UCLouvain

epl

École polytechnique de Louvain

# Ensemble Learning for Optimizing Adaptable Gesture Recognition

EAGER

Author: **Alexandre De Nys**
Supervisor: **Jean Vanderdonckt**
Readers: **Arthur Sluÿters, Nuwan T. Attygalle, Quentin Cappart**
Academic year 2025–2026
Master [120] in Computer Science

**Abstract**

Template-based gesture recognition remains a widely used approach due to its simplicity, low computational cost, and absence of a training phase. However, individual recognizers often exhibit uneven performance across gesture classes and datasets, which limits their robustness. Ensemble learning offers a potential solution by combining multiple recognizers in order to exploit complementary strengths and mitigate individual weaknesses.

This thesis investigates whether ensemble learning can improve recognition accuracy and robustness when applied to template-based gesture recognizers. Several ensemble strategies are implemented and evaluated within the QuantumLeap framework, including voting-based methods, stacking with a meta-learner, and a Dirichlet-based weighted averaging approach. The study focuses on three datasets of increasing difficulty: SHREC2019, 3DTCGS, and Kinemic, allowing ensemble behavior to be analyzed under diverse conditions.

Experimental results show that ensemble learning consistently outperforms the average accuracy of the composing recognizers, with only rare exceptions. Improvements are particularly significant on challenging datasets where individual recognizers perform poorly and leave more room for enhancement. However, ensembles do not systematically surpass the best individual recognizer, especially when performance is already close to a ceiling. A strong relationship is observed between ensemble effectiveness and diversity among recognizers, measured through variance in their predictions.

Among the evaluated approaches, the Dirichlet-based ensemble with statistically derived weights achieves the highest peak performance and the most stable improvements. These findings demonstrate that ensemble learning is a valuable strategy for template-based gesture recognition, particularly when robustness and consistency are prioritized over marginal gains beyond the best individual recognizer.

# Acknowledgments

A Master's thesis is never the work of a single person. For this reason, I would like to sincerely thank the many individuals whose support and contributions made this work possible.

First and foremost, I am deeply grateful to my supervisor, Jean Vanderdonckt, for guiding me throughout this thesis in the field of gesture recognition. I thank him for his availability, his reactivity to my questions, and his clear and insightful explanations on numerous topics during the entire duration of this work.

I would also like to thank Arthur Sluÿters, who helped me understand and use the QuantumLeap framework and kindly agreed to act as a reader for this thesis, as well as Sarah Steeman, who provided explanations regarding the dataset she recorded.

I would like to extend my thanks to my other readers, Nuwan T. Attygalle and Quentin Cappart, who kindly agreed to serve as members of my jury.

Finally, I would like to thank my family and friends for their continuous support throughout this period, especially during the final stretch of the thesis. During the many days and evenings spent working in my room, I could always rely on their encouragement and moral support.

I would also like to mention that ChatGPT was used during this thesis for text editing and assistance with code documentation.

# Contents

# Chapter 1

# Introduction

## 1.1 Context of the Problem

In recent years, motion-tracking technologies have become significantly more accessible. At the same time, visual display systems have grown larger while their prices have decreased. Together, these evolutions make mid-air gestures a more practical alternative to traditional contact-based input devices such as remotes or keyboards. Despite this increasing accessibility, further research is still needed to improve the accuracy of these systems in order to avoid errors in gesture detection.

As a result of this improved accessibility, gesture recognition has found a growing number of applications throughout its development, including human–computer interaction (HCI), sign language recognition and translation, healthcare applications, and security or authentication systems [23,32]. These application domains highlight both the potential impact of gesture-based interaction and the importance of reliable recognition performance.

At the same time, a wide range of sensing technologies for gesture recognition has emerged. Among them, the Leap Motion Controller (LMC), released in 2012, records user gestures through camera-based vision and remains one of the most commonly used hand-tracking sensors in the literature [2,5,25,26,30,31,45,53,54]. The device is typically placed in front of the user, facing upwards or forwards, to track hand movements such as shown on 1.1. However, one of the main limitations of this static sensor is its sensitivity to occlusion, and in particular self-occlusion, where parts of the hand may hide crucial information. The LMC cannot see through objects because it relies on an infrared depth camera, similar to the technology used in the Microsoft Kinect, a mainstream consumer device familiar to a broad audience. But this technology has also been used more recently, in virtual reality devices such as the Apple Vision Pro and the Meta Quest 3, which also rely on camera-based and infrared tracking.

While LiDAR-based systems could reduce occlusion-related issues, they remain expensive and, since affordability is an important aspect of gesture-based interaction, such solutions are less widespread than the LMC. Other hand-tracking devices based on contact sensing [32], such as wristbands [19], smart rings [47], ElectroMyoGraphy (EMG) sensors [37], and wired gloves [17], also exist but have not achieved the same level of user adoption. Controllers such as the Wii Remote are also available; however, they rely on physical handheld devices, which contradicts one of the main advantages of gesture recognition, namely hands-free and more natural interaction.

Recent literature largely reports end-to-end systems built around a single primary classification model (e.g., SVM/CNN/KNN), where a single model is responsible for predicting the class of an input gesture, while comparatively fewer works emphasize combining multiple

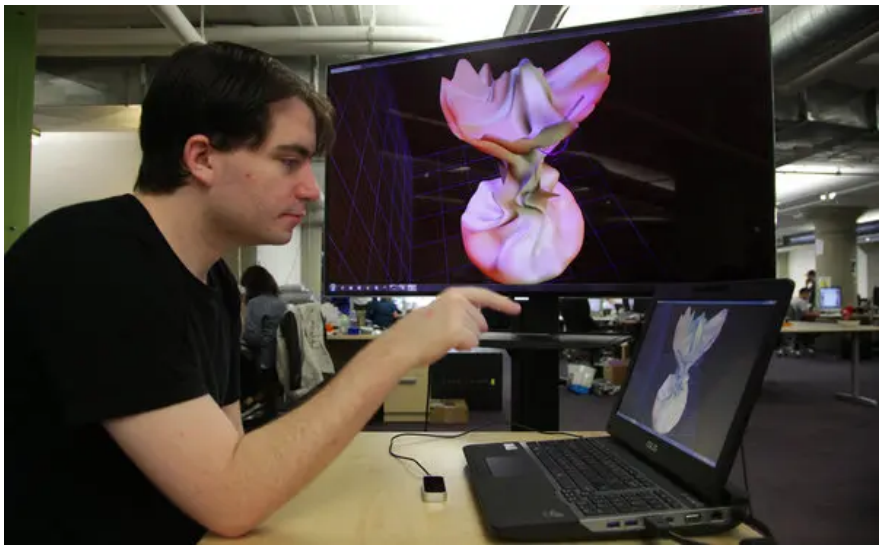

Figure 1.1: A user interaction with an app using the LMC. [4]

recognizers [34]. While this approach is effective for a limited number of gesture classes, a noticeable decrease in precision often appears as the number of classes increases. In practice, a recognizer may achieve high performance for some gesture classes (such as above 90% in the case of [32], and occasionally approaching 95%), yet exhibit lower precision for other classes, where it drops below the 90% threshold. This variability suggests that a single recognizer does not perform uniformly across all gesture classes.

## 1.2 Mission Statement

### 1.2.1 Statement

The objective of this master's thesis is to address the following research question: **Can ensemble learning increase recognition accuracy on established gesture-recognition benchmark datasets when using commonly used template-based recognizers?**

Ensemble learning is a machine learning approach that combines multiple models in order to improve the robustness and accuracy of the final prediction. It has notably been used to address issues related to limited data and model instability [33].

In the context of gesture recognition, recognizers [9,22,49,52] are models that predict the class of a candidate gesture. This concept is closely related to classification in machine learning, where a classifier assigns a label to unseen data. In gesture recognition, the unseen data corresponds to the candidate gesture, and the classifier takes the form of a recognizer.

Template-based recognizers form a specific subgroup within gesture recognition techniques. More generally, recognizers can belong to several families, such as Convolutional Neural Networks (CNNs), Hidden Markov Models (HMMs), decision trees, but also template-based approaches. Template-based matching recognition relies on comparing the candidate gesture against a set of stored gesture templates. This process typically consists of three main steps, illustrated in Figure 1.2:

- **Preprocessing**, which includes operations such as normalization, resampling, and rotation [46], in order to ensure that candidate gestures are comparable to the stored templates.

- **Comparing**, where the preprocessed candidate gesture is compared with each template in the dataset to compute dissimilarity measures.
- **Final processing**, which determines the predicted gesture class based on the comparison results.

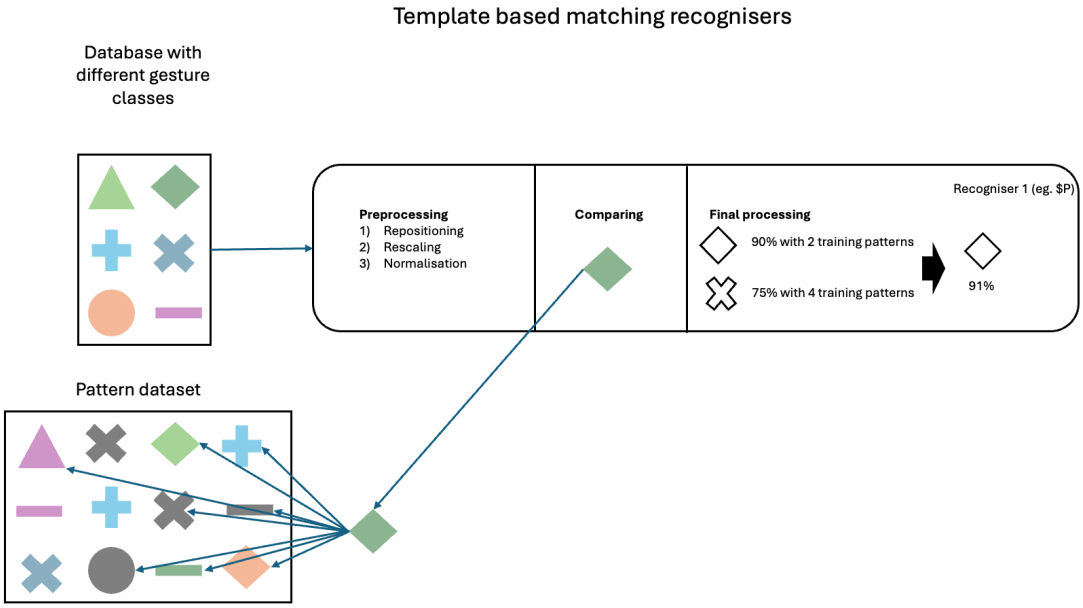


Figure 1.2: Flow of a template-based recognizer.

### 1.2.2 Working Hypotheses

In this thesis, the following working hypotheses are considered:

**H1.** *Environment*: This thesis relies on the QuantumLeap framework [42], developed by Arthur Sluÿters. As the framework already provides a structured architecture, this choice reduces development time and allows the work to focus on implementing ensemble recognizers and gathering data.

**H2.** *Datasets*: The datasets analyzed consist of commonly used benchmarking datasets in gesture recognition, as well as an internally developed dataset at UCLouvain that exhibits relatively low recognition accuracy.

**H3.** *Recognizers*: The objective is to combine several recognizers **without** modifying their internal mechanisms. Therefore, ensemble learning is explored, while fusion techniques are discussed but not directly applied.

**H4.** *Optimization objective*: Since recognizers typically produce predictions in a short time, this thesis prioritizes improving recognition accuracy, even at the cost of increased execution time.

This thesis makes the following contributions:

- **Ensemble implementations in QuantumLeap.** We implemented three families of ensemble methods for template-based gesture recognition within the QuantumLeap framework: (i) voting-based ensemble, (ii) Dirichlet-based score aggregation, and (iii) stacking with a meta-learner.
- **Dataset loader extensions.** To support our experimental study, we implemented and integrated two dataset loaders within QuantumLeap for two of the three datasets used in this thesis, enabling these datasets to be evaluated through the same execution and result-export mechanisms as the existing loaders.
- **Large-scale benchmark and cross-dataset analysis.** We conducted a benchmark of 372 experimental configurations across three datasets of varying difficulty, and we analyzed cross-dataset trends in accuracy and recognition time as a function of the template budget.

### 1.2.3 Research Method

This section outlines the research methodology adopted in this thesis. The work is structured around two main components.

#### State of the Art

To address the research question, the first step consists of reviewing the existing literature in gesture recognition and ensemble learning. This includes explain how ensemble methods operate and how they can be adapted or applied to the specific case of template-based gesture recognition.

#### Benchmarking

The second part focuses on benchmarking different model combination strategies. Various ensemble configurations are evaluated and analyzed in order to assess their impact on recognition performance. Because gesture recognition already involves many tunable parameters—and ensemble learning introduces additional ones through the choice and configuration of individual recognizers, the primary goal is to explore and gain insight into ensemble learning techniques applied to gesture recognition, rather than to exhaustively optimize parameters to achieve maximal accuracy.

## 1.3 Overview of the Thesis

This thesis is organized as follows.

Chapter 2 presents the state of the art in ensemble learning and gesture recognition. It introduces the fundamental principles of ensemble methods, including their main categories, advantages, and limitations. Existing work applying ensemble learning to gesture recognition is reviewed, with particular attention to approaches relevant to template-based recognizers. Fusion techniques are also discussed to clarify their conceptual relationship with ensemble learning. Finally, the chapter introduces the main families of template-based gesture recognizers and highlights their similarities and differences.

Chapter 3 describes the experimental setup used to evaluate ensemble learning for gesture recognition. It details the three datasets employed in this work—SHREC2019, 3DTCGS, and Kinemic—and explains the rationale behind their selection. The chapter then presents the

ensemble learning strategies implemented within the QuantumLeap framework, including voting-based methods, stacking, and Dirichlet-based weighted averaging. The necessary framework adaptations, testing protocol, parameters, and evaluation metrics are also defined to ensure consistent and reproducible experiments.

Chapter 4 reports the experimental results. It first presents the performance of individual recognizers on each dataset, followed by the results obtained with the different ensemble configurations. Selected combinations are analyzed in detail in order to highlight representative and meaningful behaviors rather than exhaustively listing all tested configurations.

Chapter 5 provides an in-depth analysis of the results. It examines global trends across datasets, such as improvements over average accuracy, limitations in surpassing the best individual recognizer, the influence of template count, and the role of recognizer diversity. The chapter also analyzes ensemble behavior on poorly recognized gesture classes and discusses the impact of ensemble size and recognizer complementarity. Finally, the specific characteristics of voting-based, stacking, and Dirichlet-based ensembles are compared.

Chapter 6 discusses the limitations of the present work and outlines potential directions for future research, including alternative ensemble strategies, additional recognizer combinations, and possible extensions involving fusion techniques.

Chapter 7 concludes the thesis by summarizing the main findings and answering the research question.

# Chapter 2

# State of the Art

Chapter 1 introduced gesture recognition as well as the research question addressed in this thesis. Before evaluating whether ensemble learning can improve the accuracy of our specific project, this chapter first provides a detailed description of ensemble learning and fusion. It presents the main categories of ensemble methods and illustrates them with representative examples. It then analyzes several published studies that apply ensemble learning to gesture recognition, discussing how this methodology influences recognition accuracy. Afterwards, it reviews how fusion has been used in the literature and outlines how it could benefit the problem addressed in this thesis. Finally, since ensemble models rely on multiple recognizers, this chapter further examines how a recognizer operates and highlights the similarities and differences between various recognizer families.

## 2.1 Overview of Ensemble Learning

### 2.1.1 Definition

Ensemble learning is a machine learning technique that combines multiple models to obtain a more accurate and robust final prediction. Instead of relying on a single model, ensemble learning aggregates the outputs of several base learners to improve generalization and reduce the risk of errors specific to individual models [33].

### 2.1.2 Advantages and Limitations

As mentioned earlier, the primary objective of ensemble learning is to enhance the overall predictive accuracy of a system. This improvement is achieved by combining the outputs of several models and leveraging their collective knowledge to produce a final, more reliable decision.

One of the key strengths of ensemble learning lies in its ability to address the bias–variance trade-off. The bias represents the error between the predicted and the true outputs: a lower bias indicates that the model is well optimized for the underlying patterns. Conversely, the variance measures how much the model's predictions fluctuate when trained on different datasets or under varying random conditions. A model with high variance tends to overfit the training data, leading to unstable predictions.

The total prediction error of a model can be expressed by Equation 2.1 [13]. :

$$\text{Error} = \text{Bias}^2 + \text{Variance} + \text{Irreducible Error} \tag{2.1}$$

This equation shows that a model's total error originates from both bias and variance, along with an unavoidable component known as the irreducible error. Ensemble methods help mitigate bias and variance by combining multiple models whose individual errors may compensate for one another, leading to better overall generalization.

Furthermore, research has demonstrated that the diversity among models plays a crucial role in reducing prediction errors [6]. The more diverse the base learners are, the more effectively they can complement each other's weaknesses. This property is at the heart of ensemble learning, where diversity and model combination typically result in superior performance compared to individual models.

However, ensemble methods also introduce certain drawbacks. Their main limitation is the increased computational cost and memory usage due to training multiple base models, as seen with Savaş and Ergüzen [41] with the time-per-step results obtained using base models and subsequently with ensemble learning. An increase in time per step occurs when using the latter method.

Moreover, if the ensemble's final decision results from only minor differences among its base models, it indicates a lack of diversity within the ensemble. In such cases, the combined prediction becomes less reliable, and overall performance tends to degrade [39].

## 2.2 Main Categories of Ensemble Methods

In the field of ensemble learning, several categories exist. The three main categories of ensemble techniques are *bagging*, *stacking*, and *boosting*, which will be developed in this section. The most representative and widely used techniques are presented as examples. These categories are among the most widely used and collectively illustrate key distinctions between sequential vs. parallel and homogeneous vs. heterogeneous ensemble methods which will be explained in this section. However, other approaches also exist, such as blending, weighted-average ensembles among others.

### 2.2.1 Bagging-Based Methods

Bagging—short for *bootstrap aggregating*—is a homogeneous parallel method. It is homogeneous because all base learners share the same learning algorithm, and parallel because all base learners are built simultaneously, either on the same dataset or on different subsets of it. In this case, the latter applies, as each model is trained on a different dataset created from the original one through bootstrap sampling. In this process, new training sets are generated by randomly drawing samples from the original dataset *with replacement*, meaning some samples may appear several times while others may be omitted. Each resampled dataset is used to train a distinct base model, and their predictions are then aggregated—typically through averaging or majority voting—to produce the final output.

A well-known example of a bagging method is the *Random Forest* algorithm. Random Forests apply the bagging principle using decision trees as base learners. Each decision tree is trained on a different bootstrap sample of the original dataset and on a random subset of features at each split. This randomization increases diversity among the trees and improves the model's generalization compared to a single decision tree.

### 2.2.2 Stacking

Stacking is a heterogeneous parallel method. Unlike bagging, which increases diversity by varying the training datasets of identical models, stacking diversifies its ensemble by combining different learning algorithms as base learners.

Once these various base learners have been trained, their predictions on unseen data are collected and used as inputs for a second-level model called the *meta-learner*. The meta-learner learns how to best combine the outputs of the base models to produce the final prediction.

This meta-learner does not need to be a different model from the base learners, but it is also not required to be the same one. The data provided to the meta-learner can take several forms; in multiclass classification, for example, it can be the final predicted class labels or the full probability score vectors.

### 2.2.3 Boosting-Based Methods

Finally, boosting is a sequential ensemble method, as this approach trains each base model iteratively, where every new model is built based on the performance of the previous one. In this approach, each new learner attempting to correct the errors made by its predecessor. The process begins with the creation of an initial base learner, which is typically weak and achieves relatively low accuracy. Similar to bagging, boosting also relies on sampling the dataset; however, the sampling is not random. Instead, it emphasizes the instances that were misclassified by the previous model so that the next learner can focus on these difficult cases. This procedure is repeated iteratively for as many base learners as desired. Once all learners have been trained, their predictions are weighted and combined to produce the final output.

The key difference between various boosting algorithms lies in how they prioritize the errors of previous learners. Two of the most widely used techniques illustrate this distinction well. In *Adaptive Boosting (AdaBoost)*, the algorithm assigns higher weights to the samples that were incorrectly predicted, ensuring that subsequent learners pay more attention to them. In contrast, *Gradient Boosting* relies on the concept of residual errors: it computes the difference between the predicted and true values and uses these residuals as new targets for the next learner, progressively reducing the overall error.

## 2.3 Applications of Ensemble Learning to Gesture Recognition

Having introduced the fundamental principles of ensemble learning, we now turn to the literature to examine how this approach has been applied in the field of gesture recognition and what improvements it has brought in practice. In this section, we analyze studies from various domains of gesture recognition that employ different types of data. As a reminder, our project based on the QuantumLeap framework focuses on vision-based 3D hand-tracking data and wearable inertial wrist-motion data, such as orientation and rotation, which correspond to the third dataset presented in Section 3.

### 2.3.1 Bagging-style application

Peng *et al.* [37] addressed the instability often observed in single Extreme Learning Machines (ELMs) [51] when classifying surface electromyography (sEMG) signals, which are muscle gestures recognized by placing electrodes on the skin to measure electrical activity. They first

applied a preprocessing step to extract and select the sixteen most relevant features from the signals before training their models. The ensemble they used combined the predictions of several ELMs through majority voting under five-fold cross-validation, forming a bagging-like parallel ensemble but doesn't include the bootstrap resampling. In the three experimental exercises conducted, the ensemble learning method consistently achieved between 1% and 2% higher accuracy than the best-performing individual model that did not use ensemble learning. This demonstrates that ensemble learning improved both accuracy and stability compared to single ELMs.

### 2.3.2 Weighted average application

Savaş and Ergüzen [41] analyzed how ensemble learning can improve robustness in vision-based hand gesture recognition. Using the HG14 dataset, which contains 14,000 RGB images representing 14 gesture classes performed by 17 participants, they compared 22 pre-trained convolutional neural networks and selected the four best—VGG16, VGG19, MobileNet, and MobileNetV2—for fine-tuning. Each model was then individually trained.

They proposed a *Dirichlet Ensemble* that combines these four pre-trained networks through adaptive probabilistic weighting instead of simple averaging. The weights, optimized on the validation set, highlight the most reliable models and help capture uncertainty among predictions. The ensemble achieved a 98.88% test accuracy—more than 2 percentage points higher than the best single model (MobileNet, 96.79%)—and produced consistent results across multiple runs.

Although the method demonstrated strong accuracy and stability, it also showed slightly lower validation accuracy, indicating mild overfitting, and required substantial computational resources due to the multiple deep models involved. Moreover, because the HG14 dataset contains only static images, temporal gestures were not considered.

Overall, this heterogeneous parallel ensemble—conceptually related to bagging—demonstrates that combining different CNN architectures with adaptive weighting can enhance reliability in vision-based gesture recognition, offering insights directly relevant to the 3D hand-tracking work conducted with the QuantumLeap framework.

## 2.4 Overview of Fusion

### 2.4.1 Definition

Fusion, like ensemble learning, aims to improve accuracy and robustness by combining information. However, unlike ensemble learning, which aggregates the outputs of multiple models operating on the same data source, fusion combines information originating from different sources or modalities within the recognition process.

### 2.4.2 Advantages and Limitations

One of the main advantages of fusion is its ability to reduce ambiguity in recognition tasks. A gesture may appear ambiguous or difficult to distinguish when observed through a single modality, whereas combining information from multiple modalities can provide complementary cues and lead to more reliable predictions. This approach is particularly well established in the context of autonomous vehicles, where relying on a single modality is considered insufficient. Consequently, fusion is commonly applied to combine vision, radar, and LiDAR data in this context [1,24].

In the field of gesture recognition, several fusion strategies have also been explored. These include fusion between similar modalities, such as using two Leap Motion Controllers positioned differently—one facing upwards and the other facing forwards [18,38]. Fusion has also been applied across heterogeneous modalities, for example by combining vision-based data with electromyography (EMG) signals [10].

Despite these advantages, fusion also introduces several limitations. First, it typically requires additional hardware, which increases the overall system cost. Second, multiple sensors must be carefully calibrated to operate together, and even minor changes in the experimental setup may require recalibration. Finally, as with ensemble learning, fusion increases computational complexity: combining multiple data sources results in larger data volumes, additional preprocessing steps, and more complex decision-making processes.

## 2.5 Types of Fusion

Information fusion can occur at different stages of the gesture recognition pipeline. Three main types of fusion are commonly distinguished and are discussed in this section: *early fusion*, *mid fusion* (also referred to as intermediate fusion), and *late fusion*.

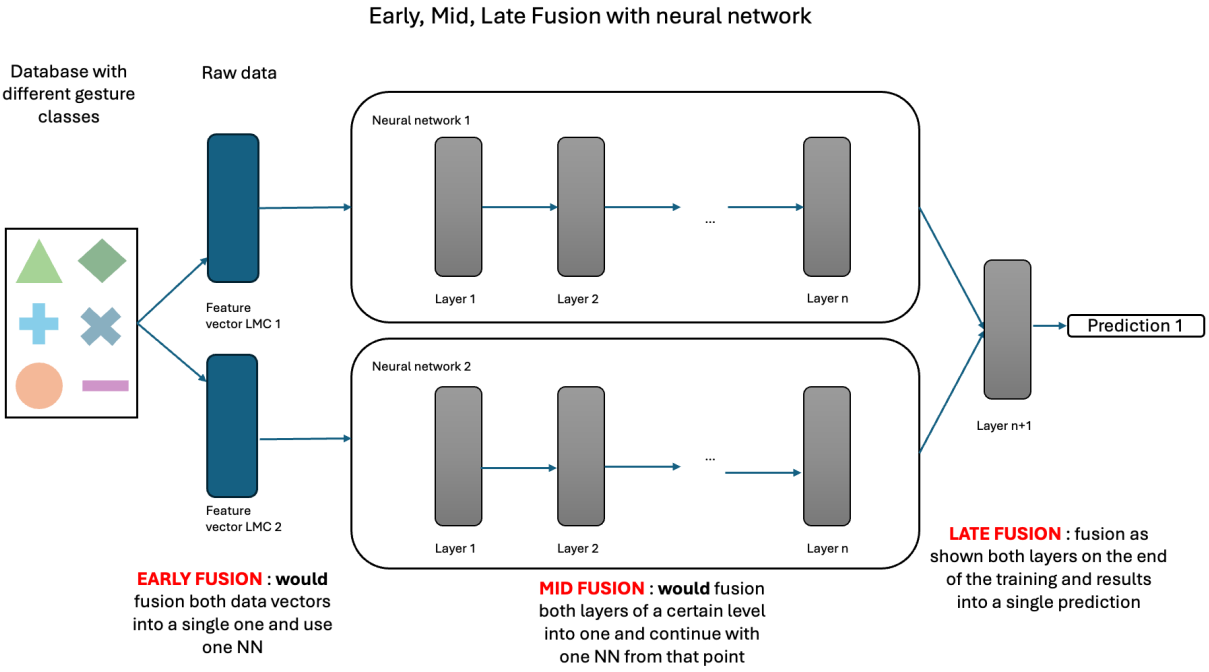


Figure 2.1: Different types of fusion using neural networks.

### 2.5.1 Early Fusion

Early fusion takes place at the very beginning of the recognition process. Instead of training multiple models on separate raw data sources, the raw data from different modalities are first combined and then provided to a single model. This allows the model to capture correlations between modalities at an early stage of processing.

However, because the fusion occurs before any high-level feature extraction, the data to be fused can be highly heterogeneous. As a result, preprocessing becomes more complex, and changes in the set of modalities often require redesigning the fusion strategy [1].

### 2.5.2 Mid Fusion

Mid fusion occurs at an intermediate stage of the recognition pipeline. In this case, data from different modalities are first processed independently by separate models or feature extractors. The fusion then takes place within the recognizer itself, for instance by merging feature representations at a specific layer in a neural network.

This approach offers increased modularity, as individual feature extractors can be adjusted or replaced independently. Nevertheless, it still requires compatible feature representations, and access to internal model features is not always possible, which can limit its applicability.

### 2.5.3 Late Fusion

Late fusion operates at the final decision stage of the recognition process. The outputs of separate models—each typically associated with a different modality—are combined to produce a final prediction. This approach is closely related to ensemble learning, with the key difference that late fusion combines models trained on different modalities, whereas ensemble learning generally combines models trained on the same modality.

Late fusion is often considered the easiest fusion strategy to apply [1], as it does not require access to internal model features or the handling of heterogeneous raw data. Only the final predictions need to be combined. On the other hand, since fusion occurs only after each model has produced its output, the data are not jointly analyzed from the beginning of the process. This may prevent the exploitation of certain cross-modal relationships that early or mid fusion could capture.

## 2.6 Recognizers

This section presents the six individual recognizers used in this thesis: $P3+, $Q3, 3Cent, Jackknife, $\mu$V, and $\mu$F. These recognizers were selected because, as mentioned in the working hypotheses, the QuantumLeap framework is used as the experimental environment, and these recognizers are already implemented within this framework. One additional recognizer, $1, is also introduced, as it is often considered the "father" of gesture recognizers.

The recognizers are grouped in this section to better highlight the similarities and differences between them, which is a key aspect when applying ensemble learning, as discussed in Section 2.1.

All these recognizers are template-based algorithms. This means that they assign a class label (corresponding to a gesture name) to a candidate gesture by comparing it to a set of labeled templates stored in the dataset. In contrast, neural network–based approaches require a dedicated training phase. Template-based recognizers only rely on a dataset of gestures from which distances are computed. As a result, when new gestures are added to the dataset, there is no need to retrain the entire model from scratch.

### 2.6.1 $1

As stated earlier, the $1 recognizer [52] is not used in any of the ensemble learning models considered in this thesis. However, since it inspired many subsequent approaches [7], it is presented here as a starting point to understand the foundations of gesture recognition.

The $1 algorithm is a 2D uni-stroke gesture recognizer. Uni-stroke gestures correspond to continuous paths traced by a finger or palm to create a gesture in the air. The recognizer relies

on basic geometry and trigonometry. As illustrated in Figure 1.2, the recognition process is divided into three main phases.

The first phase is preprocessing. In this step, gestures are resampled to a fixed number of tracked points $N$, resulting in an ordered sequence of equidistant points, as shown in Figure 2.2. These points are then rotated by the angle formed between the centroid of all points and the first point of the gesture. Next, the points are translated so that the centroid is positioned at the origin $(0, 0)$. As a final preprocessing step, the gesture is scaled in a non-uniform manner [28].

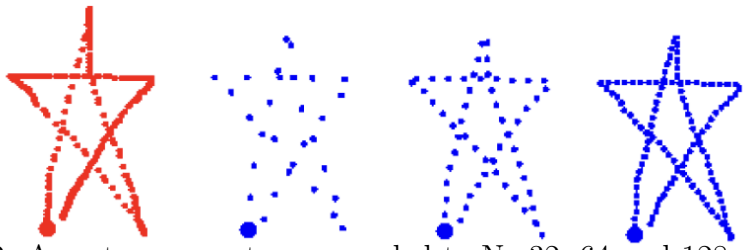

Figure 2.2: A gesture as a star resampled to N=32, 64 and 128 points [52]

After preprocessing, the algorithm compares the candidate gesture to each stored template in the dataset. This comparison is performed by computing the Euclidean distance between each pair of corresponding points from the candidate gesture and the template gesture. By summing the distances over all points, a dissimilarity measure is obtained, which allows the similarity between gestures to be assessed [28]. This value is then converted into a similarity score in the range $[0, 1]$.

Finally, during the decision phase, the $1 recognizer assigns a class label to the candidate gesture. The predicted class corresponds to the template gesture for which the sum of point-wise distances is minimal, making the algorithm both simple and computationally efficient.

### 2.6.2 Point-Cloud Extensions

Researchers have identified several limitations of the $1 recognizer. Since it treats multi-stroke gestures as uni-stroke gestures, it suffers from increased memory usage and execution costs [3]. In addition, users may not perform gestures using the same stroke order or direction. For example, a rectangle can be drawn clockwise starting from the bottom-left corner or counterclockwise starting from the middle of the top edge. The core issue lies in the use of ordered points, which impose a predefined order both on the strokes and on the points within each stroke.

To address these limitations, Vatavu *et al.* [49] proposed the $P recognizer, a multi-stroke gesture recognizer. $P overcomes the ordering problem by representing gestures as unordered point clouds. While the preprocessing stage remains similar to that of $1, the comparison phase differs significantly. Instead of matching points sequentially, each point is matched with the closest unmatched point in the other cloud. Since the number of available unmatched points decreases over time, each match is assigned a weight, with earlier matches receiving higher importance. The final decision process follows the same principle as in $1.

This approach led to the development of $P3+ [35], the first recognizer used in the ensemble models of this thesis. $P3+ is the 3D extension of $P+ [48] and introduces two additional modifications. While the preprocessing stage remains unchanged, the comparison phase matches each point in the first cloud to the closest point in the second cloud, even if that point has already been matched. As a consequence, the weighting scheme becomes unnecessary and is therefore removed [28].

The second recognizer in this point-cloud-based group is $Q3 [35], which is the 3D extension of the $Q recognizer [50]. $Q is a uni- and multi-stroke recognizer derived from $P and was

developed to reduce computational complexity. Several optimizations are introduced compared to $P. For instance, instead of computing the Euclidean distance—which requires a square root operation—$Q relies on the squared Euclidean distance, which preserves relative distances while reducing computation. In addition, template comparisons are abandoned as soon as the accumulated distance exceeds the current best match. These optimizations result in a reported speedup of up to 46 times compared to $P, with little to no loss in recognition accuracy.

### 2.6.3 Distortion Importance

The 3Cent recognizer [9] is also inspired by the $1 recognizer. 3Cent is a 3D uni-stroke gesture recognizer specifically designed to account for rotational variations in gestures. The authors observed that the rotation of a gesture performed by the user often has a significant impact on recognition performance. Consequently, during preprocessing, 3Cent omits the rotation normalization step. Instead, it directly computes the sum of squared distances without compensating for rotation, allowing rotational differences to be reflected in the distance computation.

Scaling and recentering are still applied during preprocessing. In addition, the gesture is scaled to a fixed length, equal to 1, using a simplified scaling procedure. This choice helps avoid shape distortion during rescaling while maintaining sensitivity to rotational differences.

### 2.6.4 Vector-Based Recognizers

This section covers the Jackknife recognizer and the $\mu$ family of recognizers.

The Jackknife recognizer [22] differs from the $ family by placing a stronger emphasis on the temporal evolution of the gesture. It represents gestures using direction vectors of unit length and relies on Dynamic Time Warping (DTW) to compute dissimilarity between the candidate gesture and stored templates. DTW enables alignment of time series with different execution speeds. Based on this vector-based comparison, Jackknife does not require explicit preprocessing. The comparison relies on the inner product of gesture path direction vectors, and the resulting score is normalized so that similar time series produce values close to one.

The $\mu$V recognizer [29] is inspired by multiple previous approaches. It combines the cloud-based matching technique of $P+ with the local shape description of the !FTL algorithm developed by Vanderdonckt [46]. In this approach, local shape distances—computed from the geometric relations between consecutive vectors and points—are used to estimate the dissimilarity between the candidate gesture and each gesture class. The sum of these local distances produces a final dissimilarity score, and the class associated with the lowest score is assigned to the candidate gesture.

Finally, the $\mu$F recognizer [27] builds upon the $\mu$V approach by extending it into a multi-feature recognizer. Instead of relying solely on local shape distances, $\mu$F also incorporates vector-based distances and global shape variations. These different dissimilarity measures are combined through a weighted aggregation scheme, allowing the recognizer to jointly exploit local and global characteristics of the gesture.

## 2.7 QuantumLeap

Finally, we describe the QuantumLeap framework, which is used throughout this thesis for all experiments and evaluations.

QuantumLeap was originally developed in the context of Arthur Sluÿters' Master's thesis [42] and has since evolved through continuous maintenance by the same author over several years.

The main objective of the framework is to provide a unified environment for gesture recognition applications using the Leap Motion Controller (LMC), allowing developers to reuse a large part of the processing pipeline while only reimplementing a limited number of components.

The framework standardizes raw sensor data into frames and already includes several template-based gesture recognizers, allowing the entire gesture recognition pipeline to be processed, as illustrated in Figure 2.3. Developers can easily swap or customize specific modules such as data loaders, segmenters, or recognizers, depending on the application or dataset being used. This modular design makes QuantumLeap particularly suitable for comparative studies and rapid experimentation.

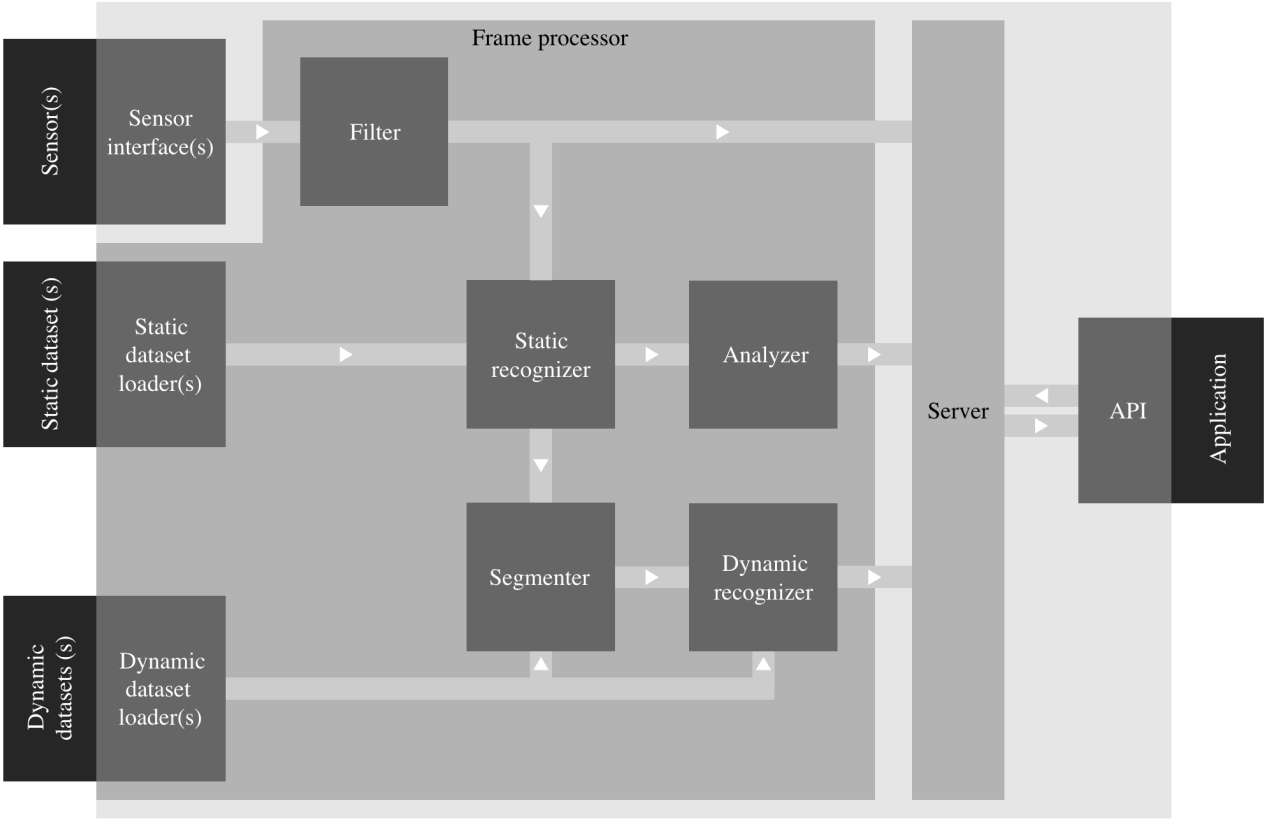


Figure 2.3: Frame Processor of quantumLeap.

QuantumLeap provides two main core functionalities. The first one is real-time gesture detection, where gestures are captured through the Leap Motion Controller and recognized on-the-fly. The second functionality is a dedicated testing environment, which allows systematic evaluation of recognizers by varying experimental parameters such as the number of templates, the tracked points, or the evaluation protocol.

Several evaluation strategies are supported by the framework. The primary one used in this thesis is the `TTS` (Train-Test Split) protocol, where in the case of QuantumLeap one single sample is selected as the test instance while all remaining samples are used as templates for training. This process is repeated multiple times to obtain statistically meaningful results. Preliminary developments toward cross-dataset evaluation also exist within the framework, enabling experiments where one dataset is used for training and another for testing, although this functionality is not fully exploited in this work.

## 2.8 Summary

This review of the literature provided essential insights into ensemble learning and its application to gesture recognition. In particular, previous studies report modest but consistent accuracy improvements, ranging from gains of 1–2% [37] to more stable improvements of approximately 2% [41]. Beyond these quantitative results, the literature highlights important conditions and design choices that influence the effectiveness of ensemble learning approaches.

In order to properly assess these methods, it was necessary to develop a detailed understanding of template-based gesture recognizers and to identify the key differences between them. Several gesture recognition use cases were analyzed to examine how ensemble learning has been applied in practice and to understand how these approaches must be adapted when working specifically with template-based recognizers.

Fusion techniques were also discussed as a complementary strategy. Both fusion and ensemble learning share the objective of combining multiple elements to improve recognition accuracy and robustness. However, they differ in their implementation: fusion focuses on combining information from different modalities or sensors, whereas ensemble learning combines the outputs of multiple recognizers operating on the same modality. Despite these differences, both approaches introduce additional computational costs, which must be taken into account when designing gesture recognition systems.

# Chapter 3

# Method

This chapter defines the experimental setup used to evaluate whether ensemble methods improve recognition accuracy. It introduces the different methodological and implementation choices made for the evaluation. In particular, it details how the various datasets are composed and explains the rationale behind the selected ensemble learning strategies. The chapter also describes the modifications required within the framework to support these approaches. Finally, it outlines the testing procedure, providing an overview of how the experiments are conducted and evaluated.

## 3.1 Datasets

This section presents the three datasets used in this thesis. Two of them, SHREC2019 and 3DTCGS, are the same datasets used in the thesis of Arthur Sluÿters [42], which allows a direct comparison between our results and those reported in Chapter 4. The third dataset, Kinemic, is an additional dataset introduced to extend the experimental evaluation. All three datasets consist of dynamic 3D gestures, which is consistent with the fact that all the gesture recognizers considered in this work operate on 3D data.

### 3.1.1 SHREC2019

SHREC2019 is a dataset created for the Eurographics 2019 Shape Retrieval Contest by Caputo *et al.* [8]. To generate the dataset, the authors designed a 3D virtual environment in which users interacted using mid-air hand movements. SHREC2019 has since become a widely used benchmark for gesture recognition evaluation [8,12,16,20,35,36,42].

The dataset contains five gestures, illustrated in Figure 3.1: cross, V-mark, caret, square, and circle. Each gesture corresponds to a 3D trajectory, although the underlying shapes are essentially planar (2D) gestures embedded in 3D space. The gestures were performed by 13 different subjects, each repeating the full gesture set three times, resulting in a total of 195 recordings. All recordings were captured using a Leap Motion Controller. No finger articulation is involved; the gestures are represented purely as hand trajectories.

The dataset itself is composed of sequences of 3D points corresponding to tracked hand features (e.g., palm position or fingertip positions) over time. Although SHREC2019 was already analyzed in the thesis of Arthur Sluÿters [42], no dedicated data loader was available in the latest version of the QuantumLeap framework [43]. As a result, a specific loader had to be implemented. In addition, the original dataset structure differs from the one expected by QuantumLeap. SHREC2019 is organized by user folders, each containing the five gestures,

whereas QuantumLeap expects a structure organized by gesture, with each gesture folder containing user subfolders and repetitions. Moreover, although the dataset is described as consisting of 13 users with three repetitions each, it is stored as 39 users performing a single repetition.

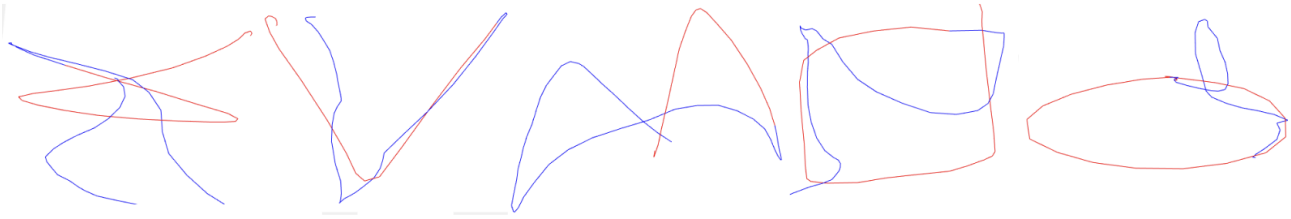

Figure 3.1: SHREC2019 3D gestures: two 2D views of the same 3D trajectory (red/blue) [8]

### 3.1.2 3DTCGS

The 3DTCGS dataset was also created by Caputo *et al.*, this time to evaluate the usability of the 3Cent recognizer [9]. Like SHREC2019, 3DTCGS contains 3D trajectories captured using a Leap Motion Controller. However, it significantly increases the complexity of the recognition task by including 26 different gesture classes, as illustrated in Figure 3.2.

One motivation for selecting this dataset is, again, the possibility of comparing our results with previously published work. In addition, the larger number of gesture classes addresses a key limitation of SHREC2019, which contains only five gestures. A dataset with 26 gesture classes presents a more challenging recognition problem and allows for a more comprehensive evaluation of ensemble methods.

The dataset consists of 221 recorded gestures performed by 14 subjects. In contrast to SHREC2019, 3DTCGS includes not only planar gestures represented as 3D trajectories, but also truly three-dimensional gestures such as polygonal chains (e.g., poly3Dxyz, poly3Dxzy, poly3Dyxz, poly3Dyzx, poly3Dzxy, poly3Dzyx) and 3D spirals.

As with SHREC2019, no loader was available to transform the dataset into the format expected by the QuantumLeap framework. Furthermore, the data in 3DTCGS is stored in CSV format, whereas QuantumLeap was developed for JSON files. Consequently, a dedicated loader was implemented to both restructure the dataset and convert the data into the appropriate format.

### 3.1.3 Kinemic Dataset

The Kinemic dataset was created by Chauvaux and Steelman [44] to study wrist-based gesture recognition. Data collection was performed using the Kinemic device [19], a wrist-worn bracelet equipped with inertial sensors. The device tracks wrist motion using an accelerometer and a gyroscope. For each time step, seven values are recorded: the $x$, $y$, and $z$ components of linear acceleration, the $x$, $y$, and $z$ components of angular velocity and orientation from the gyroscope, and a timestamp.

The dataset contains recordings from 30 participants, each performing 43 gesture classes twice, resulting in a total of 2580 recorded gestures. Among these 43 classes, 12 are predefined gestures introduced by [19], while the remaining 31 were designed by the dataset authors. The gestures are organized into several categories.

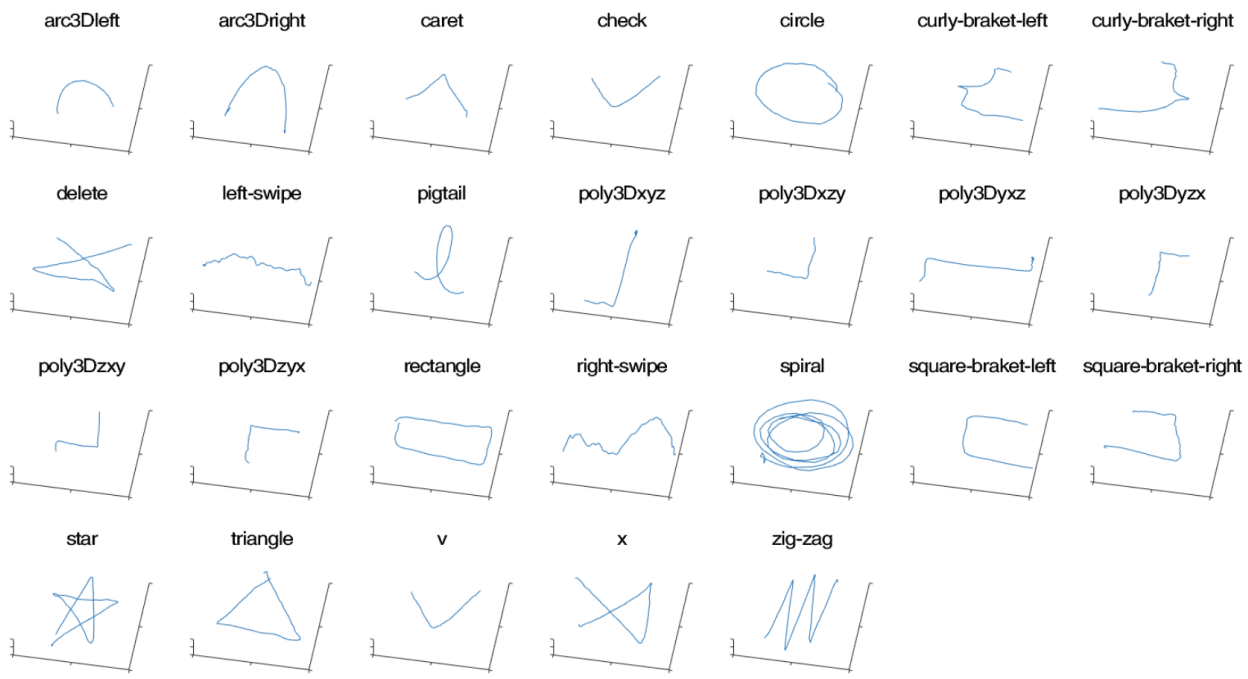


Figure 3.2: Examples of 3D gesture trajectories from the 3DTCGS dataset [9]

In this thesis, we focus exclusively on a subset of unimanual gestures that can be captured using a single wrist sensor. These gestures are the following 21; swipe up, swipe down, swipe left, swipe right, check mark, cross mark, circle clockwise, circle counter-clockwise, rotation right-left, rotation left-rigth, eartouch left, eartouch right, move forward, moveback-ward, hello, transfer, handshake, knock, mix, forward rotation and backward rotation. Bimanual gestures are excluded, as they require multiple sensors and therefore do not fit within the current QuantumLeap framework. This selection results in a final subset of 1260 recorded gestures used for evaluation.

Furthermore, the Kinemic dataset was originally developed to be used within the QuantumLeap framework, and a corresponding data loader was therefore already available. However, the structure and content of the JSON files differed from the expected input format of the existing recognizers. As a result, modifications had to be applied to the `convert` functions of each recognizer in order to correctly parse and process the dataset.

## 3.2 Ensemble Learning

This section presents the ensemble learning models investigated in this thesis. Three different ensemble strategies are introduced, each operating at distinct stages of the recognition process. The ensemble models considered are:

- majority and confidence-based voting, combined into a hybrid approach,
- a weighted aggregation using a Dirichlet distribution over similarity scores,
- a stacking approach using a decision tree as a meta-learner.

### 3.2.1 Voting-Based Ensembles

#### Majority Voting

Inspired by the majority voting strategy proposed by Peng *et al.* [37] and discussed in Section 2.3, this approach can be viewed as a bagging-like ensemble without bootstrap resampling. Each recognizer independently predicts a gesture class, and the class receiving the highest number of votes is assigned to the candidate gesture.

To implement this strategy, it was necessary to apply multiple individual recognizers to the same candidate gesture within a single ensemble model. This required extending the QuantumLeap framework, as such parallel evaluation of recognizers was not initially supported.

In practice, this approach proved to be limited in early experiments. When only two recognizers are used, majority voting becomes ineffective: either both recognizers predict the same class, yielding no benefit, or they disagree, resulting in a tie with no clear resolution. An initial tie-breaking strategy consisted in selecting the output of the first recognizer; however, this choice introduced bias and did not improve performance.

#### Confidence Voting

To address the limitations of majority voting, a confidence-based voting strategy was explored. Instead of counting votes, this method selects the class predicted by the recognizer that assigns the highest confidence score to its prediction. The confidence score corresponds to the similarity between the candidate gesture and the best-matching template.

This approach was evaluated independently before being combined with majority voting. However, early results were again unsatisfactory. This behavior can be explained by the fact that similarity scores are not directly comparable across recognizers, as each recognizer relies on a different distance formulation and normalization scheme, as described in Section 2.6.

#### Hybrid Voting

This hybrid approach limits the influence of confidence scores that are not directly comparable across recognizers. Indeed, each recognizer computes its confidence using a different distance formulation and normalization scheme, which makes raw confidence values difficult to compare. By applying confidence-based voting only in tie situations—where no clear majority emerges—the method reduces the impact of these incompatible confidence scales. Furthermore, in the absence of a strong consensus between recognizers, confidence differences tend to be more pronounced, making them more informative for resolving ambiguous cases.

#### Borda and Dowdall Voting

Following the introduction of the `recognizeAllSimilarities` method—originally implemented to support the Dirichlet-based ensemble described in the next subsection—each recognizer was extended to return similarity scores for all gesture classes rather than only the best match. This additional information enabled the exploration of rank-based voting strategies, namely Borda and Dowdall voting.

In Borda voting, each recognizer ranks all gesture classes according to their similarity scores. The worst-ranked class receives a score of 1, the second-worst a score of 2, and so on, up to the total number of classes. Scores are then summed across recognizers, and the class with the highest cumulative score is selected as the final prediction. Dowdall voting follows a similar

principle but assigns decreasing fractional weights (1, 1/2, 1/3, ...), giving more importance to top-ranked predictions.

Between the two, Borda voting was selected for further evaluation, as it provides a clearer distinction from majority voting, which only considers the top-ranked prediction, while still leveraging the full ranking information provided by each recognizer.

### 3.2.2 Dirichlet Approach

Inspired by the literature review and, in particular, by the results reported by Savaş and Ergüzen [41], a Dirichlet-based ensemble strategy was implemented. This approach was selected because it differs substantially from the previously introduced voting-based techniques and operates at a different stage of the recognition pipeline. Instead of relying solely on the final class prediction of each recognizer, it exploits the similarity scores associated with all gesture classes, while still demonstrating promising performance. As mentioned in the previous subsection, the Borda voting strategy was only introduced after the implementation of this approach, as it required the development of the `recognizeAllSimilarities` method.

The Dirichlet approach operates as follows: a weight is manually assigned to each recognizer to reflect its relative importance. These weights are then transformed using a Dirichlet distribution. Formally, as shown in Equation 3.1, $w_i$ denotes the weight associated with the $i$-th ensemble member, $\hat{y}_i$ represents the output of the $i$-th recognizer, and $\hat{Y}$ corresponds to the aggregated ensemble output.

The advantage of using a Dirichlet distribution rather than a simple weighted average lies in its balancing effect on the weights. When the assigned weights are very close to one another, the Dirichlet distribution helps accentuate subtle differences between recognizers. Conversely, when the weights differ significantly, it moderates extreme disparities by adjusting them proportionally to their relative differences.

Unlike the hybrid voting approach, which operates solely on the final class predictions, the Dirichlet-based ensemble must be applied between the comparison and final decision stages of the recognition process. To enable this, the QuantumLeap framework was adapted so that recognizers no longer retain only the best similarity score and its associated class. Instead, they store the similarity scores for all gesture classes. Importantly, this modification does not alter the internal comparison mechanisms of the recognizers: each recognizer still performs the same template-to-candidate comparisons, but an additional method returns all class-level similarity scores rather than a single best match.

Once these adaptations were in place, the Dirichlet-based ensemble computes the aggregated score for each gesture class by combining the similarity scores produced by all recognizers, weighted according to the Dirichlet-transformed weights. For each class, the weighted similarity scores are summed and stored in a mapping structure associating gesture classes with their aggregated scores. The final decision stage then selects the gesture class with the highest aggregated score as the ensemble prediction.

$$w_1[\hat{y}_1] + w_2[\hat{y}_2] + \cdots + w_n[\hat{y}_n] = [\hat{Y}] \tag{3.1}$$

### 3.2.3 Stacking

Since this thesis already explores a bagging-like method as well as a weighted averaging approach, we wanted to investigate another of the three main ensemble learning categories introduced in Chapter 2.2. The two remaining options are stacking and boosting. As stated in Hypothesis

in section 1.2.2, the objective of this work is to combine the recongizer from QuantumLeap which are fully developed individual recognizers. This objective is not compatible with boosting methods, which typically rely on weak base learners such as shallow decision trees. For this reason, the third ensemble model explored in this thesis focuses on stacking.

As explained previously, stacking consists of using a meta-learner that takes the outputs of the base recognizers as input and produces a final prediction. This is precisely the approach adopted for the third ensemble model implemented in this work.

Before implementing this method, a key design choice concerns the selection of the meta-learner. In the literature, several models are used for this purpose, with decision trees and support vector machines (SVMs) being the most common [11,15,40]. Between these two options, a decision tree was selected for two main reasons. First, SVMs are frequently used in the context of EMG-based gesture recognition [14], which does not correspond to the data modalities considered in this thesis. Our experiments rely on Leap Motion Controller data (vision-based) and the Kinemic dataset (wristband-based). Second, although execution time is not the primary focus of this thesis, stacking is the only ensemble approach considered here that requires an explicit training phase—unlike template-based recognition. As a result, training time remains a relevant consideration, and decision trees generally require less training time than SVMs, which further supports this choice.

A final design decision concerns the type of information provided to the meta-learner. Two alternatives are available: using the standard `recognize` method, which returns only the best predicted class and its corresponding score, or using the `recognizeAllSimilarities` method, which returns similarity scores for all gesture classes. Both approaches were implemented in order to evaluate which representation leads to better performance.

To implement stacking, a JavaScript machine learning library was integrated, and its decision tree implementation was used as the meta-learner. The process proceeds as follows. For each template in the dataset used for training, meta-data are extracted from the outputs of the base recognizers, either using `recognize` or `recognizeAllSimilarities`. Once this extraction is completed for all training samples, the meta-learner is trained on these meta-features. After training, the same process is applied to the test data: for a candidate gesture, the outputs of each base recognizer are computed and passed to the trained meta-learner, which then produces the final prediction.

## 3.3 Testing

As explained earlier, the testing procedure had to be adapted by modifying the TTS-Testing class in order to allow the QuantumLeap framework to evaluate ensemble recognizers composed of multiple individual recognizers. However, an additional issue needed to be addressed to ensure a correct interpretation of the results from QuantumLeap.

To obtain statistically meaningful results, each repetition of the testing procedure must rely on random sampling. In this context, randomness refers to the random selection of candidate gestures from the dataset for testing. While this is necessary to preserve statistical validity, it also leads to slight variations in the results across different runs, since the training and testing splits may differ.

## 3.4 Summary

This chapter described the experimental setup designed to evaluate whether ensemble learning

can improve gesture recognition accuracy. It introduced the datasets used in this work, the modifications required to integrate them into the QuantumLeap framework, and the different ensemble strategies that were implemented and analyzed. In addition, the testing methodology was outlined, with particular emphasis on reproducibility and statistical validity. This setup forms the basis for the experimental evaluation presented in the next chapter, where the performance of the proposed ensemble models is systematically analyzed and compared.

# Chapter 4

# Results

This chapter presents the results obtained from all the experiments conducted throughout this thesis. First, the individual performance of each recognizer is analyzed. Then, the different ensemble learning methods are evaluated and compared against these baseline results. Although numerous combinations of ensemble configurations were explored, only the most representative and meaningful ones are reported in this chapter in order to provide a clear and relevant analysis of the results.

## 4.1 Parameters and Metrics

Before analyzing the results, this section introduces the different parameters that may vary across experiments, as well as the metrics used to evaluate performance.

Within the QuantumLeap framework, each test configuration requires several parameters. Some of these parameters are known to have little or no impact on recognition performance, as demonstrated by Sluÿters [42]. Consequently, these parameters are kept constant throughout all experiments. Other parameters are varied and analyzed, as they directly influence the results.

### 4.1.1 Immutable parameters

The first parameter that remains fixed is the number of repetitions performed during each test. Following the experimental protocol established by Sluÿters, this value is set to 100. Each repetition consists of selecting one sample as the test instance while the remaining samples are used for training.

Another immutable parameter is the number of tracked points. For the SHREC2019 dataset, gestures do not involve finger articulation; therefore, tracking only the palm of the hand is sufficient. Tracking additional points, such as fingertip positions, is unnecessary and has been shown to negatively affect performance. For the 3DTCGS and Kinemic datasets, only a single point is available by design, as these sensors track either a single hand trajectory or the wrist position, respectively.

All experiments are conducted using a user-independent evaluation scenario. This choice is motivated by the fact that user-independent recognition represents a more challenging and realistic setting, typically yielding lower accuracy. Moreover, in some datasets such as SHREC2019, it is not always explicitly specified which samples belong to the same user.

Finally, the number of resampled points used during preprocessing is kept constant. Sluÿters showed that varying this parameter has little impact on recognition accuracy. Therefore, the

default values provided by the QuantumLeap framework are used: 16 resampled points for all recognizers, except for the $\mu$ recognizers, for which the default value is 8.

### 4.1.2 Mutable parameters

The main mutable parameter analyzed in this chapter is the number of templates used in the pattern dataset, denoted by $T$. Depending on the dataset, this value varies due to differences in the number of available gesture samples per class. For the 3DTCGS dataset, $T$ takes the values 2, 4, and 8. For SHREC2019, which provides more repetitions per gesture, the values 2, 4, 8, and 16 are used. Finally, the Kinemic dataset allows for larger values due to its higher number of participants, resulting in $T = 2, 4, 8, 16$, and 32.

Another variable factor is the dataset itself, as experiments are conducted on SHREC2019, 3DTCGS, and Kinemic. In addition, ensemble individual combinations are varied. The follwing five combinations of base recognizers are evaluated for each dataset:

1. All recognizers, to assess whether increasing the number of models improves performance.

2. Jackknife, $P3+, and $\mu$V.

3. Jackknife, $P3+, and $\mu$F, replacing a single recognizer of the combination.

4. $Q3, 3Cent, and $\mu$F, to test a substantially different combination.

5. $Q3 and $\mu$F, to evaluate the effect of removing one recognizer.

For the Dirichlet-based ensemble, the weights assigned to each recognizer are also varied. Two strategies are explored: a metric-based approach, where weights are derived from the individual accuracy of each recognizer, and an intuition-based approach, where weights are chosen according to design considerations reported in the literature. All tested configurations are listed in Appendix A.

In total, 372 distinct test configurations are evaluated. Since each configuration involves 100 repetitions, this results in more than 37,200 individual recognition trials.

For each batch of tests, listed in the appendix A, corresponding to a given recognizer combination, the raw results are provided in a CSV file, named `numberFirstTest-numberLastTest.csv`, while a JSON file, named `numberFirstTest-numberLastTest.json`,contains the test parameters along with the associated accuracy, execution time, and confusion matrix. Those files can be found in the zip file provided as appendix

### 4.1.3 Metrics

Four metrics are used to analyze the experimental results. The primary metric is accuracy, defined as the ratio of correctly classified gestures to the total number of classifications. The second metric is precision, also referred to as the similarity score (or simply score) in Chapter **??**. This value corresponds to the similarity returned by a recognizer and reflects its confidence in the predicted class. In addition, the average execution time per recognition trial is measured in order to assess the computational cost of each method. Finally, confusion matrices are examined in selected cases to identify gesture classes that are particularly prone to misclassification.

## 4.2 Individual Results

Before presenting the results of the ensemble models, and in order to evaluate whether ensemble learning leads to improvements, we first report the performance of the individual recognizers. These results serve as reference baselines for all subsequent comparisons.

### 4.2.1 3DTCGS Dataset

We begin with Tests 1 to 6 performed on the 3DTCGS dataset, as listed in Appendix A. The corresponding recognition accuracies are reported in Table 4.1.

Table 4.1: Accuracies (%) of individual recognizers for Tests 1–6 on the 3DTCGS dataset

| **Recognizer** | **Test 1** | | | **Test 2** | | |
|---|---|---|---|---|---|---|
| | **T = 2** | **T = 4** | **T = 8** | **T = 2** | **T = 4** | **T = 8** |
| Jackknife | 91.62 | 94.31 | 95.08 | 91.96 | 94.27 | 94.77 |
| P3+ | 86.23 | 89.69 | 92.77 | 84.62 | 87.54 | 89.23 |
| Q3 | 79.42 | 85.12 | 88.19 | 77.58 | 82.12 | 84.08 |
| 3Cent | 76.85 | 86.65 | 92.50 | 77.19 | 86.38 | 91.27 |
| $\mu$F | 90.15 | 93.08 | 94.27 | 91.08 | 93.62 | 94.12 |
| $\mu$V | 58.08 | 64.88 | 69.62 | 54.50 | 60.31 | 64.50 |

When comparing these results with those reported by Sluÿters [42], we observe that the reported accuracies for P3+ and Q3 were respectively 84.00%, 88.73%, and 90.19% for P3+, and approximately 76%, 83.5%, and 86.7% for Q3. The values obtained in our experiments are close to these references, although differences exceeding 1% are observed in some cases.

To assess whether these differences were due to randomness in the testing process, the same experiments were repeated, yielding the results reported as Test 2 in Table 4.1. These results are closer to those reported by Sluÿters, particularly for P3+. This confirms that the observed variability is largely attributable to the random selection of training and testing samples.

It is also worth noting that the $\mu$V recognizer performs poorly on this dataset compared to the other methods. This observation will prove relevant when analyzing ensemble results later in this chapter.

For the purpose of comparison with ensemble models, three reference values are extracted from these results: the average accuracy, the maximum accuracy and the standard deviation. To reduce variability and simplify comparisons, only the case with $T = 8$ templates is considered.

For the five recognizer combinations defined in Section 4.1.2 for dataset 3DTCGS, the average accuracies are 86.33%, 82.83%, 92.71%, 89.82%, and 89.10%, respectively. The corresponding maximum accuracies are 94.77%, 94.77%, 94.77%, 94.12%, and 94.12%. The standard deviation are 11.36%, 16.12%, 3.02%, 5.17%, 7.01%.

For the SHREC2019 dataset, using the same protocol with $T = 8$ templates, the five recognizer combinations yield average accuracies of 90.93%, 92.13%, 93.87%, 89.73%, and 90.80%, respectively. The corresponding maximum accuracies are 97.00%, 97.00%, 97.00%, 93.60%, and 93.60%. The associated standard deviations are 2.79%, 3.82%, 1.86%, 1.21%, and 1.70%. Compared to 3DTCGS, SHREC2019 exhibits both higher average accuracies and substantially lower variability, reflecting the lower intrinsic complexity of this dataset and the more consistent performance of the recognizers.

For the Kinemic dataset, which is notably more challenging, the same analysis reveals significantly lower performance and higher dispersion. The average accuracies obtained for $T = 8$ templates are 30.13%, 29.57%, 37.70%, 30.68%, and 33.86%, respectively, while the maximum accuracies reach 48.95%, 48.95%, 48.95%, 39.67%, and 39.67%. The corresponding standard deviations are 13.06%, 19.85%, 5.64%, 5.30%, and 2.23%. These results highlight not only the difficulty of the Kinemic dataset for template-based recognizers, but also the strong sensitivity of performance to the chosen recognizer combination.

### 4.2.2 SHREC2019 Dataset

The same procedure was applied to Tests 32 to 37 on the SHREC2019 dataset. The resulting accuracies are shown in Table 4.2.

Table 4.2: Accuracies (%) of individual recognizers for Tests 32–37 on the SHREC2019 dataset

| **Recognizer** | **Test 1** | | | | **Test 2** | | | |
|---|---|---|---|---|---|---|---|---|
| | **T = 2** | **T = 4** | **T = 8** | **T = 16** | **T = 2** | **T = 4** | **T = 8** | **T = 16** |
| Jackknife | 89.0 | 93.6 | 96.6 | 98.6 | 87.4 | 92.4 | 97.0 | 99.6 |
| P3+ | 91.2 | 96.2 | 97.0 | 97.2 | 70.4 | 79.4 | 87.0 | 90.4 |
| Q3 | 84.4 | 89.8 | 91.2 | 94.2 | 70.6 | 79.2 | 82.2 | 87.2 |
| 3Cent | 80.0 | 87.6 | 92.6 | 95.6 | 80.2 | 87.6 | 93.8 | 98.0 |
| $\mu$F | 87.2 | 91.8 | 93.6 | 95.4 | 85.2 | 89.4 | 94.4 | 96.0 |
| $\mu$V | 79.8 | 86.6 | 90.2 | 94.0 | 63.8 | 65.0 | 69.6 | 67.8 |

In this case, larger discrepancies are observed between Test 1 and Test 2, particularly for P3+ and Q3. These differences were too large to be explained solely by random variation. Additional experiments revealed that Test 2 was conducted using multiple tracked points instead of only the palm position. This significantly degraded performance across several recognizers which confirms what was told in the previous section.

Since Sluÿters did not explicitly specify which tracked points were used, an exact reproduction of his results is not possible. For consistency, all subsequent experiments therefore rely exclusively on palm tracking.

Using again the case $T = 8$, the average accuracies for the five recognizer combinations are 90.93%, 92.13%, 93.87%, 89.73%, and 90.80%. The corresponding maximum accuracies are 97%, 97%, 97%, 93.6%, and 93.6%.

### 4.2.3 Kinemic Dataset

Finally, Tests 63 to 68 were conducted on the Kinemic dataset. The obtained accuracies are reported in Table 4.3.

As expected, recognition accuracy on the Kinemic dataset is significantly lower than for the vision-based datasets. Most recognizers remain below the 50% accuracy threshold, which limits their applicability in real-world scenarios.

For $T = 8$, the average accuracies for the five recognizer combinations are 30.13%, 29.57%, 37.70%, 30.68%, and 33.86%. The maximum accuracies observed are 48.95%, 48.95%, 48.95%, 39.67%, and 39.67%.

Table 4.3: Accuracies (%) of individual recognizers for Tests 63–68 on the Kinemic dataset

| **Recognizer** | **T = 2** | **T = 4** | **T = 8** | **T = 16** | **T = 32** |
|---|---|---|---|---|---|
| Jackknife | 36.76 | 43.76 | 48.95 | 52.19 | 54.10 |
| P3+ | 30.71 | 34.38 | 38.76 | 41.43 | 44.33 |
| Q3 | 29.29 | 32.76 | 36.52 | 39.81 | 41.33 |
| 3Cent | 19.05 | 24.33 | 29.33 | 35.43 | 42.52 |
| $\mu$F | 29.24 | 34.95 | 39.67 | 44.71 | 49.57 |
| $\mu$V | 8.48 | 10.57 | 10.62 | 10.90 | 10.81 |

### 4.2.4 Raw Results

All the results presented so far are produced by the testing class of the QuantumLeap framework. In addition to these outputs, further tests and visualizations were generated directly from the raw data. Until now, the analysis mainly focused on the final accuracy of each model. However, in order to produce this final prediction, the framework selects the best candidate among all possible predictions. These predictions are based on similarity scores computed during the recognition process. By analyzing these similarity scores, it becomes possible to estimate how confident a model is in its prediction, which provides additional insight beyond accuracy alone.

Figure 4.1 shows the average precision for Tests 32–37. This figure provides useful information, in particular regarding how confident each recognizer is in its predictions. It can be observed that the confidence scores do not necessarily reflect the average accuracy of the models: some recognizers achieve high accuracy while remaining relatively uncertain in their predictions. This information plays a significant role in the Dirichlet-weighted ensemble, where similarity scores are multiplied by the assigned weights, as well as in the hybrid voting approach, where confidence values are used in case of tied candidates. Overall, Jackknife appears to be the most confident recognizer, while $\mu$V and \$P3+ are less confident. Additionally, Jackknife and 3Cent exhibit very low variance in their confidence scores. The error bars shown in the figure are computed using $\alpha = 0.05$, corresponding to a 95% confidence interval.

Next, the average recognition time was also computed from the raw data. As shown in Figure 4.2, the execution times obtained from the raw measurements are consistent with the average times reported by the testing framework in Table 4.4. This analysis also provides additional insight, as it shows that the variance in execution time across different tests is very small, and that each recognizer exhibits stable computational behavior.

Table 4.4: Average recognition time (ms) of individual recognizers for Tests 32–37 on the Kinemic dataset

| **Recognizer** | **T = 2** | **T = 4** | **T = 8** | **T = 16** |
|---|---|---|---|---|
| Jackknife | 0.03 | 0.03 | 0.05 | 0.09 |
| P3+ | 0.06 | 0.09 | 0.17 | 0.34 |
| Q3 | 0.98 | 1.19 | 1.60 | 2.63 |
| 3Cent | 0.20 | 0.19 | 0.19 | 0.20 |
| $\mu$F | 0.03 | 0.02 | 0.03 | 0.05 |
| $\mu$V | 0.03 | 0.03 | 0.06 | 0.11 |

Having access to each individual repetition also makes it possible to analyze performance at the level of individual gesture classes. This allows the identification of specific classes that exhibit

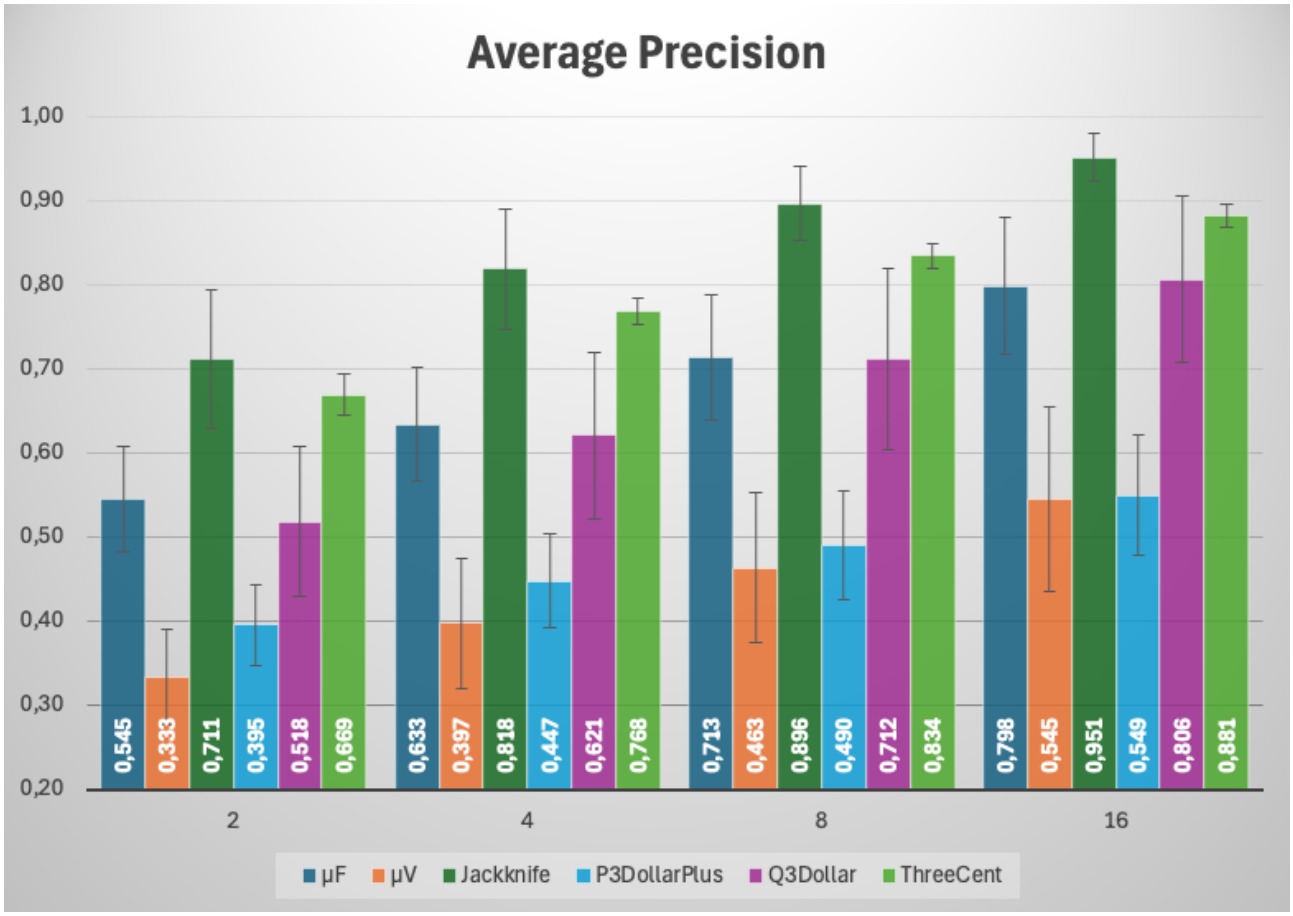


Figure 4.1: Average precision for Tests 32–37.

particularly low precision compared to others for the same recognizer. Figure 4.3 illustrates such an analysis for the *circle* gesture in Tests 32–37.

Finally, Figure 4.4 provides a global overview of the behavior of each recognizer when considering all gesture classes and all template configurations. This shows a real high variantion of recognizers over the different gestures.

The raw data for each test, along with the spreadsheets used to generate these analyses, are provided in the appendix. While similar analyses could be replicated to all ensemble configurations and gesture classes, this falls outside the scope of the present thesis.

## 4.3 Ensemble Results

We now turn to the results that directly address the research question of this thesis. As mentioned earlier, not all 93 executed tests are reported in this section. Instead, we focus on a selection of representative experiments that provide meaningful insight into the behavior and effectiveness of the different ensemble learning strategies.

### 4.3.1 Hybrid Voting

We begin with the hybrid voting approach and focus on the 3DTCGS dataset, which provides a sufficiently diverse set of gestures to analyze the effect of integrating recognizers with heterogeneous performance. In particular, Tests 12 and 13 include the $\mu$V recognizer, which was shown in the previous section to perform poorly when evaluated individually.

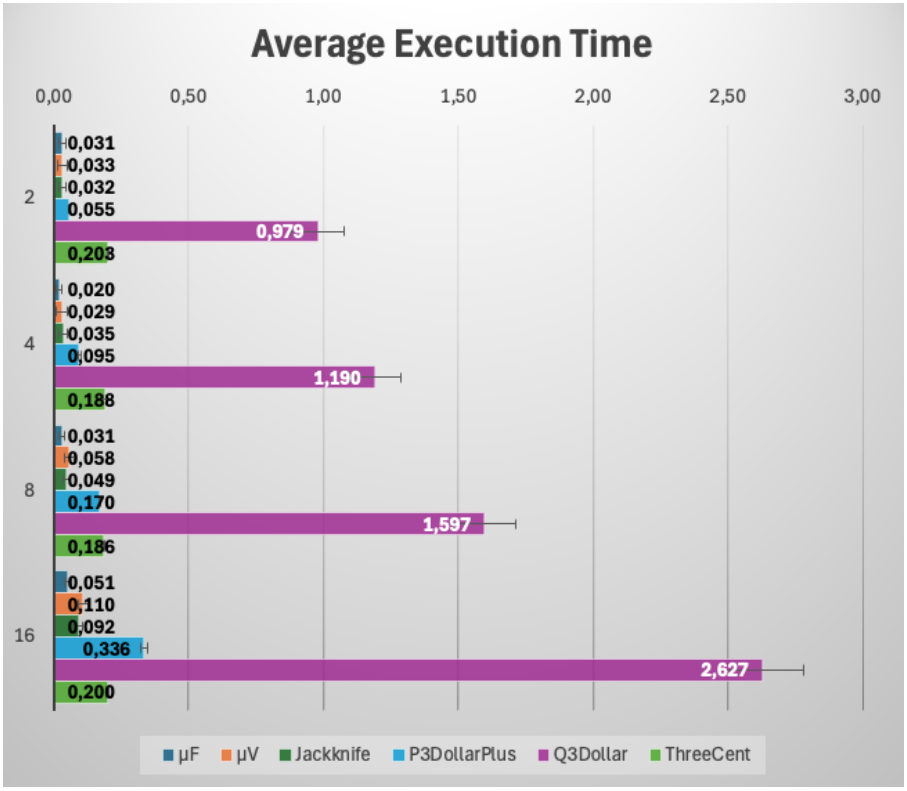


Figure 4.2: Average recognition time computed from raw data.

Table 4.5: Accuracies (%) for Hybrid Voting on 3DTCGS (Tests 12–16)

| **Test** | **T = 2** | **T = 4** | **T = 8** |
|---|---|---|---|
| Test12 | 91.2 | 94.6 | 96.8 |
| Test13 | 92.0 | 95.6 | 97.4 |
| Test14 | 89.4 | 92.4 | 96.2 |
| Test15 | 88.6 | 92.0 | 95.6 |
| Test16 | 89.4 | 92.4 | 93.6 |

Despite the inclusion of a weak recognizer, Tests 12 and 13 achieve the highest accuracies. When compared to the maximum individual recognizer accuracy, the relative gains are approximately 2.03%, 2.63%, 1.43%, 1.48%, and −0.52% for Tests 12 to 16, respectively.

When compared to the average accuracy of the individual recognizers, the improvement is substantially larger: 12.76%, 16.69%, 4.39%, 8.22%, and 5.73%. This indicates that hybrid voting is particularly effective at compensating for weaker recognizers by leveraging collective agreement.

A similar trend is observed on the SHREC2019 dataset, although the relative gains are smaller due to the already high baseline accuracy of the individual recognizers.

For the Kinemic dataset, improvements relative to the average accuracy are even more pronounced (23.25%, 12.10%, 3.97%, 9.13%, and 10.24%), reflecting the greater room for improvement given the low individual accuracies. However, slight performance losses are sometimes observed (4.43%, -7.28%, -7.28%, 0.14,%, 4.43%) when compared to the best-performing individual recognizers, highlighting a trade-off between robustness and peak accuracy.

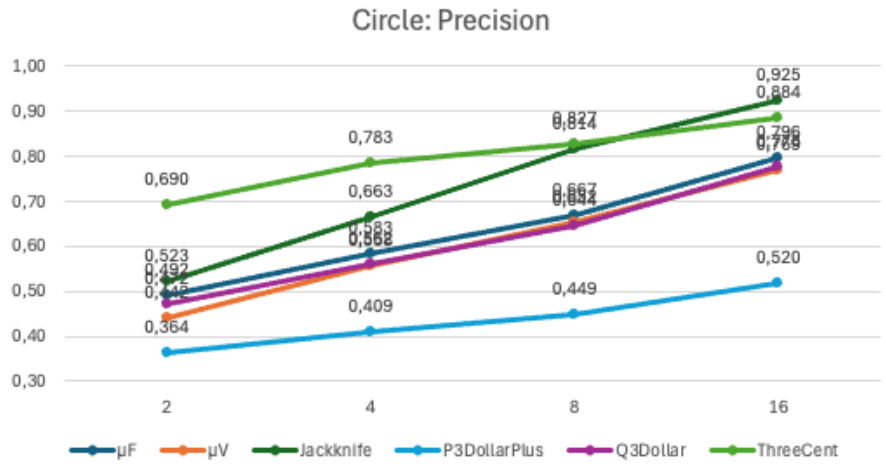


Figure 4.3: Class-specific analysis for the *circle* gesture in Tests 32–37.

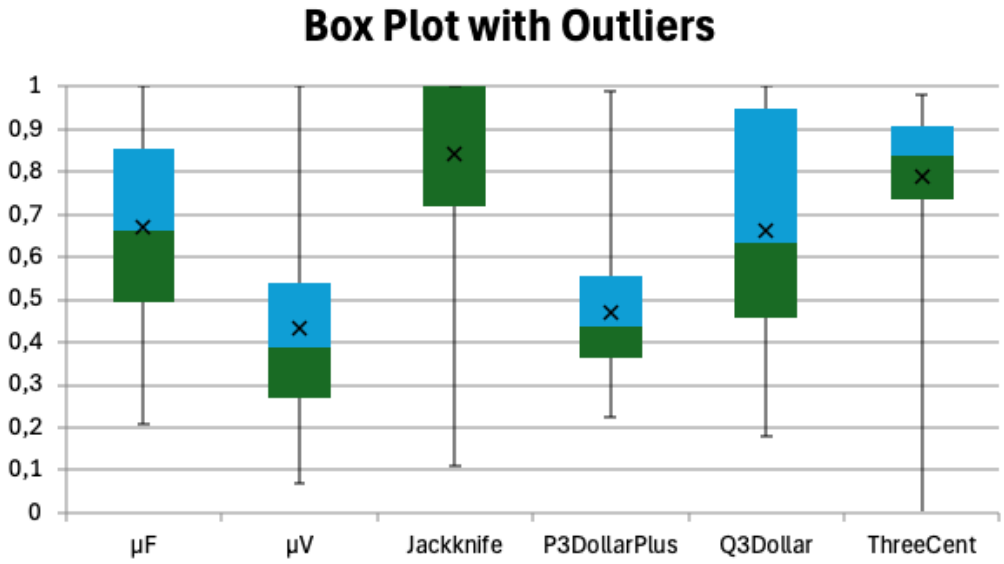


Figure 4.4: Global overview of recognizer behavior across all classes and template sizes.

### 4.3.2 Borda Voting

For Borda voting, we again focus on the 3DTCGS dataset, as SHREC2019 exhibits similar trends and will also analyze through confusion matrices to better understand unexpected behaviors. Additionally, the Kinemic dataset is analyzed on it's own for other trends in the results.

Relative to the best individual recognizer, the gains are modest (0.69%, −0.93%, 2.31%, −37.81%, and 0.5%). However, when compared to the average individual performance, Borda voting provides consistent improvements of 11.42%, 13.13%, 5.27%, −31.06%, and 6.75%.

A significant performance drop is observed for Test 10, which corresponds to the recognizer combination *Q3, 3Cent, and μF*. To investigate this behavior, the confusion matrix shown in Figure 4.5 was analyzed.

The confusion matrix reveals that early gesture classes are predicted accurately, while later classes suffer from systematic misclassification. False predictions are disproportionately assigned to early classes, suggesting a ranking bias similar to the one previously observed when using pure majority voting. Although Borda voting mitigates this issue in most cases, this specific combination still exhibits the same limitation.

On the Kinemic dataset, Borda voting shows mixed behavior. While average accuracy

Table 4.6: Accuracies (%) for Borda Voting on 3DTCGS (Tests 7–11)

| **Test** | **T = 2** | **T = 4** | **T = 8** |
|---|---|---|---|
| Test7 | 90.27 | 94.58 | 95.46 |
| Test8 | 89.00 | 93.26 | 93.84 |
| Test9 | 93.84 | 96.31 | 97.08 |
| Test10 | 61.69 | 59.12 | 56.31 |
| Test11 | 90.20 | 93.88 | 94.62 |

improvements are substantial (up to 26.33%), the maximum accuracy sometimes decreases, indicating once again a trade-off between robustness and peak performance.

### 4.3.3 Stacking

Stacking was evaluated using two variants: one based on the final class prediction returned by `recognize`, and another using the full similarity vectors returned by `recognizeAllSimilarities`. Both approaches were tested on the 3DTCGS dataset.

#### Final Prediction

Table 4.7: Stacking using final predictions (Tests 22–26)

| **Test** | **Accuracy (%)** | | | **Time (ms)** | | |
|---|---|---|---|---|---|---|
| | **T = 2** | **T = 4** | **T = 8** | **T = 2** | **T = 4** | **T = 8** |
| Test22 | 90.27 | 94.58 | 95.46 | 36.49 | 63.30 | 137.73 |
| Test23 | 89.00 | 93.26 | 93.84 | 1.96 | 4.91 | 15.11 |
| Test24 | 93.84 | 96.31 | 97.08 | 1.99 | 4.67 | 15.47 |
| Test25 | 61.69 | 59.12 | 56.31 | 32.58 | 56.61 | 119.83 |
| Test26 | 90.20 | 93.88 | 94.62 | 6.89 | 17.71 | 53.45 |

Unlike the other ensemble methods, stacking introduces a significant computational overhead. As shown in Table 4.7, the execution time exceeds the sum of individual recognizer times, shown in Tabl 4.8 by a factor ranging from 4 to 10, depending on the number of templates. This is explained by the fact that stacking is the only ensemble method that requires explicit training of a model as all the other models compute scores.

In terms of accuracy, stacking does not consistently outperform the best individual recognizer (with differences of −0.04%, −2.15%, 0.73%, −1.35%, and −1.54%). However, it provides substantial improvements compared to the average individual performance, with gains the respective gains of 10.69%, 11.91%, 3.69%, 5.40%, 4.71%.

#### All Predictions

When stacking is applied using full similarity vectors from `recognizeAllSimilarities`, performance degrades significantly in several cases.

In addition to the same training-related time overhead, accuracy losses exceeding 40% are observed in some cases. A likely explanation is that the decision tree becomes overly complex when receiving high-dimensional similarity vectors, leading to poor generalization.

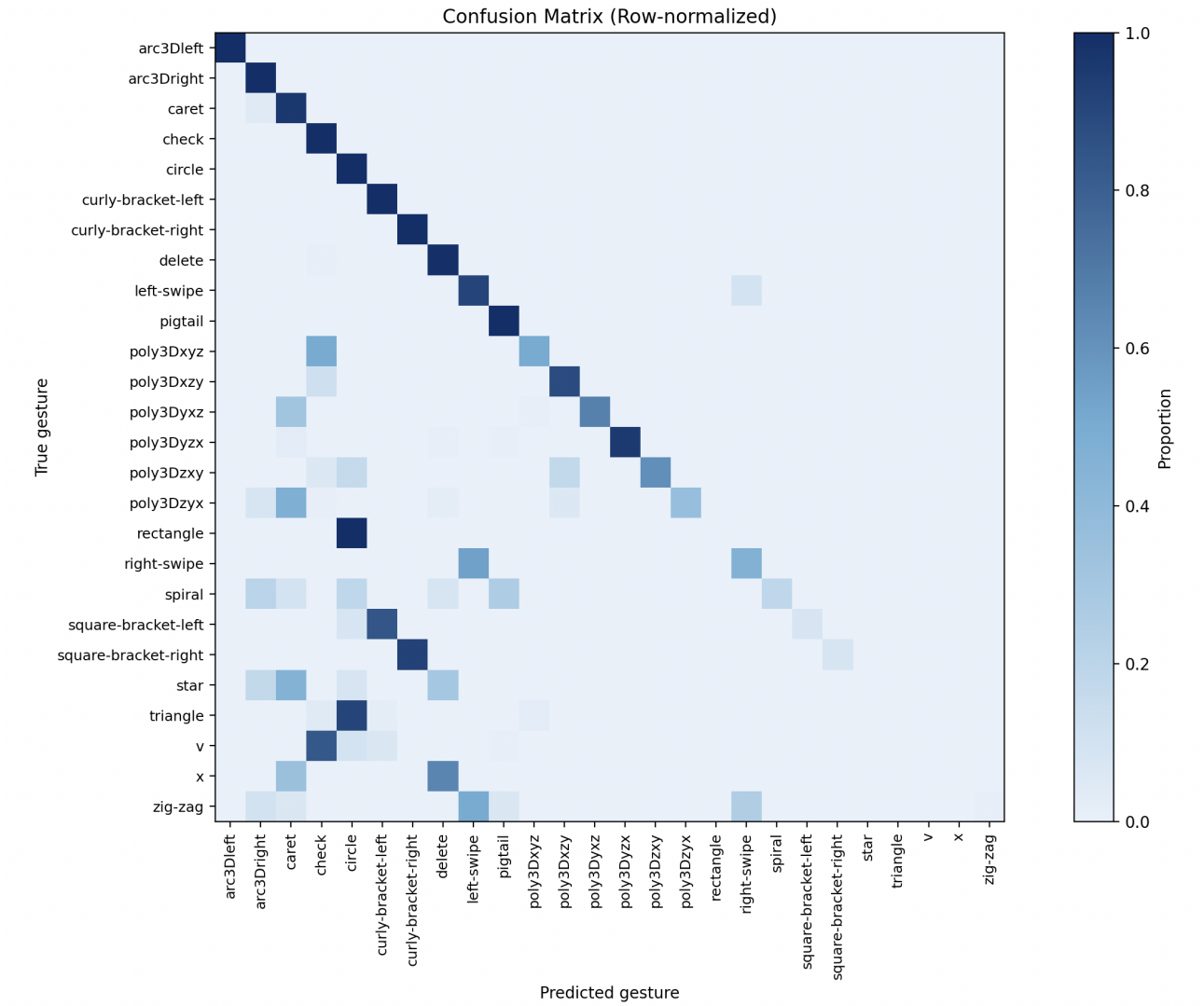


Figure 4.5: Confusion matrix for Test 10

### 4.3.4 Weighted Dirichlet Ensemble

Two weighting strategies were evaluated: a statistical approach based on individual recognizer accuracy and an intuition-based approach informed by the literature and dataset characteristics.

#### Statistical Weighting

For the statistical weighting strategy, the weights assigned to each recognizer are directly derived from their individual performance. More precisely, the accuracy obtained by each recognizer when evaluated independently with $T = 8$ templates on the same dataset is used as its corresponding weight. This choice ensures that the weighting reflects the empirical reliability of each recognizer under identical experimental conditions, without introducing additional assumptions.

This approach yields consistent improvements, with gains of 0.8%, 1.4%, 1.4%, 1.8%, 1.6% compared to the already high performing best individual recognizer and up to 7.87% compared to the average of the recognizers composing the combination. This represents the highest improvement observed on the SHREC2019 dataset.

Table 4.8: Average time (ms) of Individual Recognizers for test 1-6

| **Recognizer** | **T = 2** | **T = 4** | **T = 8** |
|---|---|---|---|
| Jackknife | 0,08 | 0,10 | 0,21 |
| P3+ | 0,27 | 0,49 | 0,96 |
| Q3 | 1,83 | 3,16 | 5,73 |
| 3cent | 6,43 | 6,52 | 6,50 |
| $\mu$F | 0,05 | 0,07 | 0,13 |
| $\mu$V | 0,08 | 0,14 | 0,27 |
| **Total Sum** | 8,74 | 10,48 | 13,80 |

Table 4.9: Stacking using all predictions (Tests 94–98)

| **Test** | **Accuracy (%)** | | | | **Time (ms)** | | | |
|---|---|---|---|---|---|---|---|---|
| **Total** | **T = 2** | **T = 4** | **T = 8** | **T = 16** | **T = 2** | **T = 4** | **T = 8** | **T = 16** |
| Test1 | 48.6 | 59.0 | 64.0 | 57.4 | 4.08 | 8.23 | 22.10 | 68.67 |
| Test2 | 49.2 | 56.4 | 63.0 | 57.6 | 0.55 | 1.44 | 4.32 | 14.21 |
| Test3 | 49.0 | 58.6 | 62.8 | 58.8 | 0.55 | 1.37 | 4.08 | 13.35 |
| Test4 | 46.6 | 60.0 | 64.4 | 67.2 | 3.35 | 6.91 | 17.05 | 51.31 |
| Test5 | 46.6 | 59.8 | 64.4 | 67.2 | 2.69 | 5.61 | 15.00 | 46.69 |

**Intuition-Based Weighting**

For the intuition-based weighting strategy, a fixed decreasing set of weights is used, namely 1.0, 0.8, 0.6, 0.4, 0.2, and 0.1. Higher weights are assigned to recognizers expected to be more reliable for a given dataset based on their design characteristics and behavior reported in the literature. The ordering of recognizers associated with these weights is dataset-dependent.

For the SHREC2019 dataset, the highest weight is assigned to $P3+, as it consistently yields strong performance on simple planar trajectories and often outperforms other recognizers on this dataset. $Q3 follows, as it shares similar characteristics while benefiting from computational optimizations that do not significantly affect accuracy on simple gestures. 3Cent is assigned the next weight, since rotation invariance is less critical for the oriented gestures present in SHREC2019. Jackknife is weighted lower due to the simplicity and short duration of the trajectories, followed by $\mu$F, whose 3D modeling capabilities provide limited benefit in this context. Finally, $\mu$V receives the lowest weight, as $\mu$F can be considered an improved variant.

For the 3DTCGS dataset, the same set of weight values is used, but the ordering is adapted. The highest weights are assigned in order to $\mu$F and $\mu$V, as their design is well suited to genuinely three-dimensional gestures. 3Cent follows, since the dataset was explicitly created to evaluate its capabilities on 3D trajectories. Jackknife is weighted next due to its ability to capture trajectory dynamics, although sensitivity to direction may limit its performance. $P3+ is assigned a lower weight, and $Q3 receives the lowest weight, as $P3+ generally outperforms it on this dataset.

For the Kinemic dataset, which relies on inertial wrist-motion data, Jackknife is assigned the highest weight due to its time-series formulation. $\mu$F follows for its ability to model 3D motion, with $\mu$V assigned a slightly lower weight. Point-cloud-based recognizers ($P3+, $Q3, and 3Cent) receive the lower weights in this order, as they are not specifically designed for inertial sensor

Table 4.10: Dirichlet ensemble with statistical weights (Tests 48–52)

| **Test** | **T = 2** | **T = 4** | **T = 8** | **T = 16** |
|---|---|---|---|---|
| Test1 | 90.4 | 93.6 | 97.8 | 98.8 |
| Test2 | 91.0 | 95.2 | 98.4 | 99.2 |
| Test3 | 90.6 | 94.6 | 98.4 | 99.4 |
| Test4 | 88.6 | 92.4 | 95.4 | 97.2 |
| Test5 | 86.6 | 91.8 | 95.2 | 98.0 |

data, with 3Cent assigned the lowest weight.

Table 4.11: Dirichlet ensemble with intuition-based weights (Tests 48–52)

| **Test** | **T = 2** | **T = 4** | **T = 8** | **T = 16** |
|---|---|---|---|---|
| Test1 | 93.0 | 94.2 | 97.0 | 97.8 |
| Test2 | 89.2 | 95.4 | 98.0 | 99.0 |
| Test3 | 90.0 | 95.2 | 97.6 | 99.2 |
| Test4 | 87.0 | 91.0 | 93.0 | 96.8 |
| Test5 | 87.8 | 91.2 | 93.4 | 97.4 |

Compared to the maximum individual accuracy, gains vary between $-1.8\%$ and $1.4\%$. When compared to the average, improvements remain substantial, ranging from $4.13\%$ to $8.47\%$. This confirms that while intuition-based weighting can be effective, its performance is more sensitive to recognizer selection.

### 4.3.5 Summary

This chapter presented the experimental results obtained throughout this thesis. It first clarified the different parameters involved in the evaluation process, distinguishing between fixed and varying parameters, and defined the performance metrics used to assess the models. The results of individual recognizers were then analyzed, followed by a detailed evaluation of the different ensemble learning methods and their variants. For each approach, the obtained accuracies and computational costs were reported and briefly discussed.

These results provide the necessary basis for a comparative analysis of the different ensemble strategies. The next chapter therefore focuses on identifying overall trends, comparing the methods across datasets, and highlighting the key observations that emerge from these experiments.

# Chapter 5

# Analysis

This chapter builds upon the results presented in Chapter 4, as well as the additional experiments reported in the appendix archive provided with this thesis. Based on these results, general trends are identified, conclusions are drawn regarding the effectiveness of the evaluated ensemble methods, and potential benefits and implications of the observed findings are discussed.

## 5.1 Overall Trends

This section analyzes the global tendencies observed across all ensemble learning experiments conducted in this thesis, with the objective of identifying consistent behaviors and practical insights that emerge independently of a specific dataset or ensemble configuration.

### 5.1.1 Average Accuracy

First and foremost, the results clearly show that ensemble techniques are effective. With the exception of two specific tests (Tests 17 and 41), which exhibit an anomalous drop in performance, every ensemble configuration evaluated leads to an improvement in accuracy when compared to the average accuracy of its composing individual recognizers.

As expected, the magnitude of this improvement strongly depends on the initial performance of the individual recognizers. Ensembles composed of weaker recognizers tend to benefit more from aggregation, as they leave more room for improvement. In contrast, when individual recognizers already perform well, the gain brought by the ensemble is naturally more limited.

Still when comparing ensemble accuracy to the average accuracy of the individual recognizers, a strong correlation appears between the standard deviation of individual performances and the ensemble gain. Configurations with higher standard deviation among their constituent recognizers tend to achieve larger improvements over the mean accuracy. This observation directly reflects a core principle of ensemble learning: as discussed in Chapter 2, ensembles benefit most from diversity. In this case, diversity is not primarily related to architectural differences, but rather to variability in predictions across recognizers.

### 5.1.2 Maximum Accuracy

However, in the context of gesture recognition, improving the average accuracy is not the sole objective. The practical goal is to design a recognizer that outperforms the *best* individual recognizer. From this perspective, the results are more nuanced.

Depending on the ensemble method and the specific combination of recognizers, improvements over the maximum individual accuracy are sometimes achieved, but not consistently. A clear tendency nonetheless emerges: ensembles composed exclusively of high-performing recognizers are more likely to outperform the best individual model than ensembles that include weaker recognizers. Introducing low-performing recognizers often degrades the ensemble's ability to surpass the strongest baseline, even if the average accuracy improves.

### 5.1.3 Number of Templates

Across all datasets, a consistent trend is observed with respect to the number of templates used. As already noted for individual recognizers, increasing the number of templates leads to higher accuracy, and this tendency directly carries over to ensemble models.

This effect is particularly pronounced for more challenging datasets such as 3DTCGS and Kinemic, which involve a large number of gesture classes. In these cases, increasing the number of training examples per gesture significantly improves recognition, highlighting the importance of sufficient template coverage in complex gesture spaces.

### 5.1.4 Bad performing recognizers

Another important tendency that appears in the results is what happens when one of the composing recognizers is clearly weaker than the others. In our experiments this is explicitly the case for $\mu V$, which shows significantly lower accuracy than the other individual recognizers on the datasets which bigger amount of gestures. Intuitively, one could expect that adding such a weak recognizer would always decrease the ensemble performance. However, the results show that this is not necessarily true: voting-based ensembles can still reach the best accuracies even when a weak recognizer is included. This indicates that, as long as the remaining recognizers provide strong and consistent predictions, the weak recognizer does not dominate the decision and its errors are effectively compensated by the collective agreement.

At the same time, the data also shows that the effect of a weak recognizer depends on the dataset difficulty. On simpler datasets with already high baseline accuracy (such as SHREC2019), adding a weaker recognizer tends to provide limited additional benefit, because the best recognizer already performs near a ceiling and there is less room to improve. On harder datasets (such as Kinemic), weak recognizers can either be absorbed by the ensemble if their mistakes are not aligned with the others, or they can contribute to losses when they systematically push predictions toward wrong labels. Therefore, a weak recognizer is not always harmful, but it can become problematic when its mistakes are highly systematic and repeated across trials.

### 5.1.5 Impact of Ensemble Size

Increasing the number of recognizers in an ensemble does not automatically improve performance on difficult gesture classes. This is explicitly visible in our tests where the ensemble size is varied through the five tested combinations, ranging from *all recognizers* to a minimal *two-recognizer* ensemble ($Q3$ and $\mu F$). In hybrid voting on 3DTCGS, the smallest ensemble (2 recognizers) achieves the lowest accuracy at $T = 8$ (93.6%) and is the only configuration that slightly underperforms the best individual recognizer (reported relative gain of $-0.52\%$). In contrast, the 3-recognizer configurations reach higher accuracies at $T = 8$ (up to 97.4%) and yield positive gains compared to the best individual recognizer (up to $+2.63\%$). This supports the idea that moving from very small ensembles to medium-size ensembles can improve robustness because

tie situations and single-recognizer dominance become less frequent, while still preserving the contribution of the strongest member.

However, the results also show that ensemble size alone is not sufficient to guarantee stable class-level behavior: what matters is whether recognizers fail in diverse or correlated ways. A clear counterexample is provided by Borda voting on 3DTCGS, where Test 10 (a 3-recognizer ensemble: $Q3$, 3Cent, $\mu F$) collapses to 56.31% at $T = 8$ (reported $-37.81\%$ compared to the best recognizer). The corresponding confusion matrix indicates a systematic pattern where later gesture classes are misclassified and false predictions accumulate in early classes, meaning that the additional recognizer does not add complementary information but instead reinforces a shared ranking bias. A similar trade-off appears on Kinemic: hybrid voting yields very large gains compared to the average (up to $+23.25\%$), yet some combinations still lose compared to the best individual recognizer (down to $-7.28\%$). This confirms that larger ensembles mainly improve robustness relative to the *average* member, while beating the *best* member remains sensitive to the chosen combination and to whether the added recognizers contribute diverse or correlated errors.

### 5.1.6 Practical Implications for Ensemble Design

These observations indicate that ensemble design for gesture recognition should not focus solely on maximizing global accuracy. When the objective is to outperform the best individual recognizer, it is crucial to avoid combinations that introduce strong correlated failure modes on difficult classes.

A robust ensemble design strategy should therefore ensure that at least one recognizer performs well on difficult gestures, and the remaining recognizers do not systematically collapse these gestures into the same incorrect class. This also explains why weighting strategies or careful recognizer selection can be more effective than simply increasing the number of recognizers in the ensemble.

## 5.2 Specific Ensemble Behaviors

This section analyzes the behavior of each ensemble family in more detail, highlighting their respective strengths, limitations, and typical use cases in the context of gesture recognition.

### 5.2.1 Hybrid Majority Voting

Across all datasets, hybrid voting behaves as the most reliable "default" ensemble strategy. Its strongest and most consistent effect is to improve performance relative to the *average* of the composing recognizers, even when the set contains one clearly weaker member (notably $\mu V$). This behavior appears on 3DTCGS where the best hybrid voting results are obtained despite including $\mu V$ (e.g., 97.4% at $T = 8$), and on Kinemic where gains compared to the average are the largest of all methods (e.g., improvements above +20 points are observed in the reported comparisons).

The difference between datasets is mainly visible when comparing against the *best* individual recognizer. On SHREC2019, where individual baselines already approach a ceiling, hybrid voting rarely brings large gains over the maximum and may even become slightly negative in some combinations. On Kinemic, although hybrid voting strongly improves robustness, it may still underperform the best individual recognizer for specific combinations (e.g., negative

differences of about $-7.28\%$ are reported), which indicates that the vote can be "dragged" by weaker models when the best baseline is significantly above the rest.

Overall, the general tendency is that hybrid voting is the most stable method across datasets: it is rarely catastrophic and almost always improves the ensemble compared to the mean, but it does not guarantee beating the best recognizer, especially in near-ceiling conditions (SHREC2019) or when the member set is poorly balanced (Kinemic).

### 5.2.2 Borda Voting

Borda voting follows the same global direction as hybrid voting in the typical case (improving over the average), but it is markedly more sensitive to the chosen recognizer combination, and this sensitivity is consistent across datasets. In favorable combinations, Borda remains competitive (e.g., reaching 97.08% at $T = 8$ on 3DTCGS). However, the key cross-dataset tendency is the existence of strong failure modes: on 3DTCGS, one combination collapses to 56.31% at $T = 8$ (Test 10), and on SHREC2019 another collapse occurs (Test 41) with accuracies decreasing as $T$ increases (down to 74.0% at $T = 16$). On both datasets, these failures are not small fluctuations but structural breakdowns, and the confusion matrix analysis supports that they correspond to systematic misclassification patterns rather than random noise.

On Kinemic, Borda can yield some of the largest gains over the average (the method benefits from the large dispersion of individual accuracies), but it remains inconsistent when the goal is to beat the best recognizer: depending on the combination, performance can be above or below the maximum baseline.

Therefore, the general conclusion is that Borda is potentially strong but less robust than hybrid voting. Its behavior is more "combination-dependent", and across all datasets it is the method most prone to rare but extreme collapses.

### 5.2.3 Stacking

Stacking exhibits a consistent cross-dataset behavior: it mainly acts as a method that improves performance relative to the *average* baseline, but it does not reliably exceed the *best* individual recognizer. On 3DTCGS, this is visible in the reported comparisons where stacking gains over the mean are clearly positive while gains over the best recognizer are small or negative depending on the combination. The same tendency appears on SHREC2019, where stacking remains below the best baseline and the near-ceiling setting makes improvements harder to obtain.

A second cross-dataset tendency is that stacking introduces a large computational overhead compared to voting-based strategies. This overhead is present for all datasets because stacking requires explicit training of a meta-model. In addition, the "all similarities" stacking variant produces a consistent degradation pattern: the SHREC2019 tests using `recognizeAllSimilarities` drop to very low accuracies (around the 50–65% range depending on $T$), showing that the high-dimensional feature representation is not suitable for this meta-learning setup.

Overall, stacking is consistent but not competitive in this benchmark when peak accuracy and real-time constraints are considered: it increases robustness (vs average), but it is expensive and not reliable for surpassing the strongest base recognizer.

### 5.2.4 Dirichlet

Dirichlet-based ensembles show the most consistent behavior across datasets when the goal is to obtain controlled improvements without relying on strict agreement between recognizers. On

SHREC2019 in particular, Dirichlet with statistical (metric-based) weights produces systematic positive gains over the best recognizer (reported around +0.8% to +1.8%), which is notable given the already high baseline. This indicates that score-level aggregation is especially beneficial in near-ceiling regimes, where voting has little margin to improve.

On Kinemic, Dirichlet remains robust and typically improves over the average, but gains over the best recognizer are more variable and depend strongly on the weighting strategy and the recognizer set. This is coherent with the dataset being highly dispersed: weighting can help stabilize the contribution of weak recognizers, but if weights do not match the dataset-specific strengths, the ensemble may fail to surpass the best baseline.

The most general and transferable conclusion is that Dirichlet is the most "controlled" ensemble family in the sense that it tends to avoid the catastrophic behaviors observed in Borda and behaves more predictably than pure voting when weights are grounded in measured performance. However, its success depends on weight adequacy: metric-based weighting is consistently safer than intuition-based ordering across datasets.

# Chapter 6

# Future Work

Now that the results and analyses have been presented, it is possible to identify several aspects that were not covered in this thesis and that could constitute promising directions for future work.

As stated earlier, the primary objective of this thesis was to explore the applicability of ensemble learning to template-based gesture recognition, rather than to exhaustively optimize a single ensemble configuration. As a consequence, a wide range of ensemble models and recognizer combinations were evaluated. The results show that ensemble learning does not benefit all combinations equally: for a given ensemble method, some recognizer combinations lead to significantly larger improvements than others. This suggests that more targeted experiments could be conducted by designing dataset-specific ensembles, where recognizers are selected and combined according to their complementary strengths on a given dataset.

In addition, several parameters were kept fixed in this work in order to limit the experimental space. Further gains in accuracy could potentially be achieved by fine-tuning parameters such as the number of templates, the number and type of tracked points, the choice of tracked joints, or the number of repetitions used during evaluation. A similar observation applies to the Dirichlet-based ensemble, where only statistical and intuition-based weighting strategies were explored; alternative weighting schemes or optimization procedures could lead to improved results.

Another limitation of this work is that only recognizers implemented within the QuantumLeap framework were considered. While this ensured a consistent experimental setup, it also restricts the diversity of the ensemble. Other template-based recognizers exist outside this framework, and hybrid ensembles combining fundamentally different approaches could be explored. In particular, combining template-based recognizers with learning-based methods such as convolutional neural networks could provide complementary strengths. However, as observed with stacking in Chapter 4, learning-based approaches introduce training time and computational costs, which must be carefully considered in real-time gesture recognition scenarios.

Beyond the specific ensemble methods evaluated, there exist many alternative ensemble techniques within each major category. While this thesis focused on the most representative approaches, each category contains numerous variants that were not explored due to time constraints. For example, in stacking, only a decision tree was used as a meta-learner, whereas many other models discussed in Section 2.2.2 could have been considered. Exploring these alternatives would significantly expand the experimental space and could lead to further insights.

Finally, data fusion was introduced in the state-of-the-art review (Section 2.4) but was not investigated experimentally, as it falls outside the main research question of this thesis. Nevertheless, fusion techniques share conceptual similarities with ensemble learning and could

further improve recognition performance by enriching the input representation. Data fusion could be studied as an independent research direction or combined with ensemble learning, particularly in early or mid-fusion settings, to exploit complementary sensor information more effectively.

# Chapter 7

# Conclusion

To conclude this work, it is useful to reflect on what has been achieved throughout this thesis.

First, a comprehensive review of the literature was conducted to explain what ensemble learning is, how it works, and which main categories of ensemble methods exist. Based on this theoretical foundation, an analysis was carried out to identify how ensemble learning could be applied to the field of gesture recognition. This review highlighted that, despite the strong potential of ensemble methods, relatively few studies currently explore ensemble learning specifically for gesture recognition, making this field both limited and promising for further research.

Fusion was then introduced to better understand how it relates to ensemble learning. Although both approaches share the objective of combining information to improve accuracy and robustness, fusion operates by combining different data sources or modalities, whereas ensemble learning combines different interpretations or models applied to the same data. This distinction helped clarify why fusion was not the primary focus of this work, while still being conceptually relevant.

Next, the structure of gesture recognizers was examined in detail. Starting from the foundational template-based recognizer $1, recognizers were organized into three main families: point-cloud-based, distortion-aware, and vector-based recognizers. This categorization provided a clearer understanding of how different recognizers operate and why combining them may lead to complementary behavior.

The experimental setup was then introduced through the three datasets used in this thesis. SHREC2019 represents a small and relatively simple dataset, 3DTCGS introduces a larger number of gesture classes and higher complexity, and the Kinemic dataset presents particularly challenging conditions with overall low baseline accuracy. These datasets allowed ensemble methods to be evaluated under varying levels of difficulty. The ensemble learning techniques considered in this work were also presented, together with their implementation details and the adjustments required in the framework to support them.

The experimental results provide several important insights. Overall, ensemble learning proves to be effective for gesture recognition: with only two exceptions, all ensemble configurations outperform the average accuracy of their composing recognizers. However, ensembles do not always surpass the best individual recognizer, especially when a very strong recognizer dominates the performance. A key observation is that ensemble learning yields the largest improvements on difficult datasets, where individual recognizers perform poorly and leave more room for improvement. Another important finding is the strong relationship between ensemble performance and diversity: higher variance among individual recognizers—reflected by larger standard deviation—generally leads to better ensemble performance. Finally, among all

tested methods, the highest peak performance was achieved using the Dirichlet-based weighted averaging approach with statistically derived weights.

These results answer and confirm the research question by establishing that ensemble learning is a valuable strategy for template-based gesture recognition, particularly when robustness and consistent performance improvements are desired.

# List of Figures

# List of Tables

# Glossary

**ANN** Artificial Neural Network.

**CNN** Convolutional Neural Network.

**HCI** Human–Computer Interaction.

**KNN** K-Nearest Neighbors.

**LMC** Leap Motion Controller.

**NN** Neural Network.

**SVM** Support Vector Machine.

# Bibliography


[1] Simegnew Alaba. A comprehensive survey of deep learning multisensor fusion-based 3d object detection for autonomous driving: Methods, challenges, open issues, and future directions, 08 2022.

[2] Soufiene Ameur, Abdellatif Ben Khalifa, and Mohamed Salah Bouhlel. A Comprehensive Leap Motion Database for Hand Gesture Recognition. In *2016 7th International Conference on Sciences of Electronics, Technologies of Information and Telecommunications*, pages 514–519. IEEE, 2016.

[3] Lisa Anthony and Jacob O Wobbrock. $ n-protractor: a fast and accurate multistroke recognizer. In *Proceedings of Graphics Interface 2012*, pages 117–120. 2012.

[4] Nick Bilton. It's the Hardware's Turn in the Spotlight. https://www.nytimes.com/2013/03/08/technology/its-the-hardwares-turn-in-the-spotlight.html, 2013. [Online; accessed 20-December-2025].

[5] Jordan J. Bird, Anikó Ekárt, and Diego R. Faria. British Sign Language Recognition via Late Fusion of Computer Vision and Leap Motion with Transfer Learning to American Sign Language. *Sensors*, 20(18):5151, 2020.

[6] Gavin Brown, Jeremy Wyatt, Rachel Harris, and Xin Yao. Diversity Creation Methods: A Survey and Categorisation. *Information Fusion*, 6(1):5–20, March 2005.

[7] Alexandre Calado, Paolo Roselli, Vito Errico, Nathan Magrofuoco, Jean Vanderdonckt, and Giovanni Saggio. A geometric model-based approach to hand gesture recognition. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, PP:1–11, 01 2022.

[8] Fabio M. Caputo, Stefano Burato, Gabriele Pavan, Andrea Giachetti, Théo Voillemin, Mehran Maghoumi, Eugene M. Taranta II, Ali Razmjoo, Joseph J. LaViola Jr., Fabio Manganaro, Stefano Pini, Giulio Borghi, Riccardo Vezzani, Rita Cucchiara, Huy V. Nguyen, and Minh T. Tran. SHREC 2019 Track: Online Gesture Recognition. In *Eurographics Workshop on 3D Object Retrieval*, Goslar, DEU, 2019. The Eurographics Association.

[9] Fabio Marco Caputo, Pietro Prebianca, Alessandro Carcangiu, Lucio D. Spano, and Andrea Giachetti. A 3 Cent Recognizer: Simple and Effective Retrieval and Classification of Mid-air Gestures from Single 3D Traces. In Andrea Giachetti, Paolo Pingi, and Filippo Stanco, editors, *Smart Tools and Apps for Graphics - Eurographics Italian Chapter Conference*. The Eurographics Association, 2017.

[10] Enea Ceolini, Charlotte Frenkel, Sumit Bam Shrestha, Gemma Taverni, Lyes Khacef, Melika Payvand, and Elisa Donati. Hand-Gesture Recognition Based on EMG and Event-Based Camera Sensor Fusion: A Benchmark in Neuromorphic Computing. *Frontiers in Neuroscience*, 14:637, 2020.

[11] Shuhong Cheng, Liyuan Wu, Shijun Zhang, Dianfan Zhang, Fei Liu, Hongbo Wang, and Ping Xie. A model for lumbar emg signal recognition based on stacking integration learning. *IEEE Sensors Journal*, 23(4):3766–3775, 2023.

[12] Federico Cunico, Federico Girella, Andrea Avogaro, Marco Emporio, Andrea Giachetti, and Marco Cristani. OO-dMVMT: A Deep Multi-view Multi-task Classification Framework for Real-time 3D Hand Gesture Classification and Segmentation. *Sensors*, 21(9):3184, 2021.

[13] David Dalpiaz. *Basics of Statistical Learning*. 2020. Department of Statistics, University of Illinois Urbana–Champaign.

[14] Wilfredo K. Deveza, John B. Ibarra, and Mark Sejera. Silent Speech Interface Based on Surface Electromyography Using Arduino and Ensemble Learning. In *Robotics, Computer Vision and Intelligent Systems*, volume 2629 of *ROBOVIS 2025*, Cham, Switzerland, 2026. Springer.

[15] Ritwik Dey and Rachit Mathur. Ensemble Learning Method Using Stacking with Base Learner: A Comparison. In *Proceedings of the International Conference on Data Analytics and Insights*, volume 727 of *ICDAI 2023*, pages 173–186, Singapore, 2023. Springer.

[16] Zorana Doždor, Tomislav Hrkać, and Zoran Kalafatić. Two-Model-Based Online Hand Gesture Recognition from Skeleton Data. *Sensors*, 16(4):596, 2016.

[17] Julián García, José Pascual Molina, Diego Martínez, Arturo S. García, Pascual González, and Jean Vanderdonckt. Prototyping and Evaluating Glove-Based Multimodal Interfaces. *Journal of Multimodal User Interfaces*, 2(1):43–52, 2008.

[18] Manuel Gil-Martín, Marco Raoul Marini, Rubén San-Segundo, and Luigi Cinque. Dual Leap Motion Controller 2: A Robust Dataset for Multi-view Hand Pose Recognition. *Scientific Data*, 11:1102, 2024.

[19] Kinemic GmbH. Kinemic Band. https://kinemic.com/en/kinemic-band/, 2025. [Online; accessed 20-December-2025].

[20] Sang Han, Seonho Lee, Hyeok Nam, Jae Hyeon Park, Min Hee Cha, Min Geol Kim, Hyunse Lee, Sangyeon Ahn, Moonju Chae, and Sung In Cho. Doodle to Detect: A Goofy but Powerful Approach to Skeleton-Based Hand Gesture Recognition. *Pattern Recognition*, 150:110322, 2024.

[21] Nan-Chen Hsieh and Lun-Ping Hung. A data driven ensemble classifier for credit scoring analysis. *Expert Systems with Applications*, 37(1):534–545, 2010.

[22] Eugene M. Taranta II, Amirreza Samiei, Mehran Maghoumi, Pooya Khaloo, Corey R. Pittman, and Joseph J. LaViola Jr. Jackknife: A Reliable Recognizer with Few Samples and Many Modalities. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, CHI '17, pages 5850–5861, New York, NY, USA, 2017. Association for Computing Machinery.

[23] R. Kumar and S. Kumar. A Survey on Intelligent Human Action Recognition Techniques. *Multimedia Tools and Applications*, 83:52653–52709, May 2024.

[24] Rahul Kumar and Sujay Jayashankar. Radar and camera sensor fusion with ros for autonomous driving. In *2019 Fifth International Conference on Image Information Processing (ICIIP)*, pages 568–573, 2019.

[25] Ghazanfar Latif, Nazeer Mohammad, Jehad Alghazo, Rawan AlKhalaf, and Raghad AlKhalaf. ARASL: Arabic Alphabets Sign Language Dataset. *Data in Brief*, 23:103777, 2019.

[26] Kevin Lupinetti, Alessandra Ranieri, Francesco Giannini, and Marco Monti. 3D Dynamic Hand Gestures Recognition Using the Leap Motion Sensor and Convolutional Neural Networks. *CoRR*, abs/2003.01450, 2020.

[27] Magrofuoco. $_F$ : $An Adaptable (R) ST - Invariant Multimodal Gesture Recognizer$. 2026.

[28] Nathan Magrofuoco, Paolo Roselli, and Jean Vanderdonckt. Two-dimensional stroke gesture recognition: A survey. *ACM Comput. Surv.*, 54(7), July 2021.

[29] Nathan Magrofuoco, Paolo Roselli, and Jean Vanderdonckt. μv: An articulation, rotation, scaling, and translation invariant (arst) multi-stroke gesture recognizer. *Proc. ACM Hum.-Comput. Interact.*, 6(EICS), June 2022.

[30] Giulio Marin, Fabio Dominio, and Pietro Zanuttigh. Hand Gesture Recognition with Leap Motion and Kinect Devices. In *2014 IEEE International Conference on Image Processing (ICIP)*, pages 1565–1569. IEEE, 2014.

[31] Robert McCartney, Jun Yuan, and Hans-Peter Bischof. Gesture Recognition with the Leap Motion Controller. In *Proceedings of the International Conference on Computer Vision Theory and Applications*. SciTePress, 2015.

[32] Aws Saood Mohamed, Nidaa Flaih Hassan, and Abeer Salim Jamil. Real-Time Hand Gesture Recognition: A Comprehensive Review of Techniques, Applications, and Challenges. *Cybernetics and Information Technologies*, 24(3):163–182, 2024.

[33] Jacob Murel and Eda Kavlakoglu. What is ensemble learning? https://www.ibm.com/think/topics/ensemble-learning, 2025. [Online; accessed 26-October-2025].

[34] Abdirahman Osman Hashi, Siti Zaiton Mohd Hashim, and Azurah Bte Asamah. A systematic review of hand gesture recognition: An update from 2018 to 2024. *IEEE Access*, 12:143599–143626, 2024.

[35] Mehdi Ousmer, Arthur Sluÿters, Nathan Magrofuoco, Paolo Roselli, and Jean Vanderdonckt. Recognizing 3d trajectories as 2d multi-stroke gestures. *Proc. ACM Hum.-Comput. Interact.*, 4(ISS), November 2020.

[36] Mehdi Ousmer, Théo Voillemin, Jean Vanderdonckt, and Radu-Daniel Vatavu. A Systematic Procedure for Comparing Template-Based Gesture Recognizers. In *Proceedings of the 14th ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, EICS '22, pages 1–16, New York, NY, USA, 2022. Association for Computing Machinery.

[37] Fulai Peng, Cai Chen, Danyang Lv, Ningling Zhang, Xingwei Wang, Xikun Zhang, and Zhiyong Wang. Gesture Recognition by Ensemble Extreme Learning Machine Based on Surface Electromyography Signals. *Frontiers in Human Neuroscience*, 16:911204, 2022.

[38] Giuseppe Placidi, Luigi Cinque, Andrea Petracca, Matteo Polsinelli, and Matteo Spezialetti. A virtual glove system for the hand rehabilitation based on two orthogonal leap motion controllers. pages 184–192, 01 2017.

[39] Robi Polikar. Ensemble Based Systems in Decision Making. *IEEE Circuits and Systems Magazine*, 6(3):21–45, 2006.

[40] Nishkam Ravi, Nikhil Dandekar, Preetham Mysore, and Michael L. Littman. Activity Recognition from Accelerometer Data. In *Proceedings of the 17th Conference on Innovative Applications of Artificial Intelligence (IAAI)*, pages 1541–1546, Pittsburgh, Pennsylvania, USA, 2005. AAAI Press.

[41] Serkan Savaş and Atilla Ergüzen. Hand Gesture Recognition with Two Stage Approach Using Transfer Learning and Deep Ensemble Learning. In *Proceedings of the International Conference on Intelligent Systems and New Applications (ICISNA'23)*, Kırıkkale, Türkiye, 2023. Kırıkkale University. [Online; accessed 26-October-2025].

[42] Arthur Sluÿters. A Framework for Engineering Gesture-Based User Interfaces Using the Leap Motion Controller: Development and Applications. Master's thesis, Université catholique de Louvain, Louvain-la-Neuve, Belgium, 2020.

[43] Arthur Sluÿters. QuantumLeap: A Framework for Gesture Recognition. https://github.com/sluyters/QuantumLeap, 2022. [Online; accessed 20-December-2025].

[44] Sarah Steeman and Chloé Chauvaux. Uni/Bi-Manual Wrist Gesture Interaction for One/Two Users: Application to Kinemic. Master's thesis, Université catholique de Louvain, Louvain-la-Neuve, Belgium, 2022.

[45] Wenbin Tao, Zhi-Hua Lai, Ming C. Leu, and Zhaozheng Yin. American Sign Language Alphabet Recognition Using Leap Motion Controller. In *2018 IEEE International Conference on Image Processing (ICIP)*, pages 2304–2310. IEEE, 2018.

[46] Jean Vanderdonckt, Paolo Roselli, and Jorge Luis Pérez-Medina. !ftl, an articulation-invariant stroke gesture recognizer with controllable position, scale, and rotation invariances. In *Proceedings of the 20th ACM International Conference on Multimodal Interaction*, ICMI '18, page 125–134, New York, NY, USA, 2018. Association for Computing Machinery.

[47] Jean Vanderdonckt and Radu-Daniel Vatavu. Four Replications of a User-Defined Gesture Study with Smart Rings. In *Adjunct Proceedings of the 27th International Conference on Mobile Human-Computer Interaction*, MobileHCI '25 Adjunct, New York, NY, USA, 2025. Association for Computing Machinery.

[48] Radu-Daniel Vatavu. Improving gesture recognition accuracy on touch screens for users with low vision. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, CHI '17, page 4667–4679, New York, NY, USA, 2017. Association for Computing Machinery.

[49] Radu-Daniel Vatavu, Lisa Anthony, and Jacob O. Wobbrock. Gestures as point clouds: a $precognizer for user interface prototypes. In Proceedings of the 14th ACM International Conference on$

[50] Radu-Daniel Vatavu, Lisa Anthony, and Jacob O. Wobbrock. $q$ : $a super - quick, articulation - invariant stroke - gesture recognizer for low - resource devices. In Proceedings of the 20th International Conference on Human - Computer Interaction u$

[51] Jian Wang, Siyuan Lu, Shui-Hua Wang, and Yu-Dong Zhang. A Review on Extreme Learning Machine. *Multimedia Tools and Applications*, 81:41611–41660, 2022.

[52] Jacob O. Wobbrock, Andrew D. Wilson, and Yang Li. Gestures without libraries, toolkits or training: a $1 recognizer for user interface prototypes. In *Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology*, UIST '07, page 159–168, New York, NY, USA, 2007. Association for Computing Machinery.

[53] Xiang Xiang, Qiang Tan, Hong Zhou, Dong Tang, and Jiancheng Lai. Multimodal Fusion of Voice and Gesture Data for UAV Control. *Drones*, 6(8):201, 2022.

[54] Ionuţ-Alexandru Zaiti, Ştefan-Gheorghe Pentiuc, and Radu-Daniel Vatavu. On Free-Hand TV Control: Experimental Results on User-Elicited Gestures with Leap Motion. *Personal and Ubiquitous Computing*, 19(5–6):821–838, 2015.

# Appendix A

# All performed tests

In this appendix you will find on the next page the pdf of the excel file which is filed with the different varying parameters of each test as well as the corresponding accuracy and time obtained and several other computed metrics such as the average of all the composing recognizer of each combination.

**ALL TEST AND CORRESPONDING RESULTS THAT HAVE BEEN OBTAINED**

| Numero | Dataset | Recognizer Ensemble Name | Recognizer Individuel1 | Recognizer Individuel2 | Recognizer Individuel3 | Recognizer Individuel4 | Recognizer Individuel5 | Recongizer Individuel6 | Respective Weight | Range Template (N) |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 3DTCGS | Jackknife | / | | | | | | | 2-8 |
| 2 | 3DTCGS | P3+ | / | | | | | | | 2-8 |
| 3 | 3DTCGS | Q3 | / | | | | | | | 2-8 |
| 4 | 3DTCGS | 3 Cent | / | | | | | | | 2-8 |
| 5 | 3DTCGS | µF | / | | | | | | | 2-8 |
| 6 | 3DTCGS | µV | / | | | | | | | 2-8 |
| 7 | 3DTCGS | Borda | Jack | P3+ | µV | 3 Cent | Q3 | µF | | 2-8 |
| 8 | 3DTCGS | Borda | Jack | P3+ | µV | | | | | 2-8 |
| 9 | 3DTCGS | Borda | Jack | P3+ | µF | | | | | 2-8 |
| 10 | 3DTCGS | Borda | Q3 | 3 Cent | µF | | | | | 2-8 |
| 11 | 3DTCGS | Borda | Q3 | µF | | | | | | 2-8 |
| 12 | 3DTCGS | Majority_Voting | Jack | P3+ | µV | 3 Cent | Q3 | µF | | 2-8 |
| 13 | 3DTCGS | Majority_Voting | Jack | P3+ | µV | | | | | 2-8 |
| 14 | 3DTCGS | Majority_Voting | Jack | P3+ | µF | | | | | 2-8 |
| 15 | 3DTCGS | Majority_Voting | Q3 | 3 Cent | µF | | | | | 2-8 |
| 16 | 3DTCGS | Majority_Voting | Q3 | µF | | | | | | 2-8 |
| 17 | 3DTCGS | Dirichlet | Jack | P3+ | µV | 3 Cent | Q3 | µF | Respective | 2-8 |
| 18 | 3DTCGS | Dirichlet | Jack | P3+ | µV | | | | Respective | 2-8 |
| 19 | 3DTCGS | Dirichlet | Jack | P3+ | µF | | | | Respective | 2-8 |
| 20 | 3DTCGS | Dirichlet | Q3 | 3 Cent | µF | | | | Respective | 2-8 |
| 21 | 3DTCGS | Dirichlet | Q3 | µF | | | | | Respective | 2-8 |
| 22 | 3DTCGS | Stacking_by_class | Jack | P3+ | µV | 3 Cent | Q3 | µF | | 2-8 |
| 23 | 3DTCGS | Stacking_by_class | Jack | P3+ | µV | | | | | 2-8 |
| 24 | 3DTCGS | Stacking_by_class | Jack | P3+ | µF | | | | | 2-8 |
| 25 | 3DTCGS | Stacking_by_class | Q3 | 3 Cent | µF | | | | | 2-8 |
| 26 | 3DTCGS | Stacking_by_class | Q3 | µF | | | | | | 2-8 |
| 27 | 3DTCGS | Dirichlet | Jack | P3+ | µV | 3 Cent | Q3 | µF | Intuitive | 2-8 |
| 28 | 3DTCGS | Dirichlet | Jack | P3+ | µV | | | | Intuitive | 2-8 |
| 29 | 3DTCGS | Dirichlet | Jack | P3+ | µF | | | | Intuitive | 2-8 |
| 30 | 3DTCGS | Dirichlet | Q3 | 3 Cent | µF | | | | Intuitive | 2-8 |
| 31 | 3DTCGS | Dirichlet | Q3 | µF | | | | | Intuitive | 2-8 |
| 32 | SHREC2019 | Jackknife | / | | | | | | | 2-16 |
| 33 | SHREC2019 | P3+ | / | | | | | | | 2-16 |
| 34 | SHREC2019 | Q3 | / | | | | | | | 2-16 |
| 35 | SHREC2019 | 3 Cent | / | | | | | | | 2-16 |
| 36 | SHREC2019 | µF | / | | | | | | | 2-16 |
| 37 | SHREC2019 | µV | / | | | | | | | 2-16 |
| 38 | SHREC2019 | Borda | Jack | P3+ | µV | 3 Cent | Q3 | µF | | 2-16 |
| 39 | SHREC2019 | Borda | Jack | P3+ | µV | | | | | 2-16 |
| 40 | SHREC2019 | Borda | Jack | P3+ | µF | | | | | 2-16 |
| 41 | SHREC2019 | Borda | Q3 | 3 Cent | µF | | | | | 2-16 |
| 42 | SHREC2019 | Borda | Q3 | µF | | | | | | 2-16 |
| 43 | SHREC2019 | Majority_Voting | Jack | P3+ | µV | 3 Cent | Q3 | µF | | 2-16 |
| 44 | SHREC2019 | Majority_Voting | Jack | P3+ | µV | | | | | 2-16 |
| 45 | SHREC2019 | Majority_Voting | Jack | P3+ | µF | | | | | 2-16 |
| 46 | SHREC2019 | Majority_Voting | Q3 | 3 Cent | µF | | | | | 2-16 |

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 47 | SHREC2019 | Majority_Voting | Jack | Q3 | | | | | | | 2-16 |
| 48 | SHREC2019 | Dirichlet | Jack | P3+ | μV | 3 Cent | Q3 | μF | Respective | 2-16 |
| 49 | SHREC2019 | Dirichlet | Jack | P3+ | μV | | | | Respective | 2-16 |
| 50 | SHREC2019 | Dirichlet | Jack | P3+ | μF | | | | Respective | 2-16 |
| 51 | SHREC2019 | Dirichlet | Q3 | 3 Cent | μF | | | | Respective | 2-16 |
| 52 | SHREC2019 | Dirichlet | Q3 | μF | | | | | Respective | 2-16 |
| 53 | SHREC2019 | Stacking_by_class | Jack | P3+ | μV | 3 Cent | Q3 | μF | | 2-16 |
| 54 | SHREC2019 | Stacking_by_class | Jack | P3+ | μV | | | | | 2-16 |
| 55 | SHREC2019 | Stacking_by_class | Jack | P3+ | μF | | | | | 2-16 |
| 56 | SHREC2019 | Stacking_by_class | Q3 | 3 Cent | μF | | | | | 2-16 |
| 57 | SHREC2019 | Stacking_by_class | Q3 | μF | | | | | | 2-16 |
| 58 | SHREC2019 | Dirichlet | Jack | P3+ | μV | 3 Cent | Q3 | μF | Intuitive | |
| 59 | SHREC2019 | Dirichlet | Jack | P3+ | μV | | | | Intuitive | 2-32 |
| 60 | SHREC2019 | Dirichlet | Jack | P3+ | μF | | | | Intuitive | 2-32 |
| 61 | SHREC2019 | Dirichlet | Q3 | 3 Cent | μF | | | | Intuitive | 2-32 |
| 62 | SHREC2019 | Dirichlet | Q3 | μF | | | | | Intuitive | 2-32 |
| | | | | | | | | | | |
| 63 | Kinemic | Jackknife | / | | | | | | | 2-32 |
| 64 | Kinemic | P3+ | / | | | | | | | 2-32 |
| 65 | Kinemic | Q3 | / | | | | | | | 2-32 |
| 66 | Kinemic | 3 Cent | / | | | | | | | 2-32 |
| 67 | Kinemic | μF | / | | | | | | | 2-32 |
| 68 | Kinemic | μV | / | | | | | | | 2-32 |
| 69 | Kinemic | Borda | Jack | P3+ | μV | 3 Cent | Q3 | μF | | 2-32 |
| 70 | Kinemic | Borda | Jack | P3+ | μV | | | | | 2-32 |
| 71 | Kinemic | Borda | Jack | P3+ | μF | | | | | 2-32 |
| 72 | Kinemic | Borda | Q3 | 3 Cent | μF | | | | | 2-32 |
| 73 | Kinemic | Borda | Q3 | μF | | | | | | 2-32 |
| 74 | Kinemic | Majority_Voting | Jack | P3+ | μV | 3 Cent | Q3 | μF | | 2-32 |
| 75 | Kinemic | Majority_Voting | Jack | P3+ | μV | | | | | 2-32 |
| 76 | Kinemic | Majority_Voting | Jack | P3+ | μF | | | | | 2-32 |
| 77 | Kinemic | Majority_Voting | Q3 | 3 Cent | μF | | | | | 2-32 |
| 78 | Kinemic | Majority_Voting | Jack | Q3 | | | | | | 2-32 |
| 79 | Kinemic | Dirichlet | Jack | P3+ | μV | 3 Cent | Q3 | μF | Respective | 2-32 |
| 80 | Kinemic | Dirichlet | Jack | P3+ | μV | | | | Respective | 2-32 |
| 81 | Kinemic | Dirichlet | Jack | P3+ | μF | | | | Respective | 2-32 |
| 82 | Kinemic | Dirichlet | Q3 | 3 Cent | μF | | | | Respective | 2-32 |
| 83 | Kinemic | Dirichlet | Q3 | μF | | | | | Respective | 2-32 |
| 84 | Kinemic | Stacking_by_class | Jack | P3+ | μV | 3 Cent | Q3 | μF | | |
| 85 | Kinemic | Stacking_by_class | Jack | P3+ | μV | | | | | |
| 86 | Kinemic | Stacking_by_class | Jack | P3+ | μF | | | | | |
| 87 | Kinemic | Stacking_by_class | Q3 | 3 Cent | μF | | | | | |
| 88 | Kinemic | Stacking_by_class | Q3 | μF | | | | | | |
| 89 | Kinemic | Dirichlet | Jack | P3+ | μV | 3 Cent | Q3 | μF | Intuitive | 2-32 |
| 90 | Kinemic | Dirichlet | Jack | P3+ | μV | | | | Intuitive | 2-32 |
| 91 | Kinemic | Dirichlet | Jack | P3+ | μF | | | | Intuitive | 2-32 |
| 92 | Kinemic | Dirichlet | Q3 | 3 Cent | μF | | | | Intuitive | 2-32 |
| 93 | Kinemic | Dirichlet | Q3 | μF | | | | | Intuitive | 2-32 |
| | | | | | | | | | | |
| 94 | SHREC2019 | Stacking_by_predictions | Jack | P3+ | μV | 3 Cent | Q3 | μF | | 2-16 |
| 95 | SHREC2019 | Stacking_by_predictions | Jack | P3+ | μV | | | | | 2-16 |
| 96 | SHREC2019 | Stacking_by_predictions | Jack | P3+ | μF | | | | | 2-16 |
| 97 | SHREC2019 | Stacking_by_predictions | Q3 | 3 Cent | μF | | | | | 2-16 |
| 98 | SHREC2019 | Stacking_by_predictions | Q3 | μF | | | | | | 2-16 |

| Accuracy | | | | | Max | Avg | Stdev | Time | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | Second test to explain randomness | | | |
| 91,96 | 94,27 | 94,77 | | Combi 1 | 94,77 | 84,04 | 11,3696322 | 0,08 | 0,1 | 0,21 | | 91,62 | 94,31 | 95,08 | |
| 84,62 | 87,54 | 89,23 | | Combi 2 | 94,77 | 80,7066667 | 16,1169548 | 0,27 | 0,49 | 0,96 | | 86,23 | 89,69 | 92,77 | |
| 77,58 | 82,12 | 84,08 | | Combi 3 | 94,77 | 91,81 | 3,0283714 | 1,83 | 3,16 | 5,73 | | 79,42 | 85,12 | 88,19 | |
| 77,19 | 86,38 | 91,27 | | Combi 4 | 94,12 | 87,3733333 | 5,17397655 | 6,43 | 6,52 | 6,5 | | 76,85 | 86,65 | 92,5 | |
| 91,08 | 93,62 | 94,12 | | Combi 5 | 94,12 | 87,87 | 7,09935208 | 0,05 | 0,07 | 0,13 | | 90,15 | 93,08 | 94,27 | |
| 54,5 | 60,31 | 64,5 | | | | | | 0,08 | 0,14 | 0,27 | | 58,08 | 64,88 | 69,62 | |
| 90,27 | 94,58 | 95,46 | | Difference | 0,69 | 11,42 | | 8,896 | 10,112 | 12,798 | | | | | |
| 89 | 93,26 | 93,84 | | between the | -0,93 | 13,1333333 | | 0,402 | 0,78 | 1,526 | | | | | |
| 93,84 | 96,31 | 97,08 | | result and | 2,31 | 5,27 | | 0,38 | 0,717 | 1,553 | | | | | |
| 61,69 | 59,12 | 56,31 | | the max or | -37,81 | -31,063333 | | 7,966 | 9,045 | 11,127 | | | | | |
| 90,2 | 93,88 | 94,62 | | avg | 0,5 | 6,75 | | 1,614 | 2,738 | 4,857 | | | | | |
| 91,2 | 94,6 | 96,8 | | | 2,03 | 12,76 | | 1,19 | 1,52 | 1,98 | | | | | |
| 92 | 95,6 | 97,4 | | | 2,63 | 16,6933333 | | 0,09 | 0,15 | 0,28 | | | | | |
| 89,4 | 92,4 | 96,2 | | | 1,43 | 4,39 | | 0,08 | 0,15 | 0,26 | | | | | |
| 88,6 | 92 | 95,6 | | | 1,48 | 8,22666667 | | 0,96 | 1,17 | 1,61 | | | | | |
| 89,4 | 92,4 | 93,6 | | | -0,52 | 5,73 | | 0,76 | 0,99 | 1,51 | | | | | |
| 93,12 | 95,08 | 96,42 | | | 1,65 | 12,38 | | 9,59 | 11,05 | 14,48 | | | | | |
| 91,92 | 94,5 | 94,58 | | | -0,19 | 13,8733333 | | 0,42 | 0,79 | 1,55 | | | | | |
| 93,27 | 95,23 | 95,46 | | | 0,69 | 3,65 | | 0,39 | 0,71 | 1,55 | | | | | |
| 87,65 | 92,77 | 94,96 | | | 0,84 | 7,58666667 | | 8,49 | 9,84 | 12,34 | | | | | |
| 86,77 | 92,62 | 94,62 | | | 0,5 | 6,75 | | 1,88 | 3,31 | 5,99 | | | | | |
| 90,04 | 93,46 | 94,73 | | | -0,04 | 10,69 | | 36,49 | 63,3 | 137,73 | | | | | |
| 91,46 | 93,15 | 92,62 | | | -2,15 | 11,9133333 | | 1,96 | 4,91 | 15,11 | | | | | |
| 91,92 | 94,69 | 95,5 | | | 0,73 | 3,69 | | 1,99 | 4,67 | 15,47 | | | | | |
| 82,88 | 88,73 | 92,77 | | | -1,35 | 5,39666667 | | 32,58 | 56,61 | 119,83 | | | | | |
| 83,23 | 89,15 | 92,58 | | | -1,54 | 4,71 | | 6,89 | 17,71 | 53,45 | | | | | |
| 92,5 | 95,19 | 95,92 | | | 1,15 | 11,88 | | 8,97 | 10,33 | 13,19 | | | | | |
| 90,46 | 94,04 | 94,65 | | | -0,12 | 13,9433333 | | 0,42 | 0,78 | 1,54 | | | | | |
| 92,62 | 95,54 | 95,85 | | | 1,08 | 4,04 | | 0,38 | 0,71 | 1,55 | | | | | |
| 90,5 | 94,08 | 95,54 | | | 1,42 | 8,16666667 | | 8,28 | 9,26 | 11,39 | | | | | |
| 89,92 | 93,81 | 95,5 | | | 1,38 | 7,63 | | 1,58 | 2,74 | 4,85 | | | | | |
| | | | | | | | | | | | | All tracking points | | | |
| 89 | 93,6 | 96,6 | 98,6 | | 97 | 90,9333333 | 2,78759155 | 0,03 | 0,03 | 0,05 | 0,09 | 87,4 | 92,4 | 97 | 99,6 |
| 91,2 | 96,2 | 97 | 97,2 | | 97 | 92,1333333 | 3,81575681 | 0,06 | 0,09 | 0,17 | 0,34 | 70,4 | 79,4 | 87 | 90,4 |
| 84,4 | 89,8 | 91,2 | 94,2 | | 97 | 93,8666667 | 1,85831465 | 0,98 | 1,19 | 1,6 | 2,63 | 70,6 | 79,2 | 82,2 | 87,2 |
| 80 | 87,6 | 92,6 | 95,6 | | 93,6 | 89,7333333 | 1,20554275 | 0,2 | 0,19 | 0,19 | 0,2 | 80,2 | 87,6 | 93,8 | 98 |
| 87,2 | 91,8 | 93,6 | 95,4 | | 93,6 | 90,8 | 1,69705627 | 0,03 | 0,02 | 0,03 | 0,05 | 85,2 | 89,4 | 94,4 | 96 |
| 79,8 | 86,6 | 90,2 | 94 | | | | | 0,03 | 0,03 | 0,06 | 0,11 | 63,8 | 65 | 69,6 | 67,8 |
| 90,4 | 95,2 | 97 | 97,8 | | 0 | 6,06666667 | | 1,275 | 1,417 | 2,042 | 3,373 | | | | |
| 92 | 97,8 | 98,6 | 98,6 | | 1,6 | 6,46666667 | | 0,091 | 0,18 | 0,287 | 0,559 | | | | |
| 90 | 95 | 97 | 98,4 | | 0 | 3,13333333 | | 0,09 | 0,138 | 0,262 | 0,535 | | | | |
| 78 | 76,8 | 74,8 | 74 | | -18,8 | -14,933333 | | 0,951 | 1,278 | 1,734 | 2,549 | | | | |
| 89,2 | 92,8 | 95,8 | 97,6 | | 2,2 | 5 | | 0,767 | 1,019 | 1,575 | 2,505 | | | | |
| 89 | 94,8 | 97,2 | 99,2 | | 0,2 | 6,26666667 | | 1,3 | 1,72 | 2,34 | 3,78 | | | | |
| 92 | 96,4 | 97,4 | 98,4 | | 0,4 | 5,26666667 | | 0,09 | 0,16 | 0,29 | 0,57 | | | | |
| 88 | 93,4 | 96,4 | 99 | | -0,6 | 2,53333333 | | 0,08 | 0,15 | 0,27 | 0,61 | | | | |
| 85,8 | 91,2 | 94,4 | 97,4 | | 0,8 | 4,66666667 | | 1,04 | 1,38 | 1,93 | 3,24 | | | | |

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 84,8 | 90,4 | 92,6 | 95,4 | | -1 | 1,8 | | 0,85 | 1,15 | 1,78 | 3,12 | |
| 90,4 | 93,6 | 97,8 | 98,8 | | 0,8 | 7,86666667 | | 1,29 | 1,5 | 2,21 | 3,58 | |
| 91 | 95,2 | 98,4 | 99,2 | | 1,4 | 7,06666667 | | 0,1 | 0,16 | 0,3 | 0,57 | |
| 90,6 | 94,6 | 98,4 | 99,4 | | 1,4 | 5,53333333 | | 0,1 | 0,15 | 0,27 | 0,55 | |
| 88,6 | 92,4 | 95,4 | 97,2 | | 1,8 | 7,46666667 | | 1,04 | 1,33 | 1,77 | 2,88 | |
| 86,6 | 91,8 | 95,2 | 98 | | 1,6 | 7,2 | | 0,84 | 1,09 | 1,6 | 2,68 | |
| 86 | 91 | 95 | 98 | | -2 | 4,06666667 | | 4,65 | 8,14 | 18,99 | 55,35 | |
| 86 | 91 | 95 | 98 | | -2 | 2,86666667 | | 0,4 | 0,96 | 2,8 | 9,29 | |
| 86 | 91 | 95 | 98 | | -2 | 1,13333333 | | 0,4 | 0,89 | 2,51 | 8,32 | |
| 85 | 87,6 | 91 | 93,4 | | -2,6 | 1,26666667 | | 3,82 | 6,78 | 15,19 | 41,76 | |
| 85 | 87,6 | 91 | 93,4 | | -2,6 | 0,2 | | 3,08 | 5,52 | 12,85 | 37,16 | |
| 93 | 94,2 | 97 | 97,8 | | 1,4 | 8,46666667 | | 1,34 | 1,81 | 2,54 | 3,88 | |
| 89,2 | 95,4 | 98 | 99 | | -1,6 | 5,06666667 | | 0,11 | 0,17 | 0,29 | 0,58 | |
| 90 | 95,2 | 97,6 | 99,2 | | -1,8 | 4,13333333 | | 0,09 | 0,14 | 0,28 | 0,51 | |
| 87 | 91 | 93 | 96,8 | | 1,4 | 8,26666667 | | 1,09 | 1,34 | 1,95 | 3,17 | |
| 87,8 | 91,2 | 93,4 | 97,4 | | 1,4 | 7,2 | | 0,81 | 1,15 | 1,69 | 3 | |
| | | | | | | | | | | | | |
| 36,76 | 43,76 | 48,95 | 52,19 | 54,1 | 48,95 | 30,125 | 13,0620745 | 0,13 | 0,2 | 0,47 | 1,05 | 2,33 |
| 30,71 | 34,38 | 38,76 | 41,43 | 44,33 | 48,95 | 29,57 | 19,8531467 | 0,21 | 0,4 | 0,77 | 1,53 | 3,06 |
| 29,29 | 32,76 | 36,52 | 39,81 | 41,33 | 48,95 | 37,6966667 | 5,63889174 | 1,58 | 2,62 | 4,76 | 8,63 | 16,34 |
| 19,05 | 24,33 | 29,33 | 35,43 | 42,52 | 39,67 | 30,68 | 5,2999088 | 25,5 | 21,44 | 21,89 | 21,69 | 21,26 |
| 29,24 | 34,95 | 39,67 | 44,71 | 49,57 | 39,67 | 33,855 | 2,22738636 | 0,06 | 0,08 | 0,14 | 0,31 | 0,88 |
| 8,48 | 10,57 | 10,62 | 10,9 | 10,81 | | | | 0,08 | 0,13 | 0,25 | 0,49 | 0,95 |
| 35,62 | 42,29 | 47,38 | 52,29 | 55,67 | -1,57 | 17,255 | | 20,25 | 23,13 | 24,69 | 31,12 | 43,78 |
| 43,76 | 50,29 | 55,9 | 60,86 | 65,14 | 6,95 | 26,33 | | 0,42 | 0,78 | 1,63 | 3,36 | 6,65 |
| 31,33 | 35,29 | 41,57 | 44,95 | 45,86 | -7,38 | 3,87333333 | | 0,43 | 0,8 | 1,61 | 3,35 | 6,61 |
| 21,76 | 25,9 | 31,48 | 35,48 | 38,81 | -8,19 | 0,8 | | 20,52 | 19,91 | 22,35 | 27,01 | 36,59 |
| 36,76 | 42,33 | 47,19 | 53 | 56,19 | 7,52 | 13,335 | | 1,76 | 3,02 | 5,59 | 10,32 | 19,74 |
| 40,29 | 48 | 53,38 | 58,67 | 63,81 | 4,43 | 23,255 | | 24,6 | 24,57 | 27,71 | 32,93 | 44,21 |
| 33,05 | 38,29 | 41,67 | 44,24 | 48,71 | -7,28 | 12,1 | | 0,42 | 0,78 | 1,54 | 3,27 | 6,52 |
| 33,05 | 38,29 | 41,67 | 44,24 | 48,71 | -7,28 | 3,97333333 | | 0,42 | 0,79 | 1,57 | 3,22 | 6,54 |
| 25,38 | 32,52 | 39,81 | 47,19 | 53,48 | 0,14 | 9,13 | | 23 | 23,98 | 25,75 | 29,24 | 36,75 |
| 33,14 | 40,24 | 44,1 | 48,48 | 51,62 | 4,43 | 10,245 | | 1,51 | 2,54 | 4,67 | 8,49 | 16,31 |
| 40,29 | 46 | 51,71 | 55,38 | | 2,76 | 21,585 | | 20,89 | 23,35 | 25,35 | 31,56 | |
| 36 | 42,81 | 46,67 | 49,24 | | -2,28 | 17,1 | | 0,43 | 0,81 | 1,65 | 3,35 | |
| 42,38 | 46,9 | 52,57 | 55,71 | | 3,62 | 14,8733333 | | 0,39 | 0,72 | 1,51 | 3,18 | |
| 33,9 | 40,1 | 45,19 | 50,38 | | 5,52 | 14,51 | | 20,04 | 21,15 | 23,54 | 28,02 | |
| 33,67 | 39,76 | 44,29 | 49,71 | | 4,62 | 10,435 | | 1,74 | 3,05 | 5,59 | 10,31 | |
| 36,43 | 42,05 | 47,05 | 51,29 | 53,95 | -1,9 | 16,925 | | 81,13 | 131,16 | 247,29 | 538,56 | 1431,41 |
| 36,43 | 42,05 | 47,05 | 51,29 | 53,95 | -1,9 | 17,48 | | 1,81 | 4,65 | 14,27 | 52,91 | 201,14 |
| 36,38 | 42,19 | 47,05 | 51,29 | 53,95 | -1,9 | 9,35333333 | | 1,72 | 4,42 | 14,53 | 54,23 | 198,39 |
| 29,86 | 34,33 | 36,86 | 40,43 | 43,05 | -2,81 | 6,18 | | 78,04 | 125,75 | 236,53 | 487,71 | 1221,25 |
| 29,86 | 34,33 | 36,86 | 40,43 | 43,05 | -2,81 | 3,005 | | 6,56 | 16,53 | 49,98 | 168,69 | 609,13 |
| 40,86 | 47,38 | 52,33 | 55,81 | 58,23 | 3,38 | 22,205 | | 22,71 | 24,64 | 29,76 | 33,83 | 49,19 |
| 36,38 | 42,14 | 46,29 | 49,57 | 52,38 | -2,66 | 16,72 | | 0,43 | 0,82 | 1,61 | 3,32 | 6,68 |
| 41,81 | 49,38 | 54,71 | 57,52 | 60,19 | 5,76 | 17,0133333 | | 0,39 | 0,72 | 1,55 | 3,13 | 6,19 |
| 34,9 | 39,76 | 46,71 | 50,71 | 53,39 | 7,04 | 16,03 | | 22,49 | 23,25 | 25,65 | 30,25 | 39,88 |
| 34,52 | 39,52 | 44,9 | 50,52 | 53,9 | 5,23 | 11,045 | | 1,74 | 3,04 | 5,6 | 10,22 | 19,6 |
| | | | | | | | | | | | | |
| 48,6 | 59 | 64 | 57,4 | | -33 | -26,933333 | | 4,08 | 8,23 | 22,1 | 68,67 | |
| 49,2 | 56,4 | 63 | 57,6 | | -34 | -29,133333 | | 0,55 | 1,44 | 4,32 | 14,21 | |
| 49 | 58,6 | 62,8 | 58,8 | | -34,2 | -31,066667 | | 0,55 | 1,37 | 4,08 | 13,35 | |
| 46,6 | 60 | 64,4 | 67,2 | | -29,2 | -25,333333 | | 3,35 | 6,91 | 17,05 | 51,31 | |
| 46,6 | 59,8 | 64,4 | 67,2 | | -29,2 | -26,4 | | 2,69 | 5,61 | 15 | 46,69 | |

# Appendix B

# Code implementations

## B.1 Loaders

### B.1.1 3DTCGS Loader

index.js

```
const path = require('path');
const fs = require('fs-extra');

const GestureSet =
↪  require('../../../../framework/gestures/gesture-set').GestureSet;
const GestureClass =
↪  require('../../../../framework/gestures/gesture-class').GestureClass;
const StrokeData =
↪  require('../../../../framework/gestures/stroke-data').StrokeData;
const { Path, Stroke } = require('../../../../framework/gestures/stroke-data');
const { Point3D } = require('../../../../framework/gestures/Point');

function loadDataset(name, datasetPath, sensorId, datasetId) {

    let gestureSet = new GestureSet(name);
    let users = fs.readdirSync(datasetPath)
                  .filter(f => fs.lstatSync(path.join(datasetPath,
                  ↪  f)).isDirectory())
                  .sort();

    let firstUser = path.join(datasetPath, users[0]);
    let gestureNames = fs.readdirSync(firstUser)
                         .filter(f => f.endsWith('.csv'))
                         .map(f => f.replace('.csv',''))
                         .sort();

    let gestureMap = {};
    gestureNames.forEach((gesture, index) => {
        const fullName = datasetId ? `${gesture}_${datasetId}` : gesture;
        const gc = new GestureClass(fullName, index);
        gestureMap[gesture] = gc;
```

```
        gestureSet.addGestureClass(gc);
    });

    let globalSampleId = 1;

    // Load samples
    users.forEach((userFolder) => {
        const folderPath = path.join(datasetPath, userFolder);

        gestureNames.forEach(gesture => {
            const csvPath = path.join(folderPath, `${gesture}.csv`);
            if (!fs.existsSync(csvPath)) return;

            const rows = fs.readFileSync(csvPath, 'utf8')
                           .split(/\r?\n/)
                           .map(line => line.trim())
                           .filter(line => line.length > 0);

            const strokeData = new StrokeData("User"+userFolder, globalSampleId);
            globalSampleId++;

            const label = "Palm_default";
            const pathObj = new Path(label);
            const stroke = new Stroke(0);
            rows.forEach(line => {
                const cols = line.split(/\s+/);

                if (cols.length < 3) return;

                const x = parseFloat(cols[0]);
                const y = parseFloat(cols[1]);
                const z = parseFloat(cols[2]);
                const t = cols[3] ? parseFloat(cols[3]) : 0;

                if (!Number.isFinite(x) || !Number.isFinite(y) ||
                ↪  !Number.isFinite(z)) {
                    console.warn("Skipping invalid row:", cols);
                    return;
                }

                stroke.addPoint(new Point3D(x, y, z, t));
            });

            pathObj.addStroke(stroke);
            strokeData.addPath(label, pathObj);

            gestureMap[gesture].addSample(strokeData);
        });
    });
    return gestureSet;
}
```

```

module.exports = { loadDataset };
```

### B.1.2 SHREC2019 Loader

index.js

```
const path = require('path');
const fs = require('fs-extra');

const GestureSet = require('../../../../framework/gestures/gesture-set').GestureSet;
const GestureClass = require('../../../../framework/gestures/gesture-class').GestureClass;
const StrokeData = require('../../../../framework/gestures/stroke-data').StrokeData;
const { Path, Stroke } = require('../../../../framework/gestures/stroke-data');
const { Point2D, Point3D, PointND } = require('../../../../framework/gestures/Point');

function loadDataset(name, datasetPath, sensorId, datasetId, sensorPointsNames) {
    let gestureSet = new GestureSet(name);
    let dirPath = datasetPath;
    let globalSampleId = 1;


    const folders = fs.readdirSync(dirPath)
        .filter(u => fs.lstatSync(path.join(dirPath, u)).isDirectory())
        .sort();

    if (folders.length === 0)
        throw new Error(`No folders found in ${dirPath}`);


    const firstFolder = path.join(dirPath, folders[0]);
    const gestureNames = fs.readdirSync(firstFolder)
        .filter(f => f.endsWith(".json"))
        .map(f => f.replace(".json", ""))
        .sort();

    let gestureClassIndexMap = {};
    gestureNames.forEach((gesture, idx) => {
        const fullName = addIdentifier(gesture, datasetId);
```

```
        const gestureClass = new GestureClass(fullName, idx);
        gestureClassIndexMap[gesture] = gestureClass;
        gestureSet.addGestureClass(gestureClass);
    });



    folders.forEach((folderName, folderIndex) => {
        const folderPath = path.join(dirPath, folderName);

        const USER_NAME = `User_${folderName}`;

        gestureNames.forEach(gesture => {
            const gestureFile = path.join(folderPath, `${gesture}.json`);
            if (!fs.existsSync(gestureFile)) return;

            const rawStrokeData = JSON.parse(fs.readFileSync(gestureFile));

            // Create strokeData and explicitly pass USER_NAME as the first
            ↪ argument
            let strokeData = new StrokeData(USER_NAME, globalSampleId);
            globalSampleId++;

            Object.entries(rawStrokeData.paths).forEach(([pathLabel, pathContent])
            ↪ => {
                const label = addIdentifier(pathLabel, sensorId);
                const pathObj = new Path(label);
                let stroke = new Stroke(0);

                pathContent.strokes.forEach(point => {
                    stroke.addPoint(new Point3D(point.x, point.y, point.z,
                    ↪ point.t));
                });

                pathObj.addStroke(stroke);
                strokeData.addPath(label, pathObj);
            });

            gestureClassIndexMap[gesture].addSample(strokeData);
        });
    });
    return gestureSet;
}

function addIdentifier(name, identifier) {
    return identifier ? `${name}_${identifier}` : name;
}

module.exports = { loadDataset };

```

## B.2 Ensemble learning

### B.2.1 Majority Voting

index.js

```
// Base class required by the QuantumLeap framework for dynamic recognizers
const AbstractDynamicRecognizer =

  ↪ require('../../../../framework/modules/recognizers/dynamic/abstract-dynamic-recognizer')
    .AbstractDynamicRecognizer;

// Majority vote ensemble recognizer (top-1 vote per base recognizer)
class Recognizer extends AbstractDynamicRecognizer {
  static name = "MajorityVoting";

  constructor(options, dataset) {
    super();

    // Per-recognizer training subset (undefined/empty => train on all gestures)
    this.trainingGestures = [];

    // List of instantiated base recognizers
    this.recognizers = [];

    // Initialize base recognizers + keep their optional training gesture filters
    options.recognizers.forEach((recognizer) => {
      this.trainingGestures.push(recognizer.additionalSettings?.trainingGestures);
      this.recognizers.push(new recognizer.module(recognizer.moduleSettings));
    });

    // Load templates/samples from dataset (if provided)
    if (dataset !== undefined) {
      dataset.getGestureClasses().forEach((gesture) => {
        gesture.getSamples().forEach((sample) => {
          this.addGesture(gesture.name, sample);
        });
      });
    }
  }

  addGesture(name, sample) {
    // Forward training sample to each base recognizer (only if allowed by its
    ↪  training set)
    this.recognizers.forEach((recognizer, index) => {
      if (isInTrainingSet(name, this.trainingGestures[index])) {
        recognizer.addGesture(name, sample);
      }
    });
  }

```

```
44    /**
45     * Majority Voting with Confidence Tie-breaking.
46     * 1. Each recognizer votes for its top candidate.
47     * 2. The class with the most votes wins.
48     * 3. If there is a tie in votes, the class with the highest individual
       ↪  confidence score wins.
49     */
50    recognize(sample) {
51      // Accumulate time from base recognizers (if they report it)
52      let totalTime = 0;
53
54      // Store top-1 result of each base recognizer: { name, score, time? }
55      const individualResults = [];
56
57      // 1) Collect the top prediction from each base recognizer
58      for (const recognizer of this.recognizers) {
59        const result = recognizer.recognize(sample);
60        totalTime += result.time ?? 0;
61        individualResults.push(result); // result contains { name, score }
62      }
63
64      // No base recognizers configured
65      if (individualResults.length === 0) {
66        return { name: "", score: 0.0, time: totalTime };
67      }
68
69      // 2) Aggregate votes (Hybrid Voting Logic)
70      // voteData[className] = { count: number of votes, maxScore: best confidence
       ↪  among voters }
71      const voteData = {};
72      let maxVotes = 0;
73
74      for (const result of individualResults) {
75        const p = result.name;
76        const score = result.score;
77
78        // Ignore empty / null predictions
79        if (p !== null && p !== "") {
80          // Initialize candidate bucket
81          if (!voteData[p]) {
82            voteData[p] = { count: 0, maxScore: -Infinity };
83          }
84
85          // Add one vote
86          voteData[p].count += 1;
87
88          // Track best confidence score observed for this class
89          if (score > voteData[p].maxScore) {
90            voteData[p].maxScore = score;
91          }
92
```

```
        // Track maximum vote count for tie detection
        if (voteData[p].count > maxVotes) {
          maxVotes = voteData[p].count;
        }
      }
    }

    // 3) Handle tied candidates (same number of votes)
    let tiedCandidates = [];
    for (const p in voteData) {
      if (voteData[p].count === maxVotes) {
        tiedCandidates.push({
          name: p,
          maxScore: voteData[p].maxScore
        });
      }
    }

    // 4) Pick the winner based on max confidence among tied candidates
    let finalPrediction = "";
    let finalScore = 0.0;
    let maxConfidence = -Infinity;

    for (const candidate of tiedCandidates) {
      if (candidate.maxScore > maxConfidence) {
        maxConfidence = candidate.maxScore;
        finalPrediction = candidate.name;
        finalScore = candidate.maxScore;
      }
    }

    // Return ensemble decision + aggregated time
    return {
      name: finalPrediction,
      score: finalScore,
      time: totalTime,
    };
  }

  removeGesture(name) {
    // Remove gesture from each base recognizer (only if it was part of its
    ↪  training set)
    this.recognizers.forEach((recognizer, index) => {
      if (isInTrainingSet(name, this.trainingGestures[index])) {
        recognizer.removeGesture(name);
      }
    });
  }

  toString() {
    // Concatenate base recognizer identifiers for logging / reporting
```

```
    return this.recognizers
      .map((r) => (typeof r.toString === "function" ? r.toString() : String(r)))
      .join(",");
  }
}

// Helper: if no training set is specified, accept all gestures
function isInTrainingSet(gesture, trainingSet) {
  if (trainingSet === undefined || trainingSet === null || trainingSet.length ===
  ↪  0) {
    return true;
  }
  return trainingSet.indexOf(gesture) !== -1;
}

module.exports = Recognizer;
```

### B.2.2 Borda Voting

index.js

```
// Base class required by the QuantumLeap framework for dynamic recognizers
const AbstractDynamicRecognizer =

  ↪  require('../../../../framework/modules/recognizers/dynamic/abstract-dynamic-recognizer')
    .AbstractDynamicRecognizer;

// Borda ensemble recognizer:
// - Each base recognizer provides a *ranking* (via per-class similarity scores).
// - Classes receive Borda points based on rank (top gets most points).
// - Final prediction is the class with the highest total Borda points.
class Recognizer extends AbstractDynamicRecognizer {
  static name = "Borda";

  constructor(options, dataset) {
    super();

    // Per-recognizer training subset (undefined/empty => train on all gestures)
    this.trainingGestures = [];

    // List of instantiated base recognizers
    this.recognizers = [];

    // Initialize base recognizers + keep their optional training gesture filters
    options.recognizers.forEach((recognizer) => {
      this.trainingGestures.push(recognizer.additionalSettings?.trainingGestures);
      this.recognizers.push(new recognizer.module(recognizer.moduleSettings));
    });

    // Load templates/samples from dataset (if provided)
```

```
    if (dataset !== undefined) {
      dataset.getGestureClasses().forEach((gesture) => {
        gesture.getSamples().forEach((sample) => {
          this.addGesture(gesture.name, sample);
        });
      });
    }
  }

  addGesture(name, sample) {
    // Forward training sample to each base recognizer (only if allowed by its
    ↪ training set)
    this.recognizers.forEach((recognizer, index) => {
      if (isInTrainingSet(name, this.trainingGestures[index])) {
        recognizer.addGesture(name, sample);
      }
    });
  }

  /**
   * Borda voting over rankings produced by each recognizer.
   * For each recognizer:
   *  - rank classes by similarity (higher = better)
   *  - assign Borda points: (m - rank), rank starts at 1
   */
  recognize(sample) {
    // Accumulate time from base recognizers (if they report it)
    let totalTime = 0;

    // 1) Collect similarities from each base recognizer
    // allSimsPerRecognizer: list of per-class similarity maps produced by each
    ↪ recognizer
    const allSimsPerRecognizer = []; // array of { className: similarity }

    // Union of all classes that appear in at least one recognizer's similarity
    ↪ map
    const classUnion = new Set();

    for (const recognizer of this.recognizers) {
      // Borda requires access to full similarity distribution, not only top-1
      if (typeof recognizer.recognizeAllSimilarities !== "function") {
        throw new Error(
          "Borda ensemble requires base recognizers to implement
          ↪ recognizeAllSimilarities(sample)."
        );
      }

      // Expected format: { similarities: {className: value, ...}, time?: number }
      const allSimilarities = recognizer.recognizeAllSimilarities(sample);
      totalTime += allSimilarities.time ?? 0;

```

```
      const sims = allSimilarities.similarities ?? {};
      allSimsPerRecognizer.push(sims);

      // Track all classes present across all recognizers
      for (const c in sims) classUnion.add(c);
    }

    // Materialize the union set into a stable array for ranking
    const classes = Array.from(classUnion);
    const m = classes.length; // number of candidates in the Borda election

    // No candidates at all => no match
    if (m === 0) {
      return { name: "", score: 0.0, time: totalTime };
    }

    // 2) Aggregate Borda points
    // classScores[className] = accumulated Borda points across recognizers
    const classScores = {};
    for (const c of classes) classScores[c] = 0;

    for (const sims of allSimsPerRecognizer) {
      // Rank ALL classes for this recognizer.
      // Missing classes get -Infinity => placed at the end => 0 Borda points.
      const ranked = classes
        .map((c) => ({ c, v: sims[c] ?? -Infinity }))
        .sort((a, b) => b.v - a.v);

      // Assign points according to rank:
      // idx=0 => rank=1 => points=m-1 (maximum)
      // idx=m-1 => rank=m => points=0 (minimum)
      for (let idx = 0; idx < ranked.length; idx++) {
        const className = ranked[idx].c;
        const rank = idx + 1; // 1..m
        const points = m - rank; // top: m-1, bottom: 0
        classScores[className] += points;
      }
    }

    // 3) Pick best class (maximum accumulated points)
    let bestClass = "";
    let bestScore = -Infinity;

    for (const className in classScores) {
      if (classScores[className] > bestScore) {
        bestScore = classScores[className];
        bestClass = className;
      }
    }

    // Safety fallback (should not happen if m>0)
```

```
    if (bestClass === "") {
      return { name: "", score: 0.0, time: totalTime };
    }

    // Optional: normalize to [0,1]
    // Max per recognizer is (m-1), so total max is (m-1)*numRecognizers.
    const maxPossible = (m - 1) * (this.recognizers.length || 1);
    const normalizedScore = maxPossible > 0 ? bestScore / maxPossible : 0.0;

    // Return Borda winner + normalized score + aggregated time
    return {
      name: bestClass,
      score: normalizedScore,
      time: totalTime,
    };
  }

  removeGesture(name) {
    // Remove gesture from each base recognizer (only if it was part of its
    ↪  training set)
    this.recognizers.forEach((recognizer, index) => {
      if (isInTrainingSet(name, this.trainingGestures[index])) {
        recognizer.removeGesture(name);
      }
    });
  }

  toString() {
    // Concatenate base recognizer identifiers for logging / reporting
    return this.recognizers
      .map((r) => (typeof r.toString === "function" ? r.toString() : String(r)))
      .join(",");
  }
}

// Helper: if no training set is specified, accept all gestures
function isInTrainingSet(gesture, trainingSet) {
  if (trainingSet === undefined || trainingSet === null || trainingSet.length ===
  ↪  0) {
    // Keep all gestures
    return true;
  }
  // Keep only gestures in the training set
  return trainingSet.indexOf(gesture) !== -1;
}

module.exports = Recognizer;

```

### B.2.3 Weighted Dirichlet

index.js

```
// Base class required by the QuantumLeap framework for dynamic recognizers
const AbstractDynamicRecognizer =
↪  require('../../../../framework/modules/recognizers/dynamic/abstract-dynamic-recognizer').A

// High-resolution timer utilities (currently not used in this file, but kept for
↪  potential profiling)
const { performance } = require('perf_hooks');

// Utility helper from the framework (currently not used in this file, but kept for
↪  consistency with other recognizers)
const { parsePointsNames } = require('../../../../framework/utils');

// Dirichlet ensemble recognizer:
// - Collects full similarity distributions from multiple base recognizers
// - Normalizes each recognizer's similarities into probabilities
// - Samples a weight vector w ~ Dirichlet(alpha) where alpha comes from recognizer
↪  weights
// - Combines distributions as a weighted sum and returns the highest scoring
↪  class
class Recognizer extends AbstractDynamicRecognizer {
        static name = "Dirichlet";

    constructor(options, dataset) {
    super();

        // Per-recognizer training subset (undefined/empty => train on all
        ↪  gestures)
        this.trainingGestures = [];

        // List of instantiated base recognizers
    this.recognizers = [];

        // Per-recognizer Dirichlet alpha parameters (or default fallback)
    this.weights = [];

                // Instantiate base recognizers and store their training gesture
                ↪  filters + weights
                options.recognizers.forEach(recognizer => {
                        this.trainingGestures.push(recognizer.additionalSettings.trainingGestu
                        this.recognizers.push(new
                        ↪  recognizer.module(recognizer.moduleSettings));
                        this.weights.push(recognizer.additionalSettings.weight);
                });

                // Load templates/samples from dataset (if provided)
                if (dataset !== undefined) {
                        dataset.getGestureClasses().forEach((gesture) => {
                                gesture.getSamples().forEach(sample => {
```

```
                                        this.addGesture(gesture.name, sample);
                                }
                                );
                        });
                }
        }


        addGesture(name, sample) {
                // Forward training sample to each base recognizer (only if allowed by its training set)
                this.recognizers.forEach((recognizer, index) => {
                        if (isInTrainingSet(name, this.trainingGestures[index])) {
                                recognizer.addGesture(name, sample);
                        }
                });
        }

    recognize(sample){
                // Per-recognizer probability distributions after normalization
        const distribution = [];

                // Accumulate time from base recognizers (if they report it)
        let totalTime = 0;

                // Collect similarities from each base recognizer and normalize them into probabilities
        for(const recognizer of this.recognizers){
            const allSimilarities = recognizer.recognizeAllSimilarities(sample);
            totalTime += allSimilarities.time;

                        // Normalize similarity map => probability map
            distribution.push(this.normalize(allSimilarities.similarities));
        }

        // Set the Dirichlet alpha parameters (fallback to 1.0 when undefined)
        const alpha = this.weights.map(w => w ?? 1.0);

        // Sample a simplex weight vector (one weight per recognizer) from Dirichlet(alpha)
        const w = this.sampleDirichlet(alpha);

        // Aggregate weighted class probabilities across recognizers
        const classScores = {};

        for (let i = 0; i < distribution.length; i++){
            const probs = distribution[i];
            for(const className in probs){
                if(!classScores[className]){
                    classScores[className] = 0;
                }
```

```
                                // Weighted mixture of recognizer distributions
                classScores[className] += w[i] * probs[className];
            }
        }

                // Select class with maximum aggregated probability
        let bestClass ="";
        let bestScore = -Infinity;

        for (const className in classScores){
            if(classScores[className] > bestScore){
                bestScore = classScores[className];
                bestClass = className;
            }
        }

                // No match fallback
        if(bestClass === ""){
            return {name: "", score: 0.0, time: totalTime};
        }

                // Return final prediction + aggregated time
        return {name: bestClass,
                score: bestScore,
                time: totalTime
        };
    }

    normalize(similarities){
                // Convert raw similarities into a probability distribution by
                ↪  sum-normalization
        const probs = {};
        let sum = 0;

        for(const c in similarities){
            sum += similarities[c]
        }

                // If sum is zero, fallback to uniform distribution over classes
        if(sum === 0){
            const n = Object.keys(similarities).length;
            for(const c in similarities) {
                probs[c] = 1/n;
            }
            return probs
        }

                // Standard normalization: p(c) = sim(c) / sum(sim)
        for(const c in similarities) {
            probs[c] = similarities[c] / sum;
        }
```

```
139
140         return probs;
141
142     }
143
144     sampleGamma(alpha) {
145                 // Gamma sampler used to build a Dirichlet sample:
146                 // Dirichlet(alpha) can be sampled by drawing independent
                    ↪  Gamma(alpha_i, 1) then normalizing.
147         let d, c, x, v, u;
148
149         // Todo check if other value start for alpha has a better impact
150        alpha += 1;
151
152                 // Marsaglia and Tsang method (commonly used for Gamma sampling
                    ↪  when alpha is not too small)
153         d = alpha - 1 / 3;
154         c = 1 / Math.sqrt(9 * d);
155
156         while (true) {
157             do {
158                 x = Math.random();
159                 v = Math.pow(1 + c * (Math.log(x / (1 - x))), 3);
160             } while (v <= 0);
161
162             u = Math.random();
163
164                         // Squeeze test
165             if (u < 1 - 0.0331 * Math.pow(x, 4)) return d * v;
166
167                         // Log test
168             if (Math.log(u) < 0.5 * Math.pow(x, 2) + d * (1 - v + Math.log(v)))
169                 return d * v;
170         }
171     }
172
173     sampleDirichlet(alpha){
174                 // Sample each component with Gamma, then normalize to sum to 1
175         const samples = alpha.map(a => this.sampleGamma(a));
176         const sum = samples.reduce((a,b) => a + b, 0);
177         return samples.map(v => v / sum);
178     }
179
180
181         removeGesture(name) {
182                 // Remove gesture from each base recognizer (only if it was part of
                    ↪  its training set)
183                 this.recognizers.forEach((recognizer, index) => {
184                         if (isInTrainingSet(name, this.trainingGestures[index])) {
185                                 recognizer.removeGesture(name);
186                         }
```

```
                });
        }

    toString() {
                // String representation (intended to list base recognizers)
    return ("",this.recognizers.map(r >= r.toString()).join(","));
  }
}

// Helper: if no training set is specified, accept all gestures
function isInTrainingSet(gesture, trainingSet) {
        if (trainingSet === undefined || trainingSet === null || trainingSet.length
        ↪  === 0) {
                // Keep all gestures
                return true;
        } else {
                // Keep only gestures in the training set
                return trainingSet.indexOf(gesture) !== -1;
        }
}

function convert(sample, selectedPoints) {
        // Convert a sample structure into a flat list of points grouped by
        ↪  articulation/stroke
        // (Typically used by point-cloud recognizers; kept here though Point is
        ↪  not imported in this file)
        let points = [];
        selectedPoints.forEach((articulation, articulationID) => {
                sample.paths[articulation].strokes.forEach((stroke, strokeId) => {
                        stroke.points.forEach((point) => {
                                // If multipoint, one stroke per articulation,
                                ↪  otherwise, keep original strokes
                                let index = selectedPoints.length > 1 ?
                                ↪  articulationID : strokeId;
                                points.push(new Point(point.x, point.y, point.z,
                                ↪  index));
                        });
                });
        });
        return points;
}

module.exports = Recognizer;

```

### B.2.4 Stacking by final class predictio

index.js

```
1
2  // Base class required by the QuantumLeap framework for dynamic recognizers
3  const AbstractDynamicRecognizer =
   ↪  require('../../../../framework/modules/recognizers/dynamic/abstract-dynamic-recognizer').A
4
5  // Machine learning library used for the meta-learner (Decision Tree)
6  const ml = require('machine_learning');
7
8  // High-resolution timer utilities used to measure recognition time
9  const { performance } = require('perf_hooks');
10
11 // Stacking ensemble (by class):
12 // - Base recognizers produce top-1 class predictions
13 // - These predicted class labels are used as categorical features
14 // - A meta-learner (Decision Tree) is trained to map base predictions -> final
   ↪  class
15 class Recognizer extends AbstractDynamicRecognizer {
16   static name = "Stacking_by_class";
17
18   constructor(options, dataset) {
19     super();
20
21     // Per-recognizer training subset (undefined/empty => train on all gestures)
22     this.trainingGestures = [];
23
24     // List of instantiated base recognizers
25     this.recognizers = [];
26
27     // Meta-learner model (Decision Tree) trained on base recognizer outputs
28     this.metaLearner = null;
29
30     // Stored training items for meta learner: { sample, label }
31     this.metaTrainingData = [];
32
33     // Set of all gesture class names seen so far (used for score normalization)
34     this.gestureNames = new Set();
35
36     // Flag indicating whether the meta-learner needs retraining
37     this.isMetaLearnerTrained = false;
38
39     // Minimal number of samples required before training the meta-learner
40     this.minSamplesForTraining = 10;
41
42     // Instantiate each base recognizer and store its optional training gesture
   ↪  filter
43     options.recognizers.forEach(recognizer => {
44       this.trainingGestures.push(recognizer.additionalSettings?.trainingGestures);
45       this.recognizers.push(new recognizer.module(recognizer.moduleSettings));
46     });
47
48     // Load templates/samples from dataset (if provided)
```

```
    if (dataset !== undefined) {
      dataset.getGestureClasses().forEach((gesture) => {
        gesture.getSamples().forEach(sample => {
          this.addGesture(gesture.name, sample);
        });
      });
    }
  }

  addGesture(name, sample) {
    // Track known class labels for scoring / normalization
    this.gestureNames.add(name);

    // Forward training sample to each base recognizer (only if allowed by its
    ↪  training set)
    this.recognizers.forEach((recognizer, index) => {
      if (isInTrainingSet(name, this.trainingGestures[index])) {
        recognizer.addGesture(name, sample);
      }
    });

    // Add sample to meta-learner dataset (raw sample + ground-truth class)
    this.metaTrainingData.push({
        sample: sample,
        label: name
    });

    // New data invalidates the current meta-learner
    this.isMetaLearnerTrained = false;
  }

  // Extract stacking features from a sample:
  // - For each base recognizer, take its predicted class name (top-1)
  // - The meta learner will use these names as categorical features
  // Updated to use the new object return structure
  extractMetaFeatures(sample) {
    const features = [];
    let time = 0;

    for (const recognizer of this.recognizers) {
      try {
        const result = recognizer.recognize(sample);

        // Accumulate time from base recognizers
        time += result.time || 0;

        // Feature = predicted class label (or empty string if no match)
        features.push(result && result.name ? result.name : "");
      } catch (err) {
        // If a base recognizer fails, keep placeholder feature
        console.error("Recognizer failed during feature extraction", err);
```

```
        features.push("");
      }
    }

    // Return both feature vector and time cost of base predictions
    return { features, time };
  }

  // Train Decision Tree meta-learner using stored metaTrainingData
  trainMetaLearner() {
    // Not enough samples yet => cannot train
    if (this.metaTrainingData.length < this.minSamplesForTraining) return false;

    // Already trained and no new data => ready
    if (this.isMetaLearnerTrained) return true;

    try {
      const trainingDataPoints = [];
      const labels = [];

      for (const item of this.metaTrainingData) {
        // Convert raw sample into stacking features (base recognizer outputs)
        // Accessing the .features property from the returned object
        const result = this.extractMetaFeatures(item.sample);
        trainingDataPoints.push(result.features);
        labels.push(item.label);
      }

      // Initialize Decision Tree classifier
      this.metaLearner = new ml.DecisionTree({
        data: trainingDataPoints,
        result: labels
      });

      // Train / build the tree
      this.metaLearner.build();

      // Mark as trained
      this.isMetaLearnerTrained = true;
      return true;
    } catch (err) {
      console.error('Meta-learner training failed:', err);
      return false;
    }
  }

  recognize(sample) {
    // Measure total classification time (meta + base feature extraction time)
    const startClassify = performance.now();

    // Ensure meta learner is trained (lazy training on first recognize call)
```

```
    const metaLearnerReady = this.trainMetaLearner();


    if (metaLearnerReady && this.metaLearner) {
      try {
        // 1) Compute stacking features from base recognizers
        const result = this.extractMetaFeatures(sample);

        // 2) Meta-learner predicts class from base predictions
        const prediction = this.metaLearner.classify(result.features);

        // 3) Calculate time: base extraction time + DT classification time
        const totalTime = (result.time || 0) + (performance.now() - startClassify);

        let predictedClass = "";
        let leafVotes = 0;

        // Some DecisionTree implementations may return an object with votes per
        ↪  class
        if (typeof prediction === 'object' && prediction !== null) {
          const keys = Object.keys(prediction);
          if (keys.length > 0) {
            predictedClass = keys[0];
            leafVotes = prediction[predictedClass];
          }
        // Or return a direct string label
        } else if (typeof prediction === 'string') {
          predictedClass = prediction;
          leafVotes = 1;
        }

        if (predictedClass) {
          // Compute a heuristic confidence score based on votes relative to data
          ↪  density
          const numClasses = this.gestureNames.size || 1;
          const totalSamples = this.metaTrainingData.length;
          const averageSamplesPerClass = totalSamples / numClasses;

          const calculatedScore = averageSamplesPerClass > 0
            ? leafVotes / averageSamplesPerClass
            : 0;

          // Return predicted class + derived score + total time
          return {
            name: predictedClass.split(':')[0].trim(),
            score: calculatedScore,
            time: totalTime
          };
        }
      } catch (err) {
        // Any runtime issue in feature extraction or meta inference
```

```
        console.error("Inference error:", err);
      }
    }

    // Fallback when meta-learner cannot be used
    console.error("The meta trainer is not trained yet");
  }

  removeGesture(name) {
    // Remove class from known labels set
    this.gestureNames.delete(name);

    // Remove gesture templates from base recognizers (only if allowed by training set)
    this.recognizers.forEach((recognizer, index) => {
      if (isInTrainingSet(name, this.trainingGestures[index])) {
        recognizer.removeGesture(name);
      }
    });

    // Remove training samples for this label from the meta dataset
    this.metaTrainingData = this.metaTrainingData.filter(d => d.label !== name);

    // Meta learner must be retrained after deletion
    this.isMetaLearnerTrained = false;
  }

  toString() {
    // String representation including base recognizers for logging / reporting
    const baseRecognizers = this.recognizers
      .map((r) => (typeof r.toString === "function" ? r.toString() : String(r)))
      .join(",");
    return `Stacking_by_class[DecisionTree](${baseRecognizers})`;
  }
}

// Helper: if no training set is specified, accept all gestures
function isInTrainingSet(gesture, trainingSet) {
  if (trainingSet === undefined || trainingSet === null || trainingSet.length === 0) {
    return true;
  }
  return trainingSet.indexOf(gesture) !== -1;
}

module.exports = Recognizer;

```

## B.3 Example of recognizerAllSimilarities

### B.3.1 Example on $P3+ recognizer

Here only the recognizeAllSimilarities and the recognize have been adapted. `p3dollarplus.js`

```
1
2  /**
3   * The $P+ Point-Cloud Recognizer (JavaScript version)
4   *
5   *  Radu-Daniel Vatavu, Ph.D.
6   *  University Stefan cel Mare of Suceava
7   *  Suceava 720229, Romania
8   *  vatavu@eed.usv.ro
9   *
10  * The academic publication for the $P+ recognizer, and what should be
11  * used to cite it, is:
12  *
13  *     Vatavu, R.-D. (2017). Improving gesture recognition accuracy on
14  *     touch screens for users with low vision. Proceedings of the ACM
15  *     Conference on Human Factors in Computing Systems (CHI '17). Denver,
16  *     Colorado (May 6-11, 2017). New York: ACM Press, pp. 4667-4679.
17  *     https://dl.acm.org/citation.cfm?id=3025941
18  *
19  * This software is distributed under the "New BSD License" agreement:
20  *
21  * Copyright (C) 2017-2018, Radu-Daniel Vatavu and Jacob O. Wobbrock. All
22  * rights reserved. Last updated July 14, 2018.
23  *
24  * Redistribution and use in source and binary forms, with or without
25  * modification, are permitted provided that the following conditions are met:
26  *    * Redistributions of source code must retain the above copyright
27  *      notice, this list of conditions and the following disclaimer.
28  *    * Redistributions in binary form must reproduce the above copyright
29  *      notice, this list of conditions and the following disclaimer in the
30  *      documentation and/or other materials provided with the distribution.
31  *    * Neither the name of the University Stefan cel Mare of Suceava, nor the
32  *      names of its contributors may be used to endorse or promote products
33  *      derived from this software without specific prior written permission.
34  *
35  * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS
36  * IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
37  * THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
38  * PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL Radu-Daniel Vatavu OR Lisa Anthony
39  * OR Jacob O. Wobbrock BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
40  * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT
41  * OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
42  * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
    ↪  CONTRACT,
43  * STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY
    ↪  WAY
```

```
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
**/

const { performance } = require('perf_hooks');

//
// Point class
//
function Point(x, y, z, id, angle = 0.0) {
        this.X = x;
        this.Y = y;
        this.Z = z;
        this.ID = id;
        this.Angle = angle; // normalized turning angle, $P+
}
//
// PointCloud class: a point-cloud template
//
function PointCloud(name, points) // constructor
{
        this.Name = name;
        this.Points = Resample(points, NumPoints);
        this.Points = Scale(this.Points);
        this.Points = TranslateTo(this.Points, Origin);
        this.Points = ComputeNormalizedTurningAngles(this.Points); // $P+
}
//
// Result class
//
function Result(name, score, ms) // constructor
{
        this.Name = name;
        this.Score = score;
        this.Time = ms;
}
//
// PDollarPlusRecognizer constants
//
var NumPoints = 32;
const Origin = new Point(0, 0, 0, 0);
//
// PDollarPlusRecognizer class
//
function P3DollarPlusRecognizer(numPoints) // constructor
{
        this.PointClouds = new Array();
        NumPoints = numPoints;
        //
        // The $P+ Point-Cloud Recognizer API begins here -- 4 methods:
        ↪  Recognize(), AddGesture(), RemoveGesture(), DeleteUserGestures()
```

```
94          //
95          this.Recognize = function (points) {
96                  const t0 = performance.now();
97                  const similarities =
                    ↪  this.RecognizeAllSimilarities(points).similarities;
98
99                  let bestClass = "";
100                 let bestScore = 0.0
101
102                 for(const className in similarities){
103                         const score = similarities[className];
104                         if(score > bestScore){
105                                 bestClass = className;
106                                 bestScore = score;
107                         }
108                 }
109
110                 const t1 = performance.now();
111                 return bestClass === "" ? new Result("No match.", 0.0, t1-t0) : new
                    ↪  Result(bestClass,bestScore,t1-t0);
112         }
113         this.RecognizeAllSimilarities = function (points) {
114                 var t0 = performance.now();
115                 var candidate = new PointCloud("", points);
116                 const bestDistance = {};
117                 for (var i = 0; i < this.PointClouds.length; i++) // for each
                    ↪  point-cloud template
118                 {
119                         var d = Math.min(
120                                 CloudDistance(candidate.Points,
                                    ↪  this.PointClouds[i].Points, +Infinity),
121                                 CloudDistance(this.PointClouds[i].Points,
                                    ↪  candidate.Points, +Infinity)
122                         );
123                         const className = this.PointClouds[i].Name;
124                         if(bestDistance[className] === undefined || d <
                            ↪  bestDistance[className]){
125                                 bestDistance[className] = d;
126                         }
127                 }
128                 const similarities = {};
129                 for(const className in bestDistance){
130                         const d = bestDistance[className];
131                         similarities[className] = 1.0/d ;
132                 }
133                 var t1 = performance.now();
134                 return {similarities, time: t1-t0};
135         }
136         this.AddGesture = function (name, points) {
137                 this.PointClouds[this.PointClouds.length] = new PointCloud(name,
                    ↪  points);
```

```
138                 var num = 0;
139                 for (var i = 0; i < this.PointClouds.length; i++) {
140                         if (this.PointClouds[i].Name == name)
141                                 num++;
142                 }
143                 return num;
144         }
145         this.RemoveGesture = function (name) {
146                 this.PointClouds = this.PointClouds.filter(pointCloud =>
                    ↪  pointCloud.Name !== name);
147         }
148         this.DeleteUserGestures = function () {
149                 this.PointClouds.length = NumPointClouds; // clears any beyond the
                    ↪  original set
150                 return NumPointClouds;
151         }
152 }
153 //
154 // Private helper functions from here on down
155 //
156 function CloudDistance(pts1, pts2, minSoFar) // revised for $P+
157 {
158         var matched = new Array(pts1.length); // pts1.length == pts2.length
159         for (var k = 0; k < pts1.length; k++)
160                 matched[k] = false;
161         var sum = 0;
162         for (var i = 0; i < pts1.length; i++) {
163                 var index = -1;
164                 var min = +Infinity;
165                 for (var j = 0; j < pts1.length; j++) {
166                         var d = DistanceWithAngle(pts1[i], pts2[j]);
167                         if (d < min) {
168                                 min = d;
169                                 index = j;
170                         }
171                 }
172                 matched[index] = true;
173                 sum += min;
174                 if (sum >= minSoFar) return sum; // early abandoning
175         }
176         for (var j = 0; j < matched.length; j++) {
177                 if (!matched[j]) {
178                         var min = +Infinity;
179                         for (var i = 0; i < pts1.length; i++) {
180                                 var d = DistanceWithAngle(pts1[i], pts2[j]);
181                                 if (d < min)
182                                         min = d;
183                         }
184                         sum += min;
185                         if (sum >= minSoFar) return sum; // early abandoning
186                 }
```

```
187         }
188         return sum;
189 }
190 function Resample(points, n) {
191         var I = PathLength(points) / (n - 1); // interval length
192         var D = 0.0;
193         var newpoints = new Array(points[0]);
194         for (var i = 1; i < points.length && newpoints.length <= n; i++) {
195                 if (points[i].ID == points[i - 1].ID) {
196                         var d = Distance(points[i - 1], points[i]);
197                         if ((D + d) >= I) {
198                                 var qx = points[i - 1].X + ((I - D) / d) *
                                    ↪  (points[i].X - points[i - 1].X);
199                                 var qy = points[i - 1].Y + ((I - D) / d) *
                                    ↪  (points[i].Y - points[i - 1].Y);
200                                 var qz = points[i - 1].Z + ((I - D) / d) *
                                    ↪  (points[i].Z - points[i - 1].Z);
201                                 var q = new Point(qx, qy, qz, points[i].ID);
202                                 newpoints[newpoints.length] = q; // append new
                                    ↪  point 'q'
203                                 points.splice(i, 0, q); // insert 'q' at position i
                                    ↪  in points s.t. 'q' will be the next i
204                                 D = 0.0;
205                         }
206                         else D += d;
207                 }
208         }
209         if (newpoints.length == n - 1) // sometimes we fall a rounding-error short
            ↪  of adding the last point, so add it if so
210                 newpoints[newpoints.length] = new Point(points[points.length -
                    ↪  1].X, points[points.length - 1].Y, points[points.length - 1].Z,
                    ↪  points[points.length - 1].ID);
211         return newpoints;
212 }
213 function Scale(points) {
214         var minX = +Infinity, maxX = -Infinity, minY = +Infinity, maxY = -Infinity,
            ↪  minZ = +Infinity, maxZ = -Infinity;
215         for (var i = 0; i < points.length; i++) {
216                 minX = Math.min(minX, points[i].X);
217                 minY = Math.min(minY, points[i].Y);
218                 minZ = Math.min(minZ, points[i].Z);
219                 maxX = Math.max(maxX, points[i].X);
220                 maxY = Math.max(maxY, points[i].Y);
221                 maxZ = Math.max(maxZ, points[i].Z);
222         }
223         var size = Math.max(maxX - minX, maxY - minY, maxZ - minZ);
224         var newpoints = new Array();
225         for (var i = 0; i < points.length; i++) {
226                 var qx = (points[i].X - minX) / size;
227                 var qy = (points[i].Y - minY) / size;
228                 var qz = (points[i].Z - minZ) / size;
```

```
                newpoints[newpoints.length] = new Point(qx, qy, qz, points[i].ID);
        }
        return newpoints;
}
function TranslateTo(points, pt) // translates points' centroid
{
        var c = Centroid(points);
        var newpoints = new Array();
        for (var i = 0; i < points.length; i++) {
                var qx = points[i].X + pt.X - c.X;
                var qy = points[i].Y + pt.Y - c.Y;
                var qz = points[i].Z + pt.Z - c.Z;
                newpoints[newpoints.length] = new Point(qx, qy, qz, points[i].ID);
        }
        return newpoints;
}
function ComputeNormalizedTurningAngles(points) // $P+
{
        var newpoints = new Array();
        newpoints[0] = new Point(points[0].X, points[0].Y, points[0].Z,
        ↪  points[0].ID); // first point
        for (var i = 1; i < points.length - 1; i++) {
                var dx = (points[i + 1].X - points[i].X) * (points[i].X - points[i
                ↪  - 1].X);
                var dy = (points[i + 1].Y - points[i].Y) * (points[i].Y - points[i
                ↪  - 1].Y);
                var dz = (points[i + 1].Z - points[i].Z) * (points[i].Z - points[i
                ↪  - 1].Z);
                var dn = Distance(points[i + 1], points[i]) * Distance(points[i],
                ↪  points[i - 1]);
                var cosangle = Math.max(-1.0, Math.min(1.0, (dx + dy + dz) / dn));
                ↪  // ensure [-1,+1]
                var angle = Math.acos(cosangle) / Math.PI; // normalized angle
                newpoints[newpoints.length] = new Point(points[i].X, points[i].Y,
                ↪  points[i].Z, points[i].ID, angle);
        }
        newpoints[newpoints.length] = new Point( // last point
                points[points.length - 1].X,
                points[points.length - 1].Y,
                points[points.length - 1].Z,
                points[points.length - 1].ID);
        return newpoints;
}
function Centroid(points) {
        var x = 0.0, y = 0.0, z = 0.0;
        for (var i = 0; i < points.length; i++) {
                x += points[i].X;
                y += points[i].Y;
                z += points[i].Z;
        }
        x /= points.length;
```

```
        y /= points.length;
        z /= points.length;
        return new Point(x, y, z, 0);
}
function PathLength(points) // length traversed by a point path
{
        var d = 0.0;
        for (var i = 1; i < points.length; i++) {
                if (points[i].ID == points[i - 1].ID)
                        d += Distance(points[i - 1], points[i]);
        }
        return d;
}
function DistanceWithAngle(p1, p2) // $P+
{
        var dx = p2.X - p1.X;
        var dy = p2.Y - p1.Y;
        var dz = p2.Z - p1.Z;
        var da = p2.Angle - p1.Angle;
        return Math.sqrt(dx * dx + dy * dy + dz * dz + da * da);
}
function Distance(p1, p2) // Euclidean distance between two points
{
        var dx = p2.X - p1.X;
        var dy = p2.Y - p1.Y;
        var dz = p2.Z - p1.Z;
        return Math.sqrt(dx * dx + dy * dy + dz * dz);
}

module.exports = {
        Point,
        P3DollarPlusRecognizer
};


```

**UNIVERSITÉ CATHOLIQUE DE LOUVAIN**
**École polytechnique de Louvain**
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**