Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Dept. of Information Security and Communication Technology

Niels Bakmann

Trail recognition for optimal training of high-performance athletes

Master's thesis in Electronic Systems Design
Supervisor: Kimmo Kansanen
July 2023

NTNU
Norwegian University of
Science and Technology

Niels Bakmann

Trail recognition for optimal training of high-performance athletes

Master's thesis in Electronic Systems Design
Supervisor: Kimmo Kansanen
July 2023

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Dept. of Information Security and Communication Technology

NTNU
Norwegian University of
Science and Technology

Abstract

During the 2022 winter Olympics, some Norwegian athletes expressed to Olympiatoppen that they felt underprepared for the competition. In response to this, Olympiatoppen approached NTNU for collaboration in creating something that could help them enhance athlete training towards a competition. Olympiatoppen suggested creating something that could identify sections of a practice trail, or Candidate, that are similar to sections of a competition trail, or Target. This defines the problem solved by this paper. To solve this problem, two different Hidden Markov Models were created and trained, before being combined into one, similar to the structure used for e.g. keyword spotting or speaker verification. One of the models was trained on the Target, while the other was trained on Candidates, which allows the different models to recognize different kinds of trails. The combined model can detect sections of a trail that is similar to the Target. The features suggested in this paper are based upon spatial geometry of trails, but a short discussion as to how and why they could be changed is also included. To model the emissions generated by the hidden states of the model(s), Gaussian Mixture Models were used, as they can approximate complex distributions, which is required to model the observed features properly. By using the combined model to evaluate different Candidates, it is possible to detect sections of a Candidate that are similar to a Target, enabling enhanced training of athletes for a given Target. Through the implementation of the system as explained in this paper, Norwegian athletes can better prepare for competitions, and Olympiatoppen can work towards one of their stated goals. In addition, Olympiatoppen could expand the number of athletes they can help through an effectivization of the work performed by a trainer.

Contents

List of Figures

Notation

Presented here is a list of the notation used in the document.

A vector is denoted as such: $\vec{\boldsymbol{v}}$

A scalar is denoted as such: s

A matrix is denoted as such: $\boldsymbol{A}$

A set is denoted as such: S=$\{s_0, s_1, s_2 \ldots, s_I\}$

A state is denoted as such: $\boldsymbol{X}$

The last element in a set/vector is denoted as such: s_I (Capital letter)

The mean of a list is denoted as such: $\overline{x}$

A HMM-model will be denoted as such: $\lambda(\boldsymbol{T}, \boldsymbol{E}, \boldsymbol{\pi})$, where $\boldsymbol{T}$ and $\boldsymbol{E}$ are the transition and emission matrix, while $\boldsymbol{\pi}$ is the initial probabilities of the model.

If a logarithm, shown as $log(\bullet)$ is not given a base, it can be assumed to be base 10, $log_{10}(\bullet)$

Definitions

This section will contain some definitions for terms used in the report.

Trail

A trail is here defined as a sequence of positions over time, assumed to either be in lat/lon/alt-format, or to be easily converted into it. In addition, there is a required temporal dimension in the sequence as well.

Target

A Target is a section of a trail that is desirable to find a good match for. An example of what can be considered a Target could be an important section of a trail used for large competitions, like the Olympics or the world cup.

Candidate

A Candidate is a trail that OLT and its athletes have available for training and that is to be considered for becoming a good match for a Target.

Ping

A ping is meant as when the Merged model emits hidden states related to the Seeker-model with a certain density within a certain length, which can be used for detecting sections of a Candidate that are similar to the Target. A ping can mathematically be indicated by $C_{i,\vec{\boldsymbol{J}}}$, where $\vec{\boldsymbol{J}}$ are the indexes of the states as described above, and C_i is Candidate i.

Abbreviations

NTNU - Norges teknisk-naturvitenskaplig universitet (Norwegian university of science and technology).

OLT - Olympiatoppen.

NIF - Norges idrettsforbund (Norwegian Olympic and Paralympic Committee and Confederation of Sports).

HMM - Hidden Markov Models.

GMM - Gaussian Mixture Model.

Lat/lon/alt - Used in place of latitude, longitude, and altitude respectively.

1 Introduction

This section will contain an introduction to the problem presented in this paper, as well as the work performed. It will also contain the aims and goals of the project. The project is named "Løypegjennkjenning og optimale forberedelser" or "Trail recognition for optimal training of high-performance athletes".

The project is written in partnership with Olympiatoppen (OLT), Norways national sports department in the Norwegian Olympic and Paralympic Comittee and Confederation of Sports (NIF).

1.1 Background

This section will contain the background information required to understand the problem, as well as the solution to the problem. As was previously mentioned, the thesis is written in cooperation with OLT. OLT has the stated goal of "strengthening high-performance sport", as well as "...maintaining Norways top position as a leading high performance sports nation..."[Olympiatoppen n.d]. This has to be kept in mind to fully understand the problem and the solution.

During the 2022 olympics in Beijing some of the Norwegian athletes felt like they hadn't prepared as well as they could, and voiced this need to Olympiatoppen. Olympiatoppen then started work on a solution to the problem in cooperation with the Norwegian University of Science and Technology(NTNU). The need is then rooted in performance and preperation, which will affect the assumptions and the desired outcome of the thesis. OLT gave some framework as to how this optimal preparation should be done: finding trails similar to a competition-trail that are available to the athletes in Norway. This allows the athletes to prepare using trails that are as similar as possible to the ones they are competing on. This method lay the grounds for a pre-project where it was shown that by considering the trails as sequences of states and comparing them using the Viterbi-algorithm, one could rank them based upon their "closeness", and also detect a sequence in a trail that could match a given trail irregardless of the structure of the sequence when it was recorded. During the pre-project some terminology was also defined: the competition trails that are desired to find matches for where called "Target(s)", while the trails that are available to the athletes or to Olympiatoppen for training where designated as "Candidates". Only one Target can be matched at a time, but the trail that is designated as the Target can change to fit the next big competition.

There where some limitations placed upon the data as well: the trails where represented in three-dimensional space using latitude, longitude and altitude, as is common in navigational applications. For the purpose of this paper, time was also introduced to the samples so that velocity and acceleration can be extracted if desireable. Although the pre-project had the goal of creating a framework for this project, it shared a lot of the same overall goals, limitations, and assumptions.

1.2 Goals

This section will contain the goals for the project. As mentioned in Sec. 1.1, OLT is responsible for maintaining Norways top position in sports-competitions. This creates the following goal: create something that allows norwegian athletes to outperform other athletes. This can be done by optimising training, which is the broad goal of this project. In addition, the project is limited in scope to distance sports, such as cross country skiing, biking, or running, not focusing on other sports such as e.g. football. To achieve this, a tool will be created that improves an athletes training towards a competition. The best solution would be to always travel to the competition trail and train on it directly, however this is not always possible, nor feasible. Combined, all this creates a goal: create something that can function as a tool that helps athletes/trainers to find trails that are similar to a competition trail (Target), and that they have available for training(Candidate). In other words: something that can search through a database and analyse several Candidates to produce a list of similar sections (pings) that are similar to the Target. A benefit to this tool would be if it could create a metric of how well the suggested pings fit the Target as well. This metric will be called "Goodness", and will be explained in Sec. 7.3.

1.3 Value Added

This section will contain the value added by the project and its work. The potential value added from this project is massive. As mentioned in section 1.1, OLT has as a goal to maintain Norway's top position in high-performance sports. By using this tool, OLT moves towards their goal by ensuring Norwegian athlete's preperation, through providing athletes and trainers with sections of Candidates that are as similar to a Target as possible. Previously this task has been given to human resources, leading to the potential for human errors to enter the resulting suggestions. In addition, the tool allows users to consider a far larger database of trails than has been previously feasible. This has a three-fold effect. Firstly, it allows OLT to effectivise their effort for high-performance athletes by moving some of the work previously performed manually to a computer, and allowing trainers to focus more on other aspects of an athelte's training. Second, it grows the population of athletes OLT can help by allowing one trainer to support more athletes, through much of the same effectiveness expressed earlier. Thirdly, it can enable greater use of the vast outdoors areas in Norway by simply uploading more Candidates to the database. When choosing trails to train on manually, the people choosing the trail has to be aware of all possible trails, which quickly grows to become unreasonably complex and tedious to do. With the tool created in this project and detailed in this report, a database of several hundred trails can be effectively used, facilitating sport and activity across greater areas of Norway. In later sections a test-database containing a total of 226 km of trails that was analysed and evaluated in < 3 s for their similarity to a Target. This scale has not been possible previously, and has the potential of adding a lot of value, both to the athletes and trainers, but also to the various actors such as landowners, sports teams, and local communities across Norway.

1.4 Prior work

This section will outline the prior work done on the subject. The project is a continuation of a pre-project completed December 2022, a short month before the start of this project. The pre-project had the aim of creating a framework for building the solution to the problem that will be defined in Sec. 2. The projects share the overall goal, and also have similar problems. The pre-project succesfully proved that the trails could be analysed and split into segments, which could be used as states in a statespace. This statespace could then be used together with the Viterbi Algorithm to evaluate the differences between two trails. The resulting method of feature extraction and considering the different trails is brought along from the pre-project. The Viterbi Algorithm will also be used in the implementation of the Hidden Markov Models(HMMs) created later in the project, as it is commonly known to be the best algorithm for comparing a sequence and a statespace.[Sklar 2001].

2 Problem Definition

This section will provide the definition of the problem solved in this work, as well as an outline of how it's broken down into manageable pieces. In addition it will discuss some limitations and expected results.

The main problem discussed in this paper, if boiled down to just theory, is to find a sequence in a database that closely resembles a given sequence. In context of the background for this paper, it becomes as such:

Find the best match for a provided trail within a database, with the goal of optimal training for high performance athletes.

This definition is also discussed during the pre-project, although it was not completely solved. With the terminology defined during the pre-project, the problem can be restated as the following: "*Find the best Candidate for a Target within the database...*". The problem can also be stated as finding subsequence $\vec{\boldsymbol{J}}$ in Candidate C_i so that the distance $d_{(C_i,\vec{\boldsymbol{J}}),T}$ between a Target , T, and a subset $\vec{\boldsymbol{J}}$ of Candidate C_i so that the difference between the Target and the subset of the Candidate is minimised, from a set *db* containing all Candidates. That is, find $C_{i,\vec{\boldsymbol{J}}}$ from set *db* such that:

$$C_{i,\vec{\boldsymbol{J}}} = \arg\min_{i,\vec{\boldsymbol{j}}}\{d(T, C_{i,\vec{\boldsymbol{J}}})\}, C_i \in db = \{C_0, C_1, \ldots, C_i, \ldots, C_I\} \tag{1}$$

This task is equivalent to finding the subsequence $C_{i,J}$ in *db* that has the highest probability of being generated by a given model $\lambda_S(\boldsymbol{T}, \boldsymbol{E}, \boldsymbol{\pi})$, which can be expressed as $P(C_{i,J}|\lambda_S)$, where P denotes probability.

$$C_{i,\vec{\boldsymbol{J}}} = \arg\max_{i,\vec{\boldsymbol{j}}}\{P(C_{i,\vec{\boldsymbol{J}}}|\lambda_S)\} \tag{2}$$

In Eq. 2, model λ_S is trained on the Target so that it can "recognise" a subsequence $\vec{\boldsymbol{J}}$ in Candidate i.

2.1 Limitations

This section will outline the limitations of the project. One of these limitations is that optimal training is subjective. What can be considered a good match and training opportunity for one athlete will not be the same for another athlete. Since what might be experienced to be a good match is different from user to user, the metric $d\{\bullet\}$ will be set to maximum probability when considering HMMs, which will be shown later. In addition, although the problem states that the ping $C_{i,\vec{J}}$ that is to be found is the single closest one to the Target, it will be altered so that all pings that have similarity above a limit l will be presented. This limit l will not explicitly be stated nor defined, but rather be a relative level, since the experienced match might be different between users. It can be deduced later, but is not of particular interest, unless the pings produced during testing are of extremely low quality.

In addition to the subjective nature of a good match, there is also a limitation placed upon the data provided to the model, and the database. It is assumed that the data is either in lat/lon/alt-format, or can be easily converted into it. This potentially limits the use of the model created in the project, as it excludes some data, and increases the work needed before a potential Candidate can be included in the database, since new data need to be screened before adding.

The stated name of the project and the problem defined in this section does not perfectly overlap, as the definition doesn't cover all sports. Since the definition only includes trails, the application of any system created to solve this problem is limited to athletes competing on trails, i.e. distance-athletes. Examples of what can be considered distance-athletes are runners, bikers, and most skiiers. Further, distance-athletes such as sprinters won't gain any real advantage in using the system, as their trails are replicated to great accuracy across several locations, e.g. a 400m track and field-running track being constructed and well defined. In addition, the features used in the project, which creates the definition for what is considered a useful trail, is limited to the spatial geometry of the trails. This is because a more fitting set of features will require background theory from the field of sport science, which is far from the field this paper is written within.

3 Design of the system

This section will contain an outline of the system designed to solve the problem defined in Sec. 2. The system contains a database and a Machine Learning(ML) model that is a combination of two other sub-models. The two sub-models will be described in Sec. 4. The sub-models will be combined into one model, which will be named the "Merged model". From the Candidates, some features will be extracted, which will be used to train the mentioned sub-models. The database will contain a number of trails designated as Candidates. Candidates and Targets are the two groups of trails used, the difference between them being that a Candidate is a trail that OLT, or one of their athletes, have available for training, while the Target is the trail an athlete is going to compete on in e.g. the Olympics. All these terms will also be explained in the section, starting with trails, before delving into an explanation of what

different parts of the suggested system is and how they are constructed. Some background theory will also be given.

3.1 Defining trails

A trail, T_i, is a set of positions, p_j, and times, t_j. When representing the trails, it is desireable to find some features that accurately represent the trail, which will be discussed later. For some later iterations, what is considered a trail might be changed to include information in regards to the physical state of an athlete, in order to capture some other aspect of the trail rather than just the spatial geometry. Collected, the chosen information-samples can be called a state, shown and stored as a vector $\overrightarrow{\boldsymbol{s_{i,j}}}$.

$$T_i = \{\overrightarrow{\boldsymbol{s_{i,0}}}, \overrightarrow{\boldsymbol{s_{i,1}}}, \overrightarrow{\boldsymbol{s_{i,2}}} \dots \overrightarrow{\boldsymbol{s_{i,j}}} \dots \overrightarrow{\boldsymbol{s_{i,J}}}\} \tag{3}$$

where each $\overrightarrow{\boldsymbol{s_{i,j}}}$ currently has three spatial dimensions, and some time information:

$$\overrightarrow{\boldsymbol{s_{i,j}}} = [x_{i,j}, y_{i,j}, z_{i,j}, t_{i,j}]^T \tag{4}$$

In vector 4 the symbols x,y, and z refer to latitude, longitude, and altitude, while t is time. In total, a sequence T_1 of length $J+1$ becomes:

$$T_1 = \left\{ \begin{bmatrix} x_{1,0} \\ y_{1,0} \\ z_{1,0} \\ t_{1,0} \end{bmatrix}, \begin{bmatrix} x_{1,1} \\ y_{1,1} \\ z_{1,1} \\ t_{1,1} \end{bmatrix}, \begin{bmatrix} x_{1,2} \\ y_{1,2} \\ z_{1,2} \\ t_{1,2} \end{bmatrix}, \begin{bmatrix} x_{1,3} \\ y_{1,3} \\ z_{1,3} \\ t_{1,3} \end{bmatrix}, \begin{bmatrix} x_{1,4} \\ y_{1,4} \\ z_{1,4} \\ t_{1,4} \end{bmatrix} \dots \begin{bmatrix} x_{1,J} \\ y_{1,J} \\ z_{1,J} \\ t_{1,J} \end{bmatrix} \right\} \tag{5}$$

From this sequence, it is possible to extract a set of features which will create some observations defining the sequence. Note that if a trail is designated to be a Candidate, this can be indicated by using C_i instead of T_i, while if it is designated as a Target, it stays as T_i. During this project, only a single Target will be considered, and the subindex i will be dropped. In Fig. 1, the top subplot is the latitude and longitude of a Candidate, while the bottom subplot contains the altitude by the distance travelled within the Candidate.

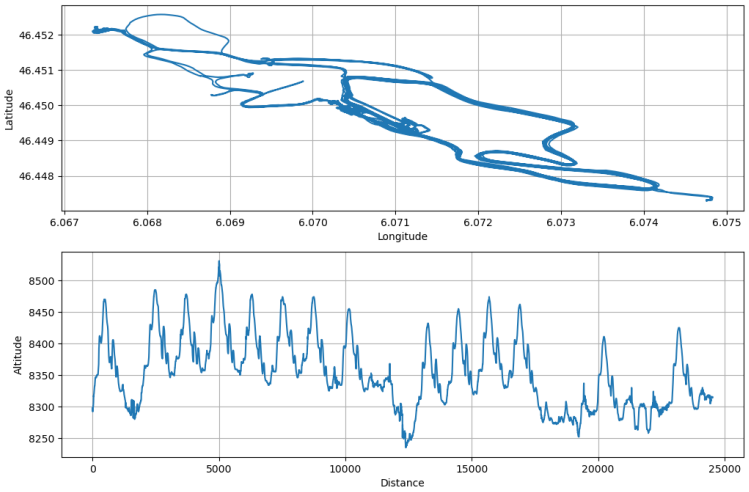


Figure 1: Birdseye view and height-profile of an example Candidate.

The Candidates will be collected in a database, the structure of which will be shown in Sec. 3.3. In addition to the spatial and temporal information, the trails are split into groups, named segments. The segments will be used later, as the basis for the states in a Hidden Markov Model(HMM). The process of defining the segments is shown in the next section.

3.2 Segmentation

As mentioned, the trails are split into some segments, which will be extremely usefull later. Shown in Fig. 2 is the groups that results from the process. The trail in the figure is what was used as the Target for the duration of the project. The segmentation is done by monitoring the individual directions(lat/lon/alt) of a trail and marking changes in their direction.

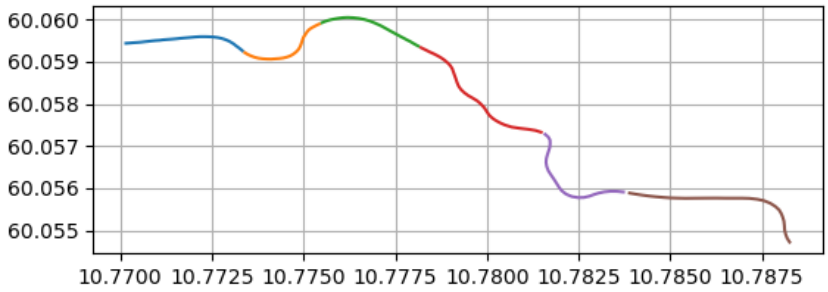


Figure 2: Segmented Target.

That is, if the athlete moves in a straight line, the direction of the different dimensions stay the same, either growing or shrinking. If there is a change in either dimension, there is a change in the geometry of the trail, which leads to a new segment starting. This is then marked, stored and used for training the models, which will be shown later, and possibly used for extracting some features in later iterations. In the pre-project, this method was used to extract features describing the curvature and climb of the segment. One can visualise the different hidden states in a HMM as one kind of segment. Pseudocode for the creation of the segmentation-markers are shown in Pseudocode(PC) 1.

Pseudocode 1 Segmentation

Require: List,limit, threshold
for i in length of List **do**
  if diff(List[i])>limit **then**
    Change count ++
  end if
  if Change count >=threshold **then**
    Markerlist[i] ← Marker Value
    Change count ← 0
  end if
end for
return Markerlist

This is done for all three dimensions simultaneously, and the segments are created by using the markers as the start and end points. In the PC, "Variable ++" means increasing the value of Variable by one, and is synonymous with "Variable ← Variable + 1". The threshold is used to ensure the segments are of usable length, so that the samples aren't split into single-sample segments. When reading about HMMs later in the section, one can visualise the different segments as the states in a Markov chain, or the hidden states of a HMM.

For the Target, the segmentation can simply be done as needed, as this should be a low number, but it is advantageous to store the segments of the Candidates. This is because the same Candidates will be considerd for several Targets, thus it becomes faster to store the information, rather than re-create it again for each new use. The Candidates and their information as described in Sec. 3.1 will be stored alongside the segments, which will be shown in the following section.

3.3 Creating a database

To be able to use the tool designed in this paper, the model will require data, which will be stored in a database. This database should only hold valid Candidates for athletes to train on, according to what was explained in Sec. 3.1, in addition to the segments. In addition to the segments, the distance between the samples of the Candidates is also stored, as this is used later. The requirements of the database mainly pertain to the contents of the Candidates: the Candidates have to be in the lat/lon/alt-format. In addition, the database also has to seperate the different Candidates, so that one Candidate can be extracted and evaluated without any inter-Candidate "contamination". This has been handled by simply adding a column naming the Candidates which is used as a handle for the extraction process. The database is then structured as shown in Tab. 1. Note that the name is added purely for practical reasons, and doesn't affect the theory of the design of the models whatsoever. The user then just have to cycle through the Candidates in the database, extracting them using the value in the "name" column, which in the current iteration is a unique integer. Using the data in the different columns, it is now possible to extract the features that will be discussed in Sec. 3.4.1 and evaluate the Candidates. Using the notation shown in Sec. 3.1, the table looks as shown in Tab. 2. As will be seen when looking at the features used, neither the Segments nor the velocity and time is needed in the database. Strictly, the distance is not required either, as it can be found through the coordinates, but for simplicity it is included. The lengths of the different segments will be used to differentiate the different states of the machine learning models that will be shown in Sec. 4.2. In this project, the data used for the training of the models is closely linked to the coordinates of the trails. Although a trail is considered to be the coordinates defining the path, this data is not usable for a the models that will be used later. In order to make them useful, some features have to be extracted from them, which will be explained in the following section. If the process of the extraction is very extensive, the features can be stored in the database as well, but with the chosen features for this iteration it is easier to extract them.

Table 1: Structure of a database with $N + 1$ Candidates.

Name	Lat	Long	Alt	Time	Segment	Distance	Velocity
0	$Lat_{0,0}$	$Long_{0,0}$	$Alt_{0,0}$	$t_{0,0}$	$Seg_{0,0}$	$Dist_{0,0}$	$v_{0,0}$
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
0	$Lat_{0,i}$	$Long_{0,i}$	$Alt_{0,i}$	$t_{0,i}$	$Seg_{0,i}$	$Dist_{0,i}$	$v_{0,i}$
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
0	$Lat_{0,I}$	$Long_{0,I}$	$Alt_{0,I}$	$t_{0,I}$	$Seg_{0,I}$	$Dist_{0,I}$	$v_{0,I}$
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
n	$Lat_{n,0}$	$Long_{n,0}$	$Alt_{n,0}$	$t_{n,0}$	$Seg_{n,0}$	$Dist_{n,0}$	$v_{n,0}$
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
n	$Lat_{n,i}$	$Long_{n,i}$	$Alt_{n,i}$	$t_{n,i}$	$Seg_{n,i}$	$Dist_{n,i}$	$v_{n,i}$
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
n	$Lat_{n,I}$	$Long_{n,I}$	$Alt_{n,I}$	$t_{n,I}$	$Seg_{n,I}$	$Dist_{n,I}$	$v_{n,I}$
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
N	$Lat_{N,0}$	$Long_{N,0}$	$Alt_{N,0}$	$t_{N,0}$	$Seg_{N,0}$	$Dist_{N,0}$	$v_{N,0}$
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
N	$Lat_{N,i}$	$Long_{N,i}$	$Alt_{N,i}$	$t_{N,i}$	$Seg_{N,i}$	$Dist_{N,i}$	$v_{N,i}$
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
N	$Lat_{N,I}$	$Long_{N,I}$	$Alt_{N,I}$	$t_{N,I}$	$Seg_{N,I}$	$Dist_{N,I}$	$v_{N,I}$

Table 2: Structure of database using notation.

Name	Trail	Segment	Distance	Velocity
0	C_0^T	$\overrightarrow{\boldsymbol{Seg_0}}$	$\overrightarrow{\boldsymbol{Dist_0}}$	$\overrightarrow{\boldsymbol{v_0}}$
1	C_1^T	$\overrightarrow{\boldsymbol{Seg_1}}$	$\overrightarrow{\boldsymbol{Dist_1}}$	$\overrightarrow{\boldsymbol{v_0}}$
⋮	⋮	⋮	⋮	⋮
n	C_n^T	$\overrightarrow{\boldsymbol{Seg_n}}$	$\overrightarrow{\boldsymbol{Dist_n}}$	$\overrightarrow{\boldsymbol{v_n}}$
⋮	⋮			
N	C_N^T	$\overrightarrow{\boldsymbol{Seg_N}}$	$\overrightarrow{\boldsymbol{Dist_N}}$	$\overrightarrow{\boldsymbol{v_N}}$

3.4 Features: How to represent a trail

This section will contain an explanation of how to represent a trail to enable analysis and evaluation. This is done by extracting some features from the trails. The features are extracted from the trail, and enable the creation of a model based on the features, rather than the raw data of the trail as defined in Sec. 3.1. This offers a plethora of benefits, mainly being that it allows preprocessing of the data in order to make the models more effective, and capture different aspects of the trail. If for example an athlete is interested only in the height-profile of the Target, only the altitude is used as the feature. What features are extracted has to be chosen carefully, and with the specific application in mind, preferably with

background theory and knowledge based upon the specific sport it is being implemented for.

During the fall-project, the feature extraction consisted of segmentation, before a measure of turning was extracted, as well as the length of each segment, and the difference in altitude. These three features defines the segment they are set for. This feature extraction lost some information due to the nature of the segmentation, and should thus be changed for this project. Since the features used in the pre-project is not used further, they will not be explained in great detail in this report. Here the chosen features that the models created in the report will be given will be presented, as well as a suggestion for other possible features that could be useful.

In general, a feature extraction could be considered a transformation, moving from a sample-space of the raw signal, or rather its recording, into a feature-space. In mathematical terms this would be the following:

$$T_i = \{\overrightarrow{\boldsymbol{s_{i,0}}}, \overrightarrow{\boldsymbol{s_{i,1}}}, \overrightarrow{\boldsymbol{s_{i,2}}} \dots \overrightarrow{\boldsymbol{s_{i,j}}} \dots \overrightarrow{\boldsymbol{s_{i,J}}}\} \rightarrow F_i = \{\overrightarrow{\boldsymbol{f_{i,0}}}, \overrightarrow{\boldsymbol{f_{i,1}}}, \overrightarrow{\boldsymbol{f_{i,2}}} \dots \overrightarrow{\boldsymbol{f_{i,L}}} \dots \overrightarrow{\boldsymbol{f_{i,L}}}\} \tag{6}$$

Where F_i is the new feature sequence. For each $f_{i,l}$ there is a general vector with $N+1$ features:

$$\overrightarrow{\boldsymbol{f_{i,l}}} = [f_{0,l}, f_{1,l}, \dots, f_{n,l}, \dots, f_{N,l}]^T \tag{7}$$

Where $f_{i,l}$ is feature i, which can be from a list to be altered later in the project. It is also possible to name some features, and change their symbol as such:

$$\overrightarrow{\boldsymbol{f_{i,l}}} = [a_{0,l}, c_{1,l}, d_{2,l}]^T \tag{8}$$

In the feature-vector shown in Eq. 8, a is the feature representing change in altitude, c is the feature for turns, and d is the feature for distance. The feature vector is what was used in the pre-project.

3.4.1 Features used

The earliest proposed features are as follows:

1. Change in altitude in the segment($a = \Delta z$)
2. The distance from the samples to a straight line between the start and end point ($c = \sum p - p_0$) where p_0 is a virtual straight line between the start and end samples in the segment. Functionally, this is the same as rotating the segment, and then sampling the data so that the result is the "magnitudes" of the segment.
3. The distance travelled within the segment, $d = \sum_{n=1}^{N}[d(p_n, p_{n-1})]$ where sample N is the last sample of the segment, and using the euclidean distance between samples.

All these features represent the spatial layout of the trail, and the comparison between different trails is done upon this layout. When considering the achieved goal, there is a better method for the representation of the trails, or the features extracted. Since the method outlined above is based mainly upon the geography of the trails, the optimal method would

be one that uses features close to the geography in terms of information-content. This would mean that the features used to compare and sort the different trails are processed as little as possible.
To achieve this, the features outlined above are exchanged with the following:

1. $\frac{\Delta x}{\Delta d}$
2. $\frac{\Delta y}{\Delta d}$
3. $\frac{\Delta z}{\Delta d}$

In the features explained above, x, y, and z refer to the latitude, longitude, and altitude of the trail, while Δd is the change in distance between two samples, which is feature d from the pre-project. By doing this, the features are only two operations away from the original information contained in the input. This is also the same as taking the differential in the different dimensions, by the distance. Although both sets of features are human-readable, the second suggestion is arguably better, since it captures the same changes in the trail, but being closer to the original information. The second feature-set also function on a sample to sample basis, while the first functions on a segment-to-segment basis. This better fits the models designed later, as they can now stretch and adapt to different sequences based on the samples alone, and doesn't require the segmentation process to happen beforehand, which might drastically change the result based on the segmentation. This will be further delved into in Sec. 8.

The differential is taken so that the features capture the changes in the paths defining the different trail, rather than simply the position. If the raw coordinates where to be used, the model would end up comparing the physical closeness to the Target, rather than the shape.

3.4.2 Other possible features

In this section there will be a short discussion regarding what features could be implemented in the feature, as well as the reason for why they haven't been implemented now. The features used and explained above are as mentioned based upon the spatial placement of the trails. This enables the model that will be created later to compare trails based upon their path, which again makes it possible to detect trails that are similar to eachother by their spatial shape. As has been shortly mentioned earlier in Sec. 3.1, other information might be included in later iterations of the system in order to enable other features. During the pre-project, there where some feedback from OLT which referenced the features used. In this discussion there where some suggestions as to what the features could try to capture, mainly being the resistance or experienced work when moving through a trail. This could be based upon information such as heart rate, power output, or oxygen consumption. Another option could be to measure movement by using accelerometers to capture the relative movement, instead of a feature based upon coordinates. By doing this, the model could capture other aspects of a trail, i.e. the experienced resistance of a trail. Another option could be to include the velocity or acceleration captured when recording the trail. When considering this problem, it is important to keep in mind how the choice of features affect the available body of Candidates. If a set of features that requires e.g. heart-rate is chosen for the basis of the

system, all Candidates that lack this information is automatically excluded. The Target also requires the same information, which might limit the available features.

The implementation of another feature-set that aims to capture a more abstract aspect of the trails will require background theory and knowledge from the field of sports science. This knowledge would be what information is required to capture the different aspects, as well as the knowledge of what is common to measure. Since this is outside the field and scope of this work, the chosen features only aimed to capture the spatial geometry of the different trails. It was also considered to include features based upon the frequency-contents of the different trails, hence the inclusion of time in the definition of a trail. Ultimately the features from Sec. 3.4.1 was used due to the reasons stated above; they capture the spatial geometry of a trail, which enables an athlete to prepare for a competition trail.

Although both feature-sets from Sec. 3.4.1 include three features, more can be included. It is possible to simply include features based upon both the spatial geometry and the physical state of an athlete simultaneously, as long as the inlcusion of more information doesn't exclude all Candidates from the database. Changing the features used requires the models to be retrained to the new feature-set.

3.5 Extracting features

This section will contain the implementation of the extraction of the features described in Sec. 3.4.1. The features extracted are to be fed to a combination of two Machine Learning models. The models are called Hidden Markov Models(HMMs). When discussing the features in relation to the models, they might be called observations. The first model, the Seeker, is trained on observations extracted from the Target. The second model, the Garbage, is trained on observations from another trail, or several. The observations are created as shown in PC 2. These observations are then provided to a HMM to train upon so that it can recognise

Pseudocode 2 Feature extraction

Require: Coordinates,Distance

```
diff lat, diff long, diff alt ← Differential(Coordinates)
diff distance ← Differential(Distance)
features[:,0] ← diff lat / diff distance
features[:,1] ← diff long / diff distance
features[:,2] ← diff alt / diff distance
return features
```

similar features in a Candidate later on. In PC 2, coordinates and the distance is assumed to be in the correct format and with the correct units. There exists the possibility of dividing by zero in the PC, but this is handled in the actual code. In order to use these features, they will be provided to a HMM which will fit its transition and emission parameters to the features.

3.6 Terms used in Machine Learning

This section will contain some terms used in the field of Machine Learning(ML), that will be used in later sections.

3.6.1 Models

In ML, a model is a programme that analyses data to find patterns and make predictions based upon them. There are a large variety of models, most of which are split into one of two categories: supervised and unsupervised models. This refers to if they require labelled data or work on unlabelled data.

3.6.2 Data augmentation

Data augmentation is used as a broad term for techniques that grows a training dataset by slightly augmenting data already present in the dataset, by e.g. rotating, adding noise, or mirroring, depending on the specific application. For this application, adding noise is the most useful.

3.6.3 Dropout

Dropout is when layers within a model is dropped from training in a certain training cycle, which might help the model learn patterns in the data in a more robust fashion.

3.6.4 Overfitting

In the field of ML, overfitting is when a model too closely learns the training data, and is thus able to perfectly model the training data. This may sound like a positive thing, but when an overfitted model is applied to new data, either through application or on test-data, the performance is far from the one achieved when training, reducing the generality of the model. Overfitting happens when the model to be trained contains more parameters than is trainable by the amount of data. Overfitting can be combated by either having more training-data, which is not always possible, or by techniques such as dropout, or artificially increasing the data through data-augmentation. It is also possible to simply decrease the number of parameters a model needs to train.

3.6.5 Underfitting

Underfitting is when a model is not complex enough to capture the structure of the data it is applied to. This results in a model that is unable to properly model data, as it has been trained to a too simple system. Underfitting can be combated by increasing the complexity of a model, or through some feature engineering. Like with the remedies for overfitting, the correct remedy will vary for different applications/problems.

3.6.6 Convergence

When training a model, changing parameters, or during testing, results have to stabilise in order to be reliable and trustable. To do this, one can monitor the results of some variable, and compare it to the mean of the variable across all iterations. Once the result of one iteration is within some tolerance-limit of the mean of all iterations, the procedure is said to have converged. If the variable is e.g. the log-probability of a model, convergence is reached when the log-probability of one test is approximately the same as it has been. Updating parameters or running more tests are not likely to produce better results, and testing is often ended at this point. When considering training ML models, the training should be stopped when convergence is reached, in order to minimise the risk of overtraining. Note that reaching convergence does not guarantee not overtraining, but it is more likely to not have happend before convergence is reached.

3.6.7 Neural networks

Neural networks(NN) are ML models that consist of layers of nodes connected by weights. By changing the way the nodes process inputs as well as the weights between nodes, one can train a Neural Network to perform some tasks, like classification, or text generation. There are numerous different structures of Neural Networks, like a Convolutional Neural Network(CNN), or a Recurrent Neural Network(RNN). A CNN is applied to structured data such as a image or another type of fixed-size array. RNNs are often applied to time-series data, such as weather-data, temperatures, or others. Both types of networks are powerful tools, but require massive amounts of labeled data.

3.7 Modelling a trail to find good Candidates

This section will contain an outline of the model that is used to detect when a section of a Candidate is similar to the Target, which it does by modelling the different Candidates. This model is named the Merged model. The Merged model will consist of two different models that will be described later. One of these models will be responsible for detecting when a sequence of a Candidate is similar to the Target, while the other will be responsible for representing any other trail that exists.

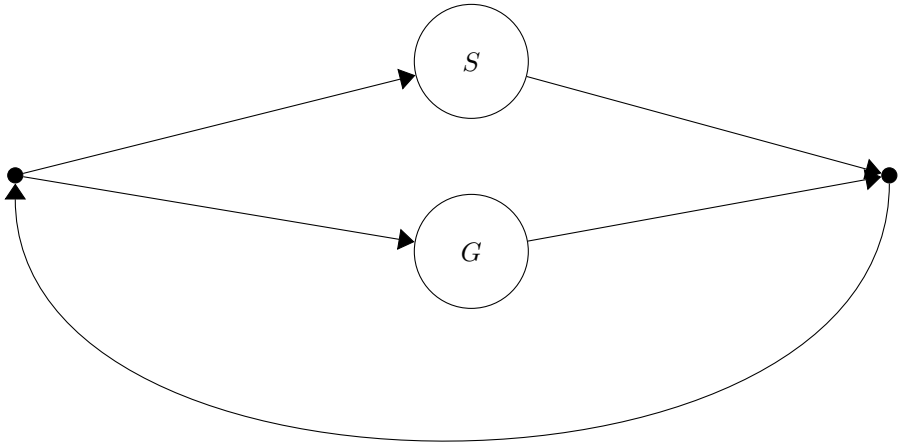


Figure 3: A general view of the Merged model.

Shown in Fig. 3 is a general view of the Merged model, where the two different models are differentiated by the letter of their node, either "S" or "G". As seen the Merged model can use either one of the two models, both of which will be explained in later sections. The "S" model is responsible for the detection of the Target-similar sections of Candidates, while the "G" model is the one representing every other section of the Candidates. The Merged model will use these two models to represent any trail it is given, and by monitoring which model is being used for a given section, it is possible to determine that it is either similar or dissimilar to the Target. As has been discussed, the trails are defined as sequences of states, meaning that they can be modelled by state space models. The goal of the Merged model is that any trail should be able to be represented within the Merged model's state space. When working with state spaces and sequences of states, the model needs to determine what sequence of states are the most appropriate for a given sequence of observations. To solve this problem, the Viterbi algorithm will be used, which is known to be the best algorithm for such tasks[Sklar 2001]. It will be briefly discussed in Sec. 3.8.

3.8 Viterbi algorithm: Finding the best state sequence

The Viterbi Algorithm is a dynamic programming algorithm that provides the most likely sequence of states, given a state space, $Z = \{\boldsymbol{X_0}, \ldots, \boldsymbol{X_m}, \ldots \boldsymbol{X_M}\}$, and some observations, $\vec{\boldsymbol{O}} = [o_0, \ldots, o_t, \ldots, o_T]$. It is often used in communications and other applications using HMMs. The resulting sequence that the algorithm determines to be the most likely is often named the Viterbi path. It was proposed by Andrew Viterbi in 1967 to be used in decoding convolutional codes[Forney Jr 2005]. Mathematically, the Viterbi path, Q, can be expressed as such:

$$Q = \{q_0, q_1, \ldots, q_t, \ldots, q_T\} \text{ such that } q_t = \arg\max_i [V_t(i) \times P(\boldsymbol{X_j}|\boldsymbol{X_i}) \times P(o_{t+1}|\boldsymbol{X_j})] \quad (9)$$

Where $V_t(i)$ is the probability of being in a state $\boldsymbol{X_i}$ at time t, $P(\boldsymbol{X_j}|\boldsymbol{X_i})$ is the transition probability from state i to state j, while $P(o_{t+1}|\boldsymbol{X_j})$ is the emission probability. In the equation, o_{t+1} is the observation at time $t+1$ from the observation vector $\vec{\boldsymbol{O}}$.

When used in the context of HMMs and statespaces, the Viterbi path Q represent the sequence states that are the most likely to have produced the observations $\vec{\boldsymbol{O}}$. By looking at these states, and which of the sub-models the states belong to, it is possivle to use the Merged model described in Sec. 3.7 to see when a Candidate is similar to the Target. This is done by looking at what sub-model the different states q_t in the Viterbi path Q stems from, being either the one similar to the Target, or the one that represents everything else.

3.9 Sub-model 1: Creating a Seeker for the Target

This section will contain a outline of the purpose of one of the previously mentioned sub-models. As mentioned, one of the sub-models are responsible for detecting when a sequence of a Candidate is similar to the Target, which will be named the "Seeker". The purpose of the Seeker can be reworded as detecting a pre-defined sequence in another, longer sequence. The Seeker then needs to be able to model the Target-like sequence better than the other sub-model. This will be acheived by training the Seeker on the Target directly. In addition to being trained on the Target, the Seeker also needs to be structured in a certain way, as it is required to detect the different sections of the Target in the correct order. This makes it so that the Seeker detects an *exact* sequence and not just any sequence containing similar states, like the other model is supposed to do. Within the context of the project, detecting a specific sequence can be explained as detecting a specified turn, followed by a straight section of a given length, and then a certain hill etc. The other sub model would in contrast simply model any turn, followed by any straight section, followed by a hill etc. This will be discussed in Sec. 3.10.

3.10 Sub-model 2: Creating a Garbage model for everything else

This section will contain a outline of the purpose of the other previously mentioned sub-models. This sub-model will be named the "Garbage" model. As was mentioned, the Garbage model is supposed to be able to represent *any* trail. The Garbage model has no requirement as to how well it should be able to represent the trails, only that the Seeker should be better at representing the Target, and sections of Candidates similar to it. For this reason it is named the Garbage model, as it should be able to represent all the "garbage" sections of a Candidate.

While the Seeker is supposed to represent a pre-defined sequence, the Garbage model should simply be able to model any trail. It thus doesn't need to have a set structure, and actually benefits from less structure as this enables it to model a greater number of trails to a higher degree. Both the Seeker and the Candidate will be designed and implemented in Sec. 4. Theory for the models will also be given in the section.

4 Creation of the two sub-models

This section will contain the implementation of the two sub-models, the Seeker and the Garbage model. It will also contain theory regarding the design and training of the models. Although the complete solution only has one model, two are designed, the reasons of which was given in Sec. 3.7. The method used to combine the two models will be given in Sec. 5. The models used in the project where Hidden markov models, which are based upon Markov chains, both of which will be described in this section.

4.1 Markov Chains for the process

A Markov process or a Markov chain is a stochastic model describing a sequence of events in which the next state is only dependant upon the current state, known as the Markov property. Markov chains can be both continuous and discrete in both time and state space. For the purpose of this paper and implementation, only discrete-time chains will be used. A Markov chain has a state space, $Z = \{\boldsymbol{X_0} \dots \boldsymbol{X_m}, \dots \boldsymbol{X_M}\}$, defined by a transition matrix, $\boldsymbol{T}$, and initial probabilities, $\vec{\boldsymbol{\pi}}$. The previously mentioned Markov property can be found described in Eq. 10.

$$P(\boldsymbol{X_{n+1}} = x | \boldsymbol{X_n} = x_n) = P(\boldsymbol{X_{n+1}} = x | \boldsymbol{X_0} = x_0, \boldsymbol{X_1} = x_1, \dots \boldsymbol{X_n} = x_n) \tag{10}$$

In the equation, $P(\boldsymbol{X_{n+1}} = x | \boldsymbol{X_n} = x_n)$ is the probability of transitioning to state $\boldsymbol{X_{n+1}} = x$ at the next time step, indicated by the subindex $n+1$, given the current state being $\boldsymbol{X_n} = x_n$. Likewise, $\boldsymbol{X_n} = x_n$ represents the current state of the process. Collected, the equation states that the probabiliy of observing the next state given the current state is the same as the probability of observing the next state given all the previous states in the sequence. More simply, this is the Markov property explained above. The value that defines the transitions between state $\boldsymbol{X_n}$ and $\boldsymbol{X_{n+1}}$ can be collected in a matrix known as a transition matrix. A general transition matrix can be found below, representing a state space with M states.

$$\boldsymbol{T} = \begin{bmatrix} p_{0,0} & p_{0,1} & \cdots & p_{0,j} & \cdots & p_{0,J-1} \\ p_{1,0} & p_{1,1} & \cdots & p_{1,j} & \cdots & p_{1,J-1} \\ \cdots & & & & \ddots & \\ p_{i,0} & p_{i,1} & \cdots & p_{i,j} & \cdots & p_{i,J-1} \\ \cdots & & & & \ddots & \\ p_{I-1,0} & p_{I-1,1} & \cdots & p_{I-1,j} & \cdots & p_{I-1,J-1} \end{bmatrix}$$

A transition matrix is always square, having the shape $M \times M$, meaning $I = M$ and $J = M$ in the matrix above. The elements in any given position (i, j) contains the probability of going from state i to state j at a time step n, shown in Eq. 11.

$$p_{i,j} = P(\boldsymbol{X_{n+1}} = j | \boldsymbol{X_n} = i) \tag{11}$$

Again, the subscript n refers to a states position in the sequence. Further, a row contains all the probabilities of transitioning from a state to another state, which is defined by the position. There are some features of a transition matrix that have to remain true for the matrix

to be valid, mainly being the shape and the values of the elements. Since the rows represent the probability of transitioning from the state, the sum of the row must be 1, and the individual elements must be positive. A value of 0 is also acceptable, representing no transition being possible. The initial probabilities is simply a vector, $\vec{\boldsymbol{\pi}} = [p_0, p_1, \ldots, p_m, \ldots, p_{M-1}]$ containing the probability of the initial state of the sequence being state m. This vector, like the rows in the transition matrix, has to sum to one.

A markov chain can be used to model e.g. the weather on a given day. This markov chain consisting of $M = 3$ states, ***Rainy***, ***Sunny***, and ***Cloudy*** can be found in Table 3.

	Rainy	Sunny	Cloudy
Rainy	0.6	0.2	0.2
Sunny	0.5	0.2	0.3
Cloudy	0.3	0.2	0.5

Table 3: Transition probabilities, $\boldsymbol{T}$.

The same Markov chain is shown in Fig. 4, where the states are indicated by the circles.

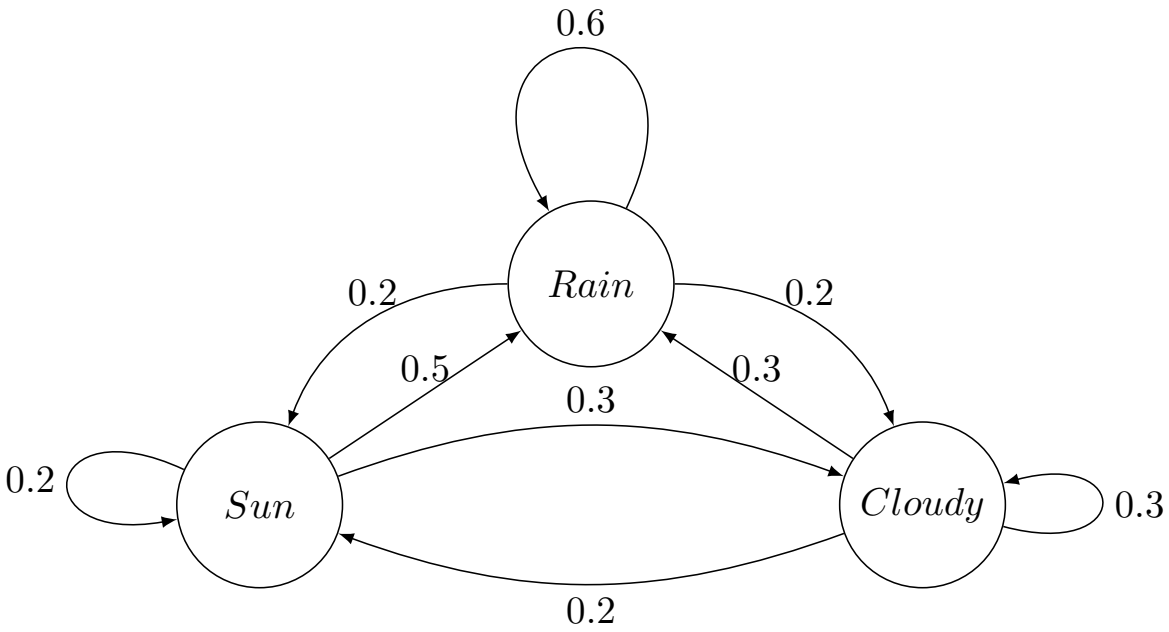


Figure 4: An example Markov chain.

Also visible in the figure is some arrows between states, indicating the probability of a transition from one state to another. The values associated with the different transitions can be found in the appropriate places in Tab. 3. Additionally, the initial probabilities have to be known, in order to properly initialise the process.

For a short Markov chain both representations are arguably equally beneficial, however when the Markov chain grows in complexity, or is used as the basis for a Hidden Markov Model, the visual element becomes more important, which will be shown later. To sum up, a Markov chain is defined by two sets of parameters: the transition parameters, $\boldsymbol{T}$, and the initial probabilities, $\vec{\boldsymbol{\pi}} = [p_0, p_1, \ldots, p_m, \ldots, p_{M-1}]$. These parameters define the state space Z of the Markov chain, which contains all the states of the process. The transition parameters is always a square matrix of size $M \times M$ where M is the number of states in the chain, while $\vec{\boldsymbol{\pi}}$ is a one dimensional list or vector of length M, simply containing the probabilities of starting in a state.

Markov chains will be used as a basis for the models designed later in the section. The markov chain will model the underlying process that is needed to be replicated to find a good match for the Target. For this purpose, the different states Z can be visualised as the movement that resulted in the coordinates obtained when recording a trail, like the one shown in Fig. 2, indicated by the different colours. In the following subsections, another set of states will also be introduced which will correspond to the features discussed in Sec. 3.4.

4.2 Hidden Markov Models for the observations

A hidden Markov model(HMM) is a model of a system that is assumed to be a Markov chain. As mentioned earlier, the Markov chain represents the process that is included in the system. The "hidden" part of the model refers to the fact that there are two sets of states: one hidden set, $Z = \{\boldsymbol{X_0}, \ldots, \boldsymbol{X_m}, \ldots \boldsymbol{X_M}\}$, and one observed set, $S = \{\boldsymbol{Y_0}, \ldots \boldsymbol{Y_p}, \ldots, \boldsymbol{Y_P}\}$. The observed set of states are an additional layer below the hidden states, connected by some emission-probabilities. If the HMM has discrete observation space, the set S is discrete, containing only the legal states, while if it is continuous, the set is defined by a distribution, which changes based upon the hidden state of the model. If the distribution of the HMM is continuous, e.g. a gaussian distribution, the observed state space will be defined as shown in Eq. 12.

$$\boldsymbol{Y_p} \sim \mathcal{N}(\mu(\boldsymbol{X_p}), \sigma^2(\boldsymbol{X_p})) \tag{12}$$

In the equation, $\mu(\boldsymbol{X_p})$ and $\sigma^2(\boldsymbol{X_p})$ refers to the changing mean and variance based upon the hidden state of the process, while $\mathcal{N}(\mu(\boldsymbol{X_p}), \sigma^2(\boldsymbol{X_p}))$ is the Gaussian distribution. If the observation space is discrete, the emission parameters resembles a transition matrix, only varying by the shape as shown below.

$$\boldsymbol{E} = \begin{bmatrix} p_{0,0} & p_{0,1} & \cdots & p_{0,j} & \cdots & p_{0,J-1} \\ p_{1,0} & p_{1,1} & \cdots & p_{1,j} & \cdots & p_{1,J-1} \\ \cdots & & & & \ddots & \\ p_{i,0} & p_{i,1} & \cdots & p_{i,j} & \cdots & p_{i,J-1} \\ \cdots & & & & \ddots & \\ p_{I-1,0} & p_{I-1,1} & \cdots & p_{I-1,j} & \cdots & p_{I_1,J-1} \end{bmatrix}$$

The emission matrix has a shape of $M \times P$, where M is the number of hidden states, and P is the number of observations. In the matrix, $M = I$ and $P = J$. This of course means that the matrix is not guaranteed to be square like the transition matrix. Instead of representing

the probability of transitioning, the elements in the emission matrix represent the probability p of observing an observed state $S = \boldsymbol{Y_j}$ given being in a hidden state $Z = \boldsymbol{X_i}$, as shown in Eq. 13.

$$p_{i,j} = P(S = \boldsymbol{Y_j} | Z = \boldsymbol{X_i}) \tag{13}$$

Like with the transition matrix, the rows of the emission matrix must sum to 1, and all values must be positive or 0. As might be guessed, the hidden states are not available for observation, and have to be inferred through the observed states. The HMM model has the aforementioned hidden states Z and observable states S, the transition matrix $\boldsymbol{T}$ and a initial probability vector, $\vec{\boldsymbol{\pi}}$. In addition it also has the emission parameters as described above, $\boldsymbol{E}$ relating to the observable states.

A system that can be modelled by a Markov chain can easily be explained through an example, which will be done below. A HMM will also be constructed from the Markov chain. Consider the following system: in the begining of a sequence, there is a 55% probability of a person being sad, and a 45% probability of the person being happy. This forms the initial probabilities, $\vec{\boldsymbol{\pi}} = [0.55, 0.45]$. If the current day is a happy, there is a probability of 70% for the following day to be happy as well. If the current day is sad, there is a 20% probability that the following day will be sad as well. Collected, these form the transition matrix, $\boldsymbol{T}$, which can be seen in Tab. 4. The process described is currently a Markov chain, shown either in Fig. 5, or Tab. 4. Notice that the table only captures the transitions, not the initial probabilities.

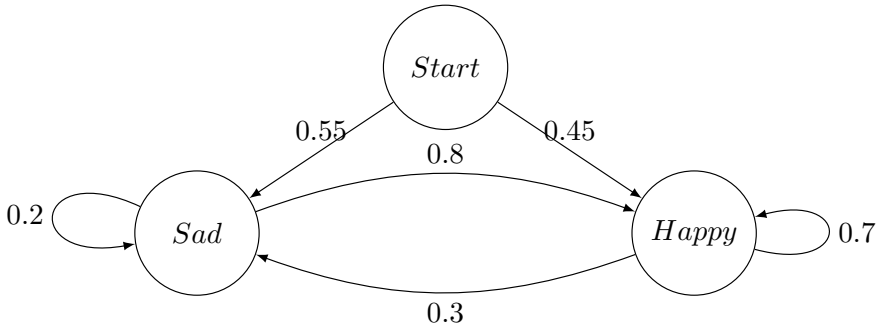


Figure 5: Markov chain for mood on a given day.

	Happy	Sad
Happy	0.7	0.3
Sad	0.8	0.2

Table 4: Transition probabilities, $\boldsymbol{T}$, for mood on a given day.

In order to "create" a Hidden Markov Model from the Markov chain, a set of observable states need to be introduced to the system, as the mood of the person is inobservable. In the given example, it could be noting some daily activities the person might

do. This could be that on a happy day, there is a 40% probability of going for a walk, and 60% probability for chores being done. If the person is sad on a given day, there is a 40% probability of reading inside, 40% probability of going for a walk, and a 20% probability of doing nothing. These events are called the observations and can be modelled as shown in Fig. 6, or in Tab. 5, and form the relationship between the hidden states being the mood, $Z = \{\boldsymbol{Sad}, \boldsymbol{Happy}\}$, and the observable states being the activities, $S = \{\boldsymbol{Doing\ nothing}, \boldsymbol{Reading}, \boldsymbol{Walking}, \boldsymbol{Doing\ chores}\}$.

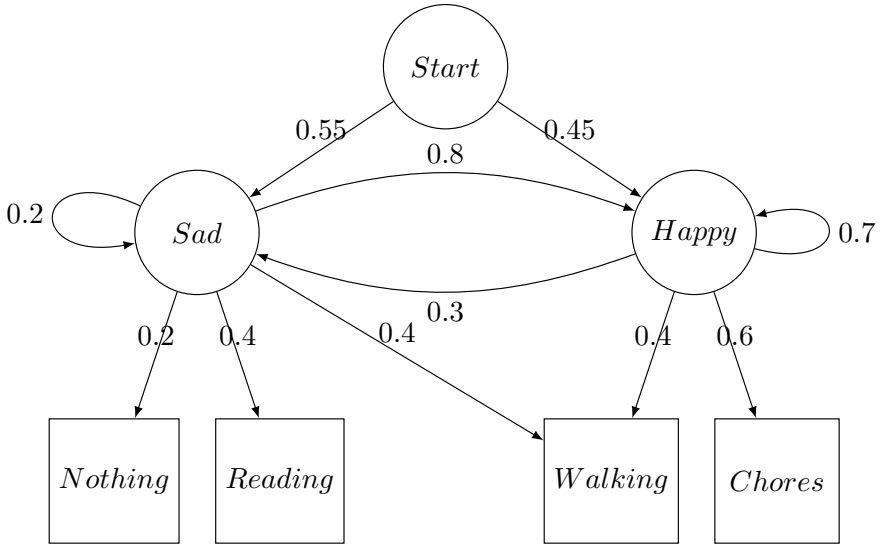


Figure 6: Hidden Markov Model for the mood-example.

	Doing nothing	Reading	Walking	Doing chores
Sad	0.2	0.4	0.4	0.0
Happy	0.0	0.0	0.4	0.6

Table 5: Emission probabilities, $\boldsymbol{E}$, for the mood-example.

As mentioned earlier, the tables and the figures have the same information, but the figure is arguably easier to read and understand. In addition, the figures are more intuitive to understand due to the visual element.

The HMM can now, by looking at the observed states and through the inclusion of some other algorithms, such as the Viterbi algorithm, be used to predict the most likely sequence of hidden states, $\vec{\boldsymbol{Z}}$ given the observed states, $\vec{\boldsymbol{O}}$. Mathematically this would be as shown in Eq. 14. In the equation the model parameters $\boldsymbol{\theta}$ are $\vec{\boldsymbol{\pi}}$, $\boldsymbol{T}$, and $\boldsymbol{E}$.

$$\hat{\vec{\boldsymbol{Z}}} = \arg\max_{\vec{\boldsymbol{Z}}}[P(\vec{\boldsymbol{Z}}|\vec{\boldsymbol{O}}, \boldsymbol{\theta})] \tag{14}$$

This is the Viterbi path through the statespace of the model for a given set of observations.

In order to use the HMM, it has to be connected to some observation, $\vec{\boldsymbol{O}}$. This process is called either training or fitting. During this process, the parameters changed are the transition matrix, $\boldsymbol{T}$ and the initial probabilities $\vec{\boldsymbol{\pi}}$, as well as the emission matrix, $\boldsymbol{E}$. If the HMM has a continuous observation space, the emission parameters is rather the parameters defining the chosen distribution. In the case of a Gaussian distribution, the parameters is the mean μ and variance σ^2. This is often done through EM-algorithms, which are described in Sec. 4.5. Important to note is that the training-process does not alter the number of states in the model, which is something that has to be known and set by the user. This decision can be made with prior knowledge of the dataset, or through some analysis. It can also be found through a trial and error method, but this process is not included in the one commonly known as "training".

For this application, the observed variables are the features extracted from the coordinates in the Target/Candidates, while the hidden variables represent the movement that led to the trail, or a specified section of it. The process of segmentation is explained in Sec. 3.2. Another example of a HMM that is fully connected is shown in Fig. 7. Note that the initial probabilities and the values of the transitions are not shown. In addition, there is only one observed state per hidden state, which might not always be the case. The emission probability could also be modelled by e.g. a Gaussian Mixture Model(GMM), which will be explained in Sec. 4.3.

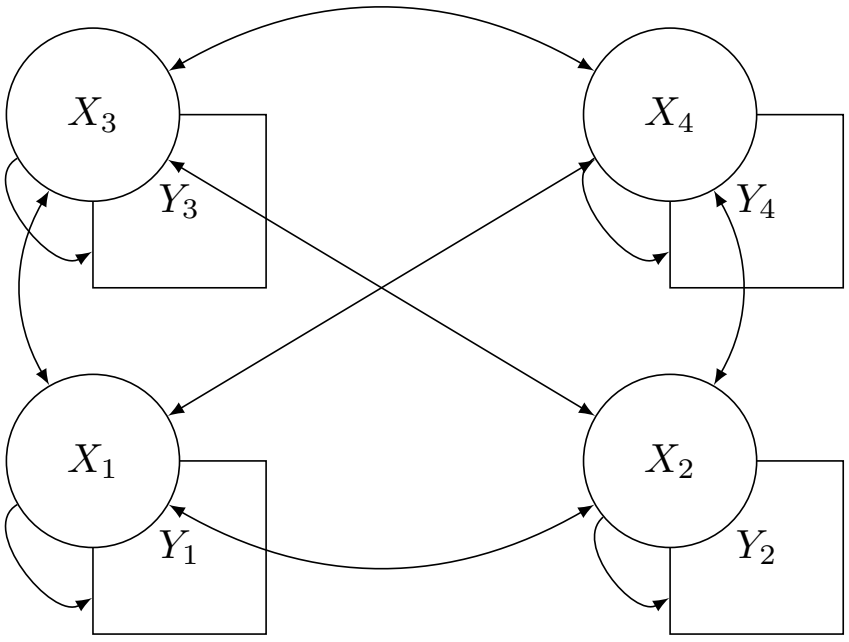


Figure 7: General, fully connected HMM with four states.

The implementation of a model as shown in Fig. 7, only with more states, will be called the "Garbage" model, which will be shown later. When implementing the model, the library responsible for the implementation itself needs to know where the different hidden states

are, and what observable samples are related to them. This is done by providing it a vector containing the lengths of the different samples relating to the different states that tells the library when to shift to a new state when training.

When modelling a trail as a HMM, the underlying process could be considered to be an athletes movement, as was described for a Markov chain in the previous section. The process then generates an observable set of features, that can be recorded or extracted, and use to infer the hidden state of the trail, which in this case is how the athlete moved to create the coordinates that are used as the basis for the trail stored in the database, as explained in Sec. 3.3. This differs from simply comparing the sequence of features directly as it allows for small differences in the different Candidates, but still allowing for a comparison. In addition, the use of HMMs allows for variable size inputs, as well as efficiently detecting variable length windows of potential good sections of a Candidate. If HMMs are not used, the process of testing every possible length of window across every Candidate will be extremely computationally demanding. Since the hidden states represent how an athlete moved in order to generate the set of features used for the observations, by using HMMs to detect when a Candidate is similar to the Target the system detects when it is possible to replicate the spatial geometry of the Target. This facilitates better training for athletes, acheiving the goal of facilitating better training as described in Sec. 1, by solving the problem from Sec. 2. In order to use a HMM on a trail, the observable state space needs to be continuous, which can be acheived by introducing another model, known as a Gaussian Mixture Model.

4.3 Gaussian Mixture Models for the emissions

A Gaussian mixture model(GMM) is a statistical model used to approximate other, arbitrary distributions. In the GMM the probability distribution of a dataset is represented as the weighted sum of K Gaussian distributions, where each distribution is known as a component. Each component is weighted relative to its importance in the dataset. With enough components, any distribution can be approximated to arbitrary precision.

For a dataset, $O = \{\overrightarrow{\boldsymbol{O_1}}, \ldots, \overrightarrow{\boldsymbol{O_k}}, \ldots, \overrightarrow{\boldsymbol{O_K}}\}$, where $\overrightarrow{\boldsymbol{O_k}} = [\overrightarrow{\boldsymbol{x_{k,1}}}, \ldots, \overrightarrow{\boldsymbol{x_{k,j}}}, \ldots, \overrightarrow{\boldsymbol{x_{k,J}}}]$ and $\overrightarrow{\boldsymbol{x_{k,j}}} = [x_{k,j,0}, \ldots, x_{k,j,d}, \ldots, x_{k,j,D}]$ being a D-dimensional observation vector, where x_d is the d-th component of the vector, the probability density function of a GMM with K components will be given by Eq. 15. The dataset $\boldsymbol{O}$ has K clusters, where each cluster $\overrightarrow{\boldsymbol{O_k}}$ has J elements.

$$p(\overrightarrow{\boldsymbol{x_{i,j}}}|\theta) = \sum_{k=1}^{K} w_k \mathcal{N}(\overrightarrow{\boldsymbol{x_{i,j}}}|\overrightarrow{\boldsymbol{\mu_k}}, \boldsymbol{\Sigma_k}) \tag{15}$$

In the equation, $\overrightarrow{\boldsymbol{x_{i,j}}}$ is a given data-point in the dataset. θ are the parameters defining the distribution. These parameters consist of the weights, w_k, means $\overrightarrow{\boldsymbol{\mu_k}} = [\mu_{k,1}, \ldots, \mu_{k,d}, \ldots, \mu_{k,D}]$, and covariance matrix $\boldsymbol{\Sigma_k}$ being a $D \times D$ matrix containing the covariances for a observation-cluster $\overrightarrow{\boldsymbol{O_k}}$. $\mathcal{N}(\overrightarrow{\boldsymbol{x_{i,j}}}|\overrightarrow{\boldsymbol{\mu_k}}, \boldsymbol{\Sigma_k})$ is the probability density function of distribution k. By summing the weighted components the GMM captures the likelihood of observing a data point $\overrightarrow{\boldsymbol{x_{i,j}}}$ given the parameters θ. For a $D = 2$ dimensional GMM with $K = 2$ clusters, the means and

covariances for component k would look as shown in Equations 16 and 17.

$$\overrightarrow{\boldsymbol{\mu_k}} = [\mu_{k,1}, \mu_{k,2}] \tag{16}$$

Where $\mu_{k,i}$ is the mean in dimension i.

$$\boldsymbol{\Sigma_k} = \begin{bmatrix} \sigma_{k,1,1} & \sigma_{k,1,2} \\ \sigma_{k,2,1} & \sigma_{k,2,2} \end{bmatrix} \tag{17}$$

In Eq. 17, $\sigma_{k,1,1}$ and $\sigma_{k,2,2}$ is the is the inter-dimensional variance, while the off-diagonal values are represent the covariance between the first and second dimensions. The weights are simply a number between 0 and 1 that represent the importance of the cluster in the dataset. The weights sum to 1.

In order to create the GMM, the first step is to determine the number of components in the model. Once the number of components is chosen, the next step is to initialise the model parameters by assigning random values to the parameters. After this, the model is trained using the Expectation-Maximisation (EM) algorithm, which works by repeatedly calculating the probabilities of the data points belonging to a component, and then updating the parameters θ, which includes the weigths w, the means μ, and the covariances Σ for each component. The EM-algorithm is further discussed in Sec. 4.5. Once the GMM is trained, it is ready to be used in applications, e.g. used in a HMM, as will be shown later.

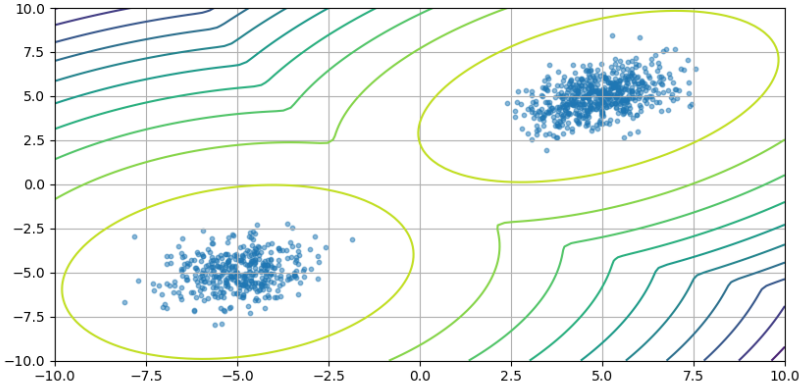


Figure 8: A 2-dimensional GMM with two clusters and two components.

In Fig. 8, an example GMM with $D = 2$ dimensions and $K = 2$ mixtures (and clusters) is shown. The single GMM can model both clusters in two dimensions, making it very useful for certain applications.

In this application, the GMM will be used by the Merged model to model the relation between a hidden state $\boldsymbol{X_i}$ and its observable state $\boldsymbol{Y_i}$. Each Candidate will be partitioned into sections that will be used as states, where the sections are as the segments discussed in Sec. 3.2, and shown in Fig. 2. Each hidden state $\boldsymbol{X_i}$ will correspond to some clusters in a

feature space, which will be converted to the parameters for a GMM, $\boldsymbol{Y_i} = \theta_{GMM,i}$, according to theory discussed in this section. By doing this, it allows the Merged model to detect more complex sections of a Candidate than if it only used a single Gaussian distribution, due to a GMM's ability to model several clusters within a single model. This use of a GMM to model the emission of observable variables dependant upon the hidden state of a HMM is known as a GMMHMM, which will be discussed in the following section.

4.4 Gaussian Mixture Model HMMs for the trails

As mentioned, a GMM can be used to model the emission from a hidden state of a HMM to the observations, which is known as a Gaussian Mixture Model HMM (GMMHMM). This makes it so that the observations can be a continuous variable, instead of the quantized values/states used in the examples in Sec. 4.2. This also applies to Gaussian HMMs, the major difference being that in a Gaussian HMM, the observations are modelled by a single Gaussian distribution. In a GMMHMM there are multiple Gaussian distributions combined to a mixture, as has been explained in Sec. 4.3.

If considering the HMM in Fig. 7, the GMM would be placed between the hidden state $\boldsymbol{X_i}$ and the observed state $\boldsymbol{Y_i}$, modelling the relationship between the observed states, S, and the hidden states, Z. This means that the emission matrix $\boldsymbol{E}$ becomes the parameters of the GMM, essientially changing the emission matrix to be emission parameters, consisting of means, covariances, and weights defining the chosen GMM. The resulting HMM will then consist of a hidden state space $Z = \{\boldsymbol{X_0}, \ldots, \boldsymbol{X_m}, \ldots, \boldsymbol{X_M}\}$, and an observable state space $S = \{\boldsymbol{\lambda_{GMM}(\theta_0)}, \ldots, \boldsymbol{\lambda_{GMM}(\theta_n)}, \ldots, \boldsymbol{\lambda_{GMM}(\theta_N)}\}$, where $\boldsymbol{\lambda_{GMM}(\theta)}$ is a GMM distribution defined by the parameters θ. This leads to some parameters, split into two groups: the transition parameters and the emission parameters. The transition parameters are simply the transition matrix, $\boldsymbol{T}$. The emission parameters, $\boldsymbol{E}$, are the parameters θ_{GMM} defining the GMMs used to model the observed data, the weights w, means $\vec{\boldsymbol{\mu}}$, and covariances $\boldsymbol{\Sigma}$ of the different components and states. The observations $\vec{\boldsymbol{O}}$ of the HMM becomes the observations O for the GMM, and is used to create the GMMHMM. Together, the transition and emission parameters define the model and allow for its implementation. As mentioned, both a normal HMM and a GMM can be trained using an EM-algorithm, which the GMMHMM can as well. Like with a normal HMM, the number of states have to be set as a hyper-parameter, as the training of the model does not alter the number of states. In addition, the number of components in the GMM also has to be set beforehand. For this project, 3 components where chosen to model the features, but this will ultimately change for different sets of features. As with choosing the number of states in the HMM, an increase in states and components will lead to more parameters, which increases the risk of overtraining.

One difference between the more "standard" HMM and the GMMHMM is the observable state space. Since this specific type of HMM has a continuous observation space that is modelled by a GMM, this allows for one major advantage: when used, it allows the model to have subsequences that might not fit the entirety of a state. Consider a situation where a trained model, λ has a hidden state space $Z = \{\boldsymbol{X_0}, \ldots, \boldsymbol{X_m}, \ldots, \boldsymbol{X_M}\}$, an observable state space $S = \{\overrightarrow{\boldsymbol{Y_0}}, \ldots, \overrightarrow{\boldsymbol{Y_n}}, \ldots, \overrightarrow{\boldsymbol{Y_N}}\}$, and is given an observation vector $\overrightarrow{\boldsymbol{O_1}} = [\overrightarrow{\boldsymbol{x_{1,0}}}, \ldots, \overrightarrow{\boldsymbol{x_{1,j}}}, \ldots, \overrightarrow{\boldsymbol{x_{1,J}}}]$,

where $\overrightarrow{\boldsymbol{x}_{1,j}}$ is a data point. In $\overrightarrow{\boldsymbol{O}_1}$, there is a section starting at index p and going to q, $\overrightarrow{\boldsymbol{O}_{1,q:p}}$, that is from hidden state $\boldsymbol{X_i}$, but some samples in the section is more akin to hidden state $\boldsymbol{X_j}$, $\overrightarrow{\boldsymbol{O}_{1,q+a:q+b}}$, where a and b are integers so that the sub-subsequence is small relative to the subsequence $\overrightarrow{\boldsymbol{O}_{1,q:p}}$. Note that $\overrightarrow{\boldsymbol{O}_{1,q:p}}$ refers to the elements within indexes q and p in sequence $\overrightarrow{\boldsymbol{O}_1}$. Model λ would still be able to sucesfully detect sub-sequence $\overrightarrow{\boldsymbol{O}_{1,q:p}}$ to be from hidden state $\boldsymbol{X_i}$, despite the samples from hidden state $\boldsymbol{X_j}$. This is becaue the probability that the rest of sequence $\overrightarrow{\boldsymbol{O}_{1,q,p}}$ comes from state i and not some other combination of states that includes state j is higher due to the samples having a higher probability than the other possible sequences. This of course assumes that the section $\overrightarrow{\boldsymbol{O}_{1,q+a:q+b}}$ is not large enough to be considered its own section and should have been designated as being generated from the hidden state $\boldsymbol{X_j}$, and that state $\boldsymbol{X_i}$ can't transition to state $\boldsymbol{X_j}$. This limit on how many samples are necesary for a subsection to be considered as being from a given hiddden state is individual to each model, and is a decided by the samples, its probability to belong to the different hidden states, and the model's transition matrix $\boldsymbol{T}$. If a sample from sequence $\overrightarrow{\boldsymbol{O}_{1,q+a:q+b}}$ fits perfectly in the distribution of hidden state $\boldsymbol{X_j}$, but the state is not available from state $\boldsymbol{X_i}$, that is, the probability of the transition is 0, or $T[i,j] = 0$, then the model is not allowed to transition to state j, and couldn't assign sequence $\overrightarrow{\boldsymbol{O}_{1,q+a:q+b}}$ to be from the state, given its already in state i. This can happen if the HMM isn't trained to allow for the transition between the two states, i.e. this transition wasn't encountered during the training of the HMM. In this application, the situation could occur if a ping is detected to be similar to the Target, but has a straight section between two turns that fit the Target perfectly. The model would then simply model the straight section at the end of the turn as a part of the turn, before continuing with the designation as a ping. The limit discussed earlier would then be how long this straight section could be before the ping should not be considered a good match any more.

4.5 Expectation maximisation algorithms for training

An expectation-maximisation (EM) algorithm is an iterative method for finding a local maximum likelihood estimate of some parameters of a statistical model, like a Hidden Markov Model, or a Gaussian Mixture Model, shown in Eq. 18

$$L(\boldsymbol{\theta}; O) = p(O|\boldsymbol{\theta}) = \int p(O, Z|\boldsymbol{\theta}) dZ \tag{18}$$

In Eq. 18, O is a a set of observed data, Z is the unobserved or missing values, and $\boldsymbol{\theta}$ is the parameters to be altered to maximise $L(\boldsymbol{\theta}; O, Z)$. There are several implementations of EM algorithms, but common to all is that they are generally split into two steps: the Expectation (E) step, and the Maximisation (M) step. During the E step, the current parameters are evaluated using the current estimate of the parameters, by finding the expected log-likelihood for each point. In Eq. 19, the E step is shown, where $Q(\boldsymbol{\theta}|\boldsymbol{\theta}^{(t)})$ is the expected log-likelihood, $\mathbb{E}[\bullet]$, of the parameters $\boldsymbol{\theta}$, with respect to the distribution Z, given O, and the current parameters $\boldsymbol{\theta}^{(t)}$

$$Q(\boldsymbol{\theta}|\boldsymbol{\theta}^{(t)}) = \mathbb{E}_{Z \sim p(\cdot|O,\boldsymbol{\theta}^{(t)})}[\log(p(O, Z|\boldsymbol{\theta}))] \tag{19}$$

During the M step, new parameters are found which maximise the log-likelihood found during the E step.

$$\boldsymbol{\theta}^{(t+1)} = \arg\max_{\boldsymbol{\theta}} Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{(t)}) \tag{20}$$

These two steps are repeated until convergence. To initialise the process, the initial parameters are typically set at random.

As mentioned, there are several kinds of EM algorithms that exist, one of which will be delved deeper into, called the forward-backward algorithm. The forward-backward algorithm is typically used to estimate the unknown parameters in a HMM.

4.5.1 Forward Backward algorithm

The forward-backward algorithm is an inference algorithm used to find the hidden state variables, $\boldsymbol{Z}$, when given a sequence of observations O, shown in Eq. 21

$$Z_{\text{pred}} = \text{argmax}(P(Z|O)) \tag{21}$$

The predicted variables are shown as Z_{pred} to indicate that they are predictions.

4.6 Creating the Garbage model

One of the models defined was named "Garbage". The Garbage model is structured as a traditional HMM as shown in Fig. 7, but with a GMM modelling the observations. The Garbage model is the model that should be able to represent any trail dissimilar to the Target better than the Seeker model, except for the Target and similar trails and sections of trails. It is designed as has been discussed in Sec. 4.2, and the implementation is shown in PC. 3. What is not shown in the PCs is a limit on how many iterations are allowed before

Pseudocode 3 Garbage training

Require: Garbage dataframe
1: number of states ← 9
2: tolerance ← 1e-4
3: obs ← Feature extraction(Garbage dataframe)
4: Garbage ← Initialise GMMHMM model
5: **while** ll diff>tolerance **do**
6: Garbage.train(obs)
7: current ll ← Garbage.score(obs)
8: ll diff ← absolute value(current ll - previous ll)
9: previous ll ← current ll
10: **end while**
11: **return** Garbage

the training stops. This is added for practical reasons, as without it the models could train and train forever without ever stopping. During testing, both models always converged after some time, however there is a non-zero chance of this not happening. The ll in lines 5, 7,

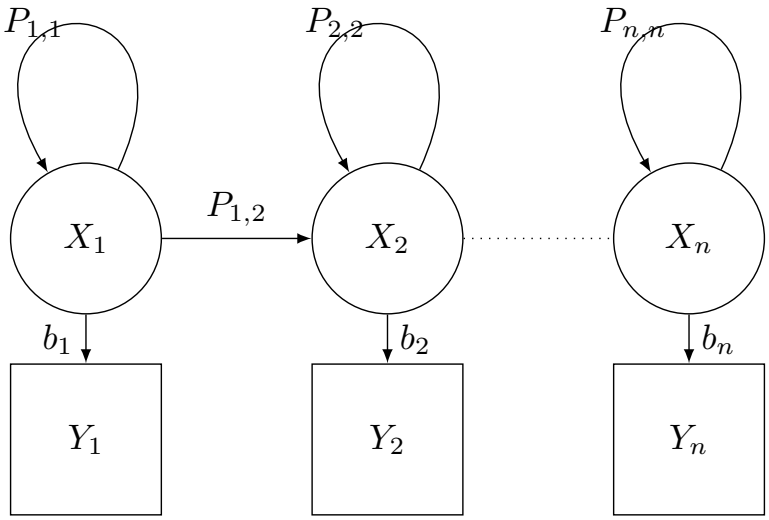


Figure 9: Time series HMM.

8, and 9 stands for log-probability. The value represents the log-probability of observing the sequence of observations O, given the model λ, as shown in Eq. 22.

$$\text{ll} = log(P(O|\lambda)) \tag{22}$$

In the equation, λ is the trained model. The model is trained using the EM algorithm. This step happens during the Garbage.train() command, as the library responsible for the implementation of the theory applies one iteration of the EM algorithm.

4.7 Structuring HMMs for time series data

In addition the method and structure described in Sec. 4.2, it is also possible to model a time-series using a HMM with some form of time as the hidden states, or the process that evolves over time. The hidden states then become an abstract measure of the specific process that produces the observed variables over time, which in this case could be the coordinates, or the features of the trail. A visual representation of such a model is provided in Fig. 9, where the HMM is shown to have n states. If $n = 4$, a transition matrix for a time series HMM can be found in Tab. 6. For longer series, more states are added, but no other changes are made.

	$\boldsymbol{X_0}$	$\boldsymbol{X_1}$	$\boldsymbol{X_2}$	$\boldsymbol{X_3}$
$\boldsymbol{X_0}$	$1 - a_0$	a_0	0	0
$\boldsymbol{X_1}$	0	$1 - a_1$	a_1	0
$\boldsymbol{X_2}$	0	0	$1 - a_2$	a_2
$\boldsymbol{X_3}$	0	0	0	1

Table 6: Transition probabilities, $\boldsymbol{T}$, for a time series HMM.

This corresponds to simply having more rows and columns in the transition matrix. In Tab. 6, variable a is set so that it accurately models the transition between the states. Note that the last state only transitions to itself, as this is the "end" of the model. This will be changed when the models are merged, as the model has to be able to transition from this state. If the model is to be used for another purpose, one might change the transition matrix to allow for looping back to the first state of the model. There are some major differences between the HMM in Fig. 6 and the one in Fig. 9, mainly that there is a set direction of the process, which is seen in the transition matrix. The initial probabilities are typically set to 0, except for the first state, as the sequence starts in the first state. Other than this, the time-series HMM doesn't differ from a "normal" HMM, still having the transition matrix $\boldsymbol{T}$, emission parameters $\boldsymbol{E}$, and initial probability vector $\vec{\boldsymbol{\pi}}$. Together these define the hidden state space $Z = \{\boldsymbol{X_0}, \ldots, \boldsymbol{X_n}, \ldots, \boldsymbol{X_N}\}$, and the observable state space $S = \{\overrightarrow{\boldsymbol{Y_0}}, \ldots, \overrightarrow{\boldsymbol{Y_m}}, \ldots, \overrightarrow{\boldsymbol{Y_M}}\}$. In order to model a time series, the data is processed, and then split into appropriate states, e.g. the segments from Fig. 2. When the data has been segmented, the segments are processed, i.e. finding a feature vector that can represent the segment, as explained in Sec. 3.4. After the emission-probabilities are found, the HMM is trained using an appropriate algorithm, like the EM algorithm, as shown in Sec. 4.5. The implementation of this model, will be named the "Seeker", and will be shown later.

4.8 Creation of the Seeker model

As has been explained, the system is to be implemented using two different models, the first model being the Garbage-model implemented in Sec. 4.6 and explained in Sec. 4.2. The theory for the second model has been outlined in Sec. 4.7. The model will be named "Seeker". The Seeker model is the model that should be able to represent trails dissimilar to the Target worse than the Garbage model, but capture the Target and similar trails and sections better than the Garbage model. It is structured as a time series HMM, as shown in Fig. 9, and Sec. 4.7. Recall the Target as shown in Fig. 2. There are six distinct groups, which can be visualised as the states the observations are grouped into. Some tests performed and shown in Sec. 6 experiment with the number of states the Seeker-model is allowed to have, but here six states will be used to help visualise the Seeker. For another Target, more or less states might be considered, however as shown in Sec. 6, the method is somewhat resilient towards too few and too many states. Although the Target in the figure is segmented according to the process described in Sec. 3.2, the Seeker model divides the observations into equal length groups, where the number of groups are the same as the number of states. This is because the first state in the Seeker should represent the first group of observations, the second state represents the second group, etc. The ltr transmat returns a transition matrix so that the model becomes as shown in Fig. 9, where $a = \frac{1}{l}$ and l is the number of groups. This makes it so that the first group is captured in the first state as desired. The matrix itself can be found in Tab. 6, for $n = 4$ states.

After both models have been trained, they have to be combined in a manner that allows both models to be available to evaluate a sequence of observations, which will be shown in both implementation and theory later.

Pseudocode 4 Seeker training

Require: Target dataframe
number of states ← 6
tolerance ← 1e-4
obs ← Feature extraction(Target dataframe)
Left to Right transition matrix ← ltr transmat(number of states)
Seeker ← Initialise GMMHMM model
Seeker.transmat ← Left to Right transition matrix
while ll diff>tolerance **do**
    Seeker.train(obs)
    current ll ← Seeker.score(obs)
    ll diff ← absolute value(current ll - previous ll)
    previous ll ← current ll
end while
return Seeker

5 Creation and implementation of the Merged model

This section will discuss how the models created in the previous section is merged. The reasons as to why the Merged model was created has been given in Sec. 3.7. In addition, there will also be some pseudocode showing how the Merged model can be applied to a database. This PC is an implementation of the process described in Sec. 3.7, applied to each Candidate in the database.

5.1 Combining HMMs

As has been previously mentioned, the two different HMMs are combined into a single model to use for the evaluation of a Candidate. The two models are the Garbage model, and the Seeker model. The Garbage model should be able to model any sequence, while the Seeker should be able to model Target-like sequences. These names will be used in this section, as well as a subindex, where 1 refers to the Seeker and 2 refers to the Garbage. The model created by the combination of the two original models will be refered to as the "Merged model". In this section will also be a further explanation of what the different sub-models are supposed to acheive within the Merged model.

The Garbage model can be trained on the entire database, meaning that the probability for over-fitting is drastically reduced due to the larger training data. As described above, the Garbage model is meant to be able to model all sequences, but worse than the Seeker model for the sequence that is to be found. It does however need to model all other sequences better than the Seeker. If both these behaviours are present, the Seeker-Garbage-combination model can be used to detect when a given sequence is present in another, longer sequence by seeing when the combination-model is using the Seeker-model.

5.1.1 The Seeker model and its purpose

As mentioned, the Seeker model is the model that is supposed to detect when a section of a Candidate is similar to the Target. To do this, the Seeker model has to be re-trained to the different Targets, which means it can run into issues related to small amounts of data, such as overfitting and lack of representativeness. The model will most likely not run into issues related to underfitting, as it is possible to simply increase the complexity to fit the assumed standard length of a Target, although this would increase the risk of overfitting.

The structure of the seeker model should be as described in Sec. 4.7, that is linear and directional. Since it should be able to capture a section *similar* but not necessarily *identical* to the Target, the model needs to be general enough to allow some level of difference between the two, but not large enough to falsely detect sections that are not similar enough. This distinction is vague and not easily defined, as the level will depend on the specific sport the HMM is applied to, as different sports allow for different levels of similarity. In addition to this comes the subjective nature of what can be considered a good match, discussed in Sec. 2.1. As such, this has to be able to be changed, and also presented to the end-user so that it can fit their needs. For this project, an arbitrary limit will be set and used, as this process is likely arbitrary and trivial for the scope of the project, due to the subjectivness of the real limit. The Seeker should as mentioned be complex enough to avoid underfitting, and simple enough to avoid overfitting. Both of these terms relate to the amount of data have available for training, which in turn becomes the length of the Target. Examples of Targets that have been presented has been approximately 1.5 km, with ≈ 180 samples. To model a Target, it is possible to either choose a set amount of hidden states, or find the "correct" amount of hidden states for each new Target. Regardless, when the desired amount of hidden states is found, the model can be trained on either a single sequence, or multiple sequences that can be created through data augmentation.

5.1.2 The Garbage model and its purpose

The Garbage model is supposed to adequately model any sequence. In comparison to the Seeker model, the Garbage model should be better at representing sections of a Candidate that are not similar to the Target. In reverse, the Seeker should be better at representing any section similar to the Target. When combining both of these "rules" the end goal becomes being able to model any sequence, but not be so good that the Garbage model outperforms the specialised Seeker model on Target-like sequences. Since the Garbage model is meant to model any sequence, it has the entirety of the database of Candidates to train upon, and will thus have a severly decreased risk of overfitting when compared to the Seeker model.

The Garbage model could be modelled as a time-series, like the Seeker model, but it can also be modelled as a more general HMM structure. For this application, a more general structure is the better option as it allows the Seeker to out compete it when a similar section is found, as well as a better fit for the large number variety in the data the Garbage model will be trained on. When building the model, the observable states of the HMM has to be given in the same format as the one provided to the Seeker, in order to keep compatibility between the models, that is being modelled by similar models, e.g. GMMs.

5.1.3 Combination of the two models

With the assumption of the training of the separate models being done properly, the last step is to combine the models into one. Recall that the Seeker model has subindex 1 and the Garbage model has subindex 2. The new HMM will not have a designated index, but rather just be the lack of an index. The first step would be to concatenate the hidden states, as well as keeping the transition probabilities between the states, which is shown in equations 23, and 25.

$$Z = Z_1 \cup Z_2 \tag{23}$$

Where Z_i is the hidden states for model i. In Eq. 24 the combination of the observations is shown.

$$S = S_1 \cup S_2 \tag{24}$$

Where S_i is the observable states for model i. Due to the hidden nature of the hidden states, they cannot be combined directly, and has to be done by combining the transition matrices that define the different state spaces.

$$\boldsymbol{T} = \begin{bmatrix} \boldsymbol{T_{11}} & \boldsymbol{T_{12}} \\ \boldsymbol{T_{21}} & \boldsymbol{T_{22}} \end{bmatrix} \tag{25}$$

Where $\boldsymbol{T_{ii}}$ is transition matrix for model i, and $\boldsymbol{T_{12}}$ and $\boldsymbol{T_{21}}$ contains the transitions between the models. The emission parameters will be further discussed in Sec. 5.1.5. Note that for the combination of the parameters, the notation is almost like in a pseudocode, where $A = [B, C]$ means that A consists of all elements in B, which are placed before the elements of C. This is applicable to equations 25 and 35.

The only new information that has to be added to the combined model is the transitions between the two states. Due to the nature of the system created, the inter-model transition matrices can be filled with some assumptions. Since the Seeker model has a set sequence, the only states that should be able to transfer to it's states is either the state itself, or the one preceding it. The only exception to this assumption will be the first state from the Seeker model, as it is always possible to start the sequence modelled by the Seeker model. Transitions from the Seeker model is also simple, as the sequence always either ends, or transfers to the Garbage states, i.e. the last state in the Seeker model can transition to any state in the Garbage model. This will be shown in the following section.

5.1.4 Creating $\boldsymbol{T_{12}}$ and $\boldsymbol{T_{21}}$

As discussed in the previous sections, some information must be added into the system when merging the two types of models created. Without this addition, the Merged model will not work as intended, as it would not be able to switch between the models, and be stuck in the single model that fits the entirety of a Candidate, not being able to detect subsections within the sequence that relates to the different models. In other applications, like keyword spotting, the transition values can be found by studying rules in grammar or other language-related systems. Likewise, the values in this application can be found by applying some logic and assumptions. Regardless of the values assigned to the inter-model transitions, there are

some rules that need to be followed. As known, the sum of the transitions from a state has to be 1, otherwise the system has the possibility of simply ceasing to exist. To not break this rule, it is important that the sum stays the same, but allowing for the inclusion of n states, where n are the states the model can transition to. This is done by splitting the sum into two parts: the first being the transitions to the state within the model, $\boldsymbol{T_{ii}}$, and secondly the transitions going to the other model $\boldsymbol{T_{ij}}$. Here i and j refer to the index of the different models used to create the Merged model, and changes depending on which one is considered. $\boldsymbol{T_{ij}}$ is the transitions between models, and $\boldsymbol{T_{ii}}$ is the transitions within a model, that is the original matrices: $\boldsymbol{T_{11}}$ or $\boldsymbol{T_{22}}$. For the Seeker model ($\boldsymbol{T_{11}}$), only the transitions from the last state are altered, as the entire Target-sequence is required before a section can be considered a good match for the Target. This means only changing the bottom row of the transition matrix, as this relates to starting in the last state of the Seeker-model. For $\boldsymbol{T_{22}}$ it is simpler, as all states must have the possibility of transitioning to the Seeker-model. To change the original transition matrix $\boldsymbol{T_{ii}}$, the ratio between the new sum and the old sum is found and multiplied with the different probabilities present in the model. The finding of the sum is shown in Eq. 27, where r is the ratio between the two sums.

$$r = \frac{S_2}{S_1} \tag{26}$$

Note that in Eq. 26, S_2 is the modified sum of the transitions from a state, while S_1 is the old sum, in essence making r become a user defined probability of transitioning between models, since $S_1 = 1$. Shown in Eq. 27 is the altering of the Garbage-transition matrix.

$$\boldsymbol{T_{22}^n} = \boldsymbol{T_{22}} \times r_2 \tag{27}$$

$\boldsymbol{T_{22}^n}$ is the new transition matrix, and replaces the $\boldsymbol{T_{22}}$ matrix in the Merged model transition matrix although not explicitly stated in Eq. 25. In Eq. 27, all elements are affected by the operation. For the $\boldsymbol{T_{11}}$ transition matrix, the operation looks as shown in Eq. 28.

$$\boldsymbol{T_{11}^n}[N] = \boldsymbol{T_{11}}[N] \times r_1 \tag{28}$$

Where N refers to the row to be modified, being the last, or bottom, row. For the remaining rows, $\boldsymbol{T_{11}^n} = \boldsymbol{T_{11}}$. This means that only row N is affected by the operation, unlike in Eq. 27. When considering equations 27 and 28, note that the two different ratios are not necessarily the same, as the probability of going from the Seeker to the Garbage and the probability of going from the Garbage to the Seeker models might not be the same. For the case of going from the Garbage model to the Seeker model, $\boldsymbol{T_{21}}$, there is a single state: the initial state in the Seeker model, or in other words start the Target sequence . For the case of going from the Seeker model to the Garbage model, $\boldsymbol{T_{12}}$, there are multiple states: all the states of the Garbage model, as the Seeker-model must be able to transition to any state in the Garbage model. The altering of the existing transition matrices is simply a downscaling of the magnitudes of the transition probabilities, as shown above.

When creating the new transition matrices, $\boldsymbol{T_{12}}$ and $\boldsymbol{T_{21}}$, the remaining, user defined, probabilty, r_i, is split across the states that the transitions are added for. For the Seeker→Garbage transitions, the number of new transitions equal the number of states in the Garbage-model.

For the Garbage→Seeker transitions, there is only one new transition: the initial state of the Seeker-model. These assumptions also determine where the probability is placed in the transition matrix, or its individual rows. It might also be useful to note that traditional transition matrices are square, as they model the probability of going from a state to a state, and this results in a square matrix. The new transition matrices $\boldsymbol{T_{12}}$ and $\boldsymbol{T_{21}}$ are not traditional matrices, as they are not square. When combined with the old transition matrices they create a new square matrix, as shown in Eq. 25, but are not square themselves, unless the two models have the same number of states. When considering a Seeker-transition matrix of $N-1$ states, or a N by N transition matrix, and a Garbage transition matrix of $M-1$ states that should be merged the result is as such:

$$\boldsymbol{T_{12}} = \begin{bmatrix} t_{0,0} \ldots, t_{0,m}, \ldots, t_{0,M} \\ \vdots, \ddots \\ t_{n,0} \ldots, t_{n,m}, \ldots, t_{n,M} \\ \vdots, \ddots \\ t_{N,0} \ldots, t_{N,m}, \ldots, t_{N,M} \end{bmatrix} \tag{29}$$

The matrix shown in Eq. 29 is not square, unless $N = M$, which is not always the case. Since the $\boldsymbol{T_{12}}$ matrix is supposed to capture the transitions from the Seeker model to the Garbage model, the row that is altered is the N-th row. The rest of the matrix is kept as zero, $t_{n,m} = 0$ for $n \neq N$, as none of the other states in the Seeker model is allowed to transition to the Garbage model. The operation of finding the new transitions looks as shown in Eq. 30, where r_1 is the user-defined probability of moving from the Seeker to the Garbage model, and x is the number of transitions, which in this case becomes M, or the number of states in the Garbage model.

$$t_{N,m} = \frac{r_1}{x} \tag{30}$$

In total, the new transition matrix looks as such:

$$\boldsymbol{T_{12}} = \begin{bmatrix} 0 \ldots, 0, \ldots, 0 \\ \vdots, \ddots \\ 0 \ldots, 0, \ldots, 0 \\ \vdots, \ddots \\ \frac{r_1}{M} \ldots, \frac{r_1}{M}, \ldots, \frac{r_1}{M} \end{bmatrix} \tag{31}$$

This setup has two assumptions: the first being that the Seeker state can exit into each state in the Garbage model with equal probability, and that only the last state in the Seeker is allowed to transition to the Garbage model. The second assumption is part of the demands of the problem, as breaking this would mean that matches for sections of the Target is allowed. The first assumption can be changed if it is discovered that there is a major overweight of one kind of state that dominates the Garbage model.

The creation of $\boldsymbol{T_{21}}$ is similar to the creation of $\boldsymbol{T_{12}}$, the only difference is the rows and

columns, as shown in Eq. 32

$$\boldsymbol{T_{21}} = \begin{bmatrix} t_{0,0} \ldots, t_{0,n}, \ldots, t_{0,N} \\ \vdots, \ddots \\ t_{m,0} \ldots, t_{m,n}, \ldots, t_{m,N} \\ \vdots, \ddots \\ t_{M,0} \ldots, t_{M,n}, \ldots, t_{M,N} \end{bmatrix} \tag{32}$$

For this transition matrix the altering happens to the first, or left-most, column in the matrix as such:

$$t_{m,0} = \frac{r_2}{x} \tag{33}$$

Where r_2 is the user-defined probability as described earlier. Here, x is again the number of states that has transitions added, which now is 1, so that matrix $\boldsymbol{T_{12}}$ becomes as shown in Eq. 34. The rest of the matrix-elements is again set to zero.

$$\boldsymbol{T_{21}} = \begin{bmatrix} r_2 \ldots, 0, \ldots, 0 \\ \vdots, \ddots \\ r_2 \ldots, 0, \ldots, 0 \\ \vdots, \ddots \\ r_2 \ldots, 0, \ldots, 0 \end{bmatrix} \tag{34}$$

This assumption is due to not wanting partial matches for the Target.

Once the two inter-model transition matrices have been created, they can be combined with the known transition matrices as shown in Eq. 25 along with the rest of the parameters, and the Merged model is ready to be used as a normal HMM. Note that for the implementation of the theory, r_1 and r_2 has been exchanged with p_{12} and p_{21}

5.1.5 Other parameters

In addition to the transition matrices, the emission variables have to be combined, which is the parameters defining the GMMs, being weigths, means and a matrix of covariances. In addition, there is the initial probabilities, which undergo the same operation. These parameters are much easier to combine, as they only require to be placed in the correct place. With the method described above, where the Seeker is placed to the top left and the Garbage is placed in the bottom right of the transition matrix, the Seeker's GMM is placed before the Garbage's GMM as shown in Eq. 35.

$$\boldsymbol{E}_{merged\ model} = [\boldsymbol{E}_{Seeker}, \boldsymbol{E}_{Garbage}] \tag{35}$$

In the equation, E is the emission parameters of the different models named in the subindex, the different θ_{GMM} trained during the different models individual training. Together this creates a new model that has the statespace of both the previous models.

5.2 Merging the models

The models created earlier are merged into one, creating a new HMM that is a combination of both, as shown in Sec. 5.1. Since the previous models have been trained, it is not necessary to train the new model. There is an assumption that both previous models are properly trained. As was discussed when presenting the theory, some new information must be introduced to the model, being the p_{21} and p_{12} values. During testing, it was discovered that the best levels for these values was 0.9 and 0.8 respectively. Interesting to note is that the results and behaviour of the model is more sensitive to the emission parameters rather than the transition parameters, which is in correspondance with advice given by Giampiero Salvi during weekly meetings. This can be seen in the mentioned tests, as the behaviour doesn't drastically change between parameters. The tests can be found in Sec. 6.

Pseudocode 5 Merge models

Require: Seeker,Garbage,p_{21},p_{12}

S1 ← number of Seeker states
S2 ← number of Garbage states

Initial probabilities 1 ← Seeker initial probabilities
Initial probabilities 2 ← Garbage initial probabilities
Initial probabilities ← Concatenate(Initial probabilities 1, Initial probabilities 2)

W1 ← Seeker Weights
W2 ← Garbage Weights
W ← Concatenate(W1,W2)

M1 ← Seeker means
M2 ← Garbage means
M ← Concatenate(M1,M2)

Covar1 ← Seeker Covariance
Covar2 ← Garbage Covariance
Covar ← Concatenate(Covar1,Covar2)

T11 ← Seeker Transition matrix
T22 ← Garbage Transition matrix
T12 ← Create transition matrix(p_{12}), shaped to S1 by S2
T21 ← Create transition matrix(p_{21}), shaped to S2 by S1
T ← Combine([T11, T12],[T21, T22])

Merged ← Create GMMHMM
Merged.Transition matrix ←T
Merged.Weights ← W
Merged.Covariance ← Covar
Merged.Initial probabilities ← Initial probabilities
return Merged

In PC. 5 the theory and method from Sec. 5.1 is implemented. In the PC., the "Create transition matric(•) creates the appropriate matrix, as shown in the previous section.

5.3 Application

At this point, the Seeker model has been trained, as well as the Garbage model. In addition, they have been combined into the Merged model. The database has been created, and the Target is available. What's left is to use the model. As mentioned earlier, the model will be given a single Candidate at a time, and it will decode the sequence, which will return the probability of the sequence being generated by the Merged model, as well as the sequence of

hidden states that most closely resembles them. This list will be analyzed and the Seeker-states will be extracted and presented as the pings. This process will be shown in PC. 6

Pseudocode 6 Using the Merged model

Require: Database, Merged model
scores ← list of length equal to number of Candidates
states ← list of length equal to number of Candidates
metrics ← list of length equal to number of Candidates
pings ← list of length equal to number of Candidates
for Candidate in Database **do**
    Observations ← Feature extraction(Candidate)
    Predicted Scores,Predicted states ← Merged model.decode(Observations)
    scores[Candidate number] ← Predicted Scores
    states[Candidate number] ← Predicted states
    metrics[Candidate number] ← Merged model.predict_proba(Observations)
    pings[Candidate number] ← Detect and split consecutive numbers(Predicted states)
end for
return pings,metrics,scores

Note that for the metric and pings, only states related to the Seeker are considered, which in practice becomes looking at the first $N-1$ columns, where N is the number of Seeker states. In the pings-list, the values stored are indexes, as the implementation is mainly working with dataframes, and the extraction of the pings is made easy when there is a list of indexes. If indexes are close together, they are combined into one pings, as they most likely are actually from the same section, and functionaly they will be close together and can be used together for an athlete. After this, the resulting subsections are stored as lists in the pings-list, and are presented as plots for the testing. When using the Merged model.decode($\bullet$), the model uses the Viterbi Algorithm to find the most likely sequence of hidden states.

6 Generality of the Merged model

This section aims to explore the generality of the model, as well as how some of the parameters affect the behaviour of the Merged model. This exploration will be conducted by testing the model, and looking at how well it recognizes the Target, with increasing levels of noise. The structure of the test will be shown in Sec. 6.1, and the expected results of this test will be shown in Sec. 6.2. The results will be shown in Sec. 6.3. In Sec. 7, a function test will be performed, where the functionality of the model will be tested.

6.1 Generality test

The generality test has the goal of testing how different a Candidate, or its subsections, can be before they are no longer recognized as good matches for the Target. This test is performed seperately from the function test, as it is desireable to not only have quantised differences

available in the actual, recorded Candidates, but rather to have a great degree of control over the Candidates being fed to the Merged-model. This is done by first training a model, $\lambda_g(\boldsymbol{T}, \boldsymbol{E}, \boldsymbol{\pi})$, and then extracting the emission-parameters, i.e. the weights $\vec{\boldsymbol{w}}$, means, $\overrightarrow{\boldsymbol{\mu_g}}$, and covariances, $\boldsymbol{\Sigma_g}$, of the trained model, before modifying it to create some GMMs. The GMMs can then be used to generate vectors of noise that can be added onto the Target to create new Candidates. By doing this, it is possible to control the level of difference between the Target and the Candidate, by adjusting the variance in the GMM. Further, it allows for exploration of the behaviour of the Merged-model when working on trails, and looking at how the model responds to differing levels of similarity between a Target and a Candidate.

The test will be performed by using a model to create a vector of noise, $\overrightarrow{\boldsymbol{n_{v,i}}}$, for each level of variance, v, for I iterations, shown in Eq. 36.

$$\vec{\boldsymbol{n}}_{v,i} \sim \mathcal{N}(\boldsymbol{\mu_g}, \boldsymbol{\Sigma}_g \times v) \tag{36}$$

After the noise has been generated, it is added to the Target, creating a new Candidate that differs from the Target by only these values. From these Candidates the features that have been discussed earlier will be extracted and provided to the Merged model.

$$C_{v,i} = \vec{\boldsymbol{T}} + \vec{\boldsymbol{n}}_{v,i} \tag{37}$$

When performing the test the areas of interest will be the average metric produced by the model when evaluating the observations, as well as how many samples are correctly identified, on average, to be a trail that is similar to the Target, which will be named the Coverage. For each variance-level v at least 100 iterations will be generated, and at most 500, however, like when training the models, if the means of the individual tests converge, the next variance-level will be started. The test will also explore how different parameters affect the performance. The parameters changed will be the number of states in the Seeker-model, and the p_{21} value. A short disucssion of the p_{12} value will also be included, but the behaviour of this parameter is expected to be closely linked to the behaviour of the p_{21} value.

6.2 Expected Results of Generality

This section will outline the expect results from the generality tests, with the acheived results shown in Sec. 6.3. The model is expected to initialy not struggle with identifying the modified Target as a good potential match for the real target, meaning that the coverage will be high. As the the amount of noise added to the sequence is increased, the Coverage is expected to decrease, as this would be the same as moving further and further away from the pure Target. The Coverage is expected to start high, at 90% to 100%, before a steep slope and reaching an end somewhere around 5% to 0% coverage. The test will be performed with different parameters, and the behaviour is expected to be approximately equal across all setups, with the exclusion of $p_{ij} = 0$ and $p_{ij} = 1$ for the transition-parameters, as these values remove the duality-aspect of the model. If the probability of going to one of the models is set to 1, the other model effectively ceases to exist, as it will always transfer to the first model after a single state. Likewise, if the probability is set to 0, the Merged model will never switch between the two models, always being stuck in the model that best fits the entire Candidate, which in almost all real cases will be the Garbage model.

6.3 Results from the testing of the generality

The test for the level of generality of the model(s) was performed as described in previous section; generating a number of random noise vectors and adding them to the Target to create new Candidates. For the test, 500 different vectors where generated at each variance-level. The noise was iid-noise.

Since the model tries to recognize sequences of samples as either coming from the Seeker-model or the Garbage-model, adding noise to the Target and giving it to the Merged model allows for the determination of what level of difference the model fails to recognize a subsection as similar to the Target, as well as helping visualising and understand the performance of the model. Presented in Fig. 10 is data from a test, where the model parameters are as shown in Table 7. In Fig. 10 through 14, there are two subplots. The upper subplot

Table 7: Parameters used for test 1.

Parameters	**Value**
Seeker states	5
Garbage states	9
p_{12}	0.6
p_{21}	0.75

contains the "score", which is a simpler name for the log-probability of the sequence being generated by the Merged model. This can be expressed mathematically as shown in Eq. 38, which shows a measure of the probability $P(\overrightarrow{\boldsymbol{O_T}}|\lambda_M)$ that a model λ_M has generated the observations $\overrightarrow{\boldsymbol{O_T}}$.

$$\text{Log probability} = log(P(\overrightarrow{\boldsymbol{O_T}}|\lambda_M)) \tag{38}$$

This expresses the logarithm of the probability that a sequence is generated from the model. The bottom subplot contains the "Coverage", which is simply the percentage of the samples deigned to be from the Seeker-model, and not from the Garbage-model.

$$\text{Coverage} = \frac{n}{m} = \frac{|\overrightarrow{\boldsymbol{X_{pred}}} \cap Z_S|}{|\overrightarrow{\boldsymbol{X_{pred}}}|} \tag{39}$$

In Eq. 39, $\overrightarrow{\boldsymbol{X_{pred}}}$ is the vector of predicted states given the observations, $n = |\overrightarrow{\boldsymbol{X_{pred}}} \cap Z_S|$ is the number of predicted states that are from the Seeker-model, indicated by the subindex S , and $m = |\overrightarrow{\boldsymbol{X_{pred}}}|$ is the length of the entire vector.

Another setup and its result is shown in Fig. 11, with the parameters shown in Tab. 8 As seen, the performance isn't as good as in test 1, indicating that for this particular Target the correct number of states was 5 rather than 4. This might change depending on the Target, but can be adjusted by the end user to capture the level of likeness desired between a Target and a Candidate. Interesting to note in the figures is the overall shape of the coverage of the models. The steep fall indicates that there is a point at which the Model struggles to recognize the altered Target when it reaches a certain level of difference. This behaviour is good, as it

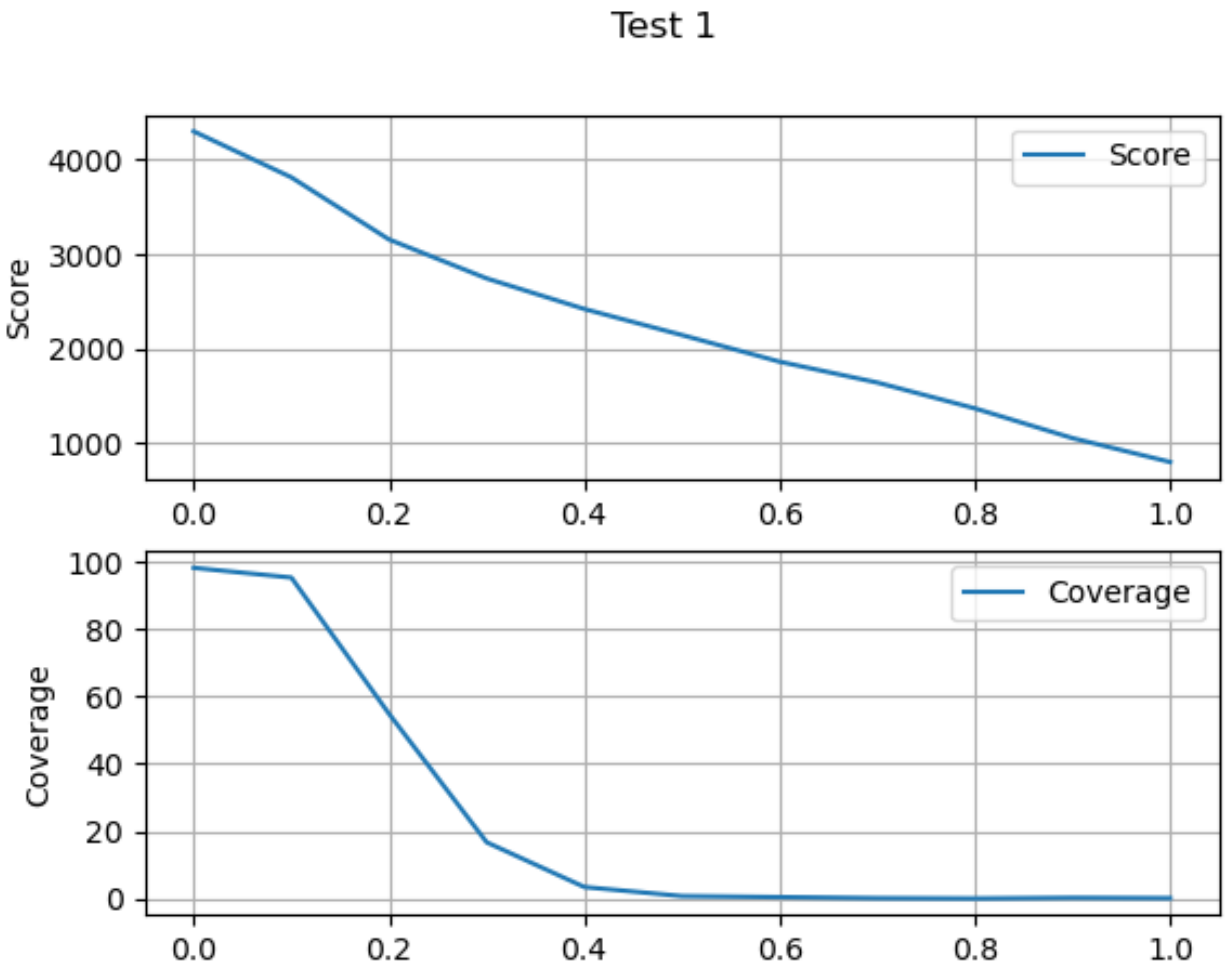


Figure 10: Results from setup 1 of the model's generality.

Table 8: Parameters used for test 2.

Parameters	**Value**
Seeker states	4
Garbage states	9
p_{12}	0.6
p_{21}	0.75

means that the model doesn't just randomly designate subsections as a match for the Target, but rather checks to see how likely it is that a sequence of samples is from the Seeker-states. By changing the parameters in tables 7 and 8, it is also possible to control where this point is.

Shown in Fig. 12 is the results from another setup, shown in Tab. 9. By comparing figures 10 and 12, as well as their corresponding tables, it can be seen that the transition values doesn't affect the performance greatly. This is as expected, as the transition-parameters are less important than the emission-parameters and tend to be overpowered by those in HMMs.

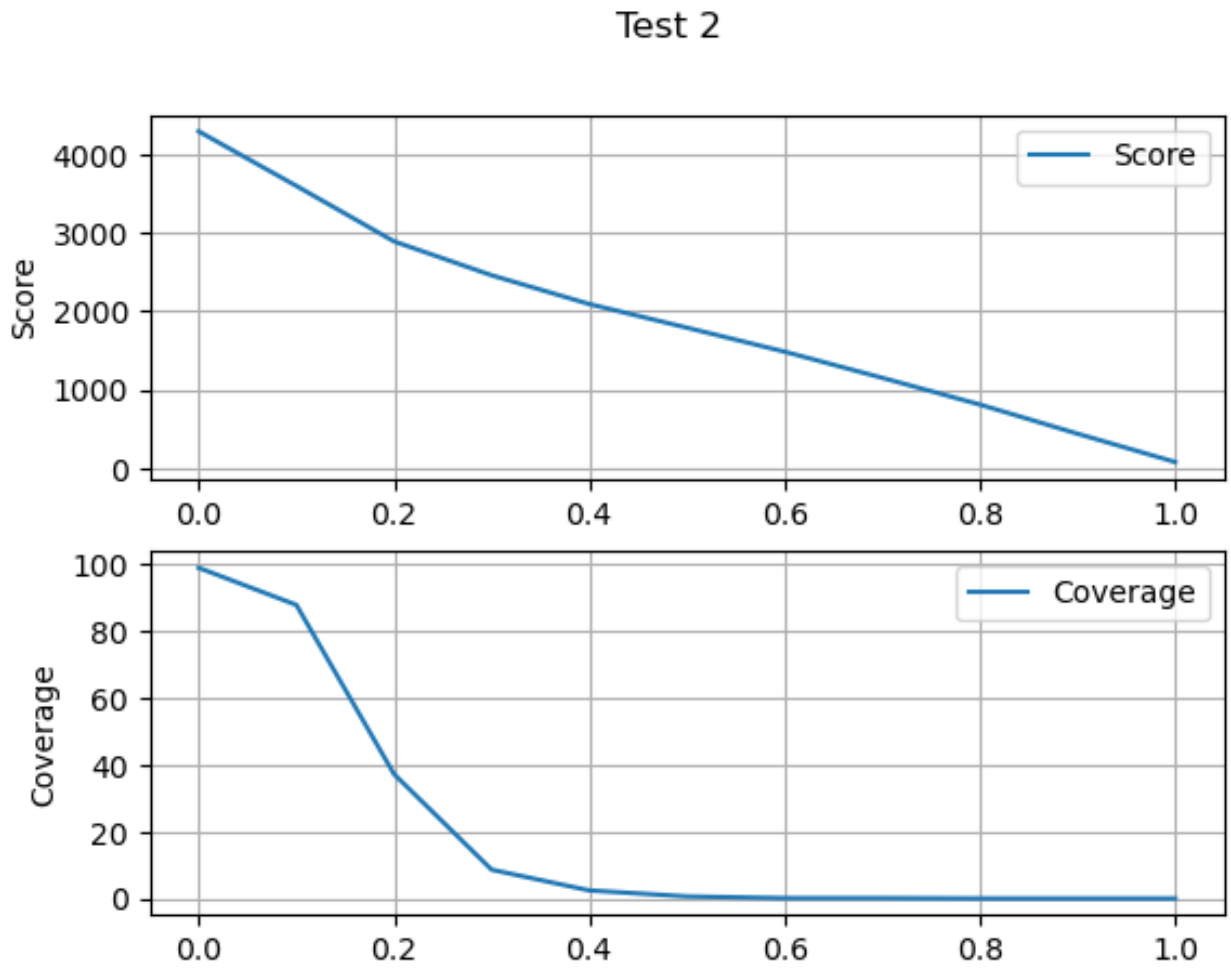


Figure 11: Results from setup 2 of the model's generality.

This of course excludes the edge-cases of $p_{ij} = 0$ and $p_{ij} = 1$, as has been discussed.

Table 9: Parameters used for test 3.

Parameters	**Value**
Seeker states	5
Garbage states	9
$p_{1,2}$	0.6
$p_{2,1}$	0.5

Testing for further exploration of the effect of the number of Seeker-states on the performance of the Merged model can be seen in Fig. 13. Easily visible in the figure is that changing the number of states in the Seeker-model greatly affects the performance of the model, and the generality of the model. Although for only generality a flat 100% coverage might be thought good, for the desired behaviour it is required that the model fails to recognize the Target

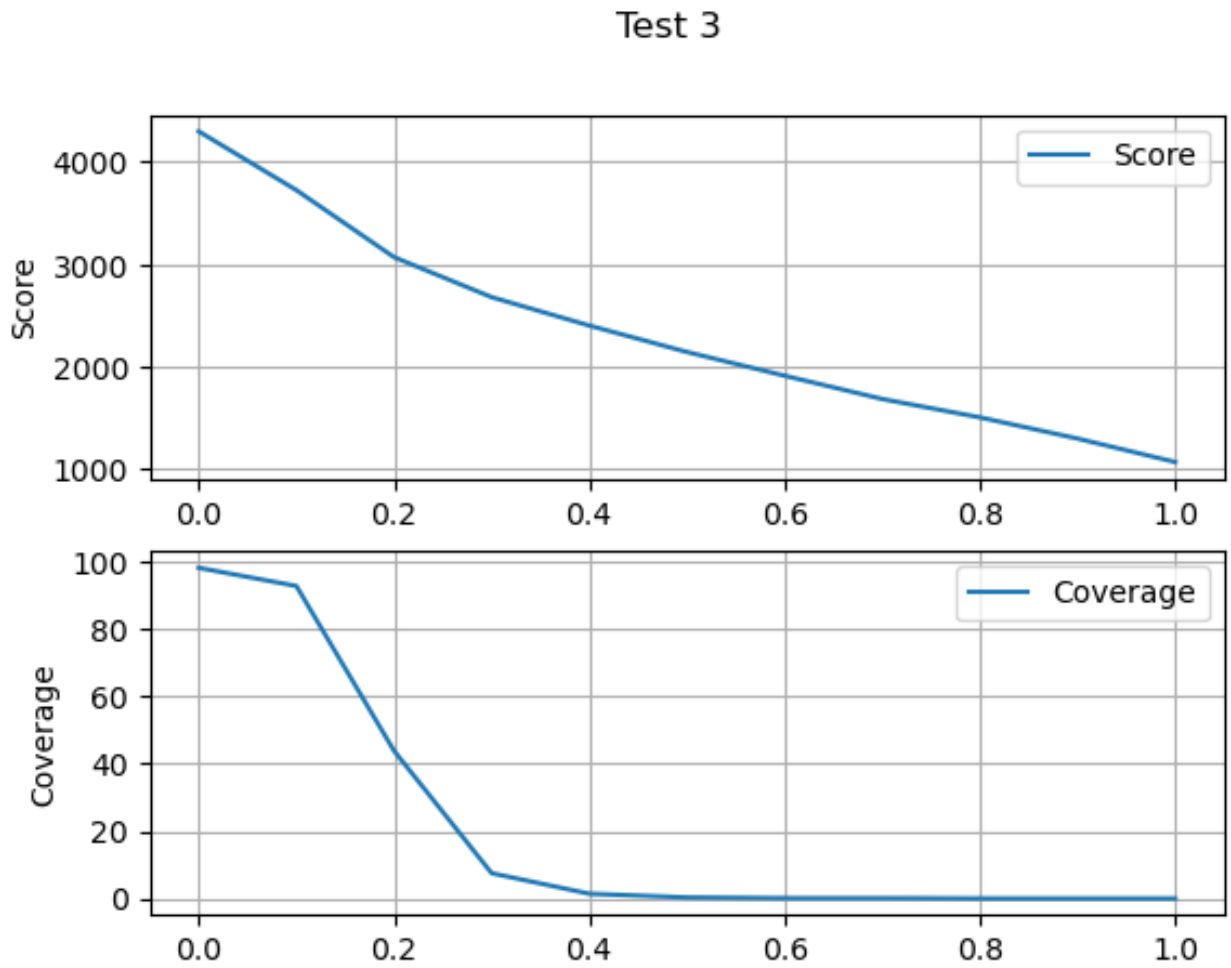


Figure 12: Results from setup 3 of the model's generality.

after some level of noise, as a noisy target equates to a Candidate that doesn't contain a good match. As has been explained earlier, the exact level of variance accepted is individual, and will change between users, models, and Targets. Interesting to note is that the performance isn't linearly related to the number of states in the Seeker, indicating that both too many and too few states are detrimental to performance. For the test of different number of states, p_{21} was set to 0.5.

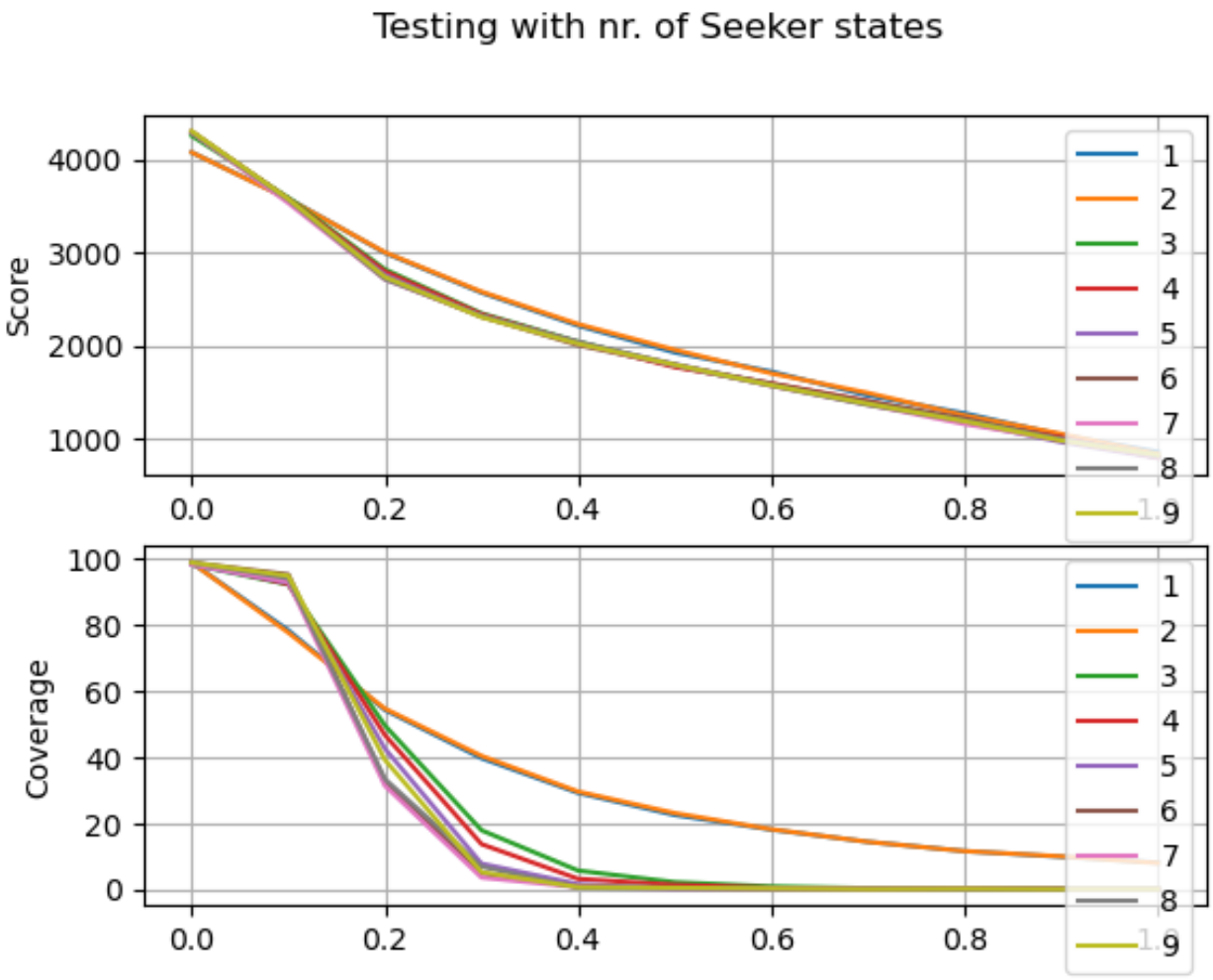


Figure 13: Results with different states in the Seeker.

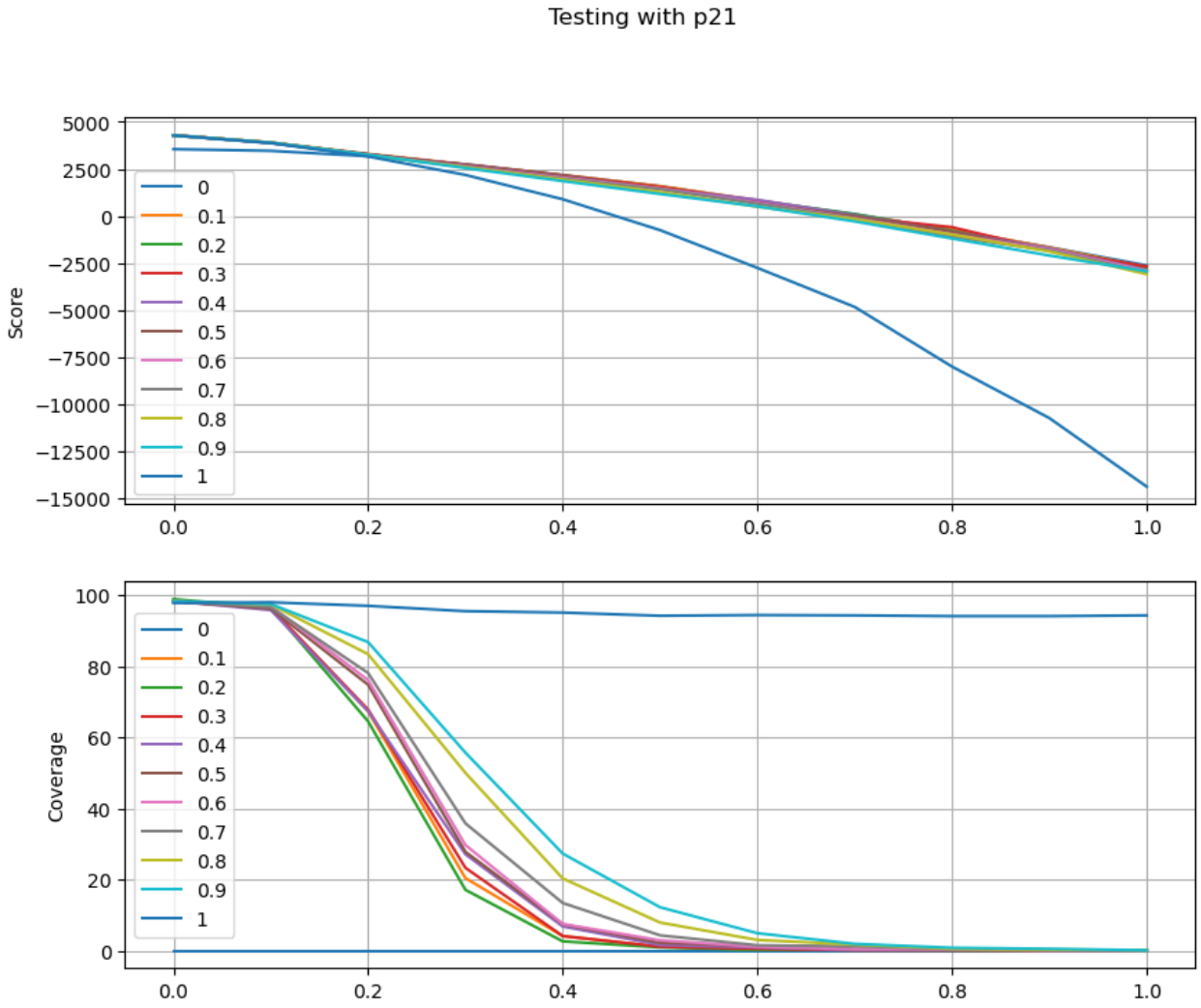


Figure 14: Results with different values of p_{21} in the Merged model.

Shown in Fig. 14 is the results from a tests changing the p_{21} value. As seen, the highest overall coverage is achieved with a p_{21} value of 0.9, closely followed by a value of 0.8, while the worst are 0.1 and 0.2, if the edge-cases are excluded, for reasons discussed earlier. Increasing p_{21} allows for higher levels of difference that the model accepts as similar to the Target, which makes sense. Testing with p_{12} produced almost similar results, only with the edge-cases producing the same behaviour, the Merged model effectively stopped using the Garbage model. This is different from the behaviour of the p_{21} parameter, as one of the behaviours there resulted in the Seeker model ceasing to exist, visible by the score falling extremely low in Fig. 14. Since the other behaviours are so similar, the data isn't discussed to great detail, as the changes to the behaviour is the same, only opposite. Increase in the probability of moving from the Seeker to the Garbage model, p_{12}, makes the Merged model less accepting of differences in the Candidates. This is because it can now jump back to the Seeker model less often than when the transition is more likely, meaning the Seeker model is used less. This can be seen in Fig. 15.

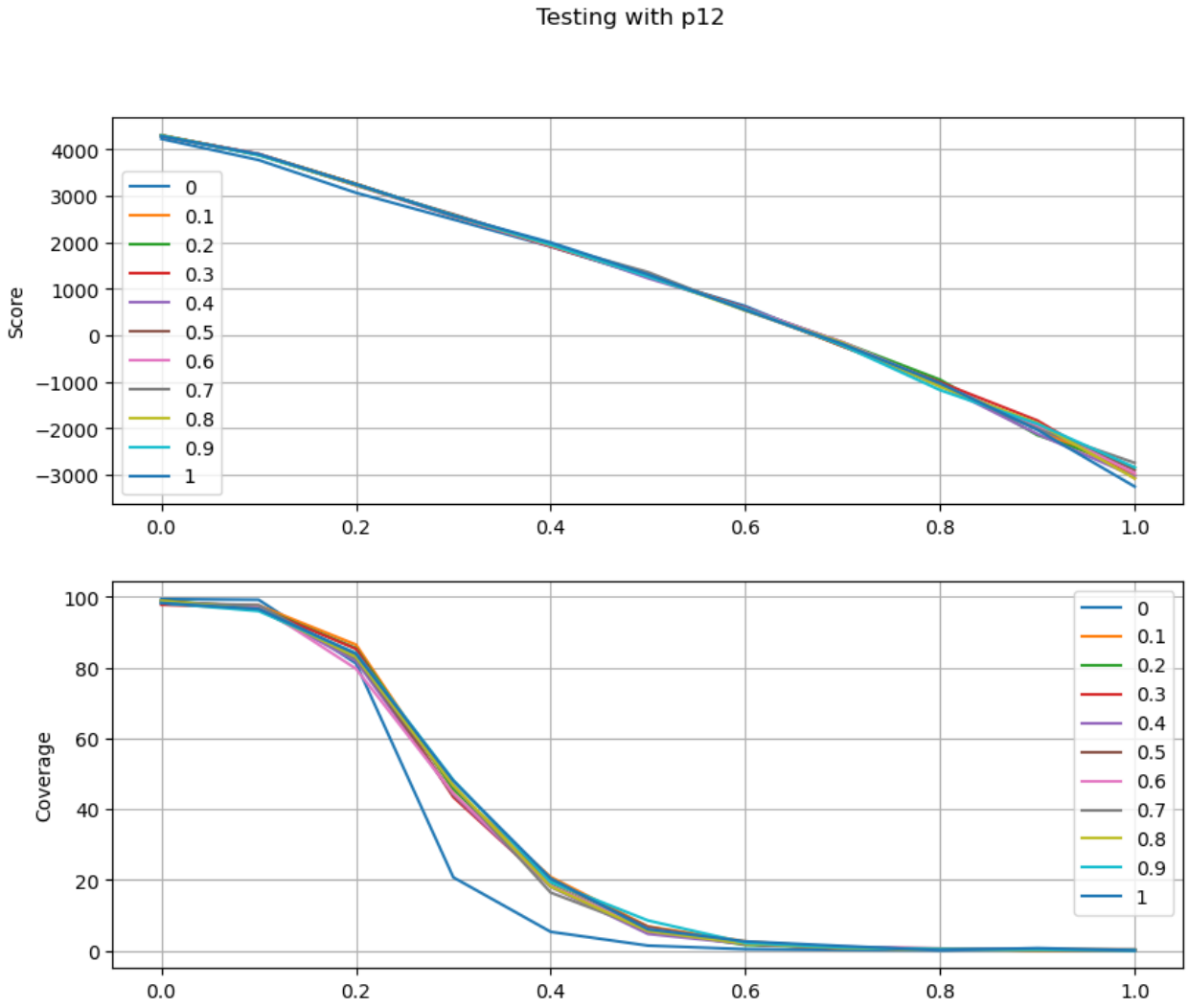


Figure 15: Results with different values of p_{12} in the Merged model.

7 Functionality of the model

This section has as its goal to explore the functionality of the system created, mainly the Merged model, as was described and designed in Sec. 5. In Sec. 6 the generality of the model was explored, as well as some parameters tested. This section will instead use the model, and look at some of the results it provides. The structure of the test, as well as the expected results will be shown in sections 7.1 and 7.2 respectively. The actual results will be shown in 7.3.

For the function testing, a consecutive group of predicted states from the Seeker model will be called a ping, which is defined by their length and the density of the samples that are from the Seeker model. This is done so that a break from the Seeker states doesn't create two groups, but that both are collected into one. If they where not connected, a lot of the groups would be broken into smaller versions. This is because the transition-matrix for the Seeker-model was altered to not stay in it's last state to include the transition to the Garbage

model when they where combined into the Merged model.

7.1 Function-test

The function test has as one of its goals of seeing if the Merged-model is able to recognize a subsection within a Candidate that is a good match for a Target, or testing for true positives. It also has as a goal to test if the model doesn't provide subsections that are not similar to the Target. This is known as testing for false positives. Since no labeled data is available, it is not possible to test for false positives. Due to this, and the fact that this is a function test, the result will be considered positive if the model manages to detect true positives, implicitly assuming that all negatives are true negatives. This is done, as the reverse assumption would mean that the model detects sections that *doesn't* fit the Target rather than sections that do.

The database used in the function-testing has 13 Candidates within it, where there are 12 unique trails, and 11 "real" Candidates. The 2 "fake" Candidates are duplicates of the Target, added as a known input which should be recognized by the model. The remaining entries in the database are recordings of different trails in Norway recorded using various GPS-devices, e.g. sport-watches, but have not been screened for potential matches. Due to this, the pings have to be evaluated for their match. To do this, the metric produced by the model when evaluating the Candidate-samples will be used and compared to the metric produced when evaluating the Target-samples. This also creates an easy method for end-users(trainers and athletes) to understand the relationship and quality of the match between the subsection of the Candidate and the Target.

The test will be performed by extracting Candidate i from the database, and then extracting some features $\vec{\boldsymbol{f}}_i$ from it. These features will be treated as the observations, $\vec{\boldsymbol{O}}_i$, for the Merged-model, $\lambda_M(\boldsymbol{T}, \boldsymbol{E}, \boldsymbol{\pi})$, which will be used to predict which states are most likely to have generated the sample(s), producing a vector of states $\vec{\boldsymbol{Q}}_i$. The predicted state sequences $\vec{\boldsymbol{Q}}_i$ will be stored in another vector $\overrightarrow{\boldsymbol{Qs}} = [\vec{\boldsymbol{Q}}_0, \vec{\boldsymbol{Q}}_1, \ldots, \vec{\boldsymbol{Q}}_i, \ldots, \vec{\boldsymbol{Q}}_I]$. When all the Candidates have been decoded, the state-sequences will be analysed for high-density groups of states from the Seeker-model, known as pings. The length and density of samples from the Seeker-model will be able to be adjusted easily and without changing the model or doing other modifications. For the purpose of this test, the length was set to $l_s = 90$, and $\rho = 50\%$, completely arbitrarily, but a good starting point. The length should be a bit shorter than the amount of samples in the actual Target, which in the text was 181 samples long, while the ρ can be adjusted as the user sees fit. This particular setup groups pings that are closer than 45 indexes together. This method is done to "connect" pings that are close together, as explained earlier in Sec. 7.

7.2 Expected results of function testing

This section will outline the expected results from the function tests, with results shown in Sec. 7.3. Some pings are expected, as the database contains many kilometers of Candidates. That being said, there exists a possibility that the database does not contain any subsection

of any Candiate that will match the Target. Due to the scale of the database, this is assumed to not be the case. It is also expected that the pings are to be somewhat similar to the Target, that is: the pings should be similar enough to make sense that they are chosen as matches for the Target. Since the Target contains some turns and a hill, the pings can't contain only flat and straight sections. The metric is also expected to be functional in measuring the similarity between the different pings and the Target, in accordance with the expectations and requests from OLT. In other words, the metric should be human-readable, easy to make sense of, and be realistic. This metric will be explained later.

7.3 Results from function testing

The function test where performed as described in Sec. 7.1, with density of $\rho = 50\%$ and a sample-length of $l_s = 90$. It might be important to note that since different devices might have different sampling frequency and different users, the difference in physical length between two samples might be different. On average the sampling frequency is about $f_s = 1\text{Hz}$, but this might vary. In addition, for some Candidates the recording contains several samples where the athlete doesn't move, or sections where the movement is very low. This is handled by the features used, as they are based upon the spatial distance, allowing the model to detect matched despite the change in sampling frequency. Combined, all this means that it is desireable to be able to vary the amount of samples designated as a good match for the Target, and not lock it to be of the same length as the Target. This is something HMMs are well suited for, but has to be kept in mind when evaluating results, and setting boundaries, such as the sample-length. Since the output of the HMM-model can be a different length than the Target, any measures of similarity that requires two equal length sequences are not usable. The chosen method was the internal metric produced by the Merged model, $P(\overrightarrow{\boldsymbol{O}_{\boldsymbol{c},\vec{\boldsymbol{i}}}}|\lambda_S)$, when it evaluates the subsequence, divided by the amount of samples in the cumulative score, shown in Eq. 40. Note that only the states stemming from the Seeker-model are considered, indicated by the subindex S.

$$\text{measure}_{c,\vec{\boldsymbol{i}}} = \frac{\sum_i P(\overrightarrow{\boldsymbol{O}_{\boldsymbol{c},\vec{\boldsymbol{i}}}}|\lambda_S)}{N_{samples\ in\ \vec{\boldsymbol{i}}}} \tag{40}$$

In Eq. 40, the subindex c refers to Candidate c, while $\vec{\boldsymbol{i}}$ refers to a subsection that satisfies the boundaries for a good match, or a ping. The equation is effectively the mean probability that a ping has been generated by the Seeker-model. This was done so that longer sample-lengths don't dominate the evaluation simply by being longer than other sequences. In addition, the detected matches are presented with a percentage indicating how similar they are, for ease of use by the end-users as mentioned in earlier sections. This will be called "Goodness" later in the section. The Goodness is the ratio between the score of the Candidate's subsection, and the Target, both evaluated using the Merged model, shown in Eq. 41, where the subindex T refers to the observations from the Target.

$$\text{Goodness}_{c,\vec{\boldsymbol{i}}} = \frac{\text{measure}_{c,\vec{\boldsymbol{i}}}}{\text{measure}_T} = \frac{P(\overrightarrow{\boldsymbol{O}_{\boldsymbol{c},\vec{\boldsymbol{i}}}}|\lambda_S)}{P(\overrightarrow{\boldsymbol{O}_{\boldsymbol{T}}}|\lambda_S)} \tag{41}$$

The Goodness will be presented as a percentage instead of a decimal number for ease of interpretation.

As mentioned, the function test had the goal of looking at the pings that the model detects as being potential good matches for the Target. It is possible to vary the amount of sections presented to the user by varying the sample-length and density requirements, or possibly retraining the model with new parameters if desired. For the function test the database contained 13 Candidates, which have been explained earlier. The test-database contained a total of approximately 226 km of trails, with a mean distance of 17 km each. The longest Candidate was 35 km long, while the shortest was the Target-duplicates, being just 1.6 km. Evaluating all these Candidates took < 3 seconds, which means that the solution is scalable to larger databases as well. The model was able to detect both Target duplicates, and will be disregarded for the rest of this section as their only purpose was to test if the model could detect them.

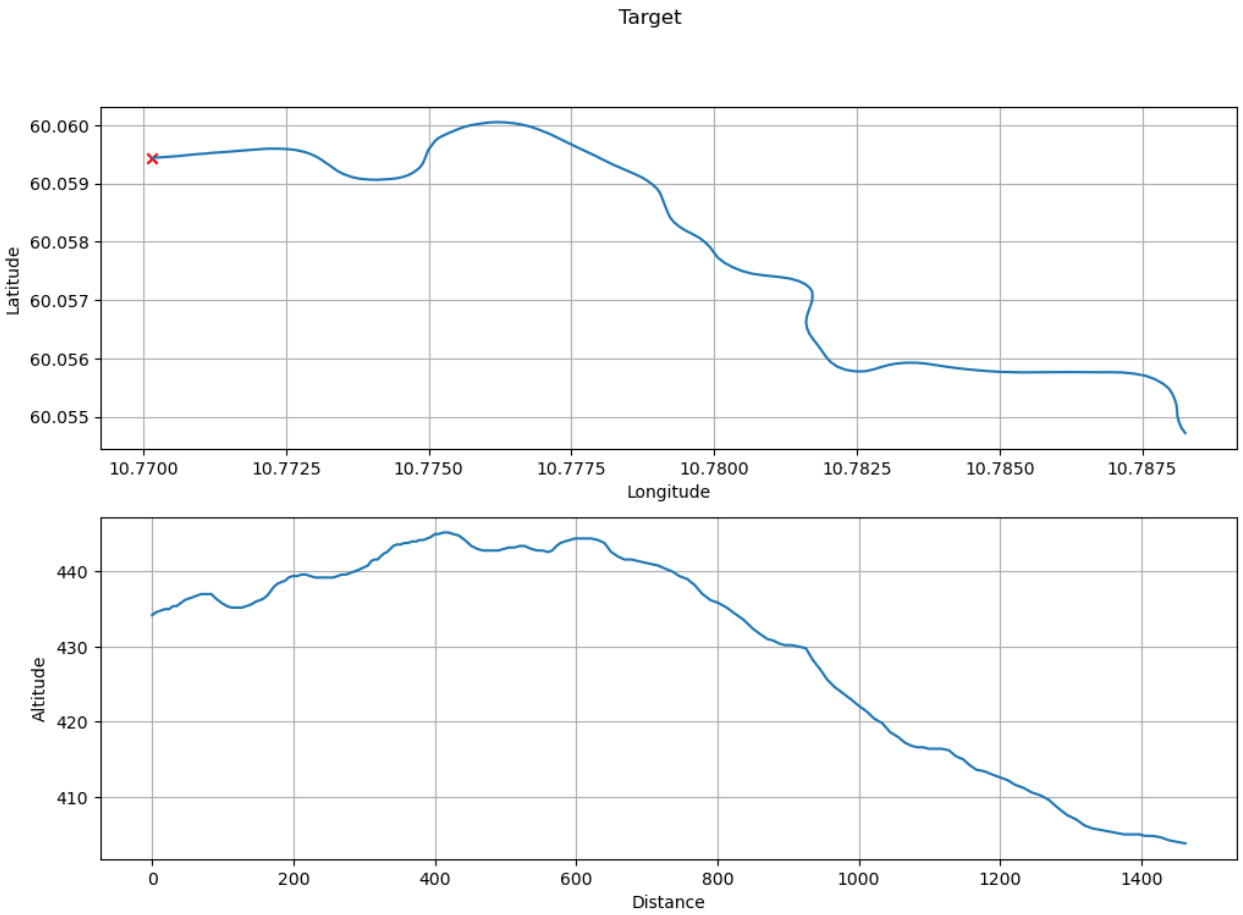


Figure 16: Birdseye view and height-profile of the Target used in the function test.

In Fig. 16 both a birdseye view and the height-profile of the Target can be found. Subsections of Candidates will be presented with the same setup. The upper subplot is the birdseye view, showing the 2-dimensional representation of the path that is defined as the Target for the

purpose of testing. The bottom subplot shows the height profile of the trail. The birdseye view consists of latitude and longitude, while the height-profile consists of the altitude by the distance. In the top subplot, the red cross signifies the start of the recording.

Shown in Fig. 17 is a ping detected by the model. The Target and the ping are of approximately the same length, and both have similar 2D representation and height profiles.

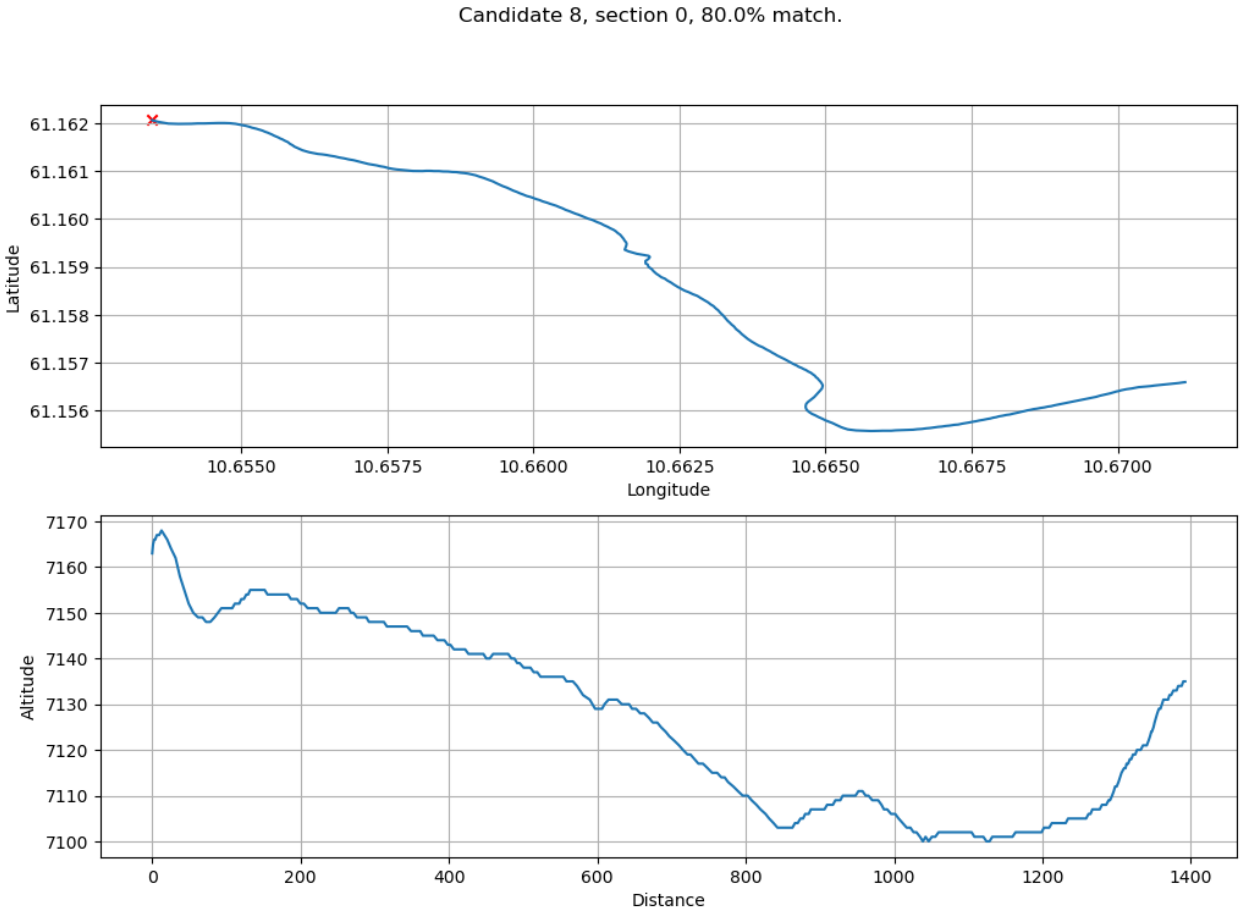


Figure 17: Plot of Candidate 8, first ping.

One major difference between the height profiles is that the Target has a short climb before descending, with some local oscillations. The ping has a steep decline and short rise, followed by a slower decline and then flattening slightly, before ending in a steep incline. In Fig. 18 the ping and the Target is shown in the same plot, making it easier to see the differences. Note that the ping has been slightly moved, but not stretched or altered otherwise, so that similar sections lines up. Again, some differences are visible, but the two sections are fairly similar, and can be considered a true positive. The ping is 353 samples long, or nearly twice as many samples as the Target, despite being approximately the same lengths, the Target being 1462 m long, and the ping being 1392 m long. This positively showcases one of the benefits of using HMMs for this purpose, an ability to detect similar sections irregardless of the sample-length of the recording. This is also made possible due to the chosen features.

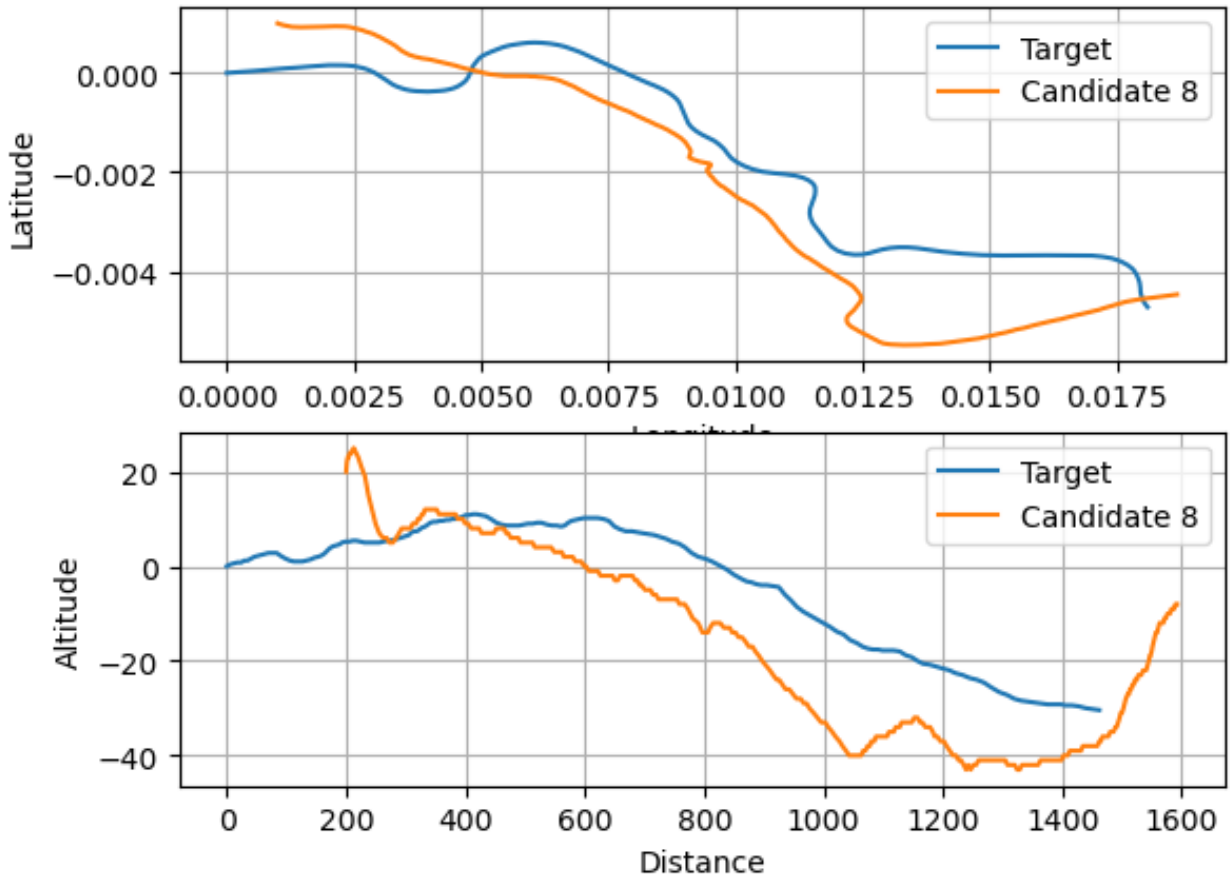
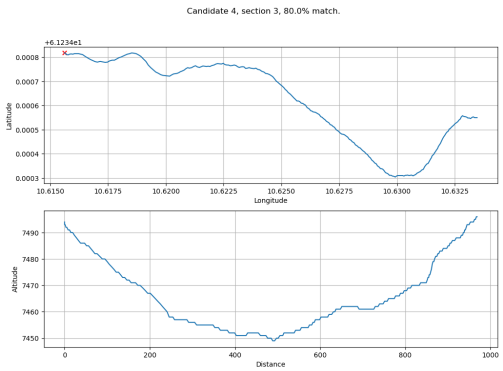


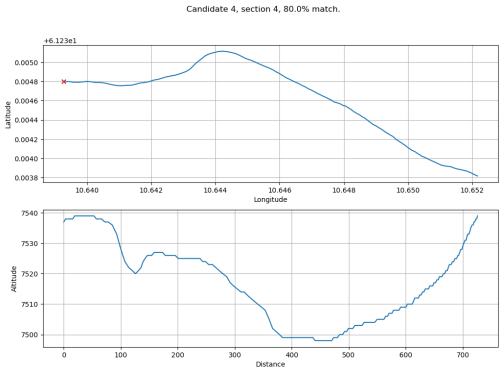
Figure 18: Candidate 8 and the Target shown in overlapping fashion.

If the features had not included the distance, the different sampling frequencies would have become a problem, and this ping would not have been detected. Likewise, it allows for fluxuations between the length, and the other aspects of the observations, ultimately depending on the chosen features.

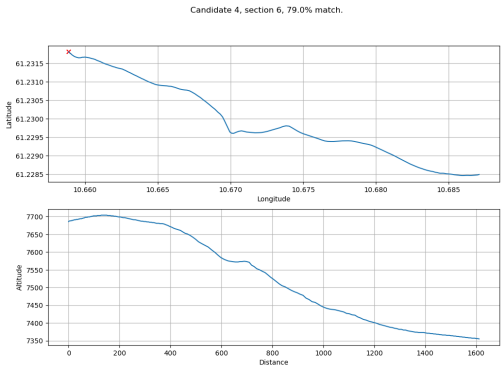
Shown in Fig. 19 is a collection of some of the pings that are detected, and deemed similar to the target.



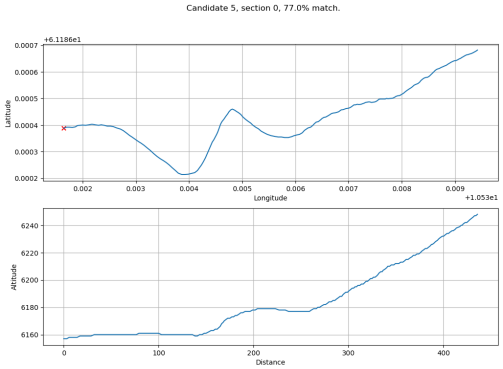
(a) Candidate 4, 4th ping.



(b) Candidate 4, 5th ping.



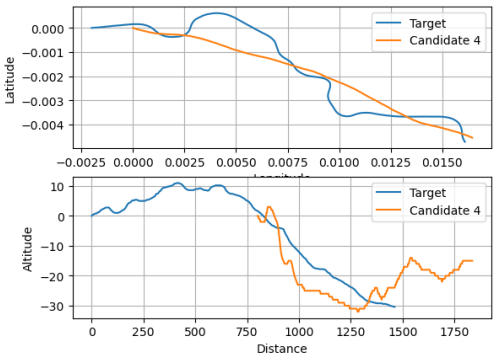
(c) Candidate 4, 7th ping.



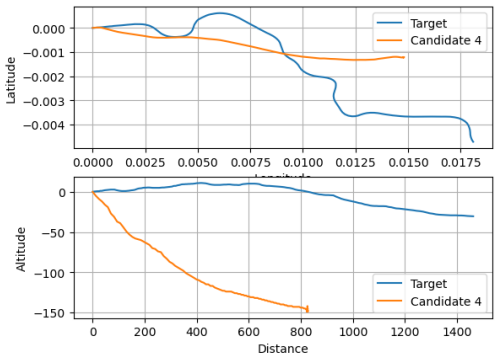
(d) Candidate 5, 1st ping.

Figure 19: A collection of pings designated as similar to the Target.

In total, the model detected 18 pings in various Candidates that where similar to the Target, totalling about 16.5 km. These figures are calculated excluding the Target-duplicates. The best match had a Goodness of 85%, and can be seen in Fig. 20a, while the worst can be seen in Fig. 20b, with a Goodness of 72%.



(a) Candidate 4, first ping, Goodness of 85%.



(b) Candidate 4, 8th ping, Goodness of 72%.

Figure 20: The best and worst pings.

Note that the figures have been moved, but not stretched, for the same reason as earlier.

The altitudes of the two trails have also been moved so that they are comparable as the raw altitude-values between the ping and the Target are so far apart any local variations within them becomes meaningless. It is interesting to note that the best Candidate finds a good match for a section of the height profile, and resembles the geography of the trail as well, while the worst Candidate doesn't seem to have matched the height profile as well. If the worst candidate is considered too bad, it can be filtered out using a filter with a limit on the Goodness. The worst ping is a good showcase of why the relation between differences in Goodness and the actual difference in the experienced similarity can be hard to quantify, as there is only a 13% difference in the Goodness between the best and worst pings. Despite this, the number is human-readable, and end-users has to keep in mind that the Goodness is biased towards the top range of the Goodness. This can be adjusted later during implementation, but has to be done in accordance with a group of athletes, as this again relates to the subjectiveness of what can be considered a "good" ping.

Another interesting note is that Candidate 4 has a lot of pings, at 10, which is more than any of the other Candidates. In this database and for this Target, Candidate 4 also has the best Candidate, but if another database or a different Target is considered, where the best ping is in one of the Candidates with fewer pings an interesting scenario arises. If the best ping is in another Candidate, the best Candidate might be considered the one with the most pings, as this Candidate would allow for more training towards the Target. On the other hand, the best Candidate could be considered to be the one containing the ping with the highest Goodness-score. This is something that has to be kept in mind when setting the ρ and sample-length for considering what is a ping and what is not. By increasing the value of ρ and sample length, some pings are grouped together, which might be useful in relation to training. The problem of what is considered the best *Candidate* stems from the subjective nature of training discussed in Sec. 2.1. The highest Goodness score is what is designated as the best ping within the project, as it has the highest probability of being generated from the Seeker-model, as was discussed in Sec. 2.

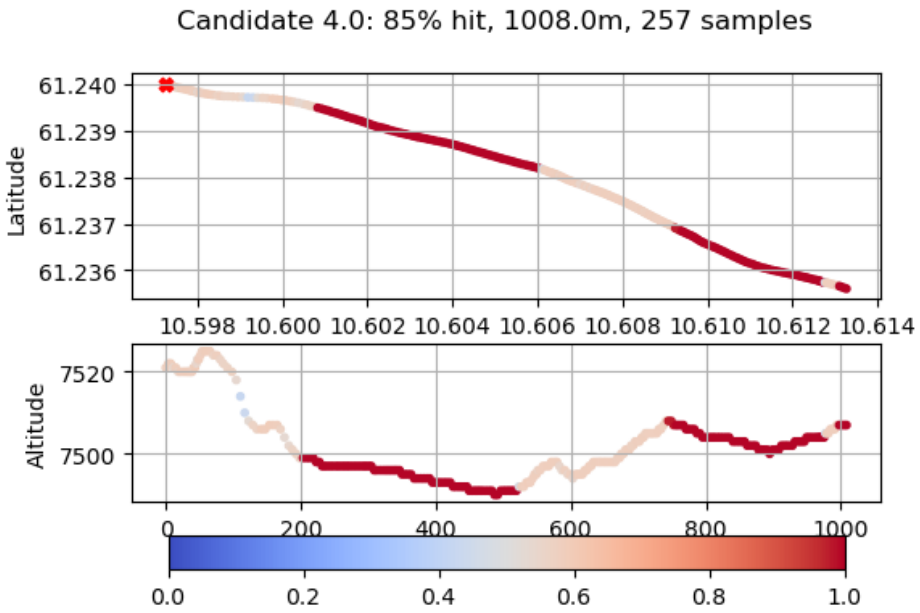


Figure 21: Confidence of the best ping.

Shown in Fig. 21 is the confidence of the different samples of the best ping. Red samples are areas where the model is very certain that the sample could have been generated by the Seeker model, while deep blue is uncertain. Note that there are no deep blue points in the sample. The colourmap is shown in the bottom of the plot. The colour of a given sample is defined by that sample's probability of belonging to a state related to the Seeker-model. The probability is found by applying the Viterbi algorithm to the model's statespace and the observed features, looking at the cumulative metric used. After this, the probabilities related to the Seeker-states are extracted and used to define the colour. By doing this, visualising the probability that a sample is from the Seeker-model, and thus also a confidence-measure for the different sections of the ping becomes easy. This is almost the same as the variables in Eq. 40, $P(\boldsymbol{O}_{\boldsymbol{c},\vec{\boldsymbol{i}}}|\lambda_S)$. Eq. 40 provides the probability that the entire sequence is from the Seeker-model. If the sample-to-sample verison is considered instead of the entirety of the sequence, the resulting measure is of the sample-to-sample probability that is used as a confidence measure for the different sections of the ping. The mean of the confidence measure is the same as the Goodness found earlier, since they are based upon the same probability.

8 Discussion

This section will contain a discussion of the work performed to achieve the goals of the paper, as well as an analysis of the results of the tests. The stated goal for the project was to create something that could achieve the following:

Find the best match for a provided trail within a database with the goal of optimal training for high performance athletes.

The tool created to achieve this was a combination of two kinds of HMM-models, one being a time-series model, and the other being a standard HMM. After the combination, the two

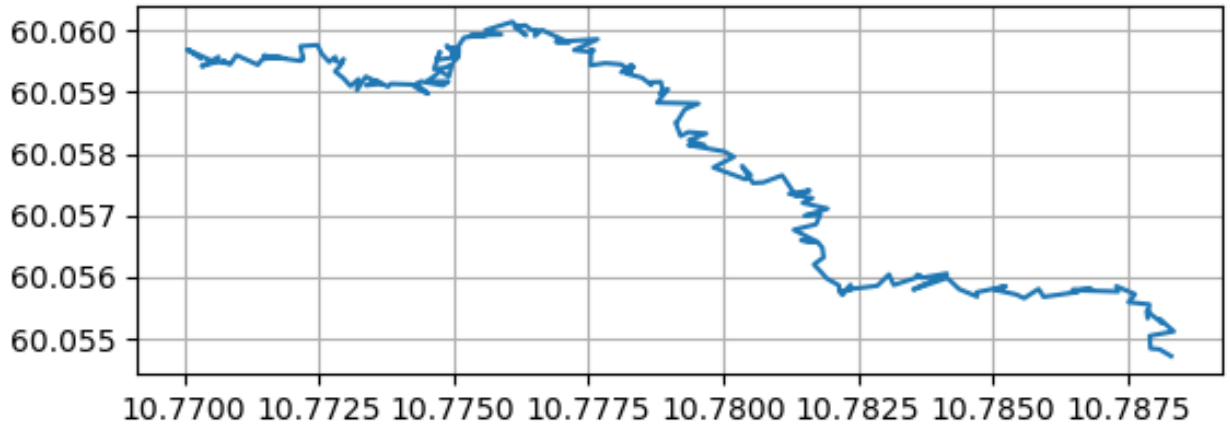


Figure 22: Target with Var-level 0.25.

models function as one, and by monitoring which states are most likely to have generated a sequence of samples, the process of which is shown in Eq. 14, the model can detect when a section of a Candidate is similar to the Target. The results of the implementation of this can be found in Sec. 7. This structure was inspired by the ML-application of key-word spotting, where a model is trained to recognize a phrase within a longer speech signal. If reformatted, this task becomes "detect a pre-determined sequence in a longer sequence", which is fairly similar to the problem in this paper. In addition, there is some similarity to the area of ML known as Speaker verification, where a model is trained to recognize a set of speakers. In both cases, the general structure is one model trained to recognize a sequence, and one trained to represent everything else, similar to the one designed in this paper. The models are compared, and the one that fits better is used, indicating what kind of sequence is observed. One major difference is the background theory and data available to the models. There are unnumbered recordings of a vowel being spoken, but only one useful recording of the Target. Despite this, the structure has proven useful for the problem. The method of considering the trails as samples and states was "discovered" in the pre-project, and has been the basis of the implementation of the HMMs in the paper.

8.1 Interpretation of results

This section will contain an interpretation of the results from Sec. 6.3 and then from Sec. 7.3. The data has been discussed analytically in the mentioned section, but here a more "human" analysis will be performed. As was shown in Fig. 10 (and the two following), the model fails to recognize the Target 50% of the time at around Var-level 0.25. For reference, this level of noise is shown in Fig. 22. Although it might sound advantageous for the model to be able to recognize the Target regardless of the noise level, this is not the case. This was previously mentioned in earlier sections. If the model could recognize the Target at all noise levels it would designate all Candidates as potential matches for the Target, as it would deem any level of difference to be equally likely to be generated by the Target model, or be equally similar. Another interesting observation is the tests with the p_{21}-value, and its edge-values, $p_{21} = 0$ and $p_{21} = 1$. In either case, the result is one of the two undesired behaviours, either

designating all levels of noise as equally good, or failing to recognize the Target at all.

Perhaps more interesting is the detected pings, and the goodness of the different pings. Although none of the pings detected where perfect matches, this is highly unlikely to be detected for longer sequences. This is because of the nature of trails. There is nothing shaping the creation of the trails, other than the geography of an area, and the person recording the Candidate. Because of this, longer sequences that are identical become increasingly unlikely to exist. With this in mind, the function-test can be considered a success. The pings detected are similar enough to be accepted as matches for the Target. Consider Candidate 8 and the Target, shown in Fig. 18. The downhill-section of Candidate 8 and the Target are fairly similar, and the Candidate contains some turns in the same section. There is also one very sharp turn towards the end of the ping, similar to the Target. These areas make the Candiate potentially valuable in an athletes preperation for a competition on the Target. It is also useful to keep in mind that these pings where detected when the model was provided with a database of 11 real Candidates. In addition, it also spent neglible time on the task, indicating that the model can easily handle larger databases. With an increase in the size of the database, there is also an increase in the probability of there existing a good match for a given Target. Had the model spent e.g. a couple of minutes evaluating each Candidate, the scaleability of the solution had been reduced massively, and thus also the effectiveness and usefulness. This was unfortunately not possible to test during this work, as the process requires extensive data collection, as well as labeling, and would provide little use for the amount of effort. Despite this, the fact that the model detected pings, and spent short time on the task means that it can safely be claimed that given a large enough database, included in which is the *best* available ping, the model will find good matches for a Target. Alternatively, this can be interpreted as given a database with a good match, the model will detect the match. One proof of this is the fact that the model detected both Target-duplicates in the database used. The detected pings, especially the pings from the Target-duplicates, show that the Merged model detects sections that have high probability of being from the Seeker model, included in which is of course the ping with the *maximum* probability of being from the Seeker model. This was the problem as defined in Sec. 2.

8.2 Strengths and weaknesses of the Method

This section will contain a short discussion and outline of the strengths and weaknesses of this implementation to the project. One could solve the problem defined in this paper in a multitude of ways. Even within Machine Learning there are many methods of doing so. One could create a function that takes in a trail and creates overlapping, fixed-length windows of a trail, and using e.g. a Convolutional Neural Network to characterize the different windows and their similarities to the Target. It would be possible to use an autoencoder to map the variable-length Candidates, and then compare them when they have been mapped to a fixed length-representation. One could also use the window-function from earlier to simply find the correlation between a window and a Target. All these suggestions have their own strengths and weaknesses. Compared to the one suggested in this paper, the main weakness is either computational costliness, or a need for large amounts of data(which are not readily available). The strengths of these methods are an increased control over the inner workings

of the method, or a simplicity that allows end-users to more easily understand the method, and thus also possibly use it more effectively, in addition to a decreased probability of human error due to an understanding of a less complex model, or a more black-box approach that doesn't require hands-on during the setup of each individual Seeker.

A problem with regards to the use of HMMs is the fact that when the Target switches, the Seeker has to be retrained. This means that one has to set the number of state for the Seeker model again, as this directly influences the number of parameters the model has to learn. If the number is to large, the model quickly becomes overtrained, and making it less useful. Likewise, if the number is too small, the Seeker can struggle with properly capturing the Target, although this is not a large risk, as was shown in the tests showcased in Sec. 6.3, and in Fig. 13. In addition, this introduces the possibility of human error, the exclusion of which is one of the benefits of using this tool.

Another problem that relates to not only HMMs but *any* tool created is the problem of what is considered a good match for a Target. Previously discussed has been the problem of comparing variable length sequences in a good way. In the pre-project euclidean distance between the sequences was used, which meant that the sequences had to be of equal length. Likewise, if one where to use e.g. correlation the sequences also has to be of equal length. In this work, this was solved by using probabilities that the different sequences belong to a model trained on the Target, as this does not pose any requirements to the lengths. Regardless of what option is used, the problem remains: does the chosen method accurately capture the quality of the sequence? As has been previously mentioned this is subjective, and will change from user to user. A good example of this problem can be seen in the different pings shown in Sec. 7. The ping that was defined to be the best sequence by the Merged model was shown in Fig. 20a. Another ping was shown in Fig. 17. Arguably, the second ping is better for training as it *looks* to line up better to the Target. This should lead to it being better than what was designated as the best ping, but the Merged model predicted that the best ping was better. The model detects sequences in other sequences, and the procedure produces a metric, which is the probability that is used here. When doing a simple eye-test, as was done in the comparison of the pings, the result is both unreliable and subjective. This makes it difficult to evaluate *any* tool for their quality, due to the lack of an objective measure, as well as the subjective nature of the "truth" in the problem. This is one of the reasons why all the detected pings are shown, as the best ping, regardless of if its the eye-test or the model's designation, is included in the detected pings. Despite this, the model designed in this paper succesfully detects what can be argued to be the most similar section of a Candiate, through the highest probability, and the detected pings are similar enough to the Target to be accepted. This weakness is simply one of the application of any model as a tool, as this weakness stems from the problem, and not the chosen model.

One of the main advantages of using HMMs is their ability to stretch and shrink in order to better fit a Candidate, showcased by the differing lengths of the pings from Sec. 7. Since a HMM is able to either go to another state, or stay in the current state, a Candidate can be recorded with some disregard of sampling frequency, as long as the trail it represents is properly represented within the data. If there is too large a difference some problems arise,

but the sample-to-sample distance between observations are not required to be equally long, and failure to have them equally long does not lead to errors in the implementation. This issue has to be handled manually if any of the other suggestions are used, but the use of HMMs allows for both an increased pool of Candidates, as well as the shrinking and stretching behaviour observed. Both of these are considered a strength of the method and system.

Another advantage of using HMMs is that they are unsupervised models. Unlike other models, like CNNs or RNNs, HMMs does not require labeled data for their training, simplifying the process of the implementation.

8.3 Reflections upon the process

This section will contain some reflections upon the process of the work performed, and what worked. The majority of the work has been performed using the following steps: Define sub-problem, link to area of theory, find and read theory, plan application, and apply. During the "find and read theory"-stage, the theory was written down as well. The process of writing had a dual purpose. It made the writing of this paper a lot easier, as the majority of the theory has already been written. The second purpose was ensuring a proper understanding of the theory beforehand, rather than a more convenient trial-and-error method. Although obvious when stated, actually following this process made the work easier and more enjoyable, as the implementation of the theory and the writing of the code was performed with less hiccups and errors. It also facilitated more effective and concise support and guidance from the supervisor of the project, as it is easier to communicate plans and their problems when expressed through the common framework of either math or other theory, especially in relation to ML models and applications.

The first step, defining sub-problems, was crucial in making the work effective. It minimised time spent reading and finding theory that wasnt applicable to the problem due to some fundamental difference between the application of the theory and the problem at hand. It also made it easier to find the theory itself, as it makes it easier to find similarities between problems. Take for example the stated problem of the paper, as defined in Sec. 2. It can be understood as "Detect a subsequence that looks like a designated sequence within another sequence". This definition makes it easy to understand the similarities between the problem and e.g. keyword spotting, which this solution is inspired by, as quickly mentioned in previous subsections. This is perhaps the clearest example of how well the method can work for solving problems and finding solutions quickly and effectively.

The work was also heavily supported by the supervisors, through weekly meetings. When the project moved outside the area of expertise of the main supervisor, Kimmo Kansanen, another co-supervisor was contacted, which also helped alot. The described steps where of great help during these meetings as well, as it allowed the supervisors to quickly assess any holes in the theory and mistakes that could have been made before the implementation of the theory.

8.4 Contributions

As was mentioned in the previous subsection, the work was helped along by the supervisors. The internal supervisor was Kimmo Kansanen, and the supervisor from OLT was Fredrik Mentzoni. In addition, Giampiero Salvi was contacted and participated in meetings as well.

Kimmo Kansanen has played a crucial role in shaping the direction of research, as well as providing valuable insight throughout the process. His experience and extensive knowledge within the field of signal processing has influenced the methodology and structure of the work greatly. His mentorship and critical input have been invaluable in refining the research objectives, interpreting the findings, and ensuring the alignment with the research objectives. The intellectual discussions and constructive feedback received from Kimmo Kansanen have been invaluable in expanding my understanding of the subject matter and enhancing my research skills. I am truly grateful for his dedicated supervision and mentorship throughout this journey.

Fredrik Mentzoni has played an equally valuable role in the work performed as well. He has provided extremely useful insight and has helped structure the direction of the work through discussions and feedback upon how the problem should be solved, as well as how it was solved. A lot of the assumptions and limits that make the implementation possible has been made through discussions and help from Fredrik, and the work would not have been possible without it. I am grateful for the opportunity to work with Fredrik and the help that he has provided during the course of the work.

Giampiero Salvi was included in the weekly meetings once the work moved into the field of Machine Learning. Through his expertise within Machine Learning and Speech technology he helped identify theory and applications that ultimately led to the implementation shown in this paper. While the solution suggested here most likely would have been discovered with enough time, it might not have been achieved within the timescale of the project. The help provided has also drastically increased the quality of the work performed, and I am grateful for the help.

9 Conclusion

This section will contain a conclusion to the paper. The conclusion will consist of a summarisation of the method and results of the project, as well as how they scale up to the problem defined in the begining of the paper. In addition, there will be a suggestion of further work that might be done to improve the system designed and outlined in this paper.

9.1 Summary

The solution to the problem defined in Sec. 2 was a HMM model that consists of a time-series HMM and a more general HMM network, combined into one. When the model is provided with a sequence of features, it can use the features to determine a sequence of hidden states

that is most likely to have generated the sequence of features. These hidden states are produced by one of the two previously mentioned models. The time-series model is trained on a Target, being able to recognize the Target, and the sequence of samples that represent it, while the more conventional network is trained on more, and other, trails so that it represents other sequences better. The result of this is that by looking at what model is most likely to have produced the samples, captured in the hidden states, it is possible to detect when a subsection of a Candidate is similar to the Target. Results of tests can be found in sections 6.3 and 7.3.

In addition to the model, a small database with the purpose of testing was also made. This database can be expanded to become a functional database for the real application of the model, but can also be completely replaced by a new database. As long as all new Candidates uphold the assumptions of being in a lat/lon/alt format, the model is compatible as it is now.

The features used in the current implementation capture the physical geography of the different trails, although there are other options available as well. If desired it would be possible to use different features to capture other aspects of trails, if so desired. This involves background knowlege specific to the sport it is implemented for.

Some experimentation was also done with regards to some hyper-parameters of the combination of the models. This included tests with differing number of states in the time-series model, and the parameters that control the switch between the different models within the combined model.

In conclusion, the Merged model designed in the paper finds sections of Candidates that are highly likely to have been generated by the Seeker model. Included in this is of course the sectino with maximum probability, the detectino of which is the defined problem of the paper. For reasons discussed earlier, the system presents several pings to the end user, as the highest probability of being generated by the model doesn't neccesarily correlate to the best training opportunity. Facilitating better training is the stated goal of the project, which is the reason for the list of pings rather than presenting the single best ping.

9.2 Significance of work performed

As was mentioned in Sec. 1.3, the project is interesting for three groups: athletes, trainers, and OLT. For OLT the system implemented in this report/work helps them achieve their goal of ensuring Norways position in international sports competition, as is one of their stated goals.

For athletes, the system will help them better prepare for future competitions. By finding similar trails to the one they are competing on, the athletes will be able to perform better during the actual competition, by having experienced trails with similar geographies during training. In addition to the physical benefits of being familiar with a trail, the athletes can also benefit from the knowledge of having performed well on a similar trail, as well as knowledge of how their body will respond to the exertion of a specific Target.

The main benefit for trainers is a effectivisation of the work performed. Simply by training a new Seeker, the trainer can be given a list of trails that will provide their athletes beneficial training. Further this allows them to either focus more on other aspects of the training that can further improve athletes, or by training more athletes at the same time. Both of these possibilities further the same goal: improving Norway's athletes at the top level, either through quantity or quality.

9.3 Further work

The work performed and recorded during this paper has achieved the defined goal, through a system of HMMs and a database, structured as described in Sec. 3. The Merged model is able to detect differences in a trail up until a certain point. It was also shown to be able to detect sections of real Candidates. This leaves two options for further work, either improve upon the current implementation of the system, or try another system to achieve the same goal.

If a new implementation is chosen, one might consider one of the methods shortly mentioned earlier in the paper. Using background theory, and more closely vetting the trails and their information, one could create a system that could use a more traditional measure of similarity to detect subsections that are similar to a Target. This vetting-process could mean enforcing a sampling-frequency, or interpolating samples in order to ensure the sample-to-sample distance stays the same between all Candidates and Target(s). Another option might be to implement either a CNN or RNN, although both of these hold their own share of problems, limitations, and strengths. Using a CNN will require extensive pre-work, as it is sensitive to the structure and dimensions of the data, similar to what was described above. A RNN might be easier to implement, as it requires a bit less pre-work since it can be applied to time-series data of unequal length. Any implementation using any form of Neural Network will require a lot of labeled data, as the models need to be trained.

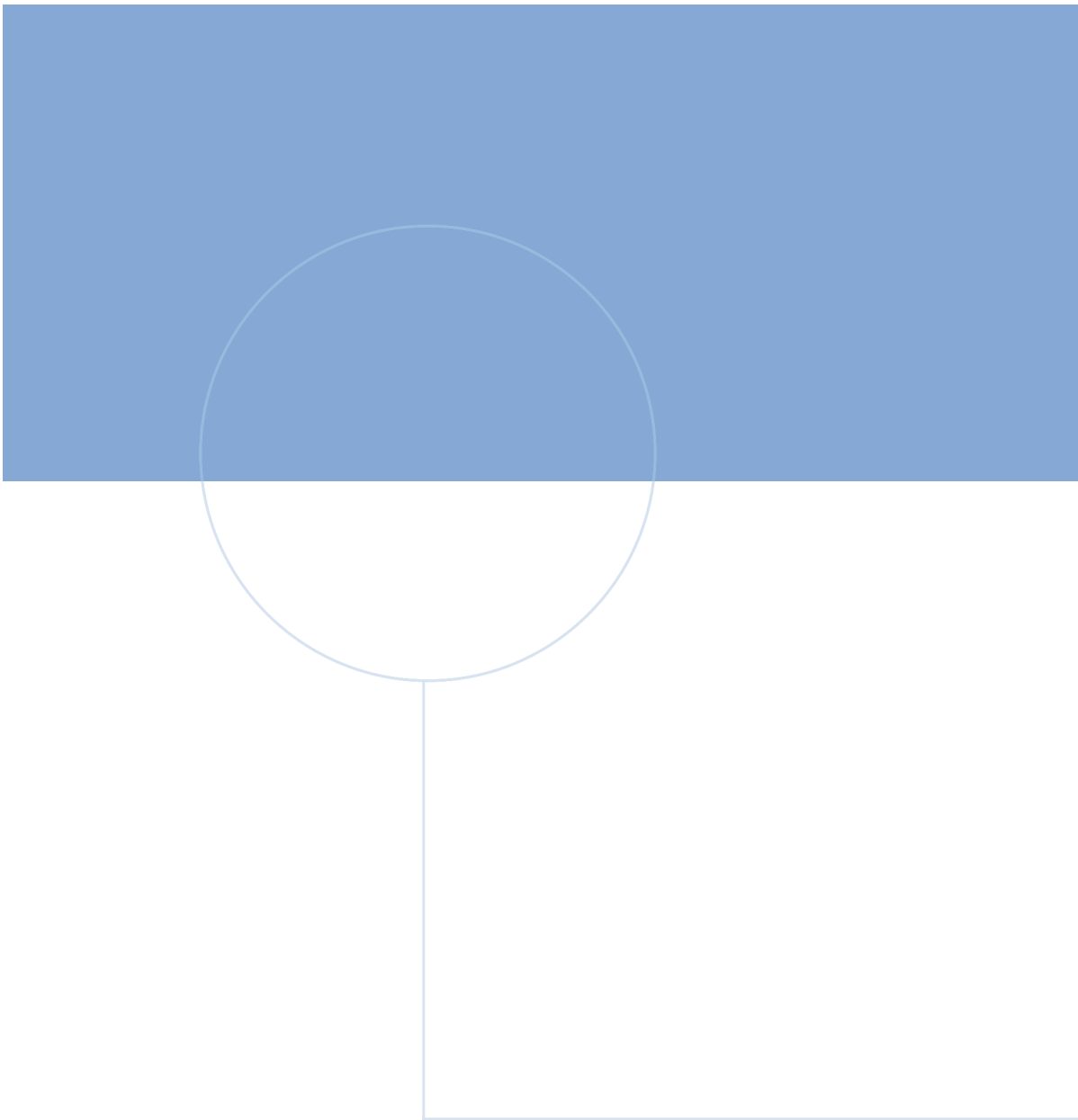
To improve upon the current implementation, one could explore different features to be extracted. The current features represent the trail's geography, as they only consider positional information. If information about an athletes physical state was introduced to the trails, one could find a Candidate that could elicit the same physical response from an athlete. An example of a situation where this is useful could be if an athlete wishes to prepare for the physical aspect of a trail, rather than the technical. By focusing more on the response, the athlete will be able to prepare for the same exertion that they will experience during the competition. This is in comparison to the more "technical" approach of the current features, capturing the turns as well as the climbs/falls of the trail. The same can also be achieved by altering the current features, e.g removing the lat/lon- from the features, leaving only the altitude. Further experimentation is limited only by imagination.

Bibliography

Forney Jr, G David (2005). "The viterbi algorithm: A personal history". In: *arXiv preprint cs/0504020*.

Olympiatoppen (n.d). *Målsetning toppidrett*. `https://olympiatoppen.no/om-olympiatoppen/malsetning-toppidrett/`. Accessed 2 May 2023.

Sklar, Bernard (2001). *Digital communications*. 2nd ed. Prentice hall Upper Saddle River, NJ, USA:



NTNU

Norwegian University of
Science and Technology