Can PredNet predict time-series to improve the generalization process?

Sjoerd Jousma

A thesis presented for the degree of
Master of Engineering

Robotics
Delft University of Technology
The Netherlands
July 12, 2022

ACKNOWLEDGEMENTS

I would like to thank my supervisors ir. C.A. van Hoof MBA and prof. dr. ir. M. Wisse for their guidance, expertise and general help with the thesis process.
I would like to thank B. van der Meer, E. Veldhuis and M. Deken for the collaboration on the development of the active inference velocity controller.
Finally, I would like to thank my girlfriend Kirsten for her support, motivation and faith.
Delft, University of Technology
July 12, 2022

ABSTRACT

This thesis is inspired by Active Inference to contribute to its improvement in the Robotics work field. However, the results and applications of this thesis are useful in a broader perspective, namely in any field that makes use of derivatives and the forecasting of a time-series signal. The goal of this study is to determine a new approach for calculating the derivatives using the finite difference technique and forecasting. The finite-difference technique is based on the Taylor expansion. The Taylor expansion uses derivatives to calculate a future signal sample, while the finite difference technique uses signal samples to estimate derivatives. The accuracy of the estimation can theoretically be increased using more signal samples and choosing signal samples that are closer to the signal sample of interest. For these reasons, the central method is interesting as it maximizes both of these properties. The central method uses future signal samples for the sample of interest which are not available in an online setting. Future values can be forecast through for example statistical models and predictive coding. Statistical models such as the auto regression method are the golden standard but cannot learn in an online fashion. Predictive coding can learn in an online fashion and research found a promising branch utilizing neural network frameworks. PredNet came out as an interesting model due to its state-of-the-art results and flexibility. The research question in this thesis is as follows: can predicting future data points by use of PredNet help improve the accuracy of the generalized coordinates?

Two models of PredNet are trained on data obtained from random commands from a simulated Jackal robot. One is trained in an online fashion and one in an offline fashion. Its predictions are compared against two other forecasting methods: copying the input and the auto regression method. The predictions are then used with the finite difference method to estimate derivatives. Four methods are used: a method using as little past data as possible, a method using as much past data as possible, a method using as little centralized data as possible and a method using as much centralized data as possible. Derivatives are determined on the same simulated Jackal data and are compared against a derivative obtained from a central method using as much data as possible that all come from the true signal.

Results show no forecasting difference between the two types of PredNet models, but there are differences in training speed: training in an online fashion is not fast enough to keep up with the data stream. This suggests that the quality of neural networks is good enough for online applications, but that more work needs to be done on optimizing neural networks for online learning to make it a practical solution. However, the PredNet predictions are not as accurate as the auto regression method. This means that neural networks can still learn from the auto regression method when used for forecasting time series. An example of improving neural networks is making hybrid models. Using the PredNet forecasts for estimating derivatives did not increase the accuracy of the derivatives meaning the predictions are not accurate enough. Besides that, using more samples, either from the past or the future, did not increase the derivative estimate either. Most of this outcome can be attributed to the choice of data set which contained many transitions between commands which could not be predicted by the PredNet models or be seen by the finite difference method. Suggestions for improvements are to focus the training of a model on either the transition period or the period in between, using more modern models and/or using different methods for estimating derivatives.

This work concludes that neural networks show promise for online applications, that training a model requires a specific data set or a complex enough model to utilize a more general data set, and that using more samples with the finite difference method is not guaranteed to provide more accurate estimates of derivatives.

Contents

List of Figures

List of Tables

Chapter 1

INTRODUCTION

It can be said that the brain is fundamental to our existence and one of the most complex and fascinating subjects for study. Understanding how the brain works will help understand how humans work in general, but also reproduce certain aspects in other areas such as robotics. Many theories exist about the various aspects of the brain (, such as learning, motor skills, emotions, etc.), but most are limited to specific functionality. The Free Energy Principle (FEP) [1] is an attempt at unifying several global theories. The FEP is a form of Predictive Coding (PC) and was inspired by the implementation of Rao & Ballard [2] who managed to optimize neuronal dynamics and connectivity using the same energy function [3]: minimizing prediction error. The FEP called this energy function "variational free energy" and extended it to include precision weighting to represent the uncertainty of components within the model.

At the core of the theory stand precision weighting and generalized coordinates. Generalized coordinates contain the state of the system. When used in the FEP it also includes derivatives of states, which make it possible to predict future states given the current states. FEP has been put into practice [4], but does not achieve state-of-the-art results yet. A common problem is obtaining accurate derivatives for the generalized coordinates. This problem also occurred in a recent attempt at implementing an Active Inference Velocity Controller for a Jackal Robot: a small four-wheeled skid-steering robot. The project is explained in chapter 2. In the implementation the derivatives were estimated using the backwards finite difference method, a mathematical model based on the Taylor expansion. However, the aim for the project was not to obtain the most accurate derivatives, but simply to create a working controller. As such, the finite difference implementation did not make use of its full potential.

The finite difference method uses data samples to estimate the derivative. The estimate becomes more accurate by using more data samples and data samples that are closer to the sample of interest. By predicting a signal through other means than the derivative it is possible to obtain an estimate for future states. These future states can be used in conjunction with past states to have as many samples as possible that are as close to the sample of interest as possible to obtain the optimal finite difference method. More details about the finite difference methods are provided in chapter 3

There are a few options to produce estimates for the future states, for example, statistical models (e.g. auto regression methods) and neural networks. Autoregressive models have been and are at the forefront of predicting time series [5] due to their accuracy, reliability, and ease of use. However, recent developments have shown that neural networks can be a strong competitor [6]. A further advantage of neural networks is that they can learn in an online setting, in contrast to the AR models for example, which need a complete and unchanging dataset to learn. In [7] various branches of PC are explored where it was found that implementations that utilize frameworks for neural networks seem to be most promising in terms of results and flexibility, in particular a model called PredNet. PredNet makes use of LSTM units which [6] found to be the most efficient unit type to perform training on time series. An overview of the workings of PredNet is given in chapter 4.

Combining the knowledge of the difficulties with obtaining generalized coordinates and the possibilities that can still be used leads to the following research question: can predicting future data points by use of PredNet help improve the accuracy of the generalized coordinates? This question can be further divided into two sub questions:

1. Does PredNet predict accurate and useful future data?

2. Does using more data points in finite difference models improve accuracy?

Accuracy is defined as the prediction error, while usefulness is defined as how much the predictions can improve the estimate of the derivatives.

This thesis is structured as follows. Firstly, in chapters 2 to 4 background information will be provided. Chapter 2 describes the implementation of the active inference controller for the Jackal robot with details how the generalized coordinates were estimated. In chapter 3 a detailed description of the finite difference method is provided which shows how it works and how it can be used most effectively. Chapter 4 explains on what the modern predictive coding models are based off and contains details about the PredNet model that is used in this thesis. Secondly, the research questions are answered in chapters 5 to 7. In chapter 5 an overview of the experiments is given. Chapter 6 shows the results of those experiments and what conclusions they provide. In chapter 7 the results are discussed and compared with related literature. Finally, an overview of the conclusions drawn in this thesis is provided in chapter 8.

Chapter 2

ACTIVE INFERENCE VELOCITY CONTROLLER

This chapter provides an understanding of the processes which this thesis aims to improve. It provides background information about the Jackal robot that is used in this thesis in section 2.1 and about the active inference velocity controller that was created to track the velocity of the Jackal robot in sections 2.2 and 2.3. It further shows where the generalized coordinates and derivatives come into play when using active inference in a practical application in section 2.4, what method was used to estimate the derivatives in section 2.5.

2.1 Jackal robot

The Jackal robot is an unmanned ground vehicle that uses differential steering to move around, a picture is provided in fig. 2.1. This means that all wheels are fixed to the body of the robot and cannot rotate relative to the robot. It steers through a difference in velocity between the left sided wheels and right sided wheels. The velocity of the left sided wheels are coupled together and the same goes for the right sided wheels. The robot can be controlled using ROS either through WiFi or a physical cable. On its own it contains an onboard computer, battery, wheel velocity sensors, integrated GPS, gyroscope and accelerometer. The gyroscope and accelerometer provide IMU data which is a time-series signal containing state information about the robots position, orientation and velocity. It is possible to add a wide variety of accessories to the robot to increase its capabilities, but these were not used in this thesis.

Figure 2.1: A picture of the Jackal robot: an unmanned ground vehicle using differential steering.

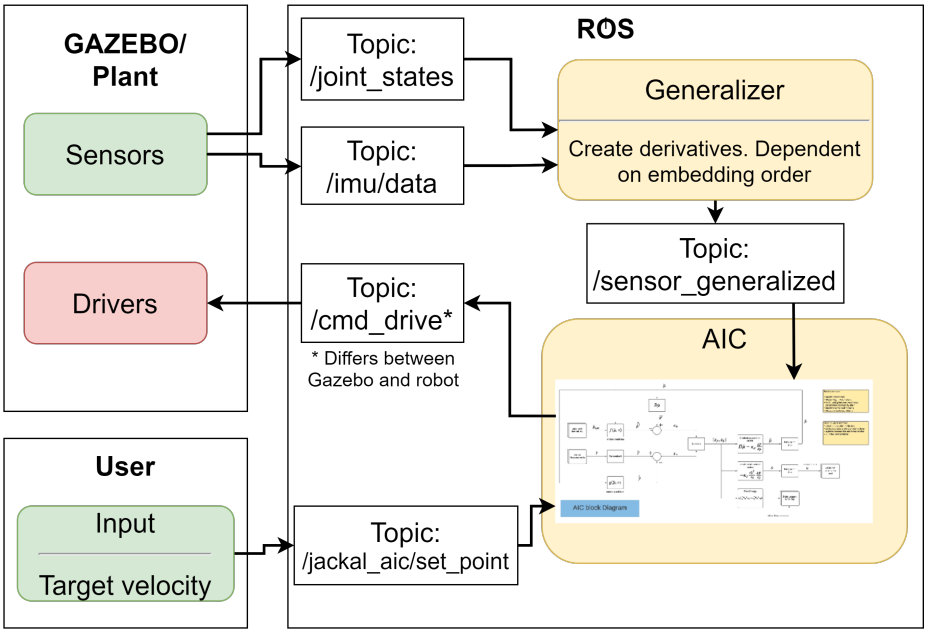

Figure 2.2: An overview of all components involved and through what ROS topics they communicate. The "Generalizer" block is the focus of this thesis: new methods for determining derivatives are explored. The "AIC" block contains the Active Inference calculations which uses a user-input and the generalized coordinates to calculate ω_L and ω_R.

2.2 Overview of the active inference velocity controller

The controller for the Jackal robot was developed by a group of students consisting of Mitchel Deken, Bob van der Meer, Erik Veldhuis and Sjoerd Jousma, who is the author of this thesis. The controller needs to send and receive information to and from the sensors to and from the drivers, see fig. 2.2. This communication is handled by ROS, which is inherently present on the Jackal robot. There are many topics available, but for the controller only a few are needed. An overview is shown in fig. 2.2. Sensory input from the wheel speeds are published on the topic /joint_states, while the speed of the center of mass μ is published on /imu/data, containing $\ddot{x}$, $\ddot{y}$ and $\dot{\theta}$. The sensory data are read by a generalization node that calculates the derivatives of the wheel speeds. These derivatives are then published on an intermediate topic: /sensor_generalized. These generalized measurements, combined with the desired velocity of the center of mass that is published on /jackal_aic/set_point, are the actual input for the AIC. The AIC calculates the action required to bring down the Free Energy and publishes drive commands on /cmd_drive when using the robot and /jackal_left_wheel/command and /jackal_right_wheel/command when using Gazebo. These topics contain the velocities that the left- and right wheel of the Jackal should have: ω_L and ω_R. The controller produces these ω's which are then used by an internal lower-level controller that transforms the drive commands to actuator voltages.

2.3 Choice of dynamic model

The dynamic model was mostly designed by Mitchel Deken. This section mostly serves as a basis for the equations which will later be used in sections 2.4 and 2.5.

For the control of a differential steering mobile robot, often no dynamical model is used [8, 9, 10, 11, 12]. Instead, only a kinematic mapping between the body velocity and the wheel velocities is used. This gives a steady-state solution of the body velocity resulting from the wheel velocities. In this way, using inverse kinematics, the desired body velocity input by the user (linear and angular) can be converted by the controller to give corresponding wheel speed inputs to the wheels. This method however, does not contain any dynamics, and therefore cannot adjust well for any disturbances, like unknown slippage.

For the implementation of active inference, we are looking for a simple solution as we are focusing on the implementation in the hardware, rather than on complex models. Therefore, we chose to opt out

of a traditional dynamic model, and use attractor dynamics instead. To make use of attractor dynamics one needs a desired state and the current state. The desired state is called the attractor state and is the velocity input by the user. The current state is given by the sensors. The technique is very simply: it aims to bring the current state as close to the attractor as possible by comparing the two. Their difference are multiplied by a gain. That outcome is then used to obtain a control signal that is send to the drivers, see fig. 2.2. This technique is already used in a few active inference implementations [13, 14], where we will use the version with the time constant [14]. These attractor dynamics will encode a preference of the states to be moving towards a prior (desired state), stated as follows:

$$f(\mu) = (\mu_d - \mu)\tau^{-1} \tag{2.1}$$

where μ are the beliefs (states in active inference), μ_d is the prior belief (desired state) and τ is a time constant.
While not using any dynamic model from literature, we still need a mapping from the states to observations (body velocities to wheel velocities) in order to compare with measured values. For this we opted for the extended differential drive model (as used by [8]), because this is the most simple and most used kinematic mapping available, while still giving sufficient performance:

$$\mu = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = r\alpha \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \\ 0 & 0 \\ -\frac{1}{\hat{b}} & \frac{1}{\hat{b}} \end{bmatrix} \begin{bmatrix} \omega_L \\ \omega_R \end{bmatrix} \tag{2.2}$$

$$r\alpha \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \\ 0 & 0 \\ -\frac{1}{\hat{b}} & \frac{1}{\hat{b}} \end{bmatrix} = G \tag{2.3}$$

where α is a slip parameter and $\hat{b}$ is the virtual width of the vehicle (as if it had two wheels). These are both empirically determined. For the $g(\mu)$ of the generative model, we need the mapping from the states to observations. And as we have access to the gyroscope, encoder and accelerometer measurements, it will look as follows:

$$y = \begin{bmatrix} \dot{\theta} \\ \omega_L \\ \omega_R \\ \ddot{x} \\ \ddot{y} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ G_{11}^{-1} & G_{12}^{-1} & G_{13}^{-1} \\ G_{21}^{-1} & G_{22}^{-1} & G_{23}^{-1} \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ \frac{1}{r\alpha} & 0 & \frac{-\hat{b}}{2r\alpha} \\ \frac{1}{r\alpha} & 0 & \frac{\hat{b}}{2r\alpha} \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} \tag{2.4}$$

where G^{-1} is the pseudo-inverse of the kinematic mapping mentioned above.

2.4 Free energy formulation

This section will focus on the definition of the free energy formula for the active inference velocity controller of the Jackal robot. It will show that the derivatives and the generalized coordinates are central to active inference and why it is important to have the information as accurate as possible. The formulas in this section were derived and implemented by all team members. In the previous paragraph

the generative model of the sensory data and of the state dynamics are defined. Equation 2.5 displays the general free energy formula as it was defined by Pezzato et al. [13]

$$F = \frac{1}{2}\sum_{i=0}^{n_d-1}[\varepsilon_o^{(i)(T)}\Sigma_{o^{(i)}}^{-1}\varepsilon_o^{(i)} + \varepsilon_\mu^{(i)(T)}\Sigma_{\mu^{(i)}}^{-1}\varepsilon_\mu^{(i)}] \tag{2.5}$$

Based on the models defined earlier the error terms, ε_o and ε_μ, can be computed. These can be found in equations 2.6 and 2.7. Furthermore, to find the first generalized order these questions are differentiated once to obtain equations 2.8 and 2.9. Note that in this example only the first derivative is shown. However, in the actual application the derivatives can go up to the 5th order.

$$\varepsilon_o = o - g(\mu) = o - R\mu$$

$$\varepsilon_o = \begin{bmatrix} \dot{\theta} \\ \omega_L \\ \omega_R \\ \ddot{x} \\ \ddot{y} \end{bmatrix} - \begin{bmatrix} 0 & 0 & 1 \\ \frac{1}{r\alpha} & 0 & \frac{-\hat{b}}{2r\alpha} \\ \frac{1}{r\alpha} & 0 & \frac{\hat{b}}{2r\alpha} \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} \tag{2.6}$$

$$\varepsilon_\mu = \mu' - f(\mu) \tag{2.7}$$

$$\begin{aligned} \varepsilon_y &= \tilde{y} - R\mu \\ \varepsilon_y' &= \tilde{y}' - R\mu' \\ &\vdots \\ \varepsilon_y^{(5)} &= \tilde{y}^{(5)} - R\mu^{(5)} \end{aligned} \tag{2.8}$$

$$\begin{aligned} \varepsilon_\mu &= \mu' - f(\mu) = \mu' - (\mu_d - \mu)\tau^{-1} \\ \varepsilon_\mu' &= \mu'' + \mu'\tau^{-1} \\ &\vdots \\ \varepsilon_\mu^{(5)} &= \mu^{(6)} + \mu^{(5)}\tau^{-1} \end{aligned} \tag{2.9}$$

For readability, equations 2.8 and 2.9 can be written in matrix form (equation 2.10). Furthermore, the same can be done for the precision matrices, which is done in equation 2.11. Note that in this case $\widetilde{\varepsilon}$ and $\widetilde{\Pi}$ consist of all generalized orders.

$$\widetilde{\varepsilon} = \begin{bmatrix} \widetilde{\varepsilon}_o \\ \widetilde{\varepsilon}_\mu \end{bmatrix} \tag{2.10}$$

$$\widetilde{\Pi} = \begin{bmatrix} \widetilde{\Pi}_o & 0 \\ 0 & \widetilde{\Pi}_\mu \end{bmatrix} \tag{2.11}$$

Combining equations 2.5, 2.10 and 2.11 equation 2.12 can be found. This is the free energy formula for the active inference velocity controller for a jackal robot.

$$F = \frac{1}{2}\tilde{\varepsilon}^T \tilde{\Pi} \tilde{\varepsilon} \tag{2.12}$$

From the free energy formula the belief update can be found, the belief update is defined in equation 2.13. In equation 2.14 this has been written out for 2 generalized orders.

$$\dot{\tilde{\mu}} = D\tilde{\mu} - k_\mu \left(\frac{\delta \tilde{\varepsilon}_\mu}{\delta \tilde{\mu}} \frac{\delta F}{\delta \tilde{\varepsilon}_\mu} + \frac{\delta \tilde{\varepsilon}_y}{\delta \tilde{\mu}}^T \frac{\delta F}{\delta \tilde{\varepsilon}_y}\right) \tag{2.13}$$

$$\begin{bmatrix} \dot{\mu} \\ \dot{\mu}' \\ \dot{\mu}'' \end{bmatrix} = \begin{bmatrix} 0 & I & 0 \\ 0 & 0 & I \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \mu \\ \mu' \\ \mu'' \end{bmatrix} - k_\mu \left(\begin{bmatrix} \frac{I}{\tau} & 0 & 0 \\ I & \frac{I}{\tau} & 0 \\ 0 & I & \frac{I}{\tau} \end{bmatrix} \begin{bmatrix} \Pi_{\mu}\varepsilon_{\mu} \\ \Pi_{\mu'}\varepsilon_{\mu'} \\ \Pi_{\mu''}\varepsilon_{\mu''} \end{bmatrix} - \begin{bmatrix} R^T & 0 & 0 \\ 0 & R^T & 0 \\ 0 & 0 & R^T \end{bmatrix} \begin{bmatrix} \Pi_{y}\varepsilon_{y} \\ \Pi_{y'}\varepsilon_{y'} \\ \Pi_{y''}\varepsilon_{y''} \end{bmatrix} \right) \tag{2.14}$$

Lastly the action update is defined in equation 2.15. This has been written out for 2 generalized orders of motion in equation 2.17. Note that the term $\frac{\delta \tilde{\varepsilon}_o}{\delta a}$ is replaced by the forward model (F) as defined in equation 2.16.

$$\dot{a} = -\kappa_a \frac{\delta \tilde{\varepsilon}_y}{\delta a} \frac{\delta F}{\delta \tilde{\varepsilon}_y} \tag{2.15}$$

$$F = \begin{bmatrix} \frac{-r\alpha}{\hat{b}} & 1 & 0 & 0 & 0 \\ \frac{r\alpha}{\hat{b}} & 0 & 1 & 0 & 0 \end{bmatrix} \tag{2.16}$$

$$\dot{a} = -\kappa_a \begin{bmatrix} F & 0 & 0 \\ 0 & F & 0 \\ 0 & 0 & F \end{bmatrix} \begin{bmatrix} \Pi_{y}\varepsilon_{y} \\ \Pi_{y'}\varepsilon_{y'} \\ \Pi_{y''}\varepsilon_{y''} \end{bmatrix} \tag{2.17}$$

The actual belief and action values are calculated using the forward Euler method:

$$\mu_k = \mu_{k-1} + \dot{\mu}_k \tag{2.18}$$

2.5 Generalization

The previous section showed the importance of the derivatives to active inference. This section will show the method that was used to produce the derivatives as input for the active inference velocity controller. Improving this method is the main goal of this thesis.

There is only a limited amount of derivatives real sensors can measure. For example, if we have access to a position sensor, a velocity sensor and an acceleration sensor, then we have an embedding order of 2 which comes from direct measurements. However, what if some of these type of sensors are not available or what if the highest state is acceleration and we need derivatives of that? In that case we need to synthesize the higher order derivatives from the available measurements.

To retrieve the higher order derivatives we made use of the Master's thesis of I.L. Hijne [15]. A robot receives measurement samples in discrete time. In that case a simple differentiating technique can be used: finite differences. It is possible to extract higher order derivatives from a Taylor expansion, full details of this process be provided in chapter 3. For the active inference velocity controller we made use of a precomputed matrix containing the coefficients with which to multiply the signal to obtain the derivatives, see eq. (2.19).

$$\begin{bmatrix} y_k^{(0)} \\ y_k^{(1)} \\ y_k^{(2)} \\ y_k^{(3)} \\ y_k^{(4)} \\ y_k^{(5)} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & -\frac{1}{h} & \frac{1}{h} \\ 0 & 0 & 0 & \frac{1}{h^2} & -\frac{2}{h^2} & \frac{1}{h^2} \\ 0 & 0 & -\frac{1}{h^3} & \frac{3}{h^3} & -\frac{3}{h^3} & \frac{1}{h^3} \\ 0 & \frac{1}{h^4} & -\frac{4}{h^4} & \frac{6}{h^4} & -\frac{4}{h^4} & \frac{1}{h^4} \\ -\frac{1}{h^5} & \frac{5}{h^5} & -\frac{10}{h^5} & \frac{10}{h^5} & -\frac{5}{h^5} & \frac{1}{h^5} \end{bmatrix} \begin{bmatrix} y_{k-5} \\ y_{k-4} \\ y_{k-3} \\ y_{k-2} \\ y_{k-1} \\ y_k \end{bmatrix} \tag{2.19}$$

This matrix solves the problem and we get the derivatives of the sensory input with a constant h. As will be explained in chapter 3, it provides the minimum effort and accuracy for obtaining the derivatives. It can be adapted to include more time samples for the lower derivatives or to include future signal samples as well as past signal samples. Exploring the effectiveness of these options is the goal of this thesis.

For the implementation in the Jackal robot the matrix was further optimized. Using eq. (2.19) it is required to keep track of multiple old data samples, which causes the information to have overlap. This equation can be rewritten to only contain data of the sample we are interested in and the previous sample plus its derivatives. A special vector $\check{\tilde{\mathbf{y}}}$ can be constructed that contains the current sample and the previous sample with higher order derivatives of the previous sample:

$$\tilde{\mathbf{y}} = Q\check{\tilde{\mathbf{y}}}, \qquad \check{\tilde{\mathbf{y}}} = [\mathbf{y}_k, \mathbf{y}_{k-1}^\top]^\top.$$

With these definitions, the new matrix becomes:

$$\begin{bmatrix} y_k^{(0)} \\ y_k^{(1)} \\ y_k^{(2)} \\ y_k^{(3)} \\ y_k^{(4)} \\ y_k^{(5)} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{h} & -\frac{1}{h} & 0 & 0 & 0 & 0 \\ \frac{1}{h^2} & -\frac{1}{h^2} & -\frac{1}{h} & 0 & 0 & 0 \\ \frac{1}{h^3} & -\frac{1}{h^3} & -\frac{1}{h^2} & -\frac{1}{h} & 0 & 0 \\ \frac{1}{h^4} & -\frac{1}{h^4} & -\frac{1}{h^3} & -\frac{1}{h^2} & -\frac{1}{h} & 0 \\ \frac{1}{h^5} & -\frac{1}{h^5} & -\frac{1}{h^4} & -\frac{1}{h^3} & -\frac{1}{h^2} & -\frac{1}{h} \end{bmatrix} \begin{bmatrix} y_k \\ y_{k-1}^{(0)} \\ y_{k-1}^{(1)} \\ y_{k-1}^{(2)} \\ y_{k-1}^{(3)} \\ y_{k-1}^{(4)} \end{bmatrix} \tag{2.20}$$

This form is initialized with the higher order derivatives at zero. The derivatives will be filled in time step per time step.

The form eq. (2.20) is used in the Jackal robot. The sensor measurements are synchronized so that h is the same for all measurements during a time step. Their derivatives are computed based on eq. (2.20). However instead of making one large matrix that matches the size of $\check{\tilde{\mathbf{y}}}$, it is more efficient to split this

$\tilde{\tilde{\mathbf{y}}}$ into several single vectors of a measurement $\check{y}_{k,i} = [y_{k,i}, y^{(0)}_{k-1,i}, y^{(1)}_{k-1,i}, y^{(2)}_{k-1,i}, y^{(3)}_{k-1,i}, y^{(4)}_{k-1,i}]$, where each $\check{y}_{k,i}$ is a unique sensor measurement denoted by i. Then use these smaller vectors and the 6×6 matrix Q to calculate its derivatives as:

$$\tilde{y}_{k,i} = Q\check{y}_{k,i}$$

$$\check{y}_{k,i} = [y_{k,i}, y^{(0)}_{k-1,i}, y^{(1)}_{k-1,i}, y^{(2)}_{k-1,i}, y^{(3)}_{k-1,i}, y^{(4)}_{k-1,i}]$$

This way Q only has to be calculated once and can be reused for every vector of $\check{y}_{k,i}$.

2.6 Summary

The Jackal robot is an unmanned ground vehicle using differential steering. An active inference velocity controller was created to control the velocity of the left- and right wheels. The left-and right wheel states are coupled to the robot states by the dynamic model. The dynamic model is used in the free energy formulation which shows a dependence on error and measurement derivatives. These derivatives are not readily available and need to be estimated which is done with finite differences. The finite difference method that was used can still be optimized which is further explored in this thesis.

Chapter 3

FINITE DIFFERENCES

To understand how the finite difference method can be optimized, a thorough background of the method and how it can be used will be provided in this chapter. A large part of this chapter was been derived from [16].

3.1 Derivation of finite difference method

Finite difference method is used for discrete signals and is based on the Taylor expansion. The Taylor expansion computes signal sample y_{k+h} from sample y_k and a weighted sum of the derivatives of y_k and is defined as:

$$y_{k+h} = \sum_{i=0}^{i_{max}} \frac{y_k^{(i)}}{i!}(k+h-k)^i = \sum_{i=0}^{i_{max}} \frac{y_k^{(i)}}{i!}h^i. \tag{3.1}$$

where (i) denotes the i^{th} order derivative of y_k and h is an interval step $<< 1$. Note that i_{max} goes to infinity and that $0^0 = 1$. Finally, the derivative of order zero of f is defined to be f itself and $(x-a)^0$ and 0! are both defined to be 1.

In practice it is not possible to set i_{max} to infinity as we have to deal with numerical and computational limits as well as difficulties with obtaining the derivatives. Instead, i_{max} can be chosen arbitrarily and determines the number of derivatives to be taken into account. A non-infinite expansion becomes an estimation of the true value, meaning that the expansion will now also include an error term. For example, setting $i_{max} = 1$, will give:

$$y_{k+h} = y_k + hy_k^{(1)} + \mathcal{O}(h^2). \tag{3.2}$$

$\mathcal{O}(h^n)$ denotes the order of magnitude of the error, also called the error order, and is equal to the highest omitted derivative, which in the above example is the second derivative. There are two ways of increasing the accuracy of the estimation: by reducing h and by increasing i_{max}. A reduction of h will make the time steps smaller which will make the prediction lie closer to y_k, reducing the error as well. Increasing i_{max} will add an extra derivative term to the equation as well as an increase to the power of the error order which, with $h << 1$, will significantly decrease the error. There are no rules for determining the required number of derivatives as they are strongly dependent on the application and the availability of derivatives.

Equation (3.2) can now be rewritten so that the derivative can be calculated based on y_k and y_{k+h}:

$$\begin{aligned} y_{k+h} &= y_k + hy_k^{(1)} + \mathcal{O}(h^2) \\ y_k^{(1)} &= \frac{1}{h}(y_{k+h} - y_k) + \mathcal{O}(h). \end{aligned} \tag{3.3}$$

Equation (3.3) is an example of a forwards finite difference derivative method as it makes use only of data that lies in the future. A more common method is a backwards finite difference derivative which

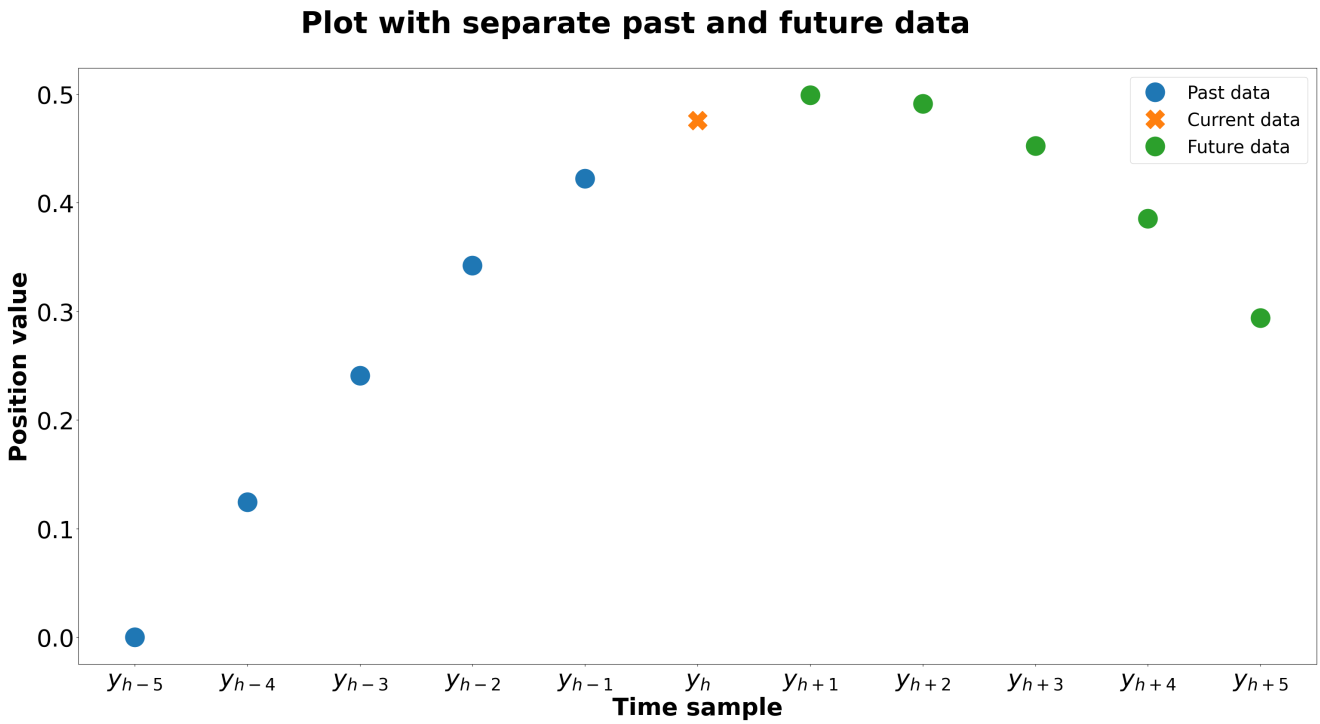

Figure 3.1: This figure shows what data points the three finite difference techniques make use of. Backwards uses past and present data, forwards uses present and future data and central uses past, present and future data.

only makes use of data that lies in the past:

$$y_{k-h} = y_k - h y_k^{(1)} + \mathcal{O}(h^2)$$
$$y_k^{(1)} = \frac{1}{h}(y_k - y_{k-h}) + \mathcal{O}(h).$$

The error order for the derivative is of a lower power than the error order for the predicted point in the original equation. This shows a possible cause as to why estimating derivatives through this method quickly becomes unstable and why improvements are important. There is a third common method called the central method for which we will see an example shortly. An overview of the different methods is shown in fig. 3.1.

If we would like to obtain a term for the second derivative, we would have to raise i_{max} to 2. However, that alone will leave us with more unknowns than equations. In order to add an extra equation an extra data sample has to be added. The data samples can be collected in a stencil:

$$\mathbf{y} = [y_1, ..., y_N]$$

where N denotes the length of the stencil.

To solve for a second derivative we can use an example of the final common finite difference method: a central difference derivative which makes use of a balanced number of past and future data points: $\check{\mathbf{y}} = [y_{k-1}, y_k, y_{k+1}]$. In eq. (3.3) the derivative was obtained by simply rewriting the eq. (3.2). Continuing this approach we now need to solve

$$y_k^{(2)} \approx c_1 y_{k-1} + c_2 y_k + c_3 y_{k+1} + \mathcal{O}(h) \tag{3.4}$$

for the coefficients $[c_1, c_2, c_3]$. If we expand all the terms using eq. (3.1) and then collect them we get:

$$
\begin{aligned}
y_k^{(2)} \approx\ & c_1\frac{(-1h)^0}{0!}y_k + c_1\frac{(-1h)^1}{1!}y_k^{(1)} + c_1\frac{(-1h)^2}{2!}y_k^{(2)} + \\
& c_2 y_k + \\
& c_3\frac{(1h)^0}{0!}y_k + c_3\frac{(1h)^1}{1!}y_k^{(1)} + c_3\frac{(1h)^2}{2!}y_k^{(2)} + \mathcal{O}(h) \\
\approx\ & (-1^0c_1 + c_2 + 1^0c_3)y_k + \frac{h}{1!}(-1^1c_1 + 1^1c_3)y_k^{(1)}h + \frac{h^2}{2!}(-1^2c_1 + 1^2c_3)y_k^{(2)} + \mathcal{O}(h^3) \\
\approx\ & (c_1 + c_2 + c_3)y_k + \frac{h}{1!}(-c_1 + c_3)y_k^{(1)} + \frac{h^2}{2!}(c_1 + c_3)y_k^{(2)} + \mathcal{O}(h).
\end{aligned}
$$

This can easily be solved using linear algebra, especially when rewritten in matrices:

$$
\begin{bmatrix} 1 & 1 & 1 \\ -1 & 0 & 1 \\ 1 & 0 & 1 \end{bmatrix}\begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} = \frac{2!}{h^2}\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} + \mathcal{O}(h)
$$

$$
\begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} = \frac{1}{h^2}\begin{bmatrix} 1 & 1 & 1 \\ -1 & 0 & 1 \\ 1 & 0 & 1 \end{bmatrix}^{-1}\begin{bmatrix} 0 \\ 0 \\ 2 \end{bmatrix} + \mathcal{O}(h) = \frac{1}{h^2}\begin{bmatrix} 1 \\ -2 \\ 1 \end{bmatrix} + \mathcal{O}(h) \tag{3.5}
$$

where we find the coefficients $[1,\ -2,\ 1]$. We can substitute these back into eq. (3.4): $y_k^{(2)} \approx 1y_{k-1} - 2y_k + 1y_{k+1}$, which gives us the option to estimate the 2^{nd} order derivative based on the stencil $[-1,\ 0,\ 1]$ with an error order of h.

From expanding the finite difference technique to the second derivative we can learn a few things. First, in order to solve for derivative d, we need to make sure that there are enough data samples in stencil $\check{\mathbf{y}}$ to solve the equations: $N > d$. Second, linear algebra is invaluable for calculating the coefficients of the finite difference methods. Third, since it is possible to add more data samples to the stencil to solve for higher derivatives, it should also be possible to add more data samples while solving for the same derivative. For example, if we use the same stencil and solve for the 1^{st} derivative instead of the 2^{nd} derivative we get:

$$
\begin{aligned}
y_k^{(1)} \approx\ & c_1\frac{(-1h)^0}{0!}y_k + c_1\frac{(-1h)^1}{1!}y_k^{(1)} + c_1\frac{(-1h)^2}{2!}y_k^{(2)} + && (3.6)\\
& c_2 y_k + && (3.7)\\
& c_3\frac{(1h)^0}{0!}y_k + c_3\frac{(1h)^1}{1!}y_k^{(1)} + c_3\frac{(1h)^2}{2!}y_k^{(2)} + \mathcal{O}(h^2) && (3.8)\\
\approx\ & (-1^0c_1 + c_2 + 1^0c_3)y_k + \frac{h}{1!}(-1^1c_1 + 1^1c_3)y_k^{(1)}h + \frac{h^2}{2!}(-1^2c_1 + 1^2c_3)y_k^{(2)} + \mathcal{O}(h^2) && (3.9)\\
\approx\ & (c_1 + c_2 + c_3)y_k + \frac{h}{1!}(-c_1 + c_3)y_k^{(1)} + \frac{h^2}{2!}(c_1 + c_3)y_k^{(2)} + \mathcal{O}(h^2), && (3.10)
\end{aligned}
$$

$$
\begin{bmatrix} 1 & 1 & 1 \\ -1 & 0 & 1 \\ 1 & 0 & 1 \end{bmatrix}\begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} = \frac{1!}{h}\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} + \mathcal{O}(h^2)
$$

$$
\begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} = \frac{1}{h}\begin{bmatrix} 1 & 1 & 1 \\ -1 & 0 & 1 \\ 1 & 0 & 1 \end{bmatrix}^{-1}\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} + \mathcal{O}(h^2) = \frac{1}{2h}\begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} + \mathcal{O}(h^2). \tag{3.11}
$$

Equation (3.6) shows that if more data samples are added to the sample, the order of the error term increases, similarly to how adding derivatives to the Taylor expansion increases the error order for the Taylor equation. In fact, the error order for the finite difference technique follows from the

desired derivative and the number of data samples: $n = N - d$. Furthermore, eqs. (3.5) and (3.11) show similarities in how the linear algebra is solved. The equations can be written in a more general fashion. Let $\mathbf{c}$ be the vector that contains the coefficients c, $\mathbf{S}$ be the stencil that contains the signal samples s and $\mathbf{d}$ be the vector containing the desired derivative d:

$$\mathbf{c} = \begin{bmatrix} c_1^0 \\ \vdots \\ c_N \end{bmatrix} \qquad \mathbf{S} = \begin{bmatrix} s_1^0 & \cdots & s_N^0 \\ \vdots & \ddots & \vdots \\ s_1^{N-1} & \cdots & s_N^{N-1} \end{bmatrix} \qquad \mathbf{d} = \begin{bmatrix} 0 \\ \vdots \\ d! \\ \vdots \\ 0 \end{bmatrix}.$$

We then get the general equation and matrix calculation:

$$s_1^n c_1 + \ldots + s^n c^c c_N = \frac{d!}{h^d}\delta(n-d) \text{ for } 0 \leq n \leq N-1 \tag{3.12}$$

$$\begin{bmatrix} c_1^0 \\ \vdots \\ c_N \end{bmatrix} = \frac{1}{h^d} \begin{bmatrix} s_1^0 & \cdots & s_N^0 \\ \vdots & \ddots & \vdots \\ s_1^{N-1} & \cdots & s_N^{N-1} \end{bmatrix}^{-1} \begin{bmatrix} 0 \\ \vdots \\ d! \\ \vdots \\ 0 \end{bmatrix} \tag{3.13}$$

$$\mathbf{c} = \frac{1}{h^d}\mathbf{S}^{-1}\mathbf{d}. \tag{3.14}$$

Now, for any stencil, we have a formulation to calculate the coefficients, so long as $N > d$. This means that the stencil can contain any data sample, including values that are extremely far away from the current data sample. However, in practice values that are far away will have to be memorized for a long time, which is inefficient. Furthermore, stencils that are not centered around the current data samples will be less reliable due to artifacts. This is visible in a comparison of the backwards, central and forwards methods, see [15]. On close inspection of the backwards method it can be seen that that estimate of the derivative lags behind the true derivative. This causes an underestimation in increasing derivatives and an overestimate in decreasing derivatives. The forwards method on the other hand, has the opposite effect by leading the true derivative. The central method cancels both effects which has no lag or lead at all. This means that the central method is theoretically more accurate, even when the error order is the same as for the other methods.

3.2 Stencil choice

There are two more consequences of choosing a stencil. The first being that a stencil for the central method always increases with steps of 2: one extra sample from the past and one from the future. This is different from how the stencils of the backwards and forwards methods increase, which can increase in steps of 1. The consequence of this is that a fully central method cannot achieve every error order desirable, and, dependent on the height of the derivative, will always be an even error order, or uneven. The second consequence of stencil choice is how soon and with how much lag or lead a derivative can be calculated. The earliest derivative estimate that can be made is dependent on the data sample in the stencil that is the furthest in the past. This means that at the start of the signal, it might be necessary to wait a few time steps before all derivatives can be calculated as accurately as designed. Conversely, using the forwards method means having to wait for all future signals to have come in, creating a delay dependent on the data sample in the stencil that lies the furthest in the future. Also, given a finite signal, it is not possible to calculate the final derivatives as the signal will no longer have future values. The central method naturally inherits the downside of both the backwards and forwards method. The ideal method is therefore dependent on the circumstances, as sometimes a signal is fully available, called the offline setting, while at other times the signal comes in live, this will be called an online setting.

Since all data samples are available in an offline setting, all methods can be used without downsides. This means that the central method is preferable as there will be no lag. In an online setting, future samples are not available, meaning that the backwards method is the only method available. Finally, in any setting, it is advised to use data samples as close to the current data point as possible.

3.3 Summary

The finite difference technique is a direct derivation from the Taylor expansion. It is possible to estimate any derivative, provided there are more signal samples available than the order of the derivative. The accuracy of the derivative is dependent on the difference in number of used signal samples and derivative order and how close the signal samples are to the sample of interest which is dependent on choice and the time step between samples. For these reasons, the best estimate would in theory come from a central method utilizing as many samples as possible around the sample of interest.

Chapter 4

PREDNET

This chapter contains a background for predictive coding (PC) and for the PredNet model used in this thesis. It shows why predictive coding is an interesting method for forecasting time-series and why PredNet is chosen. PC is a broad term with different meanings between different fields of study. In neuroscience the term conveys a theory that the brain contains a model of the outside world that is constantly being generated and updated. The model itself generates predictions about future states or inputs which are compared with the true values to create a prediction error. The prediction errors are then used to update the internal model. The aim of the model is to make these prediction errors as small as possible. PC provides biologically plausible statistical models that aim to find the most likely causes that can explain the states it attempts to predict. However, the description for PC is broad and there exist many different implementations and hypotheses about PC [17]. Practically all modern versions of PC are based on the work of Rao & Ballard [2]. It was an influential paper that also inspired Active Inference [3], the theory that was used to create a velocity controller (chapter 2). The model of Rao & Ballard will be described and discussed in section 4.1. The different implementations are discussed in [7] where it was found that PredNet was a very promising model. PredNet will be used throughout this thesis and how it works, how it can be installed and used is described and discussed in section 4.2.

4.1 Predictive Coding

Rao & Ballard [2] poured the PC theory into practice by connecting several building blocks called "Predictive Estimator" in a hierarchical fashion. An overview of how these blocks were connected and what they contained is shown in fig. 4.1. Each PE sends a prediction downwards and receives an error signal from the bottom. If there is another block above in the hierarchy, then a prediction is received from above and the error is send back upwards. The main component of each block is the representation matrix $\mathbf{r}$ which holds a model of the state below. This means that the blocks attempt to predict the models of the blocks below, which reoccurs until the lowest block which tries to predict an input. This hierarchy makes it that higher blocks have more abstract views of the input. The predictions p are calculated mathematically:

$$p = f(\mathrm{U}\mathbf{r}) + \varepsilon = f(\sum_{j=1}^{k} \mathrm{U}_j r_j) \tag{4.1}$$

where p predicts either the input I or the $\mathbf{r}$ of the layer below. $\mathbf{U}$ is a weight matrix containing basis vectors that transforms the representation to the prediction and ϵ is stochastic noise or the difference between p and $f(\mathrm{U}\mathbf{r})$, in other words the error of the prediction. Finally, f is typically a sigmoidal function, for Rao & Ballard *tanh* was used.

The predictions are compared with the input to create the error $\mathbf{r} - \mathbf{r^{td}} = \varepsilon$ in what we will call the "error calculation" which is used to update the representation units. Updating is done similarly to modern neural networks by performing gradient descent on an optimization function which is directly derived from eq. (4.1):

$$\frac{dr}{dt} = -\frac{k_1}{2}\frac{\partial E}{\partial r} = \frac{k_1}{\sigma^2} U^{\mathrm{T}} \frac{\partial f^{\mathrm{T}}}{\partial \mathrm{x}} (\mathbf{I} - f(U\mathbf{r})) + \frac{k_1}{\sigma_{td}^2} \left(\mathbf{r}^{td} - \mathrm{r}\right) - \frac{k_1}{2} g'(\mathbf{r}). \tag{4.2}$$

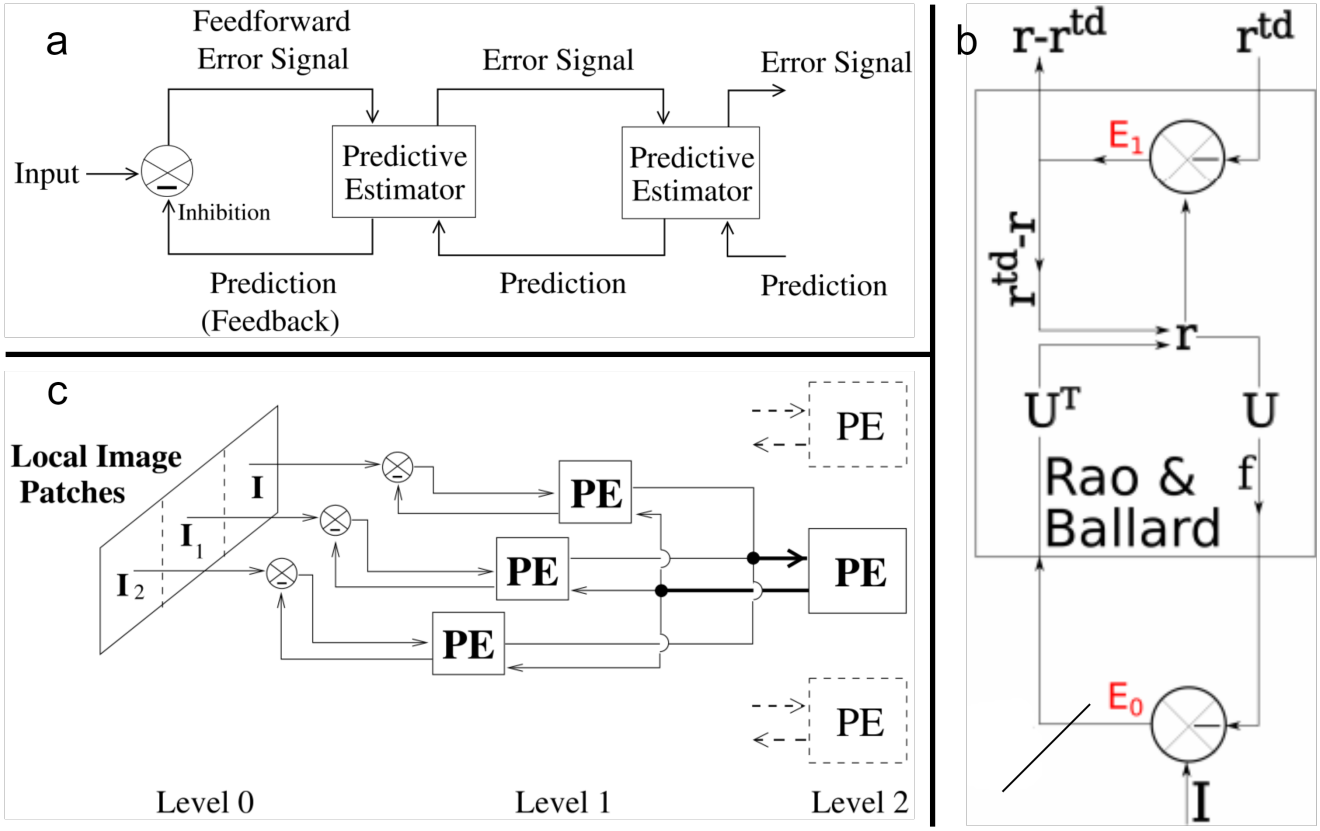

Figure 4.1: An overview of the design of Rao & Ballard [2]. a) An overview of all the components. Error signals are send upwards, predictions downwards. b) A look inside a Predictive Estimator (PE) showing the components it holds and how they are connected. c) An example of how the model can be built hierarchically. The input image is split into three regions; Receptive Fields, each with their own PE. These three PE's combine into a single PE in the layer above it.

This double use of eq. (4.1) is perhaps an even more important insight than the results obtained as it shows that perception and learning are intricately linked [3]. Through various hierarchical constructions, Rao & Ballard were able to reproduce several neurological phenomena, including the end-stopping effect, which was not reproduced before.

4.2 PredNet

PredNet is a type of recurrent neural network designed by Lotter, Kreiman & Cox [18] to mimic the Predictive Coding network from Rao & Ballard [2]. Lotter et al. took the same hierarchy and building blocks, but replaced the internal matrices with components from modern neural network frameworks. Besides interchanging the matrices, they also changed some of the connections and used a different optimization function, so that the workings are slightly different. How PredNet works is discussed in section 4.3 and how it is used in ..?

To use the PredNet model, some preparations need to be done. Firstly, the setup needs to work which is discussed in section 4.4. Secondly, the dataset to be used needs to have the correct input format. This format is described in section 4.5. Thirdly, to maximize the potential of the PredNet model, the hyper parameters need to be tuned. How this was done is discussed in section 4.6.

4.3 PredNet inner workings

In fig. 4.2 the two networks are shown side by side. The components used by Lotter et al. offer the same functionality as the matrices from Rao & Ballard. The representation matrix r is replaced by a convolutional LSTM (Long-Short-Term-Memory) layer. The convolutional aspect can be seen as

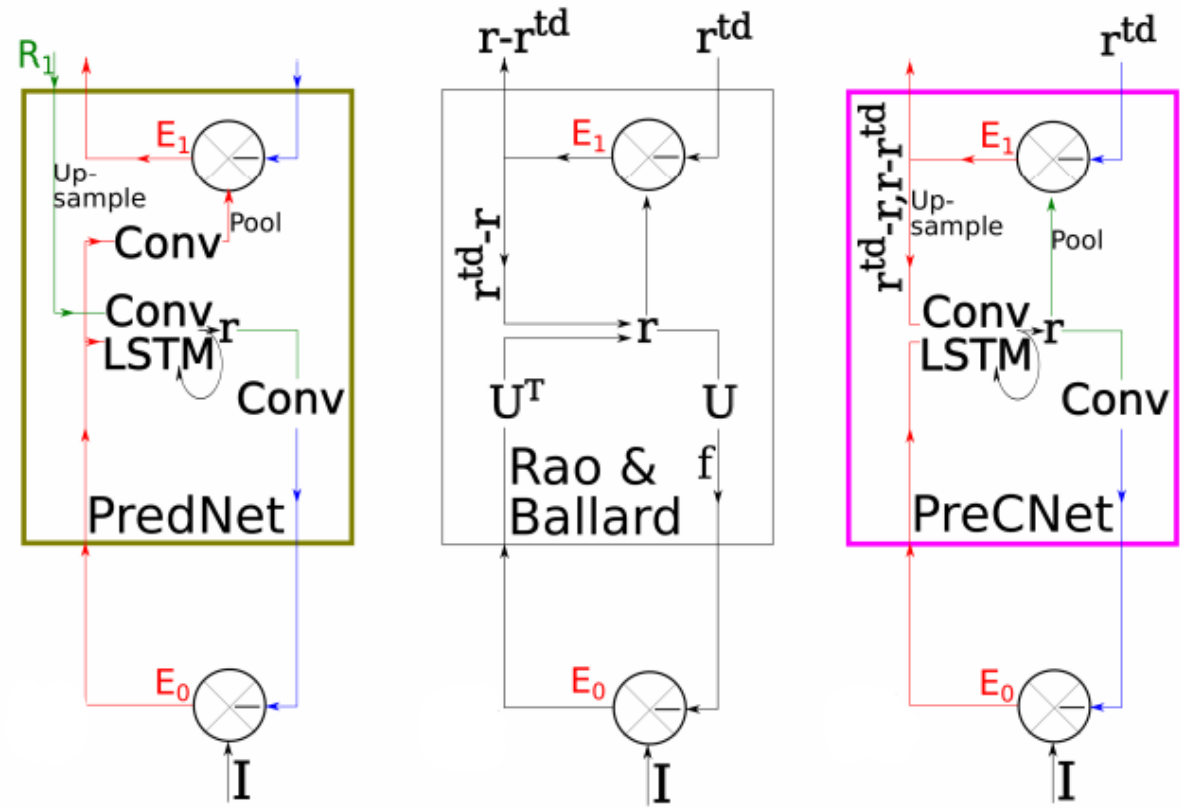

Figure 4.2: On the left an overview of PredNet, in the middle an overview of Rao & Ballard and on the right an overview of PreCNet. It can be seen that PredNet not only has different building blocks taken from DL, but also has different connections: E_0 connects to E_1, and R_1 connects to **r**. PreCNet has the same connections as Rao & Ballard and the same building blocks as PredNet.

analogous to the **U** matrix as they both have the aim to compress or expand a signal. The LSTM part is extended to also include a mathematical convolution part that combines two functions into a third function, while the LSTM part itself is able to hold the actual representation for a time-series. However, the signals that are being used by each part differ between the two networks.

PredNet has swapped two connections between components in comparison with the Rao & Ballard network. Firstly, the ε of a block connects to the error calculation of the block higher in the hierarchy, instead of to the representation of the block itself. The result is that the lowest error calculation at the input level is a comparison of a prediction with the input, while the error calculation at all other instances are comparisons of predicted error and actual error. The different error connections cause the error at the input level to be propagated through the entire hierarchy and each layer tries to predict the error of the layer below. This is in essence, similar to a Taylor expansion and is a big difference compared to the network from Rao & Ballard. The connections of Rao & Ballard are generally accepted and not often deviated from because it causes higher levels to have higher levels of abstractness. Nevertheless, the definition of PC does not include how errors and predictions are connected, making PredNet a viable and true PC network. Secondly, the representation connects to the representation of the block lower in the hierarchy instead of the error calculation in the same block. This means that every representation has some kind of idea of the representations higher up in the hierarchy, although this change seems to have only a marginal effect [19] and might just exist as a result of the changed error connections.

PredNet also uses a different optimization algorithm. Instead of using local prediction and error comparisons like Rao & Ballard, it uses one main optimization function that collects all the error terms in the network. Each LSTM unit then takes the partial derivative to that LSTM unit to update its model. In these frameworks it is possible to customize the optimization function and to choose which error terms are included and with what weight. PredNet was tested with two different combinations of

error terms: L_0 and L_{all}. L_0 only used the actual prediction error from the lowest layer at the input level ($\lambda_0 = 1, \lambda_{l>0} = 0$), while L_{all} set the weight for the error predictions higher in the hierarchy to a small fraction of the prediction error ($\lambda_0 = 1, \lambda_{l>0} = 0.1$). The paper showed that the L_0 algorithm worked best on most of the tests, which is not something the theory for PC would expect [20]. It is possible that this is due to the errors of the higher layers to represent error predictions rather than abstractness, but this has not been confirmed.

The results obtained by PredNet were state-of-the-art. The paper itself achieved this in video-frame prediction and steering angle estimation, but several other fields were also able to obtain similar state-of-the-art results [21], [22], [23]. It has also been shown to be able to reproduce several illusions and extra-classical receptive field effects [24], [25], similar to the effects Rao & Ballard's model was able to replicate. Since the release of the paper several extensions and improvements have been created [26], [27], [28].

4.4 Using PredNet (Python 3) with ROS Melodic/Kinetic

This thesis uses two components that need to be connected. Firstly, the Jackal simulation and its data through ROS and Gazebo. Secondly, the PredNet model using the Keras framework that uses the software language Python 3. Gazebo relies on Robotic Operating System (ROS) for all its communication. Both system versions are connected. Each version of Gazebo has a corresponding version of ROS or vice versa. Generally, ROS uses C++ for programming, although it also supports Python. However, older versions of ROS, like ROS Melodic or Kinetic, only support Python 2.7 intrinsically. Therefore, use the Keras framework, and thus PredNet, on these older versions of ROS, it is necessary to integrate Python 3 with those older versions. This integration is accomplished by using a catkin virtual environment. The latest ROS version (ROS Noetic) intrinsically supports Python 3, making the connection between ROS and the Keras framework much simpler.

The first step for integrating Python 3 with ROS is to make sure that all the programs are installed. Python 3 can be downloaded from https://www.python.org/downloads/. Choose the version that fits the dependencies of the packages you want to use. For example, PredNet made use of specific versions of the packages Keras, TensorFlow and hickle which required Python version 3.6. Then, a copy of ROS needs to be installed. Some operating systems have ROS preinstalled, for example Unix system Ubuntu 18 comes with ROS Melodic. ROS installations can be obtained at https://www.ros.org/blog/getting-started/. With ROS installed, the dependencies for the implementation need to be downloaded. In the case of the Jackal robot the following packages need to be installed:

```
ros-melodic-jackal-simulator
ros-melodic-jackal-navigation
ros-melodic-jackal-desktop
ros-melodic-velocity-controllers
```

. An easy way to install these packages is through a package manager like `apt-get` or `pip`. Finally, the catkin tools that will be used consists of the following packages and dependencies:

```
python-catkin-pkg
python-empy
python-nose
python-setuptools
libgtest-dev
build-essential
```

, which can be installed in the same way as the Jackal packages. More detailed instructions can be found at https://wiki.ros.org/catkin#Installing_catkin.

With the programs installed the next step is set up a configuration that allows ROS to make use of Python 3. It starts by building a standard catkin workspace using the following commands (http://wiki.ros.org/catkin/Tutorials/create_a_workspace):

```
$ source /opt/ros/melodic/setup.bash
$ mkdir -p ~/catkin\_ws/src
$ cd ~/catkin\_ws/
```

```
$ catkin\_make
```

This will create a standard ROS workspace with a `build`, `devel` and `src` folder. The `src` folder will contain the packages of the project. Each project requires the `CMakeLists.txt` and `package.xml` files. To make use of the catkin virtual environment, both files need to be modified. In `package.xml` a build dependency needs to be added:

```
<build_depend>catkin_virtualenv</build_depend>
```

, along with python dependencies, for example the mathematical python package Numpy requires:

```
<build_depend>python-numpy</build_depend>
<exec_depend>python-numpy</exec_depend>
```

. The file `CMakeLists.txt` needs three additional sections:

```
find_package(catkin REQUIRED ... catkin\_virtualenv ...)

catkin_generate_virtualenv()

catkin_install_python(
    PROGRAMS
    scripts/do_python_things
    DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION})
```

. The python scripts should not be made executable. The website https://github.com/locusrobotics/catkin_virtualenv provides more possibilities of the virtual environment under section "**Bundling virtualenv**". It is then possible to add a folder '`scripts`' to the package that can contain all the Python scripts. It should be noted that not every package needs to make use of a virtual environment. Packages that do not require Python 3 for example, could simply use the traditional dependencies. The final workspace should look like this:

```
Workspace
├─ build
├─ devel
└─ src
   ├─ package_1
   │  ├─ include
   │  ├─ launch
   │  ├─ msg
   │  ├─ scripts
   │  ├─ src
   │  ├─ CMakeLists.txt
   │  └─ package.xml
   └─ CMakeLists.txt
```

. Make sure that there is no file called `setup.py` present in the directories. That file can be part of a different solution for integrating Python 3 with ROS, but it breaks this solution.

4.5 The structure of PredNet input

This section provides an overview of the data structure used by PredNet. If one would like to make use of PredNet themselves this section could help with preparing the data necessary.

PredNet makes use of three data sets: a training set, a test set and a validation set. The purpose of these sets is as follows. The training set is used by the model to learn its parameters. The training set is

Dimension	5	4	3	2	1
Form	n_samples	n_columns	n_rows	n_channels	Data
Type	list	numpy.ndarray	numpy.ndarray	numpy.ndarray	numpy.uint8

Table 4.1: Table showing the data layout that PredNet depends upon for 'X' - the raw data. This data has 5 dimensions. Each dimension is related to something physical which is shown in '**Form**'. Each dimension is a specific object type in Python, shown in the column '**Type**'.

Dimension	3	2	1
Form	n_samples	sequence name	number within sequence
Type	list	numpy.bytes_	int

Table 4.2: Table showing the data layout that PredNet depends upon for `sources`. This data has 3 dimensions. Each dimension is related to something physical which is shown in '**Form**'. Each dimension is a specific object type in Python, shown in the column '**Type**'.

fed to the model and the errors that are created are then used to update the model. Normally, after the entire training set has been fed through the model, one epoch has passed. After every epoch, the model is validated with the validation set to make sure that the model does not overfit to the training data. Because the model has seen data samples from both the training and the validation set during training, neither of these sets can be used for experiments. For that reason there is a third set called the test set with data that the model has never seen before.

Each set consists of two multidimensional files. One file called '`X`' contains the raw data. The raw data consists of one 5-dimensional entry. The first dimension is related to the samples in the data set. Each entry of the first dimension is a signal sample. The second dimension are the columns of each sample. The third dimension are the rows of each column. The fourth dimension are the channels of row and finally, the fifth dimension are the data samples within the channels. This fifth dimension was originally an 8-bit integer, as PredNet was trained on RGB images. However, it is possible to change the bit size to suit the needs of the data. For this thesis a 32-bit integer was used (see section 4.6). The use of integers as input restricts the possible input types. Only input signals that are bounded are possible. Each dimension is a specific object type in Python. An overview is given in table 4.1. Besides the specific object types, the dimensions need to have a specific shape as well. Each layer of the PredNet model reduces the size of the rows and columns by a factor of two. This means that the model input must have a shape for which the number of rows and columns is divisible by 2^{l-1}, where l is the number of layers of PredNet. For example, if you have input data with 16 columns and 20 rows, you can use at most 2 layers.

The other file is called '`sources`'. Table 4.2 shows the structure of this file. '`sources`' contains sequence numbers which tell PredNet which data samples belong together in a single sequence. It contains one 3-dimensional entry that is related to '`X`'. The first dimension is similar to '`X`' and is also related to the samples in the data set. The second dimension is an arbitrary sequence name. The third dimension is the sequence number which is used to tell different sequences apart.

4.6 Tuning PredNet hyper parameters

Using neural networks requires choosing hyper parameters for the model. Hyper parameters can have a large impact on the performance of a neural network. The process of finding good hyper parameters is called tuning. Tuning was done for the following hyper parameters:

- Integer size
- Learning rate

. All tests concerning the hyper parameters have been performed on a sinusoidal signal containing Gaussian noise.

Bits	Steps	Stability
8	2.56e+02	Stable
16	6.55e+04	Stable
32	4.29e+09	Stable
64	1.84e+19	Unstable

Table 4.3: The relationship between bit size and the number of different values that are available is shown here. An integer size of 32 is used for the remainder of this thesis as it provides the highest precision of the stable options.

Setting	Stability	MSE
LR 0.01 - no schedule	Stable	1.672×10^{-5}
LR 0.01 - with schedule	Stable	1.727×10^{-5}
LR 0.1 - no schedule	Unstable	7.117×10^{-2}
LR 0.1 - with schedule	Unstable	1.545×10^{-5}

Table 4.4: Table showing the MSE obtained by the models trained with different learning rate settings when predicting the next frame. The final model at epoch 10000 was used. The model with a learning rate of 0.01 and no scheduler had the lowest MSE of the stable models.

The original PredNet model was trained on images, whose pixels generally have values represented in 8-bit integers, or values from 0 to 255. However, in this thesis we are interested in float values which require much more precision. Table 4.3 shows how the bit size of integers relate to precision that is possible. Ideally, a 64-bit integer would be used to convey the signal as it provides the maximum amount of precision. However, tests showed that the signal contained artifacts when 64-bit integers were used. As such, 32-bit integers were chosen as they have the highest precision without sacrificing stability.

The learning rate is an important hyper parameter for neural networks. It is directly related to how quickly a neural network changes its parameters. Setting it too high makes a model learn quickly, but it also overwrites what is has learned previously much faster. It makes it difficult for a model to reach the highest accuracy. A learning rate that is too slow will cause learning to take a very long time. To find a good learning rate, four different methods were used. Two methods used a static learning rate of 0.1 and 0.01, and two methods used a learning rate schedule , also starting at 0.1 and 0.01. A learning rate schedule allow the learning rate to be changed during training. The scheduler decreased the learning rate after 5000 epochs by a factor of $\sqrt{10}$. All models were trained to 10000 epochs. The learning process of the different method is shown in fig. 4.3. The figure shows that a learning rate of 0.1 can be unstable. A learning rate of 0.01 is always stable. Table 4.4 shows that scheduling makes no significant impact on how well the model learns. It is therefore decided to use a learning rate of 0.01 without scheduling to keep the model simple.

4.7 Summary

In neuroscience, predictive coding is regarded as a biologically plausible and accurate method for forecasting time-series. The general scheme was made effective by Rao & Ballard which was shown to mimic neurological phenomena and the creation of abstractness in the layers of the model. Inspired by their work, branches in different directions were created to improve upon the work of Rao & Ballard. From these branches, PredNet is a very promising one due to its utilization of the modern neural network frameworks. It used a slightly different scheme than Rao & Ballard, but was able to produce state-of-the-art results in the entire field of neural networks and predictive coding.

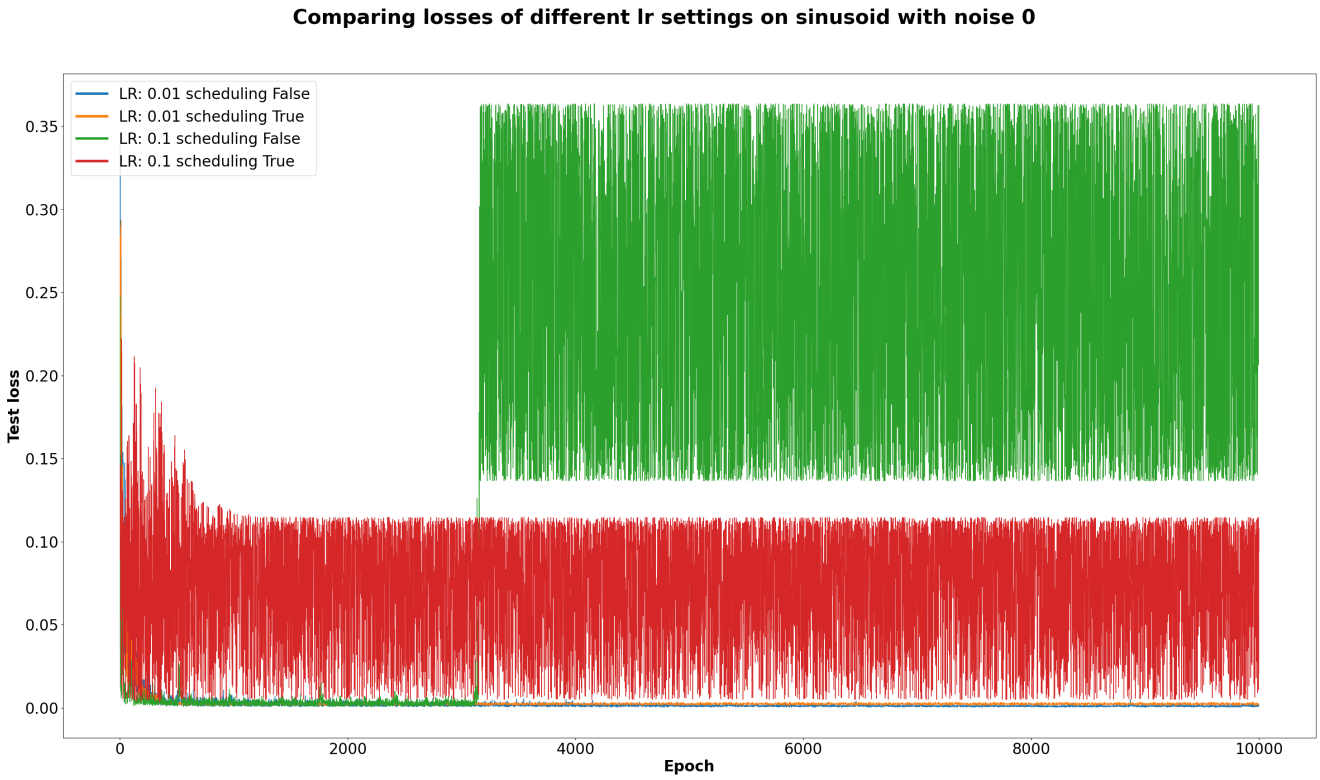

Figure 4.3: This figure shows the training losses for various learning rate strategies when PredNet is trained on a noisy sinusoid. The models with a learning rate of 0.1 show unstable behaviour, while the models with a learning rate of 0.01 are stable.

Chapter 5

METHODS

With the theoretical background obtained it is now possible to set up the experiments that should help in answering whether forecasts of PredNet can help improve the accuracy of estimated derivatives. The experiments are aimed at answering the question whether predicting future data points by use of PredNet help improve the accuracy of the generalized coordinates? First, the signal needs to be predicted and then secondly, the predicted signal can be used to determine derivatives. To make the experiments as realistic as possible, data from a simulated Jackal robot has been used, as described in section 5.1. All tests are conducted with as much similarity between the models as possible. The specifics of these are discussed in section 5.2. Testing of the signal prediction will be described in section 5.3, while the method for derivative testing is described in section 5.4.

5.1 Description of data

All experiments described in this section have been performed on data obtained by simulating the Jackal robot. Simulations were done through Gazebo 9.0 on an Ubuntu 18.04.6 operating system. It is possible to interact with Gazebo through ROS (Robot Operating System). A tutorial on how to set up the software is given in section 4.4. The Jackal robot which was used for the experiments in chapter 2 is available for Gazebo through the website of ClearPathRobotics. The data from the robot and how it was obtained will be described first. The data from the Jackal robot needs to suit the needs of PredNet which are described in section 4.5. This section ends with a description about how the data was transformed to PredNet input.

The Jackal robot has two main sources for positional data: ROS topics `/imu/data` and `/joint_states`. The sensors of the robot publish information at a rate of $50Hz$. The topic `/imu/data` contains information about the center of gravity of the robot; its orientation, angular velocity and covariance and linear acceleration and covariance. The topic `/joint_states` contains information about the joints of the robot. In the case of the standard Jackal robot, the joints consist of its four wheels. The Jackal controller (chapter 2) drives the wheel velocities. Therefore the wheel velocities from `/joint_states` were used in the experiments. Also, because PredNet input needs to be bounded (see section 4.5), the position data can not be used. Since the robot uses differential steering, the two left wheels are connected to each other, as well as the two right wheels, leaving only 2 unique joint states.

The data was obtained by commanding a Jackal robot in a Gazebo simulation. The Jackal robot can be commanded through three different ROS topics: `/jackal_aic/set_point`, `/jackal_left_wheel/command` and `/jackal_right_wheel/command` and `/cmd_drive`. The topic `/jackal_aic/set_point` is further processed by the self-made AIC node to drive the wheels. This is explained in chapter 2. The other nodes can be used to directly drive the wheels of the robot. Further details about the different topics can be found in section 2.2. The topic `/jackal_aic/set_point` was used to command the robot as it is easiest to command and provides more realistic joint states that are the result from the AIC node.

The topic `/jackal_aic_set_point` consists of a horizontal velocity x, a vertical velocity y and a rotational velocity ϕ. Since the Jackal robot always stays on the ground, y can remain 0, which means that each command needs to contain an x value and a ϕ value. Both values were limited: $-2 < x < 2$

Data set	Time recorded [s]	Data samples	Commands
Training	300	15214	153
Test	240	11980	120
Validation	60	2985	30

Table 5.1: An overview of the data sets that were used. The number of samples are not perfectly related to the recording time. This is because the time was measured with a stopwatch and the process of starting and stopping the measurements was a manual process.

and $-1 < \phi < 1$, which corresponded to a minimum and maximum wheel velocity of $-23\,rad/s$ and $23\,rad/s$ if both values are at their minimum or maximum, respectively.

Each command chose random values for x and ϕ to obtain a diverse data set containing both the settling mechanics of the controller as well as state transitions between commands. A new command was given every 2 seconds, which was shorter than the settling time.

To make use of the PredNet models we need to have three sets of data: a training set, a test set and a validation set, see section 4.5. For each of these sets the same routine was performed, but with a different duration so as to produce different amounts of data samples for each of the sets. This was done to make sure that each set also contained the start-up phase of the robot and to make continuity within the sets easier. Time was recorded with a stopwatch, because accuracy is not crucial and it was easier to use. An overview of the data sets is provided in table 5.1. After these routines, the topic data was stored in an external file.

The data obtained from the Jackal robot needed to be transformed for PredNet to use it, see section 4.5. This meant three things. First, the raw data needed to be put in the correct 5-dimensional shape. Since only two unique values are available, PredNet could only be a single layer. This would severely limit the power of PredNet as the strength of neural networks generally lies in having multiple layers. However, by duplicating the values it was possible to create a 2×2 matrix, allowing two PredNet layers. Second, the data needed to be discretized to 32-bit integers. Each signal sample was shifted, divided by the wheel velocity range, and then multiplied by the number of possible values:

$$y = \frac{x_{old} - x_{min}}{x_{max} - x_{min}} * s. \tag{5.1}$$

with y the discretized data, x_{old} the original data, x_{min} and x_{max} the minimum and maximum wheel velocity respectively, and s the number of possible values. Thirdly, a `source` file had to be created, linking the signal samples to sequences. The sequence number started at 0 and was increased by 1 with every command given, starting from the first command. The transformed data and the `sources` were both saved in external files to create the three data sets.

5.2 Model settings

After the data has been prepared, the model needs to be prepared. PredNet is a neural network which, by design, contains many hyper parameters that can be tuned to alter the performance. Tuning was done on two types of sinusoid: a clean version and a noisy version containing colored noise. In total 8 hyper parameters were tuned to prepare the models. Four of these parameters were determined by the circumstances of the model and the goal of the thesis to design an online predictor, and four parameters were determined by running various tests on the sinusoidal data, see section 4.6. An overview of all the final hyper parameters is shown in table 5.2.

Table 5.2 shows the settings used to simulate offline learning. These settings were kept constant for all experiments.

The model parameters are the number of epochs, the number of samples per epoch, the number of samples per validation and the batch size. In normal circumstances, training a single epoch means that a neural network has "seen" and trained on all available training data once. This means that the number of samples per epoch is simply equal to the number of samples in the training data set. Normally the same is true for the number of samples per validation, that is usually equal to the number of samples in

Table 5.2: Table showing the settings of the PredNet models.

Model parameters	Online	Offline
NO. Epochs	10000	20
NO. Samples per Epoch	1	500
NO. Samples per validation	1	100
Batch size	1	1

(a) Overview of the online and offline PredNet settings. These settings were not changed for any experiment.

Optimized parameters	
Integer size	32
Sample depth	2
NO. Frames	10
Learning rate	0.01

(b) Overview of the settings determined through optimization. These settings are used for all experiments following.

the validation data set. Furthermore, the batch size is a parameter that determines how many samples the model enters before updating itself. Setting this higher than 1 means that the model will aggregate results of several inputs before updating, resulting in some accuracy loss and a increase in training speed. However, training in an online setting is not a normal circumstance and requires some bending of the norm. To train and use a model in an online setting it was necessary to retrieve an outcome after every new sensory input. To do so, it was necessary to save the model after every training sample which can only be done after a validation step which is only done at the end of an epoch. This means that every epoch can only consist of a single training and validation sample with a batch size of 1 to perform online training. Finally, the number of epochs simply determines how long the model will keep training. Setting this higher will allow more training cycles, potentially increasing the performance. This parameter was set at 10000 for the online model as testing showed that training losses barely decreased anymore.

To keep the online and offline models on the same playing field, the parameters of the offline model also do not follow the norm and are instead designed to make sure that both models see an equal amount of training samples. The number of samples per epoch was chosen rather arbitrarily, it was simply made sure to be much larger than for the online setting and that the total number of training samples of the online model was divisible by this number. The number of samples per validation were then determined by using the ratio of number of data samples that both data sets have, see table 5.1. The batch size was kept the same to prevent accuracy loss and to keep both models similar. The parameters that have been optimized were the integer size and the learning rate, see section 4.6.

5.3 Predictions of PredNet

To get a sense of the accuracy of the PredNet models they have been compared with two other prediction methods: a very naive method of simply copying the input (I-C) and an auto regressive (AR) model. All models made use of the 'training' data set discussed in section 5.1. The PredNet models also made use of the 'validation' set and 'test' set. The validation set is used after every epoch as an interim performance test. For testing, only the left wheel velocities have been used, see fig. 5.1.

The parameters of the PredNet model have been discussed in section 5.2. With these settings and the data each PredNet model was trained 15 times as the training of a neural network is stochastic. After every 500 training samples the model parameters were stored to gain insight in the training process. This meant every 500^{th} epoch for the online model and every epoch for the offline model.

The AR model was given a lag of 10, to give it the same reach as the PredNet models. No moving average (MA) was used. Creating an AR model is, in contrast to a neural network, not stochastic, which meant that creating a single model was enough.

Copying the input is a very simple process and similar to the AR model has no variability.

Each model was used to predict up to 5 frames ahead. For every prediction level the MSE of each model was calculated by comparing the predictions with the true signal. In the case of the PredNet models this was done for every saving point, which gave 15 different MSEs per prediction level per saving point. The results of the intermediate saving points were used to show the progress of the PredNet models. The online PredNet models from all saving points were compared with each other, like the offline models. All results were checked for normality which was not always the case, meaning a

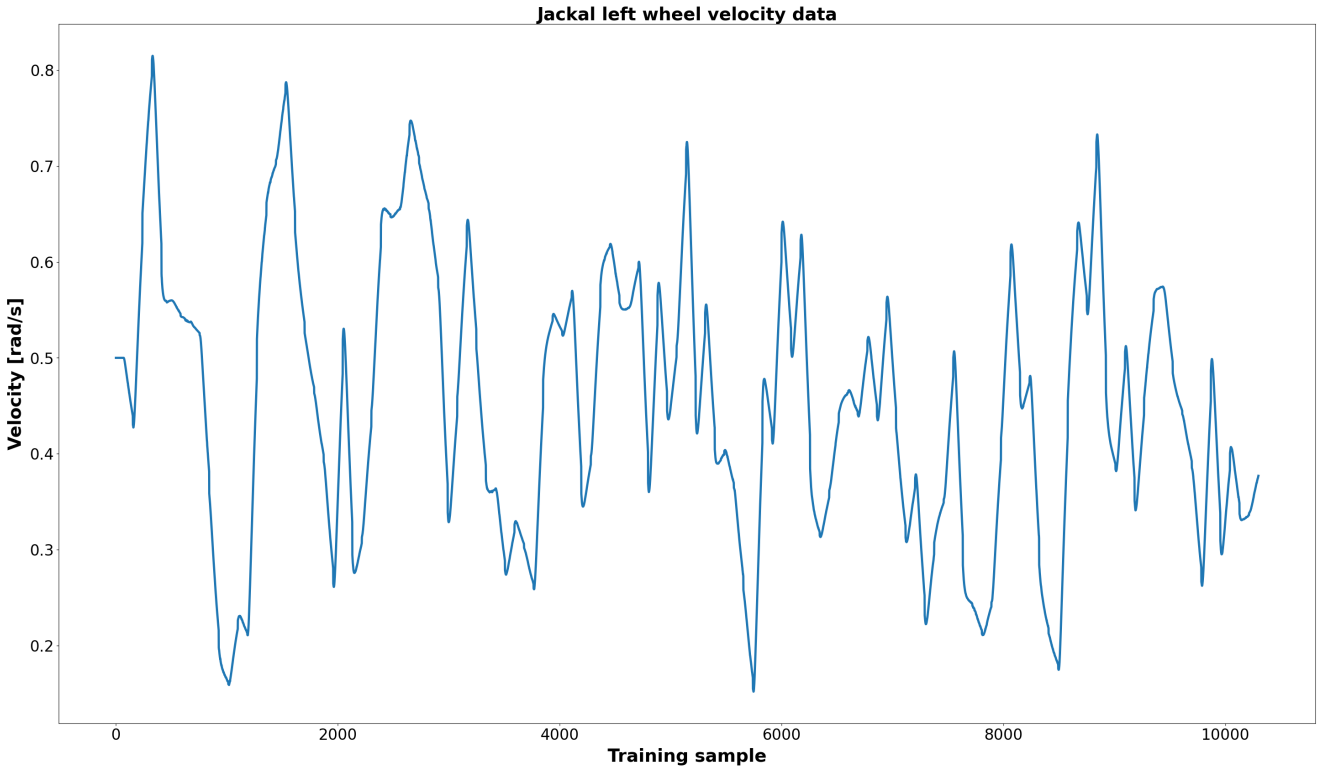

Figure 5.1: This figure shows the front left wheel velocities of the Jackal robot (ω_L); the training data for the PredNet models. Every two seconds the signal changes due to a new command being given to the robot. The changes can be steep due to the random choices of the commands.

non-parametric test had to be used. The results had a small population and results from different saving points were assumed to come from different distributions. This meant that Wilcoxon Rank-Sum test (or Mann-Whitney test) had to be used. The final saving points after 10000 training samples from both PredNet models were compared with the AR and copy-input models. All models were again checked on normality, which was sometimes violated. This meant a non-parametric test needed to be used for a small population of results that came from different distributions. These comparisons were also done using the Wilcoxon Rank-Sum test. However, when the AR model or copy-input model was involved, the comparison was adapted to a one-sample Wilcoxon Rank-Sum test, as the AR and copy-input models have no variance.

5.4 Derivatives of finite difference methods

5.4.1 Obtaining the derivatives

To get a sense of the usefulness of PredNet predictions and to determine whether the original method of the velocity controller can be improved, derivatives were calculated using the predictions from PredNet. Four variants of the finite difference method were used and compared against each other. Two variants used the backwards method and two variants used the central method. In both cases one of the variants used as few data points as possible, keeping the error order as low as possible, while the other variant attempted to have an error order as high as possible. Forwards methods were not tested because they only use predicted data which will never be more accurate than the true data, making it strictly worse than the backwards method. An overview of the variants is shown in table 5.3. There were 30 outcomes for the central derivatives due to 15 outcomes per PredNet model. The input-copy model was not evaluated as the derivatives would always be zero and thus meaningless. The AR model was not evaluated, because that model can not be updated once it has been established. Since the backwards methods make no use of the predictions, the outcomes are always the same and contain no variance.

Method	Explanation
B-1	Backwards method with minimum amount of data points. All error terms equal to $\mathcal{O}(h)$.
B-F	Backwards method utilising the maximum number of data points. Error order starts at $\mathcal{O}(h^5)$ and decreases by 1 with each derivative.
C-1	Central method with minimum amount of data points. Error order starts at $\mathcal{O}(h^2)$ and increases by 1 with each derivative .
C-F	Central method utilising the maximum number of data points. Error order starts at $\mathcal{O}(h^{10})$ and decreases by 1 with each derivative.

Table 5.3: Explanations for the finite difference methods used in the experiments for determining the derivatives of the signal.

All obtained derivatives were compared against a "true" derivative to obtain an MSE. The "true" derivative was determined by a **C-F** method used on the true signal, as there was no measured acceleration. Then, comparisons were made between the different variants and between the PredNet models. For each variant the PredNet models were compared separately. Not all outcomes were normally distributed, so non-parametric tests were used. Wilcoxon tests have been used because the outcomes are from different distributions and have a small population. Because the backwards methods contain no variance, a one-sampled Wilcoxon Signed-rank test was used when the backwards methods were compared. In other cases the Wilcoxon Rank-sum test was used. Comparison between the PredNet models was only done for the central methods as there is no difference between the outcomes of **B-1** from online and offline PredNet, nor between the outcomes of **B-F**. These comparisons also used the Wilcoxon Rank-sum test.

5.4.2 Analysis of error contribution

In the case of the central methods, the MSE of the derivatives can be related mathematically to the error from the predictions. Equation (4.1) shows that the value of a future data point consists of a predicted value and a prediction error. This prediction error is propagated through the finite difference methods, separate from the error terms of the finite difference methods themselves. Table 5.4 shows how much each data point in the stencil contributes to determining the derivatives. By combining the weights corresponding to the predicted values with the MSE of the frame predictions from table 6.1 the propagated error can be calculated. For difference method **C_1** and model P-ON this becomes:

$$\begin{bmatrix} \frac{1}{2h} & 0 & 0 & 0 & 0 \\ \frac{4}{3h^2} & \frac{-1}{12h^2} & 0 & 0 & 0 \\ \frac{-13}{8h^3} & \frac{1}{h^3} & \frac{-1}{8h^3} & 0 & 0 \\ \frac{-122}{15h^4} & \frac{169}{60h^4} & \frac{-2}{5h^4} & \frac{7}{240h^4} & 0 \\ \frac{323}{48h^5} & \frac{-13}{2h^5} & \frac{87}{32h^5} & \frac{-19}{36h^5} & \frac{13}{288h^5} \end{bmatrix} \begin{bmatrix} MSE_{f1} \\ MSE_{f2} \\ MSE_{f3} \\ MSE_{f4} \\ MSE_{f5} \end{bmatrix} = \begin{bmatrix} \varepsilon_{f1} \\ \varepsilon_{f2} \\ \varepsilon_{f3} \\ \varepsilon_{f4} \\ \varepsilon_{f5} \end{bmatrix}. \tag{5.2}$$

Another analysis was performed to evaluate the contribution of the command transitions in the signal. The transition periods were found for the first derivative by inspecting the true derivative. Each transition period was found to have a duration of 6 frames for the first derivative. A transition occurred every 100 samples, meaning that transitions take up around 6% of the total signal. The transition periods found for the first derivative were used for all derivatives and the same samples were used for both the true derivative and the model derivatives as this is how the original MSE is also calculated. For the transition periods the MSE was calculated separately and compared with the MSE over the whole signal by compensating for the difference in the number of samples.

y_{k-5h}	y_{k-4h}	y_{k-3h}	y_{k-2h}	y_{k-h}	y_k
0	0	0	0	$\frac{-1}{h}$	$\frac{1}{h}$
0	0	0	$\frac{1}{h^2}$	$\frac{-2}{h^2}$	$\frac{1}{h^2}$
0	0	$\frac{-1}{h^3}$	$\frac{3}{h^3}$	$\frac{-3}{h^3}$	$\frac{1}{h^3}$
0	$\frac{1}{h^4}$	$\frac{-4}{h^4}$	$\frac{6}{h^4}$	$\frac{-4}{h^4}$	$\frac{1}{h^4}$
$\frac{-1}{h^5}$	$\frac{5}{h^5}$	$\frac{-10}{h^5}$	$\frac{10}{h^5}$	$\frac{-5}{h^5}$	$\frac{1}{h^5}$

(a) The finite difference matrix corresponding to method **B_1**

y_{k-5h}	y_{k-4h}	y_{k-3h}	y_{k-2h}	y_{k-h}	y_k
$\frac{-1}{5h}$	$\frac{5}{4h}$	$\frac{-10}{3h}$	$\frac{5}{h}$	$\frac{-5}{h}$	$\frac{137}{60h}$
$\frac{-5}{6h^2}$	$\frac{61}{12h^2}$	$\frac{-13}{h^2}$	$\frac{107}{6h^2}$	$\frac{-77}{6h^2}$	$\frac{15}{4h^2}$
$\frac{-7}{4h^3}$	$\frac{41}{4h^3}$	$\frac{-49}{2h^3}$	$\frac{59}{2h^3}$	$\frac{-71}{4h^3}$	$\frac{17}{4h^3}$
$\frac{-2}{h^4}$	$\frac{11}{h^4}$	$\frac{-24}{h^4}$	$\frac{26}{h^4}$	$\frac{-14}{h^4}$	$\frac{3}{h^4}$
$\frac{-1}{h^5}$	$\frac{5}{h^5}$	$\frac{-10}{h^5}$	$\frac{10}{h^5}$	$\frac{-5}{h^5}$	$\frac{1}{h^5}$

(b) The finite difference matrix corresponding to method **B_F**

y_{k-5h}	y_{k-4h}	y_{k-3h}	y_{k-2h}	y_{k-h}	y_k	y_{k+h}	y_{k+2h}	y_{k+3h}	y_{k+4h}	y_{k+5h}
0	0	0	0	$\frac{-1}{2h}$	0	$\frac{1}{2h}$	0	0	0	0
0	0	0	$\frac{-1}{12h^2}$	$\frac{4}{3h^2}$	$\frac{-5}{2h^2}$	$\frac{4}{3h^2}$	$\frac{-1}{12h^2}$	0	0	0
0	0	$\frac{1}{8h^3}$	$\frac{-1}{h^3}$	$\frac{13}{8h^3}$	0	$\frac{-13}{8h^3}$	$\frac{1}{h^3}$	$\frac{-1}{8h^3}$	0	0
0	$\frac{7}{240h^4}$	$\frac{-2}{5h^4}$	$\frac{169}{60h^4}$	$\frac{-122}{15h^4}$	$\frac{91}{8h^4}$	$\frac{-122}{15h^4}$	$\frac{169}{60h^4}$	$\frac{-2}{5h^4}$	$\frac{7}{240h^4}$	0
$\frac{-13}{288h^5}$	$\frac{19}{36h^5}$	$\frac{-87}{32h^5}$	$\frac{13}{2h^5}$	$\frac{-323}{48h^5}$	0	$\frac{323}{48h^5}$	$\frac{-13}{2h^5}$	$\frac{87}{32h^5}$	$\frac{-19}{36h^5}$	$\frac{13}{288h^5}$

(c) The finite difference matrix corresponding to method **C_1**

y_{k-5h}	y_{k-4h}	y_{k-3h}	y_{k-2h}	y_{k-h}	y_k	y_{k+h}	y_{k+2h}	y_{k+3h}	y_{k+4h}	y_{k+5h}
$\frac{-1}{1260h}$	$\frac{5}{504h}$	$\frac{-5}{84h}$	$\frac{5}{21h}$	$\frac{-5}{6h}$	0	$\frac{5}{6h}$	$\frac{-5}{21h}$	$\frac{5}{84h}$	$\frac{-5}{504h}$	$\frac{1}{1260h}$
$\frac{1}{3150h^2}$	$\frac{-5}{1008h^2}$	$\frac{5}{126h^2}$	$\frac{-5}{21h^2}$	$\frac{5}{3h^2}$	$\frac{-5269}{1800h^2}$	$\frac{5}{3h^2}$	$\frac{-5}{21h^2}$	$\frac{5}{126h^2}$	$\frac{-5}{1008h^2}$	$\frac{1}{3150h^2}$
$\frac{41}{6048h^3}$	$\frac{-1261}{15120h^3}$	$\frac{541}{1120h^3}$	$\frac{-4369}{2520h^3}$	$\frac{1669}{720h^3}$	0	$\frac{-1669}{720h^3}$	$\frac{4369}{2520h^3}$	$\frac{-541}{1120h^3}$	$\frac{1261}{15120h^3}$	$\frac{-41}{6048h^3}$
$\frac{-41}{7560h^4}$	$\frac{1261}{15120h^4}$	$\frac{-541}{840h^4}$	$\frac{4369}{1260h^4}$	$\frac{-1669}{180h^4}$	$\frac{1529}{120h^4}$	$\frac{-1669}{180h^4}$	$\frac{4369}{1260h^4}$	$\frac{-541}{840h^4}$	$\frac{1261}{15120h^4}$	$\frac{-41}{7560h^4}$
$\frac{-13}{288h^5}$	$\frac{19}{36h^5}$	$\frac{-87}{32h^5}$	$\frac{13}{2h^5}$	$\frac{-323}{48h^5}$	0	$\frac{323}{48h^5}$	$\frac{-13}{2h^5}$	$\frac{87}{32h^5}$	$\frac{-19}{36h^5}$	$\frac{13}{288h^5}$

(d) The finite difference matrix corresponding to method **C_F**

Table 5.4: An overview of the finite difference matrices used. An explanation for each method is given in table 5.3.

5.5 Summary

To answer the research question several experiments will be performed on a wheel velocity time series from a simulated Jackal robot. Two types of PredNet models are trained on that data: one model is trained in an online setting and the other is trained in an offline setting. Furthermore, an auto regressive model is trained on the same data. The first experiment is concerned with the accuracy of the forecasts from the PredNet models and tests this accuracy by comparing the predictions from the PredNet models with the predictions from the autoregressive model as well as a naive method that predicts its own input. To test the usefulness of the forecasts as well as the accuracy of different finite different methods a second experiment is set up. Four finite difference methods are used: a backwards method using as few past samples as possible, a backwards method using as many past samples as possible, a central method using as few past samples and samples predicted by the PredNet models as possible and a central method using as many past samples and samples predicted by the PredNet models as possible. The derivatives are compared against a derivative obtained from a central method using as many samples as possible that all come from the true signal.

Chapter 6

RESULTS

This chapter will provide the results of the experiments described in chapter 5. The results will be linked to the research questions to draw conclusions.

6.1 Predictions of PredNet

For online PredNet, significant differences have been found between models with consecutive training sample intervals for the models trained with 500 and 1000, 1000 and 1500 and 1500 and 2000 samples. Offline PredNet only showed significant differences between models trained with 500 and 1000 and 1000 and 1500 samples. More significant differences were found when looking at non-consecutive intervals. However, after 5000 training samples, there were almost no significant differences to be found anymore with models trained with more samples. This is where the training did not improve much anymore as can be seen in fig. 6.1. Online PredNet showed exceptions for the model trained with 9500 samples, which was an unusually strong model. It had significant differences between the models trained with 5000, 5500, 6000, 6500, 8000 and 9000 samples. Offline PredNet showed exceptions for the model trained with 8000 samples, which was significantly different from the models trained with 9000 and 9500 samples.

When comparing online and offline PredNet, significant differences were found between the models trained with 4500, 7000, 8500 and 9500 samples.

Predictions from the final models are shown in table 6.1. It is clear that the AR model produces the most accurate predictions, regardless of how many frames ahead the predictions are. When increasing the number of frames the MSE of the I-C and AR models both increase with a factor between 1.5 and 3, while PredNet does not have such a constant factor and instead has more variability in how well it performs.

6.2 Derivatives of finite difference methods

To get a sense of the usefulness of PredNet predictions and to determine whether the original method of the velocity controller can be improved.

When comparing the different finite difference methods within the same model, almost all comparisons are statistically significant. For both PredNet versions there was no significant difference between B-1 and B-F and C-1 and C-F for derivative 5, as the results obtained from these methods were exactly the same (the stencil used to create the Taylor matrix is the same for this derivative). Furthermore, offline PredNet had no significant difference for C-1 vs C-F on derivative 1 and 2 and no significant difference between B-F and C-F for derivative 3.

No significant differences between the differently trained models were found.

Higher order derivatives had such a high error that they are not useful in practice ($> 10^3$).

The results of the contribution analysis are shown in table 6.3 and section 6.2. The prediction errors contribute less than 1% to the total derivative error for the first derivative, and less than 0.002% to the error of the higher order derivatives. In contrast, the error accumulated during the transitions between commands to the robot are able to explain up to 99% of the error in the derivatives.

Frames	Method	MSE	SD
1	P-ON	1.599×10^{-07}	8.357×10^{-08}
	P-OFF	1.844×10^{-07}	1.224×10^{-07}
	I-C	9.209×10^{-07}	0
	AR	4.169×10^{-08}	0
2	P-ON	1.469×10^{-06}	4.925×10^{-07}
	P-OFF	2.022×10^{-06}	1.270×10^{-06}
	I-C	2.702×10^{-06}	0
	AR	1.188×10^{-07}	0
3	P-ON	1.085×10^{-06}	4.355×10^{-07}
	P-OFF	1.252×10^{-06}	6.363×10^{-07}
	I-C	5.344×10^{-06}	0
	AR	2.657×10^{-07}	0
4	P-ON	5.527×10^{-06}	1.550×10^{-06}
	P-OFF	7.846×10^{-06}	4.562×10^{-06}
	I-C	8.861×10^{-06}	0
	AR	5.197×10^{-07}	0
5	P-ON	3.320×10^{-06}	1.444×10^{-06}
	P-OFF	3.602×10^{-06}	1.468×10^{-06}
	I-C	1.327×10^{-05}	0
	AR	9.162×10^{-07}	0

Table 6.1: **P-ON:** online PredNet, **P-OFF:** offline PredNet, **I-C:** Input copy, **AR:** Auto regressive Model. Table showing the mean MSE and standard deviation of multiple models for predictions from 1 to 5 frames ahead. Boxes in cyan denote the model with the lowest error per prediction level. Note that "Input copy" and "AR" do not have variation in their models and thus have no standard deviation.

Finite method \ Predictor	**Online PredNet**	**Offline PredNet**
B-1	$7.969 \times 10^{-03} \pm 0.000 \times 10^{+00}$	
B-F	$9.461 \times 10^{-03} \pm 1.735 \times 10^{-18}$	
C-1	$8.460 \times 10^{-03} \pm 3.519 \times 10^{-04}$	$8.829 \times 10^{-03} \pm 8.290 \times 10^{-04}$
C-F	$8.724 \times 10^{-03} \pm 2.970 \times 10^{-04}$	$8.838 \times 10^{-03} \pm 4.600 \times 10^{-04}$

Table 6.2: This table shows the MSE obtained determining the 1^{st} derivative for the different finite difference methods. The standard backwards methods has the lowest MSE of all methods.

Difference method	**C_1**		**C_F**	
Predictor	**P-ON**	**P-OFF**	**P-ON**	**P-OFF**
1^{st} derivative	2.261×10^{-5}	3.283×10^{-5}	4.814×10^{-5}	8.829×10^{-5}
2^{nd} derivative	1.432×10^{-3}	1.820×10^{-3}	4.937×10^{-4}	1.558×10^{-3}
3^{rd} derivative	6.728×10^{-1}	1.139	1.300	2.232
4^{th} derivative	7.794×10^{1}	1.370×10^{2}	1.022×10^{2}	1.795×10^{2}
5^{th} derivative	1.281×10^{4}	2.226×10^{4}	1.281×10^{4}	2.226×10^{4}

Table 6.3: This table shows the absolute contribution of prediction error to derivative error. It shows that the contribution of the prediction errors to the derivative errors are very small and have little impact.

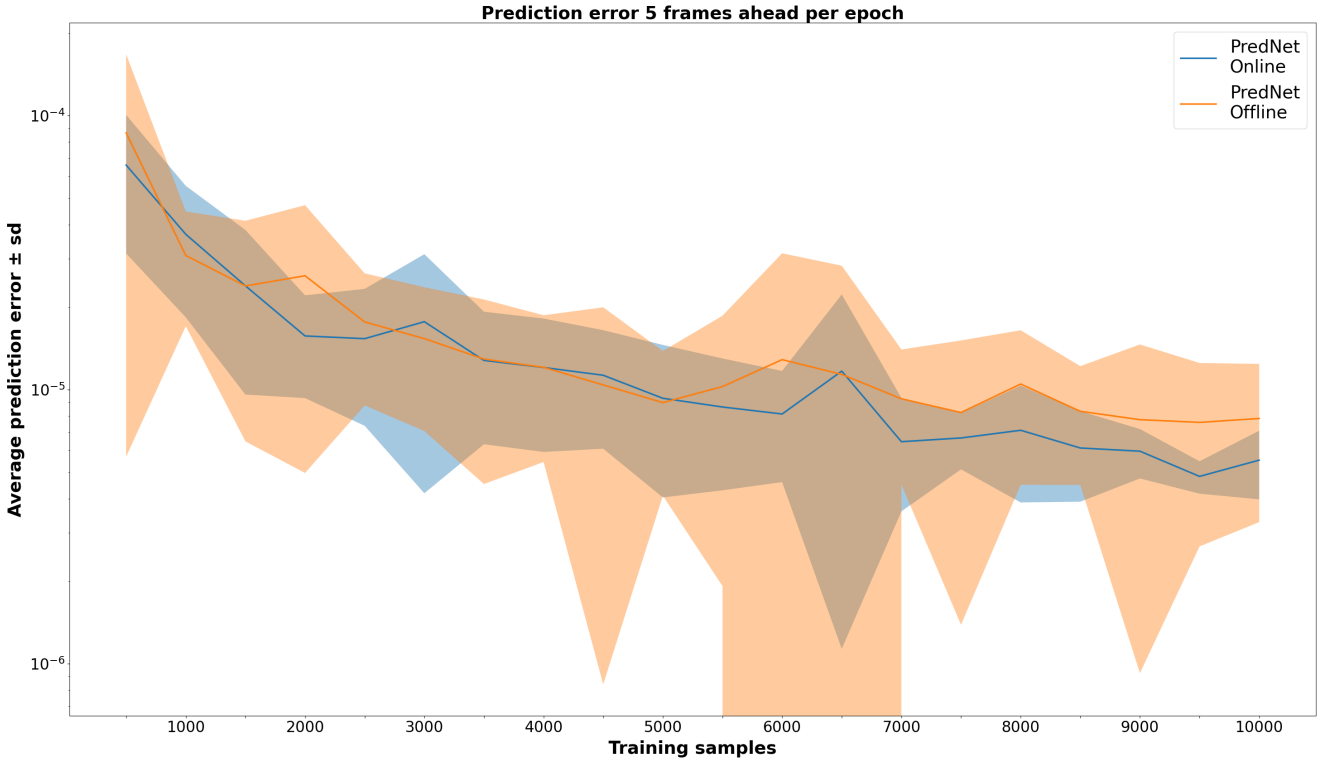

Figure 6.1: This figure shows the error of the prednet model predictions after having trained on various amounts of training data. The predictions of the models were 5 frames into the future. Due to the logarithmic y-axis, the standard deviation sometimes goes to 0 when the standard deviation is larger than the average.

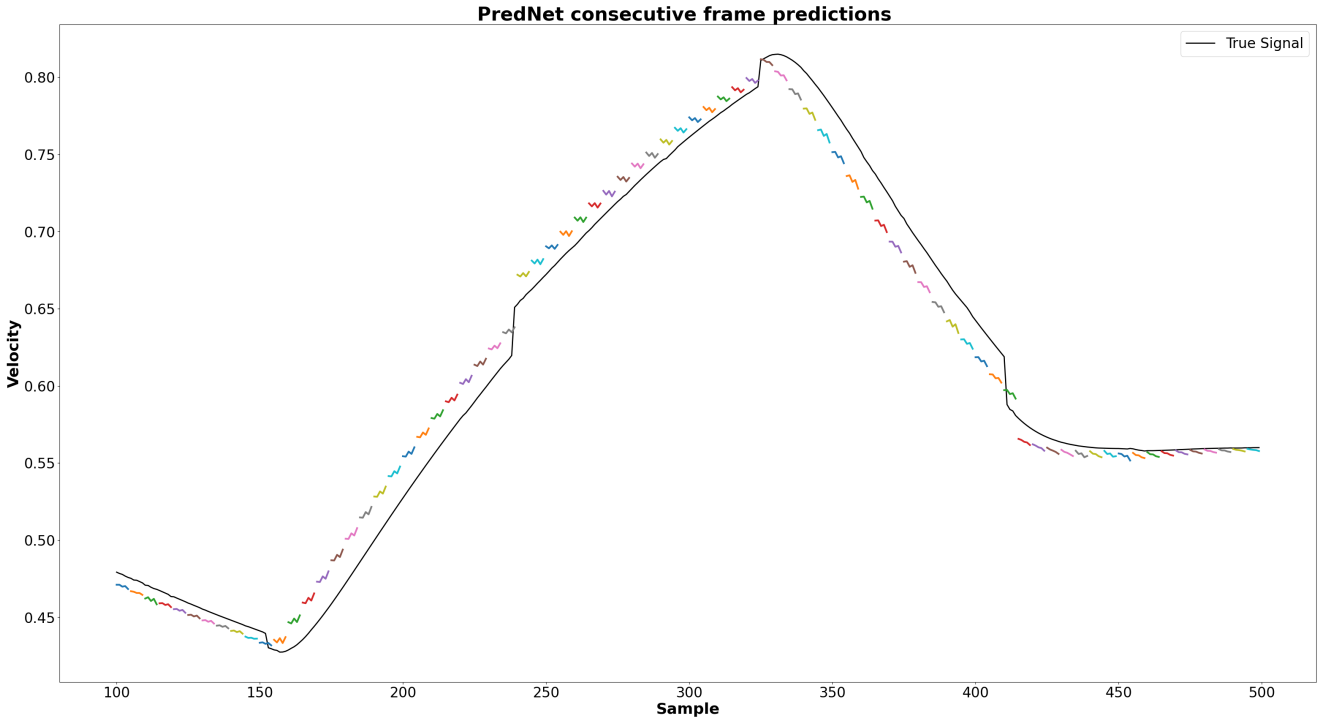

Figure 6.2: This figure shows the full 5 frame prediction from a single PredNet output for a portion of the signal. It shows that predict

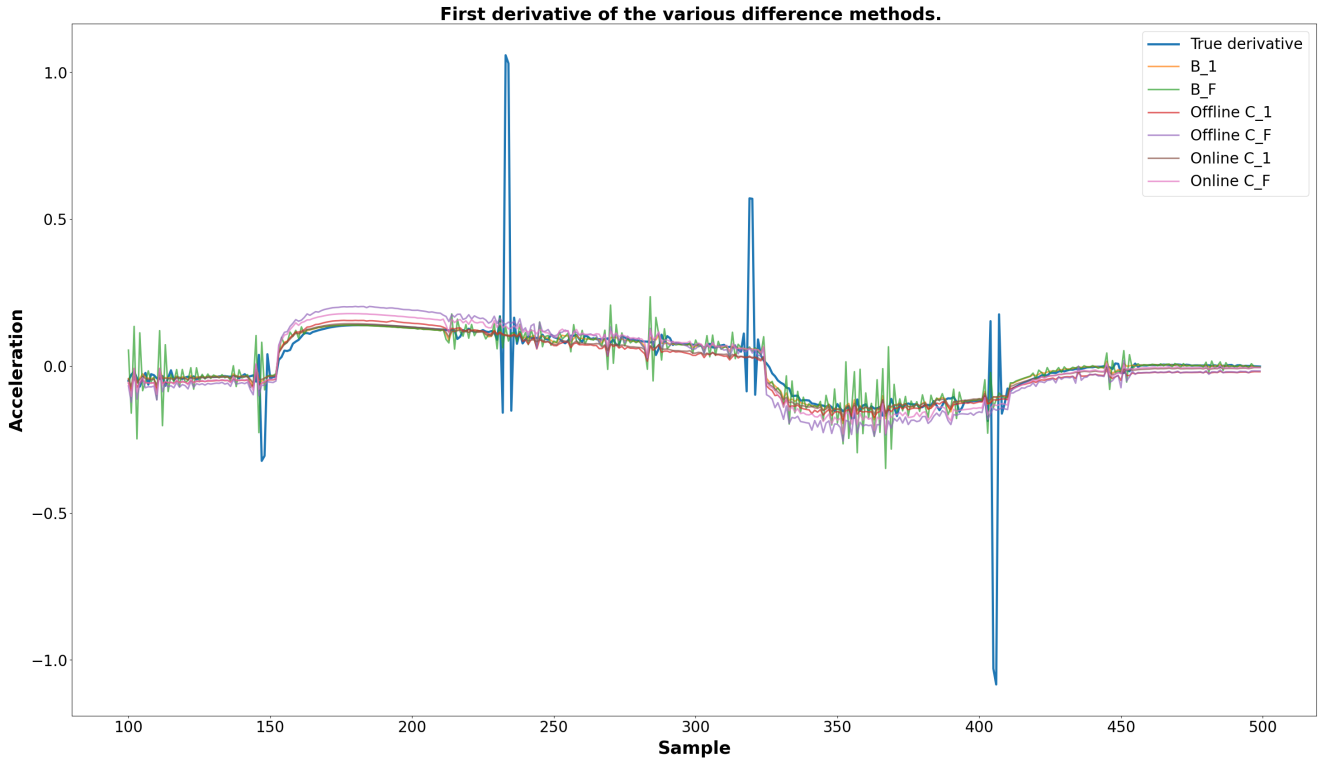

Figure 6.3: This figure shows the estimated first derivative of the finite difference methods as well as the true derivative. It can be seen that the central methods generally overestimate the derivative. The B_F method shows instability in between transitions and none of the methods are able to estimate the derivative around the transition periods accurately.

Method	**B_1**	**B_F**	**C_1**		**C_F**	
			P-ON	**P-OFF**	**P-ON**	**P-OFF**
1^{st} der.	7.792×10^{-03}	7.886×10^{-03}	7.729×10^{-03}	7.758×10^{-03}	7.385×10^{-03}	7.434×10^{-03}
	97.78%	83.35%	91.36%	87.86%	84.65%	84.12%
2^{nd} der.	$9.361 \times 10^{+01}$	$9.422 \times 10^{+01}$	$9.432 \times 10^{+01}$	$9.387 \times 10^{+01}$	$9.455 \times 10^{+01}$	$9.399 \times 10^{+01}$
	98.86%	71.86%	92.17%	89.30%	90.06%	87.70%
3^{rd} der.	$3.125 \times 10^{+05}$	$3.466 \times 10^{+05}$	$3.118 \times 10^{+05}$	$3.115 \times 10^{+05}$	$3.163 \times 10^{+05}$	$3.151 \times 10^{+05}$
	98.63%	63.03%	82.98%	79.67%	63.96%	56.80%
4^{th} der.	$1.325 \times 10^{+10}$	$1.346 \times 10^{+10}$	$1.333 \times 10^{+10}$	$1.329 \times 10^{+10}$	$1.337 \times 10^{+10}$	$1.332 \times 10^{+10}$
	99.26%	97.62%	89.06%	88.58%	85.53%	84.76%
5^{th} der.	$1.943 \times 10^{+13}$	$1.943 \times 10^{+13}$	$1.979 \times 10^{+13}$	$1.967 \times 10^{+13}$	$1.979 \times 10^{+13}$	$1.967 \times 10^{+13}$
	95.90%	95.90%	56.97%	47.84%	56.97%	47.84%

Table 6.4: Absolute and relative contribution of the transition periods in the signal to the derivative error. It can be seen that the transition periods are responsible for a large part of the derivative error.

Chapter 7

DISCUSSION

Now that the experiments have been performed, the research questions can be answered. The aim of this thesis was to discover whether predicted data from PredNet can help in improving the accuracy of the generalized coordinates. This question was split up to learn more about the accuracy and the usefulness of the predictions and whether more data points in the finite difference method could improve its performance. In section 7.1 the accuracy is discussed, section 7.2 contains a discussion about usefulness of PredNet predictions and the finite difference methods are discussed in section 7.3. Finally, the choice of data set and its consequences will be discussed in section 7.4.

7.1 PredNet accuracy

The predictions from PredNet were found to be between 3 and 10 times less accurate than the AR model, but between 1 and 5 times more accurate than copying the input. This agrees with literature [6] which shows that statistical regression is only beat with well-adjusted neural networks. Given that PredNet is not a cutting edge model anymore and that the models have not been trained in an optimal setting this outcome supports that view. Improvements can be made by using neural networks that are better suited for forecasting [29], by combining AR models with neural networks in a single model [30], by increasing the duration of training samples, or by increasing the sample depth to have more information over time.

PredNet might not be the most optimal choice when it comes to maximizing prediction accuracy, but it is capable of learning in an online fashion. Research has shown [31] that learning in an online fashion is capable of performing similarly or better than offline learning. The results of this thesis support that idea as the online and offline PredNet models overall showed little difference, except for some instances where the online model performed slightly better than the offline model. This can be attributed to the stochastic nature of the learning process, but also to the fact that the online version used more validation samples during learning. In terms of prediction performance, it seems that the number of samples that have been used is a larger contributor than the number of epochs or time spent learning. However, a large difference was found in the learning speed. Because the online method could only learn one sample per epoch and evaluation after every epoch is a slow process, the whole process of training the network was much slower than the offline model. A solution to this problem could be to train the network beforehand on a different set of data before using it in an online fashion.

7.2 PredNet usefulness

The usefulness of PredNet is mainly determined by the results of the derivatives. The results show little difference between the different methods when looking at the full error, but suggest that the simplest method (B_1) is the most accurate. Upon evaluating the errors, a majority of the error in derivative estimation comes from the transition between robot commands. This is unsurprising as the changes in command provide an immediate and non-smooth change in the velocity signal. It is therefore impossible for the backwards methods to see this change, but the central methods are dependent on the predictions from the PredNet models which are also unable to account for such a sudden change, meaning that neither method is capable of obtaining accurate derivatives around the transition points.

After subtracting the error due to the transitions, the left-over errors are several orders of magnitude smaller. However, even in this case, the second derivative will not be accurate enough to be useful in a practical setting as the error is already greater than 1. If we compare the left-over errors with the errors due to the predictions we find that even now, the predictions are not a large contributor to the error in the derivatives. Other sources for error are the lag due to using the backwards method and the general error due to estimating a derivative with a finite number of samples.

Aside from the derivatives there was another aspect contributing to the usefulness of PredNet predictions, namely its practicality. Each learning step of the online PredNet model took longer than the sampling time of the sensors. This means that if used in an online setting the model would start to lag behind or that samples would have to be skipped. Skipping samples would not be a preferred solution as it already takes much time for the model to reach accurate predictions and that would only increase that time. A more preferable solution would be to optimize the learning process to go faster which can be done in a few ways, depending on where the bottleneck lies. Firstly, the model itself can be optimized for processing and evaluating single samples as currently many models are optimized for processing large quantities of data. Many implementations of neural networks require large and intricate models for which you need to have and process large amounts of data. Forecasting time-series however, is a relatively small and simple task that does not have those requirements and could therefore be adapted to better fit the purpose for online training. Secondly, the process of retrieving and transforming the data to the right format could be optimized, as the implementation in this thesis used several steps to get the raw Jackal data to PredNet input.

The results show that only the first derivative could be used, regardless of the finite difference method used. Higher order derivatives are so inaccurate that they could not serve a purpose. Since predictions from PredNet did not increase the accuracy of derivatives and that obtaining and using the predictions is much more effort than simply using the backwards method which achieves similar results, the predictions are not useful for the purpose of obtaining derivatives.

7.3 Finite difference technique

The second sub question concerned the finite difference technique itself and whether it could be optimized using more samples to increase the error order. Comparing the B_1 and C_1 methods with B_F and C_F respectively, showed that the full methods have a higher MSE than the single methods (table 6.2), which is not in line with the theory. By evaluating the transition period in the signal it can be seen that the absolute errors between the full and single methods are very similar. However, relatively speaking, the full methods have less error in the transition period and thus more error between the transition periods. This could perhaps also be explained by the larger reach of the full methods. By using more samples, the sudden change of the transition is present for a longer period of time. This does show in the data, but fig. 6.3) also shows that there are large errors in between the transition periods that could be the cause for the higher MSE.

For the central methods the contribution of the transition periods to the error of the derivatives is somewhat smaller. For these methods, there was also a much larger error in between transition periods. The derivatives are clearly overestimated, especially for the **C_F** methods. This is most likely due to the PredNet predictions. As can be seen in fig. 6.2 the predictions overestimate the signal which will also overestimate the derivatives. Table 6.3 shows that the contributions due to the predictions errors are larger for the **C_F** method than for the **C_1** method, although neither of the contributions are shown to contribute to the overall error significantly.

7.4 Training data

We have seen that the transitions in the data played a large role in several outcomes of this thesis. The commands for the Jackal robot were given so frequently that most of the training data consists of the rise from step responses. The data could have been created differently as to obtain less acute changes and more realistic fluctuations. This could have been done by making sure that the robot reached close to the set point or waiting until after the settling time. Furthermore, having parts of training data with

varying length could improve robustness. Also, training could have been done between commands as to avoid the state transitions. This would likely have improved performance, but might also make the model less robust in situations where sudden state transitions would occur. Another possibility would be to extend the model with long-term dependencies or a type of switch for different scenarios so the model can deal with long-term effects or with different types of behaviour.

7.5 Related and future work

A focus of this work was forecasting values from a time-series using a neural network. The performance was not better than the traditional statistical methods. The performance could be improved by using components that are designed for forecasting of time-series, such as attention mechanisms or by using different techniques such as data augmentation and transfer learning [32]. A different direction is to combine neural networks with statistical models, which shows promise [33]. Furthermore, neural networks still require a lot of manual work in order to implement them. This is another reason why neural networks are not yet widely used for time-series forecasting. It would be worthwhile to set up a more solid infrastructure around the neural networks that would help in data-transformation, estimating the network architecture and the model hyper parameters, see for example a review from Torres et al. [34]. Finally, many neural networks are not designed with online learning in mind. Online learning can be a powerful tool in various implementations and as such, a framework to adapt neural networks to an online version would be beneficial. Alternatively, neural network development could be focused towards online training which would help discover the difficulties of online learning and how to overcome them.

This work also focused on calculating derivatives due to the implementation for Active Inference. This remains a difficult endeavour and the question arises whether higher-order derivatives are truly necessary for a strong implementation of Active Inference [35]. There are many successful implementations of Active Inference which only make use of the first derivative [4], [36]. Finding out exactly how much higher order derivatives would contribute to Active Inference models could help determining whether it is worth the effort of obtaining them. This could be done by comparing performance of an Active Inference model with various levels of access to true derivatives.

Still, derivatives are useful in many applications and while this thesis lay the focus on using the finite difference method, this is not the only method. A different avenue for calculating derivatives is concerned with using filters. One such filter is the recently developed RLPAD [37]: a recurrent low-pass algebraic differentiator. Another interesting filter would be the Savitzy-Golay filter [38]. This filter needs future values, so this might need to be combined with an accurate prediction method.

Chapter 8

CONCLUSION

In this thesis we have explored the possibilities of the finite difference technique and the PredNet model and their combination. It was found that what the PredNet model is capable of learning is heavily dependent on the data it trains on and care should be taken to fit the model to the needs. Furthermore, the potential between online and offline training of the PredNet model is similar, but online learning takes much longer to train in real-time because the framework for neural networks is not optimized for it. When attempting to predict both model behaviour and state transitions with PredNet, the predictions are not accurate enough to warrant a change to the central finite difference method. Finally, using more data samples with the finite difference method to calculate a derivative is not guaranteed to provide a more accurate derivative as that result is also dependent on the quality of the data.

Bibliography

[1] Karl Friston, James Kilner, and Lee Harrison. "A free energy principle for the brain". In: *Journal of physiology-Paris* 100.1-3 (2006), pp. 70–87.

[2] "Predictive coding in the visual cortex: A functional interpretation of some extra-classical receptive-field effects". In: *Nature Neuroscience* 2.1 (1999), pp. 79–87. ISSN: 10976256. DOI: 10.1038/4580.

[3] Karl Friston. "Does predictive coding have a future?" In: *Nature neuroscience* 21.8 (2018), pp. 1019–1021.

[4] Corrado Pezzato, Riccardo Ferrari, and Carlos Hernández Corbato. "A novel adaptive controller for robot manipulators based on active inference". In: *IEEE Robotics and Automation Letters* 5.2 (2020), pp. 2973–2980.

[5] Yasser Elfahham. "Estimation and prediction of construction cost index using neural networks, time series, and regression". In: *Alexandria Engineering Journal* 58.2 (2019), pp. 499–506.

[6] Hansika Hewamalage, Christoph Bergmeir, and Kasun Bandara. "Recurrent neural networks for time series forecasting: Current status and future directions". In: *International Journal of Forecasting* 37.1 (2021), pp. 388–427.

[7] S. Y. W. Jousma. "Hierarchies in Predictive Coding". Apr. 2021.

[8] Anthony Mandow et al. "Experimental kinematics for wheeled skid-steer mobile robots". In: *2007 IEEE/RSJ international conference on intelligent robots and systems*. IEEE. 2007, pp. 1222–1227.

[9] Georgia Anousaki and Kostas J Kyriakopoulos. "A dead-reckoning scheme for skid-steered vehicles in outdoor environments". In: *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*. Vol. 1. IEEE. 2004, pp. 580–585.

[10] S Ali A Moosavian and Arash Kalantari. "Experimental slip estimation for exact kinematics modeling and control of a tracked mobile robot". In: *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2008, pp. 95–100.

[11] Tianmiao Wang et al. "Analysis and Experimental Kinematics of a Skid-Steering Wheeled Robot Based on a Laser Scanner Sensor". In: *Sensors* 15 (May 2015), pp. 9681–9702. DOI: 10.3390/s150509681.

[12] Neal Seegmiller. "Dynamic Model Formulation and Calibration for Wheeled Mobile Robots". PhD thesis. Pittsburgh, PA: Carnegie Mellon University, Oct. 2014.

[13] C. Pezzato, R. Ferrari, and C. H. Corbato. "A Novel Adaptive Controller for Robot Manipulators Based on Active Inference". In: *IEEE Robotics and Automation Letters* 5.2 (2020), pp. 2973–2980. DOI: 10.1109/LRA.2020.2974451.

[14] Mohamed Baioumy et al. "Active Inference for Integrated State-Estimation, Control, and Learning". In: *IEEE International Conference on Robotics and Automation (ICRA)* (May 2020), pp. 4665–4671.

[15] I Hijne. "Generalised Motions in Active Inference by Finite Differences". In: *TU Delft repository* (Aug. 2020).

[16] Cameron R. Taylor. *Finite Difference Coefficients Calculator*. https://web.media.mit.edu/~crtaylor/calculator.html. 2016.

[17] Michael W. Spratling. "Distinguishing theory from implementation in predictive coding accounts of brain function". In: *Behavioral and Brain Sciences* 36.3 (2013), pp. 231–232. ISSN: 14691825. DOI: 10.1017/S0140525X12002178.

[18] William Lotter, Gabriel Kreiman, and David Cox. "Deep predictive coding networks for video prediction and unsupervised learning". In: *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings* (2016), pp. 1–18. arXiv: 1605.08104.

[19] Matin Hosseini and Anthony Maida. "Hierarchical Predictive Coding Models in a Deep-Learning Framework". In: *arXiv preprint arXiv:2005.03230* (2020). arXiv: 2005.03230. URL: http://arxiv.org/abs/2005.03230.

[20] Roshan Prakash Rane et al. "PredNet and Predictive Coding: A Critical Review". In: *Proceedings of the 2020 International Conference on Multimedia Retrieval*. New York, NY, USA: ACM, June 2020, pp. 233–241. ISBN: 9781450370875. DOI: 10.1145/3372278.3390694. arXiv: 1906.11902. URL: https://dl.acm.org/doi/10.1145/3372278.3390694.

[21] Xia Huang, Hossein Mousavi, and Gemma Roig. "Predictive Coding Networks Meet Action Recognition". In: *arXiv preprint arXiv:1910.10056* (2019). arXiv: 1910.10056. URL: http://arxiv.org/abs/1910.10056.

[22] Marcio Fonseca. "Unsupervised predictive coding models may explain visual brain representation". In: *arXiv preprint arXiv:1907.00441* (2019).

[23] Ardiansyah Fauzi and Norimi Mizutani. "Potential of deep predictive coding networks for spatiotemporal tsunami wavefield prediction". In: *Geoscience Letters* 7.1 (2020), pp. 1–13.

[24] Eiji Watanabe et al. "Illusory motion reproduced by deep neural networks trained for prediction". In: *Frontiers in Psychology* 9.MAR (2018), p. 345. ISSN: 16641078. DOI: 10.3389/fpsyg.2018.00345.

[25] William Lotter, Gabriel Kreiman, and David Cox. "A neural network trained to predict future video frames mimics critical properties of biological neuronal responses and perception". In: *Arxiv* (2018), pp. 1–18. arXiv: 1805.10734. URL: http://arxiv.org/abs/1805.10734.

[26] Nelly Elsayed, Anthony S. Maida, and Magdy Bayoumi. "Reduced-gate convolutional long short-term memory using predictive coding for spatiotemporal prediction". In: *Computational Intelligence* (2020). ISSN: 14678640. DOI: 10.1111/coin.12277.

[27] Junpei Zhong et al. "AFA-PredNet: The Action Modulation Within Predictive Coding". In: *Proceedings of the International Joint Conference on Neural Networks*. Vol. 2018-July. IEEE. 2018, pp. 1–8. ISBN: 9781509060146. DOI: 10.1109/IJCNN.2018.8489751. arXiv: 1804.03826.

[28] Zdenek Straka, Tomas Svoboda, and Matej Hoffmann. "PreCNet: Next Frame Video Prediction Based on Predictive Coding". In: *arXiv preprint arXiv:2004.14878* (2020). arXiv: 2004.14878. URL: http://arxiv.org/abs/2004.14878.

[29] Oskar Triebe, Nikolay Laptev, and Ram Rajagopal. "Ar-net: A simple auto-regressive neural network for time-series". In: *arXiv preprint arXiv:1911.12436* (2019).

[30] Yong Zhou, Lingyu Wang, and Junhao Qian. "Application of Combined Models Based on Empirical Mode Decomposition, Deep Learning, and Autoregressive Integrated Moving Average Model for Short-Term Heating Load Predictions". In: *Sustainability* 14.12 (2022), p. 7349.

[31] Fitash Ul Haq et al. "Comparing offline and online testing of deep neural networks: An autonomous car case study". In: *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)*. IEEE. 2020, pp. 85–95.

[32] Hassan Ismail Fawaz et al. "Deep learning for time series classification: a review". In: *Data mining and knowledge discovery* 33.4 (2019), pp. 917–963.

[33] Bryan Lim and Stefan Zohren. "Time-series forecasting with deep learning: a survey". In: *Philosophical Transactions of the Royal Society A* 379.2194 (2021), p. 20200209.

[34] José F Torres et al. "Deep learning for time series forecasting: a survey". In: *Big Data* 9.1 (2021), pp. 3–21.

[35] Pablo Lanillos et al. “Active inference in robotics and artificial agents: Survey and challenges”. In: *arXiv preprint arXiv:2112.01871* (2021).

[36] Manuel Baltieri and Christopher L Buckley. “An active inference implementation of phototaxis”. In: *arXiv preprint arXiv:1707.01806* (2017).

[37] E.P. Veldhuis. “A new SIMO filter for the estimation of higher order derivatives”. Dec. 2021.

[38] Abraham Savitzky and Marcel JE Golay. “Smoothing and differentiation of data by simplified least squares procedures.” In: *Analytical chemistry* 36.8 (1964), pp. 1627–1639.