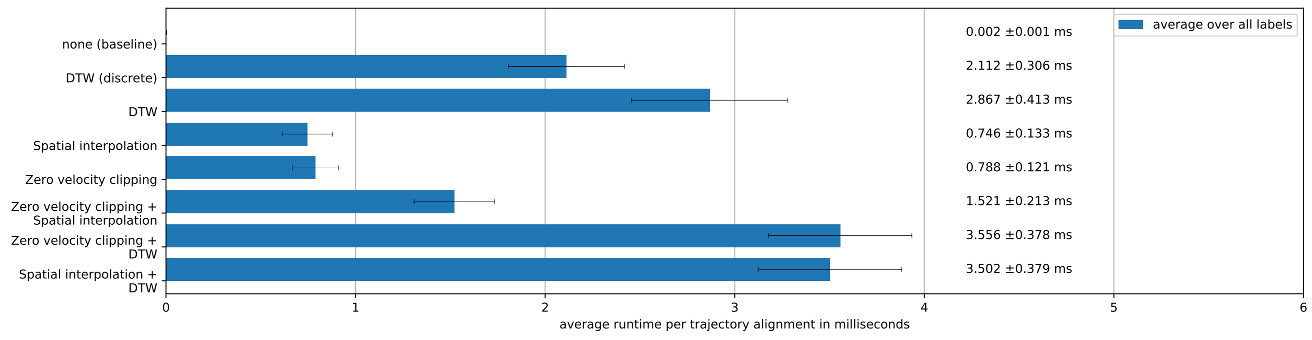
We can see great differences when looking at the runtime required to perform the sequence alignment using the various methods presented in the previous section. Figure 4.8 visualizes the runtime required for the alignment of a single trajectory. The first obvious result is that the local optimization methods run far longer than all other ones. Even the fastest local optimization run (*Zero velocity clipping + Local opt. (original) N=8*) runs about 26 times slower than the slowest other approach (*Zero velocity clipping + DTW*). This was to be expected, since the numerical solution of the optimization problem is computationally very expensive, as it often requires hundreds to thousands of function evaluations to converge, which each involve the computation of the warping function and more importantly the interpolation of the multidimensional trajectory to new time warping. The large runtime might be a factor to abandon this method depending on what response times are acceptable in a real-world application. Further work could be done on tuning the optimization routine by setting a less strict convergence criterion or using a more efficient/suitable optimizer altogether. Because of the significant differences between the methods, figure 4.8b excludes the runtime of local optimization methods.
The entry *none (baseline)* measures the overhead of the function calls and shows that they are insignificant, costing $2\,\mu s$ per trajectory. We can see that all other methods take less than 4 ms per trajectory for alignment, with spatial interpolation being the fastest at about 0.7 ms and zero velocity clipping with DTW being the slowest at about 3.5 ms. As was to be expected, continuous DTW is roughly the same as discrete DTW plus the time for alignment, which is roughly equal to the computational cost of spatial interpolation. Similarly, zero velocity clipping followed by spatial interpolation is as fast as the two methods added up.
In general, the runtime reported here should be used with caution, as DTW for example uses highly optimized *Numba* routines, while the rest is implemented in *Numpy*, *SciPy* and plain *Python 3* code. However, it does give rough estimates and upper bounds for more efficient implementations. The main insight gained by this comparison is the high computational cost of all approaches incorporating local optimization.

4.1.7 Conclusion on Preprocessing

Based on the error evaluation, both DTW and the original local optimization combined with either zero velocity clipping or spatial interpolation seem very promising. However, due to the vastly larger computation times of the local optimiza-



(a) Average runtime required to align a single trajectory in milliseconds. Lower is better.



(b) Average runtime required to align a single trajectory in milliseconds, excluding local optimization for better readability. Lower is better.

Figure 4.8: The average runtime of the different time sequence alignment methods. The alignment methods were tested on the entire *salad* data set. The runtime was averaged over 10 consecutive runs and over all labels. The standard deviation is provided as error bars. Because of the large differences in the results, a second plot is provided excluding the local optimization variants. Details on the algorithms in use are given in section 4.1.5.

tion, it will not be considered in the following evaluation. Instead, we will evaluate the classification with the following methods:

- None, again as a baseline,
- Spatial interpolation,
- Zero velocity clipping with Spatial interpolation and
- Zero velocity clipping with DTW.

4.2 ProMPs

This section will present general results on ProMPs. It will cover possible alternatives and challenges in the implementation, a note on discovered numerical problems and a visualization of ProMPs when applied to the *letters* data set. Finally, ProMPs are evaluated quantitatively on all four data sets to allow a comparison with other methods like SVMs.

4.2.1 On the Implementation

As an alternative to ridge regression, *principal components regression* (PCR) [73, pp. 79f] can be used for the calculation of the weights as well. Evaluation on the self-recorded data set *salad* resulted in accuracy scores that differed by less than 0.1‰ when compared with ridge regression. It was discarded as it is substantially more expensive to compute due to the involved eigendecomposition and did not provide any significant advantage over simple ridge regression.
At first, the implementation was very slow, and profiling was employed to identify bottlenecks in the calculation pipeline. As it turned out, the vast majority of the time was spent pseudo-inverting the $\mathbf{\Phi}$ matrices for the weight space transformation (equation (2.5)). Caching that inverse resulted in a speedup of more than one magnitude. Furthermore, the evaluation of the PDF of the multivariate normal distribution $\mathcal{N}(\boldsymbol{\mu}, \mathbf{\Sigma})$ provided by `scipy.stats.multivariate_normal.pdf` [67] involves the calculation of eigenvalues of the covariance matrix $\mathbf{\Sigma}$, which was rather computationally expensive. Thus, caching the result of `scipy.stats.multivariate_normal` also provided a significant speedup and allowed an easier evaluation of the different algorithms.

4.2.2 Numerical Stability

Both the offline and online algorithm used to calculate the GMM seemed to be running into numerical problems since the resulting covariance matrices making up the ProMPs were often singular at the end. This could be solved by transforming the matrix into an invertible one by applying either the normalization of ridge regression (i.e. adding some small factor to the diagonal) or by the mechanism of PCR (i.e. specifically making very small values in the spectrum slightly positive). Again, the selection of the method did not have a notable impact on the performance of the method. Adding the normalization of ridge regression improved the accuracy to a level where ProMPs could be used at all. A factor of $\lambda = 1 \times 10^{12} \cdot \epsilon \approx 2.22 \times 10^{-3}$ was used in the experiments with offline ProMPs, where $\epsilon \approx 2.22 \times 10^{-16}$ is the machine epsilon of the 64-bit “double precision” IEEE 754 floating point numbers used for all calculations. For the online learning of ProMPs, the results were only usable with the large factor of $\lambda = 1 \times 10^{16} \cdot \epsilon \approx 2.22$. But even that did not prevent the covariance matrix from becoming unstable in the case of the *MSR Action3D* data set.

4.2.3 Visual Example with the *letters* Data Set

This section provides a visualization of ProMPs, starting out with the distribution of the original ten trajectories of the symbol “9”. On the left of figure 4.9, the trajectories are plotted along with the euclidean mean at each point in time. Note that this would generally not be a good reference trajectory for time sequence alignment, as explained further in section 3.6.5. The plot on the right of figure 4.9 displays the same mean as the original trajectories on the left, but now has artificial trajectories sampled from the distribution of the original ones in trajectory space (i.e. not by going through weight space). As we can see, this nicely reproduces the distribution as one would intuitively expect it to. Sampling the entire stacked vector using the covariance matrix is different from simply sampling at each point in time using the variance at than spot, since that approach would not produce such smooth, plausible and *self-consistent* trajectories.
The transformation into weight space follows next and is depicted in figure 4.10. The first row simply shows the same original trajectories as before, but now separates them into the two spatial dimensions and plots their mean and variance over time. Next up is the transformation into weight space by applying equation (2.5). Using the original relation

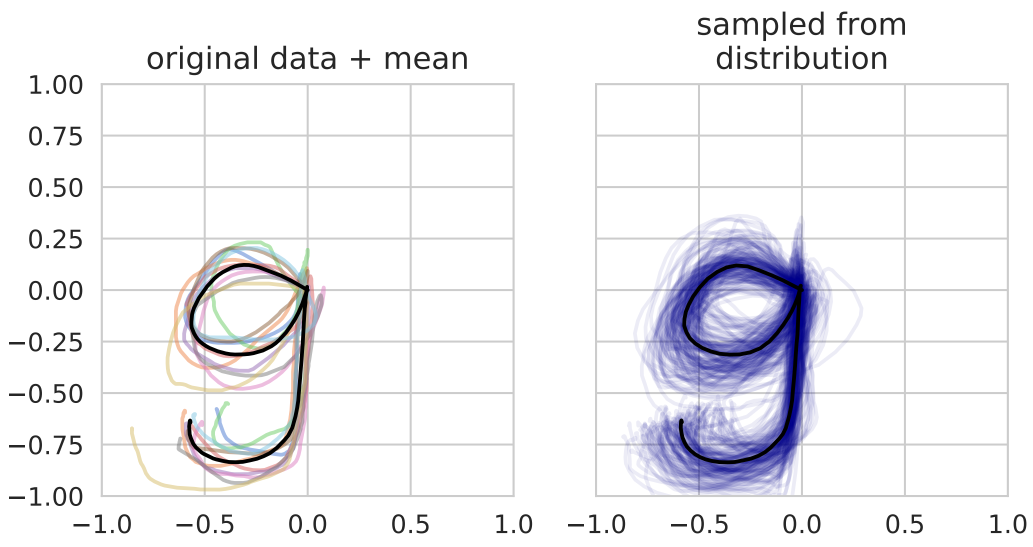


Figure 4.9: Visualization of the ten demonstrations of the the symbol "9" from the *letters* data set. The bottom axis is the x direction, and the axis to the left is the y direction. On the left are original trajectories in faint colors along with the mean painted in black. The right shows the same mean, but now consists of artificial trajectory examples in blue. They are drawn from the probability distribution of the assumed normally distributed original trajectories using the method explained in section 2.5.

$\boldsymbol{\tau} \approx \boldsymbol{\Phi} \cdot \boldsymbol{w}$, we can transform it back into weight space and can see how well the trajectories are preserved. This is depicted in the second row as the individual demonstrations layed over each other. The mean and covariance of the calculated weights form the ProMP, and transforming that back into the trajectory space results in the visualization of the third row. We can see that the general shape is preserved and that the shown 2σ interval very roughly resembles the variability in the input data. We can also notice some smoothing being introduced, which is in line with the decision of not employing any other means of noise reduction motivated in section 3.3.
The distribution formed by the ProMP can again be visualized in 2D, analogous to figure 4.9. This is shown in figure 4.11, where the blue lines are the artificial trajectories sampled in the weight space and transformed back into trajectory space. We can see that while the shape has been slightly deformed, it still captures the essence of the performed action. Note that here again, the use of the covariance matrix yields very self-consistent trajectories.

4.2.4 The Reconstruction Error of the Weight Space Transformation

The effect of different numbers N of RBFs/weights on the reconstruction error ($\boldsymbol{\tau} \rightarrow \boldsymbol{w} \rightarrow \boldsymbol{\tau}'$) is shown in figure 4.12. This experiment shows that ProMPs can accurately represent gestures, as the time series of the trajectories can be reconstructed with a low error if a sufficient number of weights is used. Knowing that our data was scaled to lie within $[-1, 1]$ for this experiment, we can see that, for example, $N = 15$ weights provided a mean absolute reconstruction error of about 0.5%. The reconstruction error stays roughly constant beyond about $N = 80$, which can be explained by numerical limitations of the implementation based on floating-point numbers.

4.2.5 Classification Performance of GMMs

This section will provide results on the classification performance of ProMPs/GMMs, evaluated both with offline and online learning. Results in the form of accuracy and runtime are provided per learner and are split by the preprocessing methods selected previously in section 4.1.7. The classifiers are tested on the *letters*, *salad*, *MSR Action3D* and the reduced data set with the cross-validation schemes found in table 3.2. However, online learning could only be tested on the first two, as the runtime did not allow for a proper evaluation of the *MSR Action3D* data set. The results of the classification are shown in figure 4.13.
As a first observation, we can see that ProMPs can perform decent gesture classification on both the *letters* and *salad* data set, reaching accuracy scores of up to 100%. This is true for both offline and online learning. The classification of the original *MSR Action3D* data set is less successful, but it can be seen that at least up to 23.7% (with no time sequence alignment) of the gestures are classified correctly. This is significantly better than plain guessing on a data set with 20

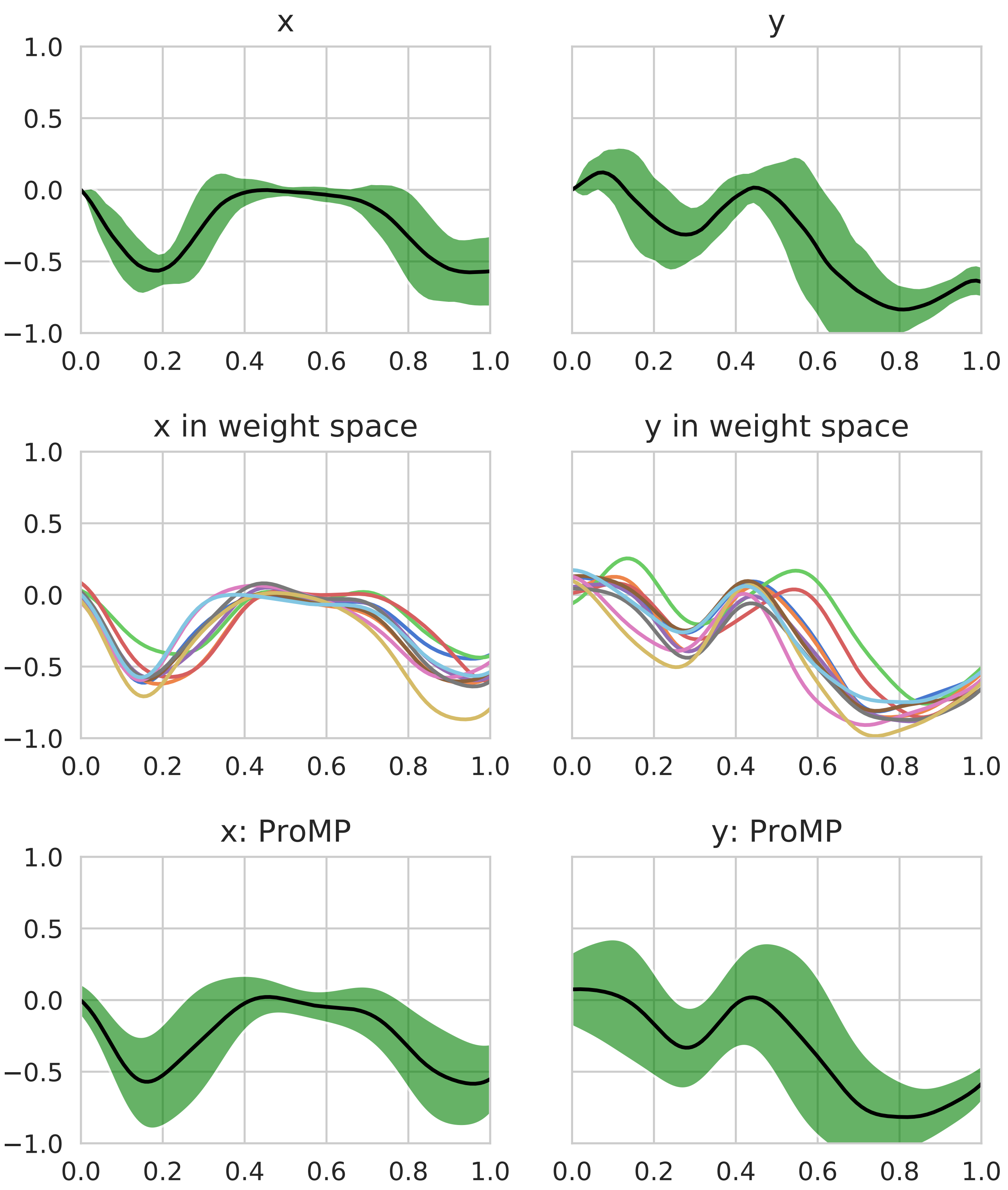


Figure 4.10: Illustration of ProMPs learned on ten demonstrations of drawing the symbol "9". The first row shows the mean and covariance as the double of the standard deviation σ per each dimension over time. Following are the trajectories recovered after being transformed into weight space and finally we can see the resulting ProMPs. These consist of the mean and covariance of the weights and are displayed after being transformed back to the trajectory space. They are shown as the input data in the first row. A comparison with that input shows that the general shape of the trajectories, as well as their variability, is captured successfully.

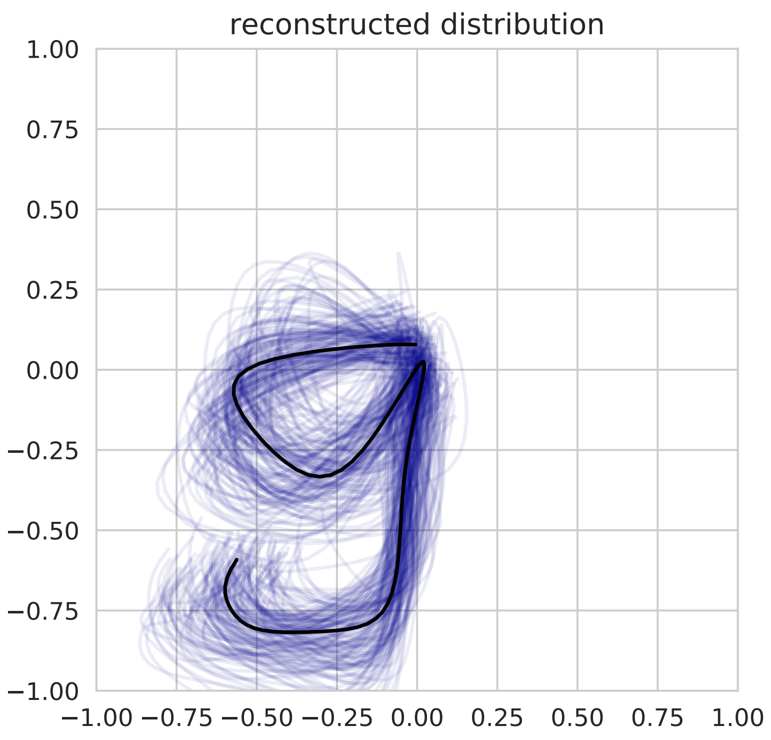


Figure 4.11: This 2D visualization shows artificial weights being sampled from the distribution defined by the ProMPs, which were transformed back into trajectory space in blue. The black line shows the transformed mean. This can be compared nicely with the original distribution of the gestures directly in trajectory space found in figure 4.9.

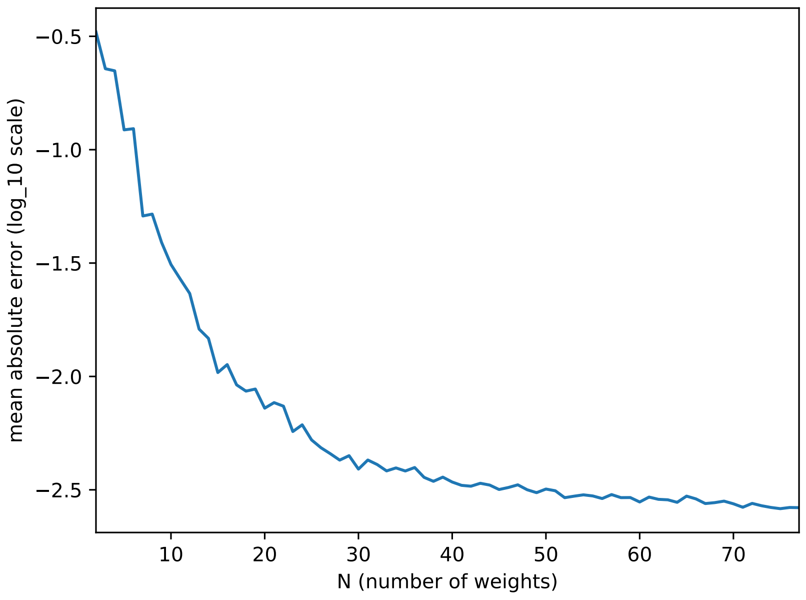


Figure 4.12: Visualization of the effect of the number of weights per dimension N on the reconstruction error when transforming from trajectory space into weight space and back. The trajectory used for this visualization stems from a real recording of the letter "9" (no synthetic data). The error is the mean absolute error (MAE) per sample point on a base 10 logarithmic scale. This plot serves only the visualization of the effect of N, while in practice such high numbers of weights rarely make sense since they defy the goal of smoothing and reducing the dimensionality.

different labels. Also, the GMM achieves even slightly better performance on the *reduced* data set at 26.9%, again with no time sequence alignment being applied.
The following percentage differences indicate absolute percentage points. Spatial interpolation and its combination with zero velocity clipping improved the classification of both the *letters* (+15.0%, +15.0%) and the *salad* data set (+1.4%, +2.0%) for offline learning. For online learning, the baseline already saturated the data set giving 100% accuracy, and the improvement on the classification of the *salad* data set was minuscule as well (+0.9%, +0.5%). All three methods also did not increase, but considerably decreased the performance of the classification of the *MSR Action3D* data set in both the original and reduced variant. This is especially true for the zero velocity clipping combined with DTW: Apart from the offline learning of the *letters* data set, it never resulted in a significant improvement in the accuracy. It even decreased the score for the *MSR Action3D* data set down to an accuracy of only 4.8% and 4.1%, for the original and reduced data set respectively. This is similar to guessing a constant label, as it is roughly the same as the portion $1/20$ of a single label.

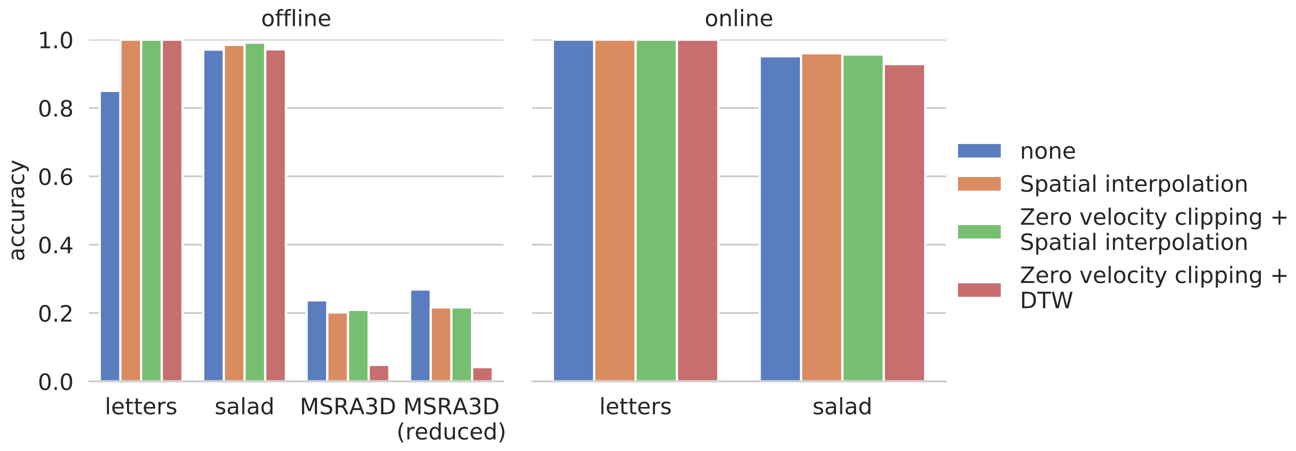


Figure 4.13: Classification results as accuracy scores for both offline and online learning of GMMs/ProMPs. The four time sequence alignment methods selected previously have been applied first. Results are given on all four data sets, although online evaluation on the original and reduced *MSR Action3D* data set was infeasible due to runtime limitations.

4.3 Classification using SVMs

This section reports on SVMs being tested on the four data sets with the aforementioned cross-validation procedures (cf. section 3.9). Both offline and online learning was performed, both using SGD SVMs as presented in section 2.6 and section 2.6.2. Online learning on the *MSR Action3D* data set in original and reduced form was infeasible due to performance issues, as was the case in section 4.2.5. A question approached by this evaluation was whether the transformation into weight space as in equation (2.5) increases the classification performance of the SVMs. Thus, both SVMs in the trajectory space as well as in the weight space are considered in the following.

4.3.1 Classification using SVMs in Trajectory Space

Figure 4.14 shows the accuracy achieved by SVMs being trained on the raw trajectories, i.e. with features lying in for example $1 \times 3 \times 80 = 240$ dimensions for the *salad* data set. The original *MSR Action3D* even consists of features in $20 \times 3 \times 80 = 4800$ dimensions.
Generally, we can see that the SVMs can correctly classify the gestures of the *letters* on both offline and online training scenarios without any errors, except when DTW is used. It significantly degrades the performance on this data set, and lowers the accuracy down to 80.0% for offline and 95.0% for online learning, respectively. They are also able to learn the structure of the *salad* data set offline, reaching a score of 96.5% with spatial interpolation. That improves the performance compared to no time sequence alignment being performed by 3.0 percentage points, and first applying zero velocity clipping results in an improvement of only 2.4 percentage points above the baseline. DTW again decreases the performance, here by notable 24.9 percentage points. The SVM seems to have problems learning the *salad* data set online, at it reaches accuracy scores of only 26.5% (baseline) and less. It is interesting that DTW slightly decreases the accuracy only very slightly while both methods involving spatial interpolation have a more negative effect.

Learning gestures of the *MSR Action3D* data set is more challenging. The algorithm performs very poorly in that test, reaching an accuracy of only 12.1% (using spatial interpolation). The reduction to only 3 joints, however, yields an improvement to 17.3% (using not time sequence alignment). In any case, DTW decreases the performance the most, resulting in scores of 5.0% (offline) and 5.9% (online), which basically amounts to guessing, as $1/20 = 5\%$.

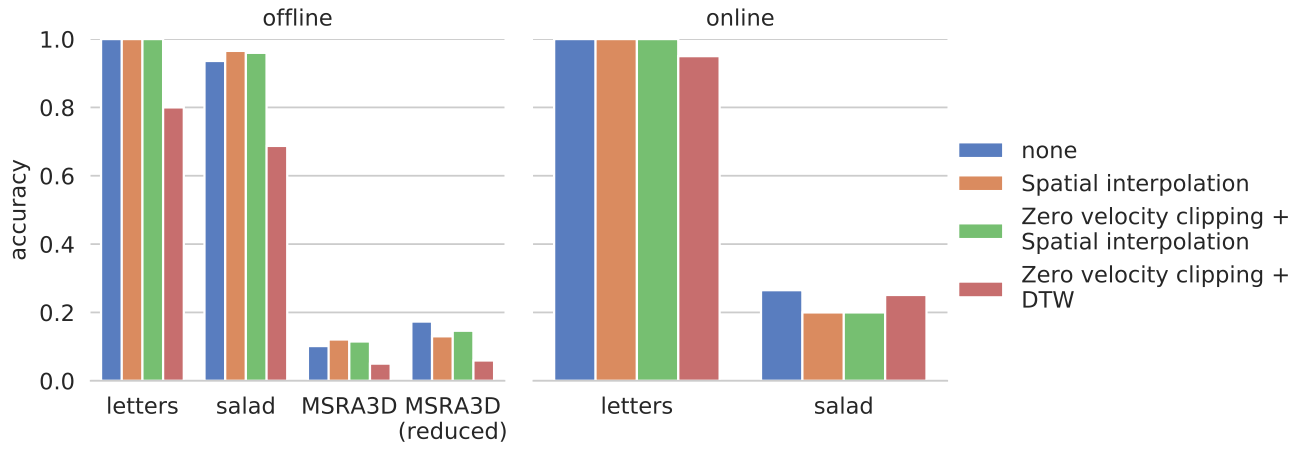


Figure 4.14: Classification results on all four data sets using SVMs in the trajectory space, i.e. without a prior weight space transformation. The evaluation is given for each of the previously selected time sequence alignment methods. The trajectory was sampled at 80 points in time. As before, online evaluation was not performed on the *MSR Action3D* data set.

4.3.2 Classification using SVMs in Weight Space

Figure 4.15 shows the accuracy achieved by SVMs being trained with SGD in the weight space, i.e. with each dimension being reduced to $N = 8$ weights. Time sequence alignment was performed before that, and no other means of preprocessing were applied after the transformation.

Usually, SVMs require input data with zero mean and unit variance to perform reliably [36]. However, using `sklearn.preprocessing.StandardScaler` or `tslearn.preprocessing.TimeSeriesScalerMeanVariance` only worsened the scores significantly instead of improving it, and as such was not applied.

The general observation is that the weight space transformation had different effects on offline and online training. For offline training, it did not change the score on the *letters* data set at all. The differences on the *salad* data set were minor (except for the badly performing DTW), resulting in differences of +1.1, +0.7, −0.5 and +9.7 percentage points, for the time sequence alignment methods baseline, spatial interpolation, the same with zero velocity clipping and DTW, respectively. For online training, however, the application of the weight space transformation hurt the accuracy on the *letters* data set, decreasing the score significant. It, however, did not have a significant effect on the performance while online learning on the *salad* task.

The classification of the *MSR Action3D* data set (offline) is similar to the classification using SVMs on the raw trajectories, in that the performance is generally worse than on the other data sets, but slightly better on the reduced variant than on the original full data set. For both the original and the reduced data set, the same pattern occurs: Applying no time sequence alignment works the best, yielding accuracies of 19.6% and 25.4% for offline and online learning, respectively. All other methods show decreasing accuracy scores in the order they are mentioned above, reaching 12.1% and 8.2% for the DTW methods. The combination of SVMs in the weight space without preprocessing on the reduced *MSR Action3D* data set were able to correctly classify a fourth of the gestures for the first time.

4.4 Comparison of the Classification Performance

So far, GMMs and SVMs in both trajectory and weight space have been tested on multiple data sets with various preprocessing methods and with both offline and online learning. Figure 4.16 now presents these results condensed down to the accuracy scores of each of the three learners obtained on the best preprocessing method each. It shows that ProMPs outperform both variants of SVMs in all of the six scenarios.

All three methods achieved 100% accuracy on the *letters* data set with both offline and online learning, demonstrating that both approaches are practically feasible. That is already different when looking at the second data set: While all

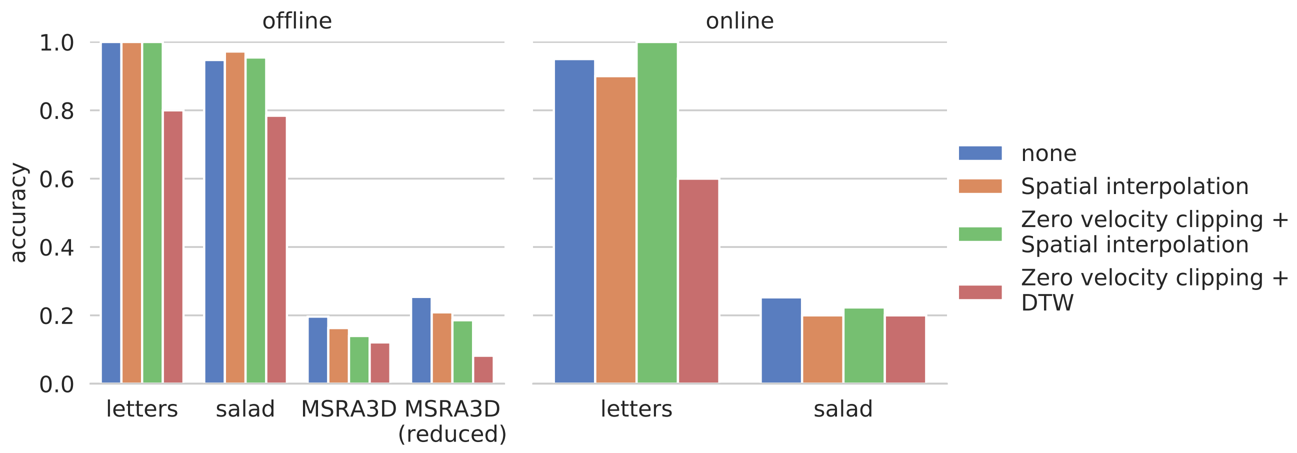


Figure 4.15: Classification results on all four data sets using SVMs in the weights space. For this purpose, each dimension sampled at 80 points in time was first reduced to $N = 8$ weights. The evaluation is given for each of the previously selected time sequence alignment methods. As before, online evaluation was not performed on the *MSR Action3D* data set.

methods where able to classify gestures of the *salad* data set at above 95% accuracy using the offline approach, only GMMs were able to do the same when performing online learning. More specifically, GMMs reached an accuracy of 99.1% for offline and 96.2% of online learning, beating both SVM types with an accuracy of 96.5% and 97.2% (online) as well as 26.5% and 25.3% (offline). SVMs were unusable in the online scenario on the *salad* data set. All methods performed less successful on the higher-dimensional *MSR Action3D* data set, but GMMs are best again at 26.9%, although not as dramatically as with the online learning of the previous data set. All three methods performed better when applied to the variation with the reduced number of joints. The weight space transformation resulted in an improvement in the performance of the SVM in both cases, by 5.2 percentage points on the original and 5.8 percentage points on the reduced data set respectively.
These results should generally be interpreted with caution, as the evaluation could be further improvement by first performing a hyperparameter search. ProMPs or GMMs do not need a lot of these variables to be set correctly, but the used implementation of SVMs allows for a lot of parameters to be tuned. Finding the right set of parameters would likely allow the SVM to perform a lot better at both online learning in general and more specifically on the *MSR Action3D* data set, as better results using SVMs have already been published (cf. section 2.8.1). However, these might not have used SGD and thus might not allow for online learning, which is very relevant for our use case. On the other hand, complete re-training of the classifier might be acceptable in some scenarios, e.g. when the learning is cheap anyways, when it could be deferred to some server or when some delay in the updated behavior was acceptable.

4.5 Effects of Dimensionality Reduction

This section contains two parts: Firstly, for a qualitative evaluation of the embeddings, entire trajectories of the letters and salad data set are embedded into two dimensions, as that is easily interpretable by humans. Secondly, classification using dimensionality reduction is attempted.
The implementation used for PCR is provided by `sklearn` [36] and was configured to the defaults. For UMAP, the reference implementation `umap-learn` by McInnes et al. [39] was used and tuned to preserve as much of the global structure as possible. This sacrifices on the local structure, which was deemed rather irrelevant for this classification task. Thus, it was configured to `min_dist=1e-3` and `n_neighbors=250`.

4.5.1 Visual Evaluation of Embedding Trajectories

First, the two data sets *letters* and *salad* were reduced to two components for the entire trajectory as a whole. PCA is an unsupervised procedure, while UMAP was used as a supervised transformation, i.e. it was fed both the raw trajectories as well as the labels. Visual results can be seen in figure 4.17 for *letters* and in figure 4.18 for *salad*, respectively.
Generally, both PCA and UMAP – with and without the weight space transformation – are able to preserve the global structure. This means, that they project points of different classes into different parts of the latent space. This should allow classification to be done in the latent space while yielding results according to the original labels. the result obtained

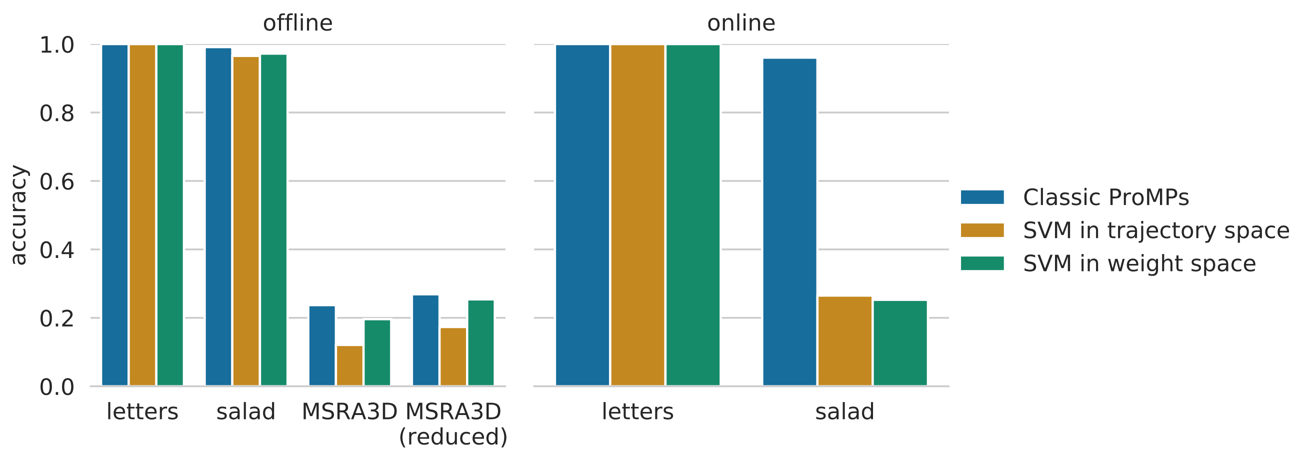
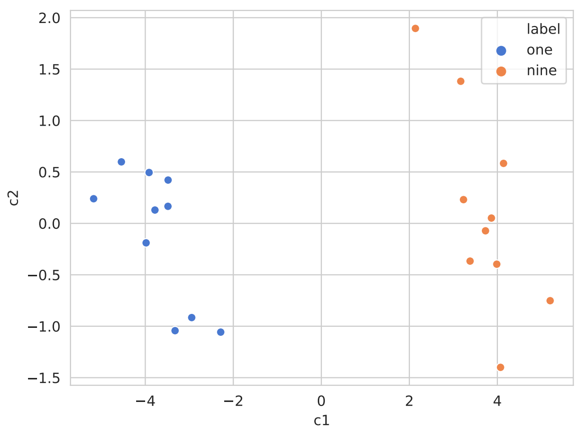


Figure 4.16: This plot shows the best accuracy results obtained by each classifier on each of the four data sets. The classifiers are GMMs, SVMs trained in the trajectory space and SVMs trained in the weight space. Results are given for both offline and online learning, although no online evaluation on the *MSR Action3D* data set was possible due to runtime limitations. It shows that the GMMs outperform this implementation of SVMs in all scenarios in which the data set was not already saturated by both.

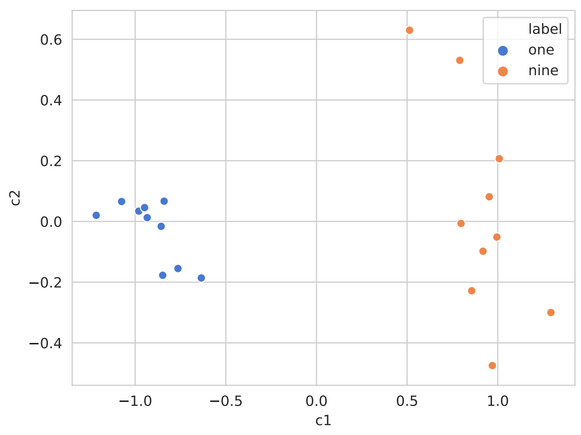
from UMAP is better suited for (GMM) classification, as the resulting intra-cluster distance is minimized, while this is not the case for PCA. Visually, the weight space transformation does help very little in achieving this denser packing.
The *salad* data set is more challenging, as PCA struggles to preserve the global structure well. The resulting clusters show a significant amount of overlap both with and without the weight space transformation. In this configuration, the additional conceptual and computational cost of UMAP pays out, as the clusters present in the ambient space result in very distinct locations of the projected classes in the ambient space. This is true for both the direct application of UMAP and the application after a weight space transformation.
Section 4.4 has shown that gestures from the *letters* and *salad* data sets could be classified with high accuracy. However, there were difficulties with the classification in the *MSR Action3D* data set. Dimensionality reduction techniques might be able to improve on this. Thus, a reduction of individual skeletons was attempted, resulting in trajectories being curves in the 2- or 5-dimensional latent space, rather than the 60-dimensional ambient space. No weight space transformation was performed. The results are shown in figure 4.19. The results are difficult to interpret, but we can see that some structure has been preserved. However, the gestures are not simple grouped lines, but comparatively chaotic for some places. However, it should be noted that no clear clusters were to be expected, as this does not reduce the entirety of the Trajectory, but only single poses instead. Further evaluation is required.
Next, an embedding of entire trajectories of the *MSR Action3D* data set was attempted. For the computational feasibility, the trajectories were first converted into the weight space. The input data thus consists of $D \times N = 60 \times 8 = 480$ dimensions, and an embedding into two dimensions was performed. Figure 4.20 shows the results of applying PCA and UMAP like that. We can see that PCA fails to create an embedding useful for classification tasks, as the projections of multiple labels end up overlapping in the latent space. UMAP, however, does provide a visually convincing separation, but it can be seen that gestures of a single label sometimes end up in multiple disjoint clusters. This might be hard to classify, as one-vs-the-rest linear SVMs and GMMs with a single mixture component per label cannot appropriately capture these complex shapes. Note, that this transformation was performed supervised, contrary to the visualization in figure 4.19.

4.5.2 Classification of Dimensionality Reduced Gestures

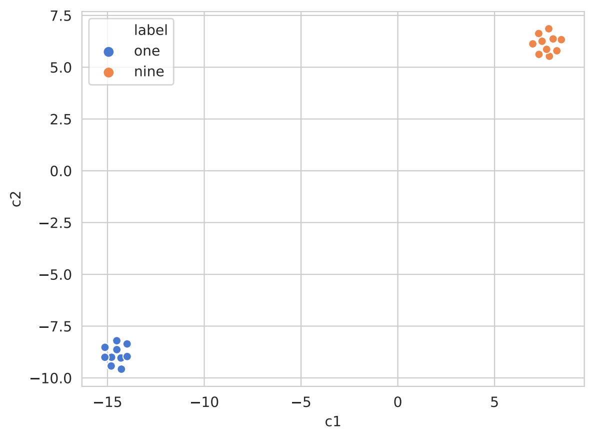
For classification, the entire trajectories of the *letters*, *salad* and *MSR Action3D* data sets were embedded into a two-dimensional latent space as a whole. This was again done using both PCA and UMAP, with both the weight space transformation performed beforehand and without it. The resulting accuracies for offline classification with both GMMs and SVMs as used previously can be seen in figure 4.21.
The *letters* data set is again easiest to classify and nearly all classifiers reach complete accuracy on it. A small exception are support vector machines combined with UMAP, but we have no supplementary data to explain this (outlying) observation. GMMs perform better at classifying the *salad* gestures, reaching 95.1%, as opposed to a maximum of 90.0% for SVMs. The same is true for the *MSR Action3D* data, where GMMs reach up to 22.8% and SVMs only reach 12.3% accuracy. Neither PCA nor UMAP are consistently better than the other. While UMAP is always better than PCA on the *salad* data



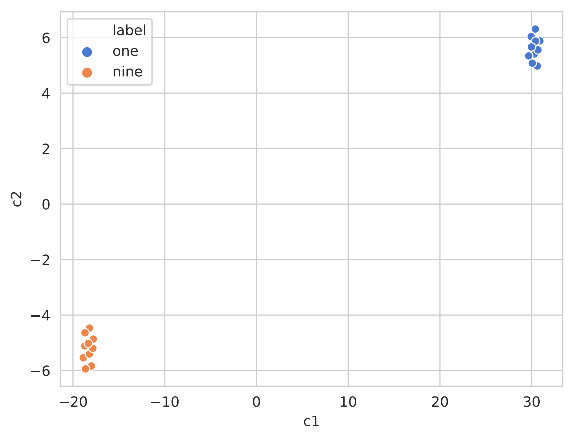
(a) Reduction using only PCA.



(b) Reduction using a weight space transformation and PCA.

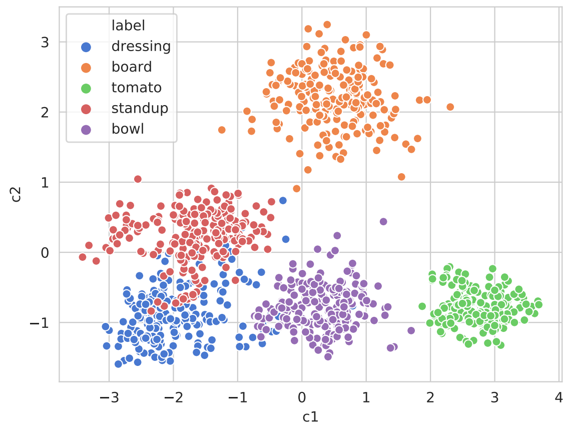


(c) Reduction using only UMAP.

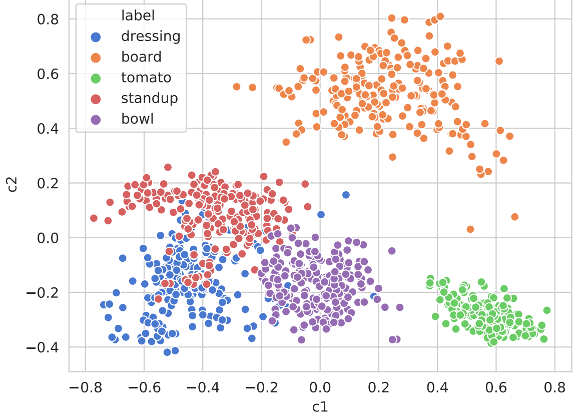


(d) Reduction using a weight space transformation and UMAP.

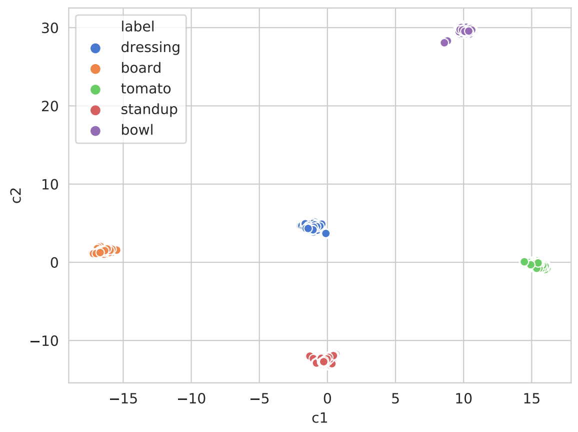
Figure 4.17: Dimensionality reduction applied to all entire trajectories of the *letters* data set, down to the two components c_1 and c_2 per trajectory. The reduction was performed with PCA and UMAP, and both with and without applying a weight space transformation first. Both algorithms are able to successfully project data of different labels into separate regions of the latent space. However, UMAP groups the elements in the emerging clusters closer together. The weight space transformation does not yield a significant visual benefit.



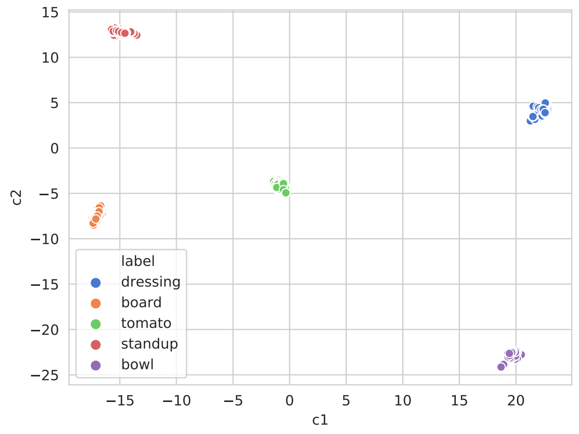
(a) Reduction using only PCA.



(b) Reduction using a weight space transformation and PCA.

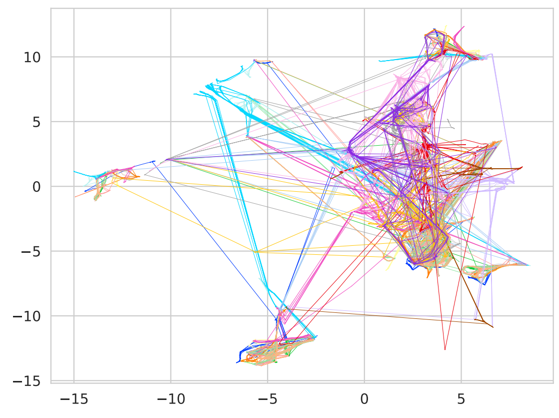


(c) Reduction using only UMAP.

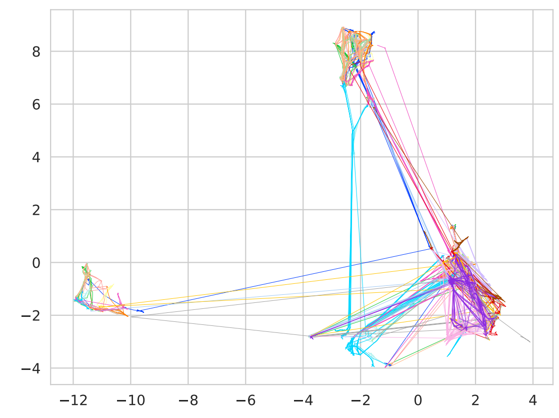


(d) Reduction using a weight space transformation and UMAP.

Figure 4.18: The *salad* data set, reduced down to two components c_1 and c_2 per trajectory using PCA and UMAP. Both were tested with and without applying a weight space transformation first. UMAP outperforms PCA in that it better preserves the global structure and clusters are well separated in the latent space. PCA fails to provide a clean separation. Here again, the weight space transformation does not have a significant visual impact.

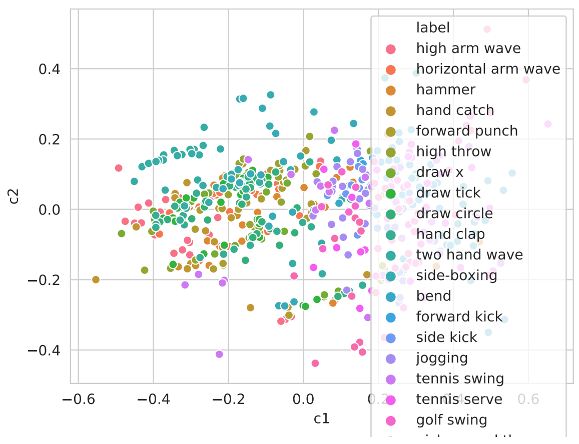


(a) Reduction using UMAP into a two-dimensional latent space.

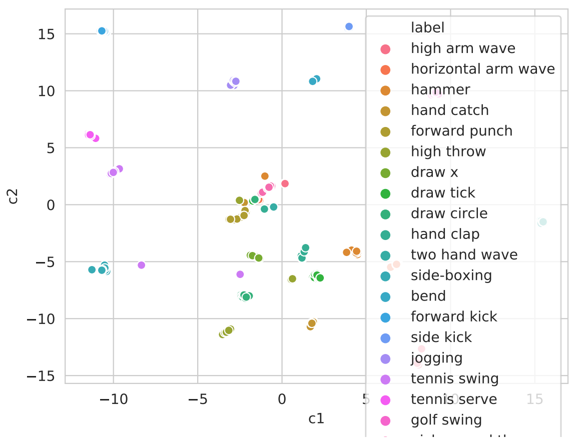


(b) Reduction using UMAP into a five-dimensional latent space. The first two components are shown.

Figure 4.19: Reduction of all individual skeletons of the *MSR Action3D* data set using UMAP. The trajectories are visualized as curves in the latent space and are colored by the membership to a label. Some structure is preserved, but it is difficult to draw concrete conclusions from this visualization.



(a) Reduction using a weight space transformation and PCA.



(b) Reduction using a weight space transformation and UMAP.

Figure 4.20: Reduction of all entire trajectories of the *MSR Action3D* data set using PCA and UMAP. The entire trajectories were embedded into two dimensions after being transformed into the weight space with $N = 8$ weights per dimension. PCA does not provide a projection suitable for classification, whereas UMAP separates the clusters to some degree.

set, it is exactly the other way around in the case of *MSR Action3D*, where UMAP significantly reduces the accuracy when compared with the more simple PCA. Another finding is the confirmation of the impression gained in section 4.5.1: That the weight space transformation has only a moderate impact on the quality of the dimensionality reduction output. This is now quantitatively confirmed, as the difference in the accuracy score in percentage points when moving to the weight space is −1.7/+0.5 (*salad*) and +0.2/+0.3 (*MSR Action3D*) for PCA/UMAP with GMMs and −8.41/+0.4 and −0.2/−0.7 for the SVMs, respectively. Note, that the effect of applying PCA to the *salad* data set significantly reduced the accuracy of performing classification with SVMs. However, it can be used as a coarse dimensionality reduction method to speed up subsequent computations.

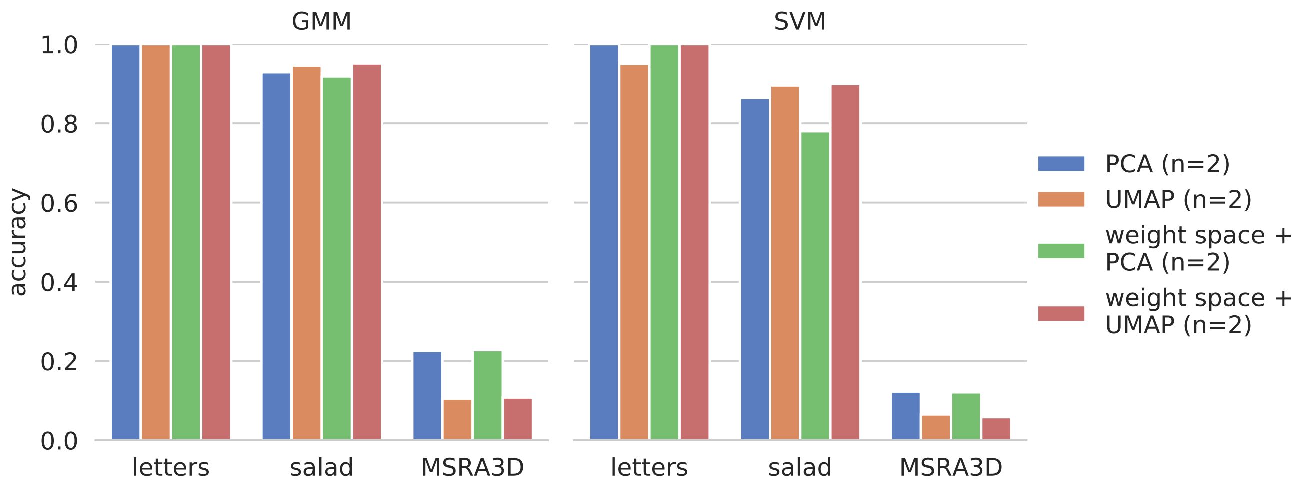


Figure 4.21: This figure compares how GMMs and SVMs can classify the dimensionality reduced data sets *letters* and *salad*. The reduction was performed using PCA and UMAP, both with and without prior weight space transformation to $N = 8$ weights per dimension. A reduction down from the entire trajectory over all time steps down to only two latent space variables was attempted. As this comparison shows, most of the relevant information gets preserved by this transformation.

A different approach discussed in section 3.8 than the transformation of the entire gesture is to embed the skeleton space into a lower dimension. This is especially promising for the gestures of *MSR Action3D*, since these contain a high amount of redundancy (e.g. the center of the hip should always be between the right and left part) and reducing the skeletons individually still allows for general time sequence processing. That includes time sequence alignment discussed above as well as future approaches, to for example detect actions too and not only classify them. Thus, the data set was reduced down to 2 and 5 dimensions and classification with both ProMPs/GMMs and SVMs in trajectory and weight space was attempted. Two were chosen since it is good for visualization purposes, and five were chosen as *Dermy et al.* have already reported good results when reducing full-body skeletons down to five dimensions, although using autoencoders [63]. The resulting accuracy scores can be found in figure 4.16. It can be seen that the GMMs barely correctly classify any gestures, achieving accuracies of 0.0% and 0.3% on the 2 and 5 dimensional skeletons. In this case, the weight space transformation does improve the classification accuracy, causing an increase of 4.2 and 6.3 percentage points, for the 2- and 5-dimensional skeletons, respectively. Furthermore, the higher dimensional latent space is better suited for classification, resulting in a score of 9.2 percentage points better than the latent space of only two dimensions when applying the weight space transformation. The achieved 23.3 percent are only very slightly better than the best result obtained from reducing the trajectory as a whole, wich a difference of only 0.5 percentage points. This is, however, not as good as the performance of a GMM on the weight space (ProMP), which reached 26.9% as shown in section 4.4.

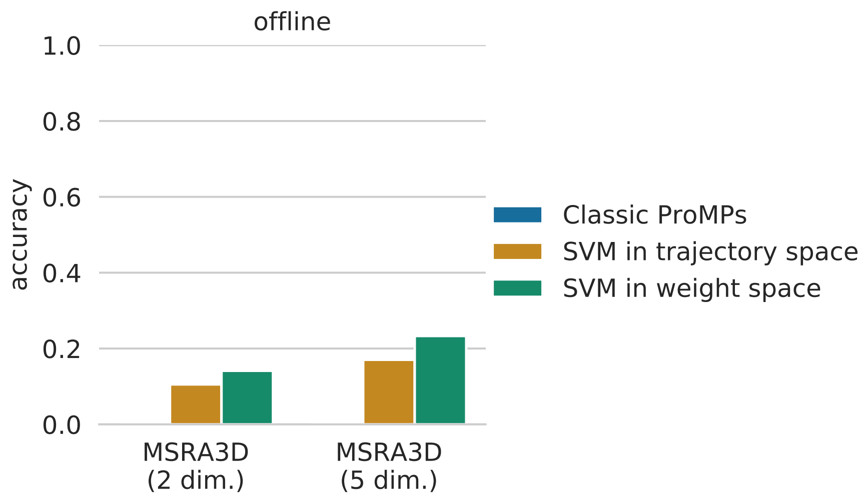


Figure 4.22: Effect of using UMAP to embed each skeleton in a 2- or 5-dimensional latent space. That reduced gesture representation is subsequently used to perform classification using GMMs and SVMs (in weight and trajectory space). It shows that GMMs completely fail to learn anything useful for classification, while SVMs can classify some gestures, and do so better in the higher dimensional embedding.

5 Conclusion and Future Work

The goal of this thesis was to use ProMPs for the classification of gestures in the context of assisting in elderly care. ProMPs were selected due to their desirable theoretical properties. However, they have not yet been systematically benchmarked for classification tasks and this gap shall be filled. As part of that, different preprocessing methods are evaluated, namely zero velocity clipping, spatial interpolation, dynamic time warping and local optimization, in various combinations. Going further, dimensionality reduction using PCR and UMAP is explored. For the actual classification, the GMMs used typically together with ProMPs are compared to SVMs learned in the weight space of ProMPs and on the raw trajectories as well. The SVMs are trained using sequential gradient descent, as that allows for both offline and online learning as is possible with GMMs and is required in this domain to allow for constant improvement of the assistive robot being developed. The evaluation is done on four data sets: A small data set containing the drawing of two numbers to allow for first tests; a newly recorded hand trajectory data set with over 1000 demonstrations in a robotic kitchen assistance scenario to evaluate the applicability to robotics; the widely adopted MSR Action3D data set to allow for comparison of the classification performance; and a manually dimension-reduced variant thereof to ease learning of the gestures.
As a first step, different preprocessing methods are evaluated by measuring the similarity after aligning trajectories of the same labels in both synthetic and real-world data. The results showed that both local optimization and DTW perform well, and the similarity score could even be improved by combining it with either zero velocity clipping or spatial interpolation. Additionally to the measurement of the alignment itself, the runtime was also captured, revealing that the implementation of local optimization took around 26 times longer to compute than all other approaches, which might render it unusable in real-time systems.
Three promising preprocessing methods (spatial interpolation, zero velocity clipping together with spatial interpolation and zero velocity clipping with DTW), as well as no preprocessing being applied, have subsequently been evaluated with GMMs and the two SVM variants in order to evaluate the classification performance. The results show that applying DTW quite consistently decreases the accuracy of the learners, while the other two methods have shown both positive and negative effects on the classification. Overall, the feasibility of classification of gestures using ProMPs could be confirmed, with accuracy scores on the new data set reaching 99.1% with offline learning. SVMs were not able to learn these kitchen data set gestures nearly as well as the GMMs did, as they only reached a score of about 26% accuracy instead of the 96.2% of the GMM. All three classifiers generally worked better in offline than in online scenarios, and higher dimensionality of the input data resulted in decreased performance. The performance on the MSR Action3D data set was poor, reaching a mere 26.9% using GMMs and even less using SVMs. The weight space transformation, however, did improve the general classification performance of SVMs. Dimensionality reduction with PCA and UMAP was attempted and successfully applied to the kitchen data set, but could not improve the performance of the classifiers when applied to the MSR Action3D data set.

There are many possible ways to extend this work. First of all, the preprocessing steps, as well as the dimensionality reduction methods and classifiers, depend on many hyper-parameters to be tuned in order to work properly. This includes, for example, clipping thresholds, DTW slope constraints, the number of components for the weight space transformation or the loss function used in the SVM. Systematically exploring that space of possibilities might yield different results then obtained from the hand-tuned methods presented above. A first step would, therefore, be to make the implementation more closely follow the interfaces of `sklearn`, to allow for better evaluation using existing systematic exploration methods. This will probably result in the SVMs performing significantly better, as accuracies beyond 90% have already been reported on the MSR Action3D data set using some type of SVM. Additionally, a measure like informedness might be used for better insight into the quality of the obtained learners. One criterion for selecting the lerner was the ability to classify unknown gestures as such, which has been implemented for the GMM but no evaluation on that was performed yet.
The use of different time sequence alignment methods could be investigated as well, for example by optimizing the runtime of the local optimization approach to allow it to be used in actual systems. Additionally, other time sequence alignment methods like Canonical Time Warping by *Zhou et al.* [74] could be used as well. Furthermore, the expressiveness of the time sequence alignment evaluation could be further improved by replacing the MSE by a more appropriate measure of tier performance. The average MSE used in this thesis only measures the similarity between the aligned trajectories within each label individually, or in other words how similar the processed trajectories are to each other. However, the actual performance of such a feature transformation is not determined by the similarity within a class alone, but also

on the structure-preserving nature as well. It should additionally be measured whether previously dissimilar trajectories of different labels stay as such or even move further apart in the newly created feature space. As part of that, some transformation, which is invariant to left- and right-handedness and rotation, should be considered.
Dimensionality reduction is a promising approach, but the anticipated improvements did not take effect on the data sets used in this evaluation. One future direction of research would be the incorporation of autoencoders, for example, using the *AE-ProMPs* by Dermy et al. [63]. A slightly different approach would be to use methods specifically targeted at feature extraction. This might yield better results, as the movement of the hip is often less useful than the movement of a hand when attempting classification. It would require experimental evaluation.
In the broader picture, classification using GMMs is only one approach at gesture classification and others might be worth to consider too. Furthermore, classification alone does not yet allow a robot to actually understand human gestures, let alone react in an assistive way. First of all, there needs to be some technique for noticing an ongoing gesture, which in itself is a non-trivial problem. The result of that is – strictly speaking – only an action, and the actual intent conveyed through it must be determined in order to assist a human. That can probably only be achieved by integrating other sensing methods as well. This is a challenging task, at times causing problems even between humans, as it sometimes requires a significant amount of context specific to some domain and subject. Summarizing, there are many interesting and challenging topics, which need further development until practical use can be made of them. They are currently active areas of research.

Bibliography

[1] Franka Emika GmbH, “Kobo hilft im alltag.” Accessed 9th of May 2019.

[2] Sergio Escalera, Computer Vision Center and University of Barcelona, Catalonia (Spain), “Multimodal gesture recognition: Montalbano V2,” 2014. Accessed 14th May 2019.

[3] L. L. Presti and M. L. Cascia, “3D skeleton-based human action classification: A survey,” *Pattern Recognition*, vol. 53, pp. 130–147, 2016.

[4] Wikimedia Commons user “Programminglinguist”, “Dynamic Time Warping,” July 2015. Accessed 27th of July 2019.

[5] Wikimedia Commons user “Larhmam”, “SVM margin,” October 2018. Accessed 22nd of Sept. 2019.

[6] F. Petitjean, A. Ketterlin, and P. Gançarski, “A global averaging method for dynamic time warping, with applications to clustering,” *Pattern Recognition*, vol. 44, no. 3, pp. 678–693, 2011.

[7] M. Cuturi and M. Blondel, “Soft-DTW: a differentiable loss function for time-series,” in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 894–903, JMLR. org, 2017.

[8] R. Tavenard *et al.*, “tslearn, a machine learning toolkit dedicated to time-series data.” unpublished as of this writing, N.D.

[9] D. Koert, S. Trick, M. Ewerton, M. Lutter, and J. Peters, “Online learning of an open-ended skill library for collaborative tasks,” in *Proceedings of the International Conference on Humanoid Robots (HUMANOIDS)*, 2018.

[10] European Commission, Department of Economic and Financial Affairs, “Ageing report: Europe needs to prepare for growing older,” May 2012. Accessed 9th of May 2019.

[11] Bundesministerium für Bildung und Forschung, “Kobo34.” Accessed 9th of May 2019.

[12] S. Trick, D. Koert, J. Peters, and C. A. Rothkopf, “Multimodal uncertainty reduction for intention recognition in human-robot interaction,” *CoRR*, vol. abs/1907.02426, 2019.

[13] A. Jaimes and N. Sebe, “Multimodal human-computer interaction: A survey,” *Computer Vision and Image Understanding*, vol. 108, no. 1, pp. 116–134, 2007. Special Issue on Vision for Human-Computer Interaction.

[14] I. Laptev, M. Marszalek, C. Schmid, and B. Rozenfeld, “Learning realistic human actions from movies,” in *CVPR 2008 - IEEE Conference on Computer Vision & Pattern Recognition*, (Anchorage, United States), pp. 1–8, IEEE Computer Society, June 2008.

[15] M. Ziaeefard and R. Bergevin, “Semantic human activity recognition: A literature review,” *Pattern Recognition*, vol. 48, no. 8, pp. 2329–2345, 2015.

[16] A. Sanin, C. Sanderson, M. T. Harandi, and B. C. Lovell, “Spatio-temporal covariance descriptors for action and gesture recognition,” in *2013 IEEE Workshop on Applications of Computer Vision (WACV)*, pp. 103–110, Jan 2013.

[17] E. Aarts and R. Wichert, “Ambient intelligence,” in *Technology Guide: Principles – Applications – Trends* (H.-J. Bullinger, ed.), pp. 244–249, Berlin, Heidelberg: Springer Berlin Heidelberg, 2009.

[18] S. Mitra and T. Acharya, “Gesture recognition: A survey,” *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 37, pp. 311–324, May 2007.

[19] A. Bleiweiss, D. Eshar, G. Kutliroff, A. Lerner, Y. Oshrat, and Y. Yanai, “Enhanced interactive gaming by blending full-body tracking and gesture animation,” in *ACM SIGGRAPH ASIA 2010 Sketches*, SA ’10, (New York, NY, USA), pp. 34:1–34:2, ACM, 2010.

[20] N. Gosalia, P. Jain, I. Shah, A. R. Joshi, N. Katre, and S. Sahasrabudhe, “3D gesture-recognition based animation game,” *Procedia Computer Science*, vol. 45, pp. 712–717, 2015. International Conference on Advanced Computing Technologies and Applications (ICACTA).

[21] S. B. Erica A. Cartmill and S. Goldin-Meadow, “A word in the hand: action, gesture and mental representation in humans and non-human primates,” *Philosophical Transactions of the Royal Society B*, vol. 367, pp. 129–143, 2012.

[22] A. Kendon, *Gesture: Visible Action as Utterance*. Cambridge University Press, 2004.

[23] L. Miranda, T. Vieira, D. Martínez, T. Lewiner, A. W. Vieira, and M. F. M. Campos, “Online gesture recognition from pose kernel learning and decision forests,” *Pattern Recognition Letters*, vol. 39, pp. 65–73, 2014. Advances in Pattern Recognition and Computer Vision.

[24] D. McNeill, *Hand and Mind: What Gestures Reveal about Thought*. Hand and Mind: What Gestures Reveal about Thought, University of Chicago Press, 1992.

[25] S. Goldin-Meadow, *Hearing Gesture: How Our Hands Help Us Think*. Belknap Press of Harvard University Press, 2005.

[26] H. Sakoe and S. Chiba, “Dynamic programming algorithm optimization for spoken word recognition,” in *Readings in speech recognition* (A. Waibel and K.-F. Lee, eds.), pp. 159–165, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1990.

[27] M. Shokoohi-Yekta, B. Hu, H. Jin, J. Wang, and E. Keogh, “Generalizing DTW to the multi-dimensional case requires an adaptive approach,” *Data Mining and Knowledge Discovery*, vol. 31, pp. 1–31, Jan 2017.

[28] S. Salvador and P. Chan, “FastDTW: Toward accurate dynamic time warping in linear time and space,” in *3rd Workshop on Mining Temporal and Sequential Data*, 2004.

[29] A. Paraschos, C. Daniel, J. R. Peters, and G. Neumann, “Probabilistic movement primitives,” in *Advances in Neural Information Processing Systems 26* (C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, eds.), pp. 2616–2624, Curran Associates, Inc., 2013.

[30] A. Paraschos, C. Daniel, J. Peters, and G. Neumann, “Using probabilistic movement primitives in robotics,” *Autonomous Robots*, vol. 42, no. 3, pp. 529–551, 2018.

[31] J. Friedman, T. Hastie, and R. Tibshirani, *The Elements of Statistical Learning. Data Mining, Inference, and Prediction*. No. 12 in Springer Series in Statistics, Springer Series in Statistics, 2 ed., 2017.

[32] C. M. Bishop, *Pattern recognition and machine learning*. Springer, 3 ed., 2009.

[33] P. M. Engel and M. R. Heinen, “Incremental learning of multivariate gaussian mixture models,” in *Advances in Artificial Intelligence – SBIA 2010*, pp. 82–91, Springer Berlin, Heidelberg, 2010.

[34] J. E. Gentle, *Computational statistics*, vol. 308. Springer, 2009.

[35] T. Zhang, “Solving large scale linear prediction problems using stochastic gradient descent algorithms,” in *ICML 2004: Proceedings of the twenty-first International Conference on Machine Learning. Omnipress*, pp. 919–926, 2004.

[36] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, “Scikit-learn: Machine learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[37] W. Xu, “Towards optimal one pass large scale learning with averaged stochastic gradient descent,” *arXiv preprint arXiv:1107.2490*, 2011.

[38] L. McInnes, J. Healy, J. Melville, *et al.*, “umap-learn: UMAP reference implementation,” 2018–.

[39] L. McInnes, J. Healy, and J. Melville, “UMAP: Uniform manifold approximation and projection for dimension reduction,” 2018.

[40] F. Han, B. Reily, W. Hoff, and H. Zhang, “Space-time representation of people based on 3D skeletal data: A review,” *Computer Vision and Image Understanding*, vol. 158, pp. 85–105, 2017.

[41] L. Wang, J. Zhang, L. Zhou, C. Tang, and W. Li, "Beyond covariance: Feature representation with nonlinear kernel matrices," in *2015 IEEE International Conference on Computer Vision (ICCV)*, pp. 4570–4578, Dec 2015.

[42] J. Cavazza, A. Zunino, M. S. Biagio, and V. Murino, "Kernelized covariance for action recognition," in *2016 23rd International Conference on Pattern Recognition (ICPR)*, pp. 408–413, Dec 2016.

[43] M. Meshry, M. E. Hussein, and M. Torki, "Linear-time online action detection from 3D skeletal data using bags of gesturelets," in *2016 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pp. 1–9, IEEE, 2016.

[44] Y. Du, W. Wang, and L. Wang, "Hierarchical recurrent neural network for skeleton based action recognition," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[45] A. A. Chaaraoui, J. R. Padilla-López, P. Climent-Pérez, and F. Flórez-Revuelta, "Evolutionary joint selection to improve human action recognition with RGB-D devices," *Expert systems with applications*, vol. 41, no. 3, pp. 786–794, 2014.

[46] A. Shahroudy, G. Wang, T. Ng, and Q. Yang, "Multimodal multipart learning for action recognition in depth videos," *CoRR*, vol. abs/1507.08761, 2015.

[47] R. Vemulapalli, F. Arrate, and R. Chellappa, "Human action recognition by representing 3D skeletons as points in a lie group," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014.

[48] E. Cippitelli, S. Gasparrini, E. Gambi, and S. Spinsante, "A human activity recognition system using skeleton data from RGBD sensors," *Computational intelligence and neuroscience*, vol. 2016, p. 21, 2016.

[49] H.-J. Jung and K.-S. Hong, "Enhanced sequence matching for action recognition from 3D skeletal data," in *Asian Conference on Computer Vision*, pp. 226–240, Springer, 2014.

[50] S. Fothergill, H. M. M. , P. Kohli, and S. Nowozin, "Instructing people for training gestural interactive systems," in *CHI '12 Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 1737–1746, ACM, May 2012.

[51] G. Maeda, M. Ewerton, R. Lioutikov, H. Ben Amor, J. Peters, and G. Neumann, "Learning interaction for collaborative tasks with probabilistic movement primitives," in *2014 IEEE-RAS International Conference on Humanoid Robots*, pp. 527–534, Nov 2014.

[52] G. Maeda, "Time alignment of trajectories using local time warping." Accessed 10th of Sept. 2019, available here: `https://github.com/gjmaeda/LocalTimeWarping/`.

[53] J. Zhang, W. Li, P. O. Ogunbona, P. Wang, and C. Tang, "RGB-D-based action recognition datasets: A survey," *Pattern Recognition*, vol. 60, pp. 86–105, 2016.

[54] S. Ruffieux, D. Lalanne, E. Mugellini, and O. Abou Khaled, "A survey of datasets for human gesture recognition," in *Human-Computer Interaction. Advanced Interaction Modalities and Techniques* (M. Kurosu, ed.), (Cham), pp. 337–348, Springer International Publishing, 2014.

[55] R. S. Olson, W. La Cava, P. Orzechowski, R. J. Urbanowicz, and J. H. Moore, "PMLB: a large benchmark suite for machine learning evaluation and comparison," *BioData Mining*, vol. 10, pp. 36:1–36:11, Dec 2017.

[56] J. Vanschoren, J. N. van Rijn, B. Bischl, and L. Torgo, "OpenML: networked science in machine learning," *CoRR*, vol. abs/1407.7722, 2014.

[57] Wikipedia, "List of datasets for machine-learning research." Accessed 29th of Sept. 2019, available here: `https://en.wikipedia.org/wiki/List_of_datasets_for_machine-learning_research#Action_recognition`.

[58] W. Li, Z. Zhang, and Z. Liu, "Action recognition based on a bag of 3D points," in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition – Workshops, CVPRW 2010*, pp. 9–14, July 2010.

[59] S. Herath, M. Harandi, and F. Porikli, "Going deeper into action recognition: A survey," *Image and Vision Computing*, vol. 60, pp. 4–21, 2017. Regularization Techniques for High-Dimensional Data Analysis.

[60] A. B. Sargano, P. Angelov, and Z. Habib, "A comprehensive review on handcrafted and learning-based action representation approaches for human activity recognition," *Applied Sciences*, vol. 7, no. 1, 2017.

[61] S. Schaal, J. Peters, J. Nakanishi, and A. Ijspeert, “Learning movement primitives,” in *Robotics Research. The Eleventh International Symposium* (P. Dario and R. Chatila, eds.), (Berlin, Heidelberg), pp. 561–572, Springer Berlin Heidelberg, 2005.

[62] G. Maeda, M. Ewerton, R. Lioutikov, H. Ben Amor, J. Peters, and G. Neumann, “Learning interaction for collaborative tasks with probabilistic movement primitives,” in *2014 IEEE-RAS International Conference on Humanoid Robots*, pp. 527–534, Nov 2014.

[63] O. Dermy, M. Chaveroche, F. Colas, F. Charpillet, and S. Ivaldi, “Prediction of human whole-body movements with AE-ProMPs,” in *2018 IEEE-RAS 18th International Conference on Humanoid Robots (Humanoids)*, pp. 572–579, Nov 2018.

[64] C. Bhattacharyya, L. Grate, M. I. Jordan, L. E. Ghaoui, and I. S. Mian, “Robust sparse hyperplane classifiers: application to uncertain molecular profiling data,” *Journal of Computational Biology*, vol. 11, no. 6, pp. 1073–1089, 2004.

[65] W. Li, Z. Zhang, and Z. Liu, “Action recognition based on a bag of 3D points,” in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition – Workshops*, pp. 9–14, June 2010.

[66] T. Oliphant, “NumPy: A guide to NumPy.” USA: Trelgol Publishing, 2006–. [Online; accessed 3rd Oct. 2019].

[67] E. Jones, T. Oliphant, P. Peterson, *et al.*, “SciPy: Open source scientific tools for Python,” 2001–. [Online; accessed 3rd Oct. 2019].

[68] W. McKinney, “Data structures for statistical computing in python,” in *Proceedings of the 9th Python in Science Conference* (S. van der Walt and J. Millman, eds.), pp. 51–56, 2010.

[69] J. Nocedal and S. Wright, *Numerical optimization*. Springer Science & Business Media, 2006.

[70] R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu, “A limited memory algorithm for bound constrained optimization,” *SIAM Journal on Scientific Computing*, vol. 16, no. 5, pp. 1190–1208, 1995.

[71] C. Zhu, R. H. Byrd, P. Lu, and J. Nocedal, “Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization,” *ACM Trans. Math. Softw.*, vol. 23, pp. 550–560, Dec. 1997.

[72] S. C. Endres, C. Sandrock, and W. W. Focke, “A simplicial homology algorithm for lipschitz optimisation,” *Journal of Global Optimization*, vol. 72, pp. 181–217, 2018.

[73] T. Hastie, R. Tibshirani, J. Friedman, and J. Franklin, “The elements of statistical learning: data mining, inference and prediction,” *The Mathematical Intelligencer*, vol. 27, no. 2, pp. 83–85, 2005.

[74] F. Zhou and F. Torre, “Canonical time warping for alignment of human behavior,” in *Advances in Neural Information Processing Systems 22* (Y. Bengio, D. Schuurmans, J. D. Lafferty, C. K. I. Williams, and A. Culotta, eds.), pp. 2286–2294, Curran Associates, Inc., 2009.