Karlsruhe Institute of Technology

Domain Knowledge Infusion in Machine Learning for Digital Signal Processing Applications

An in-depth case study on table tennis stroke recognition

Master's Thesis of

Christoph Ludwig Richard Wieland

At the Department of Informatics

Reviewer:	PD Dr. Victor Pankratius
Second reviewer:	Prof. Dr. Michael Beigl
Advisor:	PD Dr. Victor Pankratius

Duration: December 10, 2020 – June 9, 2021

Karlsruher Institut für Technologie
Fakultät für Informatik
Postfach 6980
76128 Karlsruhe

I declare that I have developed and written the enclosed thesis completely by myself, and have not used sources or means without declaration in the text.

I followed the rules for securing a good scientific pracise of the Karlsruhe Institute of Technology (Regeln zur Sicherung guter wissenschaftlicher Praxis im Karlsruher Institut für Technologie (KIT)).

Karlsruhe, June 9, 2021

Signed Wieland

. . . . . .

(**Christoph Ludwig Richard Wieland**)

Abstract

This work explores the infusion of domain knowledge as a way to improve machine learning applications in signal processing. Table tennis stroke detection is used here as an in-depth exemplary use case with signals collected from the sensors of a wrist-worn smartwatch. Domain knowledge is leveraged at various abstraction levels to improve stroke detection and classification, ranging from low-level signal corrections, context-dependent sensor fusion, stroke type characteristics, to valid gameplay state machines that allow self-correction of misclassifications. An LSTM based prototype is shown to successfully distinguish between play/no play, stroke/no stroke, length of stroke, eight types of strokes (drive, loop, block, push for forehand/backhand), as well as to predict metrics of future strokes based on past strokes. Training and validation was performed by two semi-professional table tennis players using a table tennis robot serving 3770 precisely controlled shots.

Zusammenfassung

Diese Arbeit untersucht die Infusion von Domänenwissen als eine Möglichkeit zur Optimierung von Anwendungen des maschinellen Lernens in der Signalverarbeitung. Als Anwendungsbeispiel wird die Erkennung von Tischtennisschlägen anhand von Signalen detailliert analysiert. Die Signale stammen von Sensoren, die in einer am Handgelenk getragenen Smartwatch integriert sind. Domänenwissen wird auf verschiedenen Abstraktionsebenen verwendet, um die Schlagerkennung und -klassifikation zu verbessern. Diese reichen von der Wahl und Fusion tischtennisrelevanter Sensoren, über Low-Level-Signalkorrekturen, bis hin zu Zustandsautomaten, die basierend auf dem Wissen über gültige Schlagsequenzen eine Selbstkorrektur von Fehlklassifikationen ermöglichen. Die Evaluation des LSTM-basierten Prototyps zeigt, dass er erfolgreich zwischen Spiel/kein Spiel, Schlag/kein Schlag, und acht Schlagarten (Vorhand/Rückhand Konter, Topspin, Block, Unterschnitt) unterscheiden kann, sowie Metriken zukünftiger Schläge zur Analyse des Spielstils basierend auf vergangenen Schlägen vorhersagen kann. Das System wurde basierend auf 3770 Schlägen von zwei langjährigen Tischtennisspielern entwickelt und validiert. Die Daten wurden in einer kontrollierten Umgebung unter Zuhilfenahme eines Tischtennisroboters gesammelt, der Bälle präzise servieren kann.

Contents

Abstract v

Zusammenfassung vii

1 Introduction 1

2 Related Work 3
2.1 Domain Knowledge Infusion . . . 3
2.2 Table Tennis Stroke Recognition . . . 4
2.2.1 Academic Context . . . 4
2.2.2 Patents . . . 6
2.3 Summary . . . 7

3 Technical Background 9
3.1 Long Short-Term Memory Network . . . 9
3.2 Finite State Machine . . . 11
3.3 Relevant Signal Processing Techniques . . . 12
3.3.1 Spline Interpolation . . . 12
3.3.2 Signal Energy . . . 12
3.3.3 Wavelet Transform . . . 13
3.3.4 Digital Filter . . . 15
3.3.5 Z-Score . . . 16
3.3.6 Linear Regression . . . 16
3.4 Summary . . . 16

4 Domain Knowledge 17
4.1 Terminology . . . 17
4.2 Table Tennis . . . 18
4.2.1 Stroke Phases . . . 18
4.2.2 Stroke Types . . . 19
4.2.3 Stroke Sequences . . . 20
4.2.4 Table Tennis Robot . . . 21
4.3 Influence of Domain Knowledge on Data Acquisition . . . 21
4.4 Summary . . . 22

5 The Table Tennis Stroke Recognition System 23
5.1 Data Collection Environment . . . 23
5.2 Sensors . . . 25
5.3 Modules . . . 25
5.3.1 Data Collection Module . . . 25
5.3.2 Preprocessing Module . . . 25
5.3.3 Model Training Module . . . 26
5.3.4 Stroke Extraction & Classification Module . . . 26

5.3.5 Stroke Analysis Module . . . 27
5.4 Apps . . . 27
5.4.1 Data Collection . . . 27
5.4.2 On-Device Inference . . . 32
5.5 Summary . . . 36

6 Internals of the Processing Pipeline 37
6.1 Preprocessing . . . 37
6.2 Machine Learning Models . . . 43
6.2.1 Early Stopping . . . 43
6.2.2 Sliding Window . . . 43
6.2.3 Classifier Architecture . . . 43
6.2.4 Forecaster Architecture . . . 44
6.3 Stroke Extraction and Classification . . . 45
6.3.1 Stroke and Non-Stroke Noise Inference . . . 45
6.3.2 Stroke Extraction . . . 46
6.3.3 Stroke Classification . . . 50
6.4 Stroke Analysis . . . 51
6.5 Influence of Domain Knowledge on Processing . . . 53
6.5.1 Preprocessing . . . 53
6.5.2 Stroke Recognition . . . 53
6.6 Summary . . . 54

7 Evaluation 55
7.1 Data . . . 55
7.1.1 Data Collection Process . . . 55
7.1.2 Data Sets . . . 56
7.2 Machine Learning Models . . . 57
7.2.1 Performance Metrics . . . 57
7.2.2 Classifiers . . . 58
7.2.3 Forecaster . . . 62
7.2.4 Mobile Device Optimization . . . 62
7.3 Player-Dependent Models . . . 63
7.4 Stroke Extraction . . . 64
7.5 Stroke Classification . . . 65
7.6 Stroke Analysis . . . 68
7.7 Excursus: Amateur Data . . . 69
7.8 Summary . . . 70

8 Discussion and Outlook 71
8.1 Edge Devices and the Machine Learning Models . . . 71
8.2 Sensor Placement . . . 72
8.3 Individuality of Table Tennis . . . 73
8.4 Future Work . . . 73
8.4.1 Improvements . . . 73
8.4.2 Suggestions . . . 74
8.5 Summary . . . 75

9 Conclusion 77

Bibliography 79

Appendix **83**
A Exemplary Stroke Signals 83
B Exemplary Preprocessing Results 88
C Stroke Execution 93
D Transition Probabilities 97
E Evaluation 98
E.1 Stroke Extraction and Classification 98
E.2 Stroke Analysis 100

List of Figures

3.1 A simple RNN with a one hidden unit A (input x_t, output h_t) and a recurrent weight matrix W_{hh} (adapted from [13]). . . . 9
3.2 An LSTM unit containing the cell state C_t, and the three gates f_t, i_t, and o_t (adapted from [13]). . . . 10
3.3 A highly simplified version of this work's non-deterministic finite state machine. . . . 12
3.4 A simple spline interpolation through the points (0,1), (1,4), (2,2), (3,5), (4,1), and (5,3). The separate spline segments are colored differently. . . . 13
3.5 The decomposition function (left) and the reconstruction function (right) of the bior3.9-wavelet. . . . 13
3.6 Two different signals composed of several sinusoids (1, 2, 5, and 10 Hz) and their respective spectra after applying a Fourier transform (adapted from [22]). . . . 15

4.1 Backswing phase a forehand drive. . . . 18
4.2 Stroke phase of a forehand drive. . . . 18
4.3 Swing-out phase a forehand drive. . . . 19
4.4 Return phase a forehand drive. . . . 19
4.5 The shakehand grip in forehand and backhand. . . . 20
4.6 Unrolled Stroke Sequence State Machine for the drill "Forehand, Middle, Backhand". . . . 20
4.7 Donic Robo-Pong 2040. . . . 21
4.8 Domain knowledge infused data collection tasks. . . . 21

5.1 Custom illustration of a Fossil Gen 5E Carlyle HR with its coordinate system and specifications. . . . 23
5.2 The watch placements: In the left case, only one watch is placed on the racket-holding wrist. In the right case, an additional watch is positioned slightly above the right knee. . . . 24
5.3 The temporal data collection and testing environment in the author's basement consists of a table tennis table and a training robot. . . . 24
5.4 The five modules of the processing pipeline and their respective device contexts. RD/PD: raw/preprocessed data, ES: extracted strokes, TC/TF: trained classifiers/forecaster . . . 26
5.5 The IMU Logger prompts the user to select the usage mode (standalone/-client) at its initial startup. . . . 28
5.6 User interface of the standalone IMU Logger. . . . 28
5.7 The process of scanning and connecting to a server if the smartwatch acts as a client. . . . 29
5.8 GATT Server UI with the abilities to start the server (here: already pressed), start and stop logging, and to change the current configuration. . . . 30

5.9 Visualization of the interaction between user, server, and a client during a data collection process. 31
5.10 Simplified view of the primary classes of the IMU Logger. The IMU Logger only initializes the *GattClient* if it acts in client mode. 32
5.11 Simplified view of the *GattServer* class of the GATT Server app. 32
5.12 An exemplary on-device inference process. Images (a)–(j) show the UI of the TT Classifier, while image (k) enlarges the plot of (j) for better readability. 33
5.13 Simplified class diagram of the TT Classifier containing its primary classes and their relationships. 35

6.1 Activity diagram of the preprocessing workflow. For better distinction, dashed paths indicate additional steps only required for training. Input: raw data with labels. Output: preprocessed data. 38
6.2 The acceleration and pressure data of a time series containing eight forehand drives. The raw pressure data does not show a trend. The preprocessed data contains non-stroke noise actions (black) and the raw data (pale). 39
6.3 The pressure data of a time series containing eleven forehand drives. The raw pressure data shows a trend. The preprocessed data contains non-stroke noise actions (black) and the raw data (pale). 39
6.4 A sliding window of size 10 consecutive data points $= 100\,\text{ms}$. 43
6.5 The classifier architecture. X: number of input features, Y: number of output classes. 44
6.6 The architecture of the Stroke Forecaster. X: number of input features, Y: number of output features. 45
6.7 The impact of the noise factor 1.5 on the decision space. The area under each curve represents non-stroke noise actions, while the area above each curve represents valid strokes. 46
6.8 A classified time series containing eight backhand loops. 46
6.9 Illustration of the three stroke extraction methods applied on a stroke sequence with eight forehand drives. For acceleration, it only visualizes *FULL* strokes. The pressure curve is not smooth because its values are truncated after five decimal places. 48
6.10 Stroke sequences containing eight forehand drives (left) compared to eight backhand drives (right). The cumulative acceleration of each backhand stroke shows strong fluctuations, while the forehand strokes are smooth. . . 49
6.11 The fully connected Stroke Sequence State Machine used to enrich the LSTM stroke classifications with domain knowledge from typical table tennis drill descriptions (see [18]). 50
6.12 The impact of tanh on the state machine probabilities. 51
6.13 Visualization of the delta calculation (see Equation 6.12). The delta between both curves is positive for both positive forecasted values and negative ones. 52
6.14 Domain knowledge infused tasks in preprocessing and stroke recognition. . . 53

7.1 Training of player one's Stroke, Noise, Stroke Future, and Combined Model. 61
7.2 Confusion matrices of player one's Stroke and Combined Model on his test set. 61
7.3 Comparison of the confusion matrices when mixing Stroke Models and test data from two different players. 63
7.4 Confusion matrices of player two's Stroke and Combined Model on his test set. 64

7.5 Comparison of the classification of eight backhand blocks using the modular approach with and without the state machine. Green: correct classifications, red: misclassifications. 67
7.6 An incorrect stroke extraction causes the second misclassification when using the modular approach. The Stroke Sequence State Machine could not cure it. 67
7.7 Faulty stroke classifications using the non-modular approach. The Stroke Sequence State Machine could not solve either misclassification. 67
7.8 The cumulative acceleration and angular velocity of a sequence containing eight forehand drives compared to their forecasted values. 68
7.9 Inference of amateur data with pretrained Stroke Models. 69

8.1 Alternative sensor placement (Arduino Nano 33 BLE Sense) directly on the racket handle. 72
8.2 The confusion matrices of the Stroke Models of both players on their respective test sets. In both cases, it is most difficult for the models to distinguish drives and blocks. 73

A.1 Raw sensor data of eight forehand drives. 83
A.2 Raw sensor data of eight forehand loops. 84
A.3 Raw sensor data of eight forehand blocks. 84
A.4 Raw sensor data of eight forehand pushes. 85
A.5 Raw sensor data of eight backhand drives. 86
A.6 Raw sensor data of eight backhand loops. 86
A.7 Raw sensor data of eight backhand blocks. 87
A.8 Raw sensor data of eight backhand pushes. 88
B.9 Interim results of the preprocessing pipeline applied to a time series containing eight forehand drives. The preprocessed data contains non-stroke noise actions (black) and the raw data (pale). 88
B.10 Interim results of the preprocessing pipeline applied to a time series containing eleven forehand drives. The preprocessed data contains non-stroke noise actions (black) and the raw data (pale). 91
C.11 Execution of a forehand drive. 93
C.12 Execution of a forehand loop. 94
C.13 Execution of a forehand push. 94
C.14 Execution of a forehand block. 95
C.15 Execution of a backhand drive. 95
C.16 Execution of a backhand loop. 96
C.17 Execution of a backhand push. 96
C.18 Execution of a backhand block. 97
E.19 Exemplary stroke extraction and classification results on the test set of player 1. Comparison of the modular and the non-modular output. 98

List of Tables

4.1 A selection of drill descriptions from [18]. 20

5.1 Sensors of the Fossil Gen 5E Carlyle HR. 25
5.2 List of time series parameters with their possible values in English and German. 28

7.1 Total number of recorded strokes per stroke type and data set (player one | player two). 56
7.2 Confusion matrix (adapted from [39, p. 92]). TP: true positives, FP: false positives, TN: true negatives, FN: false negatives. 57
7.3 Performance metrics of various Stroke Models trained on several input feature combinations of player one's data. L: loss, F1: F1-score. 59
7.4 Comparison of alternative models using player one's accelerometer, gyroscope, and magnetometer data. L: loss, F1: F1-score. 60
7.5 The training results of the Stroke Future Model. L: loss, RMSE: root mean squared error, MAE: mean absolute error. 62
7.6 Influence of model shrinking on model size, including the floating point operations needed for inference (FLOPS). 62
7.7 Training results of player two's machine learning models. 64
7.8 The stroke extraction results on the test sets of player one (P1) and player two (P2). 65
7.9 The average number of windows containing non-stroke noise actions on the test sets of player one (P1) and player two (P2). Pre Noise refers to windows before and Post Noise to windows after a stroke sequence. 65
7.10 The stroke classification results on the test sets of player one (P1) and player two (P2). 66
7.11 The average deviation in acceleration, angular velocity, velocity, and angle on the test set strokes performed by player one (P1) and player two (P2). . 69

D.1 The transition probabilities of the Stroke Sequence State Machine derived from [18]. Note that the transitions from the state X to the Waiting state are not mentioned, because these transitions are only used at the end of a time series to return to the Waiting state and therefore do not need to be considered here. 97
E.2 The stroke analysis results of the strokes visualized in Appendix E.1. Only the modular approach is considered. 100

Chapter 1

Introduction

Today's mainstream machine learning techniques have gaps in explainability and traceability of their inner workings [1]. Given the popularity of machine learning and its integration into all kinds of decision processes, new regulations are pressing for more technical accountability. An example is the European right to not be subjected to an exclusively automated decision-making process (Art. 22 GDPR, Recital 71). The inclusion of human domain knowledge in machine learning is a promising complement to enhance the explainability of processes that used to be black boxes. However, research on domain knowledge infusion in this area is still sparse, which is why this thesis will conduct an in-depth case study to evaluate new research opportunities. It addresses the research question "Can machine learning benefit from domain knowledge to compute more trustworthy and reliable outputs?".

The automated detection of table tennis strokes in sensor signals serves as the basis for this study. Concretely, this thesis explores ways to incorporate interdisciplinary domain knowledge into different steps of the processing pipeline, ranging from the selection of appropriate sensors and preprocessing techniques to result interpretation. This use case is interesting, as human activity recognition "has gained significant interest from computer scientists and researchers over the past decade because of its wide range of applications" [2]. So far, however, there is a lack of such monitoring systems in the context of table tennis.

This work presents a machine learning based table tennis stroke recognition system capable of extracting, classifying, and analyzing strokes in fused sensor signals provided by an accelerometer, a gyroscope, a magnetometer, and a pressure sensor integrated into a wrist-worn smartwatch. All in all, the system considers eight stroke types: drive, loop, block, and push, each in forehand and backhand. It explicitly leverages domain knowledge to automatically self-verify and self-correct its machine learning based stroke classifications. In addition to the classification of strokes, the system is able to detect non-stroke actions in recorded time series. The complete processing can either run on a computer or directly on a smartwatch. Players can use the system to classify and monitor stroke sequences in a controlled environment where a table tennis robot serves balls precisely.

The thesis is structured as follows: The first chapter after this introduction discusses related work in the areas of domain knowledge infusion in machine learning and table

tennis stroke recognition (Chapter 2). The subsequent chapters present basic knowledge relevant to this work, including technical fundamentals (Chapter 3) and table tennis related knowledge (Chapter 4). Chapters 5 and 6 introduce the table tennis stroke recognition system. The first presents a high-level view of the data acquisition environment, relevant sensors, and system components. The second dives deeper into the system internals and explains the complete processing pipeline. The subsequent chapter evaluates the system components on data from two semi-professional table tennis players (Chapter 7). Chapter 8 discusses the use of machine learning models on edge devices, alternative embodiments, and challenges due to the individuality of table tennis. It closes with possible future advancements of the system. This thesis ends with a conclusion about the impact of domain knowledge on machine learning (Chapter 9).

Chapter 2

Related Work

The major research topics of this thesis are the infusion of domain knowledge in machine learning and the recognition of table tennis strokes. The following sections discuss projects with similar subjects. In general, there is currently not much research on domain knowledge infusion in machine learning for digital signal processing applications. This is especially the case in the context of human activity recognition, such as table tennis stroke recognition. For this reason, Section 2.1 only presents some general thoughts and ideas on domain knowledge infusion along with exemplary use cases. The subsequent section discusses related work from research and industry in the area of table tennis stroke recognition (Section 2.2). The last part of this chapter briefly summarizes the differences between related work and this work's approach (Section 2.3).

2.1 Domain Knowledge Infusion

Islam et al. [3] introduced a system to detect network attacks such as DDoS or Web Attacks based on a data set containing criteria for each attack type. They pursued the goal of making the detection of such attacks more explainable. For this purpose, domain knowledge represented by the CIA principle (confidentiality, integrity, availability), "a popular network security principle" [3], is leveraged to reduce the feature space from 78 features per attack type to 22 more understandable domain features. They compared these features with multiple classifiers, such as artificial neural networks, support vector machines, or random forests. While the results for the domain features are slightly worse than the results using the complete feature set, the domain knowledge infused approach "provides better explainability" [3] and "better execution time and resiliency with unknown attacks" [3].

Radovanović et al. [4] proposed a stacking logistic regression framework that integrates domain knowledge to reduce the impact of non-linear relationships between input attributes on the final classification. They consider the binary prediction of thirty-day hospital readmissions as an exemplary use case. Clinical Classification Software (CSS) provides domain knowledge, more precisely, hierarchically arranged information on "costs, utilization, and outcomes associated with particular diagnoses and medical procedures" [4]. First,

Radovanović et al. cluster these multi-level data to create input features for the classifier. Next, they apply linear regression stepwise based on these hierarchical groups instead of considering all input attributes at once. Thus, the likelihood increases that a classification step only considers features with linear relationships. This modification of the classical linear regression approach leads to improved classification accuracy and precision. In addition, the decision process becomes easier to understand.

Kursuncu et al. [5] developed a concept to integrate domain knowledge directly into the learning process of neural networks. For this purpose, they developed a neural network layer that takes the previous hidden representation, the second-previous hidden representation, and a vector representing domain knowledge as an input. Here, knowledge graphs contain these domain-specific concepts and their relationships. A custom embedding function turns knowledge into vectors. Kursuncu et al. developed two additional functions for training: the Knowledge-Aware Loss Function decides "whether to infuse knowledge or not at a particular stage in learning" [5], and the Knowledge Modulation Function merges "the differential knowledge representation with the learned representation" [5]. The authors hope that the combination of knowledge graphs and deep learning "will further enhance the performance and accelerate the convergence of advanced learning processes" [5]. Furthermore, infusing domain knowledge during learning could increase the explainability of decision-making processes, reduce the need for large data sets for training, and "potentially avoid social discrimination" [5] caused by automatic decisions.

The first two mentioned papers use domain knowledge for feature engineering and extraction. Islam et al. use it to reduce the feature space and Radovanović et al. to create a "deep feature extraction model" [4] whose structure relies on domain knowledge. Kursuncu's work, on the other hand, follows an entirely different approach. It integrates domain knowledge directly into the learning process. All of them aim for more reliable and explainable decision-making processes. In contrast, the thesis at hand takes the approach of infusing domain knowledge after classification to make the system more reliable. The developed system verifies its classification results based on domain knowledge and adjusts them automatically as needed. In addition, this thesis discusses the influence of domain knowledge on design decisions throughout the complete processing pipeline, from feature selection to their preprocessing, classification, and the interpretation of final results.

2.2 Table Tennis Stroke Recognition

This section presents prior art in the field of stroke recognition in table tennis and other racket sports, including findings from the academic context (Section 2.2.1) as well as from patents (Section 2.2.2). None of them use domain knowledge to improve classification results or LSTM-networks for classification, even though this neural network architecture is beneficial for classifying and analyzing time series. Except for one patent, they only provide stroke recognition capabilities, not stroke analysis. More differences are individually discussed per publication.

2.2.1 Academic Context

Liu et al. [6] presented a system for table tennis stroke recognition based on human inertial data. The system can detect five types of strokes: forehand drive, block shot, forehand chop, backhand chop, and smash. It consists of a sensor network for data acquisition (upper arm, lower arm, back) and a wireless module to send data via Wi-Fi to a computer for further processing. The sensor network collects acceleration and angular velocity. The system detects strokes if both properties exceed a threshold (twice the mean of the raw data) in one-second sliding windows. It extracts the following features for classification

purposes: mean, variance, kurtosis, covariance, correlation coefficient, skewness, energy, and spectral entropy. PCA is applied to reduce the feature dimensions. The system uses an SVM for stroke classification. In total, Liu et al. collected 270 samples from nine players. The system achieved an accuracy of 97.41 % on the collected data. In contrast to this thesis, Liu et al. consider fewer stroke types (only five compared to eight), collected data from only two sensors instead of four (accelerometer, gyroscope, magnetometer, pressure sensor), and use derived features rather than raw data for classification. Furthermore, Liu et al. use a sliding window approach with fixed-sized windows to extract strokes. Thus, each detected stroke lasts a multiple of one second, regardless of whether the movement is actually shorter.

Dokic et al. [7] developed a system that can only distinguish between forehand and backhand strokes using a simple feedforward network. They do not consider different stroke types. The data comes from an Arduino Nano 33 BLE Sense worn at the racket-holding wrist. This device collects acceleration and angular velocity and sends it via BLE to a computer responsible for preprocessing and training. The network is deployed directly on the device to classify strokes. The system achieved an accuracy of 100 % on forehand strokes and 96 % on backhand strokes, respectively. Compared to this thesis, the scope of Dokic's work is significantly smaller. The system only distinguishes between forehand and backhand, which makes it somewhat unsuitable for real-world use.

Fu et al. [8] developed a table tennis stroke recognition system using a smartwatch to gather sensor data and a CNN for classification. The system relies on acceleration, angular velocity, and magnetic field. The smartwatch sends collected data to an Android smartphone via BLE, which forwards it to a web server on a computer that performs stroke detection. Strokes are extracted based on the variances in each data dimension within a sliding window. If the summed variances are greater than a threshold, the system considers the window as a stroke. Subsequently, the system combines all data within a time window into one large vector and uses it as input for the CNN. The extraction and classification process is completely done on the computer, not on the smartwatch itself. The system considers a total of eight stroke types: forehand attack, forehand drive, forehand chop, forehand pick, backhand dial, backhand drive, backhand chop, and backhand twist. Fu et al. evaluated it on 2275 strokes from twelve different players, of which they used 1800 for training and the remaining 475 strokes for testing. The system achieved a classification accuracy of 95.46 % on the test set. In contrast to this thesis, Fu's system cannot be used directly on a smartwatch. Instead, a computer performs the processing steps. Furthermore, the need for three devices, a smartwatch for data collection, a smartphone solely for data forwarding, and a computer for processing is not user-friendly compared to the single-device solution of this work. Similar to Liu's approach, stroke detection uses sliding windows, which again create strokes with fixed lengths. Unfortunately, Fu et al. did not provide performance metrics for their stroke extraction approach. They evaluated only the classification of extracted strokes.

Blank et al. [9] developed a system, which is capable of detecting and classifying four different stroke types per hand (forehand/backhand): topspin, drive, block, and push. They use a miPod sensor attached to a table tennis racket to collect acceleration and angular velocity. Blank et al. collected data from ten players with predefined two-player exercises. An expert inspects the data and decides between valid strokes and random racket movements, serves, or faulty strokes. He manually labeled 1982 of the 3004 collected ball contacts as valid strokes. The system detects strokes by applying a Butterworth high-pass filter to the energy of the accelerometer signals along with a threshold function containing the mean and the standard deviation. Afterward, the researchers define each stroke interval as $[t - 600\,\text{ms}, t + 400\,\text{ms}]$, where t denotes the timestamp of a detected peak in the energy signal. All in all, this algorithm detected 3097 ball contacts, of which

the human expert manually marked 1971 as valid shots. Blank et al. used only the 1971 detected valid strokes for further evaluation of the classifier. Compared to the 1982 actual valid strokes, the detection algorithm achieved a precision of 95.7 % and a recall of 98.2 %. They tested stroke classification with several classical approaches, such as SVM and kNN. For this purpose, they use a total of 60 features, e.g., mean, kurtosis and energy. The SVM achieved the best result with an accuracy of 96.7 % for the 1971 detected valid strokes. One major difference between Blank's work and this thesis is the sensor placement. Blank et al. attached a sensor directly to the racket. This likely results in more fine-grained data compared to the smartwatch solution used in this thesis, as it allows the detection of wrist movements important for some table tennis strokes, instead of arm movements, but comes with the downsides of changed racket weight and balance, and the need of charging the racket from time to time, which makes this solution less user-friendly. Furthermore, stroke extraction solely relies on one characteristic, namely the energy of the acceleration. However, table tennis is a highly individual sport, and sometimes accelerations are smaller when performing the same stroke type. Hence, using more features for extraction would probably result in better results.

2.2.2 Patents

Jin et al. [10] invented a system that is capable of extracting and classifying table tennis strokes. Furthermore, it provides information about the strength and speed of a stroke. The system collects acceleration and angular velocity data with a device that is worn at the racket-holding wrist. This is explicitly not a smartwatch, but a device with a display, Bluetooth module, and sensors specially made for this application. The system detects motion by looking at the acceleration: it detects a stroke if the acceleration at a specific time is greater than the average acceleration during the ten time steps before. The system uses wavelet packet decomposition to compute feature vectors from the acceleration and angular velocity energies. Before classification with a Kohonen neural network, also known as Self-Organizing Map, it reduces these feature vectors by linear discriminant analysis. In contrast to this thesis, Jin et al. use a custom device to collect data. This proprietary solution is not as user-friendly as smartwatches because users must carry and maintain an additional device. In addition, many users already use smartwatches daily, which makes proprietary solutions less attractive. The stroke extraction based on the last ten time steps is an inspiring approach that is probably suitable for real-time stroke recognition. However, it could reach its limits if the strokes are continuously getting slower, e.g., due to exhaustion. In addition, the use of wavelet coefficients for stroke classification likely results in huge feature vectors and, thus, compared to the use of raw sensor data, a higher computational cost for feature space reduction and classification.

Han et al. [11] developed a rally detection system for racket sports. They utilize a stroke classifier that can distinguish between strokes and non-stroke actions. The data comes from an IMU (accelerometer and gyroscope) which is directly attached to the racket. Data are not getting processed directly on the smart racket. Instead, the sensor sends them to a client device responsible for preprocessing data and detecting strokes and rallies. The system detects strokes if the acceleration and the angular velocity in a sliding window are greater than a threshold and uses an SVM for classification. The feature vector consists of acceleration and rotation data in short time windows. In contrast to this thesis, Han et al. did not specifically optimize their system for the domain table tennis, but they claim that the system is suitable in this context.

Kim et al. [12] invented a system for racket sports that uses step and swing information, such as step or swing acceleration, step width, and jump height, to analyze a player's motion and skills. They claim that the results are useful for comparing the athletic perfor-

mance of different players. These features are more general than the table tennis related information considered in the thesis at hand.

2.3 Summary

This selection of previous work illustrates that current research primarily focuses on domain knowledge for feature engineering and extraction. Only a few attempts to explicitly integrate domain knowledge into classification tasks exist. While this research yielded promising results, there is still great potential to incorporate domain knowledge into decision-making processes to make them more reliable and trustworthy.

The existing table tennis stroke recognition systems suffer from limited numbers of detectable stroke types, proprietary sensor devices, unnatural sensor placements, high-dimensional feature spaces, or complex processing equipment that hinder daily use. Neither of them explicitly uses domain knowledge to select appropriate sensor types, optimize processing pipelines, or verify the machine learning based classifications. In addition, they only consider stroke detection and do not provide stroke analysis capabilities. Therefore, plenty of opportunities exist to further develop and improve these systems to make them suitable for everyday use.

This work presents a table tennis stroke recognition system that leverages domain knowledge explicitly to self-verify and automatically adjust machine learning based stroke classifications. Moreover, it integrates domain knowledge implicitly into its system design as the selection of appropriate sensor types, sensor placements, and processing techniques relies heavily on table tennis related knowledge. This approach demonstrates the importance of domain knowledge in interdisciplinary topics. Besides that, this thesis also differs from prior art in terms of the sensors, processing techniques, considered stroke types, and the test environment that includes a table tennis robot. The following chapters present the system and highlight its distinctive features.

Chapter 3

Technical Background

This chapter introduces the theoretical principles that form the basis for the table tennis stroke extraction and classification system developed in the practical part of this master thesis. These include a special kind of recurrent neural networks, the so-called Long Short-Term Memory networks (Section 3.1), finite state machines as behavioral models for analyzing typical table tennis stroke sequences (Section 3.2), and various signal processing techniques (Section 3.3). This chapter ends with a brief summary and an outlook on the next chapter (Section 3.4).

3.1 Long Short-Term Memory Network

Traditional artificial neural networks, such as feed-forward networks, suffer from the problem that they are not able to put their inputs in a temporal context [13]. Therefore, they are unable to use knowledge about previous events to classify the current input. Recurrent neural networks (RNNs) address this issue by adding loops to their hidden layers [13]. These loops describe weighted, recurrent connections between the hidden layers of the current and previous time steps, enabling recurrent neural networks to process temporal sequences of data [13]. Figure 3.1 shows an example of a simple RNN with such a loop.

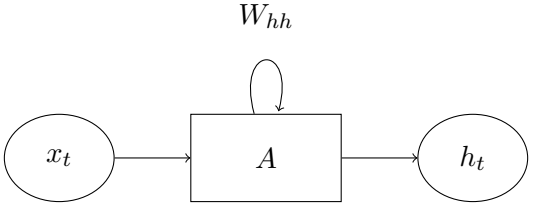


Figure 3.1: A simple RNN with a one hidden unit A (input x_t, output h_t) and a recurrent weight matrix W_{hh} (adapted from [13]).

A drawback of traditional RNNs is that they suffer from vanishing or exploding gradients the longer the time frames are, which makes learning long-term dependencies nearly impossible [14]. This behavior arises from the large number of multiplications of loop weights

during the calculation of the gradients when training with Backpropagation Through Time (BPTT) [15]:

$$\frac{\delta h_{t'}}{\delta h_t} = \prod_{t' \geq i > t} \frac{\delta h_i}{\delta h_{i-1}} = \prod_{t' \geq i > t} W_{hh}^T \, diag(\sigma'(h_{i-1})). \tag{3.1}$$

This gradient vanishes if the largest eigenvalue λ of the recurrent weight matrix W_{hh} is less than 1 and explodes, when λ is greater than 1 [15].

Hochreiter and Schmidhuber introduced a special RNN-version, the so-called Long Short-Term Memory Networks (LSTMs), which addresses vanishing gradients caused by long-term dependencies. They developed complex hidden units that are maintaining their current state in an additive way, instead of the RNNs' simple hidden units with their multiplicatively linked outputs [14]. Typical use cases for LSTMs are time series classification or forecasting. The following is a brief introduction to the behavior of LSTMs. For more information, see [13, 14, 16].

LSTMs consist of multiple LSTM units (see Figure 3.2), containing a cell state C_t, which includes information about the history of the unit, and three gates, namely the forget gate f_t, the input gate i_t, and the output gate o_t, which are responsible for maintaining the cell state and for calculating the output of each unit [13].

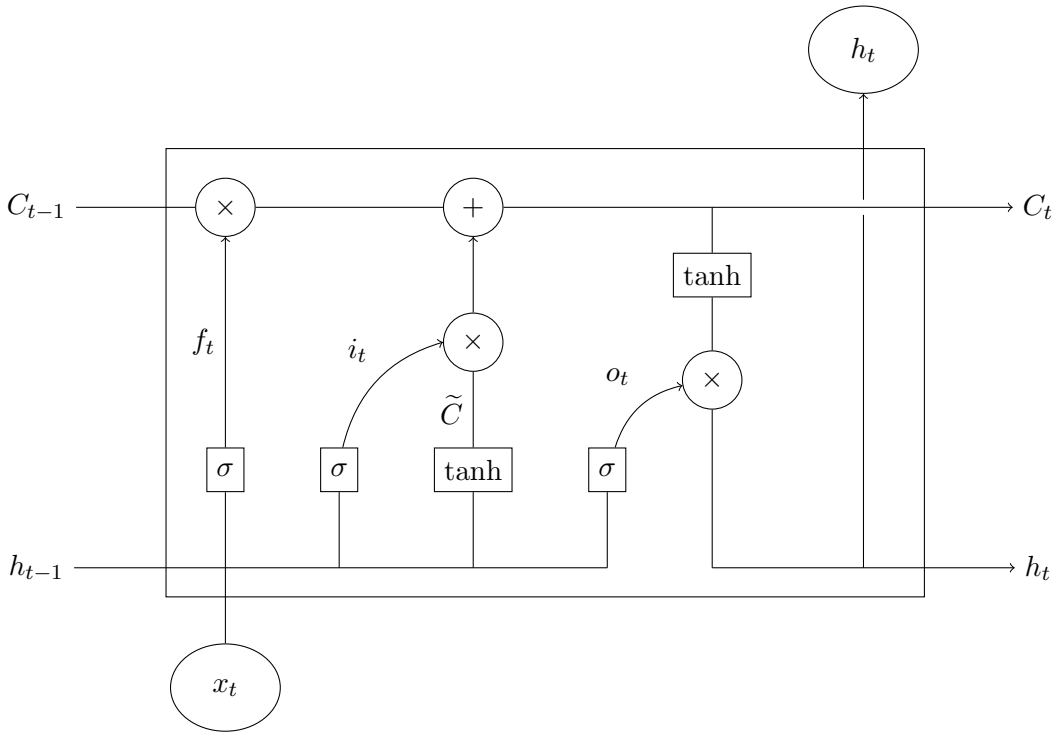


Figure 3.2: An LSTM unit containing the cell state C_t, and the three gates f_t, i_t, and o_t (adapted from [13]).

The forget gate uses a sigmoid layer to decide which information from the old cell state to keep and which to discard by feeding the layer with the output of the previous hidden unit h_{t-1} and the current input x_t [13]:

$$f_t = \sigma(W_f * [h_{t-1}, x_t] + b_f). \tag{3.2}$$

The larger the calculated value, the more information should be retained.

The input gate is responsible for selecting the information to be added to the cell state at a time step [13]: First, it uses a sigmoid layer similar to the one of the forget gate to

determine which values of the cell state to update [13]:

$$i_t = \sigma(W_i * [h_{t-1}, x_t] + b_i). \tag{3.3}$$

Secondly, a tanh layer, which takes the same input h_{t-1} and x_t, computes candidate values for the updates [13]:

$$\widetilde{C}_t = \tanh(W_C * [h_{t-1}, x_t] + b_C). \tag{3.4}$$

Lastly, the results of the first two steps are additively combined and the cell state is updated accordingly [13]:

$$C_t = C_{t-1} * f_t + i_t * \widetilde{C}_t. \tag{3.5}$$

The output gate is responsible for calculating the output of the unit h_t at the current point in time [13]. Again, this calculation takes multiple steps: First, a sigmoid layer determines which parts of the updated cell state C_t to emit, following the same procedure as the sigmoid layers of the forget and the input gates [13]:

$$o_t = \sigma(W_o * [h_{t-1}, x_t] + b_o). \tag{3.6}$$

Secondly, a tanh fits the values of the updated cell state C_t into the interval $[-1, 1]$ [13]. Lastly, the output gate multiplies the results of the previous steps to calculate the final output h_t of the unit at this point in time [13]:

$$h_t = o_t * \tanh(C_t). \tag{3.7}$$

As mentioned above, the cell state is responsible for maintaining long-term dependencies. The derivative of its additive update procedure (Equation 3.5) is as follows [16]:

$$\frac{\delta C_{t'}}{\delta C_t} = \prod_{t' \geq i > t} \sigma(v_i), \tag{3.8}$$

where v_i denotes the input to the forget gate (Equation 3.2). The main difference between the gradient calculation of classical RNNs (Equation 3.1) and the LSTM-version (Equation 3.8) is that the latter does not contain the factor W_{hh} which causes the RNNs' gradients to approach zero for long-term dependencies. Besides the fact that "this quantity can certainly approach zero" [16], LSTMs do not suffer from vanishing gradients caused by intrinsic factors, such as weight matrices [16].

3.2 Finite State Machine

A finite state machine consists of a finite set of states, an alphabet, a transition function, a start, and an end state [17]. For deterministic finite state machines, the transition function is unambiguous [17]. Non-deterministic state machines, on the other hand, allow ambiguous state transitions so that they can reach several states with the same input [17].

This work uses a non-deterministic finite state machine to verify and, if necessary, to adjust the outputs of the classification LSTM. Its states describe the different types of strokes; its transition function represents common table tennis stroke sequences. Each transition has a probability P based on its frequency in the drill descriptions of [18]. So-called ε-transitions [17] allow transitions between states that are not actually connected based on the drill descriptions. Examples are the connections between strokes and the initial *Waiting* state in both directions. Figure 3.3 shows a simplified version of the finite state machine used in this work. For simplicity, it only consists of three states: the initial *Waiting* state, one state that subsumes all types of forehand strokes (*Forehand*), and another one that combines all types of backhand strokes (*Backhand*).

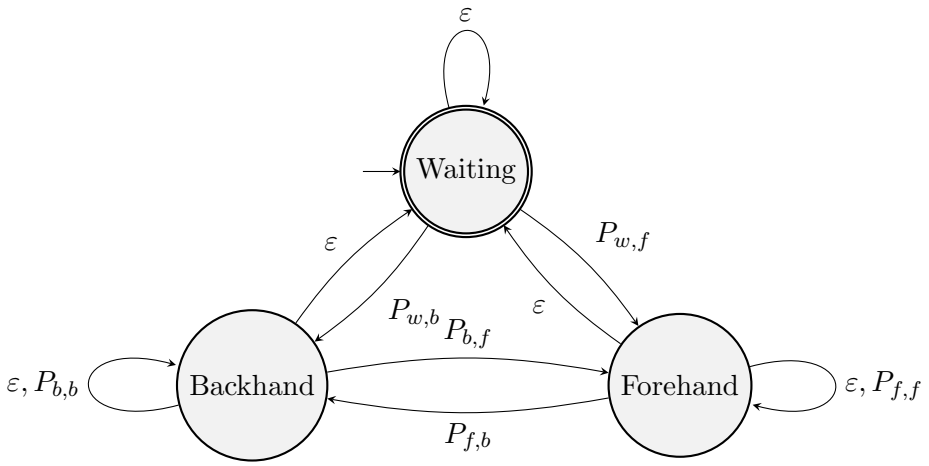


Figure 3.3: A highly simplified version of this work's non-deterministic finite state machine.

3.3 Relevant Signal Processing Techniques

Digital signal processing is an important task when using sensor signals in software applications. This also applies to the practical part of this thesis. The following sections present signal processing techniques relevant to this work.

3.3.1 Spline Interpolation

Interpolation is the task of fitting a continuous curve to n supporting points. Various interpolation techniques exist, such as polynomial interpolation or spline interpolation. Whereas the polynomial interpolation suffers from oscillation in the resulting curves, the spline interpolation provides smoother results [19, pp. 76 ff.]. The basic idea behind the spline interpolation is to fit a polynomial function $s_i(x)$ to each pair of supporting points $[(x_i, y_i), (x_{i+1}, y_{i+1})]$ for $i \in [1, n-1]$ that is two times continuously differentiable [19, pp. 76 ff.]. This results in $n-1$ terms:

$$S(x) = \begin{cases} s_1(x) & , x \in [x_1, x_2] \\ s_2(x) & , x \in [x_2, x_3] \\ \dots & \\ s_{n-1}(x) & , x \in [x_{n-1}, x_n] \end{cases}, \tag{3.9}$$

where the spline must match the points at the interval boundaries:

$$S(x_i) = y_i \text{ for } i \in [1, n]. \tag{3.10}$$

A spline is called "Cubic Spline" if it uses polynomials with rank three to interpolate two points [19, pp. 76 ff.]:

$$s_i(x) = a_i x^3 + b_i x^2 + c_i x + d_i. \tag{3.11}$$

Figure 3.4 shows an example cubic spline interpolation. Please have a look at [19, pp. 76 ff.] for more information about the construction of cubic splines.

3.3.2 Signal Energy

The area under the signal curve describes the signal's energy. Because this work's sensors do not provide continuous but discrete values, the energy of a signal x corresponds to the following expression [20, p. 7]:

$$E_x = \sum_{n=-\infty}^{\infty} |x(n)|^2 . \tag{3.12}$$

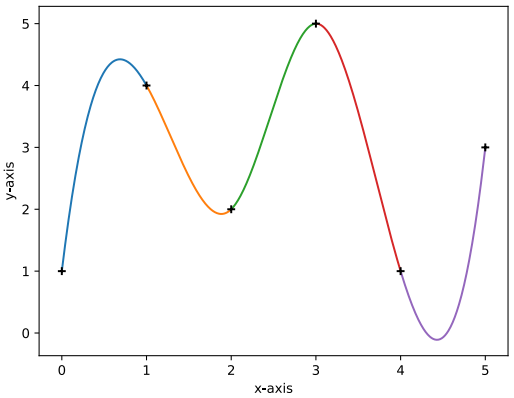


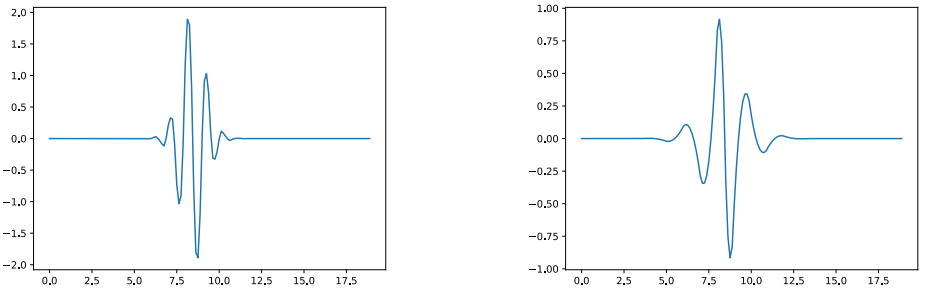
Figure 3.4: A simple spline interpolation through the points (0,1), (1,4), (2,2), (3,5), (4,1), and (5,3). The separate spline segments are colored differently.

This work adjusts the limits of the sum since table tennis strokes only occur in a short period of time $[t_0, t_1]$. This results in the following equation:

$$E_x = \sum_{n=t_0}^{t_1} |x(n)|^2 . \tag{3.13}$$

3.3.3 Wavelet Transform

Wavelet transforms are commonly used in image compression, noise reduction, and pattern recognition tasks [20, p. 311]. Wavelets are wavelike oscillations with specific frequencies, finite energy, and zero mean. The wavelet transform shifts them through a signal to transform it into the time-frequency domain [21, p. 10]. Several wavelet families exist that represent wavelets with similar properties, such as biorthogonal wavelets, Daubechies, or Coiflets [20, p. 342–350]. For instance, biorthogonal wavelets allow smooth reconstructions of input signals due to their symmetry and biorthogonality properties [21, pp. 150 f.]. In the context of this work, practical experiments have shown that the bior3.9-wavelet is best suited for noise reduction of the accelerometer, gyroscope, and magnetometer signals because it preserves their general signal shapes. Figure 3.5 visualizes bior3.9-wavelet.



(a) Decomposition function (b) Reconstruction function

Figure 3.5: The decomposition function (left) and the reconstruction function (right) of the bior3.9-wavelet.

The general idea behind wavelet transforms is to pass multiple versions of a chosen mother wavelet $\psi(t)$ with different frequencies through a signal $x(t)$, so that the wavelet transform can analyze a signal at different frequencies [21, pp. 12 ff.]. The mother wavelet is defined by the scaling factor $a = 1$ and a translation parameter $b = 0$ [21, p. 12]. The scaling factor is responsible for squeezing or stretching the signal in time, which implies a change in the frequency of the chosen wavelet [21, p. 12]. The translation parameter is responsible for shifting the wavelet through the signal [21, p. 12]. The modification of these parameters results in other version of the mother wavelet, the so-called daughter wavelets $\psi_{a,b}(t)$ [22]. Mathematically expressed, the daughter wavelet function is defined as follows [21, p. 14]:

$$\psi_{a,b}(t) = \frac{1}{\sqrt{a}}\,\psi\left(\frac{t-b}{a}\right), \tag{3.14}$$

where the factor $\frac{1}{\sqrt{a}}$ ensures that all daughter wavelets have the same energy.

With this knowledge, Addison [21, p. 14] defines the wavelet transform as:

$$T(a,b) = \int_{-\infty}^{\infty} x(t)\,\psi_{a,b}(t)\,dt. \tag{3.15}$$

In the discrete case, the parameter a is commonly sampled by a logarithmic discretization of the scale and the parameter b is defined by a fixed step size [21, p. 93].

Noise can be reduced in three steps using wavelets [20, pp. 361 f.]:

1. Choose a wavelet, e.g., a biorthogonal one, and apply wavelet packet decomposition to the input signal. This results in a series of n wavelet coefficients c_i.

2. Low coefficients are essentially responsible for noise, while large coefficients carry essential signal information. Therefore, modifying the resulting coefficients by applying a threshold function ε is a valid technique for removing high-frequency noise. For example, this work replaces all coefficients smaller than a threshold with 0 and move the others towards 0:

$$\hat{c_i} = \begin{cases} c_i - \varepsilon & , c_i > \varepsilon \\ c_i + \varepsilon & , c_i < -\varepsilon \\ 0 & , \ |c_i| \leq \varepsilon \end{cases}. \tag{3.16}$$

   This technique is commonly known as soft thresholding [23]. Johnson [24] suggests a threshold of

$$\varepsilon = \sigma * \sqrt{2 * log(n)}, \tag{3.17}$$

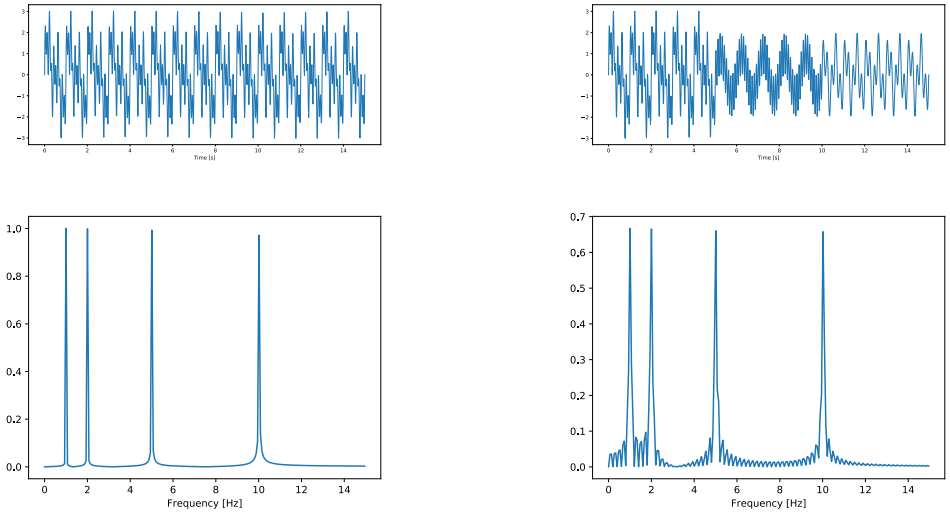
   where σ refers to the median absolute derivation of the wavelet coefficients c, and σ to the noise level of the signal [24, 23].

3. Reconstruct the filtered signal using the wavelet's reconstruction function.

Figure B.9c compares the completely preprocessed acceleration of an exemplary stroke sequence with its original shape. This includes smoothing with the above-mentioned bior3.9 wavelet.

One of the main advantages of wavelet transforms over Fourier transforms is their locality in time and frequency, which makes them particularly useful for analysis of non-stationary, non-periodic signals [22]. Fourier transforms do not offer this property because they always consider the input signals as a whole [22]. Figure 3.6 compares the Fourier transforms of two signals composed of different combinations of the same 1, 2, 5, and 10 Hz sinusoids. The left signal 3.6a combines all the sine waves over the entire fifteen seconds, while the

right signal 3.6b uses three different combinations of the sine waves at three five-second intervals. Specifically, the first interval combines the sinusoids of all frequencies, the second one combines the 1 with the 10 Hz sinus waves, and the third one uses the 2 and the 5 Hz sinusoids. Although the left signal contains all frequencies at all times and the right signal contains different compositions at three intervals, their spectra are very similar.



(a) Stationary, periodic signal composed of all sinusoids (top) with its corresponding spectrum (bottom).

(b) Non-stationary, non-periodic signal composed of three intervals containing different sinusoids (top) with its corresponding spectrum (bottom).

Figure 3.6: Two different signals composed of several sinusoids (1, 2, 5, and 10 Hz) and their respective spectra after applying a Fourier transform (adapted from [22]).

Another important advantage of the Wavelet transform is its linear computational complexity $O(n)$ when using the fast biorthogonal wavelet transform [25, p. 311] compared to the Fast Fourier Transform which requires $O(n \log n)$ operations [20, p. 175].

3.3.4 Digital Filter

Another way to smooth signals is to apply a digital filter [26, p. 261]. One of the most commonly known filters is the moving average filter [26, p. 277]. Despite its simplicity, it is well suited for noise reduction tasks [26, p. 277]. As the name suggests, the moving average filter calculates the average of all values in a fixed range around the current point to be changed. The filter kernel is as follows:

$$\frac{1}{N} * [1]^N, \text{ with } N\text{: filter size.} \tag{3.18}$$

In mathematical terms, we can define the filtered version x' of an input signal x as:

$$x'[n] = \frac{1}{N} \sum_{i=0}^{N-1} x[n-i]. \tag{3.19}$$

This equation is similar to the one proposed in [26, p. 277], but uses the values from the left side of the point to calculate the moving average instead of its right side.

3.3.5 Z-Score

Barnett and Lewis define an outlier in a data set as "an observation (or subset of observations) which appears to be inconsistent with the remainder of that set of data" [27, p. 7]. They also make it clear that inconsistency is a subjective perception. Therefore, it is vital to distinguish how to deal with outliers, i.e., whether they should be removed, adjusted, or remain untouched [27, p. 7]. The analysis of this work's data revealed that its outliers mainly relate to unusually large values in the recorded data. This work adjusts them by comparing the z-scores

$$z(x) = \begin{cases} \frac{x-\mu}{\sigma} & , \sigma_x \neq 0 \\ 0 & , \text{else} \end{cases} \tag{3.20}$$

of each data point x in a time series to a predefined threshold t and clipping their values [28]. Here, μ denotes the mean and σ the standard deviation of the time series. If the absolute z-score of a data point is greater than t, the data point is considered as an outlier and adjusted. This work uses the following metric to calculate the adjusted values $\hat{x}$:

$$\hat{x} = \begin{cases} \mu + t * \sigma & , z(x) > t \\ \mu - t * \sigma & , z(x) < t \\ x & , \text{else} \end{cases}, \tag{3.21}$$

with $t = 3$, because this is a value commonly used in literature [28].

3.3.6 Linear Regression

Robson [29] describes a trend in a time series as a progressive increase or decrease in its values over time. Depending on the observed data, trends can occur for several reasons. Some examples are problems with the data records or environmental factors, such as variations in the climate [29]. Depending on the individual processing goals, scientists must weigh which types of trends to address or tolerate. The analysis of this work's data revealed a trend in some pressure signals (cf. Figure B.10j) the system must treat to enable pressure based stroke detection. As this image suggests, the trend can be considered linear. Therefore, linear regression techniques, such as Least-Squares Fit, can approximate the trend with a line

$$Y = A + BX \tag{3.22}$$

optimized by minimizing the sum of squared residuals E [30, pp. 83 ff.]:

$$\sum_i E_i^2. \tag{3.23}$$

Here, E denotes the distance between an actual data point and its corresponding value on the approximated normalization line. The proposed system reduces trends by subtracting the line from the original signal. Figure B.10k visualizes the pressure data of an exemplary record after removing the linear trend.

3.4 Summary

This chapter introduced fundamental techniques relevant to this thesis, ranging from machine learning based classification methods and behavioral models to relevant signal processing techniques. They form an essential pillar for the table tennis stroke recognition system developed in this work. The next chapter introduces table tennis related domain knowledge as the second pillar.

Chapter 4

Domain Knowledge

Automatic recognition of table tennis strokes in sensor signals is an interdisciplinary task whose solving requires both technical know-how about sensors and processing algorithms as well as knowledge about the domain table tennis. Therefore, it is important to clarify the concept of "domain knowledge". This chapter explains the term "domain knowledge" (Section 4.1) and puts it in the context of table tennis (Section 4.2). It further discusses the influence of domain knowledge on data acquisition (Section 4.3). The last section briefly summarizes the importance of table tennis related knowledge (Section 4.4).

4.1 Terminology

The Oxford University Press defines the terms "domain" and "knowledge" as follows:

- A domain is "an area of knowledge or activity; especially one that somebody is responsible for" [31].
- Knowledge is "the information, understanding and skills that you gain through education or experience" [32].

Consequently, domain knowledge refers to information about a specific topic that an individual (or machine) has acquired, making him or her an expert in that field.

Croft summarizes this by defining domain knowledge as "information about the important topics or concepts in a particular domain and how they relate to each other" [33].

Chiesi et al. give a similar definition. They define "knowledge of a domain as an understanding of its basic concepts, as well as its goals, rules, and/or principles" [34], where "concept involves not simply the definition of the concept but its relations and usage" [34].

These are just a few example definitions of the term "domain knowledge". However, they provide a useful overview of domain knowledge aspects relevant to this thesis.

4.2 Table Tennis

Table tennis is one of the most popular sports in the world with approximately 250 million players worldwide [35, p. 19]. There are numerous ways to play table tennis with countless individual variations possible [35, p. 194]. This section mainly focuses on basic table tennis concepts, which are necessary for understanding this work. These include table tennis related knowledge about the different phases of a table tennis stroke (Section 4.2.1) and typical stroke types (Section 4.2.2). In addition, more than 100 descriptions of typical table tennis drills from [18] guide this work (Section 4.2.3). This section ends with a brief presentation of a table tennis robot that helps players practice their stroke techniques and allows us to create a controlled research environment (Section 4.2.3).

4.2.1 Stroke Phases

A typical stroke movement consists of three phases: the backswing phase, the stroke phase, and the swing-out phase, plus an additional return phase [35, pp. 74 ff.]:

1. Backswing phase: The player moves the racket backward to initialize the next stroke. Depending on the stroke type to be played, the backswing movement is smaller or larger. For example, the backswing motion is much larger when playing a loop than when playing a push.

(a) (b) (c)

Figure 4.1: Backswing phase a forehand drive.

2. Stroke phase: This phase denotes the actual stroke movement, including the ball impact point. The player moves the racket forward until it hits the ball. This phase is crucial to the success of the stroke.

(a) (b) (c) Ball impact point

Figure 4.2: Stroke phase of a forehand drive.

3. Swing-out phase: The racket-holding arm swings out forward after playing a stroke. This phase is an extension of the stroke phase. Similar to the backswing phase, the length of the outswing differs depending on the stroke type. The faster the played stroke is, the longer lasts its swing-out phase.

(a) (b) (c)

Figure 4.3: Swing-out phase a forehand drive.

4. Return phase: The final phase describes the motion back to the base position to get ready for the next stroke. This phase may merge into the backswing phase of the subsequent stroke.

(a) (b) (c)

Figure 4.4: Return phase a forehand drive.

4.2.2 Stroke Types

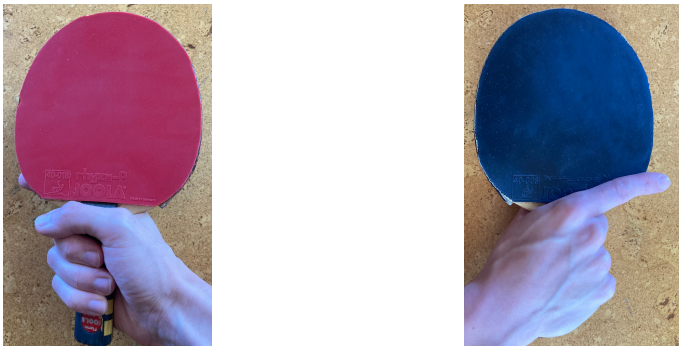
Table tennis is a complex stroke sport in which a ball can be played with numerous stroke techniques and individual variations [35, ch. 3]. The following is a brief introduction to the stroke types relevant to this work. All in all, this work considers four basic stroke types per racket side (forehand/backhand). This results in eight different stroke types:

- Forehand/Backhand Drive (ger.: Vorhand/Rückhand Konter) [35, p. 95]: A drive is a stroke with minimal forward spin that hits the ball almost centered on the slightly closed racket face with a short upward motion.
- Forehand/Backhand Loop (ger.: Vorhand/Rückhand Topspin) [35, pp. 112 f.]: A loop is a versatile, offensive stroke that pulls the ball tangentially from the bottom to the top. Many variations of this stroke type exist to react individually to the incoming ball. They differ in speed, spin, and racket angle. Simply put: The more the player moves the racket forward, the higher the speed. The more he pulls the racket up, the higher the spin.
- Forehand/Backhand Block (ger.: Vorhand/Rückhand Block) [35, p. 144]: A block is a stroke with a short movement that enables the player to position or, if desired, to slightly accelerate the ball. The primary purpose of a block is to respond to an incoming, offensive ball.
- Forehand/Backhand Push (ger.: Vorhand/Rückhand Unterschnitt) [35, p. 81]: A push is a controlled stroke where the racket is pushed under the ball to create backspin. The spin's intensity changes depending on the ball impact point, the acceleration, and the wrist movement.

Appendix C provides demonstrations of each considered stroke type.

There are many variations of these strokes. In addition, each player can individually alter the speed, spin, racket angle, and ball positioning and can also use different racket grips.

This work considers the shakehand grip, a grip technique commonly used by Europeans (Figure 4.5). Here, the hand completely encloses the racket handle, with the index finger resting on the lower edge of the racket face. However, individual variations are possible here as well [35, ch. 2.2]. For more details on the different techniques and other stroke types, see [35, ch. 3].



(a) Forehand (b) Backhand

Figure 4.5: The shakehand grip in forehand and backhand.

4.2.3 Stroke Sequences

In training sessions, players usually play drills to improve their technique or prepare for upcoming matches. These exercises range from simple warm-ups to stroke sequences based on real game situations. [18] contains more than 100 typical table tennis drill descriptions with various levels of complexity and difficulty. Table 4.1 lists some examples. They vary in stroke types to be played, spin, speed, ball placement, and ball frequencies. This thesis considers these drill descriptions as valid gameplay knowledge. We can represent this knowledge using a finite state machine by converting these stroke sequences into state sequences, where each stroke describes a state. Moreover, we can determine the frequencies of different stroke type sequences and calculate transition probabilities accordingly. This thesis refers to this state machine as Stroke Sequence State Machine (cf. Figure 6.11). As an example, Figure 4.6 shows the unrolled state sequence for exercise "Forehand, Middle, Backhand" (cf. Table 4.1). Each stroke sequence starts in the *Waiting* state, in which the player is in his base position. The state machine returns to *Waiting* after the player completes the drill.

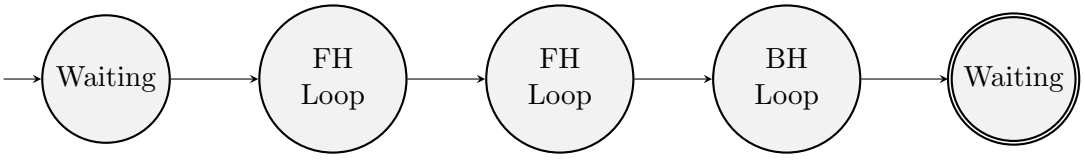


Figure 4.6: Unrolled Stroke Sequence State Machine for the drill "Forehand, Middle, Backhand".

Table 4.1: A selection of drill descriptions from [18].

Drill Name	Stroke Sequence
Backhand Game Situation	BH Push – BH Loop on backspin – BH Loop
Forehand, Middle, Backhand	FH Loop – FH Loop – BH Loop
Short Backhand, Deep Forehand	BH Push – FH Loop
In-And-Out Footwork	FH Push – FH Loop – BH Push – BH Loop

4.2.4 Table Tennis Robot

We use a table tennis robot (Donic Robo-Pong 2040) to create a controlled data collection environment. The robot can serve balls repeatedly with high precision and configurable speed, frequency, and rotation. Table tennis players can use such a device to practice their technique with consistently and regularly played balls. This makes it perfectly suitable to create a research environment with minimized risk of unexpected disruptions. Figure 4.7 shows an image of the robot.

Figure 4.7: Donic Robo-Pong 2040.

4.3 Influence of Domain Knowledge on Data Acquisition

Before collecting training and test data, we must answer some essential questions (cf. Figure 4.8): Where should we place the sensors to capture meaningful data? Which sampling rate is required to capture valuable data? Which sensor types are sufficient to meet the system goals?

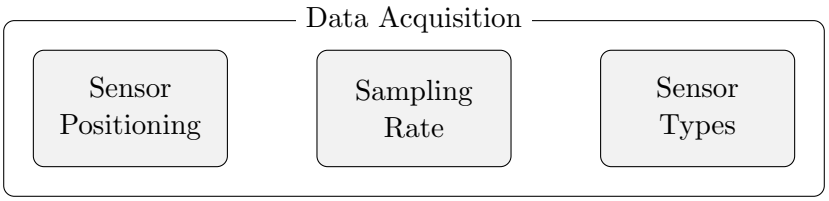


Figure 4.8: Domain knowledge infused data collection tasks.

Depending on the context, system developers or machine learning engineers cannot adequately answer these questions. In table tennis stroke recognition, domain experts hold general knowledge about existing stroke types, stroke execution, and typical game situations (stroke sequences). Table tennis experts know which body parts are mainly involved in stroke movements. Of course, the arm holding the racket provides vital information about stroke movements, but footwork also plays an important role in table tennis. In addition to sensors on the racket-holding arm, sensors attached to the thigh or other body parts could also provide valuable results.

Furthermore, expert knowledge helps to determine the most suitable sampling rate. Table tennis strokes are executed very quickly and only last for a short period of time. Hence, higher sampling rates are necessary to capture essential movement characteristics.

Lastly, domain knowledge helps to select the most appropriate sensor types. Besides the acceleration or the angular velocity, which are crucial in table tennis to accelerate the ball and create spin, other data sources may also be of interest for stroke detection systems. For instance, a domain expert could argue that the height of a table tennis racket changes during a swing, with different strokes having different height variations. Therefore, capturing height differences using a pressure sensor could provide information about the stroke motion from an entirely new perspective. In addition, system developers must also consider the circumstances of the movements when selecting the sensors. An expert could argue that a stroke is not performed in one dimension only; instead, the racket can move freely in space. Consequently, it is worth considering three-dimensional data.

4.4 Summary

Table tennis is a versatile and complex sport with various stroke techniques and opportunities to react to incoming balls. Small nuances in racket angle and speed determine whether a stroke succeeds or fails. Coaches have the important task of analyzing the playing style and technique of their protégés to improve their skills and reduce the likelihood of stroke failures. The automatic analysis of strokes can help players to gain initial insights into their daily performance, identify mistakes, and solve them while they are playing.

This table tennis related knowledge guides the development and design of a table tennis recognition system. For example, it helps select proper sensor types and placements to collect relevant data and provides guidance on what information stroke analysis should provide to players.

The following chapter presents this work's table tennis stroke recognition system.

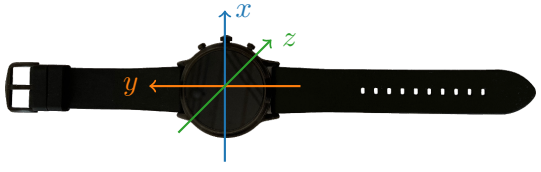
Chapter 5

The Table Tennis Stroke Recognition System

This work aims to explore the impact of domain knowledge on machine learning applications based on a system capable of extracting and classifying table tennis strokes from sensor data. The presented system acts on signals collected from several smartwatches in a controlled environment. This chapter starts with an overview of the data collection environment (Section 5.1) and the used sensors (Section 5.2). Afterward, it introduces the system and its components from a high-level perspective (Sections 5.3 and 5.4). This chapter ends with a brief overview of the table tennis stroke recognition system (Section 5.5).

5.1 Data Collection Environment

The system uses four devices to recognize table tennis strokes: two smartwatches (Fossil Gen 5E Carlyle HR) to collect sensor signals, a smartphone to coordinate the data collection, and a table tennis robot to control the data collection environment. The following sections describe these devices and their configurations.

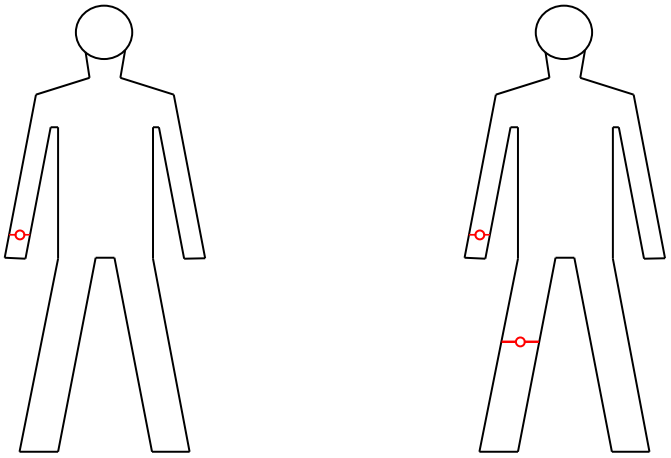


CPU:	Qualcomm® Snapdragon™ Wear 3100 (4x1.2 GHz)
RAM:	1 GB (400 MHz LPDDR3)

Figure 5.1: Custom illustration of a Fossil Gen 5E Carlyle HR with its coordinate system and specifications.

A player can either use the watches standalone to collect data and classify movements on-device or set up a sensor network consisting of several smartwatches as clients and a smartphone as a master. The sensor network is capable of collecting data from multiple sensor devices synchronously. The client devices (two Fossil Gen 5E Carlyle HR smartwatches) connect to the master (a Xiaomi Redmi 9A smartphone) via Bluetooth Low

Energy (BLE). Afterward, the master initiates and stops the data collection process on its clients. Players must wear the clients on their bodies. Figure 5.2 shows the two sensor placement variants considered in this work. In both cases, players wear a smartwatch at their racket-holding wrist. The second variant adds a smartwatch shortly above their right knee. The wrist location naturally provides plenty of information about stroke movements. Because table tennis is a sport in which footwork plays an important role, the hope was that the second position provides insights about a player's body movements.



(a) One watch (b) Two watches

Figure 5.2: The watch placements: In the left case, only one watch is placed on the racket-holding wrist. In the right case, an additional watch is positioned slightly above the right knee.

Initially, we wanted to collect data from players of the local table tennis club and test the system with them. Unfortunately, we had to discard this plan due to the global pandemic (Covid-19), its accompanying contact restrictions, and the closure of the gyms. Therefore, improvisation was necessary. For this reason, we set up a temporal training facility in our private basement consisting of a ping pong table, the training robot, and some table tennis balls (see Figure 5.3).

Figure 5.3: The temporal data collection and testing environment in the author's basement consists of a table tennis table and a training robot.

5.2 Sensors

The system relies on pre-calibrated signals from four different sensor types: a three-axis accelerometer that captures acceleration data in $\mathrm{m\,s^{-2}}$ including gravity, a three-axis gyroscope that captures the angular velocity in $\mathrm{rad\,s^{-1}}$ with automatic gyro-drift compensation, a three-axis magnetometer that captures changes in the geomagnetic field in μT with hard iron calibration, and a pressure sensor that captures the atmospheric pressure in hPa. Table 5.1 lists the specifications of these sensors. The given sensor ranges are read directly from the sensors of a Fossil Gen 5E Carlyle HR through built-in Android functions. More information on their built-in calibrations can be found in the Android WearOS documentation [36].

Table 5.1: Sensors of the Fossil Gen 5E Carlyle HR.

Sensor	Axes	Sensor Range/Full Scale	Sampling Rate
Accelerometer LSM6DSO	x, y, z	$\pm 156.906\,\mathrm{m\,s^{-2}} \approx \pm 16$ g	50 Hz
Gyroscope LSM6DSO	x, y, z	$\pm 34.906\,\mathrm{rad\,s^{-1}} \approx \pm 2000$ dps	50 Hz
Magnetometer AK0991X	x, y, z	$\pm 4912.0\,\mu$T	50 Hz
Pressure ICP101XX		$25.0 - 1150.0$ hPa	25 Hz

According to the Android documentation, accelerometers, gyroscopes, and magnetometers in Android devices are using a coordinate system which "is defined relative to the device's screen when the device is held in its default orientation" [36]. Figure 5.1 illustrates the coordinate system on a Fossil Gen 5E Carlyle HR.

5.3 Modules

The table tennis stroke extraction and classification system consists of several Python scripts packed into five main components: the Data Collection Module, the Preprocessing Module, the Model Training Module, the Stroke Extraction & Classification Module, and the Stroke Analysis Module. Figure 5.4 visualizes the interaction between these modules and gives an overview of which modules run on which devices. All modules operate sequentially and pass their computed output to the next (or multiple upcoming) stages. The following sections present the high-level purposes of these modules. For more details on the internals of the processing pipeline, see Chapter 6.

5.3.1 Data Collection Module

The Data Collection Module manages the acquisition of sensor data. It captures ten raw data streams from four different sensors and transfers them to the Preprocessing Module. These include three acceleration time series, three angular velocity time series, three magnetic field time series, and one air pressure time series. The Data Collection Module must run on smartwatches, as these are the only devices in the system capable of collecting sensor data.

5.3.2 Preprocessing Module

The Preprocessing Module prepares the raw data for further processing and transfers its results to the Model Training Module, the Stroke Extraction & Classification Module, and the Stroke Analysis Module. Preprocessing includes the temporal alignment of the various data streams, potentially coming from multiple smartwatches placed in different locations, spline interpolations, and the application of noise reduction, outlier removal, and trend analysis techniques. For training purposes only, the module also extracts non-stroke actions, such as random movements at the beginning and the end of a time series. The Preprocessing Module can run either directly on a smartwatch for on-device stroke inference or on an external computer, e.g., to validate the system on multiple data sets.

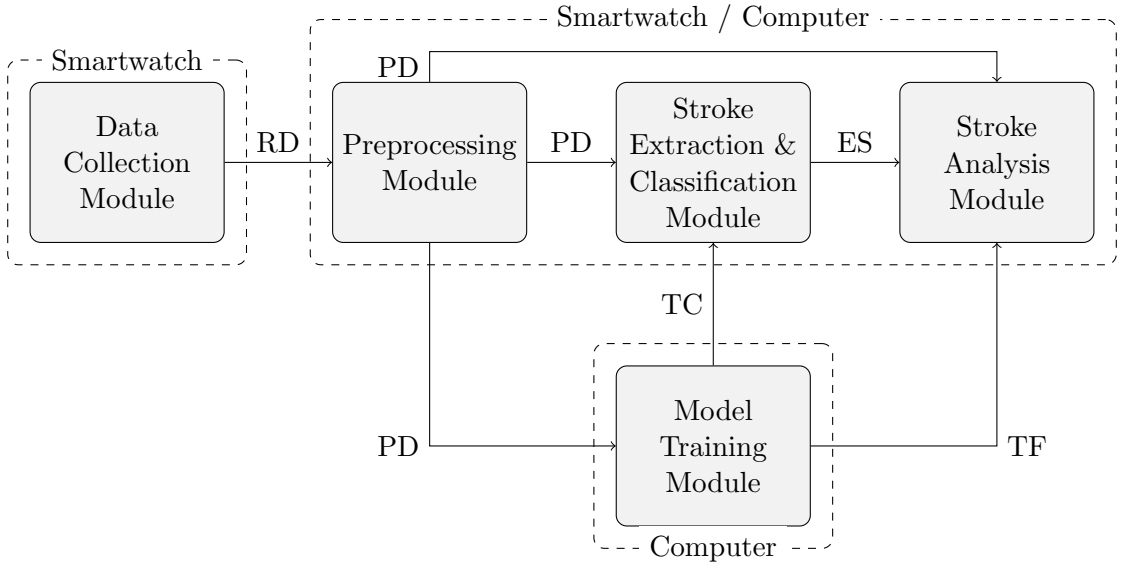


Figure 5.4: The five modules of the processing pipeline and their respective device contexts. RD/PD: raw/preprocessed data, ES: extracted strokes, TC/TF: trained classifiers/forecaster

5.3.3 Model Training Module

The Model Training Module is responsible for training four Keras models (see [37]) based on the preprocessed accelerometer, gyroscope, and magnetometer signals. Three of them act as classifiers and one as a forecaster.

- The Stroke Classifier distinguishes between the eight different types of strokes forehand/backhand loop, drive, push, and block. This classifier operates on the Stroke Model.
- The Noise Classifier extracts time slices from the beginning and the end of a time series containing data unrelated to valid strokes, e.g., small movements while waiting for the first ball to play or other random non-stroke actions. This classifier operates on the Noise Model.
- The Combined Classifier unites the responsibilities of the Stroke and the Noise Model as an alternative classification approach. This classifier operates on the Combined Model.
- The Stroke Forecaster predicts the stroke motion in the next time step based on its learned stroke movement patterns. This forecaster operates on the Stroke Future Model.

After training, the Model Training Module passes the classifiers to the Stroke Extraction & Classification Module and the Stroke Forecaster to the Stroke Analysis Module. Due to the limited amount of processing power in current edge devices, this module runs off-device on a computer.

5.3.4 Stroke Extraction & Classification Module

The Stroke Extraction & Classification Module detects strokes in preprocessed time series in the following way: First, the classifiers label the data. Then, the module compares the results to cut non-stroke actions from the data. Lastly, it extracts stroke intervals using metrics, such as cumulative acceleration or pressure, and labels them with stroke

types based on the classification results and domain knowledge. The Stroke Extraction & Classification Module forwards the labeled stroke intervals to the Stroke Analysis Module for further analysis. The Stroke Extraction & Classification Module can run either directly on a smartwatch for on-device stroke inference or on an external computer for faster inference or to validate the system on multiple data sets.

5.3.5 Stroke Analysis Module

The Stroke Analysis Module forms the last module in the processing pipeline. It takes the preprocessed accelerometer, gyroscope, and magnetometer data as an input for the Stroke Forecaster that predicts future stroke movements based on learned stroke patterns. The module analyzes current strokes by comparing their actual signal data with the forecasted values. This analysis provides information about differences in the average acceleration, speed, angular velocity, and racket angle during a stroke. These results allow the user to infer possible improvements to his stroke movements. In addition, he can gain insights into why a stroke failed, e.g., "the ball fell into the table tennis net because the racket angle was too small" or "the ball flew too far and did not touch the table because acceleration was too high". Again, this module can run either directly on a smartwatch for on-device stroke inference or on an external computer for faster inference or to validate the system on multiple data sets.

5.4 Apps

This work presents a total of three mobile apps: two for data collection (Section 5.4.1) and one for on-device inference (Section 5.4.2).

5.4.1 Data Collection

This work provides two methods to collect the data needed to train the machine learning models of the table tennis stroke recognition system. The first one records data from only one body location using a single smartwatch, and the second one uses a network of several smartwatches distributed over the entire body. The standalone case requires only one application, namely the IMU Logger. This app collects data from the four different sensors accelerometer, gyroscope, magnetometer, and pressure sensor. Although inertial measurement units (IMUs) usually do not include pressure sensors, the name sufficiently clarifies the purpose of the app: the collection of sensor data. On the other hand, the sensor network requires two applications to record signals from multiple locations. The first app serves as a client responsible for data acquisition, while the other one acts as a server that coordinates the data collection on all connected devices. Regardless of whether one device (standalone) or several devices (sensor network) collect sensor signals, the data collection process stays the same. For this reason, the same IMU Logger can either be used standalone or as a client as needed. The user can set the IMU Logger's usage mode at its startup (see Image 5.5).

The second app, namely the GATT Server, turns a smartphone into a master that coordinates data collection on each connected client device. Its name comes from the GATT protocol used for communication between the clients and the server (see [38] for more details on this protocol). Both apps together, the IMU Logger and the GATT Server, represent the Data Collection Module. The following sections describe both the standalone and the sensor network based data collection.

Figure 5.5: The IMU Logger prompts the user to select the usage mode (standalone/client) at its initial startup.

Standalone

A user can collect data from a single body location, such as the racket-holding wrist, using a smartwatch with a standalone IMU Logger installed. Figure 5.6 shows the user interface of the standalone IMU Logger. The user can start data collection directly on the device by pressing the button *START LOGGING*. Afterward, the button changes its value to *STOP LOGGING* and offers the functionality to stop the current data acquisition. After the user stopped logging, the IMU Logger saves the collected data in a .csv file. Additionally, it offers the possibility to change various parameters before recording a time series. These include the smartwatch location, the stroke type with its corresponding racket hand, and the number of strokes to be played. The logger stores the values in the filename of the .csv file using the pattern *YYMMdd_HH.mm.ss_Location_Hand_Type_#Strokes_log.csv*. By maintaining these parameters, the user reduces the data preparation effort before the actual processing tasks, as he labels the collected data directly during data collection. Consequently, recorded signals must not be labeled manually afterward. The maintenance of these parameters is only possible because of the controlled environment created by the configurable table tennis robot. Table 5.2 lists these parameters with their possible values.

Figure 5.6: User interface of the standalone IMU Logger.

Table 5.2: List of time series parameters with their possible values in English and German.

Parameter	Values (en.)	Values (ger.)
Location	{Wrist, Right Thigh}	{Handgelenk, Rechter Oberschenkel}
Hand	{FH, BH}	{VH, RH}
Type	{Base Position, Drive, Loop, Block, Push}	{Grundstellung, Konter, Topspin, Block, Unterschnitt}
#Strokes	{1, 2, ..., 9}	

Sensor Network

To collect data from multiple body locations, the user must initialize a sensor network consisting of several smartwatches as clients and a smartphone as a server. The smartwatches run the IMU Logger in client mode, and the server runs the GATT Server application. The devices communicate via Bluetooth Low Energy (BLE). The connection process is as follows. For simplicity, this description considers only one client.

1. The user starts the server on the smartphone by pressing *START SERVER*. The server offers two GATT characteristics that connected clients can read: the *LOGGING* characteristic initializes and stops data collection, whereas the *LOGGING_PARAMS* characteristic manages the parameters of each data collection operation, e.g., stroke type, number of strokes, or racket hand.
2. The user configures the smartwatch as a GATT client by pressing *CLIENT*.
3. The client automatically searches for a GATT server for up to 60 seconds. The button *CONNECT TO SERVER* becomes available if it found a server; else, the scan stops. The user can restart it manually by pressing the button *SCAN*.
4. The user presses the button *CONNECT TO SERVER*. After that, the client establishes a BLE connection between the GATT client and the GATT. This step includes a subscription to both characteristics offered by the server. Connected clients listen to the characteristics and can thus react to changing values. If the connection fails at any point in time, the client tries to reconnect automatically.

Figure 5.7 visualizes the connection process from a client's point of view. After the client found a server, the user can connect the client by pressing the now enabled button *CONNECT TO SERVER*.

(a) Start scan (b) GATT Server discovered (c) Connect to GATT Server

Figure 5.7: The process of scanning and connecting to a server if the smartwatch acts as a client.

Compared to the standalone mode, the client mode offers fewer modifiable logging parameters. The device location is still changeable directly on a client, but the rest of the parameters are moved to the GATT Server to be set globally (see Figure 5.8). The server notifies the clients if the user changes the global parameters. This ensures that all clients store their collected data under the same filename. Furthermore, the server is responsible for initializing and stopping the collection of sensor data on the clients. The sequence diagram in Figure 5.9 visualizes the consecutive steps involved in data gathering. First of all, the user sets the logging parameters of the next data collection process on the smartphone. The server changes the values of its *LOGGING_PARAMS* characteristic and notifies the clients of the updated configuration. The clients adjust their local parameters. Secondly, the user initializes data collection by pressing the button *START LOGGING* on the GATT

Server (see Figure 5.8). The button switches its text to *STOP LOGGING*, and the server updates the value of the *LOGGING* characteristic. Subsequently, it notifies the clients about the changed characteristic. The clients read its value and start data collection. The info section beneath the *SCAN* button indicates the start of the logging process ("Logging started"). The user stops data collection after completing the training exercise by pressing the button *STOP LOGGING*. The button changes its text back to *START LOGGING*, and the server updates the value of the *LOGGING* characteristic one more time. The clients stop the data collection process after reading the updated characteristic. Finally, each client saves the recorded data in a .csv file. Please note that both BLE operations, the notifications about changed characteristics and reading their values, do not happen instantaneous due to varying latencies. Therefore, the player must wait until all clients have started recording before playing an exercise to ensure that no data are lost.

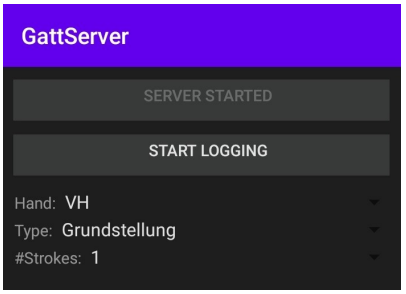


Figure 5.8: GATT Server UI with the abilities to start the server (here: already pressed), start and stop logging, and to change the current configuration.

App Architectures

The following sections present the most relevant aspects of the architectures of the IMU Logger and the GATT Server.

Figure 5.10 visualizes the primary classes of the IMU Logger in a simplified manner:

- The *SensorHandler* coordinates the data logging process of a particular sensor (acceleration, gyroscope, magnetometer, or pressure sensor). It is capable of starting (*startLogging()*) and stopping (*stopLogging()*) the data collection. Furthermore, it acts as an event listener which stores incoming sensor data in its private variable *mData* using the *onSensorChanged()* method. *stopLogging()* returns the collected data.
- The *Logger* takes all data recorded by all *SensorHandlers* and stores it in a .csv file via the method *logDataAsCSV()*.
- The *GattClient* handles the connection between a client and the GATT Server using the *mBluetoothGattCallback*. It handles a wide variety of BLE related functions, such as scanning for and connecting to BLE devices, reconnecting in case of connection failures, and reacting to changed characteristics (*LOGGING* and *LOGGING_PARAMS*). The IMU Logger only initializes the *GattClient* if the user presses the button *CLIENT*.
- The *MainActivity* acts as a manager that coordinates data collection from various sensors using four *SensorHandlers* (one per sensor), storage of the collected data using a *Logger*, and instantiates a *GattClient* if the IMU Logger is in client mode. This class starts and stops data collection from each sensors using the *SensorHandlers*, merges the collected data, and sends them to the *Logger*, which then stores them in a .csv file.

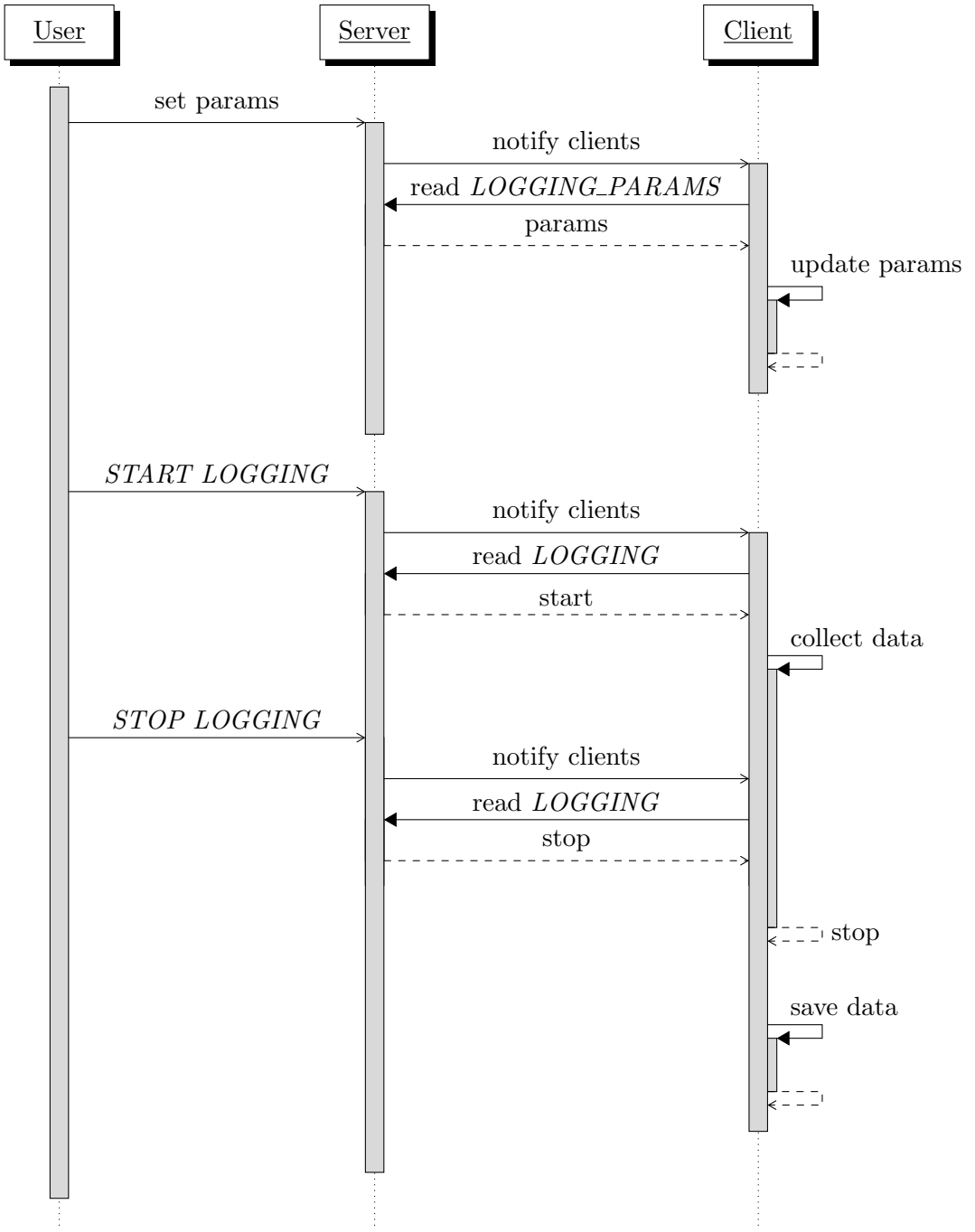


Figure 5.9: Visualization of the interaction between user, server, and a client during a data collection process.

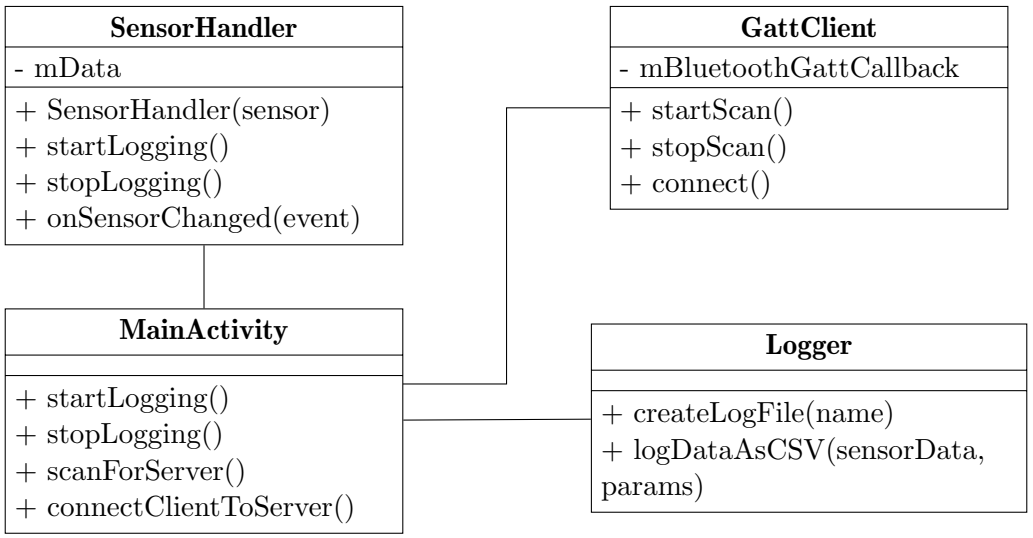


Figure 5.10: Simplified view of the primary classes of the IMU Logger. The IMU Logger only initializes the *GattClient* if it acts in client mode.

One of the most interesting parts of the GATT Server app is its *GattServer*, which is responsible for updating the two characteristics *LOGGING* and *LOGGING_PARAMS* and for notifying connected clients about the changes. Figure 5.11 contains a simplified diagram of this class. The methods *changeStatus()* and *changeParameters()* are designed to change the values of the two characteristics. Both methods call the private method *notifyConnectedDevices(characteristic)* with a modified characteristic as input to notify connected clients on changed characteristic values.

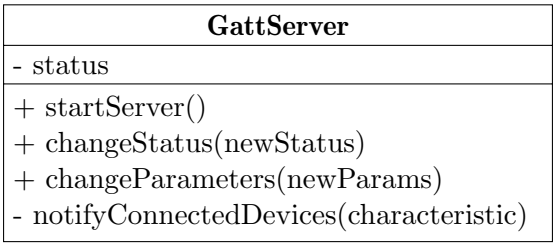


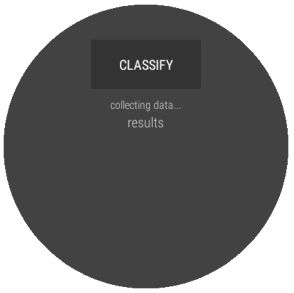
Figure 5.11: Simplified view of the *GattServer* class of the GATT Server app.

5.4.2 On-Device Inference

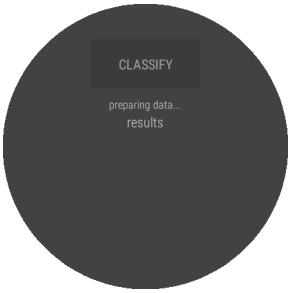
The third app, namely the TT Classifier, collects and processes sensor signals directly on a single smartwatch. It combines the Data Collection Module, the Preprocessing Module, the Stroke Extraction & Classification Module, and the Stroke Analysis Module in one application. Training machine learning models on edge devices with limited processing resources is difficult to realize. Therefore, the Model Training Module is the only module that does not run in this application. The app only uses data from a single smartwatch worn at the racket-holding wrist because there are no significant performance gains that would justify the increased computational overhead when using data from many locations (see Section 7.2.2). The following paragraph describes the workflow of this app. Figure 5.12 visualizes the workflow exemplary during the execution of three backhand pushes.

1. The user initializes the data collection process by pressing the button *START* (Figure 5.12a). The button changes its text to *CLASSIFY* and the info section states that data gets collected (Figure 5.12b). This part of the app runs the Data Collection Module.

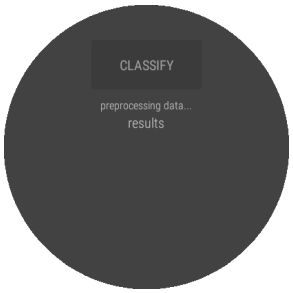
(a) Start



(b) Collecting data

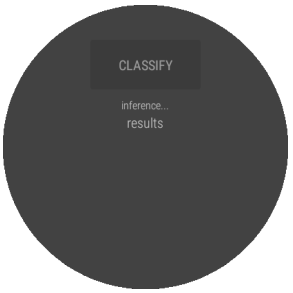


(c) Preparing data



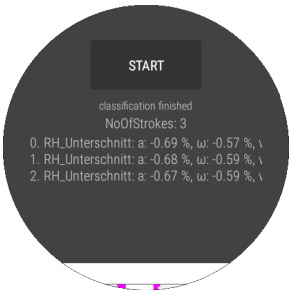
(d) Preprocessing data

(e) Handling data



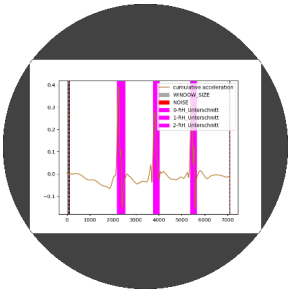
(f) Inference

(g) Calculating strokes

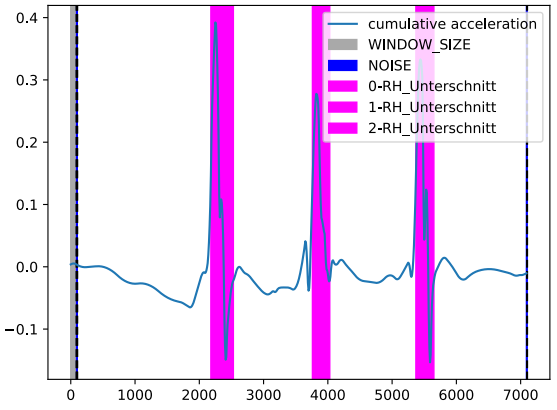


(h) Results (part 1)

(i) Results (part 2)



(j) Plotted results



(k) Plot

Figure 5.12: An exemplary on-device inference process. Images (a)–(j) show the UI of the TT Classifier, while image (k) enlarges the plot of (j) for better readability.

2. After completing an exercise, the user presses the button *CLASSIFY* to stop data collection. The TT Classifier processes the data. This includes preprocessing, stroke extraction and classification, and the analysis of extracted strokes. The user receives textual feedback about the currently executed part of the processing pipeline:

   a) The TT Classifier (Java) prepares the collected data for use in Python. Then, a Python script executed by a Python interpreter within the TT Classifier preprocesses the collected data. Afterward, the TT Classifier converts the preprocessing results from Python to Java for further processing. Figures 5.12c–5.12e illustrates these steps. The Preprocessing Module runs in this part of the app.

   b) The TT Classifier uses the Stroke and the Noise Model (resp. the Combined Model) to infer the preprocessed data. Afterward, it extracts strokes from the time series and categorizes them based on the classification results (Figure 5.12f). Furthermore, it compares the calculated and categorized strokes to learned patterns of previously executed strokes represented by the Stroke Future Model (Figure 5.12g). The Stroke Extraction & Classification Module and the Stroke Analysis Module run in this part of the TT Classifier.

3. The TT Classifier displays the results on the smartwatch (Figures 5.12h–5.12j), including information about the number of detected strokes and their corresponding stroke types. Moreover, it presents percentage deviations in acceleration a and angular velocity r between the forecasted stroke motion and the actually executed strokes, as well as absolute differences in movement speed v and racket angle θ. Below that, the app visualizes the entire time series along with the extracted stroke intervals. Figure 5.12k shows the plot in large for better readability.

This approach cannot perform stroke classifications in real-time because it always considers time series as a whole. For this reason, we developed a second version of the TT Classifier, which is capable of semi-real-time stroke classifications by considering sliding intervals of 2.5 seconds. However, we discarded this approach due to limited processing resources built into current generation smartwatches, such as the Fossil Gen 5E Carlyle HR.

Figure 5.13 visualizes the primary parts of the TT Classifier. The *AbstractTFLiteRunner* is an abstract superclass responsible for instantiating a TensorFlow Lite interpreter with a TensorFlow Lite model which is specified in its subclasses by implementing the *getModelPath()* method. The abstract classes *AbstractClassifier* and *AbstractForecaster* inherit from the *AbstractTFLiteRunner* and extend its functionality with methods for the classification and prediction of given time windows of sensor data. In case of the *AbstractClassifier*, this includes the definition of stroke type labels. Finally, the *StrokeClassifier*, *NoiseClassifier*, and *StrokeFutureForecaster* inherit from their corresponding abstract superclasses and specify the paths to their model and label files by concretizing the methods *getModelPath()* and *getLabelPath()*. The *SensorHandler* manages the data collection process of a given sensor. It operates in the same way as the *SensorHandler* of the IMU Logger. The *MainActivity* coordinates the complete workflow of the TT Classifier, from data collection to stroke extraction and classification, up to stroke analysis: First of all, data are gathered by means of the four sensor handlers. The end user initializes and terminates this step. Afterward, the *mClassifyingRunnable* automatically performs data processing in a background task. This involves the following steps:

1. The *mClassifyingRunnable* prepares the data for preprocessing and passes them to the *mPreprocessingObj*, an object which represents several Python scripts for preprocessing running in an independent Python environment. After the *mPreprocessingObj* finished preprocessing, it passes its results back to the Android WearOS app.

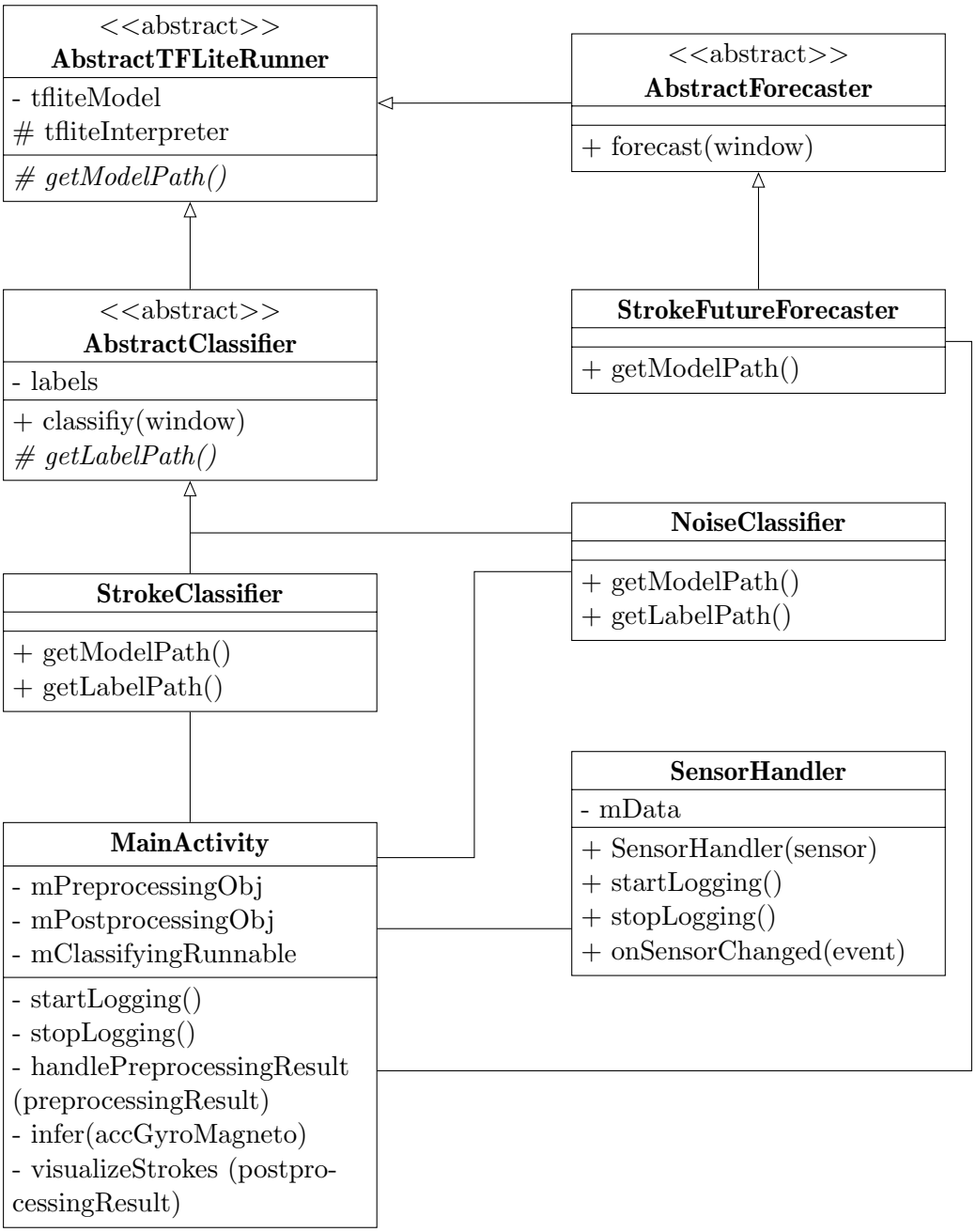


Figure 5.13: Simplified class diagram of the TT Classifier containing its primary classes and their relationships.

2. The method *handlePreprocessingResult()* prepares the preprocessed data for inference with the *StrokeClassifier*, *NoiseClassifier*, and the *StrokeFutureForecaster*. Subsequently, the *mClassifyingRunnable* performs inference via *infer()*.
3. The *mClassifyingRunnable* passes the sensor data, the classifier results, and the forecaster results to the *mPostprocessingObj*, which again represents a collection of Python scripts capable of extracting and analyzing strokes.
4. Lastly, the *mClassifyingRunnable* executes the method *visualizeStrokes()* to visualize the final results calculated by the *mPostprocessingObj*.

5.5 Summary

The table tennis stroke recognition system is designed for use in a controlled environment with a table tennis robot. It can detect, classify, and analyze strokes in sensor signals from an accelerometer, gyroscope, magnetometer, and pressure sensor either on a computer or directly on a smartwatch. The ability to analyze stroke movements directly on an edge device allows players to monitor and refine their strokes during a practice session. The following chapter explains the system's internals, including preprocessing steps, stroke extraction, classification, and analysis.

Chapter 6

Internals of the Processing Pipeline

This chapter explores technical aspects of the developed table tennis stroke extraction and classification system. It first describes the preprocessing workflow (Section 6.1) and the architecture of the machine learning models (Section 6.2). The following parts address stroke extraction, classification, and analysis (Sections 6.3 and 6.4). The chapter ends with a reflection on the influence of domain knowledge on the development of the table tennis stroke extraction and classification system (Section 6.5) and a short summary of its internals (Section 6.6).

6.1 Preprocessing

Preprocessing data is an essential task in machine learning to ensure the learning and generalization capabilities of the system under development. Figure 6.1 visualizes the complete preprocessing workflow used by the table tennis stroke recognition system. The solid paths mark the primary preprocessing steps needed for inference, while the dashed paths denote additional steps needed only for training purposes. The following paragraphs explain each step in detail. Actions that occur multiple times are only considered once. The Figures B.9 and B.10 in the appendix visualize the complete preprocessing process exemplarily for time series containing eight and eleven forehand drives.

Data Preparation

First, the Preprocessing Module must prepare the raw sensor data for further processing. Data preparation consists of two major tasks: temporal alignment of signals and normalization. Alignment is not only necessary when multiple devices collect sensor data in a sensor network, where different recording times may occur due to delays in the Bluetooth connection, but even if only a single smartwatch acts as a data provider. The reason for that is the sequential sensor initialization and termination, which causes the sensors not to start collecting data at the same time. The Preprocessing Module aligns the data in time by finding the least and the greatest common logging time across all sensor signals and truncating any overflowing data point. When using a sensor network, the Preprocessing Module first needs to merge corresponding time series based on the time stamp in their

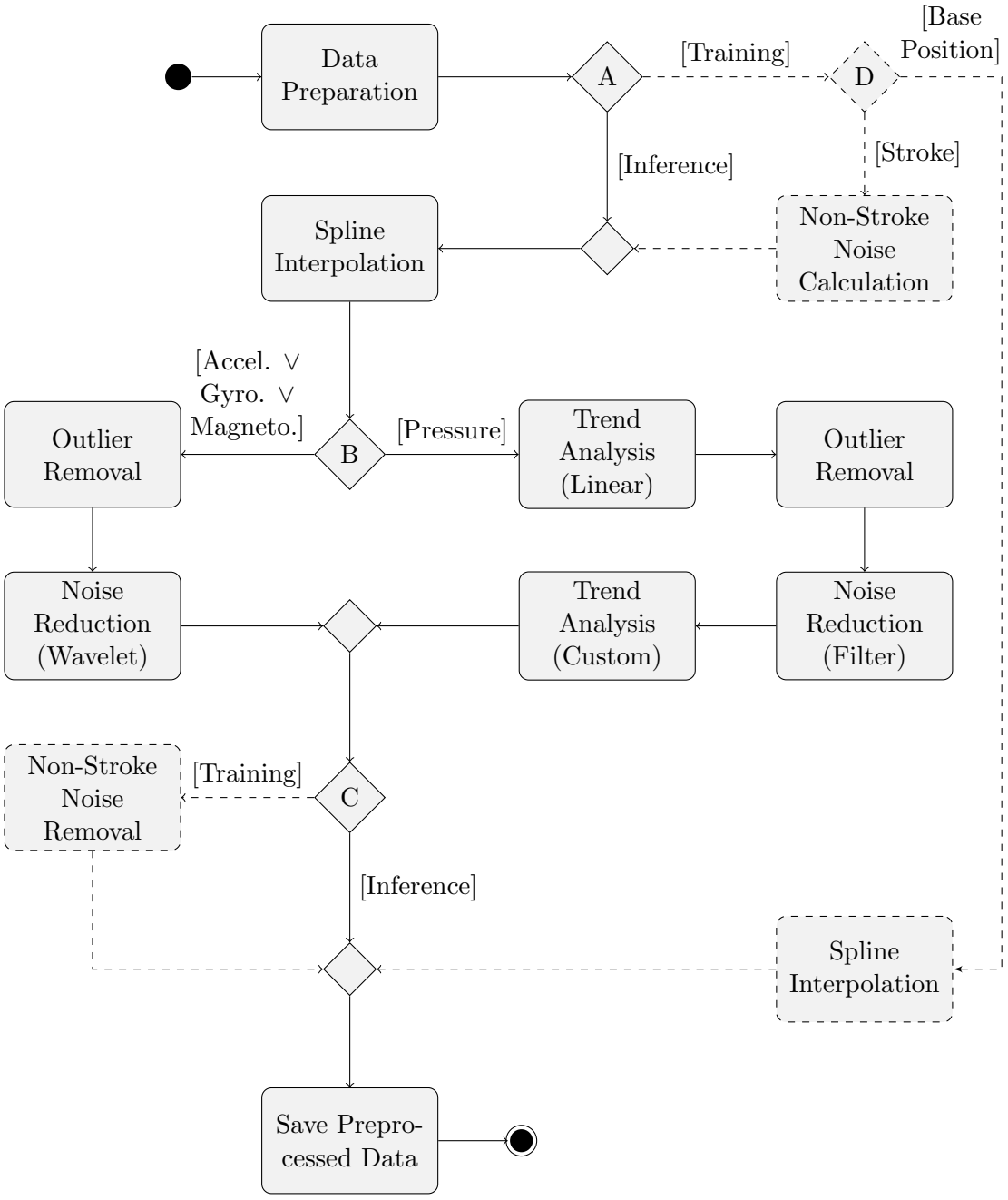
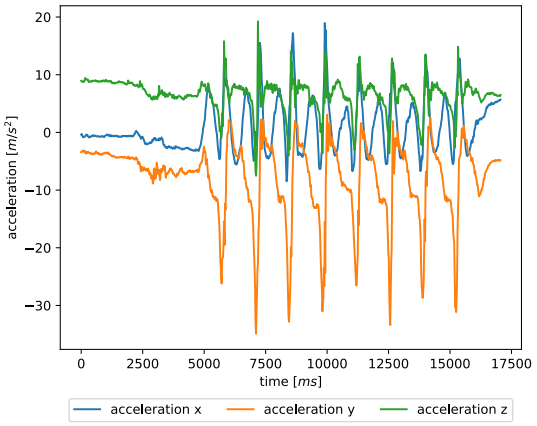
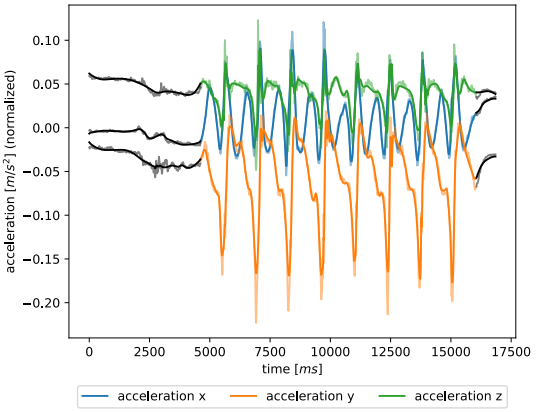


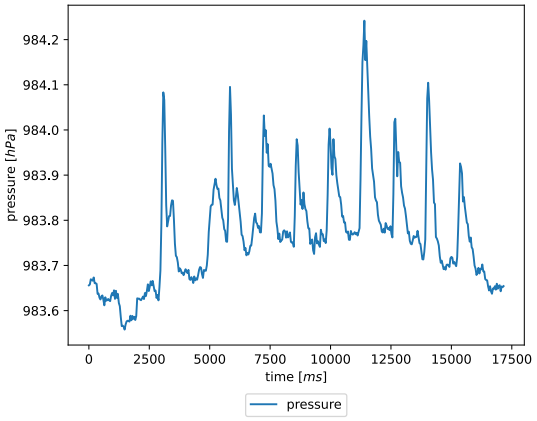
Figure 6.1: Activity diagram of the preprocessing workflow. For better distinction, dashed paths indicate additional steps only required for training. Input: raw data with labels. Output: preprocessed data.



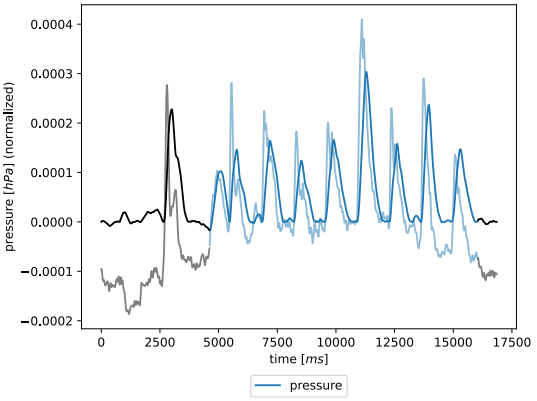
(a) Raw acceleration



(b) Preprocessed acceleration

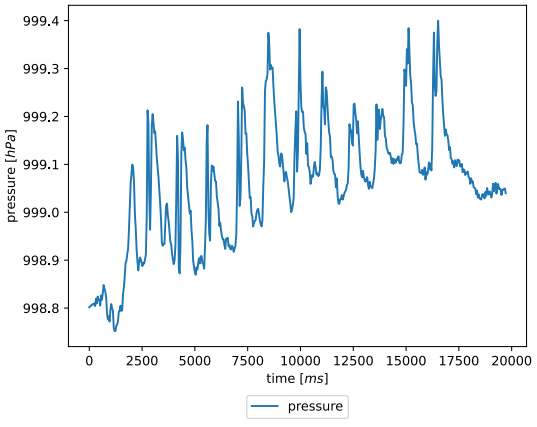


(c) Raw pressure

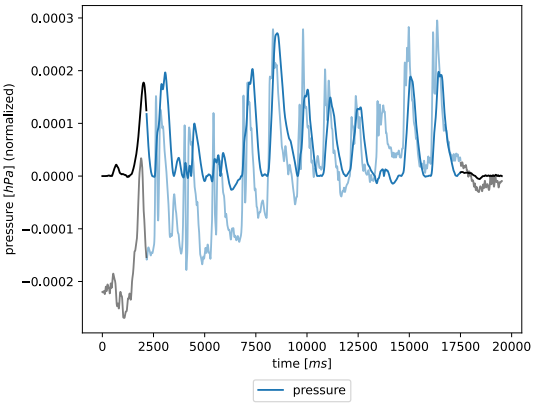


(d) Preprocessed pressure

Figure 6.2: The acceleration and pressure data of a time series containing eight forehand drives. The raw pressure data does not show a trend. The preprocessed data contains non-stroke noise actions (black) and the raw data (pale).



(a) Raw pressure



(b) Preprocessed pressure

Figure 6.3: The pressure data of a time series containing eleven forehand drives. The raw pressure data shows a trend. The preprocessed data contains non-stroke noise actions (black) and the raw data (pale).

file names before the alignment. Afterward, it normalizes each data point into the interval $[-1, 1]$ by dividing the values by the maximum of their corresponding sensor range (see Table 5.1).

Decisions A and C — Preprocessing Purpose

The preprocessing steps differ depending on the currently considered use case. Data preparation for training includes additional activities (dashed paths) in addition to the steps for inference.

Spline Interpolation

Different sensors differ not only in their sensor ranges but also in their respective sampling rates. In the case of the sensors used in this thesis, the sampling rate of the pressure sensor is half the sampling rate of the accelerometer, gyroscope, and magnetometer. Thus, the pressure sensor collects only half as many values as the other sensors in the same time series. Since the Android sensor framework [36] does not offer the possibility to change the sampling rate and instead uses a fixed sampling rate for each sensor, the Preprocessing Module must adjust the collected values. For this purpose, it first interpolates the gathered time series of each sensor with cubic splines. Then, it evaluates these splines with a fixed sampling rate of 100 Hz, i.e., every 10 ms, which is twice the sampling rate of the accelerometer, the gyroscope, and the magnetometer, and four times the sampling rate of the pressure sensor. Consequently, this method provides an approximate solution to the problem of varying sampling rates.

Decision B — Sensor Type

Different sensors require different preprocessing due to individual characteristics in their provided signals. For example, acceleration, angular velocity, and magnetic field only require outlier removal and noise reduction. Pressure data additionally requires trend removal to enable pressure based stroke extraction as some pressure time series show a trend.

Outlier Removal

The collected sensor data occasionally contains unexpected outliers. The Preprocessing Module detects them using the z-score and removes them by simply clipping the erroneous values that are higher than the threshold $t = 3$, which is commonly used in literature [28]. A disadvantage of this method is that unnaturally sharp edges occur at the points where outliers have been detected and removed. A moving average filter of size 3 is applied to the neighborhood of a detected outlier to smooth the data to overcome this issue. The neighborhood has a size of 100 ms, 50 ms in each direction, which corresponds to ± 5 data points.

Noise Reduction

Noise, such as unwanted modifications or jitter, can harm the analysis and interpretation of data. Reducing this noise while preserving the general signal shape is essential for the developed table tennis stroke recognition system. However, this noise should be handled with care, as overly aggressive smoothing can eliminate significant characteristic peaks indicating strokes. Due to the different shapes of the signals acquired by different sensors, this work uses two techniques to reduce noise while preserving the general signal shape. On the one hand, practical experiments have shown that a wavelet transform with a biorthogonal wavelet (bior3.9) is best suited for reducing noise in the acceleration, the

angular velocity, and the magnetic field while retaining the general signal shape and its characteristic peaks. Section 3.3.3 describes the noise removal process using wavelets. On the other hand, a simple moving average filter of size 30 data points = 300 ms is adequate to reduce the noise in the pressure data and to generate well-formed peaks. We chose the filter size experimentally. Larger filters might suppress peaks or merge nearby peaks, while smaller filters might not provide enough smoothing, leaving too much variation untouched.

Trend Analysis

Some collected pressure signals show a linear trend. The Preprocessing Module must handle the trend by applying the linear trend removal technique described in Section 3.3.6 to the pressure data. Otherwise, it would complicate the pressure based stroke extraction. After that, the data occasionally show a curved progression, which we also refer to as a trend in the context of this work (see Figure B.10k). This kind of trend is nonlinear, which linear trend analysis techniques cannot handle. Therefore, this thesis presents a simple custom trend analysis method that can deal with this nonlinear progression. The Preprocessing Module applies it after the initial linear trend analysis, the outlier removal, and the noise reduction to fit the data to the x-axis by calculating stepwise linear functions:

1. Detect all lows in a signal by looking at the surrounding data points of each data point. The algorithm detects a low if the left and the right data points are larger than the currently considered data point p_i. Logically expressed:

$$p_{i-1} > p_i \wedge p_{i+1} > p_i \implies p_i \text{ is a low.} \tag{6.1}$$

2. Remove detected lows that are greater than the mean of the positive portion of the pressure data.
3. Compute a normalization line $f(x) = m * x + c$ for each pair of consecutive lows.
4. Evaluate the normalization line for each data point in the interval spanned by the corresponding lows and substitute this value from the original pressure value at this point in time.

Examples of this process can be found in Figures B.9m and B.10m. Step 2 is necessary to remove the influence of minor jumps close to the peak values of the pressure data as they could lead to erroneous normalization lines. This approach is certainly not perfect as it assumes that the data are perfectly smoothed and does not contain relevant lows above the mean of the positive portion of the data. Moreover, if a peak is lower than this threshold, jitter could result in the annihilation of a pressure peak (see Figure B.10m). Nevertheless, the approach works fine in the vast majority of strokes recorded during this work. This custom trend analysis is a necessary adjustment to enable the stroke extraction based on pressure data.

Save Preprocessed Data

The last step of the preprocessing workflow is to save the preprocessed data for further processing in .csv files. For inference, the Preprocessing Module generates only one file containing the fully preprocessed signals of all sensors. It creates two additional files for training purposes. The first one only contains the portion of the time series, which contains actual strokes, while the second one represents non-stroke noise actions at the beginning and end of a time series. The Stroke Model and the Stroke Future Model rely on the actual stroke data, the Noise Model learns from non-stroke noise data, and the Combined Model uses both data types.

Additional: Decision D — Base Position

A user labels each time series either as stroke data by selecting a stroke type prior to data collection or as base position data. Base position data requires less preprocessing than stroke data, because the Preprocessing Module considers them as non-stroke noise actions.

Additional: Non-Stroke Noise Calculation

The Preprocessing Module semi-automatically analyzes time series which do not contain base position data to determine when their first strokes happen (*PRE_NOISE_INDEX*) and when their last strokes end (*POST_NOISE_INDEX*). This analysis relies on the cumulative acceleration that we define as the sum of the prepared acceleration values in each direction x, y, and z at each point in time i minus the mean μ of this sum:

$$\text{summed\_acc}_i = xacc_i + yacc_i + zacc_i, \tag{6.2}$$

$$\text{cumulative\_acceleration}_i = \text{summed\_acc}_i - \mu(\text{summed\_acc}), \tag{6.3}$$

The Preprocessing Module automatically computes candidates for stroke start and end indices. The calculation is similar to the z-score calculation used for outlier detection.

1. The algorithm calculates two thresholds:

$$t_{upper} = 2 * \sigma\left(\text{cumulative\_acceleration}^{+}\right) \tag{6.4}$$

$$\text{and } t_{lower} = -2 * \sigma\left(\text{cumulative\_acceleration}^{-}\right), \tag{6.5}$$

   where σ describes the standard deviation of the supplied data. The $+$ indicates that t_{upper} only considers the positive values of the cumulative acceleration and the $-$ that t_{lower} only considers the negative values. This distinction allows a more accurate calculation of the non-stroke noise.

2. The algorithm initializes the *PRE_NOISE_INDEX* with 50 representing $50 * 10\,\text{ms} = 500\,\text{ms}$. The user usually interacts with the logging devices in the first half second of a time series. Hence, this initialization is feasible because this part does not contain relevant information about table tennis strokes. For the same reason, we initially set the *POST_NOISE_INDEX* to $n - 50$, where n denotes the length of the time series.

3. Starting at these initial non-stroke noise indices, the algorithm traverses the cumulative acceleration forward in case of the *PRE_NOISE_INDEX* and backward for the *POST_NOISE_INDEX*. In each step, it compares the traversed values to the previously calculated thresholds. If the current value is greater than t_{upper} or less than t_{lower}, it sets the *PRE_NOISE_INDEX* (resp. *POST_NOISE_INDEX*) to the current index and the respective traversal stops.

Finally, a human analyst inspects the automatically calculated indices and can further adjust these suggestions if necessary.

Additional: Non-Stroke Noise Removal

The penultimate step splits the fully preprocessed data into stroke data and non-stroke noise data based on the calculated *PRE_NOISE_INDEX* and *POST_NOISE_INDEX*. It then passes the fully preprocessed data, the stroke sequences, and the non-stroke noise parts to the final preprocessing step, which stores them in .csv files.

6.2 Machine Learning Models

As mentioned in Section 5.3.3, this work uses four machine learning models. Three of them are responsible for classifying input data, while the third one predicts the racket movement in the next time step 10 ms later. This section introduces early stopping and the sliding window approach that the Model Training Module uses to train these models (Sections 6.2.1 and 6.2.2). It then highlights the architectural characteristics of the classifiers and the forecaster (Sections 6.2.3 and 6.2.4).

6.2.1 Early Stopping

Early stopping is a simple regularization technique that prevents models from overfitting [39, pp. 141 f.]. It stops training early if the validation loss of the model has not improved for a given number of training epochs m [39, pp. 141 f.]. As a result, this technique can stop training before it reaches the specified maximum number of epochs. This work uses a threshold of $m = 10$ and a maximum number of epochs of 50. It considers a weight update as an improvement if the change of the validation loss is greater than 0.01. For inference, the models restore the network weights that produced the minimal validation loss during training.

6.2.2 Sliding Window

The Model Training Modules applies a sliding window approach with a stride of 1 data point = 10 ms to split the preprocessed data into chunks of 10 consecutive data points = 100 ms. This results in $n - 9$ windows per n-element time series. Each time step contains data from multiple sensors, such as accelerometer, gyroscope, and magnetic field. The machine learning models use these windows for training and inference. This approach allows models to learn from multiple time steps rather than solely relying on data from a single point in time. The knowledge about the stroke history embedded in a window improves the learning possibilities of the machine learning models because it allows them to put their inputs in a temporal context. Figure 6.4 illustrates a sliding window on the cumulative acceleration of an exemplary time series.

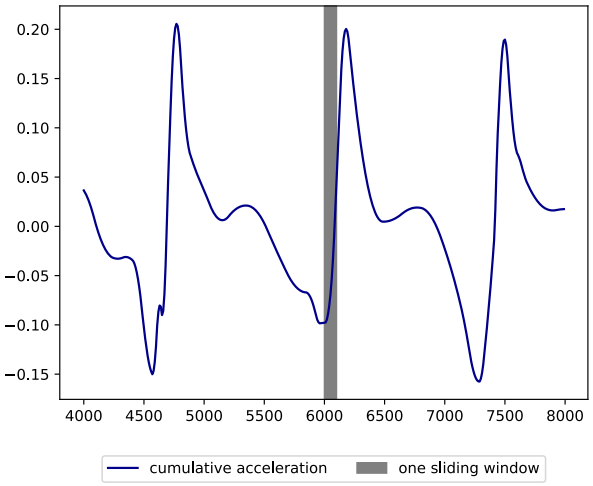


Figure 6.4: A sliding window of size 10 consecutive data points = 100 ms.

6.2.3 Classifier Architecture

All classifiers use the same underlying architecture (see Figure 6.5). The first layer is an LSTM layer consisting of fifty LSTM units. This layer takes a window of 10 data points as

input and computes one output per LSTM unit. Each data point consists of $X = 9$ input features representing the three three-dimensional signals of the accelerometer, gyroscope, and magnetometer. This input layer manages the temporal stroke context and merges the two-dimensional input matrix into a one-dimensional output vector. During training, this layer passes the resulting data to a Dropout layer. This layer type is commonly used in artificial neural networks to prevent them from overfitting by randomly setting 50 % of its inputs to zero in each training step. Inference skips this layer so that no values are lost. Afterward, two sequential feedforward layers (Dense) calculate the final output. The first one consists of fifty hidden units activated with the ReLU function. In case of the Stroke Model, the latter consists of eight output units representing the likelihoods of each stroke category. This layer calculates these probabilities using the softmax activation function. The Stroke and the Combined Model solely differ in their number of output nodes. The Combined Model includes an additional output node which represents the probability of non-stroke noise actions. In contrast, one output node activated by the sigmoid function is sufficient for the Noise Model's binary decisions. Please check the official Keras API [40] for more details about these layer types and their activations.

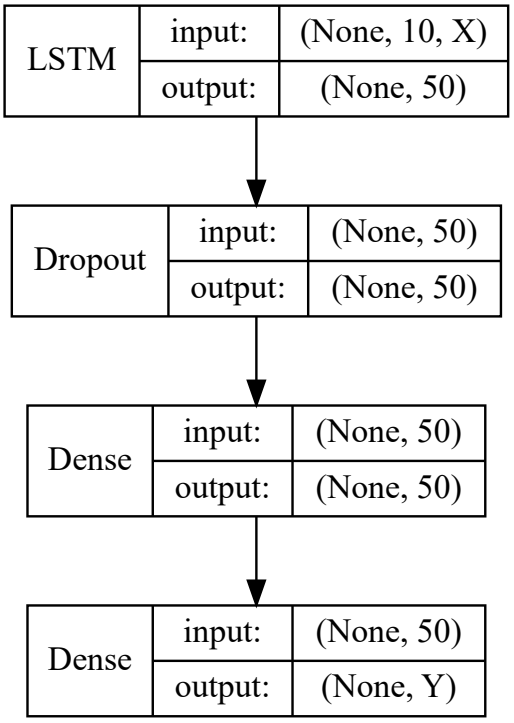


Figure 6.5: The classifier architecture. X: number of input features, Y: number of output classes.

6.2.4 Forecaster Architecture

The Stroke Forecaster takes a window as input and predicts the stroke motion in the next time step based on learned motion patterns. The system can use this information to analyze the player's stroke movements. Following the principle of simplicity, the architecture of its underlying Stroke Future Model is kept very light. It contains only a single LSTM layer with fifty LSTM units, the same amount the classifiers use in their recurrent parts. Figure 6.6 visualizes the architecture of the Stroke Future Model. For consistency, this network operates on the same combination of accelerometer, gyroscope, and magnetometer data as the classifiers. Therefore, the number of input features ($X = 9$) also stays the same. The forecaster does not classify input sequences. Instead, it acts as a regression

model and predicts future values of a time series. Consequently, it emits the same number of output features Y as there are input features X.

LSTM	input:	(None, 10, X)
	output:	(None, Y)

Figure 6.6: The architecture of the Stroke Forecaster. X: number of input features, Y: number of output features.

6.3 Stroke Extraction and Classification

One of the main tasks of the developed system is the detection of stroke intervals in preprocessed time series and their assignment of stroke types. The procedure consists of the three main steps stroke and non-stroke noise inference (Section 6.3.1), stroke extraction (Section 6.3.2), and stroke classification (Section 6.3.3).

6.3.1 Stroke and Non-Stroke Noise Inference

Before inferring a preprocessed time series, the Stroke Extraction & Classification Module must split it into windows of length 100 ms using sliding windows. Afterward, it feeds the machine learning models with these windows to calculate the start and the end point of a stroke sequence in a given time series and to assign each window a stroke type. The Noise Model estimates the probability Q_w that a window w contains a non-stroke noise action. The Stroke Model outputs the probability $P_w(t)$ per stroke type t that the window contains a specific stroke type. Then, the module compares the outputs of the two models to identify when a stroke sequence begins and when it ends. It considers a window to contain a stroke s if the probability of any stroke type in that window is greater than 1.5 times the probability of having a non-stroke action in the same window:

$$s \in w \Leftrightarrow \exists\, t : P_w(t) > 1.5 * Q_w. \tag{6.6}$$

The factor 1.5 increases the weighting of the output of the Noise Model. Therefore, a stroke can only be detected if the non-stroke noise probability is less than 66 percent. Without this factor, it would be possible to detect a stroke despite a high non-stroke noise probability. Figure 6.7 visualizes the factor's impact on the non-stroke noise detection. Based on Equation 6.6, the Stroke Extraction & Classification Module detects the start of a stroke sequence if ten consecutive windows contain strokes. The module applies the same rule backward to find the end of the stroke sequence.

Figure 6.8 shows the classification result of a stroke sequence containing eight backhand loops. The blue areas at the beginning and the end of the time series represent detected non-stroke noise actions. Please refer to Appendix E.1 for more examples.

The non-stroke noise detection is more straightforward if the Stroke Extraction & Classification Module uses the non-modular approach with the Combined Model instead of the combination of the Stroke and the Noise Model. In this case, there is no need to compare the outputs from two different models because the Combined Model classifies both stroke types and non-stroke noise actions. Hence, the Equation 6.6 is obsolete for the non-modular approach.

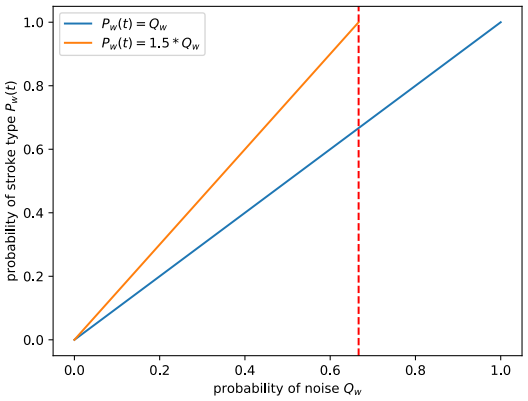


Figure 6.7: The impact of the noise factor 1.5 on the decision space. The area under each curve represents non-stroke noise actions, while the area above each curve represents valid strokes.

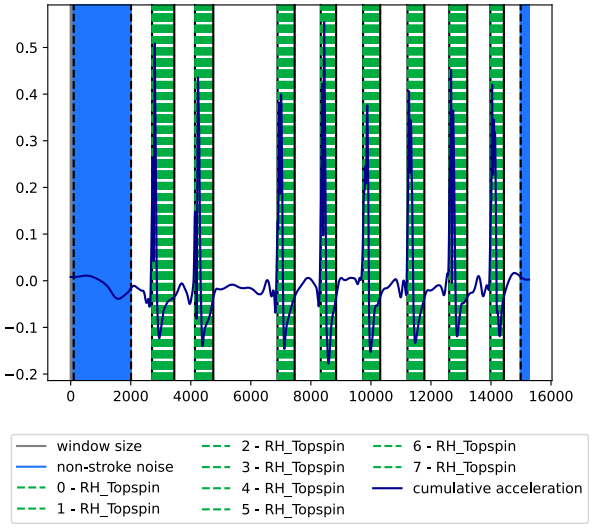


Figure 6.8: A classified time series containing eight backhand loops.

6.3.2 Stroke Extraction

The next step is the extraction of stroke intervals from a detected stroke sequence. Stroke extraction is not trivial, as each stroke type has its own peculiarities. To extract strokes, this work applies thresholding techniques to the three features cumulative acceleration (cf. Equation 6.3), its energy gradients, and the pressure data. Each step is described below.

Stroke Extraction (Cumulative Acceleration)

The Stroke Extraction & Classification Module must perform several steps to calculate stroke intervals based on the cumulative acceleration:

1. Divide the classified stroke sequence into ranges based on the most probable racket hand at each point in time. The Stroke Model (resp. the Combined Model) provides the probabilities. This preliminary step is important since investigations in this work have shown that forehand and backhand strokes are characterized by different high and low point sequences in the cumulative acceleration: A low point followed by

a high point denotes a forehand stroke. In contrast, the reverse order (high point before low point) indicates a backhand stroke.

2. Calculate the high and low points of the cumulative acceleration. The approach considers a data point as a high point if it exceeds the standard deviation of the positive part of the cumulative acceleration and as a low point if it is less than the standard deviation of the negative part of the cumulative acceleration. The thresholds correspond to the Equations 6.4 and 6.5, using the factor 1 instead of 2 for stroke extraction. This procedure results in a list of consecutive ranges labeled as highs or lows. Due to fluctuations in the time series, two highs may be close together without a low in between them. In cases where this distance is less than 100 ms, the approach assumes that both highs belong to the same stroke and combines them. The same applies to consecutive lows.
3. Create one set per hand (forehand, backhand) and categorize the highs and lows based on the hand ranges calculated in Step 1. If a high or a low overlaps with the ranges of two different hands, sort it into both sets.
4. In the forehand set, search for low-high combinations, since they denote forehand strokes, and in the backhand set, find high-low combinations because they denote backhand strokes. Label intervals that span such sequences as *FULL* stroke. Do not discard lows or highs, which are not part of a *FULL* stroke, since sometimes one of the peaks of a stroke is missing or does not exceed t_{upper} or t_{lower}. Instead, consider them as stroke candidates, i.e., *HALF* strokes.

Stroke Extraction (Energy Gradients)

The second approach to extract stroke intervals from stroke sequences relies on the gradients of the energy of the cumulative acceleration. We define the energy of the cumulative acceleration as the sum of its absolute squared values (see Equation 3.13). Strong increases in the energy correlate to significant changes in the acceleration, which indicate the execution of strokes. The easiest way to capture such energy changes to calculate its derivative and to compare the resulting gradients to the threshold

$$t_{\text{detection}} = \mu(\text{energy}') + \sigma(\text{energy}'), \tag{6.7}$$

where μ denotes the mean and σ the standard deviation. Intervals in which the gradients exceed this threshold form stroke candidate. Since the energy does not provide any information about the shape of a signal, it is not possible to distinguish with certainty whether these intervals represent strokes or random arm movements. Hence, the resulting intervals are not considered as *FULL* strokes and instead labeled as *HALF* strokes. It is natural for the energy gradients of fast movements that their peaks are of short duration. For this reason, the approach expands energy strokes that exceed $t_{\text{detection}}$ with a second, softer threshold function:

$$t_{\text{expand}} = \mu(\text{energy}'). \tag{6.8}$$

Stroke Extraction (Pressure)

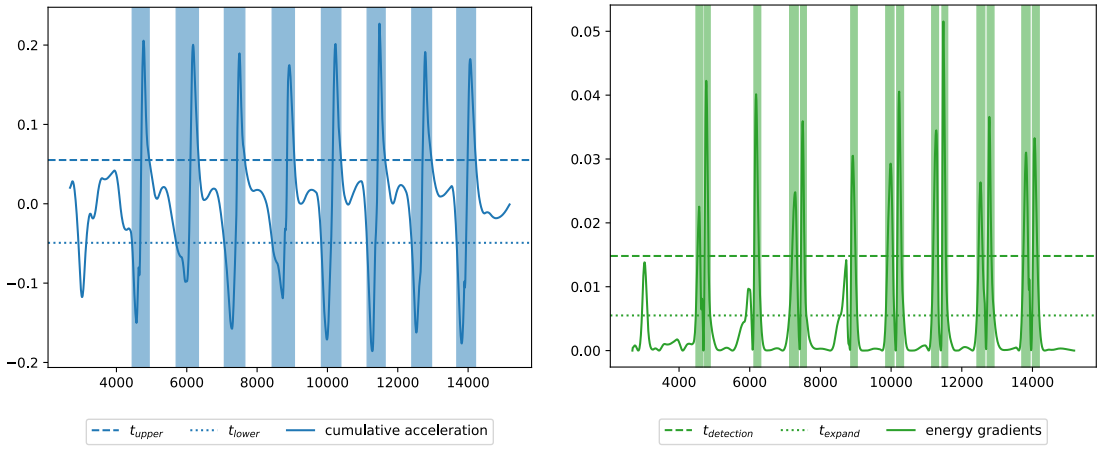
The third stroke extraction variant uses a threshold function to identify stroke intervals in pressure signals:

$$t = \mu(\text{pressure}) + \sigma(\text{pressure}), \tag{6.9}$$

where μ denotes the mean and σ the standard deviation. Pressure intervals that exceed this threshold form stroke candidates. Like the energy gradients, pressure data do not contain any information about the shape of the arm movement. Random arm movements or, even worse, environmental aspects such as changes in air pressure caused by opening a

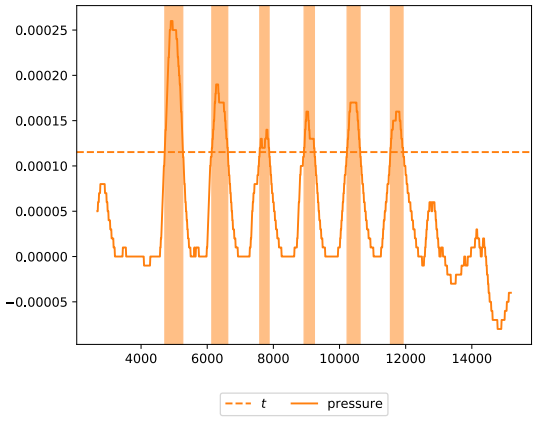
window may lead to pressure changes. Hence, this approach labels pressure stroke intervals as *HALF* strokes.

Figure 6.9 visualizes the results of each extraction step using a time series with eight forehand drives as an example.



(a) *FULL* acceleration strokes

(b) Gradient strokes



(c) Pressure strokes

Figure 6.9: Illustration of the three stroke extraction methods applied on a stroke sequence with eight forehand drives. For acceleration, it only visualizes *FULL* strokes. The pressure curve is not smooth because its values are truncated after five decimal places.

Final Stroke Intervals

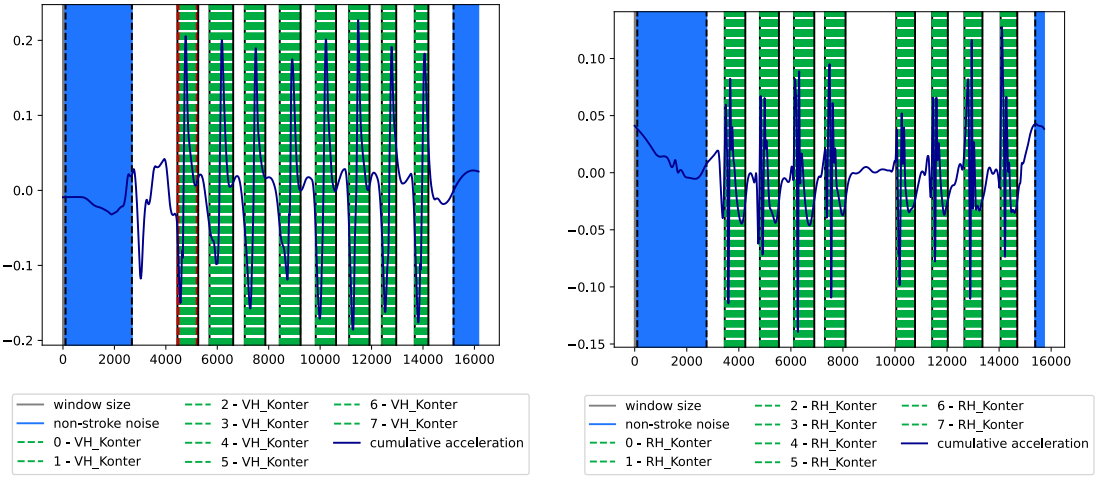
The last stroke extraction step aggregates the resulting stroke candidates into the final stroke intervals. In an ideal world with sinusoidal cumulative accelerations not containing any unexpected jumps and well-defined pressure data, it would probably be convenient to simply combine all calculated stroke candidates by extending *FULL* strokes with overlapping *HALF* strokes and consolidating overlapping *HALF* strokes combined *FULL* strokes. The resulting *FULL* strokes would then form the final stroke intervals. In reality, however, the signals are not shaped perfectly. Unexpected jumps between highs and lows can occur in the cumulative acceleration due to individual movements, sensor uncertainties, or the summation of the three acceleration dimensions. Figure 6.10 compares strokes with and without such fluctuations. For more examples, see Appendix E.1. In addition, the pressure data may show strange shapes due to occasionally occurring jumps in the data.

For instance, the time series visualized in Figure 6.2d contains two peaks instead of one at about 5500 ms. For this reason, this work uses a more complex approach that treats various edge cases:

1. Enrich pressure strokes with acceleration information in the following way:
   a) Intersect pressure stroke intervals with the *FULL* acceleration strokes, the *HALF* acceleration strokes, and the energy gradients strokes.
   b) Join intervals that form an intersection into larger intervals. Discard intervals that are not part of an intersection.
   c) Label intervals that were created by merging pressure intervals with *FULL* strokes as *FULL* strokes. Consider the others as *HALF* strokes.

   These enriched pressure intervals coupled with the *FULL* acceleration strokes form the set of stroke candidates.
2. Sort the stroke candidates by their start and end indices and combine overlapping intervals into larger intervals. Afterward, inspect the 200 ms neighborhoods of the stroke intervals: If the union of adjacent intervals contains at most one pressure stroke and one *FULL* acceleration stroke, then they are joined. Merged intervals are defined as *FULL* if they contain a *FULL* stroke, otherwise as *HALF*.
3. Remove the first detected stroke if it is a *HALF* stroke, since leading *HALF* strokes likely relate to movements associated with starting the IMU Logger or faulty reactions if the table tennis robot failed to serve the first ball. In addition, treat short, arbitrary movements by removing *HALF* strokes shorter than 100 ms.
4. Finally, inspect strokes that last longer than 1.5 seconds. If such a stroke interval contains more than one *FULL* acceleration stroke, split it into these *FULL* acceleration intervals. This is necessary because sometimes, Step 2 merges several *FULL* strokes due to overlapping pressure peaks.

Section 7.4 discusses the stroke extraction capabilities of this approach.



(a) Forehand drives without fluctuations

(b) Backhand drives with strong fluctuations

Figure 6.10: Stroke sequences containing eight forehand drives (left) compared to eight backhand drives (right). The cumulative acceleration of each backhand stroke shows strong fluctuations, while the forehand strokes are smooth.

6.3.3 Stroke Classification

Finally, the Stroke Extraction & Classification Module labels the stroke intervals. It first averages the Stroke Model's (resp. the Combined Model's) classification results P_t in a stroke interval with respect to the stroke type t. This results in a vector $\overline{P_t}$. After that, it consults the Stroke Sequence State Machine (cf. Figure 6.11) to enrich the averaged stroke type probabilities with domain knowledge.

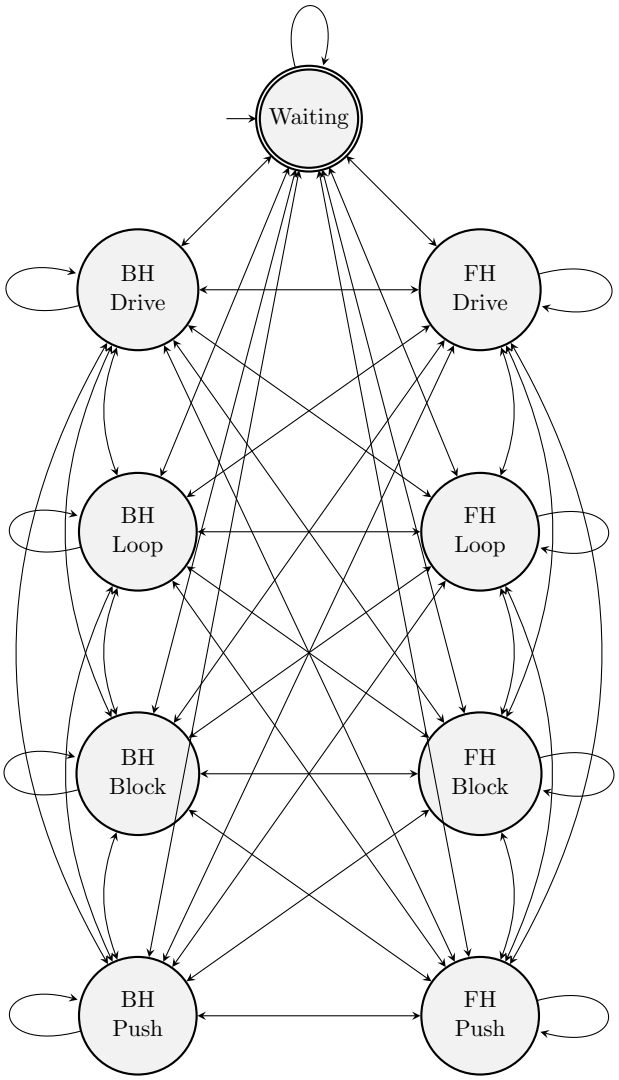


Figure 6.11: The fully connected Stroke Sequence State Machine used to enrich the LSTM stroke classifications with domain knowledge from typical table tennis drill descriptions (see [18]).

Each of the states represents one of the eight considered stroke types. If the player has not yet executed a stroke, the state machine resides in the so-called *Waiting* state, where all following strokes (states) are equally probable. After a stroke sequence has finished, the state machine returns to the *Waiting* state. Each state transition denotes the probability Q_t that one stroke type follows another in a typical game of table tennis (domain knowledge). More than 100 table tennis drill descriptions from [18], which are close to real game situations, form the ground truth of this automata. Table D.1 lists its transition probabilities. It is important to improve the state machine per player for two reasons: On the one hand, the given drill descriptions tend to favor stroke sequences in which the same stroke types follow each other. On the other hand, different players have different playing

styles. Hence, the probabilities of stroke sequences likely differ between two players. The system can individually improve this state machine over time by collecting stroke data of a player and updating its underlying stroke sequence database. The state machine keeps the stroke type executed immediately before the current stroke interval to be classified and returns the tanh of its transition probabilities Q_t. The tanh slightly increases the impact of stroke sequences with smaller transition probabilities compared to those with higher transition probabilities. This normalization is important for several reasons: The state machine relies on drill descriptions, in which the stroke types relevant to this work do not occur with equal frequency. In addition, the drill descriptions model typical stroke sequences in a controlled training environment. Since table tennis is a highly individual sport, an opponent's response during a rally is assessable but by no means predictable (e.g., a sudden change from forehand to backhand). The tanh counters these concerns by transforming the state machine probabilities into the interval $[0, 0.7632)$ and flattening the curve towards the end. Figure 6.12 visualizes the tanh-behavior.

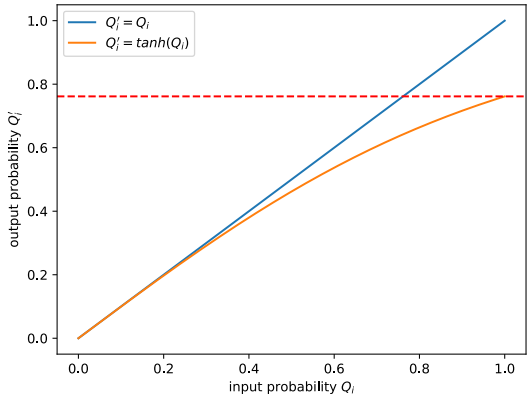


Figure 6.12: The impact of tanh on the state machine probabilities.

Finally, the Stroke Extraction & Classification Module multiplies the classifier vector $\overline{P}$ with the state transition vector Q element by element to determine the final probabilities per stroke type:

$$t = \text{index of max}\left(\overline{P} \odot \tanh(Q)\right), \tag{6.10}$$

where t represents the index of the stroke type with the highest probability and $\odot$ denotes the element-wise vector multiplication. The domain knowledge based state machine introduces a self-verification and self-correction mechanism to the otherwise machine learning based classification process. It allows the readjustment of the outputs of the Stroke Model (resp. the Combined Model) in case of unrealistic stroke sequences. Section 7.5 discusses the classification results on the test.

6.4 Stroke Analysis

The classification of strokes is not the only way to postprocess extracted stroke intervals. In addition, the developed table tennis stroke extraction and classification system provides hints on acceleration, velocity, angular velocity, and racket angle deviations in extracted stroke intervals. For this purpose, the Stroke Analysis Module compares these values between actually played strokes and the predictions of the Stroke Forecaster. The module can directly compare the actual cumulative acceleration a with its forecasted values by calculating their average deviation in a stroke interval:

$$\overline{\Delta a} = \frac{1}{n} \sum_{m=1}^{n} \Delta a_m, \tag{6.11}$$

with

$$\Delta a_m = \begin{cases} a_m^{actual} - a_m^{forecasted} & , a_m^{forecasted} \geq 0 \\ a_m^{forecasted} - a_m^{actual} & , a_m^{forecasted} < 0 \end{cases}. \tag{6.12}$$

Δa_m denotes the difference between the forecasted and the actual value at a given point in time. The reason for its partial definition is simple: For a positive forecasted value, the actual acceleration is higher if its value is greater than the forecasted acceleration, and for a negative forecasted value, the actual acceleration is higher if its value is less than the predicted acceleration. Figure 6.13 visualizes this behavior with two sinusoidal curves.

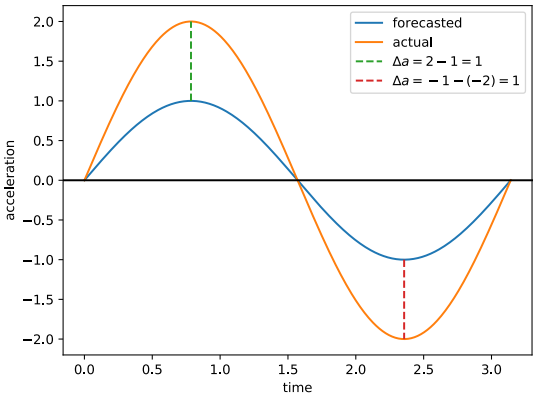


Figure 6.13: Visualization of the delta calculation (see Equation 6.12). The delta between both curves is positive for both positive forecasted values and negative ones.

With the same considerations, we define the average deviation of the cumulative angular velocity ω as follows:

$$\overline{\Delta\omega} = \frac{1}{n} \sum_{m=1}^{n} \Delta\omega_m, \tag{6.13}$$

where $\Delta\omega$ follows the same definition as Δa (see Equation 6.12). The cumulative angular velocity follows the definition of the cumulative acceleration (see Equation 6.3).

On the other hand, the Stroke Analysis Module must first calculate a stroke's velocity and racket angle to compute its deltas. It estimates the integral over the deltas between the actual and forecasted acceleration in a given stroke interval with the trapezoidal rule to calculate the average deviation of the velocity v [19, p. 98]:

$$\overline{\Delta v} = \frac{1}{n-1} \Delta t \sum_{m=1}^{n-1} \frac{\Delta a_m + \Delta a_{m+1}}{2} \tag{6.14}$$

where $\Delta t = 10\,\text{ms}$ denotes the temporal distance between two data points. The Stroke Analysis Module uses the same approach to compute the average deviation of the racket angle φ between the absolute and forecasted angular velocity:

$$\overline{\Delta\varphi} = \frac{180°}{\pi} \frac{1}{n-1} \Delta t \sum_{m=1}^{n-1} \frac{\Delta\omega_m + \Delta\omega_{m+1}}{2}, \tag{6.15}$$

where the initial factor converts radians into degrees.

Human coaches or virtual trainers could use the calculated average deltas in acceleration, angular velocity, speed, and racket angle to analyze a player's form or why a shot failed. For example, a decrease in speed or acceleration over time could be an indication of exhaustion.

6.5 Influence of Domain Knowledge on Processing

The following sections discuss the influence of domain knowledge on design decisions at different processing stages. Figure 6.14 gives an overview of the tasks which rely on domain knowledge.

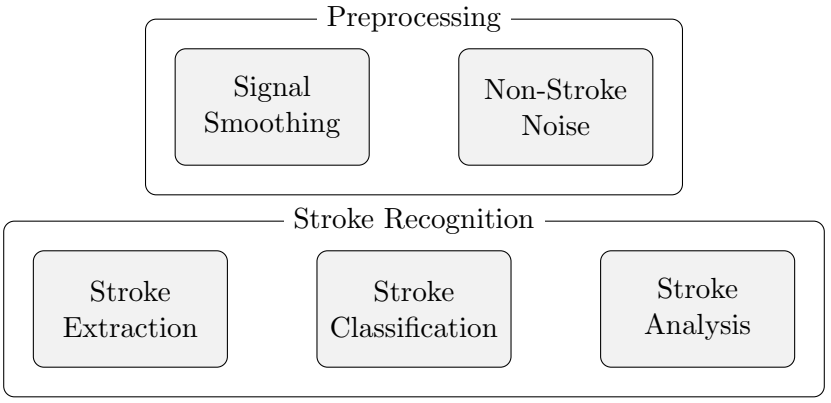


Figure 6.14: Domain knowledge infused tasks in preprocessing and stroke recognition.

6.5.1 Preprocessing

Preprocessing raw sensor data is an essential step in many machine learning pipelines. Selecting appropriate preprocessing techniques is crucial for the success of a system. On the one hand, unfavorably chosen methods can weaken or even eliminate information-bearing signal parts, while on the other hand, they can amplify irrelevant aspects. To minimize these risks, the preprocessing techniques used in this work were selected based on two leading questions: Which signal properties are important for table tennis? Which parts of a signal contain strokes? Knowledge about the signal shape in the domain table tennis is helpful to answer these questions.

Most table tennis experts do not have such knowledge unless they happen to have captured and analyzed table tennis data from the same modalities before. Nevertheless, their fundamental knowledge on stroke phases and executions can help system developers and data analysts isolate important signal features that should not be smoothed away by preprocessing.

Furthermore, the expertise of table tennis experts is beneficial if the system developers decide to apply wavelet transforms to reduce jitter in the input data, as unsuitable mother wavelets could distort the signal shape. Moreover, this type of domain knowledge is needed to distinguish whether strokes or non-stroke noise actions are present in the data under consideration.

6.5.2 Stroke Recognition

In addition to data acquisition and preprocessing, domain knowledge also plays an important role when selecting proper stroke recognition techniques. In this work, stroke extraction is optimized based on table tennis-specific knowledge about the signal shape and the information provided by the different sensors. The analysis of the cumulative acceleration revealed that highs following lows characterize forehand strokes, while backhand strokes follow the reverse order. Unfortunately, these characteristic peaks sometimes vanish. For this reason, stroke extraction also takes energy gradients and pressure data into account. However, since pressure does not contain any information about the stroke motion itself, it must be enriched with acceleration data.

Furthermore, game knowledge gained from typical table tennis drill descriptions can increase the stroke classification reliability and trustworthiness. The Stroke Sequence State Machine fulfills this verification task by comparing the machine learning based stroke classifications with typical stroke sequences and adjusts them if necessary. In doing so, the state machine explicitly introduces profound expert knowledge into the classification system to make its outputs more reliable in the context of table tennis. The output verification also alerts developers to potential misconfigurations in the machine learning models or incorrectly labeled training data in case of significant discrepancies between machine learning based predictions and domain knowledge based state machine outputs. Thus, game knowledge can help improve the classification capabilities of the system and assist developers in designing a robust system and interpreting its intermediate and final results.

Lastly, domain knowledge is essential for the selection of meaningful and understandable metrics for stroke analysis. The basic table tennis knowledge includes that stroke speed and racket angle are crucial for the success of a stroke. For this reason, metrics based on the accelerometer and gyroscope data are suitable to help players (and coaches) to analyze and improve their playing style by reflecting faulty strokes.

6.6 Summary

The table tennis stroke recognition system uses various signal processing techniques to amplify characteristic signal features. We leveraged table tennis related domain knowledge to decide which properties are relevant and select appropriate preprocessing steps. Stroke extraction uses two data dimensions: acceleration and pressure. Acceleration naturally provides important details on stroke movements. By adding pressure information, the system can also account for height differences of the racket during the stroke motion. Stroke classification relies on machine learning based classifiers and valid gameplay knowledge represented by a finite state machine. This domain knowledge allows the system to self-verify and self-correct its outputs. Stroke analysis provides insights into the acceleration, angular velocity, speed, and racket angle of extracted strokes, enabling players to analyze and perfect their stroke techniques. We chose these values based on the domain knowledge that speed and racket angle have a significant impact on the success of a stroke.

The next chapter analyzes and evaluates various aspects of the presented table tennis stroke recognition system based on data of two long-time table tennis players.

Chapter 7

Evaluation

This chapter evaluates the stroke extraction, classification, and analysis capabilities of the developed system using data from semi-professional players. Its first part describes the data collection process and the collected data (Section 7.1). Afterward, it presents the Stroke Model, the Noise Model, the Combined Model, and the Stroke Future Model, including important information on architectural decisions, training, and mobile device optimization (Section 7.2). Subsequently, it discusses the need for player-dependent machine learning models (Section 7.3). The following sections analyze each system part separately using goal-question-metric (GQM) plans (Sections 7.4–7.6). The penultimate section briefly analyzes and discusses the system qualities on data from some casual players (Section 7.7). This chapter ends with a short summary of the evaluation findings (Section 7.8).

7.1 Data

Data collection with the IMU Logger is an essential part of the developed system. This section describes the collection of training and test data for the design and evaluation of the stroke recognition system. Furthermore, it presents the data sets derived from the collected data.

7.1.1 Data Collection Process

Our initial plan was to collect an even number of data from multiple players of the local table tennis club and merge them into one common data set, which we would then use to train comprehensive baseline models. Unfortunately, we had to discard this plan due to the global pandemic (Covid-19) and its associated restrictions on public life. It was necessary to improvise since it was neither allowed to play sports in the local sports hall nor to meet in larger groups. We set up a temporary training facility in our own basement and collected data mainly from two semi-professional table tennis player with many years of experience. The first player's data serves as the basis for system design and development. The evaluation covers both players. The data collection process is as follows:

1. Set up the smart devices. Standalone: the player wears only a single smartwatch at the racket-holding wrist. Sensor network: the player wears multiple smartwatches (clients) distributed over its entire body and connects them to a smartphone (server).
2. Choose a stroke type and set up the table tennis robot accordingly.
3. Select adequate logging parameters for the stroke sequences to be performed and start the data collection process either on the smartphone or on a standalone smartwatch.
4. Wait until all sensors have started collecting data and then start the robot.
5. Stop the robot and the data acquisition process after the player played five to fifteen strokes. We targeted the recording of approximately eight strokes per data set but tolerated deviations since stroke sequences do not have fixed lengths in real game situations.

All in all, we collected 496 stroke sequences with a total of 3770 strokes from the two players. Player one performed the majority of these stroke sequences (436). Table 7.1 contains details on the stroke type distribution. Appendix A provides exemplary time series of the accelerometer, gyroscope, magnetometer, and pressure sensor for each stroke type. All of them contain characteristic peaks which indicate strokes. These peaks are clearer and more consistent for acceleration and angular velocity than for magnetic field and pressure. Depending on the stroke type, some directions of the magnetic field barely show such peaks (see Figure A.3c and A.7g). For the pressure, the data are sometimes a big mess (see Figure A.2d). Furthermore, some pressure peaks can merge (see Figure A.7h). The plots also show characteristic properties of each stroke type. For example, loops usually have higher accelerations than blocks or drives.

Table 7.1: Total number of recorded strokes per stroke type and data set (player one | player two).

Stroke Type	Training Set	Test Set	Validation Set	Total
FH Drive	302 | 35	72 | 16	64 | 8	438 | 59
FH Loop	278 | 39	64 | 16	64 | 8	406 | 63
FH Block	294 | 33	64 | 8	64 | 8	422 | 49
FH Push	254 | 33	64 | 16	56 | 8	374 | 57
BH Drive	285 | 33	64 | 16	64 | 8	413 | 57
BH Loop	292 | 40	64 | 16	64 | 8	420 | 64
BH Block	273 | 33	64 | 8	56 | 8	393 | 49
BH Push	311 | 35	72 | 16	64 | 8	447 | 59
Total	2289 | 281	528 | 112	496 | 64	3313 | 457

7.1.2 Data Sets

For each player, we divided the recorded data into three subsets using a 70/15/15 rule: the training set, the test set, and the validation set. This means that we added 70 percent of a player's data to his training set while distributing the remaining 30 percent evenly between his other two sets. The training set directly affects the weight updates during training [41]. The validation set assists the training process by inspecting the state of the model and tuning its hyperparameters after each training iteration [41]. The test set is isolated from the training process. The final model evaluation happens on these entirely unseen data [41].

Table 7.1 lists the total number of collected strokes per data set sorted by stroke types on a per-player basis. Note that the 15/15 part of the split rule is not based on the number

of strokes but on the number of time series per stroke type. That explains the slightly higher number of strokes in the test sets compared to the validation sets.

7.2 Machine Learning Models

This section evaluates the performance of the four machine learning models using a variety of performance metrics. It only considers data from the first player since this data serves as the basis for the development and design of the stroke extraction and classification techniques.

This section first introduces some performance metrics, such as the F1-score or the root mean squared error (Section 7.2.1). It then examines the discrimination capabilities of various Stroke Models with different input feature combinations (Section 7.2.2). This section also includes a comparison of alternative model architectures. The next section discusses the performance of the Stroke Future Model (Section 7.2.3). For simplicity, this model relies on the same input features as the classification models. This part of the evaluation concludes with a brief description of optimization techniques for mobile devices, as this thesis aims to use the models on mobile devices with limited resources (Section 7.2.4).

7.2.1 Performance Metrics

Many performance metrics exist for evaluating machine learning models. This thesis analyzes the classifiers using precision, recall, and the harmonic mean between these two called F1-score. These metrics can be derived from the confusion matrix shown in Table 7.2. It compares the predicted outputs with their actual labels.

Table 7.2: Confusion matrix (adapted from [39, p. 92]).
TP: true positives, FP: false positives, TN: true negatives, FN: false negatives.

		Predicted	
		Pos	Neg
Actual	Pos	TP	FN
	Neg	FP	TN

The precision [39, pp. 91 ff.]

$$p = \frac{TP}{TP + FP} \tag{7.1}$$

answers the question of how much of the predicted positive labels are actually positive. Only relying on this metric is not helpful since it only considers the samples predicted as positives. It reaches its limits when only a small fraction of the actual positive labels are recognized at all [39, pp. 91 ff.]. For this reason, we need a second metric. The recall [39, pp. 91 ff.]

$$r = \frac{TP}{TP + FN} \tag{7.2}$$

answers the question of how much of the actual positive labels the model predicted correctly [39, pp. 91 ff.]. Again, it is not useful to rely solely on this metric, as it does not perform well when the model predicts all samples as positives. The F1-score solves the disadvantages of both metrics and allows better comparisons between two different models. It is defined as the harmonic mean between the precision and the recall [39, pp. 91 ff.]:

$$F_1 = \frac{2 * r * p}{r + p}. \tag{7.3}$$

This thesis evaluates the forecaster using the root mean squared error [39, pp. 39 ff.]

$$RMSE(X) = \sqrt{\frac{1}{n} * \sum_{i=1}^{n} (\hat{y}(x_i) - y_i)^2} \tag{7.4}$$

and the mean absolute error [39, pp. 39 ff.]

$$MAE(X) = \frac{1}{n} * \sum_{i=1}^{n} |\hat{y}(x_i) - y_i| . \tag{7.5}$$

Both of them calculate deviations between the calculated output $\hat{Y}$ and the expected output Y and are therefore suitable for the evaluation of regression models [39, pp. 39 ff.]. According to Géron, the root mean squared error is the preferred metric as it weights larger errors more heavily [39, pp. 39 ff.].

7.2.2 Classifiers

The classifiers assign categories to each sliding window. The Stroke Model categorizes the data into eight stroke types, while the Noise Model decides whether a window contains a stroke or a non-stroke noise action. The Combined Model distinguishes between stroke types and non-stroke noise.

Training

We trained the classifiers on the first player's data using early stopping. During training, the categorical cross-entropy loss function computes the loss between the predicted and the actual class of a window. This loss function is a common choice for classification models which calculate class membership probabilities [39, p. 295]. The Stroke Classifier is trained on 319,535 randomly selected windows, validated on 69,039 windows, and tested on 74,554 windows. All these windows together represent the first player's stroke sequence data. The Noise Classifier, on the other hand, is trained on 496,450, validated on 99,351, and tested on 106,940 randomly selected windows. It additionally uses the non-stroke noise data to distinguish between strokes and non-stroke actions. The Combined Model is optimized and evaluated on the same training, validation, and test windows as the Noise Model. However, it additionally distinguishes between the individual stroke types.

Feature Selection

Each smartwatch collects a total of ten features per time step. Three features each for the accelerometer, the gyroscope, and the magnetometer, and one feature for the pressure sensor. Table 7.3 shows a comparison of ten Stroke Models based on different feature sets. The following paragraphs analyze the results with a focus on the losses and F1-scores on the test set of player one.

It is immediately apparent that the models relying solely on the data from one sensor perform significantly worse than the other models with larger feature combinations. The accelerometer, the gyroscope, and the magnetometer models suffer from relatively high losses around 30 percent and only achieve F1-score in the mid to high eighties. The pressure model performs even worse. These results make them four not suitable for stroke classification. Moreover, we tested three combinations, each consisting of two of the three sensors accelerometer, gyroscope, and magnetometer. We omitted pressure because its standalone results were insufficient. These three models perform significantly better than the single sensor solutions with F1-scores of up to 0.955 and losses in the interval $[0.127, 0.179]$. The top three models rely on more input features. One uses the entire feature space spanned

Table 7.3: Performance metrics of various Stroke Models trained on several input feature combinations of player one's data. L: loss, F1: F1-score.

X	Input Features	Params	Epochs	Training	Validation	Test
20	acc, gyr, mag, pres wrist + right thigh	17,158	15	L: 0.0435 F1: 0.9859	L: 0.1774 F1: 0.9594	L: 0.1280 F1: 0.9552
10	acc, gyr, mag, pres wrist	15,158	26	L: 0.0526 F1: 0.9815	L: 0.1277 F1: 0.9711	L: 0.0990 F1: 0.9664
9	acc, gyr, mag wrist	14,958	37	L: 0.0419 F1: 0.9848	L: 0.1360 F1: 0.9659	L: 0.0840 F1: 0.9727
6	acc, gyr wrist	14,358	30	L: 0.0755 F1: 0.9728	L: 0.1346 F1: 0.9563	L: 0.1277 F1: 0.9544
6	acc, mag wrist	14,358	30	L: 0.1179 F1: 0.9579	L: 0.3245 F1: 0.9074	L: 0.1789 F1: 0.9388
6	gyr, mag wrist	14,358	23	L: 0.0962 F1: 0.9648	L: 0.2114 F1: 0.9440	L: 0.1604 F1: 0.9453
3	acc wrist	13,758	36	L: 0.2473 F1: 0.9135	L: 0.2762 F1: 0.9064	L: 0.3011 F1: 0.8920
3	gyr wrist	13,758	32	L: 0.2489 F1: 0.9177	L: 0.2816 F1: 0.9050	L: 0.3059 F1: 0.8931
3	mag wrist	13,758	43	L: 0.3384 F1: 0.8687	L: 0.6158 F1: 0.8552	L: 0.3561 F1: 0.8592
1	pressure wrist	13,358	18	L: 2.0036 F1: 0.0042	L: 2.0101 F1: 0.0013	L: 2.0054 F1: 0.0040

by the two locations wrist and right thigh, while the second one uses all features collected from the wrist. Both of them achieve an F1-score higher than 95 percent. The best performing model uses the acceleration, angular velocity, and magnetic field recorded at the racket-holding wrist. It achieves an F1-score of more than 97 percent and a loss of 0.084. Compared to the usable models with fewer features, its number of parameters increases by only about 4.2 percentage points, resulting in only a minor additional computational effort for inference. The main findings from this analysis are:

1. The classifier that operates on the accelerometer, the gyroscope, and the magnetometer data are the best performing one.
2. Pressure data are not particularly meaningful, neither solely nor in combination with other sensor types.
3. The use of additional data taken from the right thigh does not really help the classification. The good results are rather attributable to the influence of the wrist data.

Based on these findings, the Stroke Classifier uses the combination of acceleration, angular velocity, and magnetic field data (each in x-, y-, and z-direction) for stroke classification. The Noise Classifier, the Combined Classifier, and the Stroke Forecaster leverage the same nine features for consistency.

Models and Alternatives

This work considers bidirectional LSTMs as alternative architectures for the Stroke and the Noise Model. In short, bidirectional LSTMs consist of two separate LSTM units with a shared output layer [42]. One of them considers the input data in a forward path, while the other one looks at the reverse input data [42]. This modification enables classifiers

to look at a bigger picture of the context but doubles the number of learning parameters in the recurrent part of their neural network architectures. Nevertheless, bidirectional LSTMs achieved significant performance gains in sequence processing tasks such as natural language processing [42]. For this reason, we wanted to check whether bidirectional LSTMs can also improve motion detection tasks. For this purpose, we trained and evaluated bidirectional versions of the Stroke and the Noise Model using the same training, validation, and test sets as for the unidirectional approaches. Table 7.4 lists the resulting performance metrics. The unidirectional and bidirectional models perform quite similarly. The deviations are within the margin of error. Hence, the bidirectional versions do not offer an improvement over the unidirectional models. Especially not in the context of mobile device usage, where the associated doubling of parameters can harm inference speed and battery life.

Table 7.4: Comparison of alternative models using player one's accelerometer, gyroscope, and magnetometer data. L: loss, F1: F1-score.

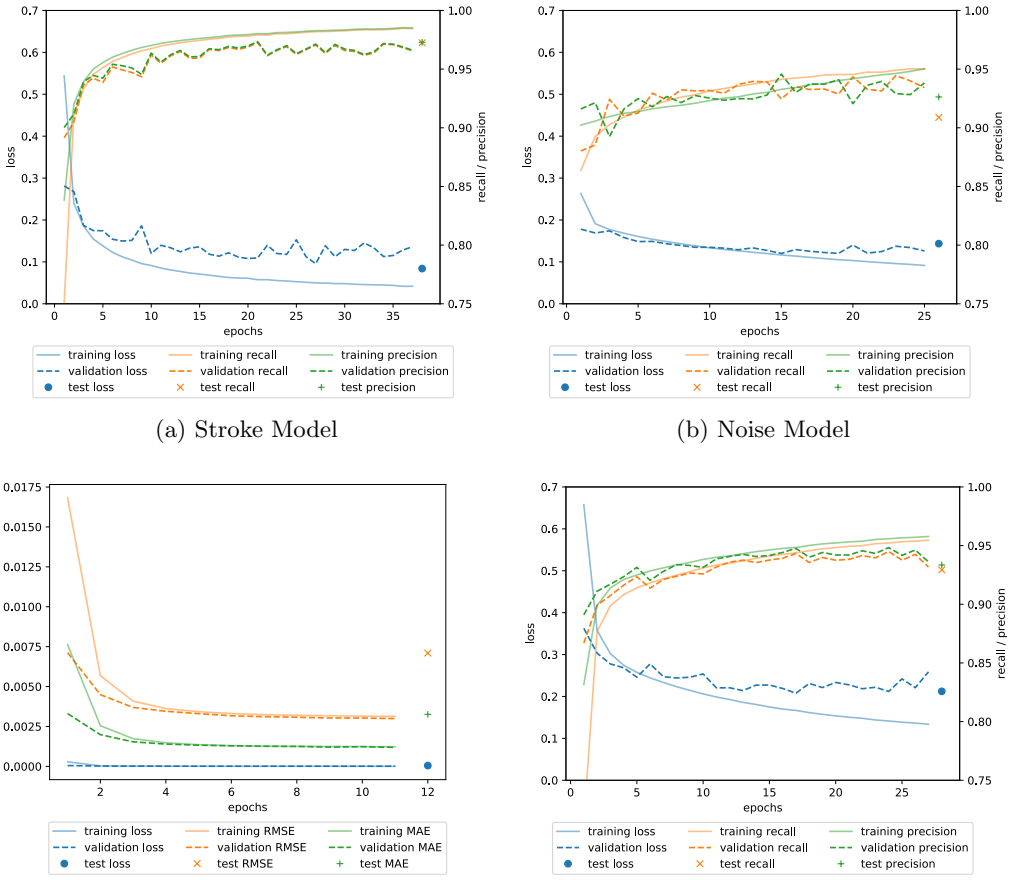
Model	Y	Params	Epochs	Training	Validation	Test
Stroke Model	8	14,958	37	L: 0.0419 F1: 0.9848	L: 0.1360 F1: 0.9659	L: 0.0840 F1: 0.9727
Stroke Model Bidirectional	8	29,458	22	L: 0.0539 F1: 0.9812	L: 0.1343 F1: 0.9646	L: 0.0811 F1: 0.9693
Noise Model	1	14,601	25	L: 0.0915 F1: 0.9502	L: 0.1260 F1: 0.9362	L: 0.1435 F1: 0.9175
Noise Model Bidirectional	1	29,101	20	L: 0.1027 F1: 0.9428	L: 0.1287 F1: 0.9309	L: 0.1424 F1: 0.9164
Combined Model	9	15,009	27	L: 0.1337 F1: 0.9562	L: 0.2588 F1: 0.9339	L: 0.2122 F1: 0.9314

Lastly, we tested the Combined Model which combines the responsibilities of the Stroke and the Noise Model. It achieves an F1-score slightly higher than the noise F1-score on the test set but more than four percentage points lower than the Stroke Model. Furthermore, the Combined Model's loss on the test set is significantly larger than the other models' losses. This could be due to short waiting times between strokes, which the Combined Model recognizes as non-stroke noise actions even though the semi-automatic non-stroke noise calculation process of the preprocessing labeled these parts with stroke types. Figure 6.8 shows an exemplary time series, where a short waiting time occurs between the second and the third stroke.

The main advantage of the non-modular, combined approach over two separate classifiers is its low parameter count and the associated energy efficiency. In absolute numbers, it requires 14,550 fewer parameters and still achieves decent results on the test set. The number of floating point operations (FLOPS) required for inference also reflect this trend. Although the modular approach using the Stroke and the Noise Model requires approximately twice as many FLOPS as the Combined Model (cf. Table 7.6), it also comes with some non-negligible advantages. For example, modularity eases the identification of reasons for inconsistent classifications and the maintenance of the models. The evaluation considers both the modular and non-modular approaches but neglects the bidirectional variants because they suffer from high weight counts and offer no particular advantages.

Figure 7.1 illustrates the training of the Stroke, the Noise, and the Combined Model. Early stopping terminates training after validation loss has not changed significantly for ten consecutive epochs. The graphs also plot the final test results for reference. Figure 7.2 shows the confusion matrices of the Stroke and the Combined Model on the test set. In

general, both models classify the input windows with high probabilities. As a result of the addition of the non-stroke noise class to the Combined Model, this model tends to be less accurate with slightly lower classification results per stroke type than the Stroke Model. In both cases, the backhand drive and block achieve the worst results with probabilities of true predictions in the low nineties for the Stroke Model or high eighties for the Combined Model. As a result of the similarity of these stroke types, the models often confuse them with probabilities between four and eight percent.



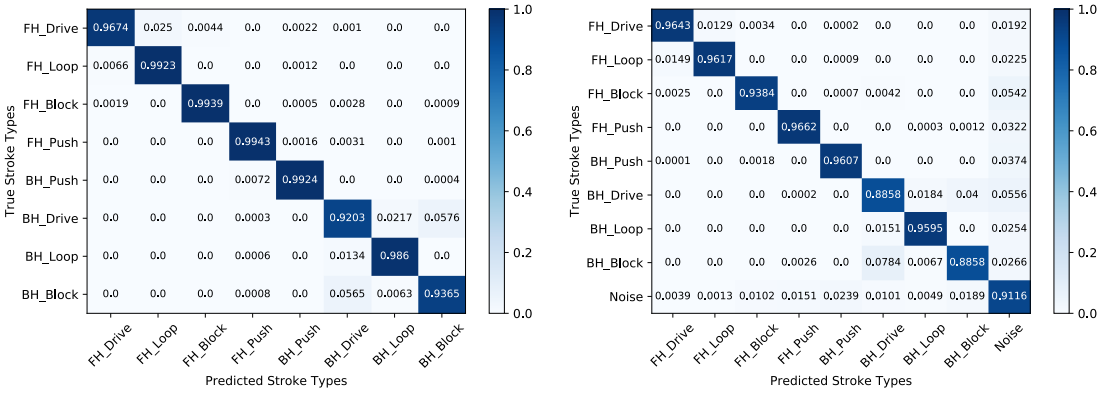
(a) Stroke Model

(b) Noise Model

(c) Stroke Future Model

(d) Combined Model

Figure 7.1: Training of player one's Stroke, Noise, Stroke Future, and Combined Model.



(a) Stroke Model

(b) Combined Model

Figure 7.2: Confusion matrices of player one's Stroke and Combined Model on his test set.

7.2.3 Forecaster

The Stroke Future Model is trained on a total of 319,534 windows, validated on 69,038 windows, and tested on 74,553 windows representing stroke data only. Again, this model solely relies on player one's data. Note that exactly one window per set is missing compared to the Stroke Classifier, even though the training of both models relies on the same stroke data. The reason for this is that the classifiers assign class labels to each window, while the forecaster always predicts the next time step. Therefore, the last window of a time series cannot be mapped to a subsequent time step because it simply does not exist. Thus, the number of windows per set is reduced by one. During training, the forecaster uses the mean squared error as loss function to validate the current model's performance. The performance of the Stroke Future Model is measured based on the root mean squared error and the mean absolute error. Table 7.5 lists the training results. Both metrics show impressive numbers on the test set with values smaller than one percent. Therefore, it seemed appropriate not to compare the model with any other architectures. Figure 7.1c visualizes the training progress.

Table 7.5: The training results of the Stroke Future Model. L: loss, RMSE: root mean squared error, MAE: mean absolute error.

Params	Epochs	Training	Validation	Test
684	11	L: 9.803e-06 RMSE: 0.0031 MAE: 0.0012	L: 8.963e-06 RMSE: 0.0030 MAE: 0.0012	L: 5.036e-05 RMSE: 0.0071 MAE: 0.0033

7.2.4 Mobile Device Optimization

Inference can either run on a computer or on a smartwatch. We converted the Keras [37] models into TensorFlow Lite [43] models to enable mobile device inference. We also applied float16 quantization to further reduce the model size. This optimization step shrinks the network weights from 32 bits to 16 bits. According to [44], performance remains nearly identical with this optimization compared to the original Keras models. A comparison of the performance metrics of both model representations confirms this statement. Table 7.6 compares the model sizes before and after the downsizing process and lists the performance metrics of the final models. The size of the classifiers shrinks by approximately 80 percent when comparing their Keras models with their quantized TensorFlow Lite models. The forecaster achieves a size reduction of 37.5 %.

Table 7.6: Influence of model shrinking on model size, including the floating point operations needed for inference (FLOPS).

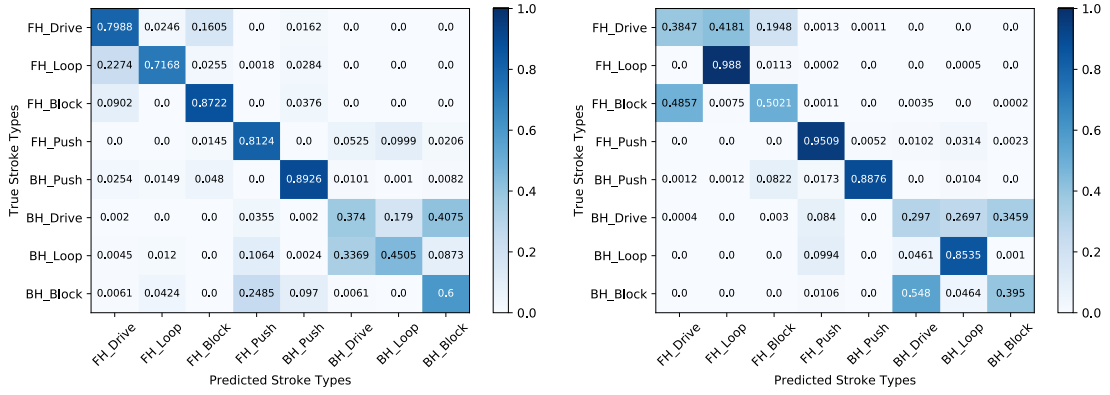
Model	Keras	TF Lite	TFLite + float16 quantization	FLOPS	Test Results
Stroke Model	211 KB	69.1 KB	41.6 KB	49,413	F1: 0.9727
Noise Model	207 KB	67.7 KB	40.9 KB	48,710	F1: 0.9175
Stroke Future Model	31.3 KB	12.0 KB	12.0 KB	1,950	RMSE: 0.0071 MAE: 0.0033
Combined Model	211 KB	69.3 KB	41.7 KB	49,513	F1: 0.9314

At this point, it is worth mentioning again that the modular approach consisting of separate Stroke and Noise Models requires a significantly larger number of floating point operations than the non-modular approach. Nevertheless, this separation of concerns makes it easier to understand the modular approach. From the energy perspective, which is very

important in the mobile sector, the non-modular approach with the Combined Model is the better choice since it drastically decreases the number of required floating point operations. Another advantage of the lower FLOPS is its associated decreased inference time end users will certainly appreciate.

7.3 Player-Dependent Models

We also collected data from a second semi-professional table tennis player with long years of experience to evaluate the system's performance. This data set contains much fewer strokes than the one from the first player (cf. Table 7.1). Since both players have a similar playing strength, we decided to investigate how the models of the first player behave on the data of the second player. Figure 7.3a visualizes the results using a confusion matrix. As we can see, player one's Stroke Model classifies player two's forehand strokes and backhand pushes with decent probabilities but struggles extremely with the remaining backhand strokes. These findings prove that table tennis is a highly individual sport, where even players with similar skill levels can have different stroke techniques.



(a) Player one's Stroke Model, player two's data (b) Player one's data, player two's Stroke Model

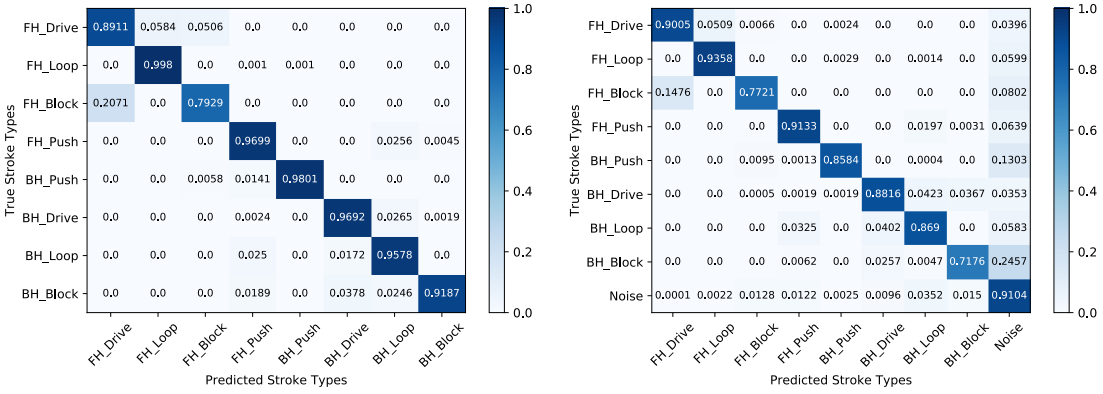
Figure 7.3: Comparison of the confusion matrices when mixing Stroke Models and test data from two different players.

With this in mind, we decided to train separate models for player two for further evaluation of the table tennis stroke recognition system. Table 7.7 lists their performance metrics. A comparison of his Stroke and Noise Models with his Combined Model reveals similar trends as the player one's models: Again, the Stroke Model achieves the best performance on the test set, while the Noise and the Combined Models show lower F1-scores. Like the Combined Model of player one, the Combined Model of player two generates the highest loss. The training results of player two are overall slightly worse than those of player one. We can explain these observations with player two's significantly smaller amount of training data.

Figure 7.4 shows the confusion matrices of the Stroke and the Combined Models of player two. They indicate that the second player executes forehand drives and forehand blocks similarly with misclassification rates ranging from five to twenty percent. In addition, the Combined Model reveals that there are relatively high similarities between strokes and non-stroke noise actions. This is especially the case for backhand blocks (25 %) and backhand pushes (13 %). Possible reasons for these similarities are the small amount of data and the non-stroke noise calculation of the preprocessing pipeline, which only treats non-stroke noise actions at the beginning and at the end of a time series but not between strokes.

Table 7.7: Training results of player two's machine learning models.

Model	Epochs	Training	Validation	Test
Stroke Model	22	L: 0.0202 F1: 0.9938	L: 0.4966 F1: 0.9238	L: 0.2429 F1: 0.9430
Noise Model	18	L: 0.0642 F1: 0.9694	L: 0.3115 F1: 0.8841	L: 0.2557 F1: 0.8860
Stroke Future Model	11	L: 2.683e-05 RMSE: 0.0052 MAE: 0.0024	L: 2.051e-05 RMSE: 0.0045 MAE: 0.0022	L: 7.648e-05 RMSE: 0.0087 MAE: 0.0045
Combined Model	15	L: 0.1432 F1: 0.9481	L: 0.4191 F1: 0.8869	L: 0.3348 F1: 0.8837



(a) Stroke Model (b) Combined Model

Figure 7.4: Confusion matrices of player two's Stroke and Combined Model on his test set.

Out of interest, we inferred the first player's test data with the second player's Stroke Model. Figure 7.3b shows the resulting confusion matrix. While the model can distinguish loops and pushes surprisingly well, it fails miserably with blocks and drives. These results emphasize the individuality of table tennis even further.

7.4 Stroke Extraction

The first part of this evaluation inspects the stroke extraction capabilities of the system using the following GQM plan:

G1: Detection of strokes in time series

Q1: How effective is the developed stroke extraction approach at extracting strokes?

M1: Mean Squared Error (MSE), False Extraction Rate (FER)

The mean squared error corresponds to the root mean squared error without the square root (see Equation 7.4). In the context of stroke extraction, we calculate the MSE between the number of extracted strokes and the actual number of strokes in a time series. Furthermore, we define the false extraction rate (FER) as:

$$FER = \frac{Missing + Additional}{Strokes}, \tag{7.6}$$

where $Missing$ is the number of falsely not extracted strokes, $Additional$ is the number of falsely extracted strokes, and $Strokes$ is the actual number of strokes.

To answer Q1, we executed the system twice on each player's test data. Once with the modular classification approach using the Stroke and the Noise Model, and another time with the non-modular approach (Combined Model). In both cases, the system extracted three additional strokes for player one and missed zero. This results in a FER of less than one percent and an MSE of less than 5 percent. For player two, on the other hand, the system neither detected additional strokes on the randomly chosen test set nor missed any strokes. All in all, only three errors occurred in a total of 640 strokes from both players. Table 7.8 contains detailed evaluation results.

Table 7.8: The stroke extraction results on the test sets of player one (P1) and player two (P2).

Model	Strokes	Missing	Additional	FER	MSE
Stroke Model	P1: 528	P1: 0	P1: 3	P1: 0.0057	P1: 0.0455
+ Noise Model	P2: 112	P2: 0	P2: 0	P2: 0.0	P2: 0.0
Combined Model	P1: 528	P1: 0	P1: 3	P1: 0.0057	P1: 0.0455
	P2: 112	P2: 0	P2: 0	P2: 0.0	P2: 0.0

Stroke extraction relies on thresholding techniques that use the mean and standard deviation of a stroke sequence. It heavily depends on the results of the non-stroke noise detection since non-stroke noise parts impact the mean and standard deviation of a time series. Moreover, stroke classification affects stroke extraction, as the latter relies on the observation that different high and low point sequences represent forehand and backhand strokes. For this reason, it is fascinating that the stroke extraction results of the modular and the non-modular approach are identical. Table 7.9 lists the average number of windows classified as non-stroke noise actions per player and approach. Despite the different noise detection mechanisms, the results are pretty similar.

Table 7.9: The average number of windows containing non-stroke noise actions on the test sets of player one (P1) and player two (P2). Pre Noise refers to windows before and Post Noise to windows after a stroke sequence.

Model	Non-Stroke Noise	Pre Noise	Post Noise
Stroke Model	P1: 203.61	P1: 339.18	P1: 68.05
+ Noise Model	P2: 242.79	P2: 316.57	P2: 169.00
Combined Model	P1: 214.91	P1: 350.18	P1: 79.64
	P2: 231.89	P2: 287.14	P2: 176.64

These results show that the developed stroke extraction approach can effectively detect strokes in recorded time series with truncated non-stroke noise actions. Both the engineering-friendly modular approach and the energy-efficient non-modular approach achieve good scores. Appendix E.1 provides exemplary outputs of the modular and non-modular approach for each stroke type.

7.5 Stroke Classification

The table tennis stroke recognition system uses multiple classifiers to categorize extracted strokes into eight stroke types. Furthermore, it leverages a state machine to enrich the classifier results with domain knowledge about typical table tennis stroke sequences. The second part of this evaluation explores the system's classification capabilities using the following GQM plan:

G2: Classification of extracted strokes

Q2: Can domain knowledge improve the classification?

M2: False Classification Rate (FCR)

Here, we define the false classification rate (FCR) as:

$$FCR = \frac{Misclassified}{ExtractedStrokes}, \tag{7.7}$$

where $Misclassfied$ is the number of falsely classified strokes, and $ExtractedStrokes$ is the number of strokes extracted by the stroke extraction mechanism. This formula uses the number of extracted strokes instead of the actual number of strokes because the classifiers can only assign stroke types to extracted stroke intervals. They do not know if strokes are falsely extracted or missing.

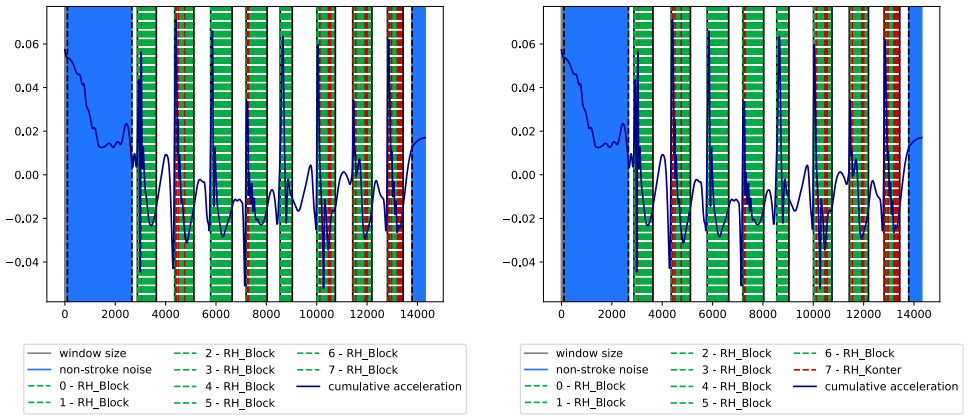
To answer Q2, we executed the system four times on each player's test data. Two times using the modular approach (Stroke and Noise Model) and two times using the non-modular approach (Combined Model). In both cases, once with and once without the Stroke Sequence State Machine.

The modular approach misclassified only one of the extracted strokes from player one when we enriched the classifier's outputs with domain knowledge. In contrast, the system misclassified two stroke intervals when it relied solely on the machine learning based classifier. The false classification rates also reflect these results: the FCR without domain knowledge (0.0038) is twice as large as the FCR with domain knowledge (0.0019). Thus, adding the Stroke Sequence State Machine resulted in a small classification improvement. Nevertheless, the false classification rates are minimal in both cases. Table 7.10 lists the complete classification results for both players. For player two, there are no incorrect stroke classifications for both classification approaches. Neither with nor without domain knowledge.

Table 7.10: The stroke classification results on the test sets of player one (P1) and player two (P2).

Model	Stroke Sequence State Machine	Extracted Strokes	Misclassified	FCR
Stroke Model + Noise Model	Yes	P1: 531 P2: 112	P1: 1 P2: 0	P1: 0.0019 P2: 0.0
	No	P1: 531 P2: 112	P1: 2 P2: 0	P1: 0.0038 P2: 0.0
Combined Model	Yes	P1: 531 P2: 112	P1: 2 P2: 0	P1: 0.0038 P2: 0.0
	No	P1: 531 P2: 112	P1: 2 P2: 0	P1: 0.0038 P2: 0.0

Figure 7.5 compares the classification in the faulty case. Domain knowledge infusion could not cure the second misclassification because the probability of the incorrect stroke type was simply too high for the Stroke Sequence State Machine to handle (cf. Figure 7.6). To make matters worse, the system incorrectly extracted this stroke interval even though it does not contain a stroke but an intermediate movement. Fortunately, this misclassification did not propagate, and the subsequent stroke was classified correctly. The non-modular approach achieved similar results for player one: The misclassifications occur in the same time series (cf. Figure 7.7). However, the addition of domain knowledge could not eliminate them as the classifier probability is simply too high.



(a) Classification with domain knowledge (b) Classification without domain knowledge

Figure 7.5: Comparison of the classification of eight backhand blocks using the modular approach with and without the state machine. Green: correct classifications, red: misclassifications.

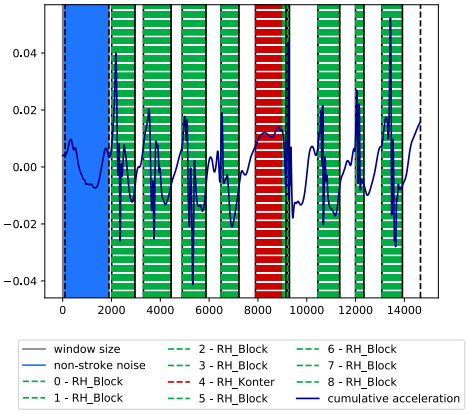
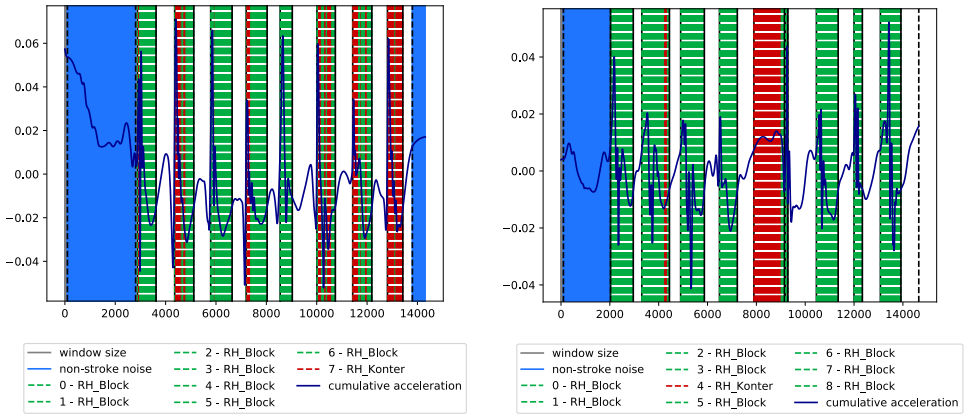


Figure 7.6: An incorrect stroke extraction causes the second misclassification when using the modular approach. The Stroke Sequence State Machine could not cure it.



(a) Eight backhand blocks (b) Eight backhand blocks

Figure 7.7: Faulty stroke classifications using the non-modular approach. The Stroke Sequence State Machine could not solve either misclassification.

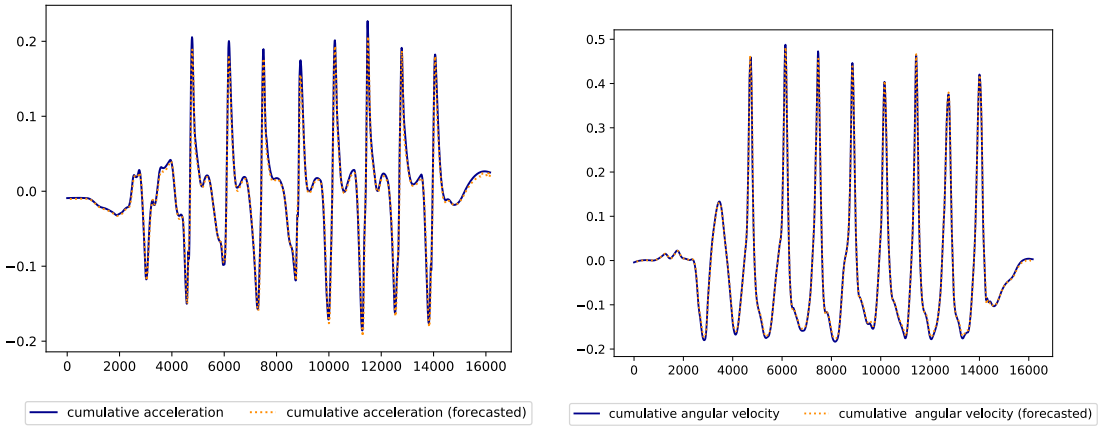
These results show that the developed approach is well suited for classifying extracted stroke intervals. However, to answer Q2, we must not consider the FCRs exclusively but must also analyze the domain knowledge infusion technique itself. The output of the Stroke Sequence State Machine depends on the last stroke classification. On the one hand, this reduces the likelihood of an incorrect stroke classification following a correct classification (cf. Figure 7.5). On the other hand, the state machine could conversely hinder the classification process of an interval following a misclassification. This could especially be a pitfall if the Stroke Model (resp. the Combined Model) is unsure and computes similar probabilities for multiple stroke types.

To answer Q2, this kind of domain knowledge infusion has the potential to improve the classification of strokes (cf. Figure 7.5). Nevertheless, this approach has some weaknesses and should be further optimized to make it more robust. See Appendix E.1 for more exemplary extraction and classification results of the modular and non-modular approach for each stroke type.

7.6 Stroke Analysis

The last aspect of the table tennis stroke extraction and classification system is the analysis of extracted strokes. Since the quality of the analysis results depends mainly on the individual subjective perception, it cannot be quantified. For this reason, we do not provide a GQM plan for evaluation. Instead, we present some facts about the results of the stroke analysis on the test data of the two players.

All in all, there are no significant deviations between performed strokes and their corresponding predictions for both players (see Figure 7.8 as an example). In both cases, the acceleration and the velocity are slightly higher on average than the predictions made by the forecaster. Hence, the players executed the strokes slightly faster than expected. The angular velocity and the racket angle, on the other hand, are slightly lower on average than the forecasted values. These numbers indicate that the racket angles of both players are slightly smaller on the test sets compared to the learned patterns.



(a) Acceleration (b) Angular velocity

Figure 7.8: The cumulative acceleration and angular velocity of a sequence containing eight forehand drives compared to their forecasted values.

There is a tendency for the differences to be larger for player two than for player one. Hence, player two probably played the strokes with less consistency than player one. Accordingly, player two presumably played the strokes with less consistency than player one. However, it is also possible that the higher differences are related to the smaller amount

of training data of player two and thus somewhat poorer generalizability of the Stroke Future Model. Table 7.11 lists the averaged stroke analysis results on the test data of both players. Appendix E.2 contains some detailed examples.

Table 7.11: The average deviation in acceleration, angular velocity, velocity, and angle on the test set strokes performed by player one (P1) and player two (P2).

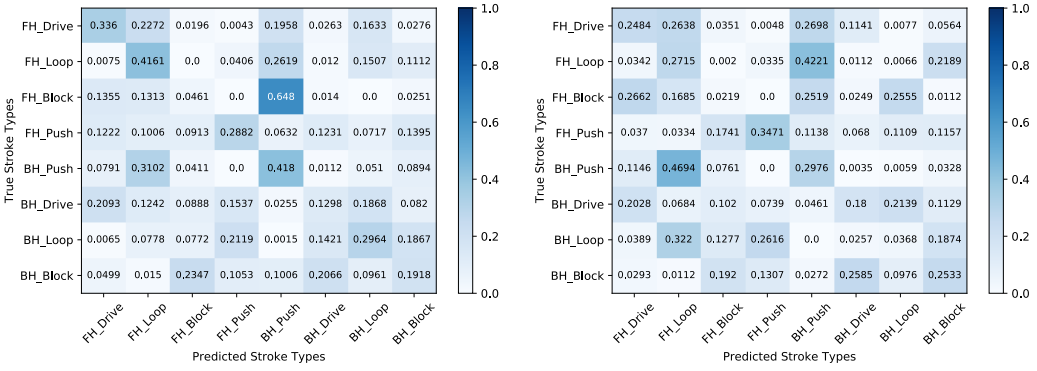
	Acceleration	Angular Velocity	Velocity	Angle
P1	0.0452 %	−0.0282 %	0.0013 m s^{-1}	−0.0248°
P2	0.4652 %	−0.0247 %	0.0050 m s^{-1}	−0.0402°

7.7 Excursus: Amateur Data

The intention behind the table tennis stroke recognition system is that semi-professional players or players above a certain skill level can use it to monitor their strokes and analyze their play. Players must at least know the differences between the considered stroke types and be able to execute them.

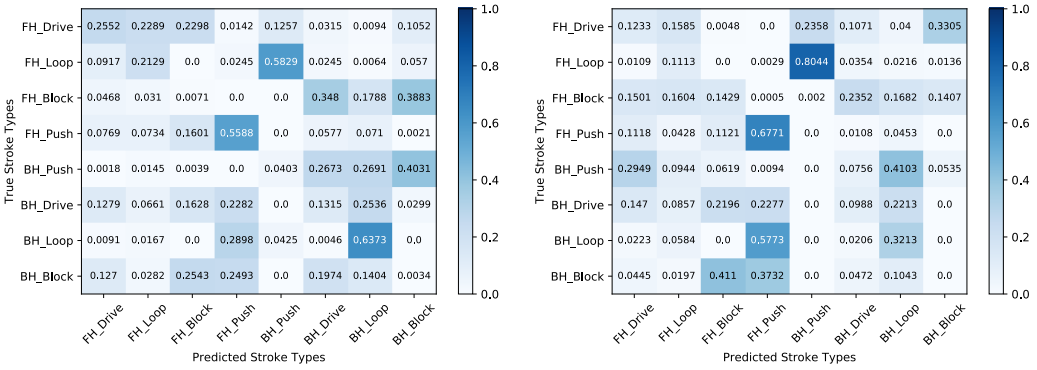
For interest, we tested the system on data of two amateurs who have not played table tennis in years and have never trained with a table tennis robot. Although they used to play table tennis, they needed a brief explanation of the eight considered stroke types. Consequently, their executions were sometimes difficult to distinguish, even to the trained eye. This was especially the case with amateur one.

All in all, we collected $3 \times 8 = 24$ strokes per stroke type from both amateurs and used them to test the performance of the Stroke Models of the two semi-professional players. Figure 7.9 presents the resulting confusion matrices. For both amateurs, the discrepancies between model predictions and performed strokes are tremendous. The stroke type classifications are mostly random. These results show that the models and the system are not suitable for hobby players. A distinction between forehand and backhand or offensive and defensive strokes would probably be sufficient for such casual players.



(a) Amateur one's data, player one's Stroke Model (b) Amateur one's data, player two's Stroke Model

Figure 7.9: Inference of amateur data with pretrained Stroke Models.



(c) Amateur two's data, player one's Stroke Model (d) Amateur two's data, player two's Stroke Model

Figure 7.9 (cont.): Inference of amateur data with pretrained Stroke Models.

7.8 Summary

We evaluated the table tennis stroke recognition system with data from two semi-professional players. Although both have similar playing strengths, their stroke executions have shown to be dissimilar. This finding confirms the statement from Section 4.2 that table tennis is a highly individual sport where each player has his own playing style.

Another interesting finding is that the Stroke Model trained on acceleration, angular velocity, and magnetic field from the racket-holding wrist achieved the best performance on the test set compared to feature combinations with fewer data sources. Furthermore, neither pressure nor data from the right thigh improved model performance. The use of bidirectional LSTMs instead of unidirectional LSTMs brought no improvements either.

The evaluation of the system's stroke extraction and classification mechanisms shows promising results with only three faulty stroke extractions and only two misclassifications. The addition of game knowledge through the Stroke Sequence State Machine further reduced the number of misclassifications.

The following chapter discusses some findings of this work and alternative embodiments. Moreover, it presents ways to improve the system in the future.

Chapter 8

Discussion and Outlook

This chapter discusses essential findings on mobile device usage, sensor placement, and the individuality of table tennis (Sections 8.1–8.3). Afterward, it gives an outlook on possible extensions and optimizations of the developed table tennis stroke extraction and classification system (Section 8.4). This chapter ends with a brief summary of these considerations (Section 8.5).

8.1 Edge Devices and the Machine Learning Models

Stroke recognition systems must be practical and portable for everyday use. Edge devices such as smartwatches or fitness trackers are particularly suitable for this purpose as many people wear them anyway. According to [45], more than two million smartwatches were sold annually in Germany between 2018 and 2020 with a clear upward trend. Unlike proprietary solutions, using traditional, widely available edge devices eliminates the need to carry, charge, and maintain additional devices.

This work considers two different classification approaches. The modular approach uses two machine learning models, namely the Stroke and the Noise Model, to distinguish between the eight considered stroke types and non-stroke noise actions. The Stroke Extraction & Classification Module merges the outputs of the two models to separate a time series into stroke and non-stroke parts. The non-modular approach, on the other hand, relies on one single model, which removes the need for the comparison step.

Although the non-modular approach shows the disadvantage of slightly poorer classification performance on the test sets, it has some significant advantages, especially when using edge devices for inference. It requires nearly half the FLOPS for inference and nearly half the storage space compared to the two separate models. Furthermore, there is no need to perform a merging step to compute the final output, which further reduces the computational requirements of this approach. These properties make the non-modular approach more attractive to edge devices with their typical resource and energy constraints. As edge devices become more and more powerful, they could be used in the future to individually retrain and optimize the machine learning models on a per-player basis. Since table tennis

is a highly individual sport with many players having their unique playing style, this personalization would probably improve the individual classification results. Also, this would probably help to determine and analyze a player's fitness level with its daily fluctuations.

8.2 Sensor Placement

Choosing appropriate and meaningful sensor locations is important in many human activity recognition applications. Domain knowledge about body parts involved in typical movements can assist the selection of appropriate sensor placements. In this work, we tested two sensor locations: the racket-holding wrist and the right thigh. While the wrist provides meaningful data, the data collected at the right thigh carry rather little information in the controlled environment. The racket-holding wrist provides important features about (fore-) arm movements. However, the wrist is not rigid during table tennis strokes, as wrist movements are essential for creating more spin or acceleration. Sensors attached to the racket handle or the racket-holding hand can detect such wrist movements and, therefore, likely provide more precise data. This additional information on movement characteristics would presumably positively affect the discrimination of individual stroke types.

Image 8.1 shows a provisional placement of an Arduino Nano 33 BLE Sense on a racket handle. This board includes all necessary sensors and can collect more fine-grained data than wrist-worn smartwatches due to its placement, but has some non-negligible drawbacks: power supply and weight. Because the board does not contain a battery, it requires an external power source, e.g., a power bank carried in a pocket. Even if this board contained a battery or an embedded solution existed, players would need to charge their rackets from time to time, making the system less attractive to end users. Also, adding sensors to a racket changes its weight and balancing, affecting the player's playing style. Players might not accept such solutions. These considerations show that this approach is not suitable for real-world use and only sufficient for testing purposes.

Furthermore, we could also explore totally different sensor placements. For example, we could place sensors on the upper body to measure table tennis-specific rotational movements that are particularly important during loops to generate higher accelerations. Likewise, we could place sensors on the forehead to analyze the player's field of view. While all of these rankings could provide some intriguing insights about the player's strokes and physique, they have the disadvantage wearing multiple devices on different parts of the body is rather inconvenient and not user-friendly. Furthermore, the usage of multiple sensors requires a sensor network for data collection and a central unit responsible for preparation and processing. Hence, the user must maintain and carry more devices at unusual body parts, making this variant less friendly to the average table tennis player.

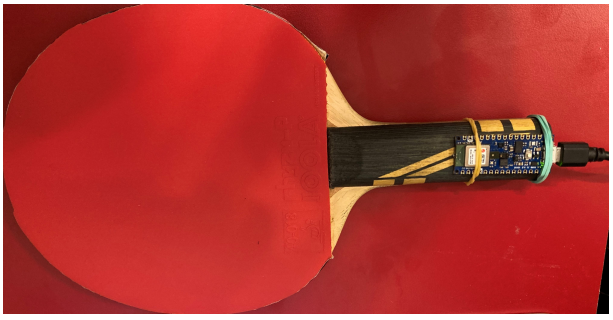
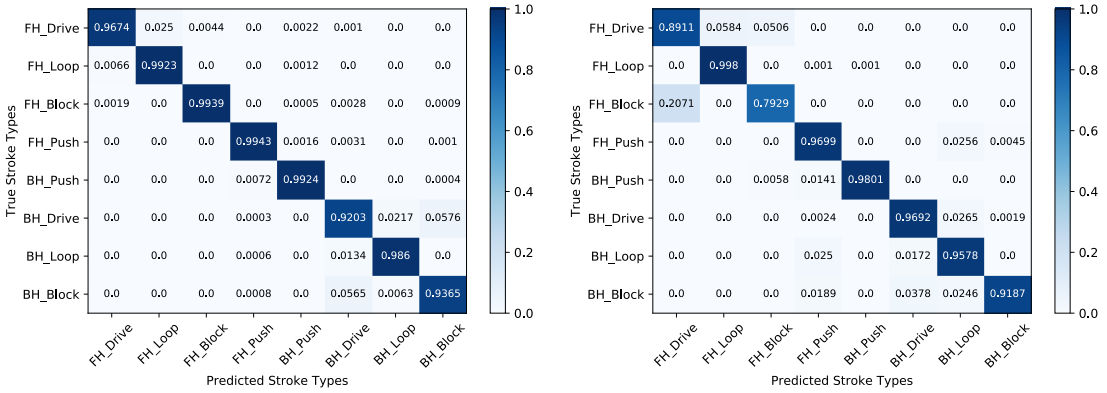


Figure 8.1: Alternative sensor placement (Arduino Nano 33 BLE Sense) directly on the racket handle.

8.3 Individuality of Table Tennis

As stated in Section 4.2, table tennis is a highly individual sport. Besides interindividual differences, which manifest themselves in different arm movement speeds or racket angles that are attributable to different playing strengths, playing styles, or preferences, intraindividual deviations can also occur. For instance, the subtle nuances in stroke movements can differ depending on a player's daily form, current fitness level and fatigue, or longer training breaks (e.g., due to Covid-19). Furthermore, a player's stroke execution can change and evolve over time. Such deviations in the stroke movements can lead to differences in detected sensor signals. Therefore, it is essential to continuously improve the machine learning models based on new data to capture such evolving stroke executions.

Moreover, the execution of some stroke types can be similar. Boundaries between stroke types are not always obvious and sometimes blurred. For instance, the backhand drive and backhand block examples shown in Figures C.15 and C.18 seem pretty similar. While the depicted strokes differ in their movement lengths, their racket angles are quite similar. For this reason, different stroke types are sometimes hard to distinguish. This behavior is also evident in the confusion matrices of the individually trained Stroke Models of the two players (see Figure 8.2): For player one, there is about 5 % overlap between backhand blocks and backhand drives. For player two, the classifier labels forehand blocks as forehand drives in more than 20 % of the test windows. Furthermore, it misclassifies slightly more than 5 % of player two's forehand drives as forehand loops and another 5 % of the test windows as forehand blocks.



(a) Player one (b) Player two

Figure 8.2: The confusion matrices of the Stroke Models of both players on their respective test sets. In both cases, it is most difficult for the models to distinguish drives and blocks.

8.4 Future Work

This section presents some ideas for improving the developed table tennis stroke extraction and classification system. In addition, it makes some general suggestions for extending the functional scope and considers alternative approaches.

8.4.1 Improvements

In general, the system performs well on the collected data sets. However, sometimes problems occur with the pressure based stroke extraction as a result of the custom trend analysis technique. This method relies on the assumption that the linear filter smooths

pressure data perfectly. This assumption does not always hold in reality and is the reason for occasionally disappearing peaks after preprocessing. Hence, we must further optimize the custom trend analysis by refining the data smoothing or improving the technique itself.

The developed machine learning models show good performance on the test sets with F1-scores in the higher nineties for the classification models and an RMSE of less than one percent for the regression model. While early stopping and dropout layers are used during training to reduce the likelihood of overfitting, there is still a great need to fine-tune the hyperparameters of the networks to improve stroke and non-stroke noise classification and stroke analysis. However, optimization should not be aimed exclusively at improving the general performance of the models as players use the system on low-power, resource-limited devices. Therefore, the number of network weights also plays an essential role and must not be too large.

In addition to the machine learning models, the explicit infusion of domain knowledge using the Stroke Sequence State Machine is also important for the system's classification approach. We could consider long-term dependencies to improve its reliability and reduce the impact of single misclassifications on subsequent classifications. The currently used drill descriptions can serve as the basis for a prototype since most of them contain rallies of more than two strokes. Because table tennis is a highly individual sport with different players having different playing styles, individual adjustments of the state transition probabilities are important to make more accurate statements about the specific player. This optimization can be done based on the collected player data. Furthermore, the machine learning based classification and the Stroke Sequence State Machine are currently in a one-to-one relationship. Even if the classifiers have high confidence, this relationship potentially leads to misclassifications in free game situations due to imbalances in the ground truth of the state machine. To overcome such issues, the output of the state machine could be incrementally weighted using the following rule: The lower the confidence of the machine learning approach, the higher the impact of the domain knowledge based state machine.

Currently, we trained and evaluated the system separately on data from two long-term table tennis players. Although this proves the principal correctness of the developed concept, it does not allow generalized statements about it. For this reason, it is crucial to collect data from numerous players and train comprehensive models that are hopefully able to generalize well. The data should include not only stroke data but also a wide variety of random movements to improve non-stroke noise detection as well.

8.4.2 Suggestions

Currently, stroke extraction and stroke classification form the core of the developed stroke recognition system. These two parts are mostly separated. While stroke extraction uses a more algorithmic approach, stroke classification relies on the outputs of the Stroke or the Combined Model and the Stroke Sequence State Machine. Nevertheless, the two components are interlocked in a certain way. On the one hand, stroke extraction uses the output of the classification models to distinguish whether forehand or backhand strokes are present in the data under consideration and to select the appropriate cumulative acceleration pattern (high-low for forehand, low-high for backhand). On the other hand, stroke classification uses the output of the classification models to label extracted stroke intervals. Therefore, it would be interesting to investigate whether a joint neural network can extract stroke intervals and discriminate between stroke types and non-stroke noise actions simultaneously. This idea requires an additional preprocessing step that extracts and labels individual stroke actions instead of the entire stroke sequence. This improved labeling technique could also improve the current classification approach, as waiting times between strokes would no longer be incorrectly labeled as strokes but as non-strokes actions.

Stroke recognition in real-time would be another exciting feature. Currently, stroke extraction uses thresholding techniques that consider each time series as a whole. This approach is not applicable in real-time scenarios because it executes all computations after the user has stopped data collection. Further research is needed to optimize it for real-time stroke extraction and classification.

8.5 Summary

The table tennis stroke recognition system runs on edge devices with severe resource constraints. This makes it important to keep the machine learning models small and their computational operations minimal. In addition, it is equally important for usability that the system is as minimally invasive as possible. This work's single-smartwatch solution fulfills this usability goal by requiring only one common device. Proprietary or multi-device solutions are somewhat counterintuitive as they increase the maintaining overhead for the user.

Playing styles and stroke techniques can differ from player to player. Moreover, a player's technique can evolve and change over time. Continuous training of the models based on individual player data is important to ensure the quality of the classifications and forecasts.

The remaining chapter highlights the main findings of this thesis and discusses their potential implications for future research in machine learning.

Chapter 9

Conclusion

This chapter marks the conclusion of this thesis. It gives an overview of the work, its intention, and its findings. Moreover, it answers the research question "Can machine learning benefit from domain knowledge to compute more trustworthy and reliable outputs?".

This work investigates the impact of domain knowledge on machine learning in signal processing applications in the context of table tennis stroke recognition. It presents a machine learning based system capable of extracting, classifying, and analyzing eight table tennis strokes within fused sensor signals from accelerometer, gyroscope, magnetometer, and pressure. The strokes comprise drive, loop, block, and push in forehand and backhand. The system can distinguish between non-stroke actions, such as random arm movements, and actual strokes. This feature allows the system to recognize stroke sequences within time series. The additional stroke analysis gives players indications of inaccurate stroke executions that may lead to stroke failures. These hints allow players to improve their stroke techniques accordingly. This monitoring and coaching ability sets the system apart from existing approaches that focus only on the classification of strokes. In the future, this ability can be further enhanced so that the system acts as a virtual coach who gives players textual suggestions on how to improve their playing style instead of simple numerical values.

The system can run either on wrist-worn smartwatches at the network edge or on computers. While other sensing solutions, such as sensors attached directly to a racket, would likely provide more accurate and fine-grained data, the smartwatch solution offers some non-negligible advantages over proprietary or embedded devices. First, the popularity of smartwatches is rapidly increasing in recent years. This wide distribution makes them the perfect candidate for such a system, as there are already plenty of smartwatch users who probably do not want to carry and maintain an additional, highly specialized device. Furthermore, a wrist-worn smartwatch does not change the weight and balance of the racket, which is crucial for the acceptance of such a system.

The presented approach infuses domain knowledge into the table tennis stroke recognition system implicitly during system design and explicitly during its execution to improve its stroke recognition capabilities. Implicit domain knowledge assists many planning and

processing stages, including the selection of proper sensors and their placements, signal processing, and stroke extraction techniques. Explicit domain knowledge, on the other hand, equips the system with self-verification and self-correction abilities. This kind of domain knowledge infusion leverages a state machine to evaluate machine learning outputs and to adjust them as necessary. This Stroke Sequence State Machine holds valid gameplay knowledge on typical table tennis stroke sequences.

We evaluated the system in a controlled environment with a ball-serving table tennis robot on stroke data from two semi-professional, long-time table tennis players. Since table tennis is a highly individual sport, we evaluated the system separately for both players. The system achieved promising results with only three false extractions and two misclassifications on the 640 test strokes when relying solely on the outputs of the machine learning models. After enriching the classification approach with game knowledge about typical table tennis stroke sequences, the number of misclassifications shrank to one. These results proof that the explicit incorporation of domain knowledge into the classification process can help to compute more trustworthy and reliable classifications.

This thesis aimed to explore the infusion of domain knowledge as a way to improve machine learning applications in signal processing using table tennis stroke recognition system as an exemplary use case. The evaluation of the developed system showed that domain knowledge has the potential to make machine learning based systems more reliable and trustworthy if the knowledge contains information about realistic state sequences. In addition, interdisciplinary knowledge helps in the selection of meaningful and user-friendly sensor placements. These considerations suggest that both domain knowledge implicitly consulted during system design and explicit domain knowledge provided during system execution can significantly impact the performance of a system. Therefore, domain knowledge should be considered more frequently when developing machine learning based applications. Further research is needed to find more general solutions for incorporating domain knowledge into machine learning applications. An example of this is the adaptation of the Stroke Sequence State Machine to other domains where several step sequences with different probabilities are possible. Other exciting research topics are, for example, the direct infusion of domain knowledge into the learning process of machine learning algorithms or the preparation of domain knowledge in a suitable way to enable automatic processing and maintenance.

Bibliography

[1] A. Holzinger, "Explainable AI (ex-AI)," *Informatik-Spektrum*, vol. 41, no. 2, pp. 138–143, Apr 2018.

[2] S. Pandya, "Understanding The Connection: Human Activity Recognition, Safety And Productivity," Jan 2021, accessed: 2021-04-01. [Online]. Available: https://www.forbes.com/sites/forbestechcouncil/2021/01/27/understanding-the-connection-human-activity-recognition-safety-and-productivity/

[3] S. R. Islam, W. Eberle, S. K. Ghafoor, A. Siraj, and M. Rogers, "Domain Knowledge Aided Explainable Artificial Intelligence for Intrusion Detection and Response," 2020.

[4] S. Radovanović, B. Delibašić, M. Jovanović, M. Vukićević, and M. Suknović, "Framework for integration of domain knowledge into logistic regression," in *Proceedings of the 8th International Conference on Web Intelligence, Mining and Semantics*, ser. WIMS '18. New York, NY, USA: Association for Computing Machinery, 2018.

[5] U. Kursuncu, M. Gaur, and A. Sheth, "Knowledge Infused Learning (K-IL): Towards Deep Incorporation of Knowledge in Deep Learning," 2020.

[6] R. Liu, Z. Wang, X. Shi, H. Zhao, S. Qiu, J. Li, and N. Yang, "Table Tennis Stroke Recognition Based on Body Sensor Network," in *Internet and Distributed Computing Systems*, R. Montella, A. Ciaramella, G. Fortino, A. Guerrieri, and A. Liotta, Eds. Cham, Switzerland: Springer International Publishing, 2019, pp. 1–10.

[7] K. Dokic, T. Mesic, and M. Martinovic, "Table Tennis Forehand and Backhand Stroke Recognition Based on Neural Network," in *Advances in Computing and Data Sciences*. Singapore: Springer Singapore, 2020, pp. 24–35.

[8] Z. Fu, K.-I. Shu, and H. Zhang, "Ping Pong Motion Recognition based on Smart Watch," in *Proceedings of the 3rd International Conference on Mechatronics Engineering and Information Technology (ICMEIT 2019)*. Atlantis Press, Apr 2019, pp. 617–625.

[9] P. Blank, J. Hoßbach, D. Schuldhaus, and B. M. Eskofier, "Sensor-Based Stroke Detection and Stroke Type Classification in Table Tennis," in *Proceedings of the 2015 ACM International Symposium on Wearable Computers*, ser. ISWC '15. New York, NY, USA: Association for Computing Machinery, 2015, pp. 93—100.

[10] P. Jin, B. Li, and W. Zhaohui, "Low-power-consumption high-precision table tennis movement identification method and device," Dec 2018, patent no.: CN109011505A, Applicants: Univ South China Tech.

[11] Z. Han, S. Guobin, and X. Dai, "Automatic rally detection and scoring," Oct 2020, patent no.: US10751601B2, Applicants: Beijing Shunyuan Kaihua Tech Limited.

[12] M. W. Kim, M. H. Kim, and J. Y. Park, "System for analyzing sport exercise by club device based on complex motion detecting senssor," Mai 2017, patent no.: KR101736489B1, Applicants: Korea Inst Footwear & Leather Tech.

[13] C. Olah, "Understanding LSTM Networks," 2015, accessed: 2021-03-24. [Online]. Available: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

[14] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, pp. 1735–1780, Dec 1997.

[15] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ser. ICML'13. JMLR.org, 2013, p. III–1310–III–1318.

[16] J. Bayer, "Learning Sequence Representations," Dissertation, Technische Universität München, München, 2015.

[17] M. Gaertler, D. Handke, and S. Wagner, "Theoretische Grundlagen der Informatik: Vorläufiges Skript zur Vorlesung von Prof. Dr. Dorothea Wagner WS 11/12," pp. 7–25.

[18] Newgy Industries, Inc., *Robo-Pong 3050XL Owner's Manual*, accessed: 2021-03-22. [Online]. Available: https://cdn.shopify.com/s/files/1/2677/3302/files/3050XL_Owners_Manual_1.5.21.pdf?v=1609863688

[19] D. Weiß, "Numerische Mathematik für die Fachrichtungen Informatik und Ingenieurwesen," Karlsruhe Institute of Technology, 2017.

[20] A. Mertins, *Signaltheorie: Grundlagen der Signalbeschreibung, Filterbänke, Wavelets, Zeit-Frequenz-Analyse, Parameter- und Signalschätzung*, 4th ed. Wiesbaden: Springer Vieweg, 2020.

[21] P. S. Addison, *The illustrated wavelet transform handbook: introductory theory and applications in science, engineering, medicine and finance*, 2nd ed. Boca Raton, FL: CRC Press, Taylor & Francis Group, 2016.

[22] J. P. Bravo, S. Roque, R. Estrela, I. C. Leão, and J. R. De Medeiros, "Wavelets: a powerful tool for studying rotation, activity, and pulsation in Kepler and CoRoT stellar light curves," *Astronomy & Astrophysics*, vol. 568, p. A34, Aug 2014.

[23] D. L. Donoho and I. M. Johnstone, "Ideal spatial adaptation by wavelet shrinkage," *Biometrika*, vol. 81, no. 3, pp. 425–455, 09 1994.

[24] C. Johnson, "Using PyWavelets To Remove High Frequency Noise," Jan 2016, accessed: 2021-04-12. [Online]. Available: https://connor-johnson.com/2016/01/24/using-pywavelets-to-remove-high-frequency-noise/

[25] S. Mallat, *A Wavelet Tour of Signal Processing: The Sparse Way*, 3rd ed. Elsevier Science, 2008.

[26] S. W. Smith, *The scientist and engineers guide to digital signal processing*, 2nd ed. San Diego, CA: California Technical Publishing, 1999.

[27] V. Barnett and T. Lewis, *Outliers in Statistical Data*, 3rd ed. New York: Wiley, 1994.

[28] E. Schubert, A. Zimek, and H.-P. Kriegel, "Generalized Outlier Detection with Flexible Kernel Density Estimates," in *Proceedings of the 14th SIAM International Conference on Data Mining (SDM)*, 2014.

[29] A. J. Robson, "Evidence for Trends in UK Flooding," *Philosophical Transactions: Mathematical, Physical and Engineering Sciences*, vol. 360, no. 1796, pp. 1327–1343, 2002.

[30] J. Fox, *Applied Regression Analysis and Generalized Linear Models*, 3rd ed. SAGE Publications, 2015.

[31] Oxford University Press, "domain," accessed: 2021-03-28. [Online]. Available: https://www.oxfordlearnersdictionaries.com/definition/english/domain?q=domain

[32] Oxford University Press, "knowledge," accessed: 2021-03-28. [Online]. Available: https://www.oxfordlearnersdictionaries.com/definition/english/knowledge?q=knowledge

[33] W. B. Croft, "User-Specified Domain Knowledge for Document Retrieval," in *Proceedings of the 9th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR '86. New York, NY, USA: Association for Computing Machinery, 1986, pp. 201—-206.

[34] H. L. Chiesi, G. J. Spilich, and J. F. Voss, "Acquisition of domain-related information in relation to high and low domain knowledge," *Journal of Verbal Learning and Verbal Behavior*, vol. 18, no. 3, pp. 257–273, 1979.

[35] B.-U. Groß, *Tischtennis Basics : [alle Grundschlagtechniken in 30 Bildreihen; Aufschläge, Beinarbeit und Stellungsspiel; Praxis- und Trainingstipps von Richard Prause]*, 6th ed. Aachen: Meyer & Meyer, 2015.

[36] Google Developers, "Sensors Overview," Dec 2019, accessed: 2021-03-22. [Online]. Available: https://developer.android.com/guide/topics/sensors/sensors_overview

[37] Keras SIG, "Keras Simple. Flexible. Powerful." accessed: 2021-04-16. [Online]. Available: https://keras.io/

[38] M. Afaneh, "Bluetooth GATT: How to Design Custom Services & Characteristics [MIDI device use case]," Jun 2017, accessed: 2021-04-03. [Online]. Available: https://www.novelbits.io/bluetooth-gatt-services-characteristics/

[39] A. Géron, *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow : concepts, tools, and techniques to build intelligent systems*, 2nd ed. Beijing: OReilly, Sep 2019.

[40] Keras SIG, "Keras layers API," accessed: 2021-04-13. [Online]. Available: https://keras.io/api/layers/

[41] V. Pankratius, "Edge-AI in Software and Sensors Applications: 5 - Neural Networks Practice: How-to, Dos & Don'ts," Karlsruhe Institute of Technology, May 2020.

[42] A. Graves and J. Schmidhuber, "Framewise phoneme classification with bidirectional LSTM and other neural network architectures," *Neural Networks*, vol. 18, no. 5, pp. 602–610, 2005, iJCNN 2005.

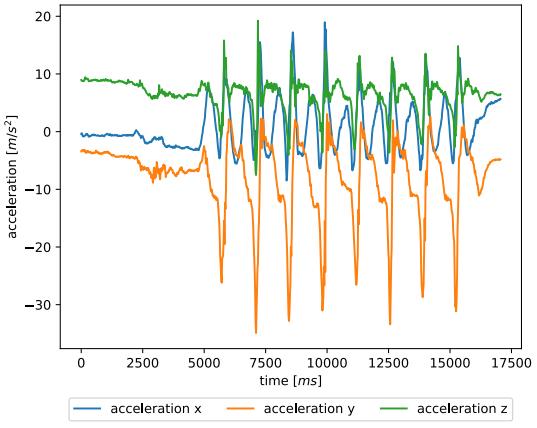
[43] TensorFlow, "Get started with TensorFlow Lite," Jan 2021, accessed: 2021-04-07. [Online]. Available: https://www.tensorflow.org/lite/guide/get_started

[44] TensorFlow, "Post-training float16 quantization," Mar 2021, accessed: 2021-04-16. [Online]. Available: https://www.tensorflow.org/lite/performance/post_training_float16_quant

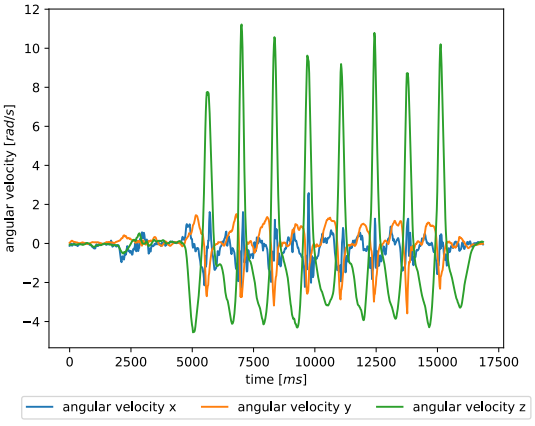
[45] Bitkom, "Absatz von Smartwatches in Deutschland in den Jahren 2018 bis 2020 (in Millionen Stück)," Statista GmbH, Aug. 2020, accessed: 2021-05-15. [Online]. Available: https://de.statista.com/statistik/daten/studie/459093/umfrage/absatz-von-smartwatches-in-deutschland/

Appendix

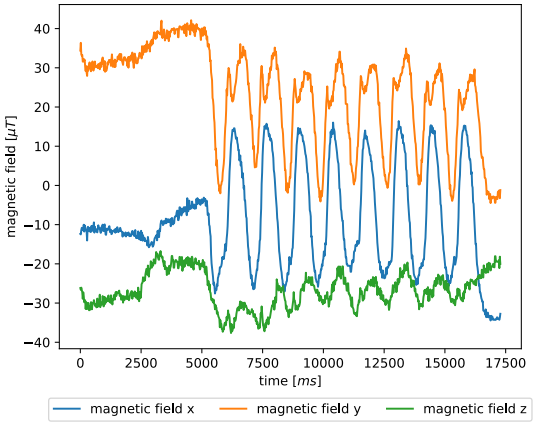
A Exemplary Stroke Signals



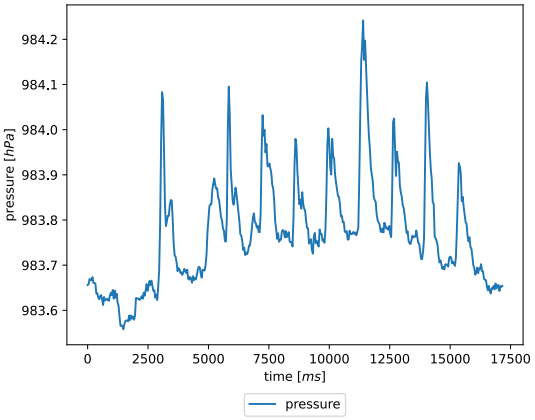
(a) Acceleration



(b) Angular Velocity

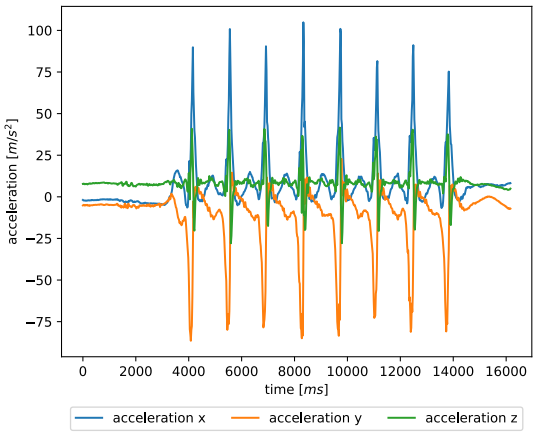


(c) Magnetic Field

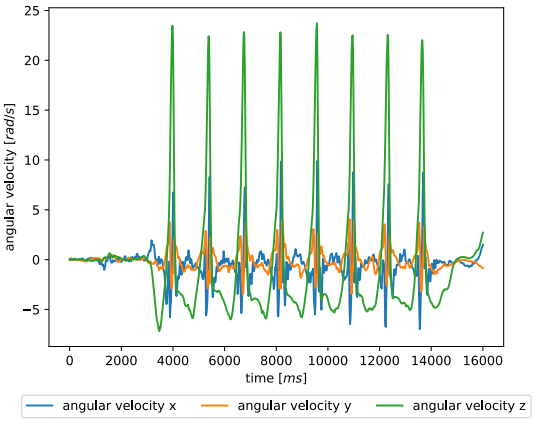


(d) Pressure

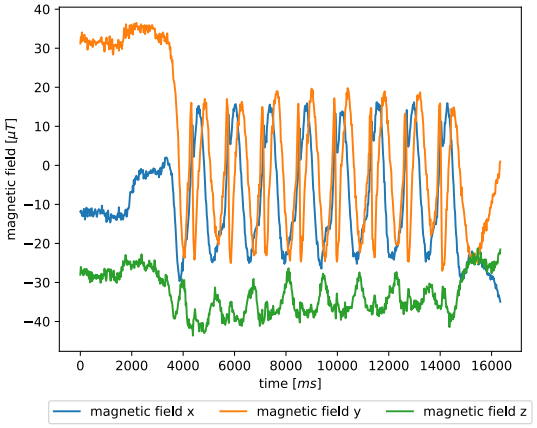
Figure A.1: Raw sensor data of eight forehand drives.



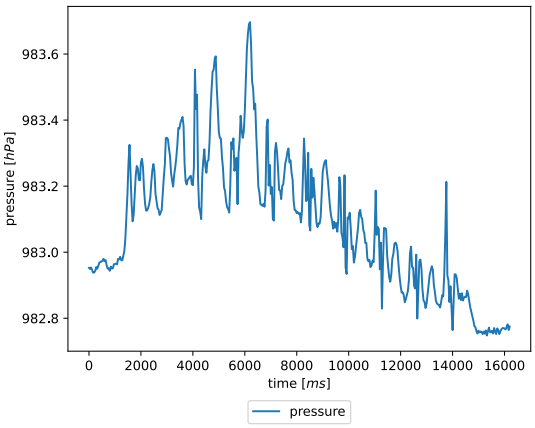
(a) Acceleration



(b) Angular Velocity

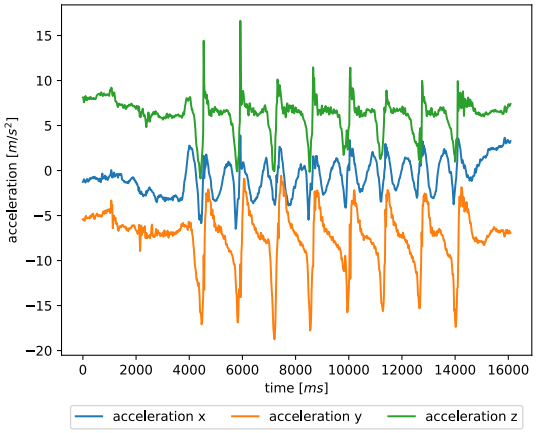


(c) Magnetic Field

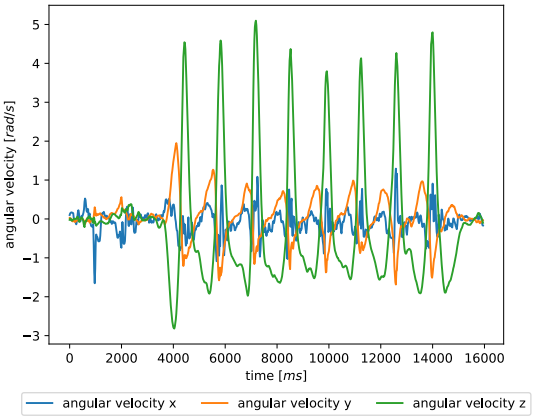


(d) Pressure

Figure A.2: Raw sensor data of eight forehand loops.

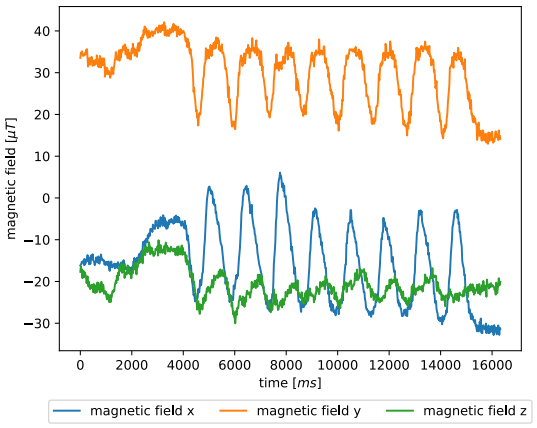


(a) Acceleration

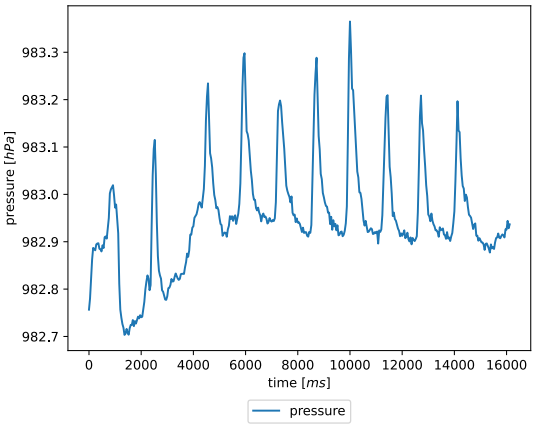


(b) Angular Velocity

Figure A.3: Raw sensor data of eight forehand blocks.

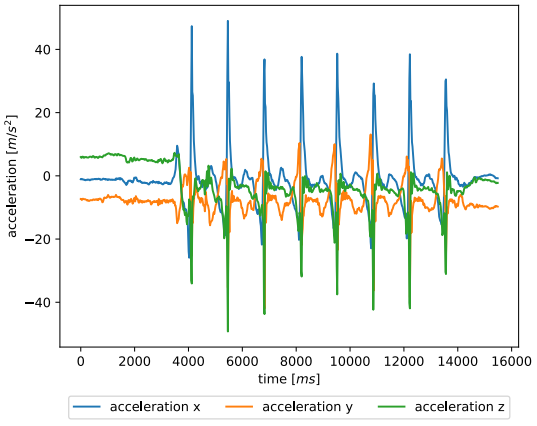


(c) Magnetic Field

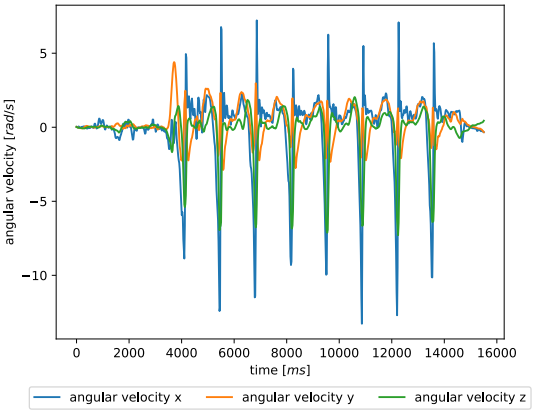


(d) Pressure

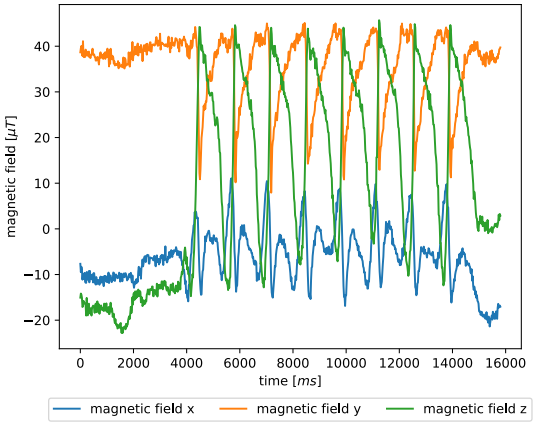
Figure A.3 (cont.): Raw sensor data of eight forehand blocks.



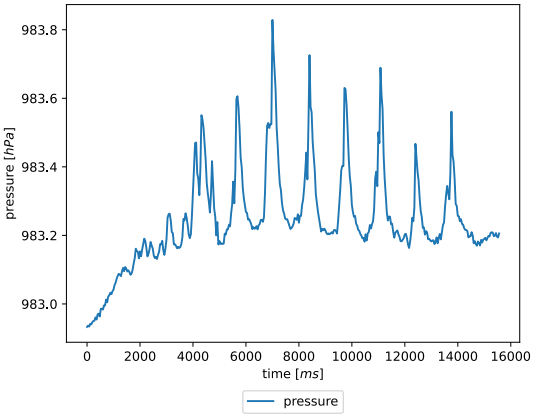
(e) Acceleration



(f) Angular Velocity

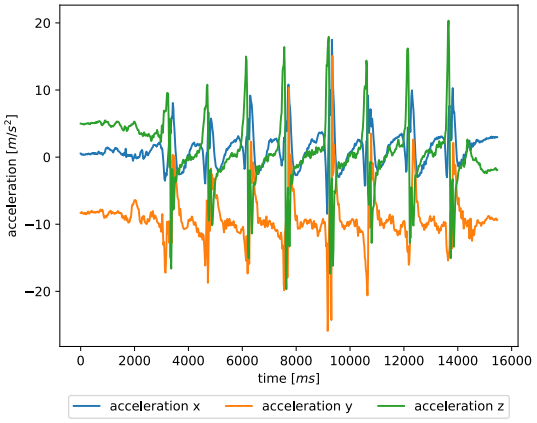


(g) Magnetic Field

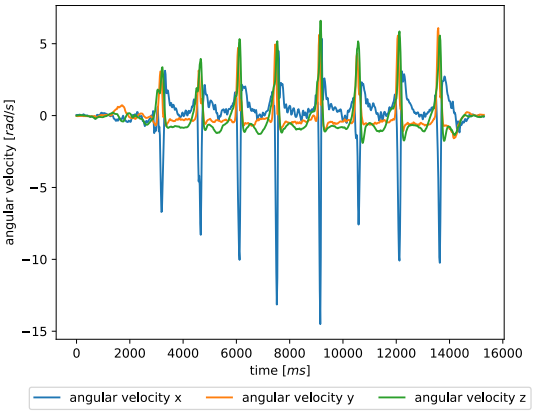


(h) Pressure

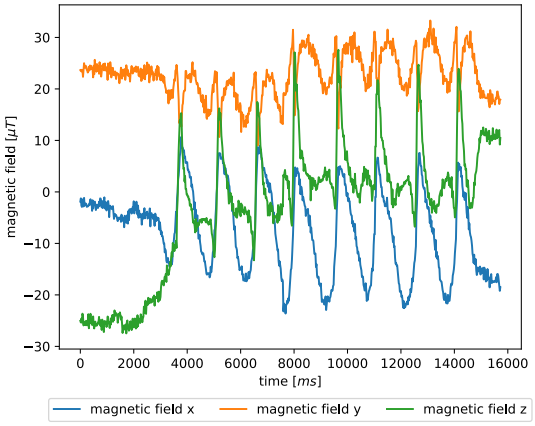
Figure A.4: Raw sensor data of eight forehand pushes.



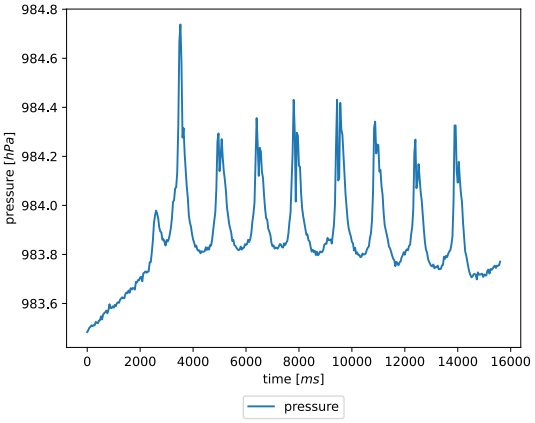
(a) Acceleration



(b) Angular Velocity

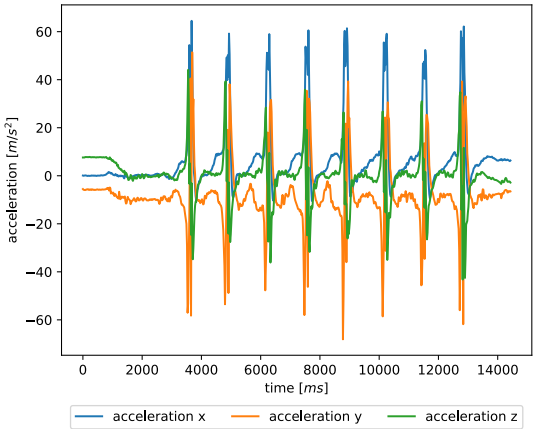


(c) Magnetic Field

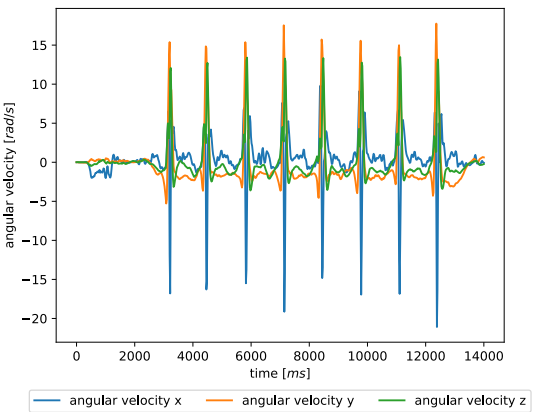


(d) Pressure

Figure A.5: Raw sensor data of eight backhand drives.

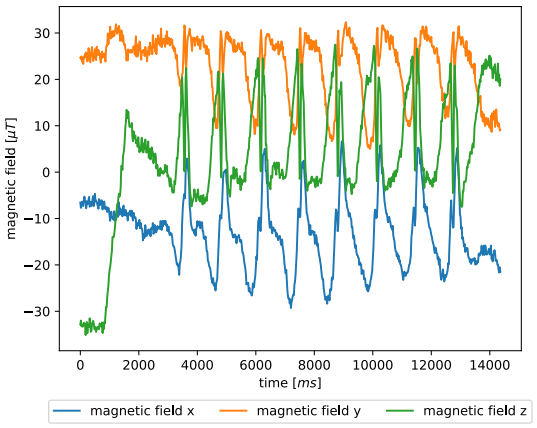


(a) Acceleration

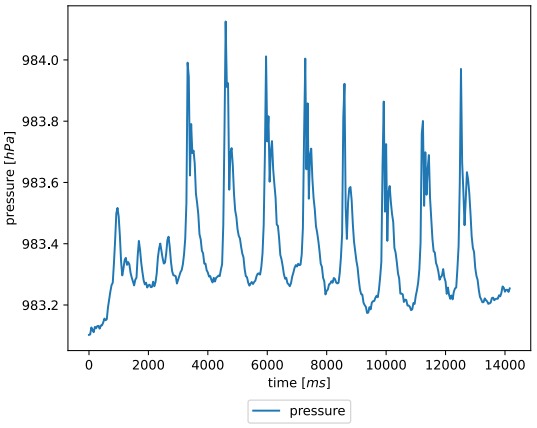


(b) Angular Velocity

Figure A.6: Raw sensor data of eight backhand loops.

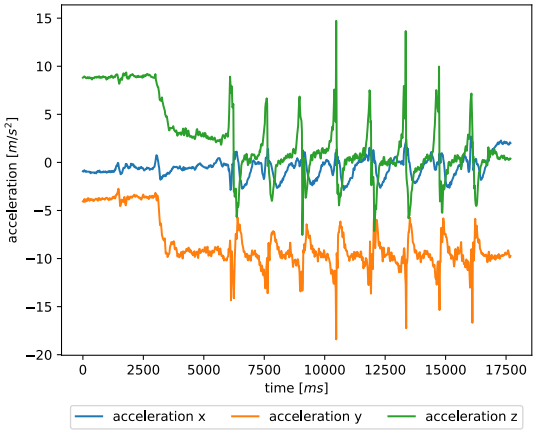


(c) Magnetic Field

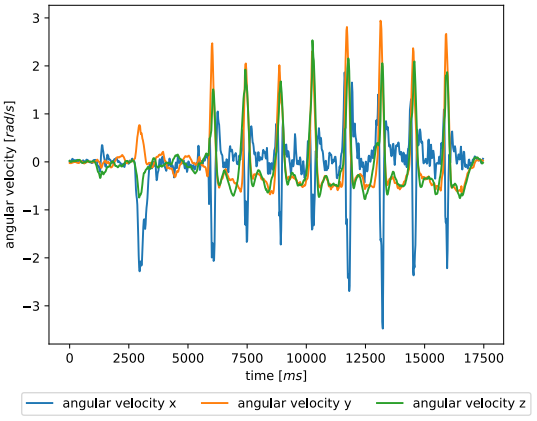


(d) Pressure

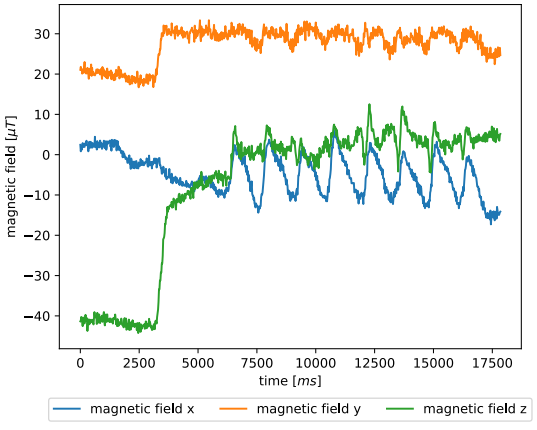
Figure A.6 (cont.): Raw sensor data of eight backhand loops.



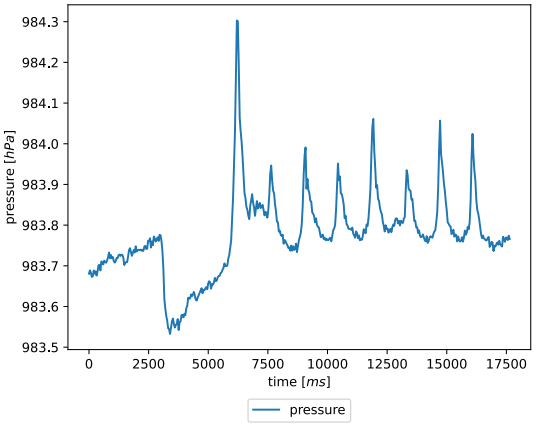
(e) Acceleration



(f) Angular Velocity

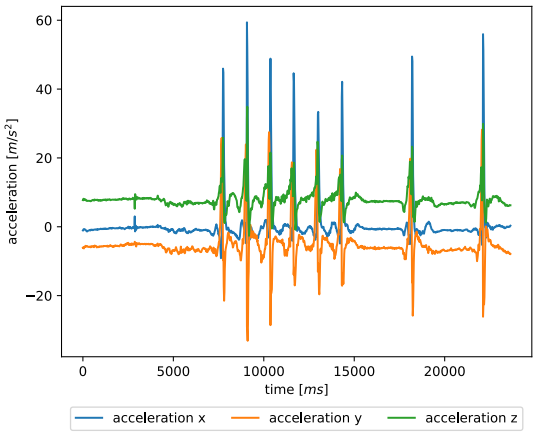


(g) Magnetic Field

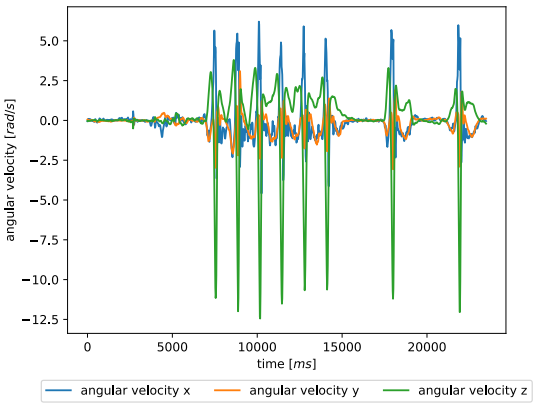


(h) Pressure

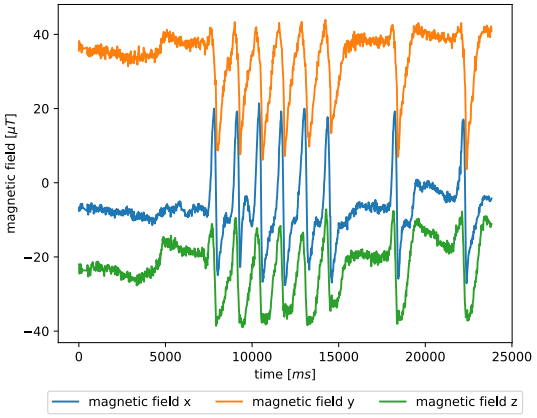
Figure A.7: Raw sensor data of eight backhand blocks.



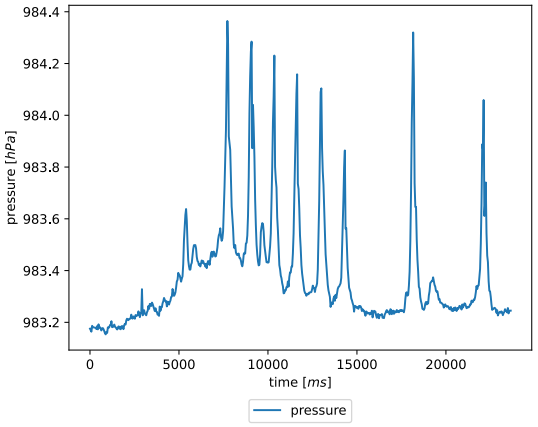
(a) Acceleration



(b) Angular Velocity



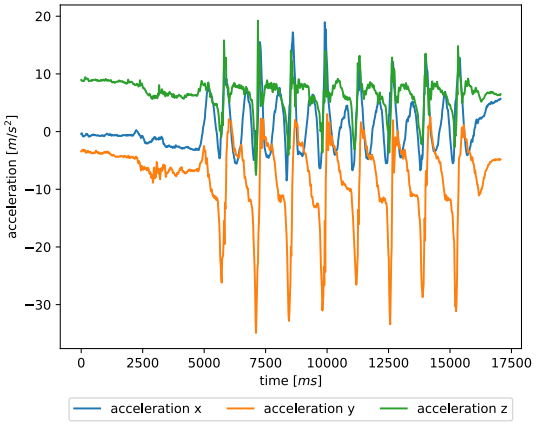
(c) Magnetic Field



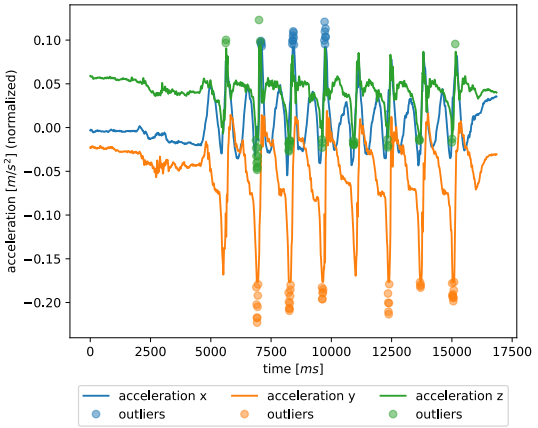
(d) Pressure

Figure A.8: Raw sensor data of eight backhand pushes.

B Exemplary Preprocessing Results

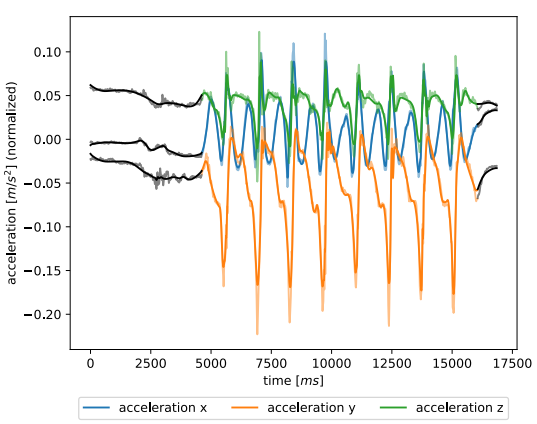


(a) Raw acceleration

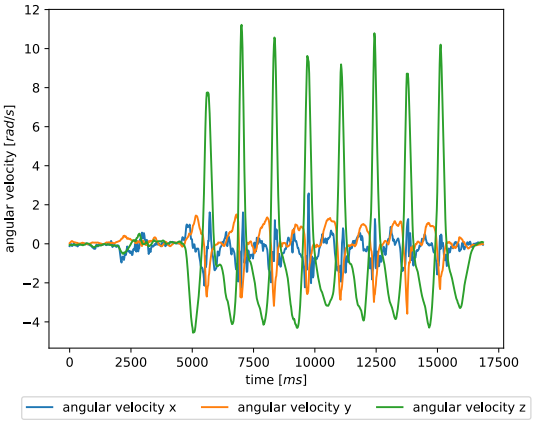


(b) Acceleration without outliers

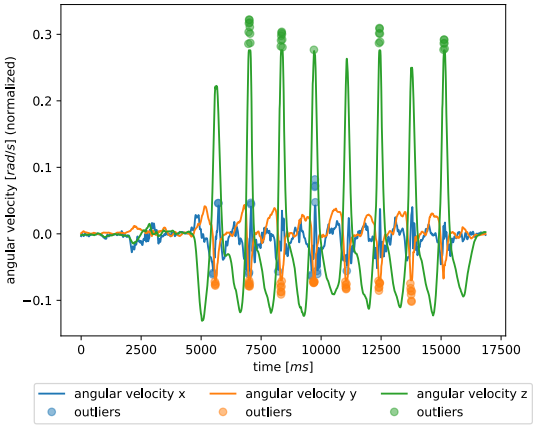
Figure B.9: Interim results of the preprocessing pipeline applied to a time series containing eight forehand drives. The preprocessed data contains non-stroke noise actions (black) and the raw data (pale).



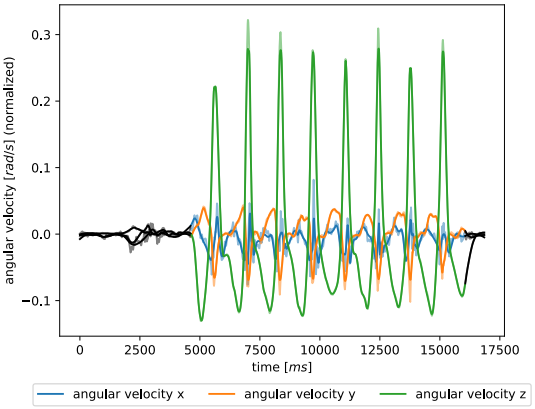
(c) Preprocessed acceleration



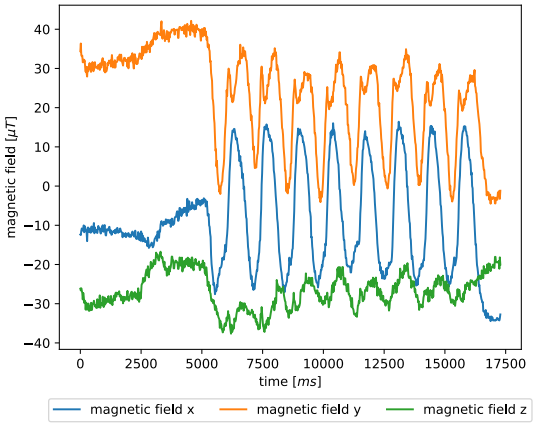
(d) Raw angular velocity



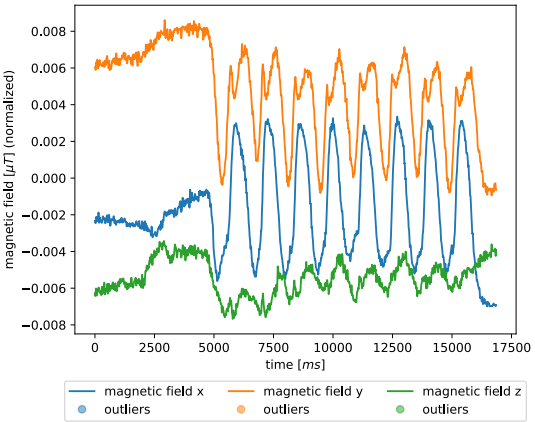
(e) Angular velocity without outliers



(f) Preprocessed angular velocity

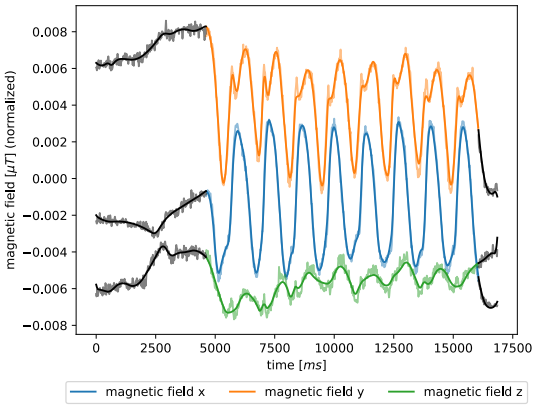


(g) Raw magnetic field

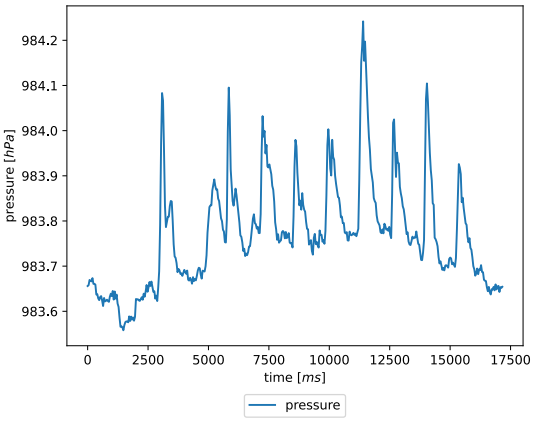


(h) Magnetic field without outliers

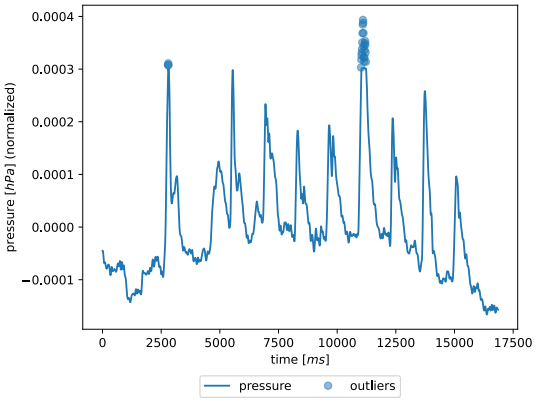
Figure B.9 (cont.): Interim results of the preprocessing pipeline applied to a time series containing eight forehand drives. The preprocessed data contains non-stroke noise actions (black) and the raw data (pale).



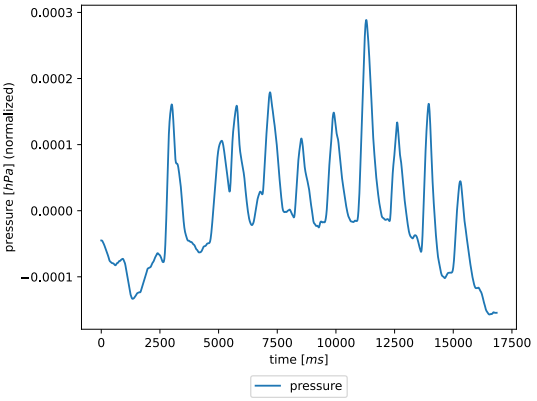
(i) Preprocessed magnetic field



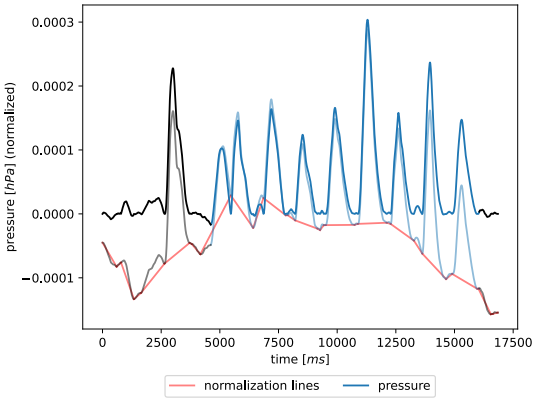
(j) Raw pressure



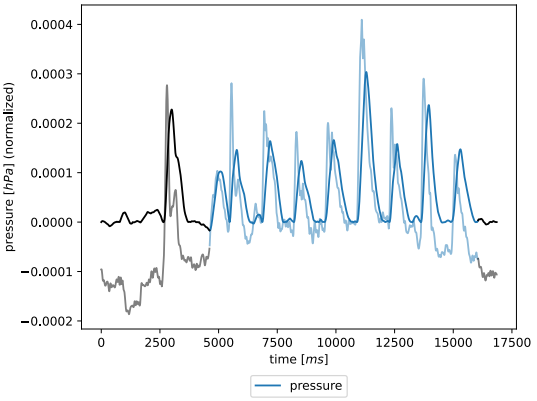
(k) Pressure without outliers and linear trend



(l) Filtered pressure

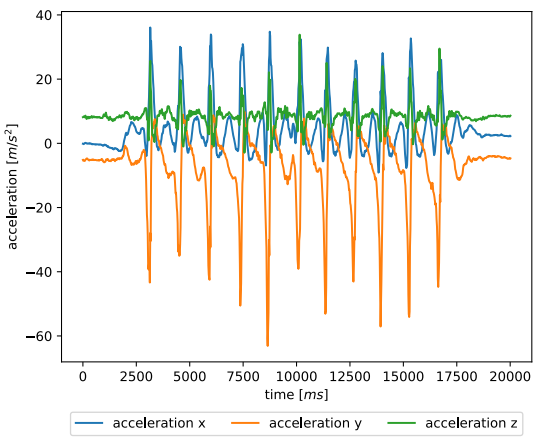


(m) Trend Analysis (custom) pressure

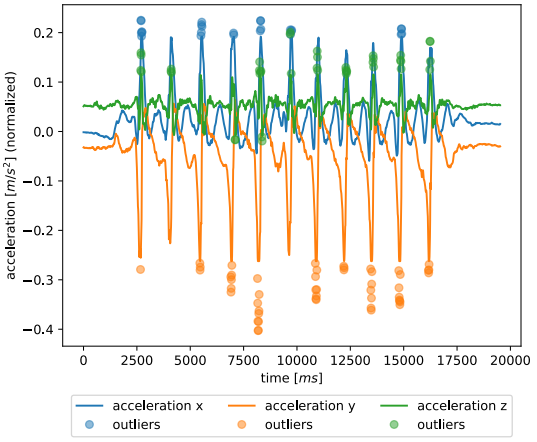


(n) Preprocessed pressure

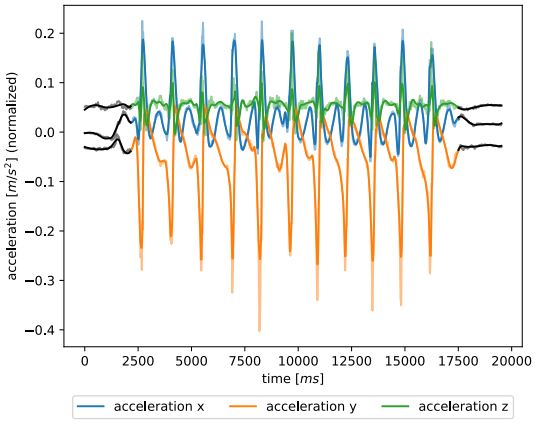
Figure B.9 (cont.): Interim results of the preprocessing pipeline applied to a time series containing eight forehand drives. The preprocessed data contains non-stroke noise actions (black) and the raw data (pale).



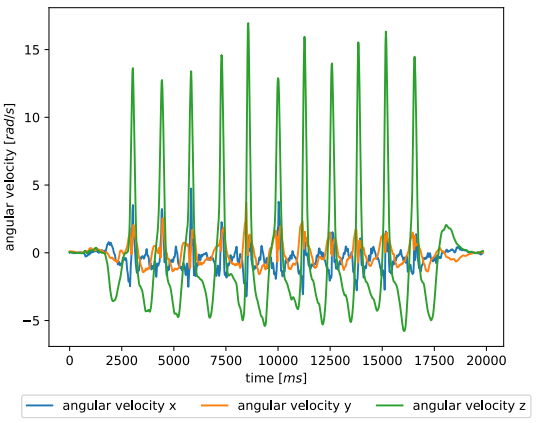
(a) Raw acceleration



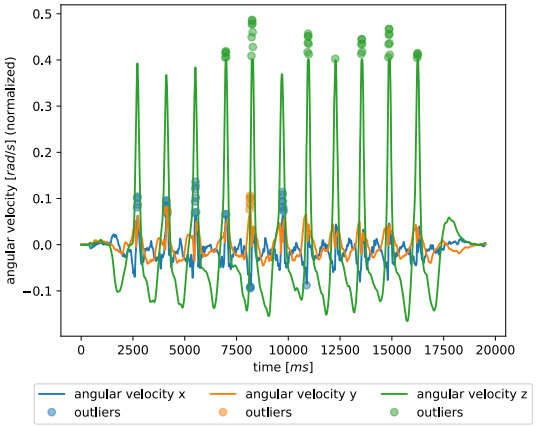
(b) Acceleration without outliers



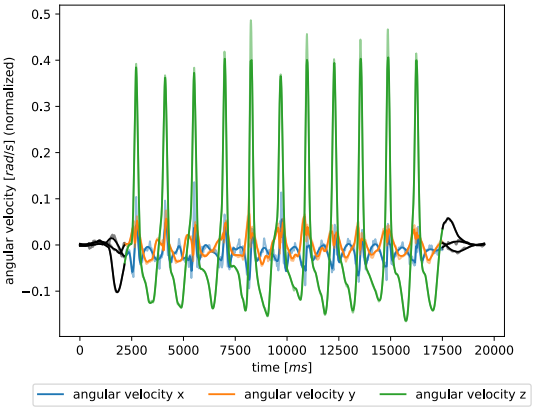
(c) Preprocessed acceleration



(d) Raw angular velocity

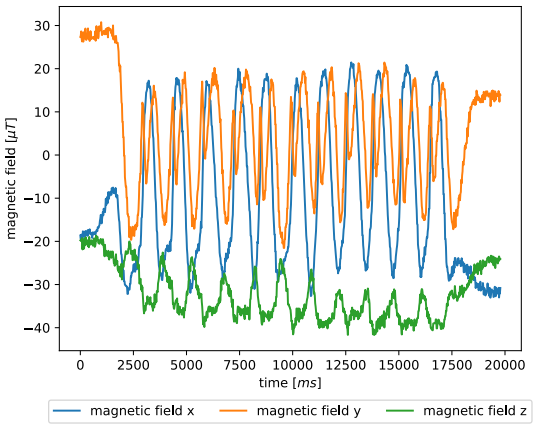


(e) Angular velocity without outliers

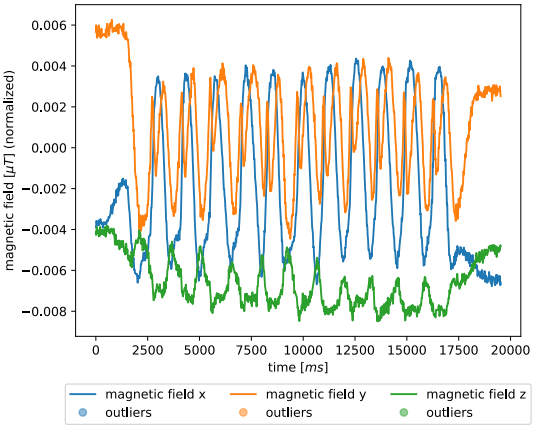


(f) Preprocessed angular velocity

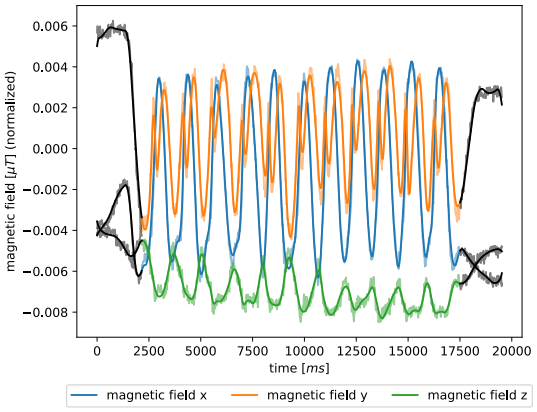
Figure B.10: Interim results of the preprocessing pipeline applied to a time series containing eleven forehand drives. The preprocessed data contains non-stroke noise actions (black) and the raw data (pale).



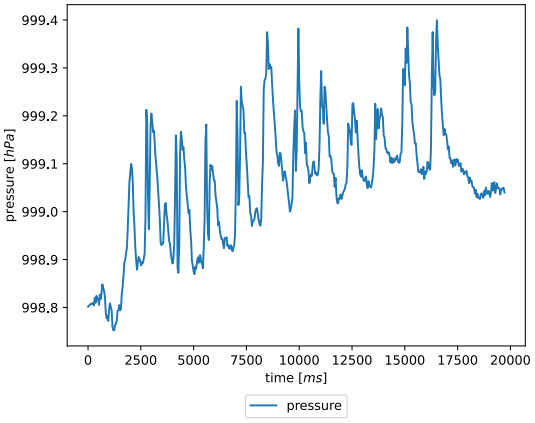
(g) Raw magnetic field



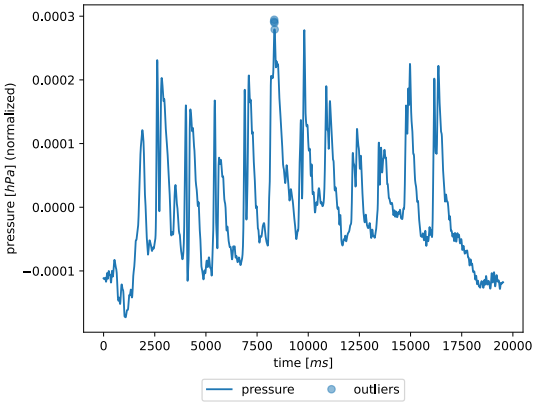
(h) Magnetic field without outliers



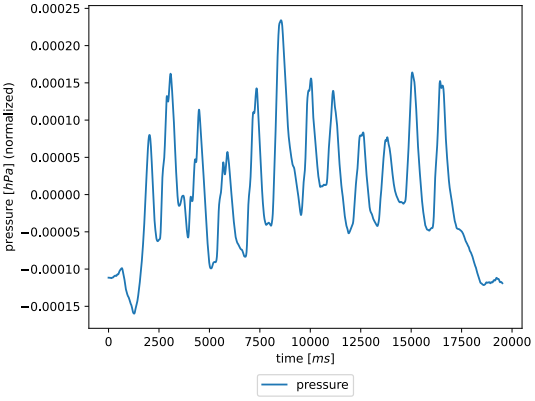
(i) Preprocessed magnetic field



(j) Raw pressure

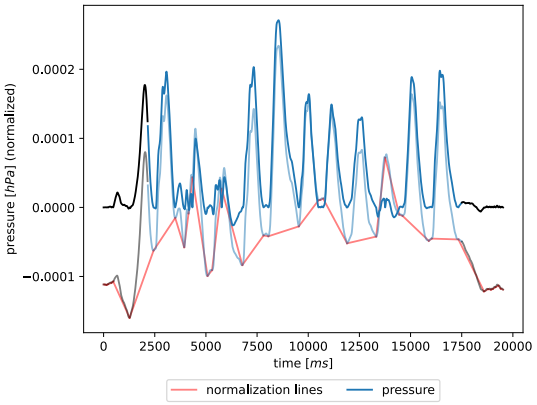


(k) Pressure without outliers and linear trend

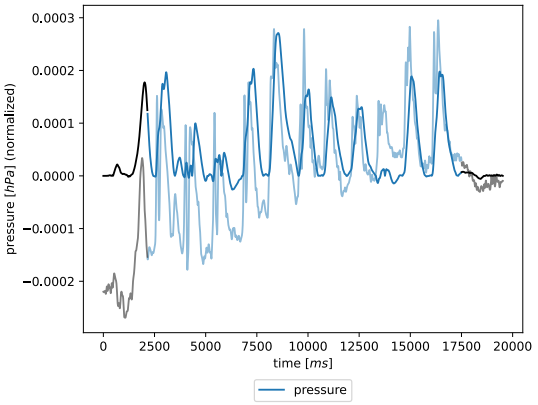


(l) Filtered pressure

Figure B.10 (cont.): Interim results of the preprocessing pipeline applied to a time series containing eleven forehand drives. The preprocessed data contains non-stroke noise actions (black) and the raw data (pale).



(m) Trend Analysis (custom) pressure



(n) Preprocessed pressure

Figure B.10 (cont.): Interim results of the preprocessing pipeline applied to a time series containing eleven forehand drives. The preprocessed data contains non-stroke noise actions (black) and the raw data (pale).

C Stroke Execution

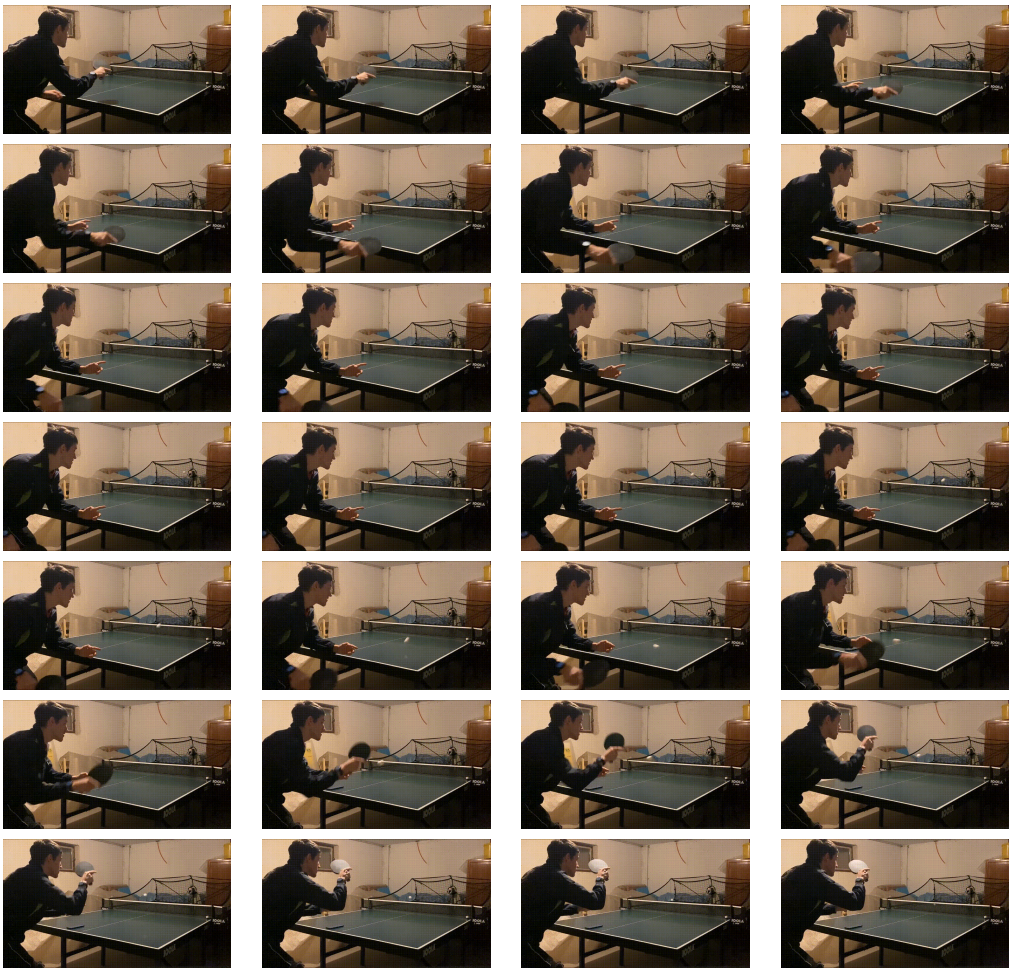


Figure C.11: Execution of a forehand drive.

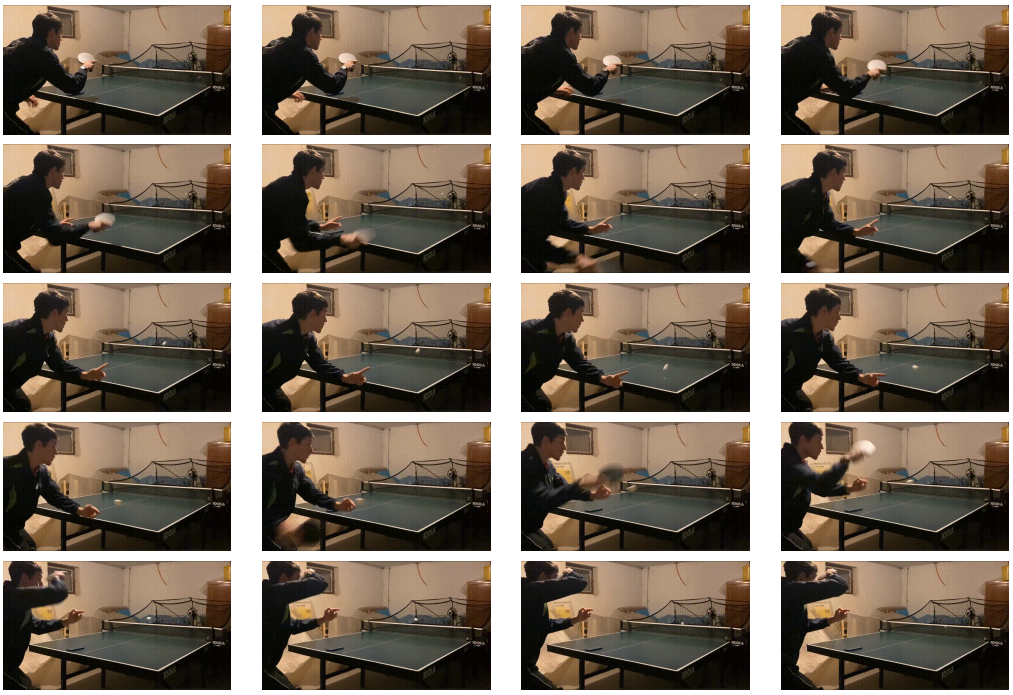


Figure C.12: Execution of a forehand loop.

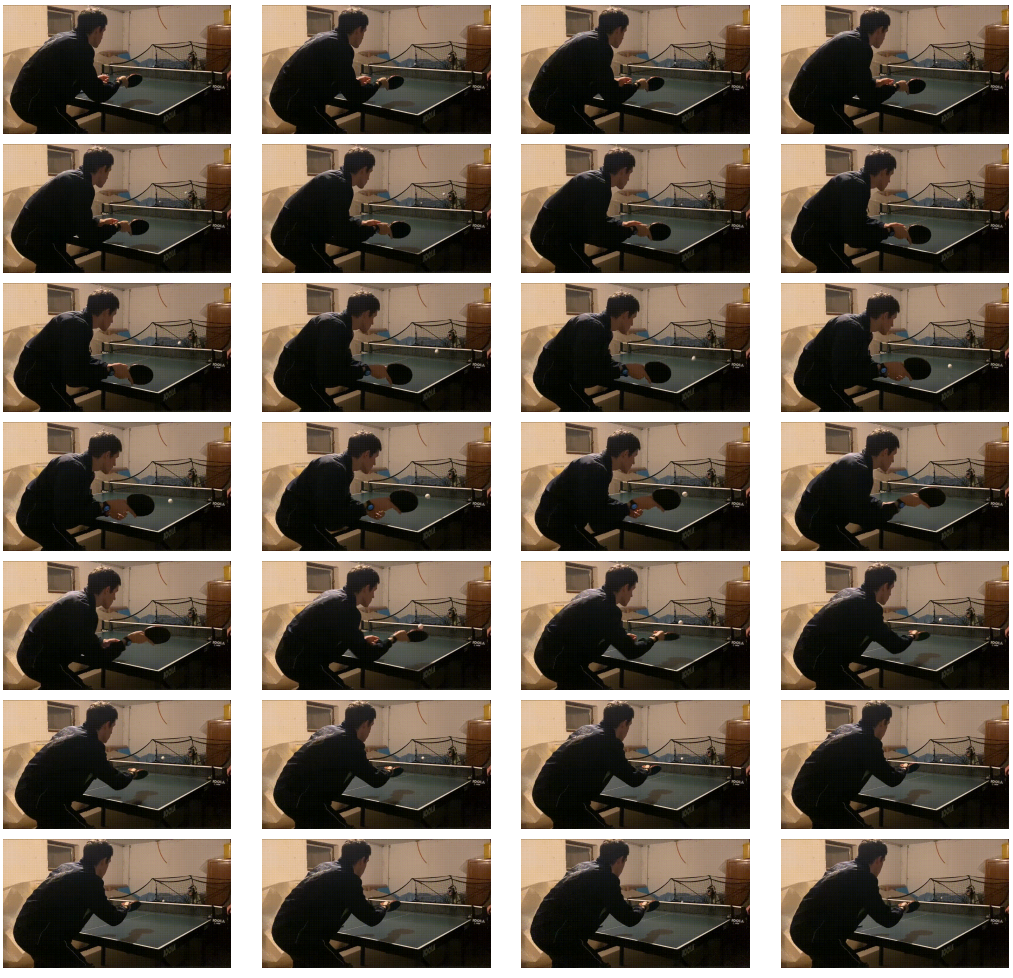


Figure C.13: Execution of a forehand push.

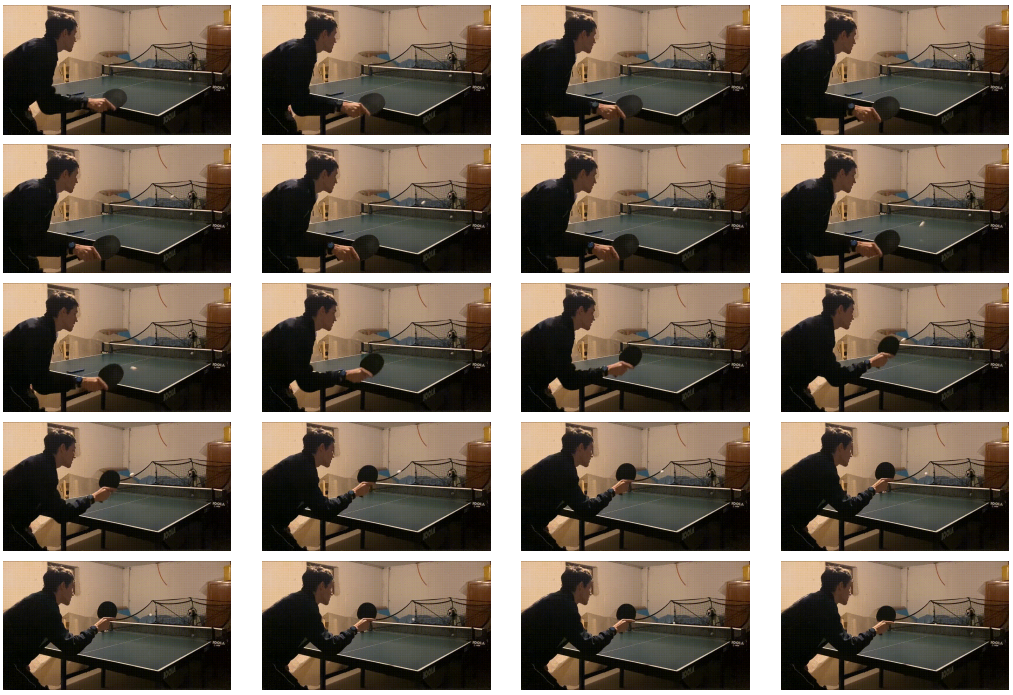


Figure C.14: Execution of a forehand block.

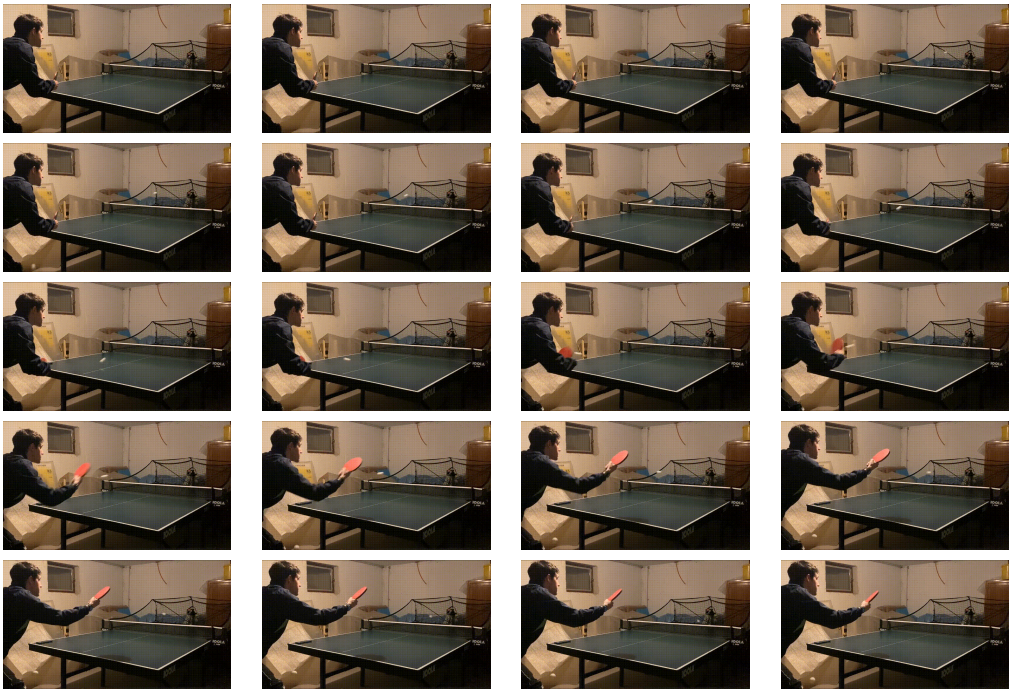


Figure C.15: Execution of a backhand drive.

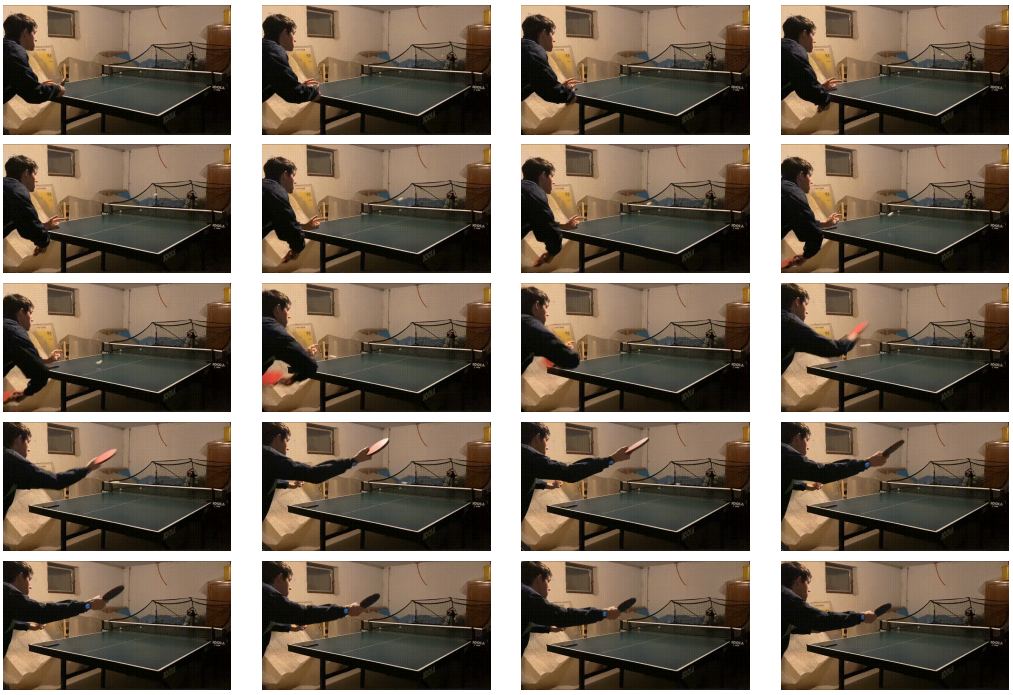


Figure C.16: Execution of a backhand loop.

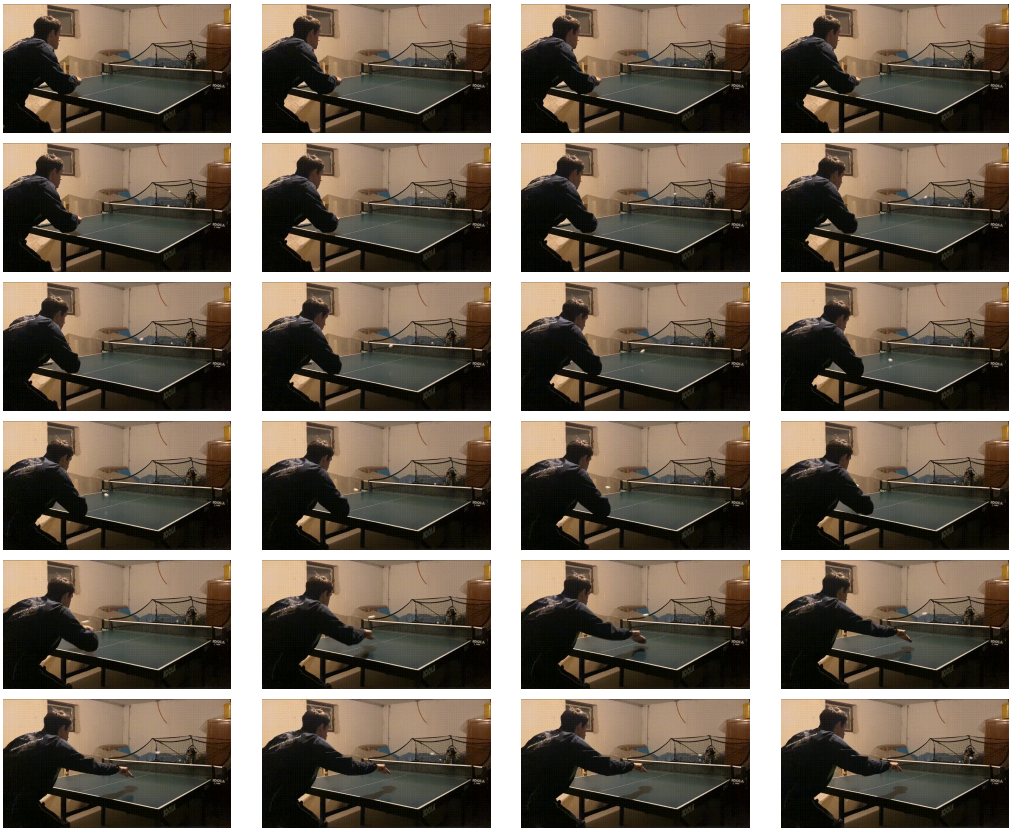


Figure C.17: Execution of a backhand push.

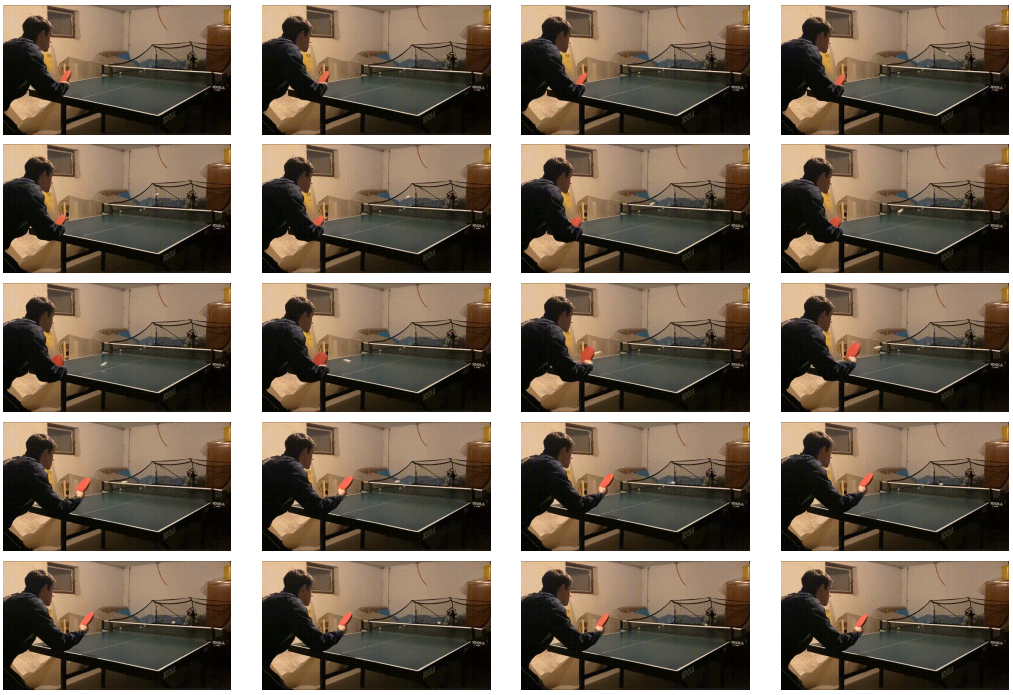


Figure C.18: Execution of a backhand block.

D Transition Probabilities

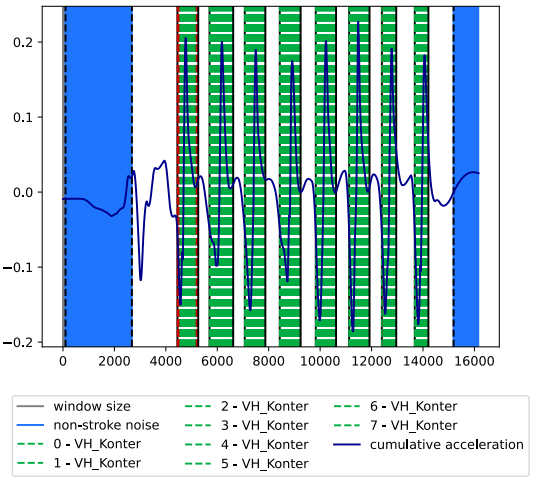
Table D.1: The transition probabilities of the Stroke Sequence State Machine derived from [18]. Note that the transitions from the state X to the Waiting state are not mentioned, because these transitions are only used at the end of a time series to return to the Waiting state and therefore do not need to be considered here.

	FH Drive	BH Drive	FH Push	BH Push
FH Drive	0.42857	0.22449	0.08163	0.02041
BH Drive	0.07576	0.5	0.01515	0.06061
FH Push	0.04225	0.02817	0.28169	0.16901
BH Push	0.03125	0.04688	0.125	0.3125
FH Loop	0.05556	0.01587	0.06349	0.00794
BH Loop	0.05618	0.06742	0.02247	0.06742
FH Block	0.07273	0.03636	0.01818	0.01818
BH Block	0.02041	0.10204	0.04082	0.02041
Waiting	0.125	0.125	0.125	0.125

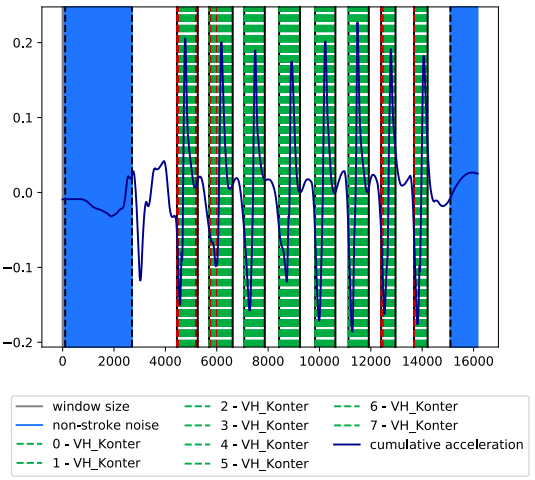
	FH Loop	BH Loop	FH Block	BH Block
FH Drive	0.18367	0.02041	0.02041	0.02041
BH Drive	0.15152	0.15152	0.0303	0.01515
FH Push	0.1831	0.07042	0.14085	0.08451
BH Push	0.09375	0.125	0.125	0.14062
FH Loop	0.57937	0.19048	0.02381	0.06349
BH Loop	0.24719	0.50562	0.02247	0.01124
FH Block	0.21818	0.09091	0.30909	0.23636
BH Block	0.16327	0.20408	0.16327	0.28571
Waiting	0.125	0.125	0.125	0.125

E Evaluation

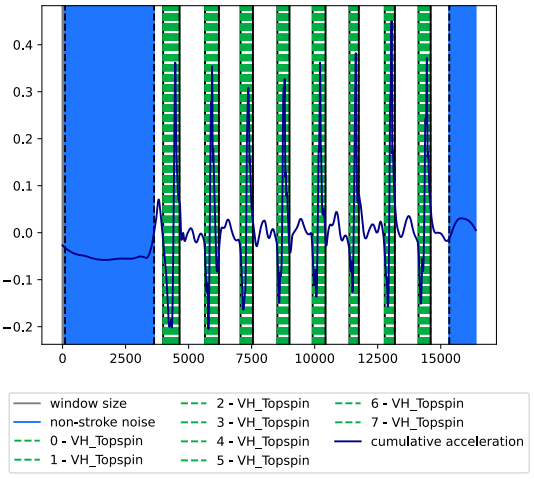
E.1 Stroke Extraction and Classification



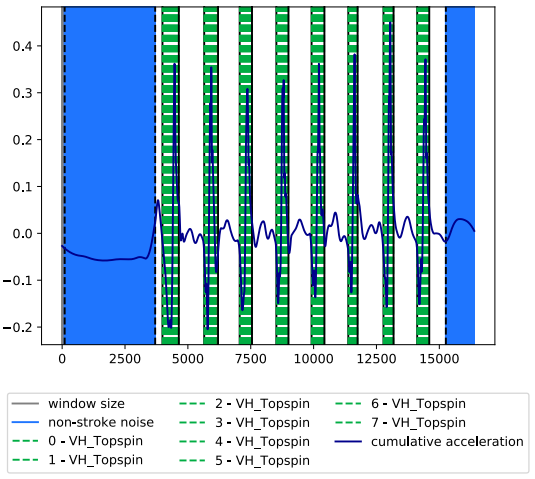
(a) Forehand Drive (modular)



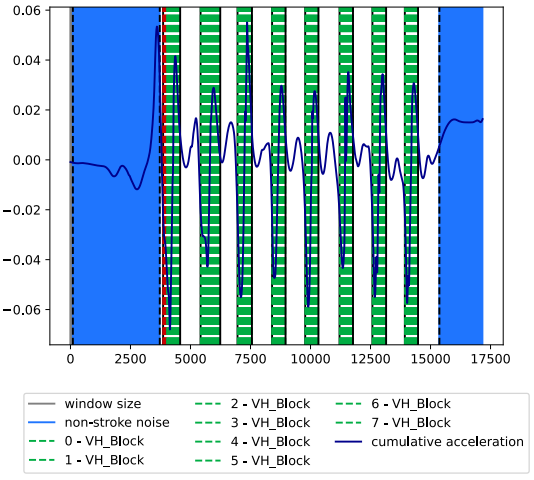
(b) Forehand Drive (non-modular)



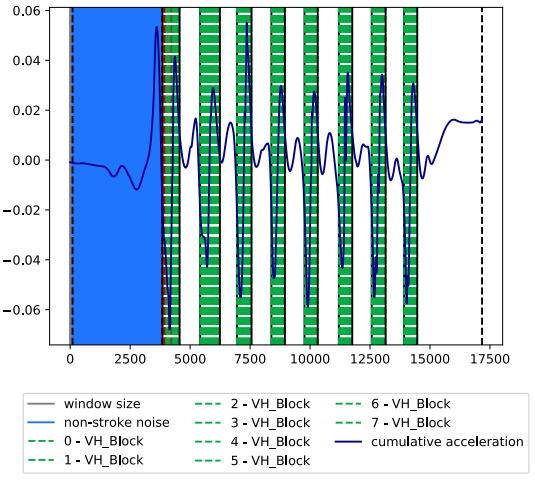
(c) Forehand Loop (modular)



(d) Forehand Loop (non-modular)

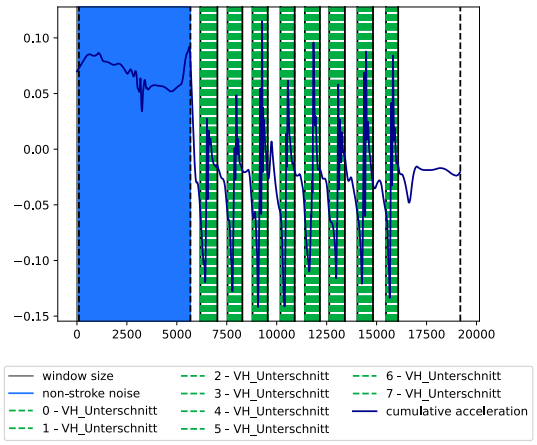


(e) Forehand Block (modular)

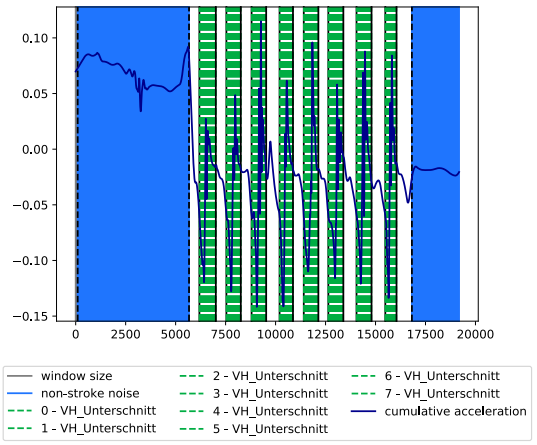


(f) Forehand Block (non-modular)

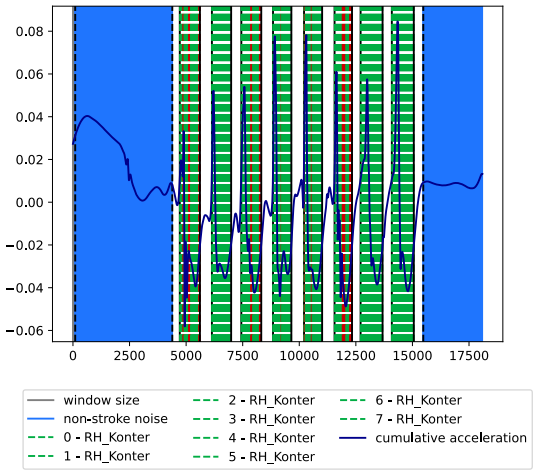
Figure E.19: Exemplary stroke extraction and classification results on the test set of player 1. Comparison of the modular and the non-modular output.



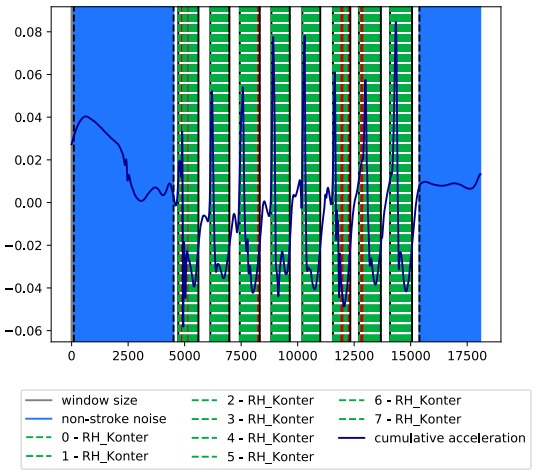
(g) Forehand Push (modular)



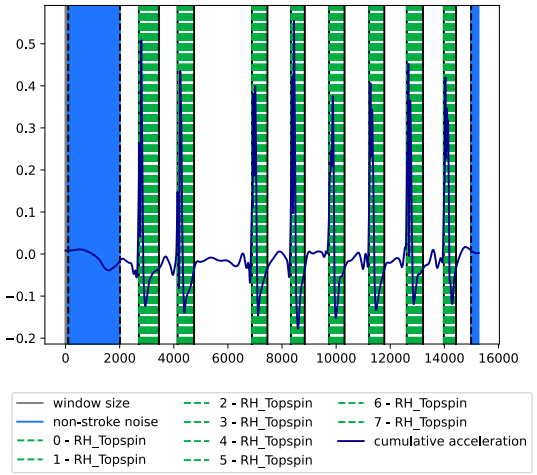
(h) Forehand Push (non-modular)



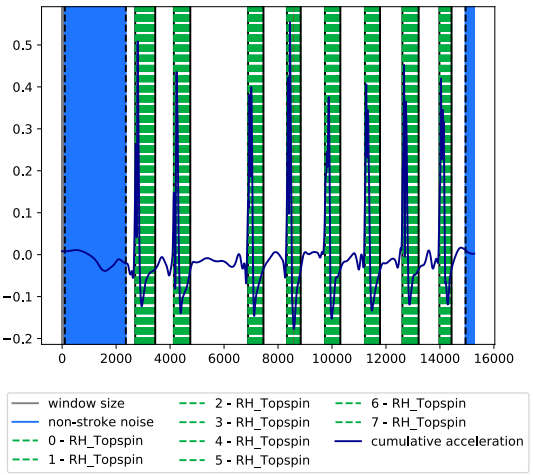
(i) Backhand Drive (modular)



(j) Backhand Drive (non-modular)

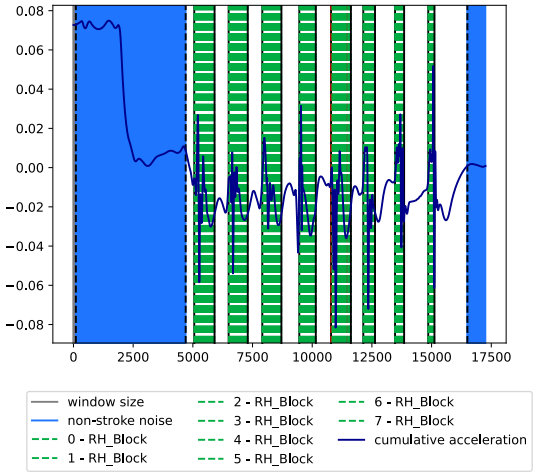


(k) Backhand Loop (modular)

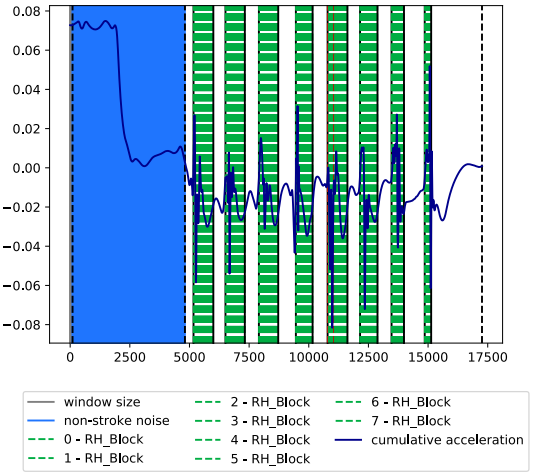


(l) Backhand Loop (non-modular)

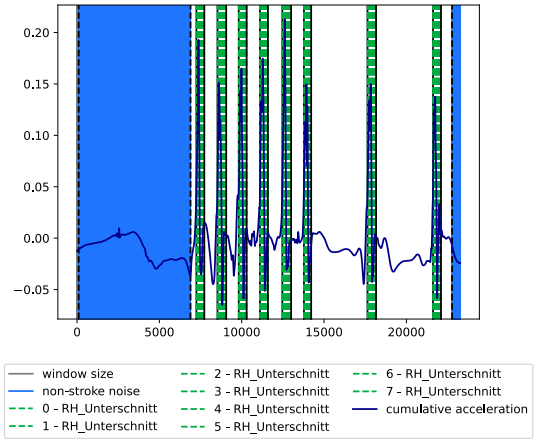
Figure E.19 (cont.): Exemplary stroke extraction and classification results on the test set of player 1. Comparison of the modular and the non-modular output.



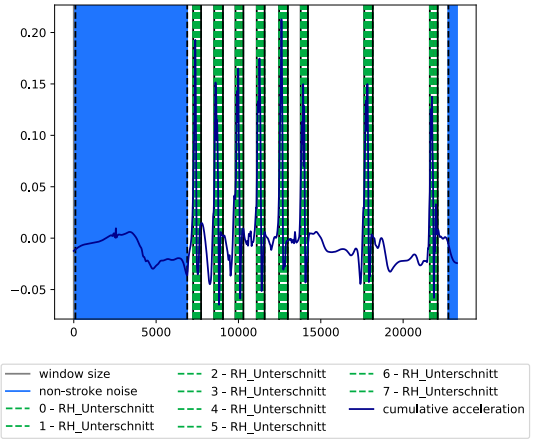
(m) Backhand Block (modular)



(n) Backhand Block (non-modular)



(o) Backhand Push (modular)



(p) Backhand Push (non-modular)

Figure E.19 (cont.): Exemplary stroke extraction and classification results on the test set of player 1. Comparison of the modular and the non-modular output.

E.2 Stroke Analysis

Table E.2: The stroke analysis results of the strokes visualized in Appendix E.1. Only the modular approach is considered.

Stroke Type-Index	Acceleration	Angular Velocity	Velocity	Angle
Forehand Drive-0	+0.44 %	+0.00 %	+0.00679 m/s	+0.003 deg
Forehand Drive-1	+1.29 %	-0.00 %	+0.00792 m/s	-0.003 deg
Forehand Drive-2	-0.60 %	-0.01 %	+0.00821 m/s	-0.019 deg
Forehand Drive-3	-1.05 %	-0.01 %	+0.00886 m/s	-0.015 deg
Forehand Drive-4	-0.67 %	-0.03 %	+0.00491 m/s	-0.036 deg
Forehand Drive-5	-1.47 %	-0.01 %	+0.00331 m/s	-0.006 deg
Forehand Drive-6	-0.68 %	-0.01 %	+0.00717 m/s	-0.030 deg
Forehand Drive-7	-0.45 %	-0.00 %	+0.00855 m/s	-0.003 deg
Forehand Loop-0	+0.21 %	-0.09 %	-0.00531 m/s	-0.134 deg
Forehand Loop-1	+0.23 %	-0.07 %	+0.00595 m/s	-0.206 deg
Forehand Loop-2	+0.03 %	-0.03 %	+0.00174 m/s	-0.132 deg
Forehand Loop-3	-0.02 %	-0.05 %	-0.00210 m/s	-0.237 deg

Table E.2 (cont.): The stroke analysis results of the strokes visualized in Appendix E.1. Only the modular approach is considered.

Stroke Type-Index	Acceleration	Angular Velocity	Velocity	Angle
Forehand Loop-4	+0.05 %	-0.05 %	+0.00320 m/s	-0.192 deg
Forehand Loop-5	+0.04 %	-0.03 %	+0.00345 m/s	-0.170 deg
Forehand Loop-6	+0.17 %	-0.02 %	+0.02048 m/s	-0.099 deg
Forehand Loop-7	+0.03 %	-0.03 %	+0.00287 m/s	-0.133 deg
Forehand Block-0	-0.04 %	+0.02 %	+0.00084 m/s	+0.006 deg
Forehand Block-1	-0.09 %	+0.01 %	+0.00082 m/s	+0.004 deg
Forehand Block-2	-0.22 %	+0.02 %	+0.00193 m/s	+0.018 deg
Forehand Block-3	-0.21 %	+0.01 %	+0.00188 m/s	+0.010 deg
Forehand Block-4	-0.08 %	+0.02 %	+0.00143 m/s	+0.014 deg
Forehand Block-5	-0.11 %	+0.01 %	+0.00063 m/s	+0.009 deg
Forehand Block-6	-0.13 %	-0.01 %	+0.00151 m/s	-0.003 deg
Forehand Block-7	-0.12 %	+0.01 %	+0.00147 m/s	+0.010 deg
Forehand Push-0	+0.14 %	-0.01 %	-0.00812 m/s	+0.010 deg
Forehand Push-1	+0.15 %	-0.02 %	-0.00889 m/s	+0.016 deg
Forehand Push-2	+0.09 %	-0.04 %	-0.00473 m/s	+0.032 deg
Forehand Push-3	+0.15 %	-0.01 %	-0.00831 m/s	+0.015 deg
Forehand Push-4	+0.10 %	-0.02 %	-0.00556 m/s	+0.017 deg
Forehand Push-5	+0.17 %	-0.05 %	-0.01044 m/s	+0.039 deg
Forehand Push-6	+0.15 %	-0.00 %	-0.00664 m/s	-0.000 deg
Forehand Push-7	+0.24 %	-0.01 %	-0.01095 m/s	+0.017 deg
Backhand Drive-0	+0.02 %	+0.08 %	-0.00076 m/s	+0.029 deg
Backhand Drive-1	-0.02 %	+0.03 %	+0.00038 m/s	+0.015 deg
Backhand Drive-2	-0.02 %	+0.05 %	+0.00044 m/s	+0.028 deg
Backhand Drive-3	-0.01 %	+0.05 %	+0.00028 m/s	+0.031 deg
Backhand Drive-4	-0.03 %	+0.04 %	+0.00076 m/s	+0.021 deg
Backhand Drive-5	-0.00 %	+0.04 %	+0.00003 m/s	+0.025 deg
Backhand Drive-6	+0.03 %	+0.04 %	-0.00035 m/s	+0.021 deg
Backhand Drive-7	+0.04 %	+0.05 %	-0.00046 m/s	+0.025 deg
Backhand Loop-0	+2.22 %	+0.04 %	+0.00390 m/s	+0.028 deg
Backhand Loop-1	+17.97 %	+0.07 %	+0.00366 m/s	+0.076 deg
Backhand Loop-2	+0.23 %	+0.03 %	+0.00862 m/s	+0.028 deg
Backhand Loop-3	+0.35 %	+0.01 %	+0.01496 m/s	+0.007 deg
Backhand Loop-4	+0.23 %	+0.06 %	+0.00661 m/s	+0.086 deg
Backhand Loop-5	+0.38 %	+0.11 %	+0.01153 m/s	+0.126 deg
Backhand Loop-6	+0.14 %	+0.01 %	+0.00582 m/s	+0.007 deg
Backhand Loop-7	+0.22 %	+0.08 %	+0.01986 m/s	+0.116 deg
Backhand Block-0	+0.04 %	+0.00 %	-0.00105 m/s	+0.000 deg
Backhand Block-1	+0.13 %	-0.02 %	-0.00373 m/s	-0.005 deg
Backhand Block-2	+0.04 %	-0.05 %	-0.00083 m/s	-0.016 deg
Backhand Block-3	+0.10 %	-0.10 %	-0.00301 m/s	-0.036 deg
Backhand Block-4	+0.08 %	-0.02 %	-0.00281 m/s	-0.009 deg
Backhand Block-5	+0.02 %	-0.10 %	-0.00039 m/s	-0.063 deg
Backhand Block-6	+0.19 %	-0.10 %	-0.00112 m/s	-0.068 deg
Backhand Block-7	-1.32 %	-0.09 %	-0.00469 m/s	-0.091 deg
Backhand Push-0	+0.07 %	-0.03 %	+0.00361 m/s	+0.025 deg
Backhand Push-1	+0.07 %	-0.35 %	+0.00420 m/s	+0.042 deg
Backhand Push-2	+0.13 %	-0.10 %	+0.00878 m/s	+0.064 deg

Table E.2 (cont.): The stroke analysis results of the strokes visualized in Appendix E.1. Only the modular approach is considered.

Stroke Type-Index	Acceleration	Angular Velocity	Velocity	Angle
Backhand Push-3	+0.13 %	-0.16 %	+0.00995 m/s	+0.097 deg
Backhand Push-4	+0.08 %	-0.04 %	+0.00502 m/s	+0.027 deg
Backhand Push-5	+0.16 %	-0.11 %	+0.01038 m/s	+0.072 deg
Backhand Push-6	+0.09 %	-0.17 %	+0.00558 m/s	+0.051 deg
Backhand Push-7	+0.14 %	-0.07 %	+0.00755 m/s	+0.036 deg