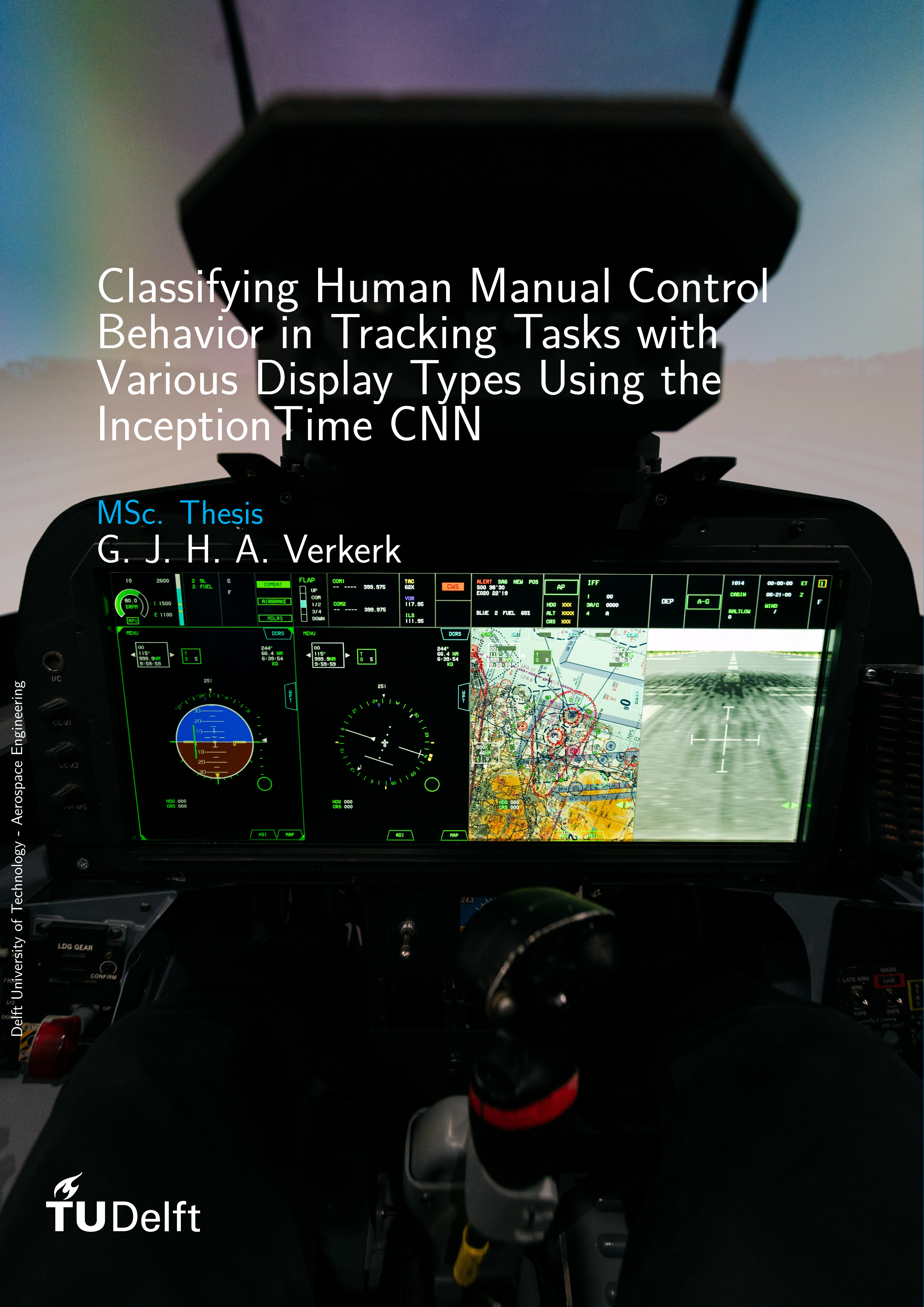


Classifying Human Manual Control Behavior in Tracking Tasks with Various Display Types Using the InceptionTime CNN

MSc. Thesis

G. J. H. A. Verkerk

Delft University of Technology - Aerospace Engineering

Classifying Human Manual Control Behavior in Tracking Tasks with Various Display Types Using the InceptionTime CNN

MSc. Thesis

by

G. J. H. A. Verkerk

to obtain the degree of Master of Science in Aerospace Engineering
at the Delft University of Technology,

Student number:	4303172		
Project duration:	November 16, 2020 – September 29, 2021		
Graduation Committee:	Prof. dr. ir. M. Mulder,	TU Delft	Supervisor
	Dr. D. M. Pool	TU Delft	Supervisor
	Dr. B. Shyrokau	TU Delft	External Member
	Dr. K. van der El	TU Delft	Additional Member

Preface

This document contains the graduation report with which I conclude my Master of Science in Aerospace Engineering. It is made up out of a main scientific paper and its corresponding appendices, as well as a preliminary report that contains a literature research and some initial simulations.

I would like to sincerely thank my supervisors Daan and Max for their help, critical views and above all, their positivity. Before I started, many people told me that having the right guidance can go a long way in having a positive graduation experience and I will certainly echo this notion. I would also like to thank my roommates, Hidde and Paul, for providing a calm yet enjoyable working environment during the somewhat turbulent times in which I wrote this thesis. Additionally, I want to thank Paul in particular for always making the time to help me with some of the programming related problems that I encountered.

Naturally, I would not have been able to even start this thesis without the support of many people during both the bachelor and master phases of my studies. First and foremost, thanks go out to my parents and my sister for always showing faith in me, yet also motivating me whenever it was necessary. I am incredibly lucky to have grown up with such a loving family and I appreciate it every day. Furthermore, I would like to thank all of my friends, both in Delft and in Amsterdam, that have made my study time such an enjoyable experience. I will never forget you nor the adventures that we experienced together.

My time studying in Delft has been a wonderful period of my life and I will always look back at it fondly. Now, however, it is time for me to move on!

G. J. H. A. Verkerk
Rotterdam, September 2021

Contents

List of Acronyms

2CC	Two-Class Classifier
3CC	Three-Class Classifier
AE	AutoEncoder
AI	Artificial Intelligence
ANN	Artificial Neural Network
ARX	AutoRegressive eXternal input
AUC	Area Under the Curve
C	Compensatory
CAM	Class Activation Mapping
CE	Controlled Element
CM	Confusion Matrix
CNN	Convolutional Neural Network
DAE	Denoising AutoEncoder
DC	Discriminative Classifier
DI	Double Integrator
DNN	Deep Neural Network
DT	Decision Tree
EoE	End-to-End
FF	Forcing Function
FFN	FeedForward Neural network
FN	False Negative
FP	False Positive
G	Gain
GAN	Generative Adverserial Network
GC	Generative Classifier
GPU	Graphical Processing Unit
GRU	Gated Recurrent Unit
HMM	Hidden Markov Model
HO	Human Operator
HOM	Human Operator Model
IR	Internal Representation
KF	Kalman Filter
LF	Loss Function
LSTM	Long Short-Term Memory
MAD	Mean Absolute Deviation
MCC	Manual Control Cybernetics
ML	Machine Learning
MLE	Maximum Likelihood Estimator
MSE	Mean Squared Error
NM	Neuromuscular
NN	Neural Network
OL	OverLap
PR	Preview
PS	Pursuit
RBF	Radia Basis Function
ResNet	Residual Network
RNN	Recurrent Neural Network
ROC	Receiver Operating Characteristic
SF	Sampling Frequency

SI	Single Integrator
SOP	Succesive Organisation of Perception
SVM	Support Vector Machine
TN	True Negative
TP	True Positive
TSC	Time Series Classification
TVCB	Time-Varying Control Behavior
WS	Window Size

List of Figures

List of Tables

I

Scientific Paper

Classifying Human Manual Control Behavior in Tracking Tasks with Various Display Types Using the InceptionTime CNN

G. J. H. A. Verkerk, *Author*, D. M. Pool, *Supervisor*, and M. Mulder, *Supervisor*

***Abstract*—Despite the increasing amount of automation in manual control tasks, such as driving a car or piloting an aircraft, the human ability to adapt to unexpected events still makes us an essential part of the control loop. Before we can remove the human completely, a better understanding of this unique characteristic is necessary so that we can apply it to automation itself. Currently, however, research opportunities are limited mainly due to a lack of techniques that can recognize such control adjustments within short periods of time. One well known form of adaptation is characterized by changes in control behavior in response to three types of displays used in manual control tasks: compensatory, pursuit and preview. This research aims to use time series classification to recognize such changes using samples of 1.5 s of tracking data from experiments with double integrator controlled element dynamics. InceptionTime, a convolutional neural network, is optimized for the task, resulting in an average attained accuracy of 95.4%. It was found that the best configuration is a basic three-class classifier that uses the e, u, x signals and their first-order derivatives as input features. Additionally, results show performance highly depends on what data sets are used for training and testing, with accuracies ranging from 60% to 99% for different train-test allocations. Most errors occur in separating the pursuit and preview strategies, where it was expected to be between compensatory and pursuit. In general, the results exceed expectations and could signify a breakthrough in recognizing and understanding human adaptation in future experiments.**

***Index Terms*—Manual control cybernetics, display, time series classification, deep learning, convolutional neural network, inceptiontime**

I. Introduction

OVER the last decades, automation has become an integral part of manual control tracking tasks such as driving a car or piloting an aircraft. Due to rapid advances in machine intelligence (e.g., highly automated cars), the responsibility of the Human Operator (HO) in such tracking tasks is becoming increasingly supervisory and less direct during regular, time-invariant operating conditions. However, what has so far been unique to humans is their ability to adapt control behavior in response to (sudden) changes in the control task [1]. As a result, we are still a critical part of the control system and essential in maintaining control in case of unexpected events. If we can learn how humans perform such adaptive, time-varying behavior, however, we can both improve human-machine interfaces to accommodate it better and apply it to automation itself in order to achieve safe, completely autonomous systems. Unfortunately, traditional cybernetics techniques lack the capability of recognizing human adaptive control behavior in real-time, making it impossible to execute the necessary experiments [1]. Consequently, new methods should be explored that can bridge this gap and revolutionize research into human adaptive control.

One widely researched form of human adaptation is characterized by changes in control behavior in response to three types of displays [2]: compensatory (C), pursuit (PS) and preview (PR). Different information on the state of the tracking task is shown on each display, such that an HO will adapt their strategy to the information available. Corresponding human operator models (HOMs) have been developed to mathematically represent these control strategies, first for the compensatory tracking task [3] and later also for pursuit and preview [4]. These models, however, rely on the limiting assumption that human controllers can be modeled as quasi-linear, time-invariant feedback controllers and require large amounts of tracking data. Attempts have been made, using several different system identification techniques, to make the models time-varying, but with limited success [5]–[9]. Additionally, what none of these methods have been able to incorporate, is that HOs have also been observed to switch between the different control strategies [2]. Theoretically, multiple HOMs should therefore be used at various times in the tracking run to accommodate these adaptations, but no methods currently exist that can recognize them in real-time. New techniques should therefore be explored that can solve this issue.

One research domain that has shown great promise over the last decade in recognizing patterns from (time series) data that were previously hidden, is Machine Learning (ML) and more specifically, Time Series Classification (TSC). TSC refers to a sub-class of ML models that purposefully factor in that the input data are chronologically structured, such that both the raw input and the order that it is in, influence the outcome [10]. The significant advantage of such algorithms with respect to more traditional methods, is that they can pick up on highly non-linear relations that can then be leveraged to classify similar groups of data. Recurrent (RNN) and Convolutional (CNN) Neural Networks in particular have shown promising advances when it comes to classification of time series [11], [12].

Recently, Versteeg et al. [13] presented an RNN that successfully recognizes – with 96% accuracy – the active

All authors are with the section Control and Simulation, Faculty of Aerospace Engineering, Delft University of Technology, 2629 HS Delft, The Netherlands.

controlled element (CE) dynamics in a manual control tracking task, based on short intervals (1.6 s) of the tracked error and control activity signals. Their result has shown that time series classifiers can be a useful tool in recognizing adaptive HO control behavior, however, it is yet unknown if these methods perform just as well in classification of control strategies that are not so distinctly different. As explained, the types of control behavior in response to different active display types (C, PS, PR) have been observed to overlap in certain situations (e.g., applied compensatory control strategy in pursuit tracking tasks and vice versa) [2]. They are therefore, in theory, indeed much more similar and are expected to result in lower classification accuracy.

Consequently, the goal of this paper is to test the capability of time series classifiers in recognizing HO control strategy in manual tracking tasks with various display types, using short samples of time series data. If successful, the classifier could be combined with existing system identification techniques to improve time-varying HOMs or applied in experiments to gain better insights into what drives human adaptation.

Different state-of-the-art classifiers are tested and the best performing one, a CNN called InceptionTime, is optimized for the problem at hand by means of hyperparameter tuning. Additionally, tests are performed to confirm that the chosen 3-class classifier is the best performing configuration, as opposed to alternatives using several 2-class classifiers. Next, the resulting classifier's performance is evaluated in more detail on different data sets. Finally, the results of one, well-balanced data set are thoroughly analyzed using leave-one-out cross-validation, to provide insights into what sets the different classes apart and to review its potential usefulness in more practical applications.

The paper is structured as follows. Section II provides background information on the manual control task and the effects of the different possible display types. Next, the process and methodology applied are summarized in Section III, after which the results are presented in Section IV and discussed in Section V. Finally, the paper concludes with Section VI.

II. Background

A. The Manual Control Task

The closed-loop manual control task considered in this paper is a target tracking task in which a human operator needs to minimize the error, $e(t)$, between a quasi-random sinusoid target signal, $f_t(t)$, and the controlled element (CE) output, $x(t)$, as shown in the block diagram of Figure 1:

$$e(t) = f_t(t) - x(t), \tag{1}$$

Here the output, $x(t)$, is a direct result of the HO's control output function ($H_o(j\omega)$)w on the stick/manipulator, $u(t)$, the controlled element dynamics, $H_{CE}(j\omega)$, and on some occasions a disturbance signal, $f_d(t)$. All information about the state of the system and the current tracking error $e(t)$ is shown on a display. Different display types show different combinations of signals and thus potentially result in different controls strategies [2].

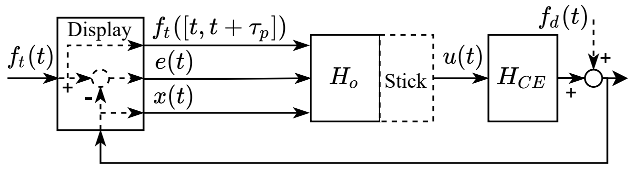


Fig. 1. Block diagram of a basic closed-loop manual control task.

B. Display Types and the SOP Framework

Three main display types have been the subject of extensive research, examples of which are shown in Figure 2. The compensatory (C) display shows the $e(t)$ signal with respect to a stationary, zero-error reference point in the center, while the pursuit (PS) and preview (PR) displays show both $f_t(t)$ and $x(t)$ from which the error can be deduced by the observer. The difference between the latter two displays is that the preview display shows part of the target signal that is still to come, allowing the HO to look τ_p seconds ahead, while the pursuit display does not. Note that while multi-axis tracking tasks exist, this research is limited to single-axis tracking tasks only, for which displacement of all signals is restricted to the horizontal screen direction.

Over the last decades many experiments have been dedicated to quantifying the difference in control performance between identical tracking tasks with different displays. To start, comparisons between compensatory and pursuit displays indicate that the latter generally leads to a better overall performance (i.e., lower average error) [14]–[16]. By being able to directly observe both $x(t)$ and $f_t(t)$, the HO can get a better understanding of the system that they control and in some cases even make short interval predictions based on experience and the location of $f_t(t)$ on the screen [17]. Quantifying these advantages, however, has proven to be difficult as many factors come into play, such as the experience of the HO and the composition of the target signal [14], [15], [18].

Similar experiments have been performed on the differences between pursuit and preview behavior. Here, the differences are even more pronounced, in favor of the preview display [19]–[21]. Clearly, being able to anticipate some seconds ahead into the future allows HOs to reduce their effective time delay, resulting in better target tracking. Depending on the CE dynamics there is, however, a limit to how much preview time is actually useful. Several experiments show that for Single (SI) and Double (DI) Integrator CE dynamics this is around 0.6 s and 1.2 s, respectively [21].

As the displays provide the HO with different types of feedback, various corresponding control strategies have been observed in response to each of them. These strategies were first explicitly defined by Krendel and McRuer [2] in their Successive Organization of Perception (SOP) framework:

1) *Compensatory*: The HOs focus solely on reducing the error ($H_o = H_{o_e}$) because:
   a) it is the only information available,
   b) they are not skilled enough to adopt more sophisticated target tracking behavior given certain

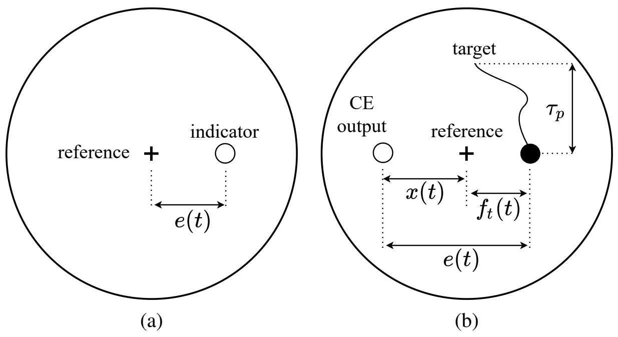


Fig. 2. Examples of compensatory (a) and pursuit/preview (b) displays in a lateral manual tracking task.

circumstances (e.g., stress/distractions etc),

c) adopting a different strategy will not lead to improved tracking [2].

2) *Pursuit*[1]: The HO has enough information and skill with respect to the tracking task to make short interval predictions on the target signal, thereby improving the performance (reduced error) with respect to that shown in typical compensatory control behavior [2]. As the operator uses all three signals, their overall control behavior consists of a combination of three different response functions: $H_o = H_{o_f} + H_{o_e} + H_{o_x}$.

3) *Precognitive*: The HO has near perfect knowledge and skill of the system and task. They are therefore able to completely predict the target signal and apply a control strategy that reduces the error to near zero. This behavior can be regarded as a pure open-loop response since there is no feedback component whatsoever, albeit for a short duration of time [2]. As this research focuses on closed-loop control tasks, precognitive control strategy is beyond scope.

An important aspect of the SOP is that it acknowledges that display type and control strategy do not always match. Point 1b), for example, states that in certain situations unskilled HOs might resort to a compensatory tracking strategy whilst performing a pursuit/preview tracking task in order to improve performance [2]. To illustrate, if we consider Figure 1, an HO can choose to 'switch' between the three different observed signals to focus on, thereby adjusting the control strategy. Such dynamic, time-varying control behavior is unique to humans and understanding it is of key importance in improving our knowledge on manual control cybernetics. We will therefore revisit this phenomenon in more detail in Section II-D.

C. The Human Operator Model

In order to both simulate and better understand/predict human operator behavior, human operator models can be constructed. For a long time, such a model only existed for compensatory tracking tasks. This 'extended crossover model', introduced by McRuer and Jex, [3], consists of a linear, time-invariant frequency response function between the error $e(t)$ and HO output $u(t)$, that consists of an operator equalization component H_o, a neuromuscular component H_{nm}, a time delay e^{τ_v} and lastly a 'remnant' n that accounts for all remaining non-linearities. A block diagram representation of this model is shown in Figure 3. H_o and H_{nm} take the following forms:

$$H_o(j\omega) = K_e \left(\frac{T_{L,e} j\omega + 1}{T_{I,e} j\omega + 1} \right) \tag{2}$$

$$H_{nm}(j\omega) = \frac{\omega_{nm}^2}{(j\omega)^2 + 2\zeta_{nm}\omega_{nm} j\omega + \omega_{nm}^2} \tag{3}$$

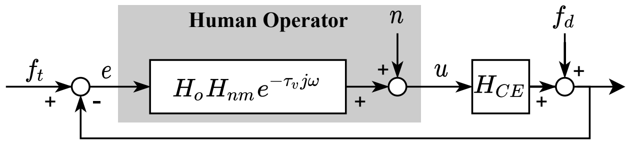


Fig. 3. Block diagram of a compensatory tracking task with the precision human operator model.

Recently, van der El et al. augmented this model in order for it to also be applicable to pursuit and preview tracking tasks [4], the result of which can be seen in Figure 4. Here, the operator equalization component is split up into two parts H_{o_f} and $H_{o_{e^*}}$, to account for the different observed signals in each display. A detailed description of how this was done is out of scope of this research but for reference see [4]. $H_{o_{e^*}}$ takes the same form as Eq. (2) with $e = e^*$, and H_{o_f} is given by [22]:

$$H_{o_f}(j\omega) = \frac{K_f}{1 + T_{l,f} j\omega} \tag{4}$$

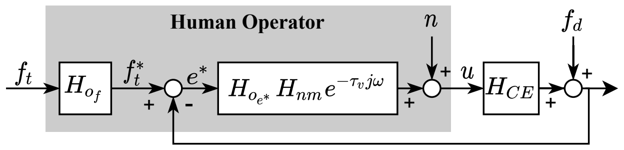


Fig. 4. Block diagram of a generalized model for human control behavior in a tracking task with any type of display.

While these traditional cybernetic models provide a good starting point for representing static human control behavior, so far, they have all been exclusively time-invariant in nature and therefore have limited use. To improve these models, new techniques are necessary to capture time-varying control behavior, preferably in real-time [1].

D. Time-Varying Control Behavior (TVCB)

1) Understanding TVCB: According to Mulder et al. [1], the main driver behind any form of adaptive control in HOs, is their Internal Representation (IR) [23]. The IR is the mental model of the control task that humans build up over time during the learning phase. If something happens that is not in line with the IR, it can trigger an adaptation to the new

[1]In the SOP, preview tracking is seen as part of the pursuit tracking category

situation in order to maintain steady control performance. This could, for example, be a change in control strategy in response to a variation in the CE dynamics or input signal bandwidth. In contrast with such an external event, there exist also internal triggers that are unique to humans and are often psychological or physiological [3]. Examples of this are fatigue, stress, distractions, etc. These effects can cause an HO to lose the mental capability to exercise a more complex control strategy and therefore make them resort to a simpler one. So far, however, much of the above is still only theory due to a lack of measurement tools. As long as there are no methods to recognize control adaptations, experiments can not be designed to determine what triggers them.

2) Modeling TVCB: From the HOMs it becomes clear that time-varying control behavior can manifest itself in two different ways. The first, is through a change of control parameters within a single strategy (e.g., applying a higher/lower gain in compensatory tracking). The second, is through a change in the operator's overall control strategy (e.g., applying a compensatory control strategy in a pursuit tracking task). Until now, most attempts to mathematically model time-varying behavior with traditional cybernetics methods have been on within-strategy parameter changes [5]–[9]. The reason for this, is that the HOM can be assumed to always stay the same (one strategy) and therefore the set of tunable parameters is too. As a result, the challenge lies mainly in finding the combination of parameter values that best fit the data as time progresses.

Several drawbacks of such techniques exist, however. First, is that if one would attempt to apply it to between-strategy adaptation, one would first need to be able to identify the applied control strategy in order to know what HOM to use. Otherwise, the wrong set of parameters could be set as initial conditions and the model will not converge. Second, is that the convergence time is often quite long, up to a few tens of seconds, such that any adaptive human control behavior prior to that is not recognized. Lastly, these traditional cybernetics methods are limited in how they interpret the data. A large part of the HOMs consists of the remnant, which implies that even in time-varying models part of the potentially useful information is lost.

Clearly, both our understanding and modeling capability of time-varying control behavior have a lot to gain from new techniques that can recognize between-strategy control adaptations, using short intervals of tracking data and without discarding any information.

III. Method

The method proposed in this paper consists of four steps, shown in Figure 5. First the classification problem is identified, after which suitable data are selected and configured and several potential classifiers are compared in order to select the best performing one. Next, the chosen classifier is optimized. The best combination of tracking signals as input features is selected, after which the hyperparameters are tuned and a trade-off is performed on two different classifier configurations. The result is an optimized classifier of which the performance is then analyzed by both using data from multiple experiments and performing leave-one-out cross validation on one specific data set. Last, the classifier results are interpreted and an attempt is made to validate them with respect to current knowledge on manual control cybernetics.

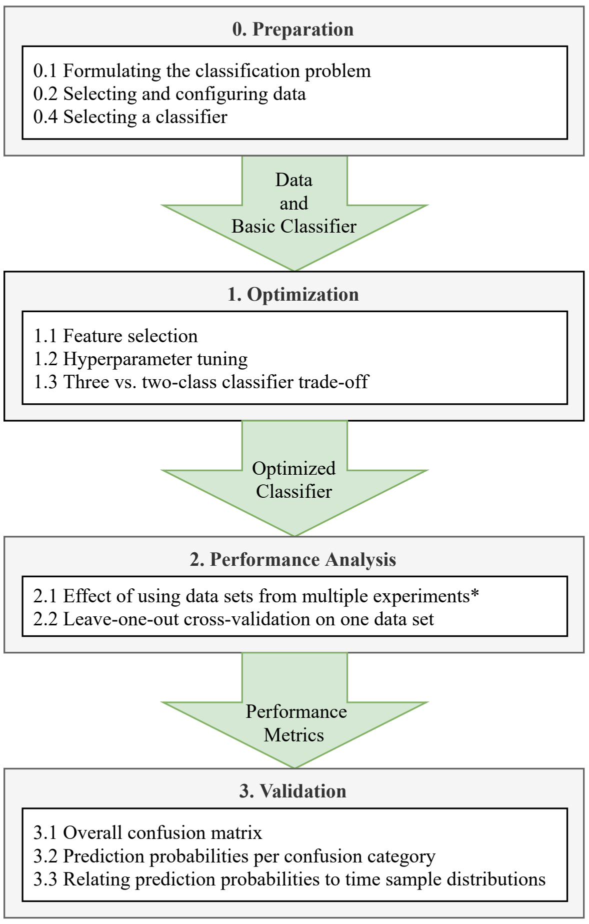

0. Preparation
0.1 Formulating the classification problem
0.2 Selecting and configuring data
0.4 Selecting a classifier
Data and Basic Classifier
1. Optimization
1.1 Feature selection
1.2 Hyperparameter tuning
1.3 Three vs. two-class classifier trade-off
Optimized Classifier
2. Performance Analysis
2.1 Effect of using data sets from multiple experiments*
2.2 Leave-one-out cross-validation on one data set
Performance Metrics
3. Validation
3.1 Overall confusion matrix
3.2 Prediction probabilities per confusion category
3.3 Relating prediction probabilities to time sample distributions

Fig. 5. Flow diagram for the method used in this research. *Multiple data sets were used for step 2.1, whereas in all other steps only the vdEl3 data set is used (see Table I).

A. The Classification Problem

We aim to develop a classifier that can distinguish between different applied control strategies in manual tracking tasks with three different types of displays, based on short samples of labeled time-series data. The classification task is thus a three-class Time Series Classification (TSC) problem where the class labels correspond to the names of the displays used by the HOs. The input is a 1.5 s sample that consists of a combination of the signals recorded in a tracking task. The output consists of three numerical values that indicate with what probability the classifier expects the input sample to belong to each class. A graphical interpretation of how the classifier works is shown in Figure 6.

While there is no specific goal for the level of accuracy that the classifier should attain, it is still seen as the main driving factor in finding an 'optimal' classifier, i.e., the optimal classifier is the one with the highest attainable accuracy. Other considerations, however, are the computational cost (training

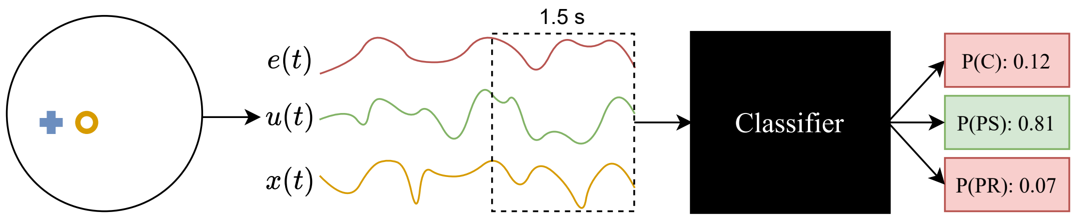


Fig. 6. Graphical representation of how the classifier works for an example pursuit tracking task.

time) and other, more specific performance metrics such as those that can be derived from a Confusion Matrix (CM) (e.g., recall) for each class.

In order to test the feasibility of the research objective, it was decided to limit the classifier scope to tracking tasks for which the control strategies are more distinctly different. Specifically, the focus of this research is on tracking tasks with DI CE dynamics ($H_{CE}(j\omega)= \frac{1}{s^2}$) and a preview time of two seconds for the preview display ($\tau_p = 2$ s). It has been shown that these task variable settings lead to the largest apparent differences in control strategies between tracking tasks with different displays [21], [24].

One thing that complicates the classification task, is the way the samples are labeled. Currently, no methods exist that can confidently distinguish between different control strategies for short intervals of time-data. Therefore, a sample can only be labeled according to the display that was active during the tracking task from which it was taken. However, as explained in Section II-D, samples likely exist for which there is a discrepancy between the HO applied control strategy and the 'ideal' control strategy corresponding to the active display. As a result, samples are labeled according to display type while ideally we would want to label them according to the applied control strategy directly. To illustrate, if the classifier makes a mistake with the current method of labeling, we can not be sure that the error is because the classifier is actually wrong, or because the sample indeed belonged to a different applied control strategy, as humans are known to switch.

This poses a problem, as it implies that we are feeding the classifier with sub-optimal information. Unfortunately, there is no currently available solution because the cause of this problem — a lack of methods to distinguish between active tracking display and control strategy — is exactly what this research is trying to address. Clearly this is a causality dilemma where a breakthrough in creating such a classification method should lead to increasingly accurate classifiers due to increasingly accurate labels. For this research it is therefore decided to accept some inaccuracy in labelling as a possible confound in the results. To obtain a sense of the magnitude of this confound, the results of the final classifier are analyzed in more detail, as will be explained in Section III-G.

B. Data Sets & Configuration

Four data sets from different previous experiments were used for this research. All experiments were conducted in either the fixed-base Human-Machine Interaction Laboratory or the moving base SIMONA simulator at the TU Delft. From each data set, only the tracking runs with DI CE dynamics and preview displays with τ_p are used. An overview of the size of each data set is given in Table I. A more detailed description on the task variable settings of each experiment can be found in the original papers for sake of conciseness.

The main data used for all computations except the final classifier review are from the vdEl3 experiment, for three main reasons. First, it contains runs with varying forcing function bandwidths. This is interesting as it will hypothetically allow the classifier to become more robust by learning from different scenarios. Second, it makes up over 60% of the available data and is equally balanced across all three classes (C, PS, PR). Last, and most importantly, it allows the classifier to be used for cross-validation across the original experiment's subjects, will be detailed in Section III-F.

TABLE I
SECONDS OF DATA PER DISPLAY, PER SOURCE, WITH SETUP IN # SUBJECTS × # RUNS × RUN DURATION

Source	Setup	C, s	PS, s	PR, s
Vos [25]	$8 \times 5 \times 90$	3,600	3,600	-
vdEl1 [24]	$8 \times 5 \times 120$	-	4,800	4,800
vdEl2 [26]	$8 \times 5 \times 120$	4,800	-	4,800
vdEl3 [22]	$\mathbf{9 \times 15 \times 120}$	**16,200**	**16,200**	**16,200**
Total		24,600	24,600	25,800

The data sets contain all time traces of the signals that make up the control task. In order to find the combination of signals that results in the best performing classifier, a trade-off shall be made based on performance (Section III-E). Finding the best combination of signals to be used as classification features, can, however, be influenced by the choice of ML model. Consequently, the classifier trade-off is performed first, with the same time-traces as the ones used by Versteeg ($e, \dot{e}, u$) [13], due to the strong similarities between the classification tasks and the type of HO data. Next, the ideal combination of signals is found as part of the Hyperparameter tuning process in Section III-E.

Before the data can be used as input for a classifier, however, it needs to be processed. Again, a similar processing method is used as the one by Versteeg, as it has proven to be both accurate (high average accuracy) and robust (low variance) [13]. First, the data are standardized with respect to the subset of data they belong to (e.g., all data points in the vdEL3 compensatory tracking data set) such that the set of samples is scaled appropriately. Next, all tracking runs are randomly allocated to either the training (80%) or test (20%) set. In case of the vdEl3 data set, this means that for a total of $9 \times 15 = 135$ tracking runs, $0.8 \times 135 = 108$ are in the training set and the remaining 27 in the test set. By splitting the tracking runs before sampling, we ensure that there is absolutely no overlap in data between the training and test set, even when we include overlap between samples. Lastly, the samples are configured similar to the best settings found by Versteeg [13]. The resulting sample configuration consists of a 1.5 s window size, sampled at a frequency of 50 Hz (i.e., 75 data points per time trace) and an overlap of 50% between consecutive samples. The resulting number of samples in the vdEl3 data set is 63,180 (21,060 per display) split into 50,544 training and 12,636 test samples.

C. Choosing the Classifier

To be sure that the final, optimally-tuned classifier provides the best possible result, three different types of time-series classifiers are compared. One requirement for choosing these three classifiers is that they all take the raw time-series as input, without the need for any feature selection in the time or frequency domain. Finding the right features can be very time consuming and it is less likely to result in the discovery of any relations we do not yet know about.

For this purpose, three candidate neural networks were implemented, based on architectures that have shown promising results in previous, comparable work:

1) *Fully Connected GRU* [13], [27]: A two layer, 100 node, fully connected Gated Recurrent Unit (GRU) network with 20% dropout and a softmax classifier layer. The network was trained for 25 epochs after which the epoch with the lowest categorical cross-entropy loss was used.
2) *GRU Autoencoder with SVM classifier* [28], [29]: A GRU Autoencoder consisting of a two layer, 100 node encoder and decoder both with 20% dropout, connected through a 20 node latent layer, of which the decoder is removed after unsupervised reconstruction training, such that the latent layer outputs can be used as feature inputs for a Support Vector Machine (SVM) classifier with an RBF kernel. The Autoencoder was trained on all data sets from Table I except the vdEl3 data set, as that was used for training the SVM. Furthermore, it was trained only once prior to the start of the main computation phase for a maximum of 100 epochs and with an early stopping criterion of five consecutive increases of MSE of the reconstruction error. The optimal values for the SVM hyperparameters C (100) and γ (1) were found using 5-fold cross-validation over a two dimensional, logarithmic grid-search between 10^{-3} and 10^{3}.
3) *InceptionTime* [30]: An off-the-shelf Convolutional Neural Network (CNN) consisting of stacked ‘Inception modules’ [31]. For a detailed description see Section III-D. In the classifier comparison, the default parameter settings as defined by Ismail Fawaz et al. [30] were used. Again, the network was trained for 25 epochs after which the epoch with the lowest categorical cross-entropy loss was used for evaluation.

All classifiers were implemented in Python 3.7 using the Keras [32] and Sklearn [33] libraries and trained using the Adam optimization algorithm and a batch size of 64 (including the Autoencoder). All networks were trained and evaluated 30 times and their resulting accuracies recorded. Results are shown in Figure 7.

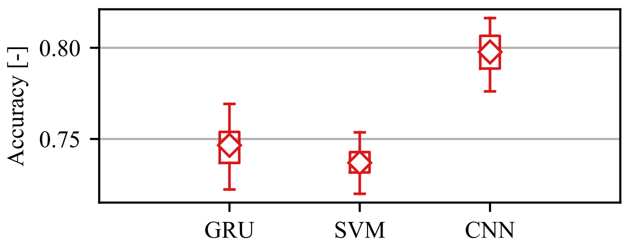


Fig. 7. Accuracy results for three different types of classifiers.

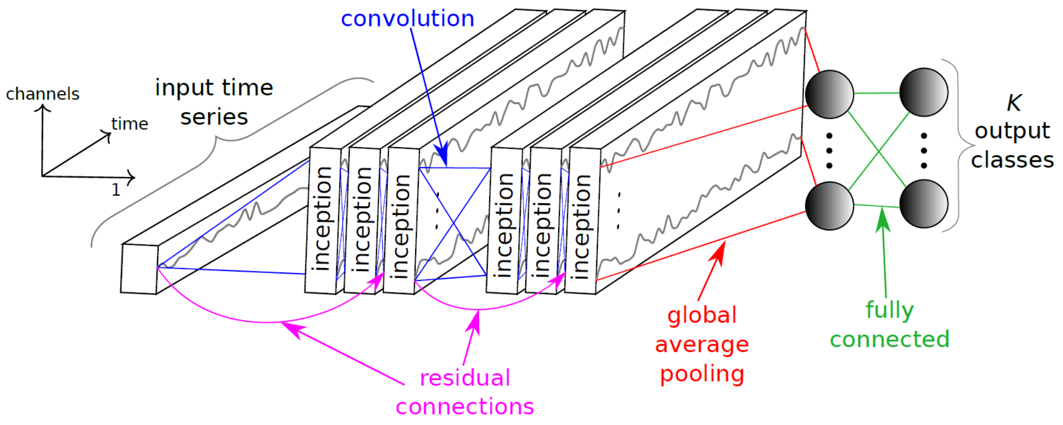
The results clearly show that the InceptionTime network significantly outperforms the other two classifiers, even with the default hyperparameter settings. It should be mentioned that the training time is also three times longer than for the others (approx. 1,200 s vs 400 s for the GRU), however, this is expected to be reduced after hyperparameter tuning and does not outweigh the benefits of the significantly higher accuracy. It was therefore decided to use InceptionTime as the main classifier for this study.

D. InceptionTime

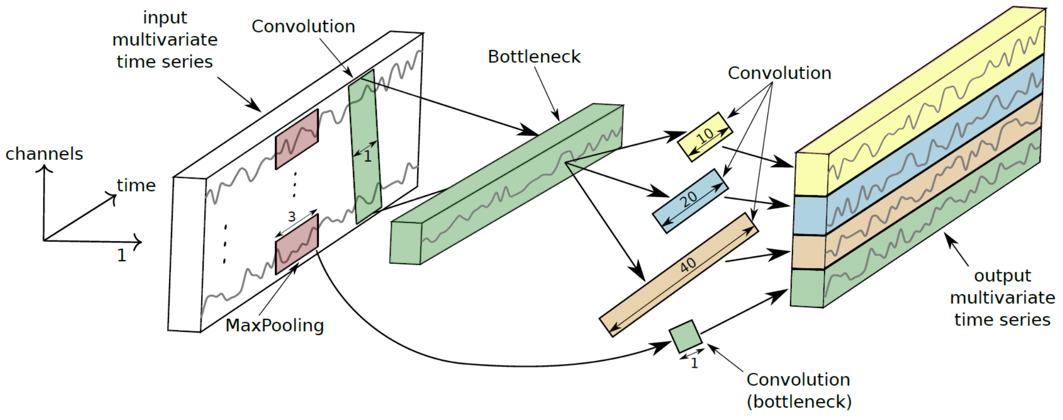
InceptionTime is a Convolutional Neural Network architecture introduced in 2020 by Ismail Fawaz et al. [30] specifically for TSC. Figure 8 shows the general architecture and that of a single Inception module. From the figure, the main building blocks and hyperparameters can be identified such that we can tune the network to fit our classification problem best. Note that this section only provides information relevant to the research at hand, for a more detailed review of InceptionTime the reader is referred to the original paper [30].

Starting with Figure 8a, the network takes an n-dimensional input time series and feeds it into the first set of Inception modules. Additionally, this input time series can be fed forward as a residual connection to every third module. A set of three modules is therefore also referred to as a ‘residual block’. The default network as designed by the author consists of two such residual blocks. The total number of modules used is referred to as the ‘depth’ (d). After the two residual blocks follows a Global Average Pooling layer and finally a fully connected softmax layer for classification.

When taking a closer look at the Inception module shown in Figure 8b, it can be seen that the first component of the module is a so-called ‘bottleneck’ layer. This layer reduces the dimensionality of the problem in that it uses m sliding filters of size and step 1 to reduce the input time series with dimension M, to one of dimension m. Note that for every module except the first, the input time series dimension is not equal to that of the original series ($M \neq n$), but to the output of the previous module. This implies that the bottleneck can also be beneficial if the output of each module is of high dimensionality [30].



(a)



(b)

Fig. 8. Graphical representation of the InceptionTime architecture (a) and a seperate Inception module (b).

As seen in the figure, the output layer consists of four parallel, concatenated sets of convolutional layers with n_f filters. Consequently, the output dimension of an Inception module (and thus the input dimension M of all but the first module) is of size $4 \times n_f$. Of the three main parallel layers, only the largest filter length (or 'kernel size') is a tunable hyperparameter l_f, the other two are simply defined as half and one quarter of this value. The last filter is a Max Pooling convolution layer of size 1.

An overview of the tunable hyperparameters is given in Table II. Depending on the classification task, different combinations of hyperparameters may provide better results. How this optimum is found, will be explained in Section III-E.

E. Optimization

To improve the performance of the classifier, several data and network 'hyperparameters' can be tuned. This is a difficult process as there are often many interdependencies between the different parameters. To give an example, changing the number of Inception modules to an InceptionTime network most likely has a large impact on the ideal number of convolutional filters and vice versa. Moreover, the ideal configuration is also highly dependent on the type and shape of data that is used.

To this end, the process of tuning the hyperparameters is started with a heuristic approach to understanding these interdependencies. Each hyperparameter is tested separately, again using only the $e, \dot{e}$ and u signals to get a sense of what the range of ideal parameters might be. Furthermore, parameters that seem to have a strong interdependence are cross-tested to narrow down the number of possible well performing combinations. Through this process, we attempt to eliminate the possibility of getting stuck in a 'local optimum', by first testing what combinations of parameters seem to correspond to a more global optimum. The resulting initial set of values is given in Table II.

Similarly, tests were done with some of the training parameters to find the suitable number of epochs (50) and the batch size (64) (see the appendix). Note that no early stopping criterion was implemented in any of the subsequent runs. The training progression of InceptionTime shows volatile progression of the test loss, meaning that the loss curve is less smooth (i.e., varies around a trend-line). Consequently, several consecutive epochs with increasing loss do not necessarily mean that a performance limit has been reached and an early stopping criterion might therefore limit performance. Instead, the network from the epoch with the lowest test *loss* was used for evaluation, because loss is a more reliable indicator of performance in classification tasks than accuracy. As with the selection of the classifier, each tuning run is repeated 30 times to take into account the stochasticity of the training process.

TABLE II
INITIAL CONFIGURATION OF THE HYPERPARAMETERS AND THE RANGE TO BE TESTED IN THE TUNING PROCESS.

Hyperparameter	Symbol	Start Value	Test Range
Use Bottleneck	-	no	yes/no
Bottleneck Size	m	$2, 4, 6$	
Use Residual Connections	-	yes	yes/no
Depth	d	4	$3-6$
Max Kernel Size	l_f	32	$16, 32, 64$
Number of Filters	n_f	16	$16, 24, 32$

All runs from here on out are executed on an NVIDIA Quadro K620 GPU, such that all training times can be meaningfully compared.

1) Feature selection: With this set of starting parameters, only the definite choice for what combination of signals should be used as input features remains, before the hyperparameter tuning process can be performed. Versteeg et al. [13] showed that some combinations of signals show very poor performance and are thus likely not a viable option. Consequently, a more methodical, greedy approach is used – as opposed to an exhaustive one – to reduce the computational load. First, all four combinations of at least two of the measured time signals are reviewed:

1) e, u, x
2) e, u
3) e, x
4) u, x

The best performing combination of this trial (e, u, x, see Figure 9) was chosen as the baseline for the subsequent steps. Next, each of the first-order derivatives ($\dot{e}, \dot{u}, \dot{x}$) was introduced to the baseline (1) separately (2-4), as well as all together (5), such that the effects of both their individual contributions and any possible synergies could be analyzed:

1) e, u, x
2) e, $\dot{e}$, u, x
3) e, u, $\dot{u}$, x
4) e, u, x, $\dot{x}$
5) e, $\dot{e}$, u, $\dot{u}$, x, $\dot{x}$

From these five options, the combination with the highest accuracy is picked for use as input features throughout the rest of the research.

2) Hyperparameter tuning: With the initial configuration of hyperparameters set and the best performing combination of signals known, all necessary conditions for hyperparameter tuning are met. To tune the parameters, it is decided to again use a greedy approach, as a full grid-search is computationally too expensive. The used approach works as follows.

All parameters are set to the start values listed in Table II. Next, a first hyperparameter A is chosen for 'optimization'. Several different values of this active hyperparameter are tested and the best performing one is selected. To decide on what the best performing setting is, a qualitative trade-off is performed between improved accuracy and training time. In general, improved accuracy is leading, but if the difference is marginal and requires a significantly longer training time, the second best, faster option may be chosen. Next, the chosen value for A is frozen, and the subsequent hyperparameter B is selected, and so forth, such that all parameters are optimized with respect to the previous ones.

It should be noted that in this method the order in which the parameters are optimized can have a significant impact. Clearly, if parameter A is frozen, the optimal value found for B then depends on A. The risk of such a greedy approach is that another, more optimal value could be found if, for example, B is frozen first. While this is a risk that exists, it is deemed unavoidable in high-dimensional hyperparameter tuning due to the rapid increase of computational time. Therefore, the result of the process described here is a pseudo-optimized classifier. The order in which the hyperparameters are optimized in this study, shown in Table II, is the same as the order that the developers of InceptionTime used for their sensitivity analysis [30].

3) Three vs. two-class classifier: So far, all tests have been performed with a three-class classifier. However, as explained, it is expected that the distinction between classes is not always clearly defined. Especially the differences between compensatory and pursuit control behavior are generally thought to be small (see Section II-D). Therefore, other methods that split the classification task into different stages might show better performance. Such a classifier, for example, could first decide on whether or not a sample seems like compensatory behavior, with a certain probability, and if not, use that probability to subsequently decided if it is pursuit or preview, i.e., low P(C) higher P(PR), high P(C) higher P(PS). To decide whether or not such an alternative method is worth investigating, the three-class classifier (3CC) is compared to the three two-class classifiers (2CC) that would be needed to replace it.

To investigate this, the accuracy metric alone is not sufficient because it should be interpreted differently for a 3CC than for a 2CC. Instead, the confusion matrices (CMs) of each classifier can be investigated to see how many samples they assign to each label. If the 3CC makes fewer or approximately equal mistakes than the 2CCs in each respective comparison, then it can be said that the 3CC performs sufficiently.

F. Performance Analysis

The result of the optimization process is a pseudo-optimized classifier with a set average classification accuracy and training time. However, so far, the classifier has only been trained (and tested) on random collections of tracking runs from the vdEl3 data set. It is therefore impossible to determine how the classifier performs when presented with data from different sources, which is essential in order to evaluate the effectiveness of the classifier in more practical applications. Note that in this case, a data source refers not only to a different experiment, but also to different subjects within an experiment. It is known that there exist differences in control behavior between different test subjects (see for example [3], [4], [16]) and the classes might thus be more easily distinguishable for one HO than for another. In order to test the robustness of the classifier with respect to such variations in the data, two different experiments are conducted.

1) Leave-one-out cross-validation: First, the vdEl3 data is cross-validated using nine folds corresponding to the original experiment's nine test subjects. Each HO's tracking runs are used as the test set of 30 training repetitions. The accuracies can then again be drawn in a box plot and the CMs can be averaged to reveal the overall evaluation of the classifier on each fold's data.

2) Generalization to multiple data sets: Second, the other data sets from Table I are used in order to test the robustness of the classifier with respect to tracking runs from experiments with different task and environment variables (e.g., different target signals $f_t(t)$). In total, three different training and evaluation steps are performed with the following train/test distributions:

1) Train: *vdEl3*, Test: *vdEl1, vdEL2, Vos*
2) Train: *vdEl1, vdEl2, Vos*, Test: *vdEl3*
3) Train: *All*, Test: *All*

Note that in each run a training ratio of 80% is maintained and that the runs are only executed once. The main purpose of these validation tests is not to make a direct comparison of which is better, but to get a feeling for the level of robustness of the classifier. The accuracies of each test are then compared to determine the importance of using multiple data sets in training for robustness. Additionally, both the accuracy and confusion matrix of the last run can be broken down per data set to see the individual performances.

G. Validation

An attempt is made to validate that the classifier is working as intended. While the performance results detailed in Section III-F give an indication of *if* the classifier is working, it does not provide any insights into *how* it is working. It simply takes a sample, makes a prediction and tells us whether or not that prediction is correct with respect to the label. However, because the label is not always completely accurate due to possible overlap of control strategies (see Section III-A), we can not directly validate whether the classifier is picking up the right things based off of the performance results alone. We attempt to solve this problem by performing an analysis on both the correct and incorrect classified sample groups. These groups are compared to see if there are any noticeable differences that can confirm that the classifier is picking up on information that is in line with current knowledge on cybernetics.

To perform this analysis, first it should be decided what constitutes as a correctly or incorrectly classified sample. Different training runs can result in different classification models and a test sample is therefore not always given the same prediction, meaning that sometimes it is correct and sometimes it is not. Therefore, data from the nine-fold cross validation on the vdEL3 data set are used (Section III-F), because an additional result of those tests is that all samples are assessed as a test sample exactly 30 times. With these predictions, it is possible to draw a confusion matrix based on what prediction is *most often* made for each sample. To illustrate, each individual assessment has one of three possible outcomes (categories): either the prediction corresponds to the actual label (diagonal entries) and the assessment is correct, or the sample is wrongly classified as one of the two other classes and the assessment is incorrect (off-diagonal entries). Since there are three different ground truths, the result is a total of $3 \times 3 = 9$ categories. At the end of the 30 cross-validation runs, the number of different outcomes for each sample are compared. Subsequently, the sample is allocated to the category to which it was assigned the most. If there is a tie, it is allocated to the 'correct'-category.

The results of the process described above, are the completed CM and the corresponding nine groups of samples that make up each of its categories. Both can provide some insights into how the classifier is functioning.

1) Overall confusion matrix: The confusion matrix by itself summarizes which mistake the classifier is making the most. This is important information for any potential future use, as it shows the classifier limits. If we consider previous studies that researched the differences in control behavior in response to each display, such as those comparing compensatory and pursuit [14], [15], [18] and pursuit and preview [19]–[21], we can make a hypothesis on what the error will be most commonly made by the classifier. The C-PS comparisons have largely shown a lot of overlap between strategies and quantifying differences in control performance has been difficult. Conversely, the PS-PR comparisons resulted in significant differences, especially for preview times of 2 s such as considered in this research. It is therefore expected that the classifier will have more trouble separating the C-PS classes than the PS-PR and especially the C-PR classes.

2) Prediction probabilities per confusion category: The average confidence with which the classifier is predicting the labels of the samples in each group can be extracted from the softmax layer output. A high average probability value for the correct sample groups would indicate that the classifier is not merely correct, but that it can easily distinguish between classes.

3) Time sample distributions: Last, the categories can be used to draw distribution diagrams that indicate the overall shape of the data in each group. These distributions show what the average sample from each category looks like per signal. If there exist any differences and they correspond to current knowledge on manual control strategies, it can be an indication that the classifier is working as intended. An example of such a difference, is that tracking tasks with preview displays generally lead to much better performance, and thus a lower variance in the error σ_e^2. Therefore, it is expected that the samples that are predicted to belong to the preview category have a distribution that is "flatter" than those predicted as compensatory or pursuit.

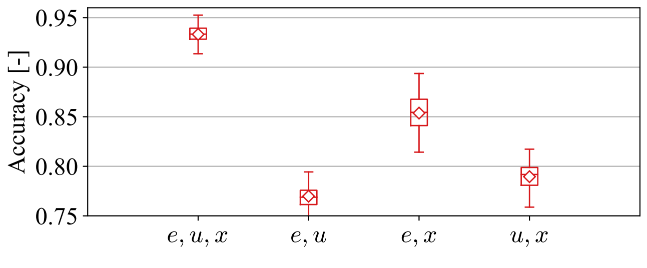
IV. Results

The main results are summarized here. It should be noted that *all* accuracies used to evaluate the classifier throughout this research are the *test* accuracies, as opposed to the training accuracies. The order in which the results are presented corresponds to the steps taken in the method: optimization first, followed by the performance analysis and finally validation.

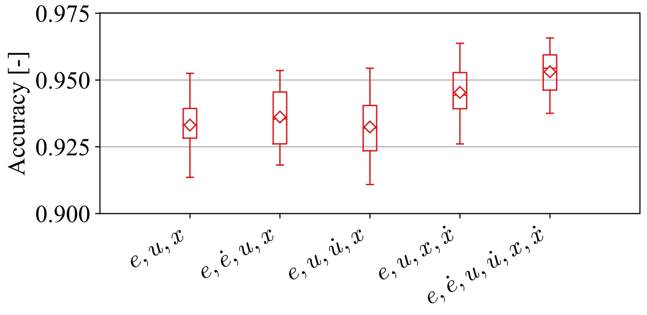
A. Optimization

1) Feature selection: The results for selecting the most optimal combinations of input signals are shown in Figure 9. Figure 9a shows that the addition of x gives the classifier a large boost in accuracy with respect to one that only uses the e/u signal combination. The other two columns indicate that the difference of almost 20% originates mostly from the "synergy" between the e and x signals. This is not unexpected, as the combination of these two signals indirectly also gives the network information on $f_t(t)$ (Eq. (1)). Consequently, the classifier might be able to recognize the different delays in response times between the different classes. Note that due to the black-box nature of neural networks, it is difficult to verify this hypothesis, but it would correspond to our current knowledge of manual control cybernetics.

Next, Figure 9b shows that the differences in performance when adding the first-order derivatives, are smaller. While the $\dot{e}$ and $\dot{u}$ signals by themselves seem not to contribute anything significant, together, all derivative signals provide an accuracy boost of approximately 2%. A logical explanation for this is that with more sources of information, the classifier has access to more potential interrelations between signals. It was therefore decided to continue the remaining simulations with all six input features. Consequently, the accuracy baseline at the start of the hyperparameter tuning is approximately 95.3%, which is achieved in an average training time of 2,220 s.



(a)

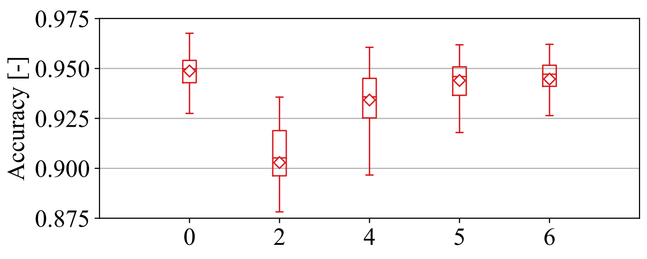


(b)

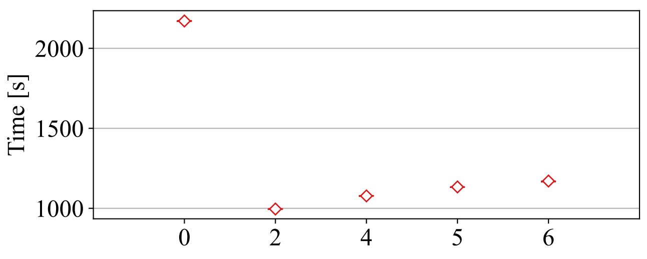
Fig. 9. Accuracy results for different combinations of time signals (a) and their first order derivatives (b).

2) Hyperparameter tuning: With the baseline set, the parameter tuning process that aims to improve the performance with respect to the baseline with each step, can start. As an example, the results of the first hyperparameter, the bottleneck size, are shown in Figure 10. The box-plots for accuracy show a clear pattern. The exclusion of a bottleneck layer (option 0, our baseline) seems to give the best average and maximum performances, although the differences with options 5 and 6 are small. If we then examine the corresponding training times, however, it becomes clear that the slight advantage in accuracy for option 0 is countered by a near doubling of the computational cost. In this situation, it is therefore decided to select a bottleneck size of 6 as the definite value for this hyperparameter.

Additionally, what can be observed, is that although a bottleneck size of 0 has the exact same settings as the baseline (Figure 9b), the results are slightly different. This implies that the stochastic component of training is still not entirely filtered out by the 30 repetitions. Therefore, for any two results that are very close together, no clear winner can be elected, because in another identical simulation they might show the exact opposite result.



(a)



(b)

Fig. 10. Classifier accuracy (a) and training time (b) for different bottleneck size settings, where 0 means that no bottleneck was used.

Similar figures are generated for the four remaining hyper-parameters, however, for the sake of conciseness, they can be found in the appendix. Table III contains the final combination of hyperparameters that results in optimal performance. The average accuracy attained by the optimized classifier is 95.4% with a training time of approximately 1,200 s. If we compare this to the baseline (95.3% accuracy in 2,220 s), we find that the gains in accuracy are negligible while the training time is reduced by 46%. This result is important as it seems to suggest that a limit has been reached on the accuracy of the classifier for this data set, though it is impossible to verify such a hypothesis without doing a comprehensive grid search over all parameters.

3) Three vs. two-class classifier: The accuracies for the four different classifiers are pictured in Figure 11. The figure shows that both the C-PS and C-PR 2CCs outperform the 3CC by 2% and 3% respectively, while the PS-PR classifier performs worst. However, the differences are small and it

TABLE III
OPTIMAL HYPERPARAMETER SETTINGS

Hyperparameter	Value
Use Bottleneck	*yes*
Bottleneck Size	6
Use Residual Connections	*no*
Depth	3
Max kernel size	64
Number of filters	24

seems as if there is not a lot to gain from experimenting with different classifier structures in terms of performance. Additionally, as explained in Section III-E, just looking at the accuracies can give a skewed representation of the performance. To understand this better, the averaged confusion matrices are shown in Table IV[2].

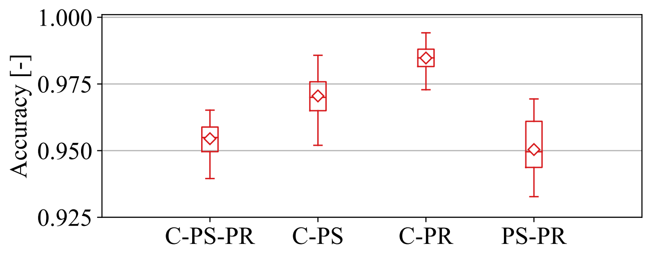


Fig. 11. Accuracy results for the 3CC and three 2CC.

TABLE IV
AVERAGED CONFUSION MATRICES, INCLUDING THE RECALL RATES AT THE BOTTOM, FOR THE 3CC (A) AND 2CC (B-C) CLASSIFIERS, WITH AN EXAMPLE OFF-DIAGONAL ENTRY COMPARISON IN BOLD.

(a)

		Predicted		
		C	PS	PR
Actual	C	4,090 97.1%	**97** **2.3%**	23 0.5%
	PS	122 2.9%	3,947 93.8%	141 3.3%
	PR	33 0.8%	156 3.7%	4,022 95.5%

(b)

		Predicted	
		C	PS
Actual	C	4,092 97.2%	**119** **2.8%**
	PS	129 3.1%	4,082 96.9%

(c)

		Predicted	
		C	PR
Actual	C	4,153 98.6%	58 1.4%
	PR	70 1.7%	4,141 98.3%

(d)

		Predicted	
		PS	PR
Actual	PS	4,031 95.7%	180 4.3%
	PR	238 5.7%	3,973 94.3%

With these matrices, we can compare the off-diagonal entries of the 3CC to those of the 2CC, e.g., comparing respectively the C-PS entry of 98 errors (3CC) to the C-PS entry of 120 errors (2CC), highlighted in bold in Tables IVa and IVb. This shows that in comparison the three-class classifier generally performs better at separating the individual classes than the two-class classifiers separately. Considering these findings, as well as the fact that a 3CC is significantly easier to implement, it is concluded that a 3CC works sufficiently well to not warrant any additional research into any 2CC alternatives.

[2]Due to rounding after averaging, both the absolute numbers and the percentages do not always add up as expected in the CMs shown in this paper.

B. Performance Analysis

In the previous section, the designed classifier was optimized for the task at hand. However, until now the classifier was trained with random subsets of data from only a single experiment (vdEl3). To get a better understanding of how the classifier performs on other data sets, the following results are examined.

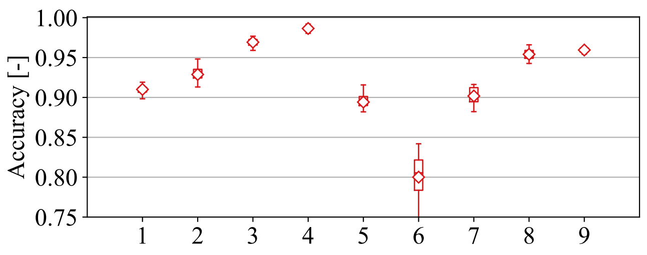


Fig. 12. Accuracy results for nine-fold cross validation on the vdEl3 data set, with each fold representing data from one subject in the original experiment.

1) Leave-one-out cross-validation: Figure 12 exhibits the average accuracies for nine-fold, leave-one-out cross-validation run on the vdEl3 data set. Each fold represents one subject from the original experiment [22]. Interestingly, the differences between some of the subjects are significant. For example, using data from fold 4 for testing, instead of fold 6, provides an average accuracy that is almost 20% higher. To get a better understanding of this, the respective confusion matrices are shown in Table IV.

From this table it can be seen that while the general performance of Subject 6 is indeed much worse, the main reason for this is the confusion between the Compensatory and Pursuit classes. In contrast, the leading source of errors for the standard and fold 4 classifiers (Table IVa), occur in the Pursuit-Preview comparison. These results therefore seem to suggest that Subject 6's applied control strategies for Compensatory and Pursuit tracking overlap a lot more than is the case for other subjects and in particular Subject 4, such that the classifier has more trouble separating them. If we observe a compensatory tracking run from Subject 6, such as run 6 partially shown in Figure 13, we can see that there are both some random short outliers, as well as more sustained periods of changed control behavior. Several possible reasons for this exist, such as a lack of experience or fatigue (see Section II-D), however, these are all difficult to verify with historic data. Regardless of their cause, these findings have significant implications for the apparent accuracy of the classifier, as will be discussed further in Section V.

TABLE IV
AVERAGED CONFUSION MATRICES FOR SUBJECT 4 (A) AND SUBJECT 6 (B) OF THE VDEL3 DATA SET.

(a)

		Predicted		
		C	PS	PR
Actual	C	2,324 99.4%	12 0.5%	2 0.1%
	PS	1 0.0%	2,276 97.3%	62 2.7%
	PR	0 0.0%	16 0.7%	2,323 99.3%

(b)

		Predicted		
		C	PS	PR
Actual	C	1,994 85.3%	293 12.5%	51 2.2%
	PS	560 23.9%	1,700 72.7%	79 3.4%
	PR	132 5.6%	286 12.2%	1,920 82.1%

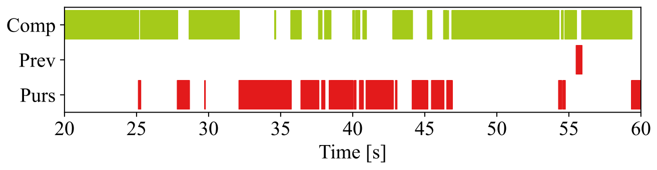


Fig. 13. Classifier prediction progression (every 0.02s) over compensatory tracking run 6 from Subject 6 of the vdEl3 data set (zoomed in for clarity).

2) Generalization to multiple data sets: Next, the results from using different data sets for training and testing are compared in Table IV. What is clear, is that if the data in the test set are not also used in training (Runs 1 and 2), the accuracy drops significantly. Moreover, if only one data set is used for training (Run 1), the accuracy drops to a level that is not much better than chance (33%). It can therefore be concluded that in order to create a robust classifier, different experimental data sets should be used when possible. However, if we compare the result from Run 3 (93%) with the accuracy presented for the 3CC trained on only the vdEl3 data (95.4%), we see that the latter is about 2.5 percentage points better. Hence the results suggest that by creating a more robust and diverse classifier, concessions are made in terms of accuracy. To examine this further, the result from Run 3 can be split up into results per data set which is shown in Table V.

TABLE IV
ACCURACIES (1 RUN) FOR THREE DIFFERENT DISTRIBUTIONS OF TRAIN AND TEST DATA.

Run	Train Data (80%)	Test Data (20%)	Accuracy
1	vdEl3	vdEl1, vdEl2, Vos	47%
2	vdEl1, vdEl2, Vos	vdEl3	58%
3	All	All	93%

Starting with the CM for the vdEl3 data shown in Table Va and comparing it to the CM from Table IVa, it is clear that the introduction of the new data has made it more difficult for the classifier to distinguish between the different classes in this data set. Specifically, the recall rate, defined as the number of correctly classified samples per class, drops over 2% for all classes, which is equal to the overall drop in accuracy. In fact, if we examine the other CMs as well, it seems that this drop in accuracy from the vdEl3 data set is the main driver behind

TABLE V
CONFUSION MATRICES FOR RUN 3 OF TABLE IV, SPLIT UP ACCORDING TO THE VDEL3 (A), VDEL1 (B), VDEL2 (C) AND VOS (D) DATA SETS, WITH THE ABSOLUTE NUMBERS AT THE TOP AND THE RECALL RATE AT THE BOTTOM.

(a)

		Predicted		
		C	PS	PR
Actual	C	3,993 94.8%	191 4.5%	28 0.7%
	PS	250 5.9%	3,842 91.2 %	120 2.8%
	PR	54 1.3%	223 5.3%	3,935 93.4 %

(b)

		Predicted		
		C	PS	PR
Actual	C	0 0%	0 0%	0 0%
	PS	6 0.5%	1,116 89.4%	126 10.1%
	PR	1 0.1%	32 2.6%	1,215 97.4%

(c)

		Predicted		
		C	PS	PR
Actual	C	1,216 97.4%	3 0.2%	29 2.3%
	PS	0 0%	0 0%	0 0%
	PR	86 6.9%	6 0.5%	1,156 92.6%

(d)

		Predicted		
		C	PS	PR
Actual	C	901 96.3%	35 3.7%	0 0%
	PS	43 4.6%	889 95%	4 0.4%
	PR	0 0%	0 0%	0 0%

the overall drop in performance, along with the poor recall rate of the Pursuit class in the vdEl1 set, but there is better generalization.

Furthermore, again comparing Tables IVa and Va, the non-diagonal entries show that a classifier trained on *only* the vdEl3 set has most trouble separating Pursuit and Preview, while for the classifier trained on *all* data, the problem lies predominantly between the Compensatory and Pursuit classes. Just like the cross-validation results from Table IV, this suggests that the training data highly influence what displays are the limiting factor in designing a universally strong classifier.

C. Validation

Last, it is attempted to explain the performance of the classifier by examining the sample categories as defined by the overall confusion matrix. The size of each category in absolute numbers and as percentage of the total is shown in Table V, along with the average confidence with which the classifier assigns the samples to each group. The fourth column shows recall, meaning that the percentages of the rows with mutual 'Actual' entries add up to 100%, e.g. 96.8+2.8+0.4=100 for the samples labeled as compensatory (C).

1) Overall confusion matrix: The table shows that, contrary to what was expected, the classifier has more trouble separating the pursuit and preview classes (661+1,075=1,736 errors) than the compensatory and pursuit classes (595+773=1,368 errors). This is surprising and can indicate that there exists either more overlap between PS and PR control behavior or less between C and PS, than previously expected. While it is difficult to verify which of these two is the case (or both), an attempt is made in Section IV-C3.

2) Prediction probabilities per confusion category: Examining the last three columns, we can see that the classification probability for the correct categories is on average very high, approximately 92% for PS-PS and 95% for C-C and PR-PR samples[3]. The difference between these approximate confidences is mainly due to the fact that for the Pursuit samples both the other two classes are seen as a somewhat viable alternative (probabilities of around 4%) and vice versa. This is in line with expectation, because we know from literature that Pursuit control behavior is more likely to overlap with the other two classes due to its 'central' position in the SOP (see Section II-D). Furthermore, the incorrect groups in Table V show that although here, the classifier is indeed making the wrong predictions, it is doing so with a higher degree of uncertainty. This suggests that the differences between classes are not always very distinctive.

TABLE V
SAMPLE CATEGORY SIZES, RECALL AND THE CLASSIFICATION CONFIDENCE FOR THE ENTIRE VDEL3 DATA SET.

Act.	Pred.	Size	Recall	P(Comp)	P(Purs)	P(Prev)
C	C	20,387	96.8%	0.949	0.042	0.010
PS	PS	19,626	93.2%	0.037	0.918	0.045
PR	PR	19,823	94.1%	0.009	0.043	0.948
C	PS	595	2.8%	0.281	0.694	0.025
C	PR	78	0.4%	0.308	0.088	0.604
PS	C	773	3.7%	0.709	0.241	0.014
PS	PR	661	3.1%	0.015	0.314	0.671
PR	C	162	0.8%	0.601	0.158	0.241
PR	PS	1,075	5.1%	0.044	0.704	0.251

3) Time sample distributions: With the categories of samples from Table V, distribution diagrams can be extracted of each signal to determine if the classifier is picking up on information that corresponds to how current cybernetics views control behavior in response to the three types of displays. The x signal distributions are added as part of the appendix because it mostly shows similar patterns as u. Figure 14 shows these diagrams for two sets of categories. The first set compares the distributions of all samples within the C and PS classes and the second of those in the PS and PR classes. A similar comparison is drawn for C and PR, however due to the limited size of those categories it is part of the appendix. It should be noted that due to the differences in category sizes (column 4 of Table V), each distribution is drawn from a different number of samples.

Although we try to explain some of the logic behind the classifier in the following paragraphs, it should be noted that these are most likely gross oversimplifications of how the classifier actually functions. The figures lump all samples together and are nowhere near sufficient to understand all the non-linearities that the classifier most likely uses. The reasoning should therefore rather be interpreted as an attempt to validate and interpret the results in light of current knowledge of manual control cybernetics, than as a complete explainability analysis. Nonetheless, the results in this section contribute significantly to our understanding of (and confidence in) the classifier and are therefore discussed here.

[3]All categories are tagged as *Actual-Predicted*.

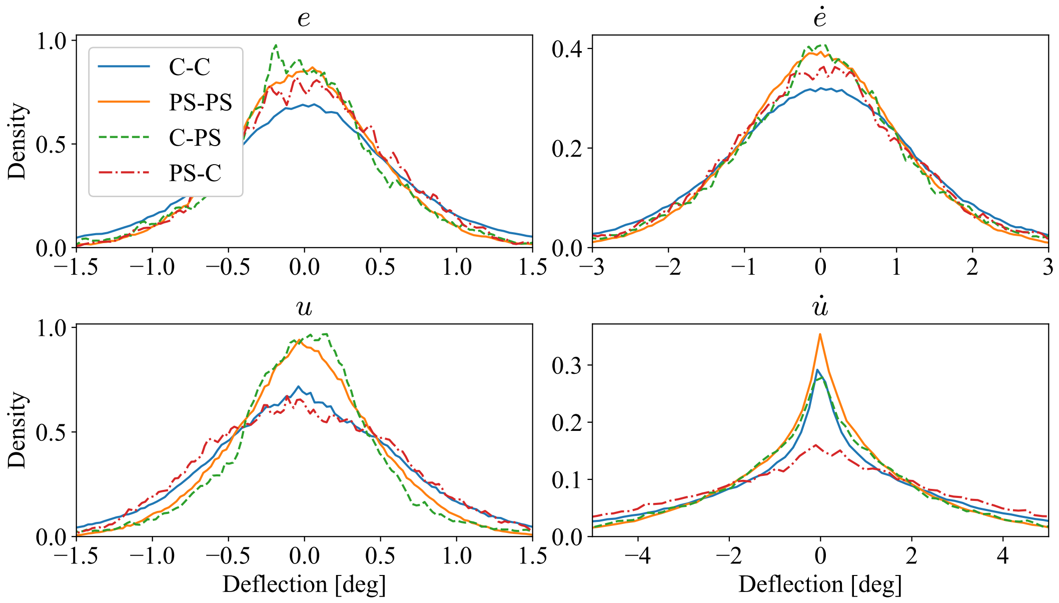
Figure 14a shows that the C-C (blue) and PS-PS (orange) categories have significantly different distributions for all four of the considered signals ($e, \dot{e}, u$ and $\dot{u}$). All signals of the Pursuit samples have on average a smaller variance σ^2, but it is the most evident for u and least for $\dot{u}$. This indicates that both the error and the control activity are on average smaller in magnitude for Pursuit tracking than for Compensatory, which is in line with expectation [17]. In contrast, the diagrams also indicate that the C-PS (green) samples have distributions much more in line with the PS-PS samples than the C-C, while the opposite is true for the PS-C (red) category. It therefore seems logical that the classifier is mistakenly assigning these samples to the 'wrong' classes, because they simply correspond better. Unfortunately, it is near impossible to verify such an hypothesis due to the black-box nature of the considered neural network. However, it does improve the credibility of the classifier, as the results align well with the expectation.

A similar analysis was performed for the PS and PR classes and is shown in Figure 14b, but the results are different. Where in the previous comparison the correctly classified samples showed distinctive distributions for all signals, in this case only e seems to set the categories apart because the distributions for u are near identical. Consequently, it is likely that the classifier is to a certain extent limited to the information in e and is therefore more probable to make a mistake. This hypothesis is strengthened if we examine the incorrect categories, because they are actually very similar in e too and right in between the correct categories. Therefore, if the classifier usually relies on the error signal to distinguish between classes, it is limited significantly when there is no useful information there. This could indicate that some overlap exists between the classes, such that the 'decision boundary' can not completely separate them, resulting in a lower accuracy.

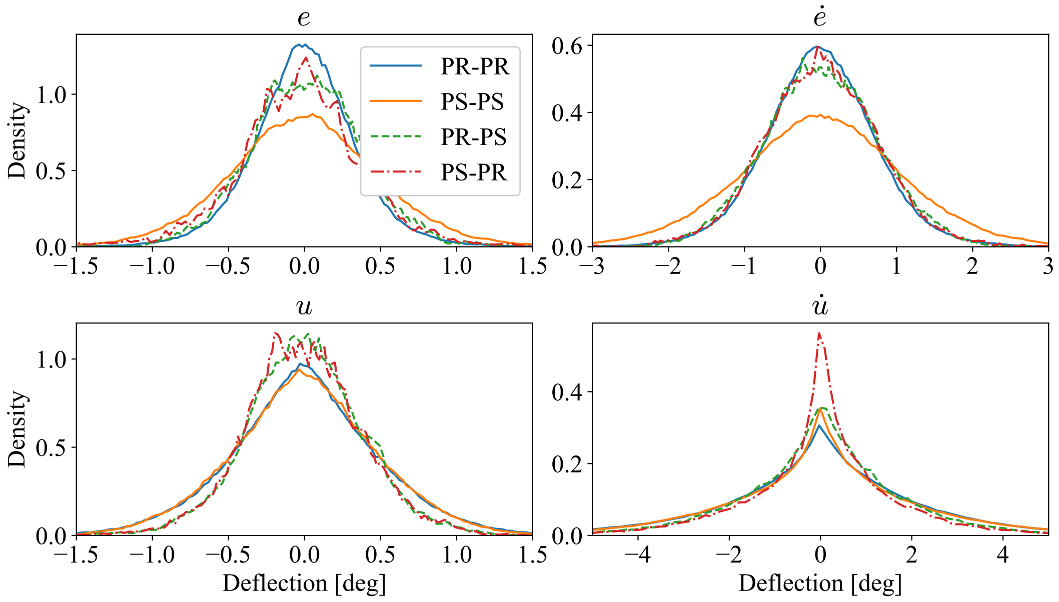
V. DISCUSSION & RECOMMENDATIONS

The objective of this study is to determine how feasible it is to design a classifier that can distinguish between different HO applied control strategies in manual tracking tasks with three types of displays. The result is an InceptionTime neural network classifier that uses 1.5 s windows of tracking data and that is optimized to an effective average accuracy between 93% and 95.4%, depending on the data that is used to train it. While this result exceeds expectations, still several challenges exist that need to be addressed before the presented classifier can be of use. In the paragraphs to follow, some of the key results presented in Section IV are reviewed and reflected upon in order to identify these challenges. Additionally, recommendations are made for future research opportunities that may contribute to solving them.

To start, it can be concluded that the choice of both the type of classifier and the input signals used for training are of key importance in achieving a high level of accuracy. InceptionTime significantly outperformed RNN alternatives, even with the default parameter settings, but the most important boost in



(a)



(b)

Fig. 14. Signal distributions for samples in various categories, comparing Compensatory and Pursuit classes (a) and Pursuit and Preview classes (b).

performance comes from the introduction of the x signal as an additional input parameter. It is hypothesized that the synergy between x and e allows the classifier to get a sense of the effective time delay in the HO response, which is a distinctive property that characterizes each control strategy. Clearly the resulting boost in accuracy of $\sim$15% is useful when training a classifier to recognize control strategies based on display type. However, Versteeg et al. showed that including both u and x in a classifier that distinguishes between controlled element dynamics, results in a perfectly accurate, trivial classifier every time due to the direct link between these signals. Therefore, if both classifiers are combined in any future applications this could result in a conflict where x will need to be dropped, resulting in a corresponding loss of accuracy. While not detrimental for the usefulness of this research, it should be heeded by anyone who wants to attempt such a combination in the future.

Continuing, it was also found that tuning of InceptionTime's hyperparameters results in a negligible increase in accuracy (0.1%), although the training time is reduced by approximately 45%. The fact that the accuracy remains relatively constant for different parameter configurations, suggests that with the available data (vdEl3) something resembling a performance limit was reached. We will revisit this hypothesis shortly.

Next, Table IV shows that if we introduce different data sources, the classifier only manages to reach similar accuracies as before ($\sim$93%) when all data sets are used for both training and testing. This is a significant result as it indicates that the classifier is susceptible to other, subtle task and environment variables, despite the standardization of all used signals. Consequently, to use the classifier in a real-time application, it needs to have learned its specific properties before the event starts. However, in most cases no data will be available to do this yet, because the task and environment variables are new/unique. This is the first major challenge to overcome before the classifier can be used in any experimental or practical applications. To solve this problem, a sensitivity analysis should be performed using data with several different task and environment variables, to see what variables have the largest influence on the classifier's ability to generalize to new data. Once this is known, the classifier can be trained on data containing sufficiently varied samples such that any newly introduced tracking task no longer falls outside of the classifier's scope. Until then, a possible solution could be to simulate a highly variable data set to train the classifier. Using a lot of simulated data might make the classifier so robust that no new training is necessary whenever it is used in new experiments. However, since no tests with simulated data have been performed, additional research is necessary to verify if this would work as intended.

Returning to the results that were generated using only the vdEl3 data set, specifically the cross-validation analysis, it became clear that there is a significant discrepancy in accuracy when evaluating data from different subjects, with the worst score at 80% and the best up to 99% (see Figure 12). While this might seem like a problem, it in fact indicates that the classifier is working properly. If one subject is, for example, less experienced/more fatigued and therefore switches more between control strategies, this will naturally lead to a lower classification accuracy. While this is thus not per definition disadvantageous, it does have a large impact on how the performance should be evaluated in any possible future applications.

To understand this concept better, a more detailed examination of what an error signifies follows here. Because while a correctly classified sample's meaning is unambiguous, an incorrect prediction can have one of four underlying drivers:

1) *Classifier*: A sample is classified erroneously due to a sub-optimal classifier that does not perceive the key defining characteristics/features, i.e., the classifier makes a mistake.
2) *Data Outlier*: A sample's characteristics do not fit any of the prescribed classes resulting in a nonsensical prediction, no matter the classifier. An example of an outlier sample from a tracking task could be a short window of time in which the operator has lost control of the CE and is in a recovery mode or even one where the HO blinks. Such a sample would consequently not correspond to any of the three known control strategies. Due to the complex nature of HOs the data will always contain these types of errors and thus they should always be considered in practical applications.
3) *Data Overlap*: The characteristic features of different classes overlap, such that there is no clearly defined decision boundary and the classifier will have trouble assigning the right label to all samples that fall within this gray area. Figure 14b shows that there possibly exists such an area between the Pursuit and Preview classes, due to very limited differences between the u signals and areas of overlap for the e signals. While the presence of these types of errors in the current classifier seems limited, it might increase for for data from tracking tasks with other task variables, as will be discussed shortly.
4) *Data Mislabeling*: A sample is fed into the classifier with the 'wrong' label, such that even a perfect classifier will always give a faulty prediction. Here, the similarities between the Compensatory and Pursuit classes from Figure 14a seem to suggest that indeed some C labeled samples fit the PS profile better, and vice versa. This could indicate that the HO switches between control strategies, as was shown in Figure 1. Note that 'wrong' labeling in this case is not an implication that any mistakes were made, but rather an unavoidable consequence of the fact that labeling based on active display type is the only option currently available (see Section III-A).

Clearly, the different reasons for errors described here have major implications for how any results should be interpreted. If all errors, for example, are of type 1, it is safe to conclude that the classifier is not designed/tuned correctly. However, if type 2 to 4 are the dominant drivers behind a limited accuracy, than any further optimization of a classifier or its hyperparameters is inane. In this study, both the observed performance limit of the classifier and the validation results from Section IV-C suggest that type 1 errors hardly occur when using only the vdEl3 data set. This is a promising result as it indicates that all remaining errors are mostly due to the data. Consequently, any further research into a better classifier would be futile. Instead, it would be better to focus on getting a new perspective on the data and labeling.

To illustrate this, consider that an ideal classifier (for the purpose described in this paper) *only* produces type 4 errors, because then any such error would undeniable signify a switch in control strategy. One possible next step to achieve this ambition is to attempt to cluster the tracking data to determine whether there are actually only three classes, or if there are perhaps more. If any additional clusters are discovered it could be an indication that there there indeed exist some hybrid control strategies that fit neither of the current labels, therefore making it easier to identify both type 2 (i.e., sample belongs to none of the clusters) and type 3 (i.e., sample belongs to one of the hybrid clusters) errors. Such samples could then either be labeled accordingly or left out entirely to improve the purity of the data.

Another possible solution to the problem of feeding the classifier with 'wrong' labels, is to use simulated data for training. Simulations are guaranteed to apply a constant control strategy, thereby making mislabeling impossible and improving the classifier's knowledge on what a typical C/PS/PR control strategy looks like, i.e., it creates a definitive ground truth. Unfortunately, simulating human-like tracking behavior is still a challenge due to the large amount of non-linearities collected in the remnant n in each of the current HOMs (see Section II-C). As a result, classifiers trained on simulations currently seem not to generalize well to samples from experimental data. New methods should therefore be explored that can either augment current HOMs with a realistic remnant, or that can generate entire tracking runs from scratch. For both these options ML could again offer a solution. In case of the former, one could train a neural network to learn what a remnant looks like by feeding it with simulated tracking runs as input data and actual recorded experimental runs as the target. This way, the network would learn how to construct experimental-like data that includes a form of a remnant from simulations. The latter option of generating data from scratch could be attempted by using Generative Adverserial Networks [34]. These types of neural networks are trained to generate completely new sets of samples based on previously seen data. Clearly, however, both these methods should be explored further before any conclusions can be drawn.

Another consideration is how to make the classifier more useful in practical applications. As shown in Figure 13, the classifier is currently susceptible to very sudden and short (0.1 s) fluctuations between predictions. It could therefore be valuable to extend the current model with methods that can filter out such volatile prediction behavior. For example, the prediction probabilities for each consecutive sample could be linked through some form of a Markov chain [35], in which the probability of sample n_i is directly influenced by the probability of sample n_{i-1}. In practice, this could mean that samples for which the prediction is different than for the previous one and for which the prediction probability is low (e.g., an outlier), the Markov chain would 'overrule' the classifier and assign the outlier to the previously predicted class. Clearly, a potential risk is that some actual, brief switches of control strategy might go unnoticed, but additional research will have to determine whether this risk outweighs the benefits.

Lastly, it should be noted that throughout this research all data used have been of experiments with DI CE dynamics and 2 s look-ahead times for the preview display. It is necessary to investigate whether the classifier performs equally well for tracking tasks with other task variable settings for which the distinction is thought to be less pronounced, such as SI CE dynamics and shorter preview times. The expectation is that the accuracy would drop, mainly due to a significant increase in type 3 errors. Research has shown that compensatory and pursuit control behavior tend to be more alike in tasks with SI dynamics [15]. Similarly, due to the fact that a pursuit display is essentially a preview display with $\tau_p = 0$ s, here too the control behaviors will become increasingly cognate for shorter preview times [21]. However, whether this will indeed pose a problem for the classifier remains to be confirmed.

While some of the challenges above might give an unfavorable impression of the applicability of the classifier, it must be stressed that the results are overall very promising. The 95.4% average accuracy currently attained, already provides a useful tool for detecting anomalous control behavior in DI tracking tasks, since any error made seems to be of type 2-4. So far, no methods existed until now that could do something similar and for all of the challenges a potential solution is suggested, which should open the door for new research opportunities.

VI. Conclusion

The goal of this study was to determine to what extent ML classifiers can be used to distinguish between varying HO control strategies in response to three different types of displays in manual control tracking tasks, using short (1.5 s) samples of time-series data. The average attained accuracy of approximately 95% exceeds expectations and confirms that the InceptionTime CNN can find patterns to distinguish between the three defined classes, which up until now was impossible with traditional cybernetics methods.

Results show that the best combination of signals to use in this specific task consists of e, $\dot{e}$, u, $\dot{u}$, x and $\dot{x}$, which can lead to a classification accuracy of approximately 95% using a single experimental data set and 93% when using several. Additionally, while tuning of InceptionTime's hyperparameters does not provide any boost in classification accuracy, it does reduce the training time by 45% and is thus strongly recommended. In contrast, alternative two-class classifiers show no improvement when compared to a simpler three-class one and no alternative configurations are therefore deemed necessary.

A more detailed inspection of one experimental data set, shows that the classifier accuracy can range between 80% and 99% depending on which human operator's tracking runs are used as a test set (and thus excluded from training). This indicates that control strategies differ significantly between different HOs. Furthermore, distributions drawn from the samples in the confusion categories indicate that there exists overlap between compensatory and pursuit control strategies as well as pursuit and preview, with the latter being slightly worse (1,368 vs. 1,736 errors, respectively). This is surprising with respect to traditional cybernetic views and offers opportunity for additional research.

While several challenges still exist, it has been shown that time series classification techniques can provide us with the ability to recognize control strategy variations in manual control tracking tasks in real-time, ultimately revolutionizing the way we develop future automation for truly safe, autonomous systems.

References

[1] M. Mulder, D. M. Pool, D. A. Abbink, E. R. Boer, P. M. T. Zaal, F. M. Drop, K. van der El, and M. M. van Paassen, "Manual Control Cybernetics: State-of-the-Art and Current Trends," *IEEE Transactions on Human-Machine Systems*, vol. 48, pp. 468–485, Oct. 2018.

[2] E. S. Krendel and D. T. McRuer, "A Servomechanisms Approach to Skill Development," *Journal of the Franklin Institute*, vol. 269, no. 1, p. 19, 1960.

[3] D. T. McRuer and H. Jex, "A Review of Quasi-Linear Pilot Models," *IEEE Transactions on Human Factors in Electronics*, vol. HFE-8, pp. 231–249, Sept. 1967.

[4] K. van der El, D. M. Pool, H. J. Damveld, M. M. van Paassen, and M. Mulder, "An Empirical Human Controller Model for Preview Tracking Tasks," *IEEE Transactions on Cybernetics*, vol. 46, pp. 2609–2621, Nov. 2016.

[5] P. Zaal and B. Sweet, "Identification of Time-Varying Pilot Control Behavior in Multi-Axis Control Tasks," in *AIAA Modeling and Simulation Technologies Conference*, (Minneapolis, Minnesota), American Institute of Aeronautics and Astronautics, Aug. 2012.

[6] R. Duarte, D. Pool, M. van Paassen, and M. Mulder, “Experimental Scheduling Functions for Global LPV Human Controller Modeling,” *IFAC-PapersOnLine*, vol. 50, pp. 15853–15858, July 2017.

[7] A. Popovici, P. M. T. Zaal, and D. M. Pool, “Dual Extended Kalman Filter for the Identification of Time-Varying Human Manual Control Behavior,” in *Proceedings of the AIAA Modeling and Simulation Technologies Conference, Denver (CO)*, 2017. Issue: AIAA-2017-3666.

[8] A. van Grootheest, D. M. Pool, M. van Paassen, and M. Mulder, “Identification of Time-Varying Manual Control Adaptations with Recursive ARX Models,” in *2018 AIAA Modeling and Simulation Technologies Conference*, (Kissimmee, Florida), American Institute of Aeronautics and Astronautics, Jan. 2018.

[9] J. Rojer, D. M. Pool, M. M. van Paassen, and M. Mulder, “UKF-based Identification of Time-Varying Manual Control Behaviour,” *IFAC-PapersOnLine*, vol. 52, no. 19, pp. 109–114, 2019.

[10] A. P. Ruiz, M. Flynn, J. Large, M. Middlehurst, and A. Bagnall, “The great multivariate time series classification bake off: a review and experimental evaluation of recent algorithmic advances,” *Data Mining and Knowledge Discovery*, vol. 35, pp. 401–449, Mar. 2021.

[11] H. Ismail Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller, “Deep learning for time series classification: a review,” *Data Mining and Knowledge Discovery*, vol. 33, pp. 917–963, July 2019.

[12] R. Zhao, “Deep learning and its applications to machine health monitoring,” *Mechanical Systems and Signal Processing*, vol. 115, pp. 213–237, 2019.

[13] R. Versteeg, “Classifying Human Manual Control Behaviour using LSTM Recurrent Neural Networks,” Master’s thesis, Delft University of Technology, Delft, 2019.

[14] E. C. Poulton, “Perceptual Anticipation in Tracking with Two-Pointer and One-Point Displays,” *British Journal of Psychology. General Section*, vol. 43, pp. 222–229, Aug. 1952.

[15] R. J. Wasicko, D. T. McRuer, and R. E. Magdaleno, “Human Pilot Dynamic Response in Single-loop Systems with Compensatory and Pursuit Displays,” Tech. Rep. AFFDL-TR-66-137, Air Force Flight Dynamics Laboratory, Dec. 1966.

[16] R. A. Hess, “Pursuit Tracking and Higher Levels of Skill Development in the Human Pilot,” *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-11, no. 4, pp. 262–273, 1981.

[17] M. Mulder, “Manual Control with Pursuit Displays: New Insights, New Models, New Issues,” *IFAC PapersOnLine*, p. 7, 2019.

[18] R. W. Allen and H. R. Jex, “An Experimental Investigation of Compensatory and Pursuit Tracking Displays with Rate and Acceleration Control Dynamics and a Disturbance Input,” NASA Contractor Report NASA CR-1082, National Aeronautics and Space Administration, Washington, D.C., 1968.

[19] M. Tomizuka and D. E. Whitney, “The Human Operator in Manual Preview Tracking (an Experiment and Its Modeling Via Optimal Control),” *Journal of Dynamic Systems, Measurement, and Control*, vol. 98, pp. 407–413, Dec. 1976.

[20] K. Ito and M. Ito, “Tracking behavior of human operators in preview control systems,” *Electrical Engineering in Japan*, vol. 95, no. 1, pp. 120–127, 1975.

[21] K. van der El, S. Padmos, D. M. Pool, M. M. van Paassen, and M. Mulder, “Effects of Preview Time in Manual Tracking Tasks,” *IEEE Transactions on Human-Machine Systems*, vol. 48, pp. 486–495, Oct. 2018.

[22] K. van der El, D. M. Pool, M. M. van Paassen, and M. Mulder, “Effects of Target Trajectory Bandwidth on Manual Control Behavior in Pursuit and Preview Tracking,” *IEEE Transactions on Human-Machine Systems*, vol. 50, pp. 68–78, Feb. 2020.

[23] H. G. Stassen, “Internal Representation, Internal Model, Human Performance Model and Mental Workload,” *Automatica*, vol. 26, no. 4, p. 10, 1990.

[24] K. van der El, D. M. Pool, M. M. van Paassen, and M. Mulder, “Effects of Preview on Human Control Behavior in Tracking Tasks With Various Controlled Elements,” *IEEE Transactions on Cybernetics*, vol. 48, pp. 1242–1252, Apr. 2018.

[25] M. C. Vos, D. M. Pool, H. J. Damveld, M. M. van Paassen, and M. Mulder, “Identification of Multimodal Control Behavior in Pursuit Tracking Tasks,” in *IEEE International Conference on Systems, Man, and Cybernetics*, pp. 69–74, 2014.

[26] K. van der El, J. Morais Almeida, D. M. Pool, M. M. van Paassen, and M. Mulder, “The Effects of Motion Feedback in Manual Preview Tracking Tasks,” in *Proceedings of the AIAA Modeling and Simulation Technologies Conference, Denver (CO)*, 2017. Issue: AIAA-2017-3472.

[27] S. Azizi, S. Bayat, P. Yan, A. Tahmasebi, J. T. Kwak, S. Xu, B. Turkbey, P. Choyke, P. Pinto, B. Wood, P. Mousavi, and P. Abolmaesumi, “Deep Recurrent Neural Networks for Prostate Cancer Detection: Analysis of Temporal Enhanced Ultrasound,” *IEEE Transactions on Medical Imaging*, vol. 37, pp. 2695–2703, Dec. 2018.

[28] P. Malhotra, V. TV, L. Vig, P. Agarwal, and G. Shroff, “TimeNet: Pretrained deep recurrent neural network for time series classification,” *arXiv:1706.08838 [cs]*, June 2017. arXiv: 1706.08838.

[29] W. Yu, “Analysis of different RNN autoencoder variants for time series classification and machine prognostics,” *Mechanical Systems and Signal Processing*, vol. Mechanical Systems and Signal Processing, no. 149, p. 16, 2021.

[30] H. Ismail Fawaz, B. Lucas, G. Forestier, C. Pelletier, D. F. Schmidt, J. Weber, G. I. Webb, L. Idoumghar, P.-A. Muller, and F. Petitjean, “InceptionTime: Finding AlexNet for Time Series Classification,” *Data Mining and Knowledge Discovery*, vol. 34, pp. 1936–1962, Nov. 2020. arXiv: 1909.04939.

[31] C. Szegedy, Wei Liu, Yangqing Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going deeper with convolutions,” in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, (Boston, MA, USA), pp. 1–9, IEEE, June 2015.

[32] F. Chollet and others, “Keras,” 2015. Publisher: GitHub.

[33] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, and others, “Scikit-learn: Machine learning in Python,” *Journal of machine learning research*, vol. 12, no. Oct, pp. 2825–2830, 2011.

[34] D. Rajan and J. J. Thiagarajan, “A Generative Modeling Approach to Limited Channel ECG Classification,” in *40th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pp. 2571–2574, 2018.

[35] P. Gagniuc, *Markov Chains: From Theory to Implementation and Experimentation 1st Edition*. Wiley, 1st ed., May 2017.

II
Appendices to Scientific Paper

A

Training Parameter Configuration

In order to ensure that the training configuration of InceptionTime would not limit the classifier performance, an analysis was performed on the settings for the number of epochs and the batch size.

First, a training progression was plotted over 75 epochs with the network hyperparameters set to the defined start values (Table II, Column 2 of the scientific paper). The result is shown in Figure A.1. From the figure, it is clear that the train progression is quite volatile, i.e., there are a lot of fluctuations in the test loss. Consequently, it is quite difficult to determine the overall trend and thus also the point at which the test loss starts increasing. It seems, however, that the minimum test loss occurs around epoch 40. As this number can vary for different training runs, it was decided to set the maximum number of epochs for training at 50 in order to ensure that the minimum test loss is always reached. Note that after every training run, the epoch with the lowest test loss was used for evaluation (e.g., for the trained network shown in the figure, the final weights correspond to those of epoch 41).

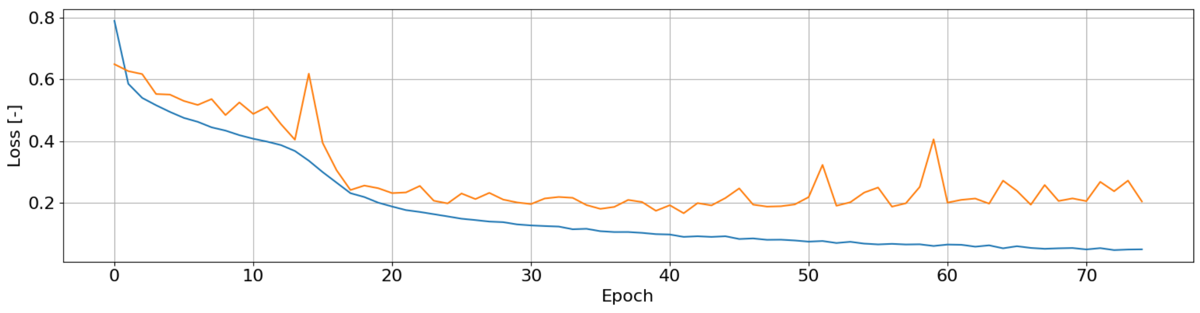


Figure A.1: Training progression of InceptionTime with the start value parameter settings defined in Table II of the scientific article.

Next, a short comparison was performed between three different batch sizes: 32, 64 and 128. The results are shown in Figure A.2. Here it can be seen that a batch size of 64 results in the largest average accuracy (79.9%) while the batch size of 128 results in the lowest training time (1100 s). While the difference in accuracy is minimal, it was decided to continue with a batch size of 64 as it was deemed most likely to provide the best results. However, the results clearly imply that for applications with less computational budget the setting of 128 could be a good alternative.

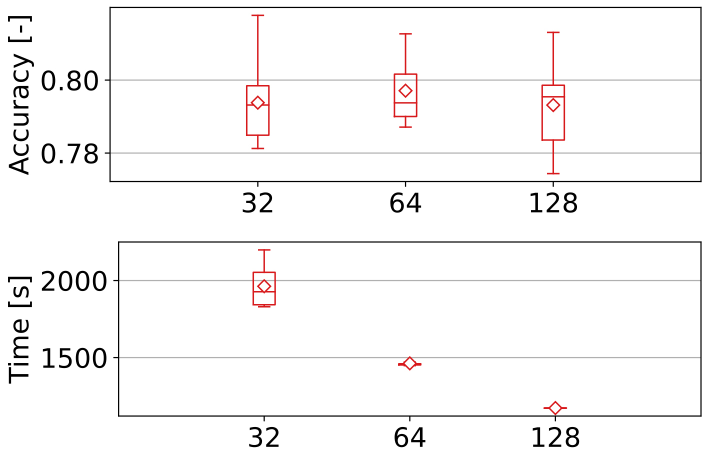


Figure A.2: Box plots of performance metrics for different batch size settings during training.

B Hyperparameter Tuning Results

In this appendix, all of the hyperparameter tuning results are presented, with an exemption of the bottleneck size which can be found in the scientific paper.

Residual Connections

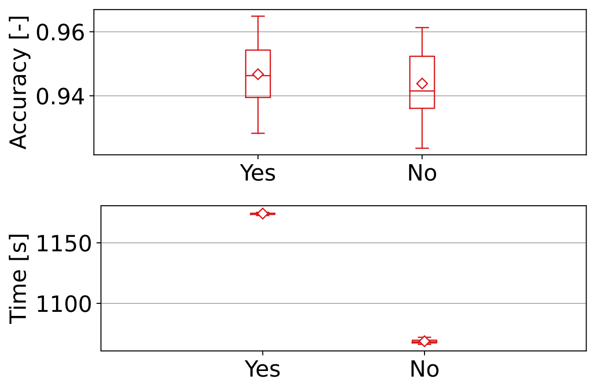


Figure B.1: Box plots of performance metrics for the Residual Connections hyperparameter.

Depth

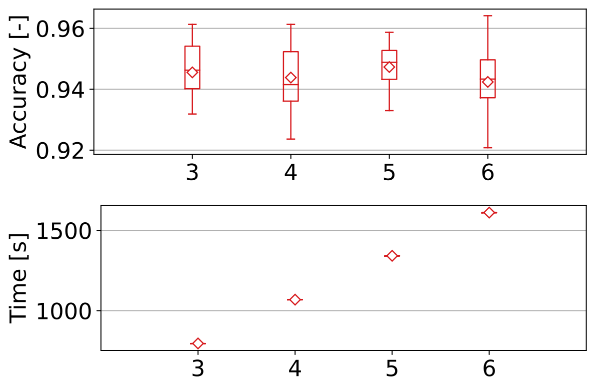


Figure B.2: Box plots of performance metrics for the Depth hyperparameter.

Kernel Size

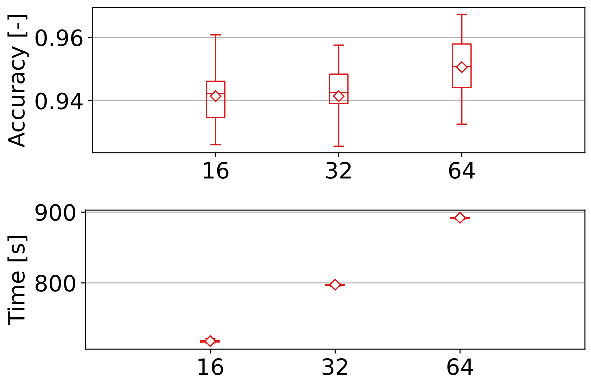


Figure B.3: Box plots of performance metrics for the Kernel Size hyperparameter.

Number of Filters

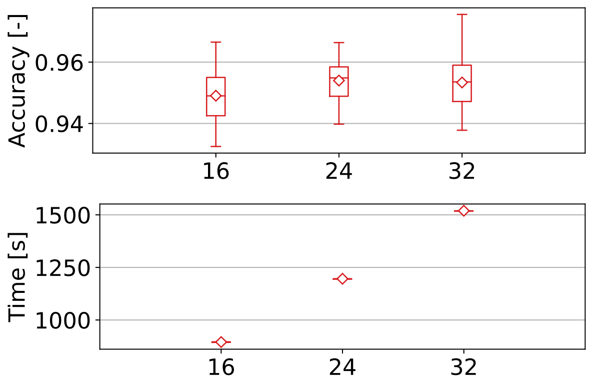


Figure B.4: Box plots of performance metrics for the Number of Filters hyperparameter.

C
Signal distributions

The signal distribution diagrams for the C-PR comparison categories are shown in Figure C.1. Clearly, the low number of samples in the C-PR and PR-C groups result in a highly variable distribution. It is therefore difficult to make any definitive statements on what the cause is of the incorrect predictions. If we consider the low probability values from Table V of the scientific article (Approximately 0.6 for both groups), however, one likely hypothesis is that most samples are outliers (type 2 errors) and therefore simply don't correspond well to either of the classes.

Contrarily, the correctly classified samples in the C-C and PR-PR groups are clearly separated. Both σ_e^2 and σ_u^2 are larger for the Compensatory samples, which is in line with expectation (higher MSE and more control activity).

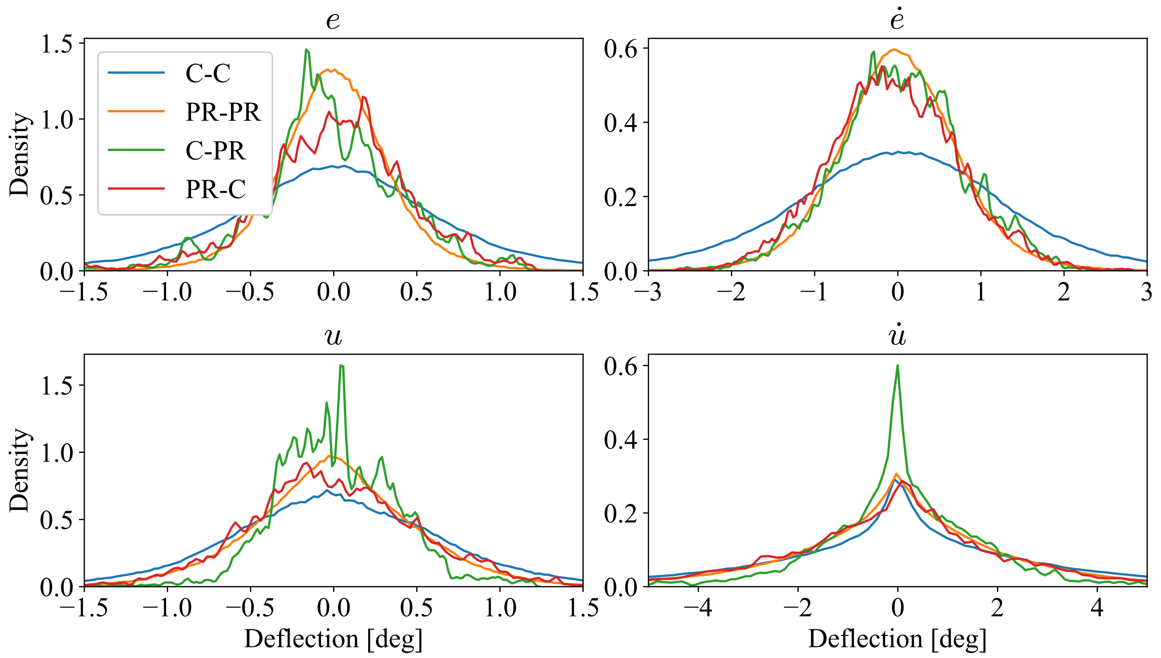


Figure C.1: Signal distributions for samples in the C-PR category.

D
Time Distributions

One hypothesized cause for adaptive human control is fatigue. It could therefore be expected that the classifier identifies more control changes towards the end of the tracking runs, thus resulting in more incorrectly classified samples. To test this hypothesis, a histogram is generated for all incorrectly classified samples of the 30 cross-validation runs for all 9 folds. The total number of errors over these 270 runs is 146,569. The bins correspond to all possible locations for the samples in the tracking run. With the defined data configuration settings (1.5s samples at 50 Hz with 50% overlap), a total of 156 samples are generated from a tracking run of 120 s, resulting in an equivalent 156 bins for the histogram. The result is as follows:

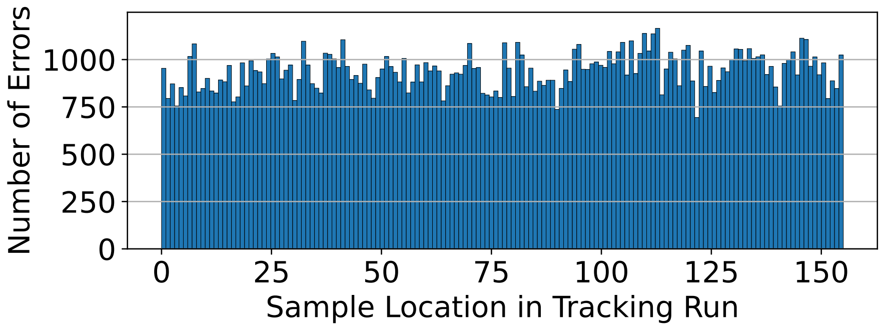


Figure D.1: Histogram drawn from all incorrectly classified samples over 30 repetitions of 9 cross-validation folds.

The figure shows no clear signs that the hypothesis is correct: the number of errors is fairly constant as the tracking runs progress. In order to determine whether this conclusion is also valid for each individual subject, corresponding histograms are generated and shown below. Clearly, it can be concluded that there is no directly apparent pattern here either.

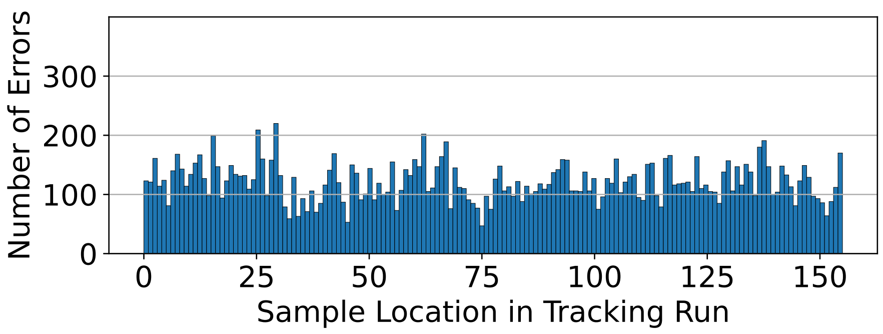


Figure D.2: Histogram drawn from the incorrectly classified samples over 30 repetitions of cross-validation fold 1.

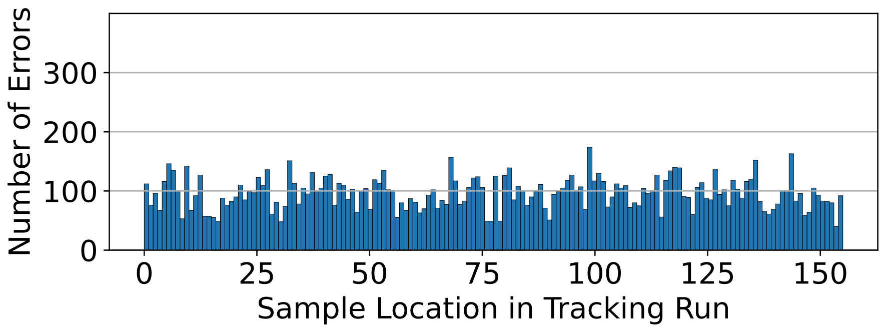


Figure D.3: Histogram drawn from the incorrectly classified samples over 30 repetitions of cross-validation fold 2.

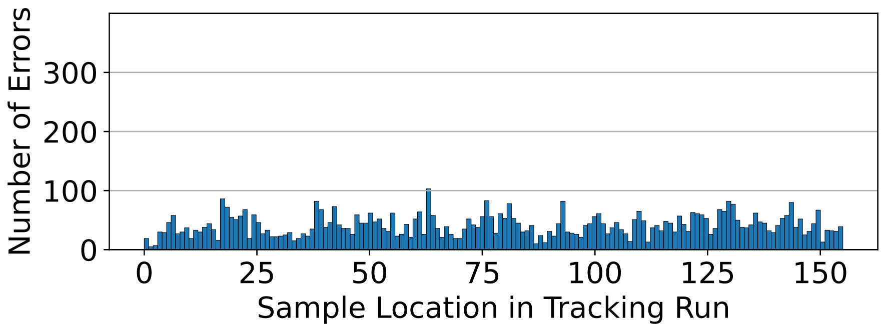


Figure D.4: Histogram drawn from the incorrectly classified samples over 30 repetitions of cross-validation fold 3.

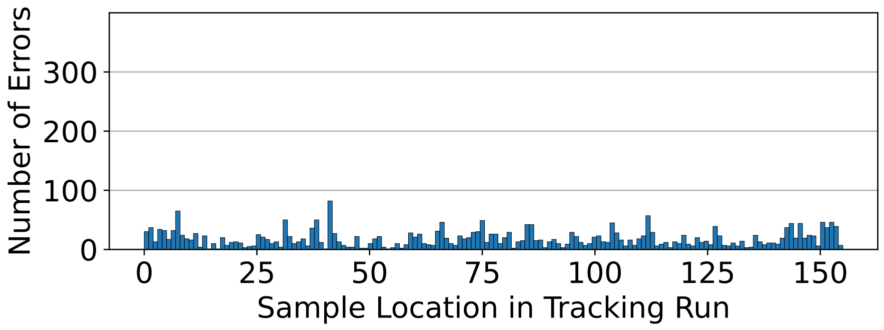


Figure D.5: Histogram drawn from the incorrectly classified samples over 30 repetitions of cross-validation fold 4.

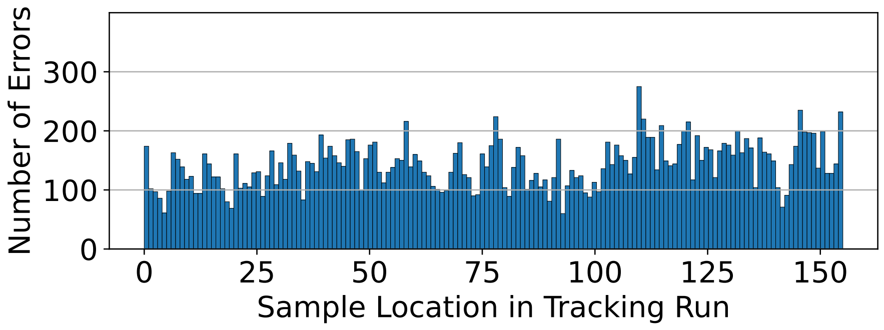


Figure D.6: Histogram drawn from the incorrectly classified samples over 30 repetitions of cross-validation fold 5.

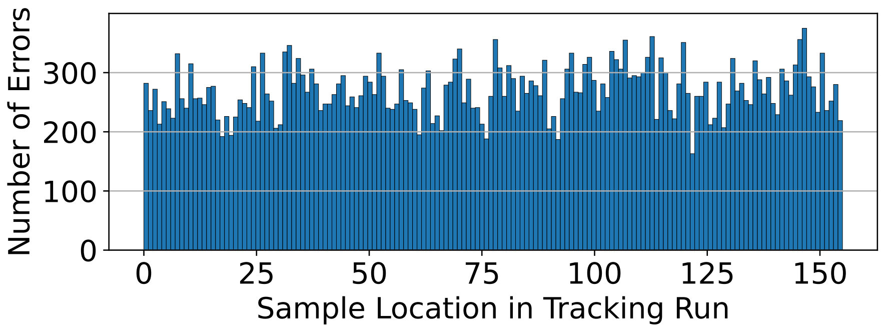


Figure D.7: Histogram drawn from the incorrectly classified samples over 30 repetitions of cross-validation fold 6.

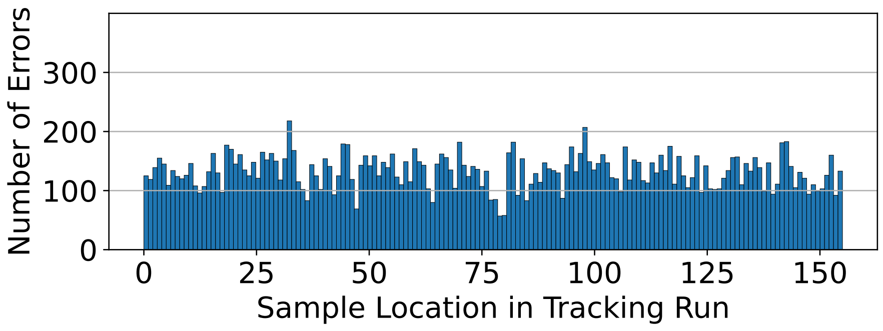


Figure D.8: Histogram drawn from the incorrectly classified samples over 30 repetitions of cross-validation fold 7.

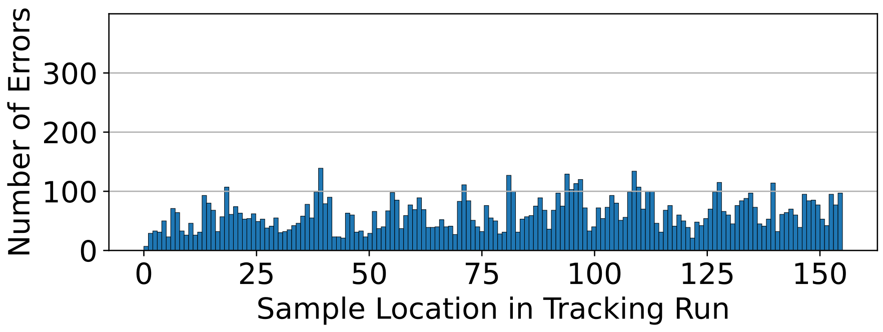


Figure D.9: Histogram drawn from the incorrectly classified samples over 30 repetitions of cross-validation fold 8.

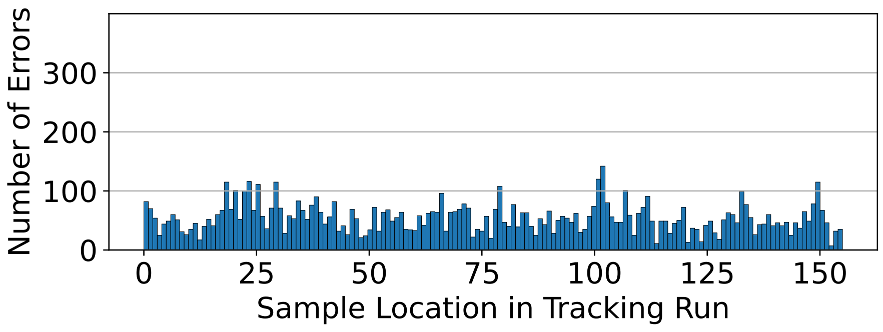


Figure D.10: Histogram drawn from the incorrectly classified samples over 30 repetitions of cross-validation fold 9.

III

Preliminary Report (Already Graded)

1 Introduction

Over the last decades automation has become an integral part of our society. An increasing amount of tasks that were traditionally executed by humans only are now fully dependent on the interaction between human and machine. Examples of this can be seen everywhere but perhaps one of the most topical cases is found in automation-assisted manual control tasks such as driving a car and flying an aircraft. Due to advances in machine intelligence (e.g., self-driving cars), the responsibility of the Human Operator (HO) in such tracking tasks is becoming increasingly supervisory and less direct. However, full autonomy in vehicles is still a long way off and so the transition phase characterised by the interaction between human and machine is of key importance [43]. In order to ensure a gradual and smooth transition, expert knowledge on human control behaviour is essential to improve both the automation itself, and its synergy with humans.

The research area described above, is referred to as 'Manual Control Cybernetics' (MCC). In an extensive literature review on the state-of-the-art of MCC in 2018, Mulder et al. [54] identified five main research objectives that should be overcome to raise the current level of knowledge to one where it can be of use in the optimisation and design of real-world human-machine interfaces. This study aims to contribute to understanding two of these subjects in particular: Human Learning & Adaptation. These aspects of human control are an essential part in what makes us perform so well in a variety of control tasks, and are therefore the key to creating versatile and comprehensible automation. Unfortunately, one of the main limitations in the current state of MCC is that no reliable methods exist that can capture the time-varying nature of HO control behaviour in real-time. Consequently, it is practically impossible to track the changes in control behaviour associated with Learning and Adaption. Therefore, new methods need to be investigated that can offer the ability to track such changes in control behaviour, preferably in real-time.

This research will investigate the capabilities of Machine Learning (ML) in classifying the applied control strategy in human-in-the-loop manual tracking tasks, by using short time-windows of tracking data. ML is a specific form of Artificial Intelligence, that uses self-learning algorithms to generate mathematical models for various applications [51]. Specifically, ML will be used to try to identify the impact that three different display types can have on HO control performance in manual tracking tasks. These display types are known as Compensatory, Pursuit and Preview displays. One difficulty, however, is that the control strategy associated with each display type is not exclusive to that specific display. This poses several problems that shall be addressed throughout the rest of this research. This report in particular, acts as precursor to the main research phase of the author's MSc. thesis.

First, in Chapter 2, an overview shall be given on the current state of MCC and specifically the different display types and their associated control strategies. Next, Chapter 3 will provide a literature review on several Machine Learning concepts and algorithms for Time Series Classification tasks. Then, the information acquired will be used to formulate a research objective and question in Chapter 4. Chapter 5 will present some insights based on several preliminary simulations, followed by a research plan for the main phase of this MSc thesis in Chapter 6. Lastly, this report is concluded in Chapter 7.

2
Manual Control Cybernetics

Research in manual control Cybernetics aims to both develop a better understanding of human control behaviour and create mathematical models that can be used to simulate and predict manual control performance [54]. Here, manual control refers to the process of a human operator (HO) being in direct control of a vehicle or device. One common application of this comes in the form of manual tracking, where a human is continuously trying to minimise a visually registered error between an input and output signal [40].

In this chapter the rudimentary principles of Cybernetics, as well as the most recent developments, will be elaborated upon in order to provide a comprehensive summary of current knowledge and methods. First, the foundations for the manual control model laid out by McRuer and Jex [48] shall be discussed in Section 2.1, followed by an in depth description of different display types and the Successive Organisation of Perception (SOP) in Section 2.2. Next, an overview of the most recent human pilot models is provided in Section 2.3. Section 2.4 will discuss the effects that different display types have on control behaviour and performance. Finally, in Section 2.5, time-varying control behaviour is examined and its relation to the research in question is explained.

2.1 The Closed-loop Manual Control Model

In order to investigate human operator behaviour in manual control tasks, it is important to first establish what the system looks like in its entirety. In 1967 McRuer and Jex [48] published the block diagram shown in Figure 2.1, identifying four different task variables that specifically determine the type of control task: the forcing function (FF), displays, the manipulator and the controlled element (CE). They also recognised that other factors exist that influence the operator's performance; environmental, operator-centred and procedural variables. However, these do not directly impact the task itself and are therefore of lesser importance when establishing what the system is composed of.

Since the task variables are of such significance, they have been the subject of many studies and much is now known on their impact on human control behaviour. In the context of this research it is therefore useful to briefly examine the individual elements. Note that since the display type is the main subject of this research it will be elaborated on in more detail in Section 2.2. First though, to ensure consistency and clarity, a basic nomenclature for the different signals is taken from [54] that differs from those shown in Figure 2.1 (the latter are shown in brackets):

- Forcing functions (i): $f_t(t)$
- Disturbances (-): $f_d(t)$
- Human operator output (c): $u(t)$
- Controlled Element output (m): $x(t)$

- System error ($e = m - i$): $e(t) = f_t(t) - x(t)$

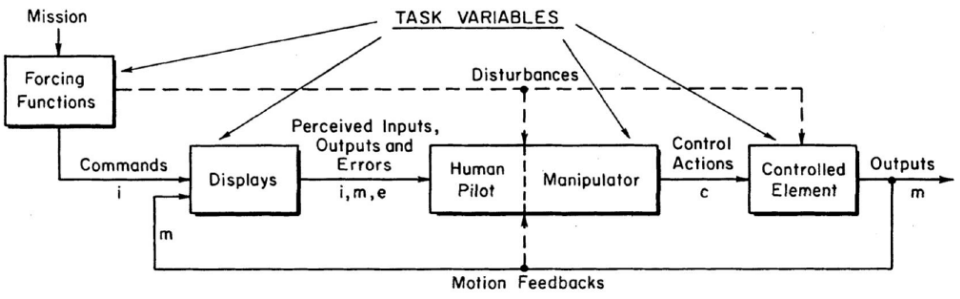


Figure 2.1: Block diagram of the closed-loop manual control task as established by McRuer and Jex [48].

The first task variable is the forcing function (FF). FFs, combined with disturbances, define the target signal that the operator has to follow. In real life situations this could be the shape of a road in a driving task or a 'tunnel-in-the-sky' trajectory while flying an aircraft [52]. Alternatively, in a more controlled test environment with a virtual tracking objective, the forcing functions can be constructed from quasi-random multisine signals [85]. This allows the experiment designer to choose the exact bandwidth and complexity of the signal by using a multitude of sine signals with different amplitudes, frequencies and phase-shifts. By doing so, it has been shown that the bandwidth (ω_i) of the target signal can be a limiting factor in the operators capabilities of accurately following the signal [49]. At a certain frequency the operator will start to ignore the higher frequencies of the signal in order to stay in control of the lower ones.

Skipping the displays (Section 2.2) and human operator (Section 2.3), the next task variable is the manipulator. The manipulator is the object with which the controller directly interacts with the system such as the steering wheel of a car or the yoke/side-stick of an aircraft. While manipulators and their relation to human control behaviour through the interaction with the neuromuscular system are certainly an area of interest in Cybernetic research (e.g., haptic feedback [58]), it is out of the scope of this literature review due to its insignificance with respect to the research objective.

The last element of the closed-loop control model is the Controlled Element ($H_{ce}(s)$). It is defined by the dynamics of the system with which the HO interacts and can often be categorised in one of three main types [48]: Gain (G), $H_{ce} = K_c$, Single Integrator (SI), $H_{ce} = K_c/s$ and Double Integrator (DI), $H_{ce} = K_c/s^2$. Similar to forcing functions, the inherent CE dynamics of real life situations depend on the physical properties of the control task and object, but in supervised experiments it can be set to fit a certain desired system response. This has allowed the CE and its effects on human control behaviour to be the subject of many experiments, leading to clear distinctions between human control strategies for the different types of CE dynamics. More on what these distinctions entail will follow in Section 2.3.

2.2 Display Types and the Successive Organisation of Perception

2.2.1 Display Types

As shown in Figure 2.1, displays are the main medium from which the human operator receives information on the state of the system and the environment, such as inputs, outputs and errors. The term 'display' in this context can refer to many different things. For example, in car driving tasks the main source of information is the outside world as seen from the windshield, while in an aircraft the pilot mainly relies on the Primary Flight Display. In the context of experimental research on manual tracking (e.g., [36, 48, 81]), displays are often electronic devices that provide the HO with continuous information on the tracking task. This is the definition that shall be used throughout the rest of this research. Exactly what information is available depends on the type of display: Compensatory, Pursuit or Preview, examples of which are shown in Figure 2.2.

As shown, Pursuit and Preview displays exhibit information on the target signal $f_t(t)$ and the CE output $x(t)$ from which an error signal $e(t)$ can be inferred, while a Compensatory display only shows the latter with respect to a reference point. The difference between Pursuit and Preview is more subtle. As the name indicates, a Preview display provides the HO with part of the target signal that is to come τ_p seconds ahead, allowing the operator to anticipate on the future state of the target and effectively reduce his or her response time. Pursuit is simply equal to Preview with a look ahead time of $\tau_p = 0$.

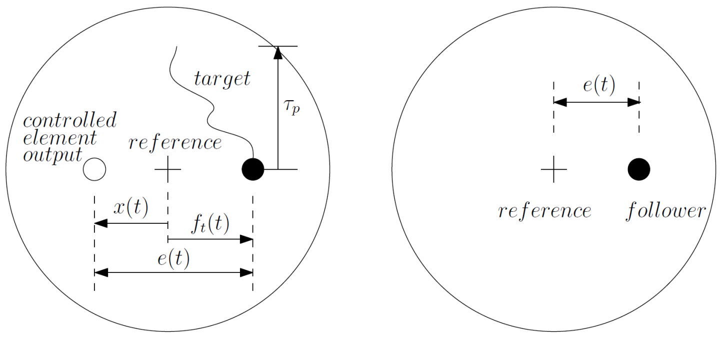


Figure 2.2: Examples of a Pursuit/Preview (left) and Compensatory (right) display in a horizontal tracking task [81].

2.2.2 The Successive Organisation of Perception

As a consequence of the differences between these three display types there are also clear distinctions to be found in human control behaviour in response to each one. These distinctions were first made explicit in the Successive Organisation of Perception (SOP) framework established by Krendel and Mcruer [40]. In this framework, they identified three sequenced control strategies that are based both on the information available to the HO (display type) and the level of skill with which the operator can execute the task. The strategies are described as follows and represented in block diagram form in Figures 2.3 to 2.5 [40, 54]. Note that Section 2.3 provides a more in depth explanation on the significance of the individual elements shown in each block diagram.

1. Compensatory: The HO focuses solely on reducing the error because a) it is the only information available, b) he/she is not skilled enough to adopt better target tracking behaviour given certain circumstances (e.g., stress/distractions etc), or c) adopting a different strategy will not lead to improved tracking performance [40].

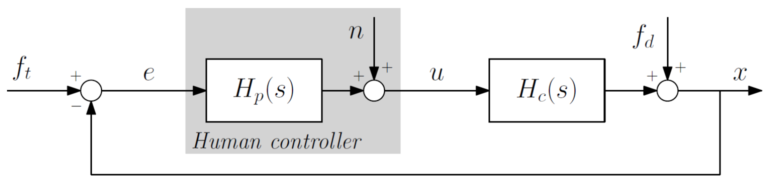


Figure 2.3: Block diagram of a Compensatory tracking task [54].

2. Pursuit: The HO has enough information and skill with respect to the tracking task to make short interval predictions on the target signal, thereby improving the performance (reduced error) with respect to that shown in typical Compensatory control behaviour [40].

3. Precognitive[1]: The HO has near perfect knowledge and skill of the system and task. He/she is therefore able to completely predict the target signal and apply a control strategy that reduces

[1]Note that Precognitive control is left out of scope for this research as it is highly specific and does not occur often in real life situations.

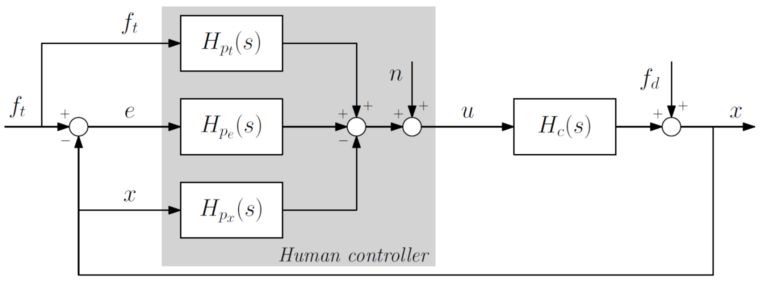


Figure 2.4: Block diagram of a Pursuit tracking task [54].

the error to near zero. This control behaviour can be described as a pure open-loop response since there is no feedback component whatsoever, albeit for a short duration of time [40].

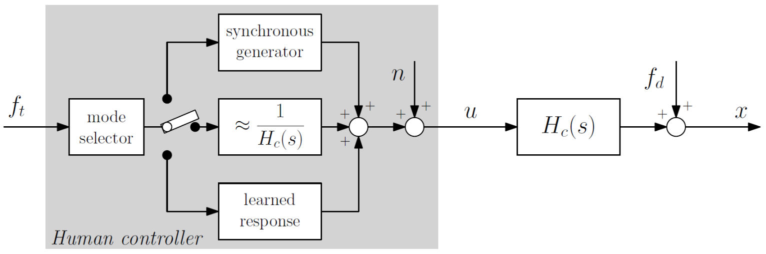


Figure 2.5: Block diagram of a Precognitive tracking task [54].

One of the most important aspects of the SOP is that it acknowledges that an HO can apply different control strategies for any given display type. It might be, for example, that the available display is Compensatory but the applied control strategy is similar to that found in tasks with Pursuit displays and vice versa. This discrepancy between control strategy and display type is an essential aspect of understanding human control behaviour. The most likely explanation is that humans adapt their control strategy with the goal of improving their performance in the ongoing tracking task [40]. Clearly this is something that a non-human model will never do, as it is bound to the mathematical rules that it is given. More on the differences between Compensatory and Pursuit as control strategies and as displays will follow in Section 2.4.

2.3 The Human Operator Model

2.3.1 The Crossover model

Having defined the closed-loop manual control model as well its individual components, it is now possible to zoom in on the final 'unknown' from Figure 2.1; the Human Pilot. While so far all of the control model's building blocks could be explicitly linked to both mathematical and physical phenomena, this is clearly not the case for human behaviour. Humans are incredibly complex and unpredictable, making it difficult to distinguish between the different responses that occur when excited by external triggers, such as during a tracking task [92]. Furthermore, as explained in Section 2.2, HOs adapt their control behaviour as circumstances and their personal level of skill change, leading to a need for time-varying models (Section 2.5).

However, McRuer and Jex [48] established the first time-invariant 'Crossover Model', that could explain and simulate constant human control behaviour in Compensatory tracking tasks. Their limited scope of single-loop human response enabled them to try a Quasi-Linear modelling approach that splits up the HO into two components: a linear, time-invariant element $H_p(s)$ and a remnant n, as shown in Figure 2.3. The former represents the HO dynamics in the form of a linear coupling between the input $e(t)$ and output $u(t)$ like in an equivalent linear system, while the latter makes up all the nonlinearities unaccounted for. What they found, is that humans will always adjust their control behaviour according to several 'Verbal Adjustment Rules'. The first and most important rule is *equalisation*, referring to the automatic adaptation of an HOs control behaviour in order to stabilise the system, by making the entire system dynamics behave as a single integrator around the 'crossover-frequency' ω_c:

$$H_p(j\omega)H_c(j\omega) = \frac{\omega_c}{j\omega}e^{-j\omega\tau_e} \tag{2.1}$$

Here, the term described by $e^{-j\omega\tau_e}$ represents the inherent (effective) time delay that humans experience in order to process and respond to the information provided. After stabilising, the HO continues with adapting using the other Verbal Adjustment Rules in order to minimise the tracking error [48]. Since H_c is defined by the controlled element, all elements the pilot can adjust are represented in H_p. Though several formulations for H_p and its different elements exist, the most comprehensive description is given by [54]:

$$H_p(j\omega) = \underbrace{K_p\left(\frac{T_L j\omega + 1}{T_I j\omega + 1}\right)}_{\text{equalisation}} \underbrace{\left(\frac{T_K j\omega + 1}{T'_K j\omega + 1}\right)}_{\text{low-freq. lag-lead}} \overbrace{e^{-j\omega\tau}}^{\text{delay}} \underbrace{\frac{1}{(T_N j\omega + 1)\left(\left[\frac{j\omega}{\omega_{nm}}\right]^2 + \frac{2\zeta_n j\omega}{\omega_{nm}} + 1\right)}}_{\text{neuromuscular dynamics, } H_{nm}} \tag{2.2}$$

According to the Crossover Model (Eq. (2.1)) which of these elements are 'active' depends entirely on the given system dynamics H_c. Thus, in experiments where H_c is known, we can reason what components we expect to be present. Then, using system identification techniques on tracking task data in the frequency domain, one can identify and isolate the active components and determine their respective parameter values. The details on how this can be done are out of scope for this research, but for reference see [7, 48, 62, 81].

2.3.2 A new model for Pursuit and Preview tracking tasks

For a long time the Crossover Model remained the only validated model of HO control behaviour. Due to its limited application (Compensatory tracking), many attempts were made to extend the model so that it can also be used in Pursuit and Preview situations, but to no avail. Recently, however, a groundbreaking study by Van Der El et al. [81] resulted in a revised and extended model that is capable of filling in the HO dynamics as seen in Figure 2.4.

One of the biggest issues so far had been that the multimodal human system dynamics H_p consists of three separate responses to $f_t(t)$, $e(t)$ and $x(t)$, while there are only two system inputs $f_t(t)$ and $f_d(t)$, leading to an inherently underdetermined problem [87]. Van Der El [81] bypassed this by utilising the fact that all three of these signals are interdependent [$e(t) = f_t(t) - x(t)$], meaning that one can use any combination of the three signals as a tool to determine HO control behaviour [81, 90]. After some trial-and-error, the combination of $f_t(t)$ and $x(t)$ was chosen ('TX'), leading to the following simplification of Figure 2.4[2]:

The next step was to determine what $H_{o_t}^{TX}$ and $H_{o_x}^{TX}$ looked like in terms of their dynamics. They used similar techniques and experiments as applied by McRuer (see [81]), and found that the diagram shown in Figure 2.6 could be restructured again to its final and most intuitive configuration shown in Figure 2.7. From this figure it becomes clear how all information on the display (shown on the left) is processed by a human operator.

First, the preview of the target signal (if present) is monitored at both a near and far 'look-ahead' point; $f_t(t+\tau_n)$ and $f_t(t+\tau_f)$. The far-viewpoint is dominant and the information at this point is filtered by H_{o_f} resulting in an expected target signal $f_{t,f}^{\star}(t)$. The HO then subtracts the CE output signal $x(t)$ to create an internally calculated error $e^{\star}(t)$ that he/she tries to minimise. The transfer functions H_{o_f}

[2]Note that in Figure 2.6 the subscript o for 'Operator' replaces the subscript p for 'Pilot' from Figure 2.4 due to a difference in nomenclature between articles.

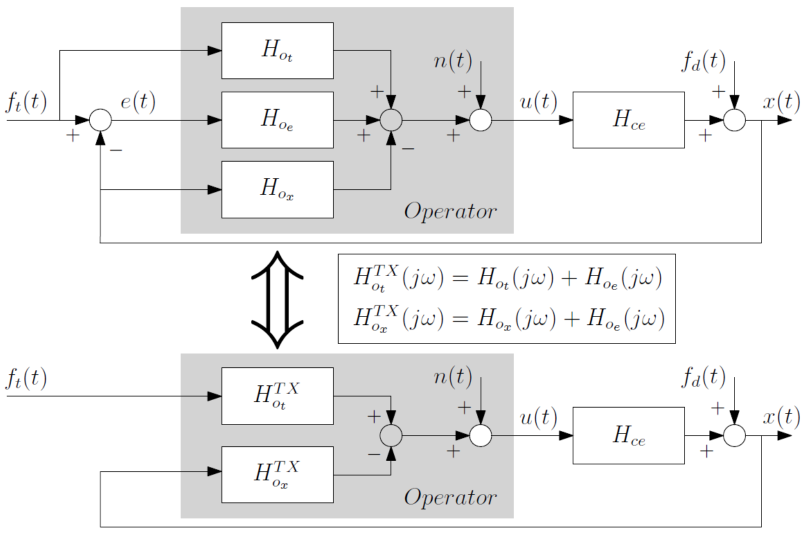

$f_t(t)$
$e(t)$
H_{o_t}
H_{o_e}
H_{o_x}
$n(t)$
$u(t)$
H_{ce}
$f_d(t)$
$x(t)$
Operator
$H^{TX}_{o_t}(j\omega) = H_{o_t}(j\omega) + H_{o_e}(j\omega)$
$H^{TX}_{o_x}(j\omega) = H_{o_x}(j\omega) + H_{o_e}(j\omega)$
$H^{TX}_{o_t}$
$H^{TX}_{o_x}$

Figure 2.6: Block diagram of the simplified two-channel 'TX' HO model in Pursuit and Preview tracking tasks [81].

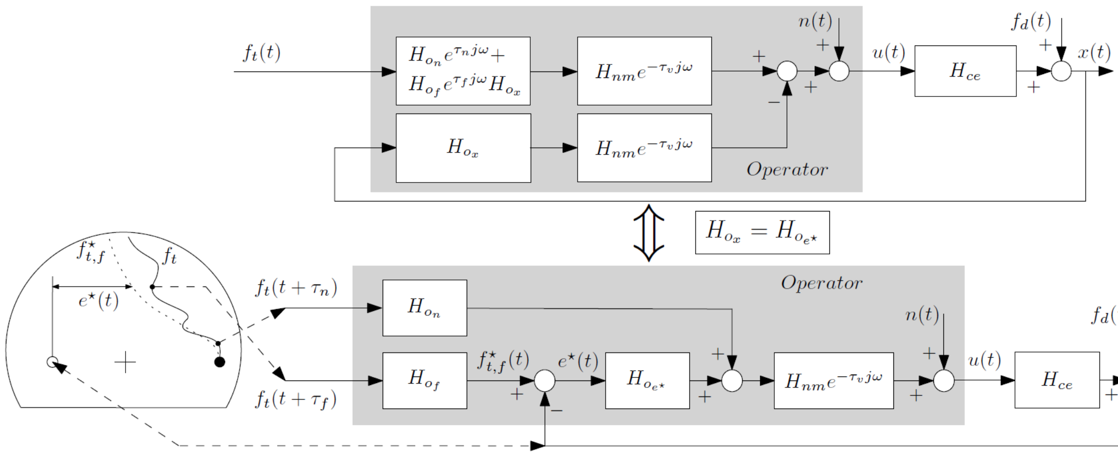


Figure 2.7: Block diagram of the derived HO model in Pursuit and Preview tracking tasks in its most intuitive configuration [81].

and $H_{o_{e^\star}}$ determine how much 'weight' the HO gives to the target signal f_t and the expected error $e^\star(t)$, i.e. what signal is leading the HO response. Similarly, the operator can in some cases monitor the near-viewpoint to provide extra information on more high frequency aspects of the target signal. However, this was witnessed very rarely and thus will be ignored throughout the rest of this research. Clearly, if the maximum preview time (τ_p) is equal to zero, neither the far nor near-viewpoint is present but the diagram is still valid, as will be shown below. Lastly, again there is an element that represents the human limitations in terms of neuromuscular response (H_{nm}) and effective time delay ($e^{-\tau_v j\omega}$).

Like in Eq. (2.2), each of the transfer functions shown in Figure 2.7 can take on certain forms depending on several aspects such as the maximum available preview time τ_p and controlled element dynamics H_{ce}. The goal of the HO (Eq. (2.1)) and the manner in which the HO adapts to these aspects remain similar to the process described by McRuer in his verbal adjustment rules, but more complex and with more variables. Ultimately, the HO 'tunes' the different parameters shown in the equations below [83], in order to optimise the tracking performance and reduce the error.

$$H_{o_{e^\star}}(j\omega) = K_{e^\star}\frac{1 + T_{L,e^\star}j\omega}{1 + T_{l,e^\star}j\omega} \tag{2.3}$$

$$H_{o_f}(j\omega) = K_f \frac{1}{1 + T_{l,f} j\omega} \tag{2.4}$$

2.4 The Effect of Display Type on Control Performance

From the last two sections it is clear that the type of display in a control task plays an important role in the human control strategy during a manual tracking task. The question that remains, however, is: "*how does all of this impact performance*" Many experiments have been conducted to answer that question comparing both Compensatory and Pursuit [2, 14, 64, 72, 90], as well as Pursuit and Preview [36, 80, 83]. An overview shall be provided in this section. First, however, we have to establish what 'performance' is in order to provide a fair comparison between display types. Two main performance measures can be identified:

1. Some function of the error signal $e(t)$, such as the Mean Squared Error (MSE) or Mean Absolute Deviation (MAD)
2. A measure of control activity $u(t)$

The first measure gives an indication on how well the operator is able to follow the target signal, while the second provides insight into how much effort the task takes.

2.4.1 Compensatory vs. Pursuit

To start, a comparison can be made between the Compensatory and Pursuit displays. The latter is different in that the HO can apply an additional Feed Forward response term to the system output. By directly being able to see this output x and it's derivatives ($\dot{x}/\ddot{x}$), the HO can get a feeling for the system dynamics and more easily adapt his/her behaviour accordingly [90]. This is especially beneficial in tasks with 'difficult' controlled element dynamics such as Double Integrators. It should be noted, that subjects in experiments have also been witnessed to apply a Compensatory-like strategy in Pursuit tracking tasks. It has been hypothesised that this can occur for two separate, seemingly contradicting reasons. The first being that the operator simply can not maintain the Feed Forward response due to inexperience/stress, the second that a HO might feel that a Pursuit strategy would not improve performance. In general, however, it is commonly accepted that Pursuit behaviour is more optimal [53].

Moreover, while it is indeed widely recognised that tracking tasks with Pursuit displays show better performance than those with Compensatory, the rate at which they differ varies significantly. For example, Poulton [64] found in 1952 that the error in Pursuit tracking tasks can be almost 50% lower than that in Compensatory (for unknown H_{ce}), while Allen and Jex [2] and Wasicko et al. [90] found only a 12% and ±0% decrease (single integrator CE), respectively. Similar differences can be found for the control activity. Clearly, many more factors are at play in these experiments, such as the experimental setup, the CE dynamics and most importantly the experience of the operator.

Experiments show that experience might indeed be a large factor in how pronounced the difference in control strategies during Compensatory and Pursuit tracking tasks is. It was first hypothesised by Hess [28] that the experience gained in Pursuit tracking tasks can be transferred to Compensatory tasks as well. The result is a larger phase margin (smaller delay) at the lower frequencies of the target signal and thus better control performance. It must be said that in the experiments that lead to this conclusion the same forcing function consisting of 8 sinusoids was used for all runs, both for training and the experiments. It is therefore also possible that the participants learned aspects of tracking task, albeit subconsciously, which allowed them to have a reduced effective time delay in responding to the signal [28]. Furthermore, recent research has identified a possible confound in Pursuit tracking experiments that might contribute to the increased performance as well. More on this will follow in Section 2.4.3

Clearly it is still difficult to draw any definite conclusions on the expected quantitative difference in performance between Compensatory and Pursuit tracking. Especially because it seems like experienced

HOs sometimes mix elements of both strategies: Compensatory tracking can be improved by experience from Pursuit, while sometimes Pursuit behaviour is replaced by Compensatory if it is deemed superfluous. However, now that a more universal model has been created that can approximate 'ideal' HO control behaviour in both situations (Figure 2.7), it might be possible to get a more objective and quantitative comparison between the two. Nonetheless, the conclusion can be drawn that Pursuit displays provide a significant advantage to the HO in most tracking tasks, especially in more difficult tasks such as those with DI controlled element dynamics [54].

2.4.2 Pursuit vs. Preview

Next, a similar comparison can be made between Pursuit ($\tau_p = 0$) and Preview ($\tau_p > 0$) displays. Here, instead of comparing just two situations, most studies have tried to show the effects of an increasing amount of preview time. An overview of some previous results is shown in Figure 2.8 [84]. The grey area marks the cut-off for which control behaviour becomes constant, indicating that any τ_p in this region is ignored (e.g., SI: $\tau_{f,max} = 0.6s$).

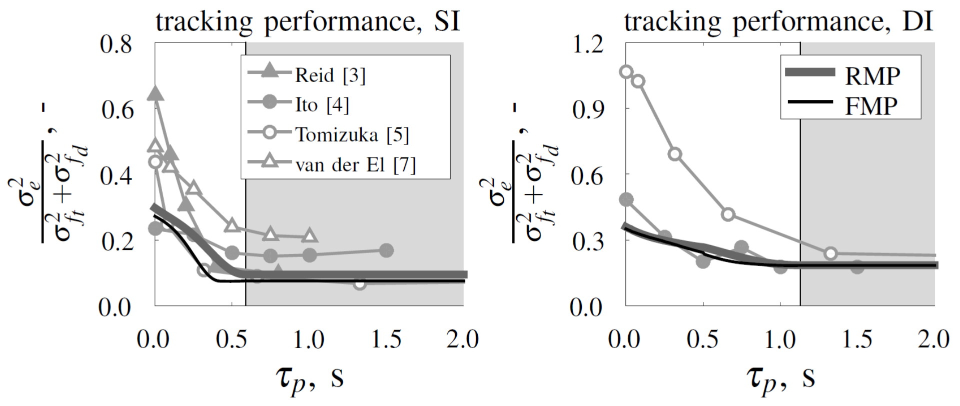


Figure 2.8: Overview of tracking performance results for an increasing preview time τ_p with Single and Double Integrator dynamics [84].

Clearly, performance increases significantly for higher values of τ_p. Again there are some differences between studies but the overall trend is undeniable. The most obvious explanation is that the increased preview time leads to a smaller time delay and a dominant Feed Forward response [84]. Furthermore, when comparing control activity between Pursuit and Preview displays, it was found that there is little difference between the two for subjects that only applied a far-viewpoint response, while the subjects who also used a near-viewpoint ($\tau_n \neq 0$) had to substantially increase their effort [83]. This can be explained by the fact that the near-viewpoint (if present) is thought to be used mostly for high-frequency parts of the target signal, therefore requiring more stick movement.

2.4.3 Possible confound in the perceived impact of display on HO control behaviour

Lastly, it is interesting to mention that recently some possible confounds have been hypothesised by on the difference in control strategy with different types of displays [18, 53]. In many previous experiments with Pursuit (and Preview) displays, the target signal and display were tuned in such a way that the signal will never leave the screen. This means that when the test subject starts to recognise this, either consciously or not, they can use this to their advantage by adopting a 'probabilistic' control strategy. For example, as the target signal $f_t(t)$ moves to the right of the screen with a certain distance from the centre, the operator can predict with a certain corresponding likelihood whether the signal is more likely to keep moving to the right or move back to the centre. The further the signal is away from the centre, the larger the probability that the signal will move back to the centre. This can lead to a predictive form of control that includes corner-cutting and therefore one that is no longer 'pure' Pursuit. Clearly, this is less likely to occur in Compensatory (no visible f_t) or Preview (target prediction already available) tracking tasks. Research by Drop and Mulder et al. has shown that this phenomenon indeed seems to be present in Pursuit tasks, especially at higher bandwidth signals, but a more in depth research is still necessary before any conclusions can be drawn [18, 53].

2.5 Time-varying Control Behaviour

To complete the overview of current knowledge on Cybernetics and relate the summarised information to the research at hand, it is necessary to explore how control behaviour changes over time. So far, the focus of this review has been on time-invariant models, however, the model parameters are likely to change over time as the operator becomes more skilled or adapts his/her control strategy due to changes in the task variables [54]. Evidence for this was discussed in the comparison between Compensatory and Pursuit behaviour in Section 2.4.1. Attempts have been made to develop time-varying models, using for example Maximum Likelihood Estimators (MLE) [38, 94], Kalman Filters (KF) [63], and AutoRegressive with eXternal input (ARX) models [69, 87]. However, none of these attempts have been of satisfying accuracy so far. In order to improve the models, two new areas of interest should be explored [54]. First, we need to understand what it is that drives human adaptation. And second, we need methods that allow us to recognise time-variance in HO control behaviour when it occurs, preferably in real-time.

Considering the first point, the intrinsic driver for time-varying behaviour has been hypothesised by Mulder et al. [54] to be the 'Internal Representation' (IR) of a system that HOs develop as they are exposed to task constraints [74]. The relation of the IR to control behaviour can be split into two separate processes: Learning and Adaptation. Learning refers to the process in which an HO is exposed to a tracking task for the first time and gains more experience from thereon out, subsequently building his/her IR from the ground up. In order to understand more about how this works, experiments need to be conducted in which subjects are closely monitored as they improve from novices to experts during a constant tracking task. Adaptation on the other hand occurs when a change in task variables causes a discrepancy between the subject's IR and observation, thereby forcing the HO to adapt a new control strategy in order to keep the performance constant. Again, more experimental research is necessary that is specifically designed to recognise whether, how fast and to what extent this human adaptation occurs.

The second focus area concerns the recognition of time-varying control behaviour in real-time. The experiments on Learning and Adaptation explained above are of little to no use if there are no methods for recognising the expected changes in control strategy. Therefore, completely new methods need to be developed that do not rely on large time-data sets, unlike the previously utilised parameter identification in the frequency domain [54]. This research will focus on developing such a method using Machine Learning classification algorithms, which is the subject of the next chapter.

2.6 Key Takeaways

This chapter has provided a summary of the most topical knowledge in the field of Cybernetics when it comes to applied control strategies by HO in human-in-the-loop tracking tasks. Some key takeaways are listed below. The ultimate goal of this research is to be a meaningful contribution to solving the issues listed. These points will therefore largely drive the rest of this research going forward.

1. HOs may not always adopt a control strategy that corresponds to the information that is available to them through the display in any tracking task [28, 90]. In other words, sometimes HOs are found to apply a different control strategy than what might be expected based on the display type, in order to increase performance.
   (a) As a result, sometimes there exists a misalignment between the two different definitions of the terms Compensatory and Pursuit. Both can either refer to the SOP, which describes control strategy, and the display type used in a tracking task.
   (b) This discrepancy can either be more or less pronounced depending on the other task variables that describe the tracking task [90]. For example, in tracking tasks with DI CE dynamics the difference between Compensatory and Pursuit control behaviour are more pronounced than in tasks with SI CE dynamics.

2. The benefits of additional preview time are limited to $\tau_p \approx 0.6s$ and $\tau_p \approx 1.1s$ for SI and DI CE dynamics, respectively [84]. After these cutoff times, no increase in performance is witnessed.
3. In order to move forward in understanding HO control behaviour, two main areas of interest in terms of future research can be identified.
   (a) Understanding what drives human adaptation and learning.
   (b) Being able to recognise time-varying control behaviour in real-time.

This research aims to contribute to points 1 and 3b in particular. The end goal is to use Machine Learning algorithms to recognise changes in control strategy by HO in tracking tasks in real-time. More specifically, with the algorithms this research hopes to be able to distinguish between Compensatory, Pursuit and Preview control strategies, such as they are defined in the SOP. A more comprehensive and detailed research objective will be presented after the literature study on Machine Learning Classifiers, in Chapter 4.

3

Machine Learning Classification

In Chapter 2 the main objective of this research is established: to explore and develop new methods that can recognise and classify aspects of human control behaviour in real-time. One research area that has shown immense progression in this field over the last several years is Machine Learning (ML). ML can be seen as a subclass of Artificial Intelligence (AI) and is described as "*the study of computer algorithms that improve automatically through experience and by the use of data*" [51]. Clearly, ML is an umbrella term that captures a wide variety of algorithms for many different tasks such as Classification, Regression, Prediction and many more. This report, however, will focus on the former and more specifically on Time Series Classification (TSC).

Specifically, Section 3.1 and Section 3.2 will provide an introduction to Machine Learning and its role in Time Series Classification. Next, Section 3.3 will explore a popular sequence modelling tool: the Hidden Markov Model (HMM). This is followed by an introduction to Artificial Neural Networks (ANN) in Section 3.4, after which the more complex Convolutional and Recurrent Neural Networks are discussed in Section 3.5 and Section 3.6, respectively. Lastly, the implications of the collected information on the rest of the research are summarised in Section 3.7.

3.1 Introduction to Machine Learning Classification

3.1.1 The Supervised Learning process

Most ML Classification tasks fall under the umbrella of 'Supervised Learning'. This means that the data consist of two main characteristic sets of information that describe each object/sample [25]:

- *Labels y*: a single representative value per object that corresponds to the"class" it belongs to. After training, any newly introduced object (described by features x) will be assigned the label ($\hat{y}$) that corresponds to the class the algorithm 'thinks' the object belongs. Examples of labels could be: tennis ball, golf ball or field-hockey ball.
- *Features x*: a set of n attributes and corresponding values that describe an object. Examples of features that correspond to the labels above could be: colour='yellow', diameter='0.1m', smooth surface yes/no = 'yes', etc.)

In Supervised Learning an algorithm is fed with a large number of training objects (often 80% of all data) consisting of the above mentioned information [25]. The algorithm uses this information to learn how to distinguish between the different classes in a data set. After it is trained, it uses its acquired knowledge to predict what class any newly introduced object belongs to. The prediction is made based on a certain 'decision boundary', which indicates a border in the feature space between two classes. The prediction ($\hat{y}$) can be directly compared to the known label (y) to determine whether an object is correctly classified or not [25]. The algorithm's training objective is to minimise the number of wrongly classified objects in the training set through a loss function $L(y, \hat{y})$. Once the model is done learning it can be 'objectively' evaluated by predicting the classes of samples from a test set (often 20% of all data), thereby avoiding overfitting on the data.

3.1.2 Generative vs. Discriminative algorithms

Exactly how an algorithm tries to define the decision boundary between classes is what sets each Classifier apart, but two general approaches can be distinguished that can be summarised as follows [4, 44, 79]:

- *Generative Classifiers (GC)*: Traditionally, GCs are defined by their reliance on estimating the (joint) probability distributions for each of the classes in the data set, based on the distribution of the features. This way, whenever a new object (with new features) is introduced, the classifier will determine what the likelihood is of that object belonging to either of the classes. The region in the feature space where the likelihoods of two classes intersect are thus equivalent to a decision boundary. An example of this is Linear Discriminant Analysis [79]. 'Generative' refers to the fact that new data can be generated from the established probability distributions. More recently new methods that are able to generate new data without explicitly estimating the probability distribution are also labelled as Generative (e.g., Generative Adversarial Networks [25])

- *Discriminative Classifiers (DC)*: DCs on the other hand do not make any assumptions on the distribution of the data, but simply try to find the most optimal decision boundary. It defines the edges of all the subspaces (classes) in the feature space, allowing the classifier to 'Discriminate' between the classes. When a new object is introduced the classifier registers in what subspace the object lies and assigns the corresponding class label. An example of this is K-Nearest Neighbour [79].

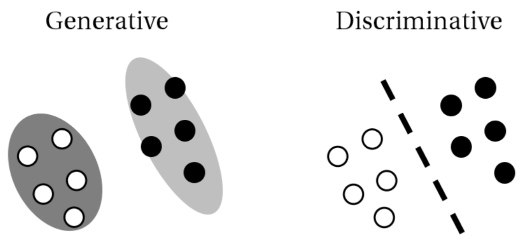


Figure 3.1: Illustration of the differences between Generative and Discriminative Classifiers.

DCs are generally seen as superior for two reasons [34, 56]. Firstly, they require no assumption to be made on the underlying probability distribution and are therefore seen as more flexible. Secondly, Discriminative models generally perform better, especially with larger data sets. Note though, that one must be careful when interpreting a model as Generative or Discriminative as it is often not as clear cut as the definitions suggest. This is reflected especially in Neural Networks, that can be both Generative and Discriminative depending on the architecture and purpose of the model[25]. More on this will follow in Section 3.6.

3.1.3 Evaluation metrics

If a comparison needs to be made between different classifiers or if you simply want to know how well your classifier is performing, clearly some evaluation criteria need to be established. The loss function mentioned in Section 3.1.1 is one measure of performance, but it is often difficult to interpret as it is dependent on several factors, such as the type of loss function and the size and magnitude of the data. Other metrics are therefore desirable that provide more insights into how the classifier is performing[1] [25]:

[1]TP = True Positive, TN = True Negative, FP = False Positive, FN = False Negative.

- *Accuracy*: The number of correctly classified samples w.r.t. the total number of samples.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} = \frac{TP + TN}{P + N} \tag{3.1}$$

- *Precision*: The number of registered True Positive results w.r.t. the total number of registered Positive results.

$$Precision = \frac{TP}{TP + FP} \tag{3.2}$$

- *Recall*: The number of registered True Positive results w.r.t. the total number of possible Positive results.

$$Recall = \frac{TP}{TP + FN} = \frac{TP}{P} \tag{3.3}$$

- *F1-Score*: The harmonic mean of Precision and Recall. The harmonic mean ensures that low values are weighted more strongly, leading to a low F1-Score if either Precision or Recall is low.

$$F1 = \frac{2}{\frac{1}{Precision} + \frac{1}{Recall}} \tag{3.4}$$

- *Area Under the Curve (AUC)*: The area under the Receiver Operating Characteristic (ROC) curve. The ROC curve is a plot of the True Positive Rate (Recall) vs. the True Negative Rate for different 'threshold' values' that indicate how strongly a True Positive result is preferred w.r.t. a True Negative result. An AUC of 1 indicates a perfect classifier, while a completely random binary classifier has an AUC of 0.5.

- *Train time*: The amount of time necessary to train an algorithm to a desired level of performance. This depends on many different things such as the type of classifier and its hyperparameters, the data, the initialisation and more. Clearly, the faster an algorithm is, the better.

All of these metrics (except for train time) can be applied to both the Classifier in general as well as on a class level. The latter can provide more insights into whether some classes are more difficult to distinguish than others.

3.2 Time Series Classification

3.2.1 TSC and its differences to regular ML Classification

So far all the described information in data has come in the form of a set of static features x. However, many real life situations are not characterised by such time-invariant attributes. Often we want to observe how a feature changes over time and use temporal correlation to tell us something about the state/class of a system. Examples of this can be found everywhere, such as in medical applications, machine health monitoring and stock market predictions [44]. Consequently, Time Series Classification (TSC) [10] has become a large sub-category within the Classification research domain. While many of the same principles explained Section 3.1 still apply, there are also some critical differences that impose additional requirements on Time Series Classifiers.

The first and foremost difference that needs to be accounted for is the simple fact that most 'standard' classifiers are not designed to cope with temporal correlation [10]. They look for a decision boundary in the n-dimensional space that is defined by the size of the feature vector x, but have no way of accounting for when that vector starts to move. Secondly, even if one would take all the series data as separate features, in many cases it it would be impossible to implement due to the data being of different lengths [10]. For example, if one would want to recognise the month of the year based on the temperature progression of each month, each month will have a different number of data points corresponding to the number of days in that month.

3.2.2 Types of Time Series Classifiers

To cope with the aforementioned limitations, two main solutions can be identified. First, one can extract (or engineer) a set of characteristic features x from the series data. These can be standard properties of data in the time domain (average, standard deviation, minimum, maximum, etc.), or the frequency domain (mean frequency, spectral skewness, spectral kurtosis, etc.). With these features, one can then design a standard ML Classifier [10, 96]. The disadvantage of this method is that it can be very difficult to find the set of features that provide the best representation of the temporal correlations. Especially when it is not yet known exactly what trends in the series data lead to distinction between classes, it can be near to impossible [34]. Furthermore, many of the possibilities for feature engineering need relatively long time series data to provide any meaningful representation of the signal, especially those in the frequency domain.

A second solution that has won in popularity over the last several years due to rapid improvement in computing power, is the End-to-End model. These are methods that are able to deal with raw time series data as input, and often rely on (Deep) Neural Networks (more on which will follow in Section 3.6) [10, 34]. The advantage of such methods is that they require little knowledge of the research domain to provide meaningful results, as no interpretation of the series data needs to be made. The downsides, on the other hand, are that they require large amounts of data, are computationally expensive and are often difficult to interpret [10].

Clearly, there is always a trade-off between flexibility and interpretability when one has to decide what type of Classifier works best for a specific use case. In this work it is decided to only work with EoE models for two main reasons. First, as explained in Chapter 2, current research in the field of Cybernetics already focuses on utilising frequency and time domain knowledge of the tracking task in order to create interpretable models. These models, however, have been unsuccessful in capturing sort term, time-varying nuances and thus new End-to-End techniques that look beyond these signal features are desirable. Secondly, an inherent limitation of real-time signal analysis is that the sample length can not be longer than one or two seconds, or it is no longer 'real-time'. Since one of the disadvantages of feature engineering is that it requires relatively long time series, it is not a suitable method for real-time/online applications.

By limiting the scope of this research to EoE models for TSC, the number of viable Classifiers for the purpose of this research shrinks significantly. From the remaining options, three main methods have been identified that have proven to be effective in comparable applications. These are explored and qualitatively compared in the coming sections.

3.3 Hidden Markov Model (HMM) Classifier

3.3.1 Theory

An HMM is a model that describes the probabilistic properties of a 'hidden' stochastic process, by witnessing a sequence of 'observed' symbols of another, complementary stochastic process [65]. Such a model is comprised of several standard elements [65]:

- N hidden states: $S_1, \ldots, S_i, \ldots, S_N$
- M individual observation symbols: $V_1, \ldots, V_k, \ldots, V_M$
- T observations O and corresponding states Q: $o_1, \ldots, o_t, \ldots, o_T$ and $q_1, \ldots, q_t, \ldots, q_T$
- Collection A of N^2 transition probabilities between states S_i and S_j: $a_{ij} = P[q_{t+1} = S_j \mid q_t = S_i]$ with $1 \leq i, j \leq N$
- Collection B of $N \times M$ observation (or emission) probabilities of symbols V_k in state S_i: $b_i(k) = P[V_k \mid q_t = S_i]$ with $1 \leq i \leq N$ and $1 \leq k \leq M$
- Collection π of N initial state probabilities: $\pi_i = P[q_1 = S_i]$ with $1 \leq i \leq N$

Furthermore, the original (first order) HMM described above relies on three main assumptions [65, 79]. Note that while these assumptions do not always hold for real world applications, using them has proven to result in sufficiently accurate models [65].

1. *Markov Assumption*: Any state q_t only depends on the previous state q_{t-1}.

$$P\left[q_t \mid q_1, \ldots, q_{t-1}\right] = P\left[q_t \mid q_{t-1}\right] \tag{3.5}$$

2. *Independence Assumption*: Any observation o_i is solely dependent on the state q_i that produced that observation, and no other factors.

$$P\left[o_t \mid q_1, \ldots, q_T, o_1, \ldots, o_T\right] = P\left[o_t \mid q_t\right] \tag{3.6}$$

3. *Stationary Assumption*: Any state transition probability from S_i to S_j is independent of the time step at which it occurs.

$$P\left[q_{t+1} = S_i \mid q_t = S_j\right] = P\left[q_{t+2} = S_i \mid q_{t+1} = S_j\right] \tag{3.7}$$

To make the somewhat abstract description more comprehensible, the following example can be used [66]. Suppose there exist N bowls behind a curtain ('hidden') containing a large number of balls with M different colours. A sequence of T symbols is started by a person drawing a ball from one randomly chosen bowl, who then speaks out the colour of this ball such that it can be recorded from the other side of the curtain ('observable'). The ball is then replaced and the process repeated T amount of times. In this example, the bowls and ball colours exemplify the states S and observations V, respectively. The set of values A represents the probabilities that the person picking the balls switches from bowl S_i to bowl S_j. The collection B on the other hand, gives the probabilities that a certain colour ball V_k is picked from each bowl S_i. Lastly, π contains all probabilities that the first pick of the sequence comes from bowl S_i.

To define a model, one can collect all three sets of parameters under the umbrella term[2] $\lambda = (A, B, \pi)$. Using this set of parameters, it is then possible to generate a sequence of observations [66]. To be able to use the HMM in more useful applications such as Classification, however, Rabiner et al. identified three other problem statements [66]:

1. *Evaluation*: Given a model λ and any corresponding observation sequence O, what is the probability of that specific sequence occurring ($P[O|\lambda]$)?

2. *Decoding*: Given a model λ and any corresponding observation sequence O, what is the state sequence Q that is most likely to have produced O?

3. *Learning*: Given a undetermined model with unknown model parameters $\lambda = (A, B, \pi)$, what are the parameters λ that maximise $P[O|\lambda]$?

The solutions to Problems 1 and 3 are what allows an HMM to be used as a Classifier (see Section 3.3.2), hence these will be briefly explained. The detailed mathematical derivations of these solutions, as well as that of Problem 2, however, are left out of scope for this research and can be found in [66].

The solution to the Evaluation problem is described by what is known as the 'Forward Algorithm' [66]. This algorithm applies an efficient way to sum over all probabilities that represent the hidden state sequences that could have lead to the observed symbol sequence, resulting in one representative probability $P[O \mid \lambda]$. The Learning problem on the other hand can not be solved analytically [79]. It is an iterative process that tries to maximise the likelihood of the parameters λ, given a set of observation sequences that belong to that specific model. This process is called the 'Baum-Welch Algorithm' [6] and it works as follows. First, one initialises the parameters λ either randomly or based on some clustering algorithm and prior knowledge of the problem. With these parameters the probability $P[O \mid \lambda]$ is calculated, or if there is more than one sequence available it is done for all. Next, through a set of mathematical manipulations that characterise the algorithm, one can update the parameters $\hat{\lambda}$ and

[2]This expression is sometimes written as $\lambda = (A, B, \pi, N, M)$ since the number of states and observations are not always known.

recalculate the probability $P[O \mid \hat{\lambda}]$. If the increase in probability is larger than some set target value ϵ the algorithm repeats, else it is terminated. Note that, depending on the initialisation, this might lead to local maxima and one can never be sure if it is the optimal solution. The algorithm is therefore often repeated several times and the best performing one is selected [79].

Lastly, it should be clear that an HMM can not always be classified as an ML algorithm, since there is no learning involved in Problems 1 and 2. It is rather a model describing the interaction between two connected stochastic processes; one hidden and one observable. When, however, the initial parameters of this model are unknown and need to be optimised (Problem 3), the model can learn.

3.3.2 Application in TSC

Having described the theory behind the Hidden Markov Model, it is now possible to apply it to TSC as follows. Starting with a set of labelled Time Series training data (observations) consisting of c different classes, c HMMs are trained. Sometimes, for large time series, raw time series data is first segmented and some latent representation of these segments replaces the original raw time series [39]. The general training process is as described by the Learning principle (Problem 3 Section 3.3.1) [20]. It is important to note though, that while some applications have a clearly defined number of states N and observation symbols M (e.g., gesture recognition [30]), in many cases they are unknown, adaptable parameters. It is then common practice to try several options for both and see what combination provides the best results [39, 55, 61].

After training there exist several theoretically optimised models λ_c that are a probabilistic representation of each class. Then, whenever a new sequence is introduced from for example the test set (O_{new}), all probabilities[3] of that sequence occurring in each of the models are calculated using the Evaluation principle (Problem 1 Section 3.3.1), resulting in c probabilities: $P[O_{new} \mid \lambda_1], \ldots, P[O_{new} \mid \lambda_c]$. Finally, O_{new} is simply assigned to the class corresponding to the model with the largest probability [20].

It should be clear that this method falls under the category of Generative Classifiers discussed in Section 3.1.2 and shown in Figure 3.1. With each model λ one can easily produce new sets of sequences and the decision boundary is determined by the posterior probabilities [79]. Lastly, it can be noted that variations to classical HMM Classifiers exist but for the sake of brevity these were left out of scope of this research (see for example [12, 17]).

3.3.3 Potential for classifying HO control behaviour

Since their first introduction for speech recognition by Rabiner et al. in 1986 [66], HMMs have been used for a wide range of sequence modelling applications. However, as the “No free lunc” theorem states [91], not one Machine Learning algorithm is all-powerful and thus there should always be a trade-off between methods based on their merit.

HMMs are clearly useful for time series modelling. Some of the advantages are that the mathematical basis is well understood and therefore the model is relatively easy to interpret [79]. Furthermore, in cases where there is prior knowledge on the phenomenon that the model is trying to represent it can be easily incorporated through the initialisation. With this one can think of the number of states or some of the transition and emission probabilities. Lastly, they are easier to train, need less data and are computationally less expensive when compared to more brute-force type methods such as Neural Networks (see Section 3.4) [77].

One of the disadvantages is that if there is little or no a priori knowledge of the system in question, the model has to be started with random initialisations. This can result in poor performance or a lot of necessary tweaking of the parameters N and M [11]. Furthermore, their reliance on the Markov Assumption (Eq. (3.5)) can pose limitations on recognising longer time dependencies [11, 59]. And lastly, due to their Generative nature, it is widely accepted that they generally perform worse than more

[3]For unbalanced classes, this is in fact the likelihood, which can then be converted to maximum posterior probability given the prior distributions of the data.

complex algorithms when there are relatively large amounts of data available [4, 34]. For a quantitative comparison between HMM and Neural networks for example, see [77].

Concerning the research at hand, it was therefore decided to not consider the Hidden Markov Model Classifier. The above mentioned advantage of prior knowledge is unfortunately not applicable, and the other advantages do not outweigh the disadvantages.

3.4 Introduction to Neural Networks

3.4.1 Neural Network Architecture

One Machine Learning technique that has become increasingly popular over the last years is the (Artificial) Neural Network (NN). The general principle behind NNs originates from biology, as they were inspired by the functioning of the brain [59]. Figure 3.2 shows the layout of a Feedforward Neural Network (FFN), the most basic network that is characterised by three main components: an input, hidden and output layer. Each of these layers consists of several neurons (or nodes/units) that are connected to all other neurons from the adjacent layers. The number of layers and the number of nodes that define the NN architecture are design parameters that need to be found empirically for each problem since no standard solution exists.

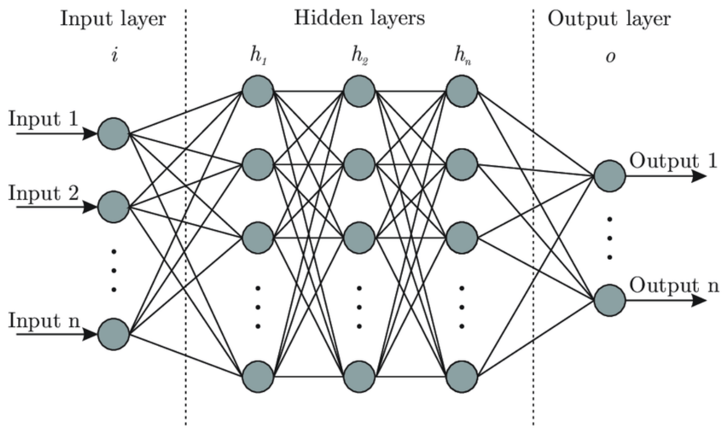


Figure 3.2: General structure of a Feed Forward Neural Network [9].

The neurons in the input layer take an input value x and pass it on to the nodes of the first hidden layer. Here, each incoming value is multiplied by a weight w and summed with a bias term b. The neurons in the hidden layer take this weighted and biased input value and use it as input to their activation function ϕ:

$$\phi(w \cdot x + b) \tag{3.8}$$

The activation function can theoretically be any function but is chosen from one of several popular functions such as the sigmoid, hyperbolic tangent (tanh), or rectified linear unit (ReLU) [59]. Each has its own use, but almost all share the desirable properties of being easily differentiable (see Section 3.4.2) and allowing the network to learn non-linear behaviour [59]. The next hidden layer in the network will take the output of the current one as input and so forth. A network with many hidden layers is called a Deep Neural Network (DNN) and is able simulate highly non-linear behaviour. More intricate DNN exist where the neurons execute more complex calculations than one activation function can provide, such as Convolutional and Recurrent Neural Networks, which will be discussed in Section 3.5 and Section 3.6, respectively. Using DNN for ML applications has become so popular that it has gotten its own name: Deep Learning.

The last layer is the output layer. Its shape is determined by the 'target value' that the network is trying to simulate. For example, in case of a regression task, the output layer will have as many nodes as the number of target data points. Again, an activation function can be chosen, but it is also often left linear [59]. The output of the last layer thus provides an estimate $\hat{y}$ of some target value y resulting in the error $\varepsilon = \hat{y} - y$. For $\hat{y}$ to come anywhere near y, the network first needs to be trained.

In classification tasks, the output layers' activation function is either the sigmoid or softmax function [25]. The former is for binary classification only. It produces any value between 0 and 1. Consequently any output below 0.5 is predicted as belonging to the class with label 0, everything above to the class with label 1. The softmax function is used for multi-class problems. The output is a vector the same size as the number of classes, with for each class the probability that the sample belongs to that class. Logically, the sample is assigned to the class with the highest probability.

Lastly, a specific form of classification is possible with ANN as well: Anomaly Detection [25]. The main idea here, is that instead of trying to create several decision boundaries between classes, there is only one to discriminate between 'Normal' and 'Anomaly'. Training an ANN for anomaly detection can be both supervised and unsupervised learning, depending on the type of data available. Anomaly Detection using time series data is a popular concept, for example in machine health monitoring [23, 45].

3.4.2 Training

To train the network the 'error backpropagation' algorithm is used [59], which works as follows. First, all weights and biases are initialised (pseudo) randomly. Then, the training data is fed through the network in batches as explained above, resulting in an initial target estimate $\hat{y}_1$ and error ε_1. The batch size indicates the number of samples per batch and is one of the hyperparameters that define the training method. Generally a batch size of 32 or 64 is maintained, depending on the size of the training data set [68].

The error can then be used to compute a loss function $L(y, \hat{y})$, which is a single number representation of the performance of the network [59]. Examples of loss functions are the Mean Squared Error (MSE) which is popular for regression, and (Binary) Cross-Entropy for classification [25]. Clearly, the goal is to minimise the error and thus the loss function, which can be done through different forms of Gradient Descent [59]. The only trainable parameters in the model are the weights and biases and so it is with respect to these parameters that we should differentiate the loss function.

By first finding the gradient ∇L w.r.t. $\hat{y}$ and then working backwards through the network we can find the gradient direction for all parameters w and b. Because of this differentiation backwards through the network it is essential that all activation functions are differentiable. Once the gradient is found for all parameters they can be updated according to an update rule, the most basic of which is shown in Eq. (3.9)[4].

$$w_{e+1} = w_e - \eta \frac{\partial L(y, \hat{y})}{\partial w_e} \tag{3.9}$$

In this equation η is the learning rate parameter that determines the size of the steps to be taken; too large and the algorithm may overshoot and diverge, too small and the training can take a long time. Once all weights are updated the process starts again. Such an iteration is called an 'epoch' (e) and can be repeated any number of times to reach the desired level of performance. This number depends on many factors such as the complexity of the task, the size of the data set and the chosen NN architecture. Furthermore, it can be set as a fixed number in advance or one can apply an early stopping criterion such as the desired level of performance.

It is important to note that backpropagation is not guaranteed to converge to a global optimum [59]. Also, since the initialisation of the weights is often random, different training runs can result in different levels of performance. It is therefore always advisable to train a network several times in order to find an average performance as a more reliable indicator.

[4]Note that this update rule is the most basic form of gradient descent. More complex learning algorithms exist, such as Adam, that can greatly reduce training time and improve performance, for a thorough study see [75].

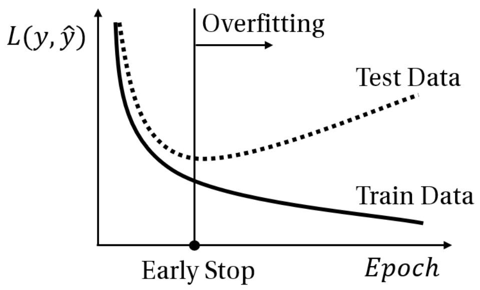


Figure 3.3: Illustration of the concept of overfitting and the recommended Early Stopping point.

Lastly, one of the pitfalls in using Neural Networks is that due to their large number of trainable parameters the risk of overfitting is relatively large compared to other methods [59]. However, several regularization techniques can be applied during training to limit this phenomenon:

- *Early stopping* [59]: As explained in Section 3.1, any model can be objectively evaluated by splitting the data into a training and a test set. As the model trains, the loss of the training data will drop consistently since that is the nature of the Gradient Descent algorithm. However, as shown in Figure 3.3 there will come a point where the loss of the test data (validation loss) will start to rise, indicating the the model is overfitting. By evaluating the model on the test set at the end of each epoch one can monitor this phenomenon and terminate the training process when it occurs, for example after 3-5 consecutive increases of the validation loss.

- *Dropout* [73]: During training, one can choose to 'drop out' a certain number of nodes with each epoch, corresponding to a chosen Dropout rate. For example with a 0.2 Dropout rate, 20% of the nodes are ignored during each training epoch and therefore all the corresponding weights remain constant. This both prevents units from co-adapting too much and reduces training time since less computations are necessary each epoch. Note that Dropout is only active during training, such that all units are activated during evaluation and application of the model.

- *L1 and L2 regularization* [59]: The L1 and L2 regularization methods add a penalty term to the loss functions corresponding to the $\ell 1$ and $\ell 2$ norms of the weights. These penalty terms can help the network to ignore weights it does not use by bringing them to zero and help smooth the output.

3.4.3 Autoencoders

One specific architecture that is interesting to examine before moving on to more complex NN types is the Autoencoder (AE) [25, 59]. This specific type of network is different from regular NN in that it consists of two separate blocks, an Encoder and a Decoder (Figure 3.4), allowing it to perform unsupervised learning tasks. As mentioned in Section 3.1, Autoencoders can both be Generative as well as Discriminative models. An example of a Generative AE model is the Generative Adverserial Network (GAN) but it is out of scope for this research [25]. The way Autoencoders work is as follows. An input vector x is fed through the network front to back, just like in a normal NN. However, at the end of the Encoder block there exists a special hidden layer with a limited number of nodes that signifies a 'latent representation' (or 'embedding') of the input vector [25]. If no latent layer is included, the reconstruction task for the Decoder becomes trivial and the network would learn little. The latent representation is then used as an input for the Decoder, that tries to recreate the original input from the limited amount of information it is given. The resulting output $\hat{y} = \hat{x}$ is then simply compared to the target $y = x$. No separate target data or labels are therefore necessary, hence it being unsupervised learning.

One popular variation to the standard Autoencoder described above is the Denoising AE [25]. The DAE does not necessarily make use of a latent representation to challenge the Decoder which can be desirable as will be explained shortly. Instead, the latent layer is a fully connected hidden layer as shown in Figure 3.2, but the input x is corrupted by artificial noise. This means that the input x and

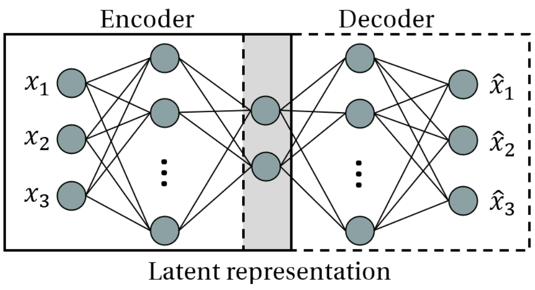


Figure 3.4: Illustration of a basic Autoencoder architecture.

target y are no longer identical, and thus the Decoder can learn to reduce the error ϵ.

There exist various techniques that make use of Autoencoders, but the two most popular applications are pretraining [19] and feature extraction [50]. Pretraining refers to the process of using an unsupervised AE to initialise the weights of a subsequent supervised learning task. In this situation, the Encoder block of the AE has the same shape as the desired network to be used such that after pretraining the decoder can simply be replaced by the desired output layer. It has been shown that this process can both reduce the training time of the supervised training as well as raise the performance [19]. This process falls under the category of semi-supervised learning. While both types of AE described above can be used for this purpose, the DAE is more popular. It does not require any latent layer that needs to be removed and thus most acquired knowledge is retained.

Feature extraction on the other hand, is a form of dimensionality reduction that revolves around the latent representation [88]. The goal here is to utilise the output of the latent layer as a lower dimensional representation of each input sample. For example, in a classification example where each object is described by a feature vector $x \in \mathbb{R}^{100}$, an embedding $x_e \in \mathbb{R}^{10}$ can transform the information stored in x to a lower dimensional space. Naturally there will be a loss of information, but this is not always a problem. An increasingly popular use of this method is to use the embeddings found by an AE as input to a standard classifier such as a Support Vector Machine (SVM) or Decision Tree (DT) [46, 93, 96].

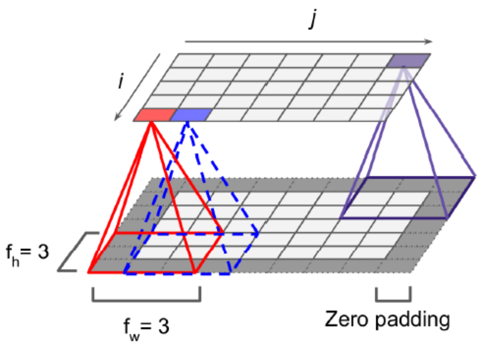
3.5 Convolutional Neural Networks

3.5.1 Theory

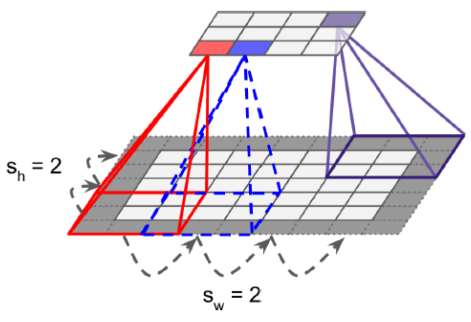
While in theory DNN should be able to simulate any real world system, no matter the complexity, in practice this has proven to be unfeasible. Not only do DNN suffer from the vanishing gradient problem (see Section 3.6.1), but the deeper a network gets the more computing memory it needs [25].

Despite the rapid advancement in computing power over the last years, the search for more efficient architectures that can circumvent these problems is ever growing. Just as with the original ANN, a breakthrough development was achieved by observing the brain and specifically the Visual Cortex [31]. It was found by examining animals that the brain processes visual input in a hierarchical way. The lower level biological neurons focus only on small parts of the images it receives and higher level neurons merge these images into something our brain can comprehend. This all happens at the subconscious level, allowing us to recognise patterns and attach meaning to them. Applying this same theorem to ANN resulted in the theoretical basis [21] for what we now call the Convolutional Neural Network (CNN). Consequently, CNN have become the standard for state-of-the-art image processing, recognition and classification [59]. To understand how they work, we need to introduce two new types of layers: Convolutional and Pooling.

A Convolutional Layer's main difference w.r.t. a normal, fully-connected layer from a regular ANN, is that the nodes are only connected to a selection of nodes from the previous layer. This concept can be



(a) Constant layer dimensions



(b) Reduced layer dimensions

Figure 3.5: Convolutional layers with receptive field $f_h \times f_w = 3 \times 3$ and strides $s_h = 2, s_w = 2$ [25].

seen in Figure 3.5. The top layer shows that each of its nodes $n_{i,j}$ is only connected to a square selection (3x3) of the previous layer called the 'receptive field' [25]. In other words, all the weights between $n_{i,j}$ and the nodes outside of the receptive fields are zero. Consequently, the number of trainable weights between layers is much smaller than for a fully connected network.

Also shown in the figure, is that intermediate layers are often padded by a border of zero valued nodes. This allows the top layer to contain the same number of nodes as the bottom one, while maintaining the desired receptive field. However, as explained above, the real strength of Convolutional layers lies in the fact that subsequent layers reduce in size, such as shown in Figure 3.5b. This allows the network to create high-level mappings of certain regions of interest. To facilitate this reduction in size, the 'stride' of the receptive field (i.e. the step size) between nodes needs to be bigger, such that there is less overlap [25]. Note that the images in Figure 3.5 are drawn in 2D (e.g., grayscale images) for illustrative purposes. However, CNN layers are usually in 3D and can thus also deal with 3D inputs, such as RGB images [25]. The stacked layers in the third dimension are called 'filters'. The ideal number of filters a hyperparameter that needs to be tuned, just like the number of neurons and layers.

When a CNN trains, the weights between each layer adapt in such a way that each layer learns certain 'feature mappings'. An example of this is shown in Figure 3.6 for human faces. Aside from the high levels of performance in image recognition that these feature mappings provide, another benefit is that it can make the resulting model a lot more interpretable than most other types of NN. By reviewing the feature mappings that a trained CNN provides, one can identify what patterns are recognised by the model. This can be of great benefit in certain tasks, as will be discussed in Section 3.5.2.

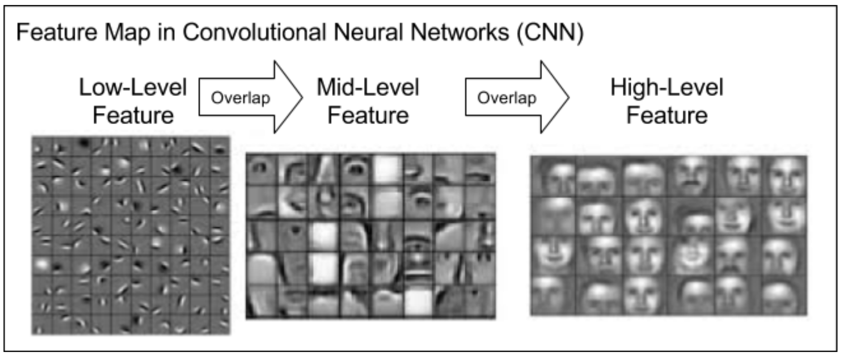


Figure 3.6: Example of feature mappings at different neuron levels in image recognition [42].

The other new layer type, the Pooling Layer, is very similar to the Convolutional layer, as can be seen in Figure 3.7. Nodes in a Pooling Layer are again connected to a limited number of nodes from the previous layer, with a corresponding receptive field and stride. The main difference though, is that the layer's inputs are not mutated by any weights. The only thing it does, is gather and pool information from the previous layer through an aggregation function such as taking the mean or max of the receptive field's neurons. The benefits of adding such a layer are: a reduction in the number of computations, less parameters and therefore less required memory and invariance to translation/rotation of features in the feature mapping [25]. A downside, however, is that it always comes with some loss of information.

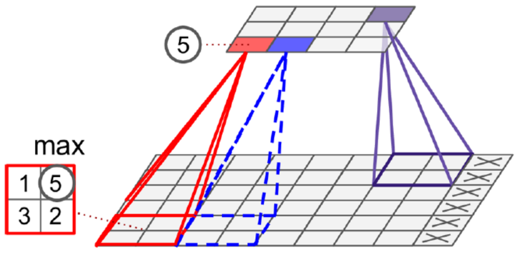
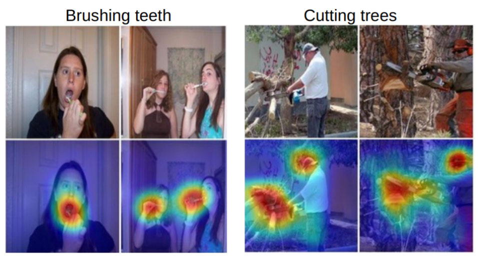


Figure 3.7: Max Pooling Layer with $f_h \times f_w = 2 \times 2$ and $s_h = 2, s_w = 2$ [25].

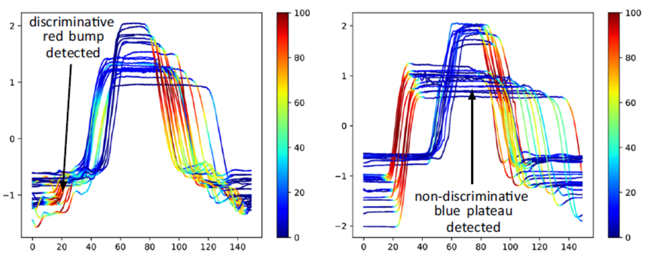
3.5.2 Application in TSC

Wile CNNs primary application has been in image recognition, is has had a rapid surge in popularity for use in TSC since some initial experiments around 2016 [89, 95]. The concepts of Convolution and Pooling described above seem to be just as effective in extracting useful information from Time Series data as from images [95]. In fact, research shows that CNN can be just as effective (if not better) in TSC as Recurrent Neural Networks (RNN) for selected data sets [5]. The current state-of-the-art in CNN for TSC has seen several different variants. Some are adapted from existing network architectures for image processing such as ResNet [27], LeNet [41] and Inception [76], others have been developed specifically for TSC [89, 95]. For a comprehensive overview and comparison between different state-of-the-art CNN classifiers, see [34].

Another benefit of CNN that carries over into TSC is Class Activation Mapping (CAM) [98], which exploits the concept of feature mapping explained in Section 3.5.1. Figure 3.8a shows the original application where a restructuring of the CNN architecture allows the user to get a look into what areas of images are recognised as 'hotspots' by the Convolutional Layers. Figure 3.8b is an application of the same mechanism with the ResNet classifier on the GunPoint data set [67]. The results are a huge advantage in terms of the interpretability of the model, but can also provide significant new insights in the data. In the GunPoint example the time data simply corresponds to motion and so there is not much more information to gain. But in time series data where it is still unclear exactly what characteristics set the classes apart, these types of figures can be very interesting.



(a) CAM applied to images [98]



(b) CAM applied to time series [34]

Figure 3.8: Examples of Class Activation Mapping.

3.5.3 Potential in classifying HO control behaviour

Since the 2019 review of CNN in TSC by Ismail Fawaz et al. [34], CNN have become a leading research topic in the field. Many new architectures are introduced every year that provide slightly better performance than the previous best on selected data sets. As a matter of fact, InceptionTime [35] one of the most recently introduced new architectures is deemed as one of the most promising in a comparison between several state-of-the-art methods by Ruiz et al. [70]. Aside from the increasing levels of performance, the extra possibilities for making the models more interpretable have gotten a lot of attention as well [13].

To summarise, CNN seem to have a lot of advantages when it comes to TSC that can also be applied to the research at hand, but there are several limitations as well. For example, there exist a large number of different types of architectures each with its own advantages and disadvantages, resulting in some performing better on data set X and others on data set Y. In other words, the many options make it difficult to pick one that is most likely to be successful on your data. Furthermore, the architectures themselves are much more complex than for example RNN, with many different types of layers and layer connections. It is therefore far from trivial to implement such a complex network, which is a disadvantage when one is not even sure if it will perform well for their application. Furthermore, the advantages of better interpretability due to CAM are not equally strong for every data set either, as is shown by Ruiz et al. [70]. For example, each sample of the available data in this research (Section 5.2) is unique. This means that even if a heat map such as shown in Figure 3.8b can be created, one needs to do it for every sample. And even then, the result might not give any interpretable information on characterising bits of information that the CNN extracts.

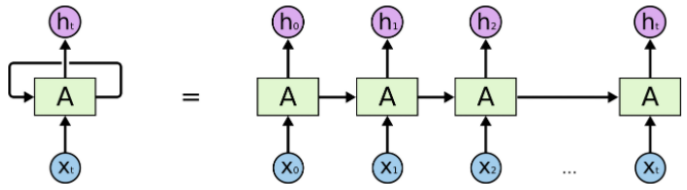
Considering the advantages and disadvantages described, it was decided to first research the control strategy classification capabilities of the easier to implement RNN. If the results are promising, it is recommended to compare the RNN to an off-the-shelve CNN for which the code is directly available, such as InceptionTime, in order to find the optimal type of network for classifying HO control behaviour.

3.6 Recurrent Neural Networks

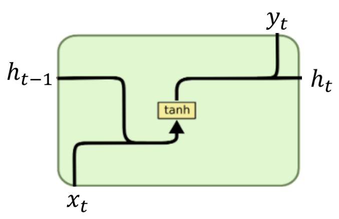
3.6.1 Theory

Just like the Hidden Markov Model, RNN are a ML model specifically designed for the modelling of sequential data such as time series [25]. The general principle, shown on the left in Figure 3.9a, is that a time signal $x_0, x_1, \ldots, x_T$ is fed into the recurrent cell one time step x_i at a time. However, aside from the current time step input x_t, the cell is also presented with what is called the 'hidden state' of the previous time step h_{t-1}. Moreover, the output y_t is always either zero or equal to this hidden state. The former is desired whenever the following layer is not recurrent and can thus not deal with sequential data, such as a regular Feedforward layer. In that case, only the output of the final time step y_T is passed on to the next layer.

As the cell works through a time signal, it uses h as a sort of memory which it can utilise to find temporal dependencies in the data. In order to visualise this process, the RNN can be unrolled to form a DNN through time such as shown on the right side of Figure 3.9a. Looking for example at $t = 2$, it should be clear that to determine output h_2 the cell makes use of both x_2 as well as h_1, which again uses x_1 and h_0 and so forth. Note that when RNN cell A from the figure is unrolled it is still just one cell A. We only visualise the calculations through time by drawing it as a sequence.



(a) Recurrent cell unrolled through time



(b) Inner workings of Vanilla recurrent cell

Figure 3.9: General principle of an RNN [57].

If we take a look inside the cell (Figure 3.9b), we can see that it is not so different from a regular ANN cell with a hyperbolic tangent activation function. The difference is that since the cell receives two different inputs, it can also weigh them separately. Aside from the earlier explained weights W and b, every type of RNN network therefore has an additional set of trainable weights U to balance the effects of the inputs x and h [25]. This results in the following input-output equation for Vanilla RNN cells:

$$h_t = \tanh(W \cdot x_t + U \cdot h_{t-1} + b) \tag{3.10}$$

To train the weights, a similar process is used as described in Section 3.4 for ANN. However, since we need to backpropagate not just through each cell, but through each time step as well, it is called 'backpropagation through time'. During training the weights are updated such that the final weight configuration shows a trade-off between the importance of the input and the memory. However, Vanilla RNN unfortunately are often unable to do so successfully due to what is called the 'vanishing gradient problem' [29]. To understand what this means, let's consider the unrolled network from Figure 3.9a as a DNN with T layers that share the same weights W and U (one cell so one set of weights). In order to backpropagate through all T layers, the gradient should be calculated w.r.t. the output at each time step. For a large number of layers, this means that the influence of the first layers on the gradient is much smaller than that of the last layers. Consider for example a system with one Vanilla RNN cell where the final output is equal to the last hidden state h_T. The weight U can be updated according to Eq. (3.9), with the gradient:

$$\frac{\partial L(y,\hat{y})}{\partial U} = \frac{\partial L(y,\hat{y})}{\partial h_T}\frac{\partial h_T}{\partial U} \tag{3.11}$$

$$h_T = \tanh(W \cdot x_T + U \cdot h_{T-1} + b) \quad \text{with,} \quad h_{T-1} = \tanh(W \cdot x_{T-1} + U \cdot h_{T-2} + b) \quad \text{etc.} \tag{3.12}$$

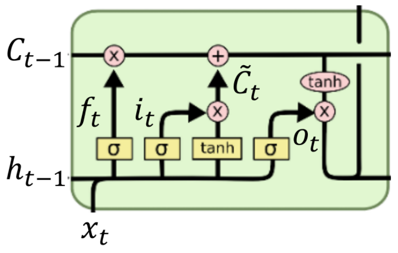
It should be clear from Eq. (3.12) that the effects of h_1 on the gradient become very small for increasing values of T. To combat this, Hochreiter et al. [29] introduced a new type of recurrent cell, the Long Short-Term Memory (LSTM) cell, shown in Figure 3.10a. The main innovation in this structure is the introduction of the 'cell state' (C) and 'gates'. C is an additional memory parameter to complement h. The gates, represented by the σ in the figure, are sigmoid functions that range from 0 to 1. They each control a parameter to determine what is passed on ($\sigma = 1$) and what is deleted ($\sigma = 0$). The following gates can be defined from left to right in the figure with each its own weights [29]:

1. *Forget gate (f)*: Determines how much of the previous Cell state C_{t-1} should be forgotten.
$$f_t = \sigma\left(W_f \cdot x_t + U_f \cdot h_{t-1} + b_f\right), \qquad \in [0,1] \tag{3.13}$$

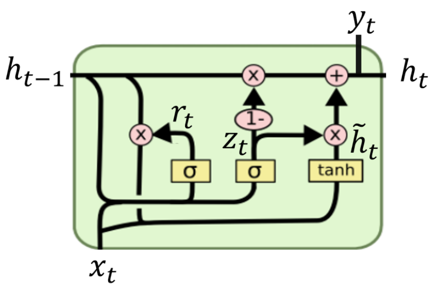
2. *Input gate (i)*: Determines how much of the 'candidate memory' $\tilde{C}_t$ should be added to the existing cell state.
$$i_t = \sigma\left(W_i \cdot x_t + U_i \cdot h_{t-1} + b_i\right), \qquad \in [0,1] \tag{3.14}$$
$$\tilde{C}_t = \tanh(W_{\tilde{C}} \cdot x_t + U_{\tilde{C}} \cdot h_{t-1} + b_{\tilde{C}}), \qquad \in [-1,1] \tag{3.15}$$

3. *Output gate (o)*: Determines how much of the updated cell state C_t should be passed on to the new hidden state h_t.
$$o_t = \sigma\left(W_o \cdot x_t + U_o \cdot h_{t-1} + b_o\right), \qquad \in [0,1] \tag{3.16}$$

The new structure has greatly improved the RNN and has been used in many time series ML applications. Due to the increased number of trainable weights (4× that of vanilla RNN) it is much more advanced. A drawback of having more parameters, is that the training inherently takes longer too. Cho et al. therefore suggested an adaptation of the LSTM which they called the Gated Recurrent Unit (GRU) [15]. The cell, depicted in Figure 3.10b, works in a way that is similar to the LSTM but with some small changes. First of all, the Cell state C is removed and all memory is passed on through the hidden state h. Secondly, some adjustments are made to the remaining gating mechanisms, resulting in the following list of new gates (again, left to right w.r.t. the figure):



(a) LSTM cell



(b) GRU cell

Figure 3.10: Different types of RNN cells (slightly adapted from [57]).

1. *Reset gate (r)*: The Reset gate is a new gate that determines how much from the previous hidden state h_{t-1} should be forgotten before entering it as input for the candidate hidden state $\tilde{h}_t$.

$$r_t = \sigma\left(W_r \cdot x_t + U_r \cdot h_{t-1} + b_r\right), \qquad \in [0,1] \tag{3.17}$$

$$\tilde{h}_t = \tanh(W_{\tilde{h}} \cdot x_t + U_{\tilde{h}} \cdot (r_t \cdot h_{t-1}), \qquad \in [-1,1] \tag{3.18}$$

2. *Update gate (z)*: The previous Input and Forget gates are merged into one single gate. This implies that whenever all memory is kept ($z_t = 0$), all information from the candidate hidden state $\tilde{h}_t$ is left out, and vice versa. Furthermore, the output gate is removed such that the update gate determines in full how much of the candidate hidden state is added to the existing memory in the hidden state h_{t-1}.

$$z_t = \sigma\left(W_z \cdot x_t + U_z \cdot h_{t-1} + b_z\right), \qquad \in [0,1] \tag{3.19}$$

$$h_t = z_t \cdot h_{t-1} + (1 - z_t) \cdot \tilde{h}_t, \qquad \in [-1,1] \tag{3.20}$$

When compared to the LSTM, the GRU clearly has less weights; 12 vs 9, respectively. This means that the GRU often trains slightly faster than the LSTM. In terms of performance, however, the cells score show very similar results despite the significant differences in their architectures [22, 37, 93]. Due to the minimal differences it is recommended to try both [22].

3.6.2 Application in TSC

Since RNN were originally designed to deal with sequential data in general, they have naturally also been a popular choice for TSC applications. In 2003 Hüsken and Stagge [32] were among the first to experiment with this, finding promising results. As most other DNN, however, it has been over the last five to ten years that RNN have become competitive with more traditional Machine Learning algorithms. The increase in computing power and the corresponding drop in training times have made it possible to experiment with different, more complex architectures. Three main methods can be identified from literature:

1. *Basic end-to-end*: A fully connected RNN consisting of a set amount of hidden LSTM/GRU layers and a sigmoid/softmax output layer [3, 71, 86]

2. *Pre-trained end-to-end*: Similar architecture to method 1 but pre-trained by use of a Denoising Autoencoder [24, 33]

3. *Autoencoder + SVM*: An Autoencoder with a latent representation layer is trained on a large amount of non-labelled data. Next, the Decoder is replaced by a Support Vector Machine (SVM) classifier that uses the embeddings of the labelled data as input [23, 46, 93]. For an explanation on SVM classifiers see Appendix A.

The models above are the core architectures, but small variations exist. Yu et al. [93] for example experiment with Bidirectional LSTM and GRU layers. These are layers that factually consist of two separate layers; one that goes through the time series front to end, while the other inverts the sequence. This has shown to increase performance since other nuances might be found [97].

3.6.3 Potential in classifying HO control behaviour

The potential of Method 1 with respect to classification of HO control behaviour is undisputed. In 2019 Versteeg tested the performance of end-to-end RNN to a Controlled Element classification task with varying Hyperparameters [86]. With a maximum measured classification accuracy of 96% on the test set it has shown that it is indeed possible to recognise tracking task variables in real-time (1.6s of data). The other methods, though yet untested seem to outperform Method 1 in several occasions [93, 96]. It is therefore a logical next step to research their performance in the research at hand, with method 1. serving as baseline.

3.7 Key Takeaways

The most important findings of this chapter are summarised below. These findings provide a starting point for choosing which ML algorithms are most likely to provide a well performing classifier that can recognise HO control behaviour in real-time.

1. Literature has shown that End-to-End, discriminative classifiers are most promising for the research at hand, specifically Deep Neural Networks [4, 34].
2. Consequently, the conclusion is made that both CNN and RNN are promising candidates for providing the best results in classification of Human Operator control behaviour in human-in-the-loop tracking tasks.
3. Due to the high complexity of CNN it is decided to first investigate the available data using three different methods of RNN. If satisfactory results are found, a comparison can be made between an off-the-shelve CNN classifier and the best performing RNN. The three chosen RNN methods are:
   (a) A basic end-to-end RNN model.
   (b) A pre-trained end-to-end RNN model.
   (c) An Autoencoder feature extractor + SVM classifier model.

4 Research Objective and Questions

In Chapter 2, the state-of-the-art in manual control Cybernetics has been summarised and some of the key problems in moving forward were identified. The research goals identified in that chapter are supplemented with the knowledge on Machine Learning algorithms gained from Chapter 3 to generate the following research objective:

> *To investigate the possibilities of recognising applied control strategy in a human-in-the-loop tracking task, in real-time, by utilising Machine Learning classification techniques on short windows of labelled tracking data in the time domain, for different combinations of task variables such as Controlled Element dynamics and preview time.*

To break this down, the most important two elements are as follows. First, it should be noted that the goal is to recognise *the applied control strategy* and not the display type. The reason for this, explained in Section 2.4, is that while the type of display is often known in tracking tasks, the control strategy that the HO applies with this display, is not. It would therefore be very valuable if any discrepancies between the two can be identified in real time, instead of only being able to recognise it post-experiment. One major difficulty in finding these discrepancies using supervised Machine Learning, is that at the moment data can only be labelled according to the display type in an experiment. This means that there is an inherent flaw in the data which no ML classifier will be able to solve. This research will therefore also attempt to provide some recommendations on possible experiments that could provide better data for the classification task.

Secondly, it is expected that the accuracy of distinguishing these described discrepancies is larger for certain combinations of other task variables than for others. For example, since the differences in control behaviour for Compensatory and Pursuit displays are more pronounced for DI controlled element dynamics, the accuracy is expected to be larger there.

With the considerations above, and the state-of-the-art of Machine Learning described in Chapter 3, the following main research question can be formulated.

> *To what level of accuracy can state-of-the-art Deep Neural Network classifiers discern between different types of control strategies that are applied by HOs in human-in-the-loop tracking tasks, for various combinations of task variables such as Controlled Element dynamics and preview time, using 0.5 to 2s windows of labelled time-domain tracking data?*

To answer this main question thoroughly, three different sub-topics need to be analysed. Each of these sub-topics can be split up in several sub-questions. The answers to all of the questions combined should provide a satisfying answer to the main research question.

1. Finding the optimal Deep Neural Network classifier.

    (a) What level of accuracy can a Recurrent Neural Network classifier get on the given data set?

   (b) What level of accuracy can a Convolutional Neural Network classifier get on the given data set?
   (c) What qualitative advantages and disadvantages do both types of Neural Networks have?
   (d) How interpretable are the results produced by both types of Neural Networks?
   (e) Considering the results of questions 1(a)-1(d), what type of Deep Neural Network is best suited for classifying HO control behaviour?
   (f) What hyperparameter settings optimise the performance of the chosen classifier?
   (g) What classifying structure works best to discern between the three classes?[1]

2. Interpreting the classifier from sub-topic 1 and finding patterns in the data that can help to better distinguish applied control strategy from display type.
   (a) How well does the classifier perform on simulated tracking data from existing mathematical models of the HO for different control strategies?
   (b) To what extent can the time-domain characteristics that the optimal classifier uses to predict the classes be extracted from the Neural Network?
   (c) What meaningful information can be extracted from an analysis of the correct and incorrect classified samples of both the experimental and simulated data sets?
   (d) To what extent can the data be 'filtered' before using it as input to the classifier, such that any potential samples with a mismatch between display type and control strategy are not used in training?

3. Analysing the effects that different task variables have on the results from sub-topic 2, by using data from experiments and simulations with different task variable configurations.
   (a) What level of accuracy can the optimal classifier get on tracking tasks with varying controlled element dynamics H_{ce}?
   (b) What level of accuracy can the optimal classifier get on tracking tasks with varying preview times τ_p?
   (c) How well do the results of questions 3(a) and 3(b) correspond to current knowledge on the effects that task variables have on HO control strategy?
   (d) How can the results from question 3(b) be used to design new experiments that might render data that is better suited for future ML classification tasks?
   (e) How do the results found in sub-topics 2 and 3 contribute to creating a better understanding of the differences between control strategies for different display types?

To get an understanding of how feasible it is to answer all of these questions, some preliminary simulations and their results will disclosed in Chapter 5. Next, the results will be used to set up a Research Plan for the main phase of this MSc. thesis in Chapter 6.

[1] Since there are 3 classes, one can either train a three-class classifier or experiment with other methods that might improve performance (e.g., Anomaly Detection).

5 Preliminary Simulations

This chapter will provide a report on the implementation and results of several preliminary simulations. First, the scope of the preliminary simulations with respect to the rest of the research is provided in Section 5.1. Next, Section 5.2 gives an overview of the different data sources and how the data is processed for use in the classifiers. Section 5.3 discusses how the models were implemented, followed by the results in Section 5.4. Finally, Section 5.5 explains the Verification and Validation process.

5.1 Scope

While the research objective for the upcoming MSc thesis has been clearly defined in Chapter 4, the scope of the preliminary simulations is more limited. Specifically, the goal of these initial tests is to provide information or new insights that can help structure the main phase of this research. This has resulted in the following three research goals:

1. To get an initial understanding of whether it is in fact possible to recognise HO control strategy based on 1-2s intervals of tracking data.
2. To get a comparison of the performance of the two RNN cell types and the three main identified architectures (Section 3.6), in order to identify if one combination significantly outperforms the others.
3. To get a comparison of the performance of networks trained with the different available data sets, in order to identify any discrepancies in data sets that might be worth examining in more detail.

The first goal is clearly the main objective, since neither the second or the third will have any meaning without it. To this end, it was decided to build the classifier in such a way that the expected difference between classes would be most pronounced. This resulted in the following two choices. First, the data used was limited to data from experiments with Double Integrator CE dynamics and a preview time of $\tau_p = 2s$. According to the literature research (Chapter 2) these configurations accentuate the differences in control strategy the most. Secondly, three separate, binary classifiers were built for each control strategy comparison. Here, the goal was to get the most accurate possible reading of how well each class can be distinguished from the others. It should be noted, that the classification accuracy will therefore most likely provide a more optimistic result than a potential future three-class classifier would. To illustrate: In a binary classifier class A can only be mistaken for class B, while in a three-class classifier there is also class C.

Goal 2. is more straightforward in that it is a matter of comparing the performance metrics of the different combinations of RNN architectures. Since all combinations of cells (2) and architectures (3) will be tested for all binary classifiers (3), the total number of compared classifiers will be $2 \times 3 \times 3 = 18$.

Lastly, since data sets from different experiments are available (Section 5.2), it is interesting to see if any large discrepancies in terms of performance can be found between them. If this is indeed found to be the case, it could be worth examining the data sets more closely to identify the cause.

5.2 Data Sources and Pre-processing

5.2.1 Data sources overview

The data quantity used in the preliminary simulations comes from five different target tracking experiments that were executed in the past. The amount of raw data per source is summarised in Table 5.1. All data sets were extracted from *.mat* files and contain each of the characteristic signals that describe a simulated target tracking task: $f_t(t)$, $f_d(t)$, $u(t)$, $x(t)$ and $e(t)$ (see Section 2.1).

Table 5.1: Total number of seconds of data for each tracked signal (CE = DI, $\tau_p = 2$) per display, per source, given in:
# subjects × # runs per subject × duration of run in seconds

Source	Compensatory	Pursuit	Preview
Roggenkämper [69]	5×5×90 = 2250	5×5×90 = 2250	-
Vos [87]	8×5×90 = 3600	8×5×90 = 3600	-
Van der El 1 [84]	-	8×5×120 = 4800	8×5×120 = 4800
Van der El 2 [82]	8×5×120 = 4800	-	8×5×120 = 4800
Van der El 3 [85]	9×15×120 = 16200	9×15×120 = 16200	9×15×120 = 16200
Total [s]	26850	26850	25850

Note that the data in this table is given in seconds since the sampling frequency for all experiments is the same at $f_s = 100$Hz. Since around 60% of the data is from the Van der El 3 set and it is equally spread over all classes, it was decided to use this set for the supervised training of all 18 classifiers. The data from the other experiments, was used as unlabelled data for the training of the Autoencoders. This way, both AE were 'blind' to the labelled data before the supervised training phase. More on this will follow in Section 5.3.

5.2.2 Data pre-processing

In order to use the data as input to an RNN it first needs to be properly formatted and processed. Versteeg has done an extensive study into what tracking signals and data preparation techniques give the optimal performance in classifying CE dynamics [86]. While it is in no way guaranteed that these resulting settings will again provide optimal performance in this simulation, it is most likely still the best starting point. The data was therefore sampled and processed according to the following settings [86]:

Table 5.2: Chosen data configuration

Setting	Choice
Input signals	e, u, $\dot{e}$
Time window	1.5s
Sampling freq.	50Hz
Window overlap	50%
Scaling method	*Standardising*

Note that in order to find $\dot{e}$, e was differentiated using the Numpy Gradient function that utilises "*second order accurate central differences in the interior points and either first or second order accurate one-sides (forward or backwards) differences at the boundaries.*" [26]. Furthermore, Standardising in the above table refers to the process of scaling the data such that all different samples have roughly the same amplitude. This is done per signal and with respect to data from all runs in each individual experiment using Eq. (5.1) [60]. After standardising the data is split up in samples of 1.5s at 50Hz, resulting in 75×3 data points.

$$x'(t) = \frac{x(t) - \mu_x}{\sigma_x} \tag{5.1}$$

Window overlap (Figure 5.1) mainly means that more samples can be won from the same amount of data, improving the chances of capturing class distinguishing temporal relations in the total set of samples. The total number of samples N per tracking run as a function of overlap o can be calculated with Eq. (5.2). In this equation, $\lfloor\cdot\rfloor$ and $\lceil\cdot\rceil$ indicate the 'floor' and 'ceiling' functions to respectively round down and up to the nearest integer, while ℓ and L represent the lengths of each sample and the total run.

$$N(o) = \left\lfloor \frac{L - \ell}{\lceil \ell(1 - o) \rceil} \right\rfloor + 1 \tag{5.2}$$

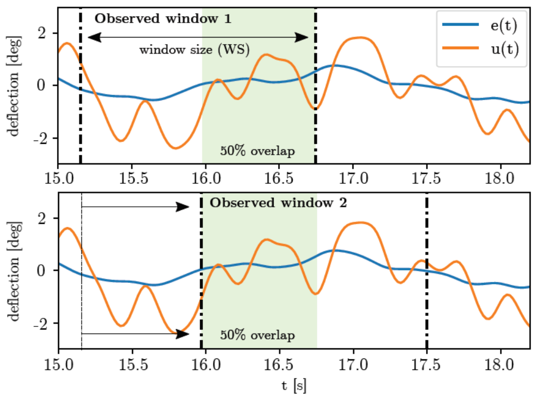


Figure 5.1: Overlapping window principle for sampling of time data [86].

The result of the above operations is a set of N input samples x per tracking run. The data is then labelled in binary fashion, according to the display type that was used in the experiments from Table 5.1 (e.g., $y_{Comp} = 0$, $y_{Purs} = 1$). One essential thing to notice here, is that the samples can thus only be labelled according to the *display type* and not according to the actual *applied control strategy*. This distinction is very important and it is explained in detail in Chapter 2. What this means is that there is an inherent flaw in the data that the Neural Network will never be able to overcome. A label might suggest that the sample it belongs to is data from a Pursuit control strategy since that was the display type active during the experiment in which the data was collected. However, it is known that with Pursuit displays, the HO sometimes applies a Compensatory control strategy. This concept is of key importance when interpreting the results, and shall therefore be revisited in Section 5.4.

5.3 Model setup

All data processing and model implementation is done in Python 3.8. The following packages were used that offer a wide range of Machine Learning and Deep Learning tools: Scikit-learn [60], Tensorflow [1] and Keras [16]. All results are generated through the following procedure:

1. Before starting the main training phase, ensure that the Autoencoders have trained and store their weights to be used by Model 2 and 3.
2. Load in the labelled data from the two classes (displays) in question according to Section 5.2.
3. Split the data randomly according to the training ratio.
4. Train the network on the training data until either the maximum number of epochs is reached, or the validation accuracy has risen e number of times (early stopping).
5. Load the weights of the best performing epoch and evaluate the network with the test data. Store the following performance metrics: Accuracy, Precision, Recall, F1-Score, and Training Time.

6. Repeat the process r number of times.

7. When the final repetition is finished, use the r number of performance metrics to draw box plots and calculate the mean and standard deviation.

The reason for repeating the process r times is that there are several stochastic elements in the training that result in different outcomes. These stochastic elements are both present in the random splitting of the data for training and testing, as well as the training of the Neural Networks itself. For example, the weight initialisation of the networks, as well as the re-shuffling of the data during each epoch. This can have a large impact on the results and it is therefore advisable to average out these inconsistencies. It was decided to set $r = 30$ for comparing the different models and $r = 10$ for comparing the data sources. The reason for this is that the training time for 30 runs is significant (24h+) and the comparison between methods was deemed more critical than the comparison between data sets. The former results in a choice of how to move forward in this research, while the latter is meant to provide more insights in subtleties in the data.

Furthermore, just as with the configuration of the data, it was decided to copy the hyperparameter settings from Versteeg [86] where possible and complement this with recommendations from literature [68]. The following sections will provide a complete overview of each RNN architecture and it's hyperparameters. Note that some short tests were done with the Bidirectional GRU/LSTM layers mentioned in Section 3.6.2. However, these additional layers significantly increased training time and showed no increase in performance. It was therefore decided to leave them out of the main comparison.

5.3.1 Model 1: Full end-to-end RNN

The first model is essentially a copy of the best performing architecture found by Versteeg. An illustration is shown in Figure 5.2. A full list of hyperparameters describing both the network structure and the training settings can be found in Table 5.3. As stated before, all models are trained with both the GRU and LSTM cell types as well as for three different binary classifiers: C vs PS, C vs PR and PS vs PR.

Table 5.3: Model 1 hyperparameters

Hyperparameter	Choice	Hyperparameter	Choice
# Hidden layers	2	Training algorithm	Adam [75]
# Nodes per layer	100	Loss function	Binary Cross Entropy [25]
Train ratio	0.8	Batch size	32
# Train samples	33696	# Epochs	Max. 50
# Test samples	8424	Early stopping	$3\times$ val. loss $\uparrow$
Data source	Van Der El 3	Dropout ratio	0.2

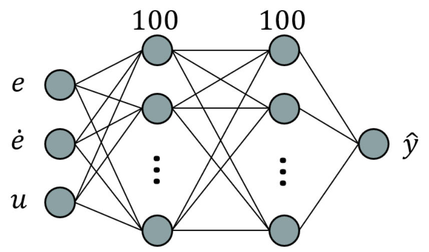


Figure 5.2: Model 1: Full end-to-end RNN model architecture.

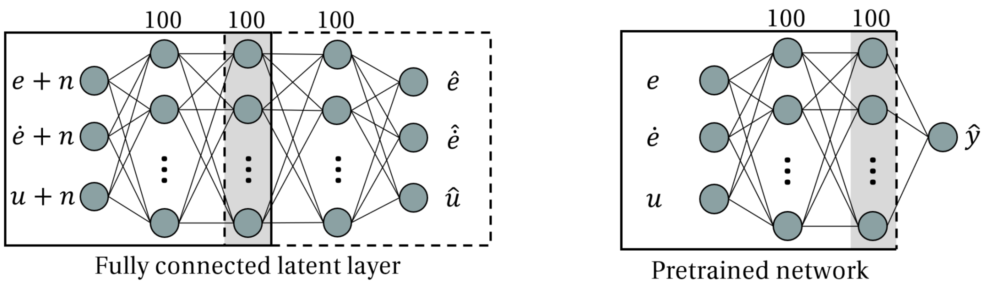
5.3.2 Model 2: Pretrained end-to-end RNN

Model 2 is very similar to the first model, except that the weights are initialised by first training a Denoising Autoencoder, as shown in Figure 5.3a. The Encoder and Decoder share one common, fully-connected layer, which is replaced by a single cell with sigmoid activation function to transform the network into a classifier. It should be noted that once the pretraining is done and the Decoder is removed, the network is identical to Model 1 with the exception of the initialisation of the weights. All the hyperparameters of the supervised training phase are therefore the same as the ones shown in Table 5.3. The Autoencoder does in fact have some slightly different settings during its unsupervised training phase, which are shown in Table 5.4. The biggest difference lies in the data and the training parameters. The Autoencoders seem to be more stochastic in the validation loss and need more epochs before the stopping criterion is met. Therefore both the maximum number of epochs and the early stopping value e are raised. Furthermore, since the AE is not trained for a classification task but more for what can be considered regression, the Loss function is set to MSE. Lastly, in order to create artificial 'noisy' data, uniform noise with a maximum magnitude of 0.4 is added to the original time samples.

One main difference between Model 1 and 2 is that the latter uses the same exact initialisation every time. Since this is expected to be one of the largest stochastic elements, the results of Model 2 are likely more consistent than those of Model 1. The remaining stochastic processes in the model are the sampling of the data for training and testing, and the order in which the model receives each sample every epoch.

Table 5.4: Model 2 - Autoencoder hyperparameters

Hyperparameter	Choice	Hyperparameter	Choice
# Hidden layers	3	Training algorithm	Adam [75]
# Nodes per layer	100	Loss function	Mean Squared Error [25]
Train ratio	0.8	Batch size	64
# Train samples	32136	# Epochs	Max. 100
# Test samples	8034	Early stopping	5× val. loss ↑
Data sources	Roggenkämper	Dropout ratio	0.2
	Vos		
	Van Der El 1		
	Van Der El 2		



(a) Denoising Autoencoder for pretraining **(b)** Pretrained classifier network

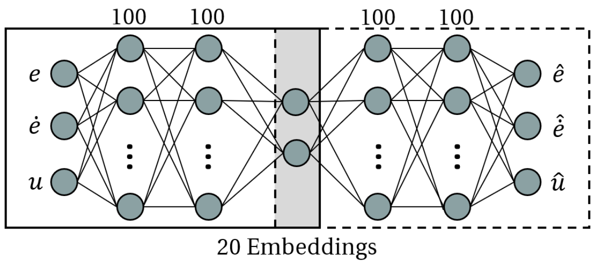
Figure 5.3: Model 2: Two stages of the pretrained end-to-end model architecture.

5.3.3 Model 3: Autoencoder + SVM

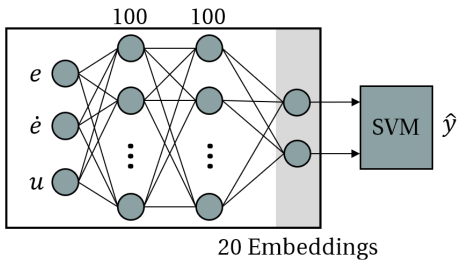
The last architecture is technically not a RNN classifier. Instead, an Autoencoder with a latent representation layer of 20 neurons is trained, again in an unsupervised manner. This means that the Decoder needs to retrieve all the information needed to reconstruct the inputs from those 20 numbers in the latent layer. The result is that for each sample those 20 embeddings can be seen as the features x

that characterise the sample. In other words, the trained Encoder automatically 'extracts' the most important information from any new sample that is fed through the network. Doing this for the n number of labelled training samples, results in a matrix of $n \times 20$ new objects that can then be used to train an SVM classifier. Note that these values are in a range between -1 and 1 since the output of both the GRU and LSTM cell type are from a hyperbolic tangent activation function. Since SVM classifiers are quite sensitive to feature scaling, tests were also run for which the embeddings were either normalised or standardised before entering into the SVM for training. It was found, however, that this only degraded the classification accuracy. The most likely cause is that the values were already scaled between -1 and 1, causing any additional transformation to result in a loss of information. The specific advantage of using an SVM, is that SVM classifiers can use kernel functions to transform the data into a new feature space that has a more pronounced decision boundary between classes (see Appendix A), which can theoretically lead to a higher classification accuracy [79].

The architecture of the AE can be seen in Figure 5.4a. To keep the comparison between the different models fair, the Encoder consists of 2 hidden layers, with the latent layer functioning as an output layer once the Decoder is removed. Aside from this, all training parameters are the same as for the AE of Model 2 (Table 5.4). For the second stage of the training process, the SVM needs to be trained. As with most ML algorithms, this means that several hyperparameters need to be tuned. The first is the kernel function which was chosen to be the Radial Basis Function (RBF) according to examples from literature [46, 93]. Second are the parameters C and γ. To find the optimal combination of parameters, it is common to apply a grid-search approach that tests all possible combinations of the two values for a given range using 5 fold cross-validation [60]. A logarithmic range was set for both parameters between 10^{-3} and 10^{3}. The resulting optimal parameters were found to be $C = 100$ and $\gamma = 1$. Note that the grid search was not executed for every combination of classifier and cell type due to the significant training time. Consequently it should be remembered when analysing the result that small amounts of improvement might be possible in a final design, since then the grid search can be reapplied to find the absolute optimal configuration.



(a) Autoencoder with 20 embeddings for feature extraction



(b) SVM Classifier

Figure 5.4: Model 3: Two stages of the Autoencoder + SVM architecture.

5.4 Results & Discussion

5.4.1 Model comparison results

The validation accuracy results for all 30 runs of the 18 different classifiers are shown in Figure 5.5 with the corresponding average training times in Table 5.5. Several things can be taken from this. First of all, it can be seen that in almost all situations the GRU outperforms the LSTM variant both in terms of accuracy and training time. While the amount of variance in the accuracy is relatively constant, the means and the medians (-) consistently show slightly better results. Furthermore, the training times for the LSTM variant are higher for both Model 1 and 2. For Model 3 the training time is higher for the GRU cell, but this is reflected in the accuracy too.

Secondly, from Figure 5.5 it can be seen that not one model seems to significantly outperform the others when comparing mean accuracies. As a matter of fact, all three models seem to win one of the

Table 5.5: 30 run average training time for each classifier model in seconds (AE training times excluded).

		C vs PS	C vs PR	PS vs PR
Model 1	GRU	157	132	143
	LSTM	209	199	182
Model 2	GRU	118	126	125
	LSTM	162	139	144
Model 3	GRU	351	90	202
	LSTM	127	48	78

three different display comparisons. The differences, however, are very small and the overlap in the boxes clearly indicates that all models are capable of scoring well. Another observation can be made by comparing the boxes to the horizontal grid lines; the higher the average accuracy, the smaller the spread. Intuitively this makes sense. The spread in the results comes largely from stochasticity in the division of samples over the train and test set in each run. To illustrate, if the test set is a good representation of the actual data, the validation accuracy will be high and vice versa. A large spread therefore indicates that a classifier is very sensitive to changes in the division of train and test data. Or in other words, the classifier has to be 'lucky' with what subset of the data it encounters in training and testing in order to make a high accuracy guess. This is an indication that the samples are more alike in classification models with large spread, compared to those with a small spread.

Another interesting result is that pretraining of the RNN does not seem to improve accuracy. Comparing Model 1 to Model 2, the former scores slightly higher two out of three times. While the differences are small, it is safe to conclude that pretraining does not boost accuracy performance, nor the consistency as was expected. Looking at the training times in Table 5.5, however, it is clear that pretraining does seem to decrease training time around 15% on average. This can add up quite quickly in training times of over 24 hours so it should not be ignored. It should be mentioned that these times are excluding the training time of the Autoencoder. However, since the AE only needs to be trained once, it can still be beneficial when doing multiple runs with for example constant architecture but different data configurations.

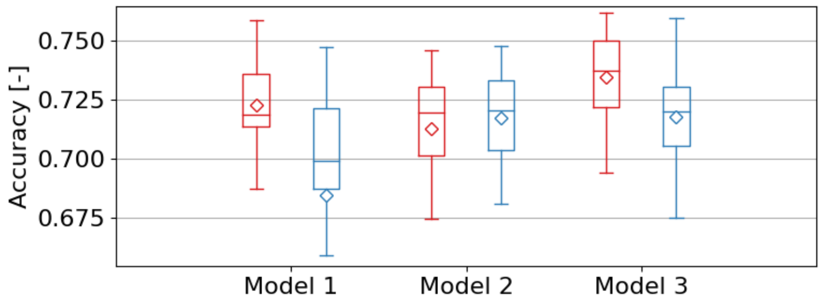
A final observation can be made looking at the training times. The SVM requires significantly longer training times than the Neural Networks for the C-PS and PS-PR comparisons. It was noted, however, that the SVM saw a huge increase in training time with the implementation of overlap and the corresponding increase in the total number of training samples[1].

5.4.2 Data comparison results

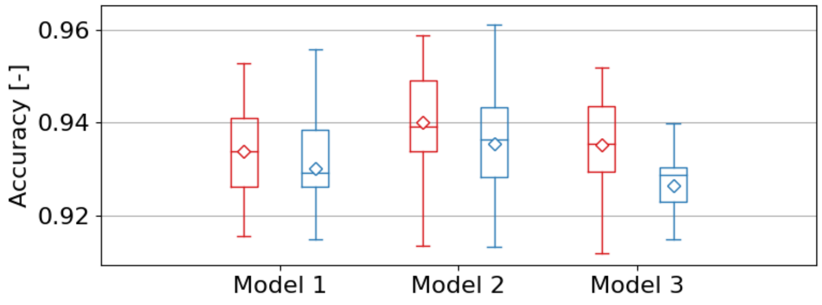
Table 5.6 shows the accuracies found for the different data sources. Again, several observations can be made. First of all, it is very likely that the Roggenkämper data did not train properly. The score of about 50% is equal to that of a completely random binary classifier. Unfortunately, no satisfying explanation for this was found, but it is most likely that the samples were not correctly labelled. The labelling was attempted according to both the description of the data file, as well as the experiment matrix provided in the original paper [69], however, to no success. It is therefore left for the next phase to potentially explore this data further.

A more useful result from the C-PS column is the difference of 11% in accuracies between the data from Vos and Van Der El 3. It provides an indication that there are indeed differences in how distinguishable the Compensatory and Pursuit control strategies are between experiments. On the other hand, the results that correspond to the different Van Der El data sets are very similar for both the C-PR and PS-PR classifiers. A more thorough analysis of the differences in task variables and experiment

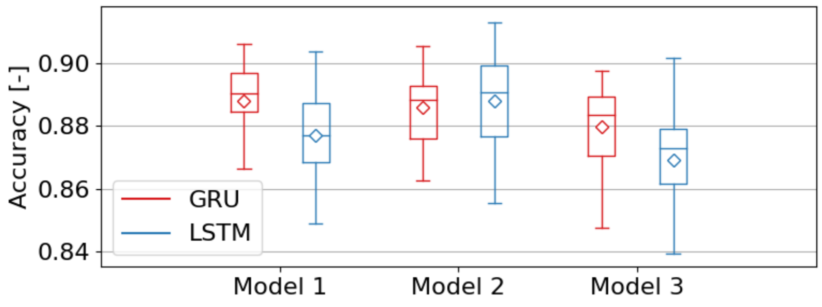
[1]Overlap and batch size have a significant impact on both the training time and accuracy of all three main models. Although a comparison of different settings for these parameters is out of scope of the preliminary simulations, it is noted for future research.



(a) Compensatory - Pursuit classifier



(b) Compensatory - Preview classifier



(c) Pursuit - Preview classifier

Figure 5.5: Box plots of the validation accuracies of three different binary classifiers over 30 runs (◊=mean).

conditions between all data sets is therefore desirable. It could help to create a better understanding of the nuances in the data that result in the shown accuracies.

5.4.3 Precision and Recall results

The final results are the Precision and Recall for each class in each binary classifier, documented in Table 5.7. Starting with the C vs PS comparison for example, it can be noticed that the value for Precision is higher for the Pursuit class, while the Compensatory class has a higher Recall rate. Similarly, Preview has a higher Precision than both Compensatory and Pursuit in their respective comparisons, while the latter two show a higher Recall. Recalling the definitions from Section 3.1 and applying them to the Pursuit class as an example, results in the following. The relatively high Precision indicates that from all samples that were labelled as Pursuit, 75% was correct. Recall, on the other hand, shows that from all samples with a Pursuit label, 71% were actually labelled as such, while the remaining 29% was incorrectly labelled as Compensatory. While the difference between these two metrics might seem subtle, the results tell a lot about the relation between the two classes, as will be explained in the discussion.

Table 5.6: Mean accuracies over 10 runs of Model 1 with a GRU cell, trained on each data set, with stand. deviation between brackets.

Data set	C vs PS	C vs PR	PS vs PR
Roggenkämper	0.52 (0.005)	-	-
Vos	0.83 (0.024)	-	-
Van der El 1	-	0.85 (0.033)	-
Van der El 2	-	-	0.94 (0.005)
Van der El 3	0.72 (0.017)	0.88 (0.014)	0.93 (0.008)

Table 5.7: Mean Precision and Recall over 30 runs of Model 1 with a GRU cell, for the three different control strategy comparisons.

	C	PS	C	PR	PS	PR
Precision	0.71	0.75	0.92	0.94	0.87	0.90
Recall	0.77	0.68	0.95	0.92	0.91	0.87

5.4.4 Discussion

The goal of the preliminary simulations was threefold:

1. To get an initial understanding of the problem.
2. To compare three selected RNN methods.
3. To compare classification on different data sources.

With the results, it is now possible to draw some conclusions on these three topics.

It has been shown that it is indeed possible, to a certain extent, to use Machine Learning classifiers to recognise HO control strategy using 1.5s windows of tracking task data. However, the level of accuracy is significantly lower than those found in the previous study by Versteeg [86]. This confirms the expectation that recognising differences in control behaviour based on the display type is a challenging task. Especially the difference between Compensatory and Pursuit control strategy is nuanced, with an average achieved classification accuracy of only around 73% in the main data set and a large uncertainty in model performance. Considering the Recall and Precision rates of these classes, however, there is also room for some optimism. The models seem to miss-classify Pursuit samples as Compensatory more often than the other way around. It is expected from literature that HOs do in fact apply a Compensatory strategy with a Pursuit display more often than the other way around (see Section 2.4.1). While currently little can be said about the direct cause of this, a more in depth analysis of the miss-classified samples in the main phase of the research might offer some interesting insights. Lastly, similar analysis of the other comparisons (C-PR and PS-PR) show the same pattern but less pronounced. All in all the results do not show anything unsuspected and there are a lot of motives for future research. What this future research will entail is the subject of Chapter 6

Secondly, the comparison between models shows that not one of the three methods significantly outperforms the others. What does not make it easier is that three different binary classifiers are trained and all methods perform best in one category. It can be concluded, however, that Model 2, which is a pretrained version of Model 1, does not show the expected improved accuracy. It can therefore be written off as a unique model. Note that it might still be useful to use a pretrained network in computational expensive simulations, since it can reduce training time anywhere between 5-25%. Furthermore, it is shown that although the differences are small, the GRU cell type does seem to outperform the LSTM. It is therefore decided to work solely with GRU cells from now on. This leaves Model 1 and

Model 3 with GRU networks to choose from. Before making a final decision, however, it could be useful to also compare the two methods in their three class classifier form. Doing so would only require a small adaptation to the existing program and so it is recommended to make this the first priority of the main research phase. This way, they can both be compared to a potential CNN implementation as well.

Finally, the comparison between different data sources show that here too an additional analysis of the data might unlock new insights into the differences between control strategies. The classifier trained on the data from Vos [87] shows an increase in accuracy of around 10 percentage points with respect to one trained on the data from Van Der El 3. Therefore, a comparison between task variables and other experimental conditions in both experiments might shed more light onto why this discrepancy exists.

5.5 Verification & Validation

5.5.1 Verification - Data pre-processing

Proper data preparation is an important step in any Machine Learning process. It is therefore essential that the data pre-processing process works as intended. The following checks were performed for each corresponding step of the code, with successful results, indicating that the program works properly:

1. *Transform data from .mat files to Numpy Arrays*: Check whether the data points in the arrays correspond to the expected location in the *.mat* files
2. *Sample features at the desired sampling frequency*: Check whether the new dimensions of the time series correspond to the expected number of time series, considering the chosen sampling rate.
3. *Randomly split each data source into a train and test set*: Check whether the number of tracking runs in the train set and test set equal 80% and 20% of the total number of runs, respectively.
4. *Standardise the data of each experiment*: Check whether the resulting means and standard deviations of all standardised data sets are in the same range.
5. *Transform the tracking runs into samples of desired length and overlap*: Check whether the resulting sample shape, as well as the total number of samples (Eq. (5.2)), correspond to the expected numbers for the given settings.
6. *Concatenate the samples of the two displays to be compared, and assign labels:* Use the resulting data in Model 1 and see if it learns.

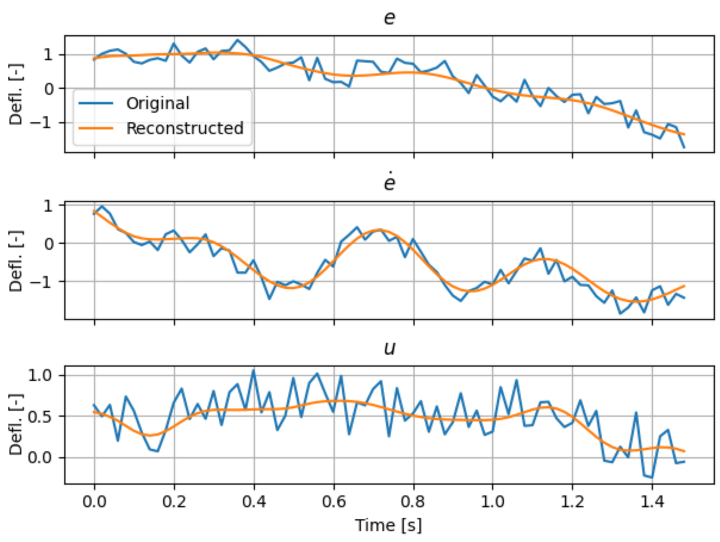
Note that all data sets pass all of these checks, except the Roggenkämper data set that fails the final test as explained in Section 5.4.

5.5.2 Verification - Autoencoders

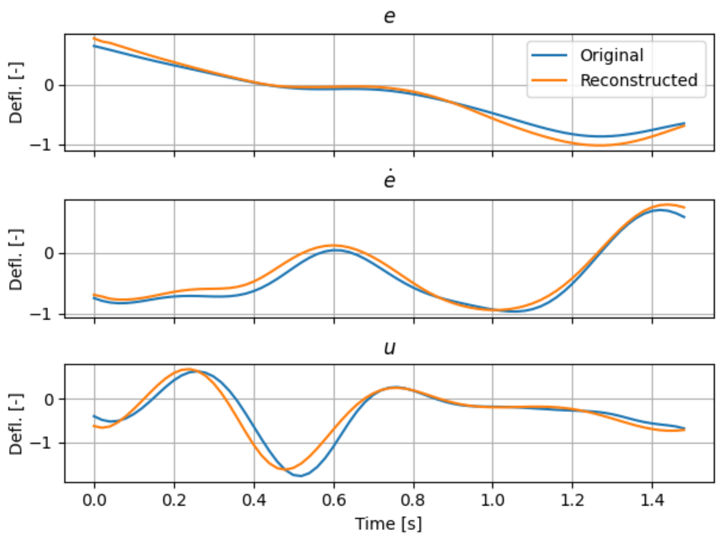
To verify that the Autoencoders run properly their performance in reconstructing the input samples says it all. Figure 5.6 shows a reconstruction example of the two types of Autoencoders for random samples. Both AE perform very well and thus it can be assumed that they are learning properly. For Model 3 we can also simply observe the results and see that they are in line with expectations, since the SVM would not be able to train if the embeddings were not representative.

5.5.3 Verification - Main models

The verification process of the main models is more difficult since we have to trust that the classifiers work properly. However, two obvious sanity check can still be performed. First, since there are three different models, it is possible to compare their results. Especially a comparison between Model 1 and Model 3 is useful since the SVM classifier relies on a completely different optimisation algorithm than



(a) Denoising AE - Model 2



(b) AE with latent layer - Model 3

Figure 5.6: Examples of reconstruction performance on random samples for the two types of Autoencoders with GRU cells.

the Neural Networks. From the results in Section 5.4 it is clear that the performances are very similar, which is a good indication that both models are performing as expected.

Secondly, for the Neural Networks it is possible to plot the training progression such as shown in Figure 5.7. This can then be compared to the classic learning curve for Neural Networks (Figure 3.3) to see if it shows the same characteristics. Looking especially at the loss function the results from Figure 5.7 are in line with expectation, in that they show a clear indication of overfitting starting around epoch 15. The loss of the test data starts to rise, while the fit on the training data seems to only get better. The accuracy plot shows a similar result, however, the decline seems to be much more gentle, if present at all. This can be explained as follows. The loss $L(y, \hat{y})$ is a function of the error between the output (any number between 0 and 1) and the target (either 0 or 1). Therefore, if the model classifies a random sample belonging to class $y = 1$ as, for example, $\hat{y} = 0.6$, the Loss function will see it as an error of $1 - 0.6 = 0.4$. The accuracy, however, will not reflect this since it sees anything above 0.5 as correctly classified to class 1. Consequently, once the model starts to diverge from the right predictions on the test set, it can take a while before this is reflected in the accuracy.

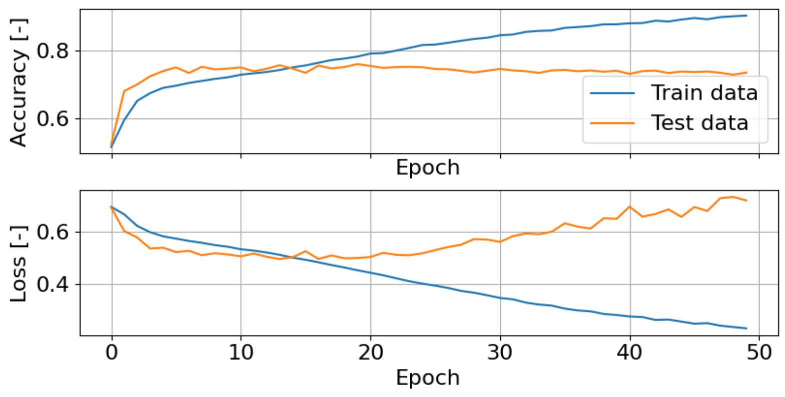


Figure 5.7: Training progression expressed in accuracy (top) and loss (bottom) of Model 1 for 50 epochs without early stopping criterion.

5.5.4 Validation

Lastly, an attempt is made to Validate the results. Again, the fact that all models are black box methods makes it difficult to check whether their inner workings correspond with the real world situation. However, the results do provide certain clues that the model is performing well. For example, the accuracies of the three different binary comparisons are in line with the expectations from literature. I.e. the accuracy of the C vs PR classifier being the highest and the C vs PS the lowest. A similar result is seen for the Precision and Recall numbers.

Unfortunately, one of the main reasons for miss-classification is unavoidable with the current data set: miss-labelling. Currently, it is only possible to label samples based on the display that was active in an experiment, while in reality we would like to label samples based on the applied control strategy. To find out if it is possible to solve this, an in depth analysis of the data and specifically the miss-classified samples is essential. If for example it turns out that it is mostly the same samples that get miss-classified, perhaps some patterns can be recognised that characterise them as such.

6 Research Plan

In order to ensure that the main research question and all its sub-components will be answered, a research plan and rough planning are established in this chapter. Since the three subtopics identified in Chapter 4 have to be treated consecutively, the plan is split up into three corresponding phases:

I Finding the optimal classifier for the task at hand, in terms of architecture, hyperparameters and use of data.

II Attempting to improve performance by both filtering experimental data and using simulated data in order to remove any discrepancies between display type and control strategy.

III Generating results on different tracking task parameters and analysing the results using current manual control Cybernetics knowledge.

The final phase of the report consists of documenting the results into a scientific article. The following three sections will provide a short plan of attack for each phase along with an initial planning. In total, the estimated duration of the main research phase is 5 months (May-September), of which 3.5 will be reserved for Phase I-III and the final 1.5 months for documentation.

6.1 Phase I: Finding the Optimal Classifier

The goal for this section is clear, but there are many different configurations to test in order to be able to find a classifier that can be deemed 'optimal'. To start, the choice needs to be made between the three remaining model possibilities:

1. Basic end-to-end Recurrent Neural Network
2. Recurrent Neural Network Autoencoder feature extractor + SVM classifier
3. Off-the-shelve CNN such as InceptionTime or Resnet [34].

To do this, a similar experiment will be run as in the preliminary simulation. A basic three-class classifier will be trained for each model and the accuracies will be compared. If the three-class classifiers seem to perform significantly worse than the binary classifiers found in this preliminary research, it might be decided to explore other classification schemes including for example Anomaly Detection. If no satisfying conclusion on what the best structure is can be drawn from the results, a qualitative analysis will be done to make a final decision. Once the classifier is chosen, its corresponding hyperparameters need to be tuned. For example, for model 3 this would be some of the specifics of the Autoencoder (number of hidden layers, number of embeddings, etc.) as well as the SVM hyperparameters (grid search of C and γ).

Lastly in this phase, the optimal data configuration should be explored. The data set used for this will be from Van Der El 3 with task variables $H_{ce} = K/s^2$ and $\tau_p = 2s$. Its great benefit is that it is quite a large data set with balanced data for all three classes, making it an ideal basis for experiments. Furthermore, it is decided to experiment with most of the same data configuration parameters as Versteeg did [86], however, two will be discarded. First, based on the results from Versteeg and some initial tests with the current code, it is decided to choose the Standardisation scaling method, and ignore the others. Secondly, no tests shall be done on a decreasing amount of subjects, number of runs, or run track length. The reason for this is as follows. Versteeg had such high initial classification accuracies that it was interesting to see how much data could be left out before performance would degrade. In this research though, the accuracies do not show the same reason for optimism. Furthermore, time is scarce and Phase 2 and 3 are deemed more important. This results in the following remaining data configuration parameters to test:

- Best combination of input variables e, $\dot{e}$, u, $\dot{u}$.
- Best combination of Window Size (WS), Sampling Frequency (SF), and Overlap (OL).

This phase should result in a reasonably well optimised classifier for the task at hand. Since this is essential for the rest of the research it should be performed thoroughly. The expected duration for Phase 1 is therefore 1.5 months.

6.2 Phase II: Analysing the Results and Interpreting the Classifier

The second Phase of the research is arguably the most important one. This thesis' research objective revolves predominantly around investigating how well Machine Learning algorithms can discern between different HO control strategies. However, the initial results presented in this report show that the nuance between display type and control strategy is quite large, especially for Compensatory and Pursuit tracking tasks. In order to deal with this, two main steps can be taken.

The first concerns an analysis of the results. For example, it could be that in multiple runs it are always the same samples that are being miss-classified. If this is indeed the case, a comparison between these samples and the consistently correct classified samples might provide insights into what the model is basing it's decision on. Additionally, attempts can be made to filter the data such that samples with an ambiguous label can be ignored when training the classifier. As the data set with the most ambiguity seems to be from the Pursuit tracking task, this will be the main focus of such an attempt. One method in particular that seems promising for filtering, is to train an anomaly detection neural network trained on for example the Compensatory data set, that can then be used on the Pursuit data set to preemptively recognise any Compensatory control behaviour. This way, any such samples can either be labelled as Compensatory for future training, or left out of the usable data altogether. Lastly, the classifier can also be tested on simulated data by using the existing HO models from Chapter 2. The advantage of this is that while HOs might adapt their internal model in order to improve performance, mathematical models are not capable of this. Consequently, the control strategy in the resulting tracking samples will always be in line with the corresponding display type and thus the accuracy is expected to be larger.

Secondly, the black box that is the classifier can perhaps be opened using explainability components of the chosen method, such as CAM for CNN. Again the goal is to understand the model better, so that it can possibly provide insights that can be useful in manual control Cybernetics. Results from this Phase might be able to provide significant new insights into discrepancies between display type and applied control strategy. Consequently, it was decided to schedule 1.5 months of research time for Phase II.

6.3 Phase III: Analysing the Effects of Other Task Variables

With the optimal classifier from Phase I and the potential new insights into separating control strategy from display type, it will be possible to run the classifier for data samples that come from experiments with different task variables such as Single Integrator CE dynamics and shorter preview times. The hypothesis is that the accuracy of the classifier will drop since the differences between control strategies are most pronounced with the task variables used in the previous Phases. However, the magnitude of this potential difference in accuracy might be able to provide some new insights.

Considering the fact that the data with other task variables is readily available, the process of testing the existing classifier w.r.t. such task variables should not take much time. The total time dedicated to Phase II is therefore expected to be around 0.5 months.

7 Conclusion

The goal of this preliminary report, was to provide a solid theoretical foundation from which the main phase of this research can be started. It contains a literature review of the state-of-the-art of the Manual Control Cybernetics (MCC) and ML Time Series Classification (TSC) research areas, as well as the results to several preliminary simulations.

The MCC literature review has shown that a clear distinction needs to be made between the two different applications of the terms Compensatory, Pursuit and Preview. The first definition being a type of display, while the second refers to a supposed corresponding control strategy. This distinction is important because it has been shown that the display type and control strategy do not always match as one might expect. Initial simulations seem to confirm this, especially for Compensatory and Pursuit tracking tasks. Furthermore, it was hypothesised that such a discrepancy is expected to be more pronounced in situations with task variables that are deemed as 'more difficult' by most HOs, such as Double Integrator system dynamics. However, this will be tested in the main research phase.

From the overview of ML techniques for TSC, two main types of Deep Neural Network classifiers were identified as the most promising for application in this research: Convolutional and Recurrent Neural Networks. Due to the RNN's relative ease of implementation for time series data, several tests were performed with different variations of architectures, but no absolute winner was found. Consequently, Phase I of the main research will consist of an empirical evaluation of off-the-shelve CNN classifier, a basic end-to-end RNN and a RNN Autoencoder with SVM classifier. Once the best performing model is chosen, the hyperparameters and data configurations will be tuned such that one 'optimal' model remains for Phase II & III.

In Phase II, the model will be used to evaluate the classifier on tracking tasks with different task variables. Lastly, in Phase III, the results will be analysed for any insights with respect to Manual Control Cybernetics and an attempt shall be made to 'open' the black-box model, in order to make the results more interpretable.

Ultimately this research aims to improve our understanding of human control behaviour by providing an effective classifier that can recognise human control behaviour in real-time. As a consequence, however, of the discrepancies that exist between display type and control strategy, especially in Pursuit tracking tasks, an attempt shall also be made to recognise patterns in the data that can lead to an improved understanding of when such discrepancies occur. Successful completion of this research would therefore not only result in an optimised classifier, but it would also pave the way for future research in understanding the intricacies of human adaptation and learning.

A

Support Vector Machines (SVM)

A Support Vector Machine (SVM) is a linear, discriminative classifier that tries to maximise the distance between the decision boundary and the nearest data points of either class [79]. An example of an SVM classifier is shown in Figure A.1. The ‘optimal’ location and slope of the decision boundary are found by solving a quadratic programming problem.

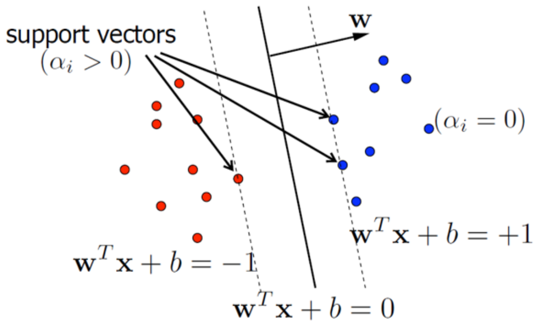

support vectors
$(\alpha_i > 0)$
$\mathbf{w}$
$(\alpha_i = 0)$
$\mathbf{w}^T\mathbf{x} + b = +1$
$\mathbf{w}^T\mathbf{x} + b = -1$
$\mathbf{w}^T\mathbf{x} + b = 0$

Figure A.1: Illustration of a Support Vector Machine classifier on a two-class problem [78].

It should be noted that though the decision boundary is linear for a standard SVM, it is also possible to ‘Kernelise’ the classifier, allowing for non-linear decision boundaries. One can map the original data points x to a new domain through a kernel function $\Phi(x)$. An example of this is shown in Figure A.2. Clearly, the original data points shown on the left can not be linearly separated while the transformed data points on the right can. The most popular Kernels are the Polynomial and Radial Basis Function (RBF).

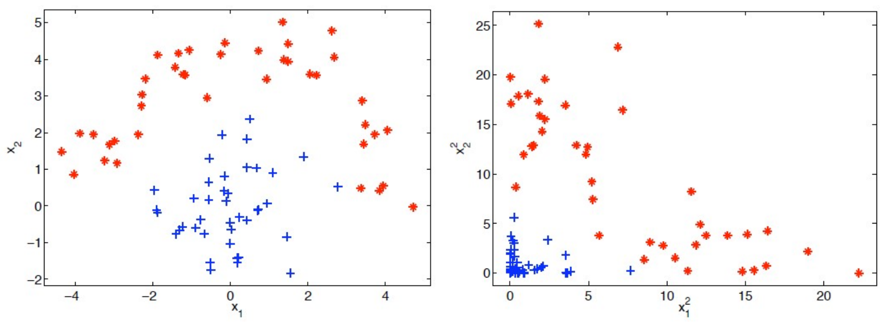


Figure A.2: Example of a polynomial kernel mapping of data points x shown on the left, to the new domain on the right [78].

Aside form training, it is possible to improve the classifier by empirically tuning the SVM with a regularisation parameters C. It provides a trade-off between the number of wrongly classified data points, and the distance between the decision boundary and the Support Vectors. An example of this is shown in Figure A.3. A small value for C means that the distance measure between the decision boundary and the Support Vectors is the dominant factor in the optimisation. With a large C, however, the dominant factor is to reduce the number of wrongly classified data points. Note that while the figure seems to suggest that a smaller C is better (less overfitting), it shows a very simple example and it usually depends on the data set what value for C is best.

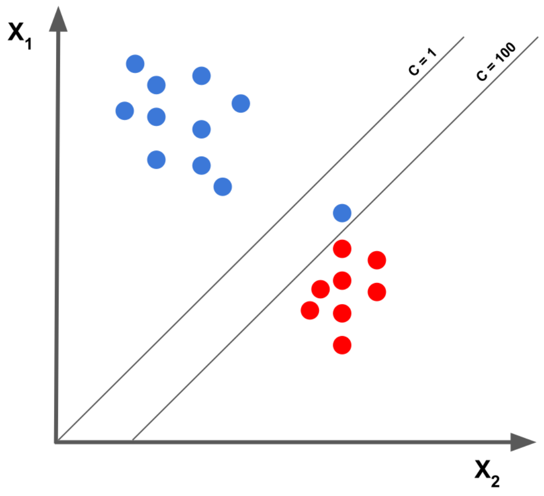


Figure A.3: The effect of C on the SVM decision boundary [47].

Lastly, one can identify another tuning parameter γ for SVMs that utilise a (Gaussian) RBF Kernel function [78]:

$$\Phi(\mathbf{x},\mathbf{y}) = \exp\left(-\gamma\|\mathbf{x}-\mathbf{y}\|^2\right), \quad \text{with} \quad \gamma = \frac{1}{\sigma^2} \tag{A.1}$$

Clearly, γ determines the size of the RBF kernel, as it is inversely proportional to the spread of the Gaussian. It therefore determines how large each of the RBF kernels (for each of the data points) is. Looking at the effect this has on the resulting decision boundary (Figure A.4), one can conclude that large values of γ lead to highly localised decision boundaries (small RBFs) in the original domain, while smaller values provide more smooth boundaries (large RBFs). Again, no ideal solution can be found, so gamma too has to be found empirically.

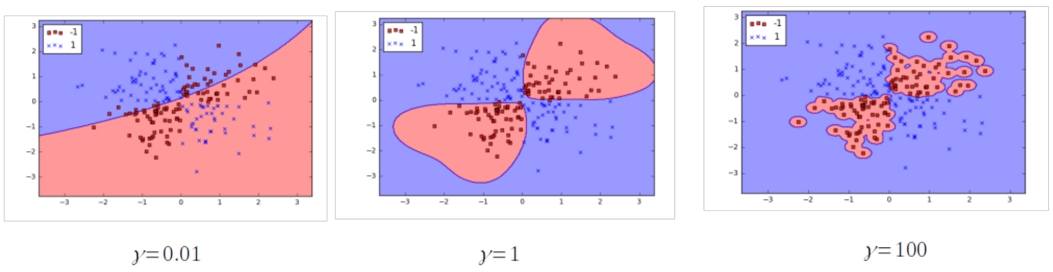


Figure A.4: The effect of γ on the SVM decision boundary [8].

As both C and γ have to be found empirically, it is common practice to apply a grid-search approach to finding the optimal configuration. For example, one could try all possible combinations of C and γ between $1e-3-1e3$ on a logarithmic scale and find the combination of parameters that provides the best performance.

Bibliography

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, and others. Tensorflow: A system for large-scale machine learning. In *12th ${$USENIX$}$ Symposium on Operating Systems Design and Implementation (${$OSDI$}$ 16)*, pages 265–283, 2016.

[2] R. Wade Allen and Henry R. Jex. An Experimental Investigation of Compensatory and Pursuit Tracking Displays with Rate and Acceleration Control Dynamics and a Disturbance Input. NASA Contractor Report NASA CR-1082, National Aeronautics and Space Administration, Washington, D.C., 1968.

[3] Shekoofeh Azizi, Sharareh Bayat, Pingkun Yan, Amir Tahmasebi, Jin Tae Kwak, Sheng Xu, Baris Turkbey, Peter Choyke, Peter Pinto, Bradford Wood, Parvin Mousavi, and Purang Abolmaesumi. Deep Recurrent Neural Networks for Prostate Cancer Detection: Analysis of Temporal Enhanced Ultrasound. *IEEE Transactions on Medical Imaging*, 37(12):2695–2703, December 2018. ISSN 0278-0062, 1558-254X. doi: 10.1109/TMI.2018.2849959. URL `https://ieeexplore.ieee.org/document/8395313/`.

[4] Anthony Bagnall, Jason Lines, Aaron Bostrom, James Large, and Eamonn Keogh. The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. *Data Mining and Knowledge Discovery*, 31(3):606–660, May 2017. ISSN 1384-5810, 1573-756X. doi: 10.1007/s10618-016-0483-9. URL `http://link.springer.com/10.1007/s10618-016-0483-9`.

[5] Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling. *arXiv:1803.01271 [cs]*, April 2018. URL `http://arxiv.org/abs/1803.01271`. arXiv: 1803.01271.

[6] L. E. Baum, T. Petrie, George Soules, and N. Weiss. A Maximization Technique Occurring in the Statistical Analysis of Probabilistic Functions of Markov Chains. *Annals of Mathematical Statistics*, 41:164–171, 1970.

[7] G. C. Beerens, H. J. Damveld, M. Mulder, M. M. Van Paassen, and J. C. Van Der Vaart. Investigation into Crossover Regression in Compensatory Manual Tracking Tasks. *Journal of Guidance, Control, and Dynamics*, 32(5):1429–1445, September 2009. ISSN 0731-5090, 1533-3884. doi: 10.2514/1.43528. URL `https://arc.aiaa.org/doi/10.2514/1.43528`.

[8] Saptashwa Bhattacharyya. Support Vector Machine: Kernel Trick; Mercer's Theorem, December 2018. URL `rstanding-support-vector-machine-part-2-kernel-trick-mercers-theorem-e1e6848c6c4d`.

[9] Facundo Bre, Juan M. Gimenez, and Víctor D. Fachinotti. Prediction of wind pressure coefficients on building surfaces using artificial neural networks. *Energy and Buildings*, 158:1429–1441, 2018. ISSN 0378-7788. doi: https://doi.org/10.1016/j.enbuild.2017.11.045. URL `https://www.sciencedirect.com/science/article/pii/S0378778817325501`.

[10] Krisztian Buza. Time Series Classification and its Applications. In *Proceedings of the 8th International Conference on Web Intelligence, Mining and Semantics*, pages 1–4, Novi Sad Serbia, June 2018. ACM. ISBN 978-1-4503-5489-9. doi: 10.1145/3227609.3227690. URL `https://dl.acm.org/doi/10.1145/3227609.3227690`.

[11] Chandralika Chakraborty and P.H. Talukdar. Issues and Limitations of HMM in Speech Processing: A Survey. *International Journal of Computer Applications*, 141(7):13–17, May 2016. ISSN 09758887. doi: 10.5120/ijca2016909693. URL `http://www.ijcaonline.org/archives/volume141/number7/chakraborty-2016-ijca-909693.pdf`.

[12] Sotirios P. Chatzis and Dimitrios I. Kosmopoulos. A variational Bayesian methodology for hidden Markov models utilizing Student's-t mixtures. *Pattern Recognition*, 44(2):295–306, February 2011. ISSN 00313203. doi: 10.1016/j.patcog.2010.09.001. URL https://linkinghub.elsevier.com/retrieve/pii/S0031320310004383.

[13] Wei Chen and Ke Shi. Multi-scale Attention Convolutional Neural Network for time series classification. *Neural Networks*, 136:126–140, April 2021. ISSN 08936080. doi: 10.1016/j.neunet.2021.01.001. URL https://linkinghub.elsevier.com/retrieve/pii/S0893608021000010.

[14] Rube Chernikoff, Henry P. Brimingham, and Franklin V. Taylor. A comparison of pursuit and compensatory tracking under conditions of aiding and no aiding. *Journal of Experimental Psychology*, 49(1):55–59, 1955. ISSN 0022-1015. doi: 10.1037/h0047938. URL http://doi.apa.org/getdoi.cfm?doi=10.1037/h0047938.

[15] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. *arXiv:1406.1078 [cs, stat]*, September 2014. URL http://arxiv.org/abs/1406.1078. arXiv: 1406.1078.

[16] Francois Chollet and others. Keras, 2015. URL https://github.com/fchollet/keras. Publisher: GitHub.

[17] R. Colombi and S. Giordano. Multiple hidden Markov models for categorical time series. *Journal of Multivariate Analysis*, 140:19–30, September 2015. ISSN 0047259X. doi: 10.1016/j.jmva.2015.04.002. URL https://linkinghub.elsevier.com/retrieve/pii/S0047259X15000901.

[18] Frank M. Drop, Rick J. De Vries, Max Mulder, and H. H. Bülthoff. The Predictability of a Target Signal Affects Manual Feedforward Control. In *Proceedings of the 13th IFAC/IFIP/IFORS/IEA Symposium on Analysis, Design, and Evaluation of Human-Machine Systems, Kyoto, Japan*, 2016.

[19] Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio. Why Does Unsupervised Pre-training Help Deep Learning? *Journal of Machine Learning Research*, 11:625–660, February 2010.

[20] Bilal Esmael, Arghad Arnaout, Rudolf K. Fruhwirth, and Gerhard Thonhauser. Improving time series classification using Hidden Markov Models. In *2012 12th International Conference on Hybrid Intelligent Systems (HIS)*, pages 502–507, Pune, India, December 2012. IEEE. ISBN 978-1-4673-5116-4 978-1-4673-5114-0 978-1-4673-5115-7. doi: 10.1109/HIS.2012.6421385. URL http://ieeexplore.ieee.org/document/6421385/.

[21] Kunihiko Fukushima. Neocognitron: A Self-Organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position. *Biological Cybernetics*, 36:193–202, 1980.

[22] Klaus Greff, Rupesh Kumar Srivastava, Jan Koutník, Bas R. Steunebrink, and Jürgen Schmidhuber. LSTM: A Search Space Odyssey. *IEEE Transactions on Neural Networks and Learning Systems*, 28(10):2222–2232, October 2017. ISSN 2162-237X, 2162-2388. doi: 10.1109/TNNLS.2016.2582924. URL http://arxiv.org/abs/1503.04069. arXiv: 1503.04069.

[23] Narendhar Gugulothu, Vishnu TV, Pankaj Malhotra, Lovekesh Vig, Puneet Agarwal, and Gautam Shroff. Predicting Remaining Useful Life using Time Series Embeddings based on Recurrent Neural Networks. *arXiv:1709.01073 [cs]*, October 2017. URL http://arxiv.org/abs/1709.01073. arXiv: 1709.01073.

[24] Priyanka Gupta, Pankaj Malhotra, Lovekesh Vig, and Gautam Shroff. Transfer Learning for Clinical Time Series Analysis using Recurrent Neural Networks. *arXiv:1807.01705 [cs, stat]*, July 2018. URL http://arxiv.org/abs/1807.01705. arXiv: 1807.01705.

[25] Aurélien Géron. *Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow.* O'Reilly Media, Inc., 2nd edition, September 2019. ISBN 978-1-4920-3264-9.

[26] Charles R. Harris, K. Jarrod Millman, St'efan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fern'andez del R'ıo, Mark Wiebe, Pearu Peterson, Pierre G'erard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020. doi: 10.1038/s41586-020-2649-2. URL `https://doi.org/10.1038/s41586-020-2649-2`. Publisher: Springer Science and Business Media LLC.

[27] K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. doi: 10.1109/CVPR.2016.90.

[28] Ronald A. Hess. Pursuit Tracking and Higher Levels of Skill Development in the Human Pilot. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-11(4):262–273, 1981. doi: 10.1109/TSMC.1981.4308673.

[29] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8): 1735–1780, November 1997. ISSN 0899-7667, 1530-888X. doi: 10.1162/neco.1997.9.8.1735. URL `https://direct.mit.edu/neco/article/9/8/1735-1780/6109`.

[30] M. Hossain and M. Jenkin. Recognizing Hand-Raising Gestures using HMM. In *The 2nd Canadian Conference on Computer and Robot Vision (CRV'05)*, pages 405–412, Victoria, BC, Canada, 2005. IEEE. ISBN 978-0-7695-2319-4. doi: 10.1109/CRV.2005.67. URL `http://ieeexplore.ieee.org/document/1443159/`.

[31] David Hubel. Single Unit Activity in Striate Cortex of Unrestrained Cats. *The Journal of Physiology*, 147:226–238, 1959.

[32] Michael Hüsken and Peter Stagge. Recurrent neural networks for time series classiÿcation. *Neurocomputing*, 50:223–235, 2003. ISSN 0925-2312.

[33] Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre-Alain Muller. Transfer learning for time series classification. *arXiv:1811.01533 [cs, stat]*, November 2018. doi: 10.1109/BigData.2018.8621990. URL `http://arxiv.org/abs/1811.01533`. arXiv: 1811.01533.

[34] Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre-Alain Muller. Deep learning for time series classification: a review. *Data Mining and Knowledge Discovery*, 33(4):917–963, July 2019. ISSN 1384-5810, 1573-756X. doi: 10.1007/s10618-019-00619-1. URL `http://link.springer.com/10.1007/s10618-019-00619-1`.

[35] Hassan Ismail Fawaz, Benjamin Lucas, Germain Forestier, Charlotte Pelletier, Daniel F. Schmidt, Jonathan Weber, Geoffrey I. Webb, Lhassane Idoumghar, Pierre-Alain Muller, and François Petitjean. InceptionTime: Finding AlexNet for Time Series Classification. *Data Mining and Knowledge Discovery*, 34(6):1936–1962, November 2020. ISSN 1384-5810, 1573-756X. doi: 10.1007/s10618-020-00710-y. URL `http://arxiv.org/abs/1909.04939`. arXiv: 1909.04939.

[36] K. Ito and M. Ito. Tracking behavior of human operators in preview control systems. *Electrical Engineering in Japan*, 95(1):120–127, 1975. ISSN 04247760, 15206416. doi: 10.1002/eej.4390950118. URL `http://doi.wiley.com/10.1002/eej.4390950118`.

[37] Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. An Empirical Exploration of Recurrent Network Architectures. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37, page 9, Lille, France, 2015. Journal of Machine Learning Research.

[38] Martin Kers. Maximum Likelihood Estimation of Linear Time-Varying Pilot-Vehicle System Parameters. Master's thesis, Delft University of Technology, August 2012.

[39] Alexios Kotsifakos and Panagiotis Papapetrou. Model-Based Time Series Classification. In Hendrik Blockeel, Matthijs van Leeuwen, and Veronica Vinciotti, editors, *Advances in Intelligent Data Analysis XIII*, volume 8819, pages 179–191, Cham, 2014. Springer International Publishing. ISBN 978-3-319-12570-1 978-3-319-12571-8. doi: 10.1007/978-3-319-12571-8_16. URL `http://link.springer.com/10.1007/978-3-319-12571-8_16`. Series Title: Lecture Notes in Computer Science.

[40] Ezra S Krendel and Duane T. McRuer. A Servomechanisms Approach to Skill Development. *Journal of the Franklin Institute*, 269(1):19, 1960.

[41] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. doi: 10.1109/5.726791.

[42] Kenneth Lee. A Theory Explains Deep Learning, January 2017. URL `https://medium.com/@kennethlee_98497/a-theory-explains-deep-learning-79b02598a2d5`.

[43] Zhenji Lu, Riender Happee, Christopher Cabrall, Miltos Kyriakidis, and Joost de Winter. Human Factors of Transitions in Automated Driving: A General Framework and Literature Survey. *Transportation Research Part F Traffic Psychology and Behaviour*, 43:183–196, 2016. doi: 10.1016/j.trf.2016.10.007.

[44] Martin Lãngkvist, Lars Karlsson, and Amy Loutfi. A review of unsupervised feature learning and deep learning for time-series modeling. *Pattern Recognition Letters*, 42:11–24, 2014. URL `http://dx.doi.org/10.1016/j.patrec.2014.01.008`.

[45] Pankaj Malhotra, Vishnu TV, Anusha Ramakrishnan, Gaurangi Anand, Lovekesh Vig, Puneet Agarwal, and Gautam Shroff. Multi-Sensor Prognostics using an Unsupervised Health Index based on LSTM Encoder-Decoder. *arXiv:1608.06154 [cs]*, August 2016. URL `http://arxiv.org/abs/1608.06154`. arXiv: 1608.06154.

[46] Pankaj Malhotra, Vishnu TV, Lovekesh Vig, Puneet Agarwal, and Gautam Shroff. TimeNet: Pretrained deep recurrent neural network for time series classification. *arXiv:1706.08838 [cs]*, June 2017. URL `http://arxiv.org/abs/1706.08838`. arXiv: 1706.08838.

[47] Satya Mallick. Support Vector Machines, November 2018. URL `https://learnopencv.com/support-vector-machines-svm/`.

[48] Duane T. McRuer and H.R. Jex. A Review of Quasi-Linear Pilot Models. *IEEE Transactions on Human Factors in Electronics*, HFE-8(3):231–249, September 1967. ISSN 0096-249X. doi: 10.1109/THFE.1967.234304. URL `http://ieeexplore.ieee.org/document/1698271/`.

[49] Duane T. McRuer, Dunstan Graham, Ezra S. Krendel, and William Jr. Reisener. Human Pilot Dynamics in Compensatory Systems: Theory, Models, and Experiments with Controlled Element and Forcing Function Variations. Technical Report AFFDL-TR-65-15, Air Force Flight Dynamics Laboratory, Wright-Patterson Air Force Base (OH), 1965.

[50] Q. Meng, D. Catchpoole, D. Skillicom, and P. J. Kennedy. Relational autoencoder for feature extraction. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 364–371, 2017. doi: 10.1109/IJCNN.2017.7965877.

[51] Tom Mitchell. *Machine Learning*. McGraw Hill, New York, 1997. ISBN 0-07-042807-7. URL `https://www.cs.cmu.edu/~tom/mlbook.html`.

[52] Max Mulder. *Cybernetics of tunnel-in-the-sky displays.* Delft Univ. Press, Delft, 1999. ISBN 978-90-407-1963-9. OCLC: 247756672.

[53] Max Mulder. Manual Control with Pursuit Displays: New Insights, New Models, New Issues. *IFAC PapersOnLine*, page 7, 2019.

[54] Max Mulder, Daan M. Pool, David A. Abbink, Erwin R. Boer, Peter M. T. Zaal, Frank M. Drop, Kasper van der El, and Marinus M. van Paassen. Manual Control Cybernetics: State-of-the-Art and Current Trends. *IEEE Transactions on Human-Machine Systems*, 48(5):468–485, October 2018. ISSN 2168-2291, 2168-2305. doi: 10.1109/THMS.2017.2761342. URL `https://ieeexplore.ieee.org/document/8088358/`.

[55] Layan Nahlawi, Farhad Imani, Mena Gaed, Jose A. Gomez, Madeleine Moussa, Eli Gibson, Aaron Fenster, Aaron D. Ward, Purang Abolmaesumi, Hagit Shatkay, and Parvin Mousavi. Prostate Cancer: Improved Tissue Characterization by Temporal Modeling of Radio-Frequency Ultrasound Echo Data. In Sebastien Ourselin, Leo Joskowicz, Mert R. Sabuncu, Gozde Unal, and William Wells, editors, *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2016*, volume 9900, pages 644–652. Springer International Publishing, Cham, 2016. ISBN 978-3-319-46719-1 978-3-319-46720-7. doi: 10.1007/978-3-319-46720-7_75. URL `http://link.springer.com/10.1007/978-3-319-46720-7_75`. Series Title: Lecture Notes in Computer Science.

[56] Andrew Ng and Michael Jordan. On Discriminative vs. Generative Classifiers: A comparison of logistic regression and naive Bayes. *Advances in Neural Information Processing Systems*, 14:8, 2001. URL `https://papers.nips.cc/paper/2001/hash/7b7a53e239400a13bd6be6c91c4f6c4e-Abstract.html`.

[57] Christopher Olah. Understanding LSTM Networks, August 2015. URL `colah.github.io/posts/2015-08-Understanding-LSTMs/`.

[58] Mario Olivari. *Measuring pilot control behavior in control tasks with haptic feedback.* PhD thesis, University of Pisa, Pisa, Italy, 2016.

[59] Josh Patterson and Adam Gibson. *Deep Learning; A Practitioner's Approach.* O'Reilly Media, Inc., 2017. ISBN 978-1-4919-1425-0.

[60] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, and others. Scikit-learn: Machine learning in Python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.

[61] A. Pikrakis, S. Theodoridis, and D. Kamarotos. Classification of musical patterns using variable duration hidden Markov models. *IEEE Transactions on Audio, Speech, and Language Processing*, 14(5):1795–1807, 2006. doi: 10.1109/TSA.2005.858542.

[62] D M Pool, P M T Zaal, H J Damveld, and M M van Paassen. Modeling Wide-Frequency-Range Pilot Equalization for Control of Aircraft Pitch Dynamics. *Journal of Guidance, Control, and Dynamics*, 34(5):14, 2011. doi: 10.2514/1.53315.

[63] Alexandru Popovici, Peter M. T. Zaal, and Daan M. Pool. Dual Extended Kalman Filter for the Identification of Time-Varying Human Manual Control Behavior. In *Proceedings of the AIAA Modeling and Simulation Technologies Conference, Denver (CO)*, 2017. doi: 10.2514/6.2017-3666. Issue: AIAA-2017-3666.

[64] E. C. Poulton. Perceptual Anticipation in Tracking with Two-Pointer and One-Point Displays. *British Journal of Psychology. General Section*, 43(3):222–229, August 1952. ISSN 03732460. doi: 10.1111/j.2044-8295.1952.tb00345.x. URL `http://doi.wiley.com/10.1111/j.2044-8295.1952.tb00345.x`.

[65] L. Rabiner and B. Juang. An introduction to hidden Markov models. *IEEE ASSP Magazine*, 3 (1):4–16, 1986. ISSN 0740-7467. doi: 10.1109/MASSP.1986.1165342. URL `http://ieeexplore.ieee.org/document/1165342/`.

[66] Lawrence R Rabiner. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. *PROCEEDINGS OF THE IEEE*, 77(2):30, 1989.

[67] Chotirat Ratanamahatana and Eamonn Keogh. Three Myths about Dynamic Time Warping Data Mining. In *Proceedings of the 2005 SIAM International Conference on Data Mining, SDM 2005*, 2005. doi: 10.1137/1.9781611972757.50.

[68] Nils Reimers and Iryna Gurevych. Optimal Hyperparameters for Deep LSTM-Networks for Sequence Labeling Tasks. *arXiv:1707.06799 [cs]*, August 2017. URL `http://arxiv.org/abs/1707.06799`. arXiv: 1707.06799.

[69] Nicole Roggenkämper, Daan M. Pool, Frank M. Drop, Marinus M. van Paassen, and Max Mulder. Objective ARX Model Order Selection for Multi-Channel Human Operator Identification. In *Proceedings of the AIAA Modeling and Simulation Technologies Conference, Washington, D.C.*, June 2016. doi: 10.2514/6.2016-4299. Issue: AIAA-2016-4299.

[70] Alejandro Pasos Ruiz, Michael Flynn, James Large, Matthew Middlehurst, and Anthony Bagnall. The great multivariate time series classification bake off: a review and experimental evaluation of recent algorithmic advances. *Data Mining and Knowledge Discovery*, 35(2):401–449, March 2021. ISSN 1384-5810, 1573-756X. doi: 10.1007/s10618-020-00727-3. URL `http://link.springer.com/10.1007/s10618-020-00727-3`.

[71] Khaled Saleh, Mohammed Hossny, and Saeid Nahavandi. Driving behavior classification based on sensor data fusion using LSTM recurrent neural networks. In *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, pages 1–6, 2017. ISBN 978-1-5386-1526-3. doi: 10.1109/ITSC.2017.8317835.

[72] Jobn W Senders and Marianne Cruzen. Tracking Performance on Combined Compensatory and Pursuit Tasks. *WADC Technical Report*, 52-39:19, February 1952.

[73] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Reserach*, 15:1929–1958, June 2014.

[74] Henk G Stassen. Internal Representation, Internal Model, Human Performance Model and Mental Workload. *Automatica*, 26(4):10, 1990.

[75] Ilya Sutskever. *Training Recurrent Neural Networks.* Phd, University of Toronto, Toronto, 2013.

[76] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, Boston, MA, USA, June 2015. IEEE. ISBN 978-1-4673-6964-0. doi: 10.1109/CVPR.2015.7298594. URL `http://ieeexplore.ieee.org/document/7298594/`.

[77] Manie Tadayon and Greg Pottie. Comparative Analysis of the Hidden Markov Model and LSTM: A Simulative Approach. *arXiv:2008.03825 [cs, stat]*, August 2020. URL `http://arxiv.org/abs/2008.03825`. arXiv: 2008.03825.

[78] David Tax. Delft University of Technology - CS4220 Machine Learning 1 - Lecture Slides, December 2020.

[79] Sergios Theodoridis and Konstantinos Koutroumbas. *Pattern recognition.* Elsevier Acad. Press, Amsterdam, 4. ed edition, 2009. ISBN 978-1-59749-272-0. OCLC: 550588366.

[80] M. Tomizuka and D. E. Whitney. The Human Operator in Manual Preview Tracking (an Experiment and Its Modeling Via Optimal Control). *Journal of Dynamic Systems, Measurement, and Control*, 98(4):407–413, December 1976. ISSN 0022-0434, 1528-9028. doi: 10.1115/1.3427058. URL `https://asmedigitalcollection.asme.org/dynamicsystems/article/98/4/407/400728/The-Human-Operator-in-Manual-Preview-Tracking-an`.

[81] Kasper van der El, Daan M. Pool, Herman J. Damveld, Marinus M. van Paassen, and Max Mulder. An Empirical Human Controller Model for Preview Tracking Tasks. *IEEE Transactions on Cybernetics*, 46(11):2609–2621, November 2016. ISSN 2168-2267, 2168-2275. doi: 10.1109/TCYB.2015.2482984. URL `https://ieeexplore.ieee.org/document/7312951/`.

[82] Kasper van der El, Joao Morais Almeida, Daan M. Pool, Marinus M. van Paassen, and Max Mulder. The Effects of Motion Feedback in Manual Preview Tracking Tasks. In *Proceedings of the AIAA Modeling and Simulation Technologies Conference, Denver (CO)*, 2017. doi: 10.2514/6.2017-3472. Issue: AIAA-2017-3472.

[83] Kasper van der El, Sharon Padmos, Daan M. Pool, Marinus M. van Paassen, and Max Mulder. Effects of Preview Time in Manual Tracking Tasks. *IEEE Transactions on Human-Machine Systems*, 48(5):486–495, October 2018. ISSN 2168-2291, 2168-2305. doi: 10.1109/THMS.2018.2834871. URL `https://ieeexplore.ieee.org/document/8363006/`.

[84] Kasper van der El, Daan M. Pool, Marinus M. van Paassen, and Max Mulder. Effects of Preview on Human Control Behavior in Tracking Tasks With Various Controlled Elements. *IEEE Transactions on Cybernetics*, 48(4):1242–1252, April 2018. ISSN 2168-2267, 2168-2275. doi: 10.1109/TCYB. 2017.2686335. URL `http://ieeexplore.ieee.org/document/7891942/`.

[85] Kasper van der El, Daan M. Pool, Marinus M. van Paassen, and Max Mulder. Effects of Target Trajectory Bandwidth on Manual Control Behavior in Pursuit and Preview Tracking. *IEEE Transactions on Human-Machine Systems*, 50(1):68–78, February 2020. ISSN 2168-2291, 2168-2305. doi: 10.1109/THMS.2019.2947577. URL `https://ieeexplore.ieee.org/document/8897675/`.

[86] R Versteeg. Classifying Human Manual Control Behaviour using LSTM Recurrent Neural Networks. Master's thesis, Delft University of Technology, Delft, 2019.

[87] Maxim C Vos, Daan M Pool, Herman J Damveld, Marinus M van Paassen, and Max Mulder. Identification of Multimodal Control Behavior in Pursuit Tracking Tasks. In *IEEE International Conference on Systems, Man, and Cybernetics*, pages 69–74, 2014. ISBN 978-1-4799-3840-7.

[88] Yasi Wang, Hongxun Yao, and Sicheng Zhao. Auto-encoder based dimensionality reduction. *Neurocomputing*, 184:232–242, April 2016. ISSN 09252312. doi: 10.1016/j.neucom.2015.08.104. URL `https://linkinghub.elsevier.com/retrieve/pii/S0925231215017671`.

[89] Zhiguang Wang, Weizhong Yan, and Tim Oates. Time Series Classification from Scratch with Deep Neural Networks: A Strong Baseline. *arXiv:1611.06455 [cs, stat]*, December 2016. URL `http://arxiv.org/abs/1611.06455`. arXiv: 1611.06455.

[90] R. J. Wasicko, Duane T. McRuer, and R. E. Magdaleno. Human Pilot Dynamic Response in Single-loop Systems with Compensatory and Pursuit Displays. Technical Report AFFDL-TR-66-137, Air Force Flight Dynamics Laboratory, December 1966.

[91] D.H. Wolpert and W.G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, April 1997. ISSN 1089778X. doi: 10.1109/4235.585893. URL `http://ieeexplore.ieee.org/document/585893/`.

[92] Laurence R. Young. On Adaptive Manual Control. *IEEE Transactions on Man-Machine Systems*, 10(4):292–331, December 1969. doi: 10.1109/TMMS.1969.299931.

[93] Wennian Yu. Analysis of different RNN autoencoder variants for time series classification and machine prognostics. *Mechanical Systems and Signal Processing*, Mechanical Systems and Signal Processing(149):16, 2021. ISSN 0888-3270. URL `https://doi.org/10.1016/j.ymssp.2020.107322`.

[94] Peter M. T. Zaal, Daan M. Pool, Q. P. Chu, Marinus M. van Paassen, Max Mulder, and Jan A. Mulder. Modeling Human Multimodal Perception and Control Using Genetic Maximum Likelihood Estimation. *Journal of Guidance, Control, and Dynamics*, 32(4):1089–1099, 2009. doi: 10.2514/1. 42843.

[95] Bendong Zhao, Huanzhang Lu, Shangfeng Chen, Junliang Liu, and Dongya Wu. Convolutional neural networks for time series classification. *Journal of Systems Engineering and Electronics*, 28(1):162–169, February 2017. ISSN 10044132. doi: 10.21629/JSEE.2017.01.18. URL `http://ieeexplore.ieee.org/document/7870510/`.

[96] Rui Zhao. Deep learning and its applications to machine health monitoring. *Mechanical Systems and Signal Processing*, 115:213–237, 2019. ISSN 0888-3270. URL `https://doi.org/10.1016/j.ymssp.2018.05.050`.

[97] Rui Zhao, Dongzhe Wang, Ruqiang Yan, Kezhi Mao, Fei Shen, and Jinjiang Wang. Machine Health Monitoring Using Local Feature-Based Gated Recurrent Unit Networks. *IEEE Transactions on Industrial Electronics*, 65(2):1539–1548, February 2018. ISSN 0278-0046, 1557-9948. doi: 10.1109/TIE.2017.2733438. URL `http://ieeexplore.ieee.org/document/7997605/`.

[98] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba. Learning Deep Features for Discriminative Localization. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2921–2929, 2016. doi: 10.1109/CVPR.2016.319.