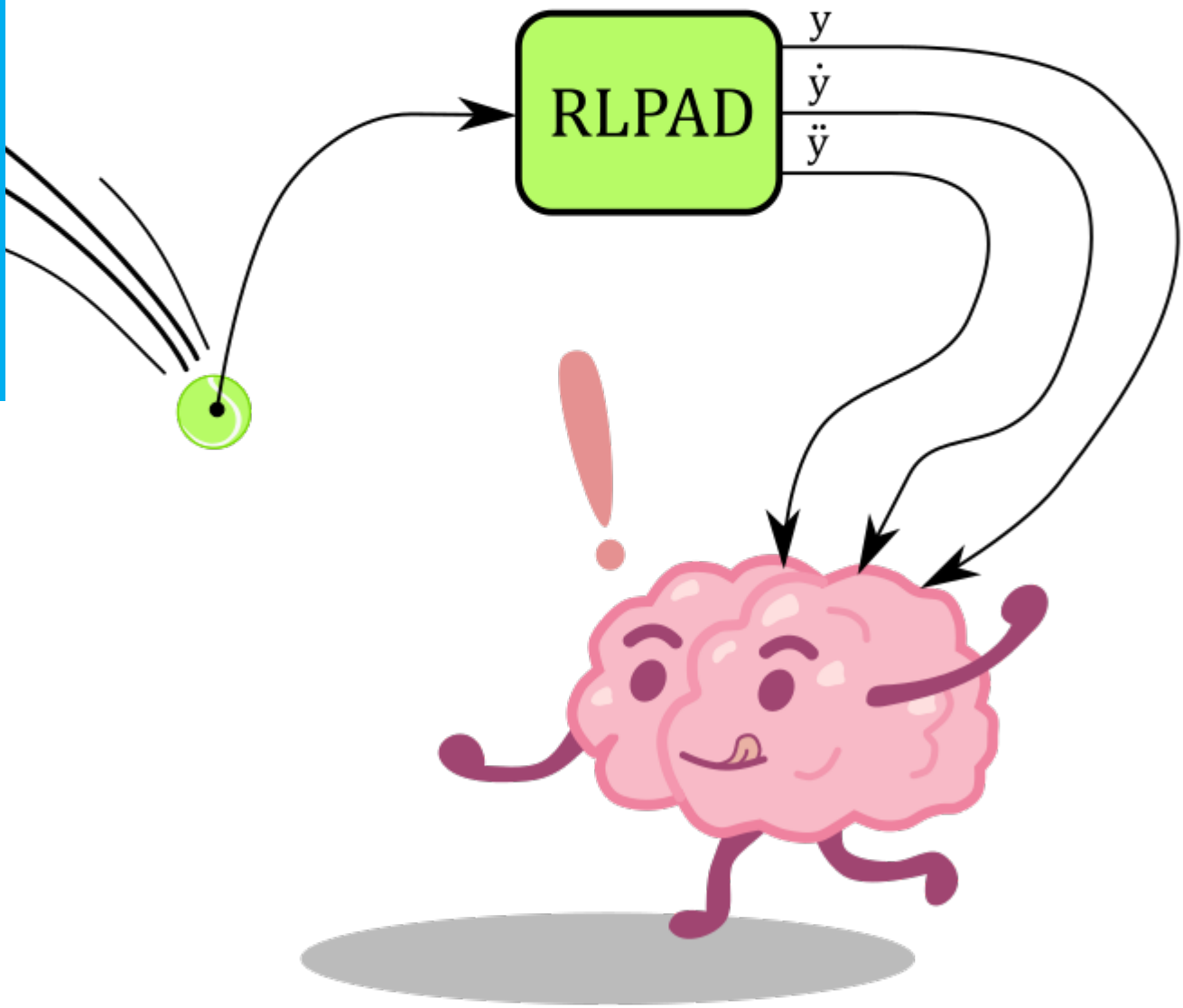
A new SIMO filter for the estimation of higher order derivatives

The recurrent low pass algebraic differentiator

E.P. Veldhuis

Master of Science Thesis



TU Delft Delft University of Technology

Department of Cognitive Robotics - ME-BMD

A new SIMO filter for the estimation of higher order derivatives

The recurrent low pass algebraic differentiator

Master of Science Thesis

For the degree of Master of Science in Mechanical Engineering with the specialization BioRobotics at Delft University of Technology

E.P. Veldhuis

December 2, 2021

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

Abstract

This research proposes a new differentiator for estimating higher order derivatives of an input signal. The main reason why higher order derivatives are necessary is that Active Inference makes use of generalized coordinates. This means that it keeps internally track of higher order temporal derivatives of states, inputs and measurements. The difficult part of generalized coordinates are the generalized measurements, because these should be measured from a physical system. However, it is not always possible to measure all states of a system and it is definitely not possible to measure all higher order derivatives. A solution to this is to estimate the derivatives of states by using real time differentiators.

A short literature study about differentiators is conducted and found that the state of the art differentiator is the algebraic estimation approach differentiator (AEAD). However, this and other differentiators have the problem that when they are converted to discrete time, they cannot track polynomial signals anymore. For that reason, this thesis proposes a new differentiator: the recurrent low pass algebraic differentiator (RLPAD).

The proposed differentiator is compared with the AEAD in two experiments. The first experiment evaluates the performance based on three analytical inputs with known higher order derivatives. The three inputs are: a polynomial, a sine and a combination of the two. Additionally these three inputs are corrupted by Gaussian white noise. This experiment concludes that the proposed method outperforms the AEAD significantly when a polynomial or polynomial combined with sine input is used. They perform similar for sine inputs, although RLPAD is considered slightly better.

The second experiment evaluates how the two differentiators perform when real sensor data is used. In order to conduct the experiment, the raw data is smoothed by a Savitzky-Golay filter to create a ground truth about the derivatives. This experiment concludes that the differentiators have an optimal set of parameters, where either the one or the other performs better. The proposed method performs a lot better when few derivatives are estimated. On the other hand AEAD performs better when there are more derivatives estimated. However, this is not true when the noise gets stronger. AEAD is more sensitive to noise than the proposed method.

Based on the experiments, the proposed method RLPAD is the better differentiator for creating generalized measurements in Active Inference for three reasons. The first is that it can track polynomial signals. The second is that it has stronger noise attenuation. And the third reason is that it is computationally faster.

Table of Contents

List of Figures

List of Tables

Preface

Interestingly the idea to write the thesis about this topic was not planned. The idea came as I was working with three other students (Mitchel, Bob and Sjoerd) on a real robotic implementation of Active Inference in ROS for a skid steer robot, which was instigated by my supervisor Martijn Wisse. This implementation and robot could then be used for numerous thesis topics. However, during implementation we stumbled upon the problem of generalized measurements. Due to time constraints we settled for a backward difference scheme that was also used by an other student (Iris) to create generalized measurements.

We obviously noticed how bad this differentiation scheme works when there is noise present on the signal, because the higher order derivative estimations were extremely inaccurate. Therefore I thought, differentiation can be done in a better way so that the generalized measurements do have more accurate values and can be made sense of. The differentiator in this thesis came to me as a sudden revelation as I was experimenting with second order dynamical systems.

Acknowledgements

Firstly, I would like to thank my supervisor prof.dr.ir. M. Wisse for his (online) assistance during the writing of this thesis and the occasional feedback.

Secondly, I would like to thank Ajith and Dennis for providing the sensor data of the drone they used for their experiments. This saved me a lot of time and it gave me additional insights in the performance of the differentiators.

Thirdly, I would like to thank dr. P. Mohajerin Esfahani for making time to asses the initial differentiator I came up with. This helped me which factors to consider when designing and reporting a filter.

Fourthly, I would like to thank my girlfriend Cynthia for supporting me during the writing of this thesis and for a final proofread. The Covid-19 pandemic caused loss of motivation, but she pulled me through.

Delft, University of Technology E.P. Veldhuis
December 2, 2021

"In theory, theory and practice are the same. In practice, they are not."
— *Albert Einstein*

Glossary

List of Acronyms

BFD	backwards finite difference scheme
CD	classical differentiator
GRED	global robust exact differentiator
HCND	hybrid continuous nonlinear differentiator
REUCOAD	robust exact convergent arbitrary order differentiator
TSEBD	Taylor series expansion based differentiator
OADE	overlapping algebraic derivatives estimator
AEAD	algebraic estimation approach differentiator
RLPAD	recurrent low pass algebraic differentiator
RMSE	root mean square error
TSB	Taylor series based approach
SMT	sliding mode technique

List of Symbols

κ_i	Represents $\omega_i h$
ω_0	Main cutoff frequency of RLPAD
ω_c	Cutoff frequency of AEAD
ω_i	Cutoff frequencies of RLPAD
$\hat{y}^{(i)}$	Output derivative estimate
e_i	Short notation for $e^{\omega_i h}$
f_{red}	Reduction factor for the $i+1$ cutoff frequencies of RLPAD
h	Time step
n	Filter size or amount of derivatives estimated including the zeroth derivative
r	Input or reference function
u	Input or input vector
$u^{(0)}$	Denotes the true system input for RLPAD
$u^{(i)}$	Special input vector for RLPAD
y	Output vector or derivative estimate
$^{(i)}$	Denotes the i-th derivative or i-th estimated derivative

Chapter 1

Introduction

Humans are able to do mundane tasks very easily, such as picking up a book or a cup without spilling water, writing, drawing, catching a ball etc. For robots such simple tasks are very complex, even in controlled environments. However, in recent years robots have become more intelligent by incorporating techniques such as machine learning, predictive coding and neural networks which are inspired by biology [50] and how the brain functions [39] [30]. These techniques allow robots to adapt to a wide range of configurations, making them seem more intelligent. But oftentimes a robot has no idea what to do with configurations and scenarios it has never seen before. The human on the other hand, can learn and adapt to many different kind of scenarios, because of how the brain functions. Creating brain-like AI, also known as strong AI or artificial general intelligence (AGI), is a long way from being realized. It is even argued that it is impossible to even create general AGI [15], like humans have. Despite that, the brain and brain-like behavior is studied intensively.

Recent research in neurology gave rise to the theory of Active Inference, developed by K. Friston [18] [20]. This theory is the most unifying theory about the human brain to date and unifies action, perception and learning by the use of prediction error minimization. Active Inference arises from the Free Energy Principle (see Chapter 2), which gives it a clean mathematical formulation.

Active Inference have caught its attention in robotics. The first reason is that the theory unifies action, perception and learning into one formulation, which can be exploited directly. This avoids the need for combining different methods that can do one task each and is therefore very interesting in robotics. There are not many practical implementations of Active Inference in robotics, but recent research [40] [5] have shown that a robotic manipulator can be controlled by Active Inference by using attractor dynamics. The second reason why Active Inference is so interesting for robotics, is that Active Inference is able to work with structured noise, also called colored noise. A real life example of colored noise, is that wind blowing from the south cannot change in a split second to blow from the north. Since Active Inference is able to cope with colored noise it can theoretically out perform a LQG controller. It has been proven that LQG and Active Inference are the same when modeled as a linear time invariant system without generalized coordinates of motion [9].

However, in order to work with colored noise, Active Inference requires generalized coordinates. This means that it models the trajectory of all time dependent states, inputs and measurements. In a nutshell, it keeps track of higher order derivatives of those states. This is a key concept in Active Inference. A difficult problem in robotic implementations of Active Inference, are the generalized coordinates of the measurements. How does one measure higher order temporal derivatives of physical system? Not all states are measurable in a system. For example there are no sensors that can measure the third order derivative of position or temperature. Besides that, a system can have a limited amount of sensors, and somehow the temporal derivatives of those sensor measurements have to be extracted to properly work with Active Inference. A naive way of extracting higher order derivatives

is to use a backward difference scheme. This generally does not work for noisy signals, therefore this thesis proposes a new causal method that can extract higher order derivatives from a single input.

1-1 Research Motivation

An important part of Active Inference is the use of generalized coordinates of motion or generalized motions. The generalized motions is a state vector or a measurement vector that also has temporal derivatives of itself it up to p-th order, also called the embedding order. This is an important concept in Active Inference. This means that a generalized state vector can contain position, velocity, acceleration and jerk for example.

Current implementation of Active Inferences in robots do not use generalized motions or have only used the first embedding order as [40] and [5] have done. They did this because the sensors of the robot arm can only measure position and velocity.

Having access to more temporal derivatives of the measurements allows one to use more embedding orders and in theory get better performance. So the question is: How can you get higher order temporal derivatives of a single signal in real time?

1-1-1 Problem formulation

In high performance control it is required to have good noise attenuation on the input signals as well as estimations of the first order derivative of the states for better performance [51]. This is often necessary in robotics, aeronautics, signal processing and space engineering to name a few.

Active Inference goes a step further, it requires higher order temporal derivatives up to an order of six for the best performance. Temporal derivatives are required for the generalized coordinates vector of Active Inference (see chapter 2). An unsolved problem in Active Inference is how to generate generalized measurements? This means, that Active Inference requires higher order derivatives of measurement signals. However, it is not a trivial task to find higher order derivative of a noisy input signal in real time.

The problem that needs to be solved is: Given a measured signal y with white Gaussian noise, how to find approximations for $\frac{d}{dt}y$, $\frac{d^2}{dt^2}y \dots \frac{d^n}{dt^n}y$ in real time? Real time tracking differentiators are a solution to this problem. They are in essence estimators which do not have modeled dynamics of a system, unlike Kalman filters. They also do not require statistical knowledge of the noise to work properly, making them useful in environments where the statistical properties of the noise can change over time.

It turns out that the most state of the art differentiator is not sufficient enough to work with Active Inference, because it cannot estimate derivatives of a polynomial input in discrete time. For that reason it is necessary to device a new method that can extract at least two or more derivatives. I created a new type of differentiator that can perform as good or better than the current state of the art differentiator.

Problem to be solved

The best differentiator has two problems if used for creating generalized measurements in Active Inference. The first and main problem is that the best performing differentiator can't estimate higher order derivatives of polynomials properly, when it is converted to a discrete time formulation. The second problem is that the noise variance of the derivative terms is unknown. Quantifying noise variance is necessary for the precision matrices in Active Inference.

To solve these issues, a new differentiator has to be devised that can track polynomials and periodic functions as good or better than the current state of the art differentiator. To device a new method three research objectives and three research questions are formulated.

Categorizing existing methods based on properties they have in common and their performance

This will show an informative table filled with existing differentiators and how they score compared to the others. This also gives a guideline for the new method in what areas it has to excel and where the new method will fit.

Create a custom differentiator

As the state of the art differentiator is not sufficient enough, there is necessity to device a new method that can extract at least two or more derivatives. A mathematical analysis is done for a new type of differentiator that can also attenuate noise and extract any order of derivative.

Quantify the variance propagation of the higher order derivatives from the input.

Active Inference uses precision matrices for weighting the sensory input vector. A precision matrix is the inverse of a variance matrix. If it is possible to quantify the variance for each higher order derivative estimate from the variance of the input, then the precision matrix will be known. This would be a very valuable outcome for Active Inference in robotics, if it can be determined directly from the noise statistics of the input and the parameters of the differentiator.

How does the proposed method perform on known signals compared to the state of the art differentiator?

The proposed method is tested and compared with the state of the art differentiator (AEAD) for predetermined functions with their true derivatives known. How do the AEAD and the proposed method perform on several types of input functions such as a polynomial, a periodic and a combination of the two? And how do they perform on these types of functions when there is additive white Gaussian noise on the signal.

How does the proposed method perform on real sensor data compared to the state of the art differentiator?

Both methods are tested and analyzed again, but now the input will be real sensor data that is generated by a quad copter drone and recorded by the OptiTrack system. The ground truth of the derivatives will be determined with offline signal analysis.

Which method is best suited for creating generalized measurements in Active Inference?

The final goal of this thesis is to have a differentiator that can construct generalized measurements for Active Inference. This question can be answered properly based on the results of the experiments of the state of the art differentiator and the proposed method.

1-1-2 Structure of the thesis

The thesis is structured as follows. In Chapter 2 the Free Energy Principle (FEP) is explained briefly and the reason why Generalized Motions are so important in Active Inference. This will also show why differentiators are a good option to generate the generalized measurements in real systems. Chapter 3 shows existing methods that are capable of creating temporal derivatives from a single input signal. They are categorized based on their performance and properties. Chapter 4 shows the proposed method to solve the main problem of this thesis. This method is analyzed thoroughly and is eventually categorized based on the categories of chapter 3 in chapter 7. This gives a good overview

where it fits in with respect to existing methods. In chapter 5, numerical experiments are conducted to see how the proposed method performs and how the state of the art differentiator perform under the same conditions. They are compared to an analytical solution with known input and derivatives. In addition to that, chapter 6 shows how the proposed method and the AEAD perform on sensor data of a real system. Chapter 7 concludes the thesis and will give a verdict of the performance of the proposed method.

Chapter 2

Free Energy principle and Active Inference

This chapter summarizes Active Inference in such a way that is understandable from a high level. The Free Energy Principle will be described with as few neurological jargon and equations as possible. The most important aspect of this chapter are generalized motions which will be relevant throughout this thesis. This is especially the case for the generalized measurements, which is a subset of the generalized coordinates. The practical difficulties involved with generalized measurements are explained at the end of this chapter

2-1 The Free Energy Principle (FEP)

The easiest way to describe the FEP is as an organizing principle for any living system that shows the characteristics of life. What kind of behavior must one show to be alive? A system must counter dispersive effects caused by random fluctuation in order to exist over a period of time [18]. A system is more likely to be found in some states than others, due to the countering of dispersive effects. This represents the ensemble density $p(\tilde{\mathbf{x}}|m)$, which is the probability of finding an agent in a particular state when observed on multiple occasions. In $p(\tilde{\mathbf{x}}|m)$ is m the agent that is affected by the generalized states $\tilde{\mathbf{x}}(t) = [\mathbf{x}, \dot{\mathbf{x}}, \ddot{\mathbf{x}}, ...]$. Generalized states are temporal derivatives of the state up to infinite order (see section 2-3). These can contain any physical quantity like gravity, temperature, position, etc that are part of or can influence the agent [17]. Having an infinite order of derivatives is not practical in implementations, and according to Friston, the generalized states stop containing useful information after an embedding order of six [24]. An embedding order represents the amount of derivatives. This means that the first temporal derivative is the first embedding order, the second temporal derivative the second embedding order etc.

There are two assumptions in the FEP. The first is that the system is ergodic. This means that a system must counter dispersive effects to exist for a period of time. In other words a system minimizes the entropy to survive [20]. The entropy $H(X)$ can be written as an integration over the life time trajectory $\tilde{\mathbf{x}}(t)$ of $p(\tilde{\mathbf{x}}|m)$ as

$$H(X) = \lim_{T\to\infty} \frac{1}{T} \int_0^T -\ln p(\tilde{\mathbf{x}}|m) dt \tag{2-1}$$

The terms $-\ln p(\tilde{\mathbf{x}}|m)$ is called the surprisal or self information. This means that, in order to be alive, the agent has to minimize the surprisal at all times. The term surprisal to live longer can also be extrapolated to our daily lives, for example if a person would stay at home for a day (not much happening). Then that person has a lower surprisal than what he would have if he drives a car on the highway which has a lot of activity going on (i.e. more risks of dying). So in order to live longer an agent must avoid getting in risky situations, or in other words minimize surprisal.

The second assumption in the FEP is that it is modeled as a Markov blanket. A Markov blanket can be described as follows. Any system that exists acts like it has a model of the world and it is trying to gather evidence for its own model of the world. A good example is the human brain, which is an inference machine. It uses sensory information to infer its world and makes use of actions that act on the outside world to verify the world model within the brain.

The Markov blanket separates states into things that are internal to the boundary (internal states) and states that are external to the boundary (hidden states). This means that the sensory states can only affect the internal states. The active states on the other side can affect the hidden states, but are not influenced by it. In fact the active states are dependent on the internal states.

Sensory observations come with noise. This noise causes the entropy to increase. This has the result that the entropy of the sensory observations is always equal or greater than the sensory observations without noise. Therefore one can say that the minimization of noisy entropy $H(Y)$ is upper bound on $H(X)$.

2-1-1 Free Energy

A problem with surprisal is that agents cannot quantify it. This is because the agent has to integrate over the unknown causes $\Psi \supset \{\tilde{\mathbf{x}}, \theta\}$ (θ are the hidden model parameters) of the sensory input:

$$-\ln p(\tilde{y}) = -\ln \int p(\tilde{y}, \Psi) d\Psi$$

A different way to minimize surprise comes from theoretical physics [14] and machine learning [29] [36]. The surprisal can be minimized by minimizing the free energy bound. This works because the free energy bound is always greater than the surprise.

This bound is generated by introducing a recognition density $q(\Psi|\zeta)$, which is parameterized by its sufficient statistics $\zeta \supset \tilde{\zeta}(t)$. This means that for a Gaussian density ζ contains the mean and covariance. When the recognition density is optimized to minimize free energy, it approaches the density on the causes of sensory data ($q(\Psi|\zeta) \approx p(\Psi|\tilde{y})$). The difference between these densities can be measured with the Kullback-Leibler divergence, which always returns a positive scalar value or zero when both densities are equal.

The free energy bound is created by the (always positive valued) divergence between the recognition density and the conditional density summed with the surprise. There are three expression for free energy by using Bayes rule [23], but the most understandable expression for free energy is:

$$F(\zeta, \tilde{y}) = D_{KL}(q(\Psi|\zeta)||p(\Psi|\tilde{y}) - \ln p(\tilde{y}) \tag{2-2}$$

In this expression is $D_{KL}(\cdot||\cdot)$ the Kullback-Leibler divergence between two densities. This formulation shows that by minimizing free energy with respect to ζ the divergence between the recognition and conditional densities are reduced so that the recognition density approximates the conditional density $q(\Psi|\zeta) \approx p(\Psi|\tilde{y})$. This corresponds to Bayesian inference on the causes of sensory signals [22]. Note that the free energy is upper bound on surprise because the divergence is always positive or zero. Finally the expression shows that the free energy is a scalar.

The formulation also shows that it is possible to minimize free energy for sensory inputs and action simultaneously. By minimizing the free energy with respect to $\tilde{y}$, action suppresses the free energy. However, $\tilde{y}$ can't be controlled directly. This is where Active Inference comes into play as we shall see in section 2-2.

The FEP is summarized as follows. In order to be alive an agent has to minimize the free energy of its internal states and its action. Free energy optimizes a probabilistic model based on sensory observations. At its core, the FEP is a prediction error minimizer by suppressing free energy.

2-2 Active Inference

Active Inference combines perception, learning and action in one formulation based of the FEP. Active inference is a form of self references learning [42]. With several assumptions in the FEP [18], Active Inference can be formulated as a gradient descent on the weighted sum of squared prediction errors. This means that a system can always minimize the total free energy by using measurement and the generative model.

The system

$$\begin{aligned} \tilde{\mathbf{y}} &= \mathbf{g}(\tilde{\mathbf{x}}, \tilde{\mathbf{v}}, \theta) + \tilde{\mathbf{z}} \\ \dot{\tilde{\mathbf{x}}} &= \mathbf{f}(\tilde{\mathbf{x}}, \tilde{\mathbf{v}}, \theta) + \tilde{\mathbf{w}} \end{aligned} \tag{2-3}$$

is the true generative model (real world), which describes how sensory data are generated for the agent. This information is not directly available to the agent so it makes the assumption that the data are generated by:

$$\begin{aligned} \tilde{y} &= g(\tilde{x}, \tilde{v}, \theta) + \tilde{z} \\ \dot{\tilde{x}} &= f(\tilde{x}, \tilde{v}, a, \theta) + \tilde{w} \\ \tilde{v} &= \tilde{\eta} + \tilde{n} \end{aligned} \tag{2-4}$$

This model is called the generative model, which is a model of the environment where the agent is immersed in. The hidden states are $\tilde{x}$ and $\tilde{v}$, where $\tilde{v}$ models the perturbations outside the agent its environment [23] and is considered autonomous. The hidden parameters are θ, which help map the generative model. They are, for example, parameters of the equations of motion.

The sensory states $\tilde{\mathbf{y}}$ and $\tilde{y}$ look similar, but they ar not. That is because the generative model is a probabilistic model of the true (stochastic) model. Another difference is that the action is embedded within equation (2-3). The generative model of equation (2-4) has the states partitioned in a non autonomous part $\tilde{x}(t)$ and an autonomous part $\tilde{v}(t)$. This causes a difference between the models. In fact this difference is what action (by the agent) tries to cancel by minimizing the free energy.

By minimizing the free energy the agent has to infer the states of the world and learn the unknown parameters responsible of its motion by optimizing the sufficient statistic of its recognition density [23]. This is perceptual inference and learning. This scheme is solved by assuming a mean-field approximation [24]. Under the Laplace approximations, which considers Gaussian distributions, it is sufficient to only optimize the expectations of the parameters. This is the case, because the covariances can be written as functions of the expectations. Therefore we can assume that the sufficient statistics are the expectations of the states ($\tilde{x}$ and $\tilde{v}$), parameters (θ) and precisions (γ), $\mu = (\tilde{\mu}_x, \tilde{\mu}_v, \mu_\theta, \mu_\gamma)$ [23], these are called the beliefs. Based on these approximations, gradient descent equations are formulated from the Free energy that describes perceptual inference, learning and action in equation (2-5) [17] [24].

The gradient descent equations are seen as biology plausible and are:

Perceptual inference:

$$\begin{aligned}\dot{\tilde{\mu}}_x &= D\tilde{\mu}_x - \frac{\partial F}{\partial \tilde{\mu}_x} \approx D\tilde{\mu}_x - \frac{\partial \tilde{\varepsilon}^\top}{\partial \tilde{\mu}_x}\xi \\ \dot{\tilde{\mu}}_v &= D\tilde{\mu}_v - \frac{\partial F}{\partial \tilde{\mu}_v} \approx D\tilde{\mu}_v - \frac{\partial \tilde{\varepsilon}^\top}{\partial \tilde{\mu}_v}\xi\end{aligned}$$

Learning:

$$\ddot{\mu}_\theta = -\frac{\partial F}{\partial \tilde{\mu}_\theta} \approx -\frac{\partial \tilde{\varepsilon}^\top}{\partial \tilde{\mu}_\theta}\xi \tag{2-5}$$

Action:

$$\dot{a} = -\frac{\partial F}{\partial a} \approx -\frac{\partial \tilde{\varepsilon}^\top}{\partial a}\xi_y$$

Free energy:

$$\begin{aligned}F &\approx \frac{1}{2}\tilde{\varepsilon}^\top \Pi \tilde{\varepsilon} - \frac{1}{2}\ln|\Pi| \\ \xi &= \Pi\tilde{\varepsilon}\end{aligned}$$

Prediction errors:

$$\tilde{\varepsilon} = \begin{bmatrix} \tilde{\varepsilon}_y = \tilde{y} - g(\mu) \\ \tilde{\varepsilon}_x = D\tilde{\mu}_x - f(\mu) \\ \tilde{\varepsilon}_v = \tilde{\mu}_v - \tilde{\eta} \end{bmatrix} \tag{2-6}$$

The Laplace approximations and neglection of mean field approximation terms [23] make the formulation of free energy F a weighted summed square error. The makes it easier to calculate the partial free energy gradients (∂F) with respect to x, v, θ and a. The prediction errors are denoted by $\tilde{\varepsilon}$, which are the difference between the sensory samples $\tilde{y}$ with the sensory mapping of the hidden states $g(\mu)$ and the error of the expected motion with the expected hidden state. D is a motion shift operator which maps the derivate of $\tilde{\mu}$ as $\tilde{\mu'} = D\tilde{\mu}$. This means that D is a square matrix with identity matrices at the appropriate locations.

The matrix $\Pi(\mu(\gamma))$ is the inverse covariance matrix or precision matrix of random effects. This means that a low covariance has a high precision matrix and boosts the prediction errors proportionally [24]. In theory the precision matrix Π is constructed based on correlated noise by using temporal precisions matrices [24]. However, when the noise is uncorrelated, the temporal precision matrix will have very high values on its main diagonal with respect to its off-diagonal values [24]. This has the effect that only the diagonal will be weighted. Therefore when the noise is uncorrelated, it can be assumed that the matrix Π will only have the inverse covariances per state on its diagonal. The noise can be considered independent [8] (most of the times) and this is how the precision matrix for the states Π_x and for the measurements Π_y are constructed in real robotic implementations of Active Inference [40] [5]. Although this assumption is appropriate for physical systems, it is not very plausible for biological processes when the noise is a result of dynamic processes itself [24].

The first two expressions in equation (2-5) describes recognition dynamics in terms of how the expected states change over time. This can be seen as the state prediction step. The minimization of prediction errors is also called predictive coding [44].

The learning part is captured in the third line of equation (2-5), which learns the parameters of θ. Note that this is a second-order differential equation, because these expectations optimize Action [17]. This equation plays an important role for learning in Dynamic Expectation Maximization (DEM) [24], in fact Active Inference is an extension to DEM.

The fourth expression describes action as gradient descent. Action cannot affect free energy directly, but action can affect the sensory prediction errors $\tilde{\varepsilon}_{y|a} = \tilde{y}(a) - g(\mu)$. This links action to perception and this is why Active Inference is unique. This means that Actions tries to fulfill perceptual predictions while at the same time perception tries to minimize the prediction error by adjusting the expectations of the hidden states. This is generally impossible to solve as both a and $\tilde{y}$ are a function of time. Therefore $\partial_a \tilde{y}$ is a functional derivative and cannot be determined if not the full history of $a(t)$ is taken into account. One solution to compute this functional derivative is by using high-dimensional space regressors, but these increase the complexity and give unreliable results [34]. A forward model that maps the input to output directly circumvents the need for this. This is extensively used in the internal model principle of control theory which states "a good controller incorporates a model of the dynamics that generate the signals which the control system is intended to track." [16]. Which means that the controller contains a model of the outside world. Current implementations of Active Inference in a real robot use a forward model with attractor dynamics [40] [5]. Forward models also show up in biology, these would be similar to the classical reflex arcs [19].

When no learning is involved and the external forces are ignored the free energy formulation can be partitioned as:

$$F(\mu, y) = \frac{1}{2}\tilde{\varepsilon}_x^\top \Pi_w \tilde{\varepsilon}_x + \frac{1}{2}\tilde{\varepsilon}_y^\top \Pi_z \tilde{\varepsilon}_y \tag{2-7}$$

This partitioning shows clearly the prediction errors of the hidden states and the sensory prediction errors. The latter term is of most interest in this thesis. This term requires to have generalized sensory prediction errors $\tilde{\varepsilon}_y = \tilde{y} - g(\mu)$. However, how does one acquire generalized measurements $\tilde{y}$ if one cannot measure higher order derivatives of a signal y. In current robotic implementations of Active Inference an embedding order of $p = 1$ is used [40] [5]. This is because the sensors of the robotic system only allow for positions and velocity measurements. Having a method that can estimate higher order derivatives, and thus having a higher embedding order, should help in making Active Inference function better.

Active Inference can be summarized as follows. Active Inference is developed by neuroscientist Karl Friston as a unifying theory for the brain. It unifies perception, learning and action into a single free energy formulation. This unification is something no other theory has done before and is what makes Active Inference unique. By using Laplace and mean-field approximations, the free energy formulation is a weighted square sum of prediction errors. From this free energy formulation, gradient descents expressions are derived for perception, learning and action. A short description of how Active Inference functions is that an agent samples the world to match its predictions and use action to fulfill these. A final note on Active Inference is that it is a closed loop control system and when the embedding order is $p = 0$ and only perceptual inference is considered, it is very comparable to Linear Quadratic Gaussian (LQG) control [24] [41] [21]

2-3 Generalized Motions

In the previous sections, the terms generalized coordinates or generalized motions are mentioned. These are an important concept in Active Inference and are necessary to make the Free Energy Principle work on dynamical systems. This section explains the most important concepts of generalized motions.

The first reason why generalized motions are important is as follows. The generalized motions allow for better estimation of the posterior $p(\tilde{x}|\tilde{y})$ than $p(x|y)$. A good explanation for this is that state x is a dynamic variable and therefore $p(x|y)$ will change over time. By including generalized motions of x it capture this change over time. A Kalman filter for example does not use this information if smooth noise acts on the system [31]. Although attempts have been made to incorporate smooth noise in a Kalman filter by using autoregressive models, but are not optimal [26] [54].

The second reason is that a dynamic equilibrium can be tracked by using generalized motions. Then based on the mean-field and Laplace approximations, the gradient descent equation of perceptual inference is [23] [24]:

$$\dot{\tilde{\mu}} = D\tilde{\mu} - \partial_\mu F(\tilde{\mu}, \tilde{y})$$

where $\tilde{\mu} = \mathrm{E}[\tilde{x}]$ and note that $D\tilde{\mu} = \tilde{\mu}'$. When there is a difference between $\dot{\tilde{\mu}}$ and $\tilde{\mu}'$ the free energy is not minimal. When $\dot{\mu} \approx \mu'$ the free energy is minimal and this also means that the motion of $\tilde{x}$ is constant. This means that the change over time of the distribution $p(\tilde{x}|y)$ is zero ($\dot{p}(\tilde{x}|y) = 0$) and thus a dynamic equilibrium is achieved.

Consider the following autonomous state space system

$$\begin{aligned} \dot{x} &= f(x) + w \\ y &= g(x) + z \end{aligned} \tag{2-8}$$

where w and z are noises. Usually these are considered white Gaussian noise in engineering. White noise has the issue that is cannot be differentiated with respect to time [25]. The derivative of white noise is infinite and contains no useful information anymore. On the other hand, real world noises are generated by dynamic processes [17] and have a smooth transition in it. This means that this type of noise is differentiable with respect to time and allows for extraction of information from higher order derivatives.

In the case of differentiable noises and omitting non linear terms, the temporal derivatives of the system of equation (2-8) are:

$$\begin{array}{ll} \dot{x} = f(x) + w & y = g(x) + z \\ \ddot{x} = \partial_x f(x)\dot{x} + \dot{w} & \dot{y} = \partial_x g(x)\dot{x} + \dot{z} \\ \dddot{x} = \partial_x f(x)\ddot{x} + \ddot{w} & \ddot{y} = \partial_x g(x)\ddot{x} + \ddot{z} \\ \vdots & \vdots \end{array} \tag{2-9}$$

For an agent in uncertain dynamic environments, the state x and the noises are unknown. The accuracy of state estimate or beliefs of x can be increased by having higher order derivatives. Think of this as a Taylor expansion of a differentiable function around a point, it also gets more accurate to the true function when more derivative terms are considered. The amount of higher order derivatives used are denoted by p and is called the embedding order. When the change of beliefs about the states are used instead of time derivatives equation 2-9 becomes

$$\begin{array}{ll} x' = f(x) + w & y = g(x) + z \\ x'' = \partial_x f(x)x' + w' & y' = \partial_x g(x)x' + z' \\ x''' = \partial_x f(x)x'' + w'' & y'' = \partial_x g(x)x'' + z'' \\ \vdots & \vdots \end{array} \tag{2-10}$$

Beliefs can be seen as the expected value of the true state, this is something that Active Inferences makes extensively use of. The prime denotes that x' is calculated from x and is the belief of the true $\dot{x}$. The primes are called "motion", so that x' is the motion of x and follow the same rules as the dot notation for temporal derivatives. The prime is just to distinguish the difference. The system above can be written compactly as the generalized system:

$$D\tilde{x} = \tilde{f}(\tilde{x}) + \tilde{w} \quad \tilde{y} = \tilde{g}(\tilde{x}) + \tilde{z} \tag{2-11}$$

The generalized state is $\tilde{x} = (x, x', x'', ...)^\top$ and the same goes for $\tilde{y}$, $\tilde{w}$ and $\tilde{z}$. The matrix for $\tilde{f}(\tilde{x})$ and $\tilde{g}(\tilde{x})$ are constructed in a similar fashion as $\tilde{f}(\tilde{x}) = (f(x), \partial_x f(x)x', \partial_x f(x)x'', ...)^\top$. Finally D is the motion shift operator or derivative operator that shifts the motions in such a way that $D\tilde{x} = \tilde{x}'$. This

means that D is a square matrix with identity matrices on its first off diagonal. Instead of a differential equation, eq (2-11) is an instantaneous probabilistic mapping of the belief about motions of state x. The noises are assumed to be zero mean Gaussian noises with differentiable covariance. The covariance matrix of the generalized noise of $\tilde{w}$ is constructed with a temporal variance matrix [24]. Without going into details how this works, the main point is that this covariance matrix is differentiable.

2-3-1 Generalized measurements

A major problem in Active Inference is how to construct the generalized measurement vector $\tilde{y}$? Real life sensors are limited and can only measure derivative of a physical signal up to some order. For example, position, velocity and acceleration are measurable. However, how is the third time derivative or higher of position measured?

When we take a look at the partitioned free energy equation of equation (2-7) and rewritten below, one can assume that the embedding order of the measurements $\tilde{\varepsilon}_y$ and the states $\tilde{\varepsilon}_x$ are independent. This is not the case, because the expression of $\tilde{\varepsilon}_y = \tilde{y} - g(\mu)$ ensures that the embedding order of the measurement must be equal to the embedding order of the states.

$$F(\mu, y) = \frac{1}{2}\tilde{\varepsilon}_x^\top \Pi_w \tilde{\varepsilon}_x + \frac{1}{2}\tilde{\varepsilon}_y^\top \Pi_z \tilde{\varepsilon}_y$$

In this equation $\tilde{\varepsilon}_y = \tilde{y} - g(\mu)$ is the sensory prediction error, with $\tilde{y}$ as the generalized measurements. Remember that $\mu = \tilde{\mu}_x$ is the generalized state when external forces and learning parameters are omitted.

A solution to equalize the embedding order is by padding the generalized measurement vector $\tilde{y}$ with zeros for the missing embedding orders. This is definitely possible as it was done before in simulations [27] and in real robotic systems [40] [10], but technically speaking it will introduce a bias. And on the other hand, the higher order beliefs will not affect the lower order beliefs much this way.

To make Active Inference perform better, the generalize measurement vector has to be constructed. A solution to create the generalized measurement vector is to have a real time differentiator estimate higher order derivatives of a signal. These can in theory estimate up to any order derivative. However, they are limited by the sample rate of signals and noise on the signal. Although they can attenuate noise, generally speaking the signal to noise ration reduces for each higher order derivative. Chapter 3 will go deeper into existing differentiators and chapter 4 shows the proposed differentiator.

Additionally, when constructing the generalized measurement vector $\tilde{y}$, the proper precision values of the noise are also necessary in Π_z. This means that the variances of the generalized measurement are required. A differentiator should be able to determine the variances of the noises for the estimated higher order measurements, however, this is not trivial. Chapter 4 goes into details about this.

Chapter 3

Methods to create Generalized Measurements

This chapter gives an overview of existing differentiators that can differentiate an input signal online. Their algorithms are listed and are described shortly. At the end of this chapter, each mentioned differentiator is tabulated within common categories and scored based on their relative performance from literature. This way it is easy to see what the best differentiator is and where the new proposed method is later to be fitted in.

3-1 The reason for differentiators

Differentiators have three advantages shortly mentioned in chapter 1. The first is that they are model free and are therefore able to estimate derivatives from any arbitrary signal. The second advantage is that differentiators work properly when the statistical properties of the noise are unknown or may vary over time. The third advantage is that they can improve a noisy signal. This means that a filtered signal has a lower variance than the raw noisy signal.

One might question the reason for using online differentiators when there are also other methods available that can estimate derivative terms such as the famous Kalman filter [31]. This is true, however, there are three reason for not using a Kalman filter. The first is that in this thesis, the dynamics of a system are unknown. This also means that the input to the system is not known. If a model would be used, it will be an autonomous model and has the form of a constant velocity or constant acceleration model for example. These models are commonly used in radar tracking. In fact Kalman filters have such a wide variety of applications in radar tracking that whole books are dedicated to it [43]. The model determines how good a Kalman filter performs [11], thus for the best results a precise model is needed. The downside of using a model based approach is that it models one specific system, which is not flexible.

The second reason is that derivative extraction is limited by the order of the model. This means that for a constant velocity model, the highest order derivative that can be extracted is the velocity. Taking the derivative of the velocity will result in zero acceleration, because a constant velocity model does not capture information about higher order derivatives. This means that other methods must be applied to get higher order derivative than the modeled ones, which is something differentiators can solve.

The third reason is that derivative extraction is not accurate at all if the model is not precise enough. A constant acceleration model is for example very bad at predicting a sinusoid. This makes it hard for a generalized model that can predict all sorts of functions.

Besides these reasons there are also several other factors that makes a Kalman filter hard to use. Such as the statistical properties of the noise must be known and the process noise must be tuned, which can be quite tedious. So for all the reasons above a Kalman filter is not considered for extracting higher order derivatives.

3-2 Two important remarks on continuous time differentiators

The first remark is that when a differentiator is defined in continuous time, it means that the input signal is also continuous. This results in accurate simulations of the differentiator, because the time step can be arbitrarily small. However, in real life the frequency of the differentiator is determined by the sampling rate of the input, usually sensor samples, which are discrete. This limits the continuous time differentiator performance significantly in practical applications. For example Taylor series expansion based differentiator (TSEBD) is very inaccurate for low values of ε while sampling it at a rate of 20 Hz. This means that the sampling rate has significant effect on the accuracy of the derivative estimation. As a general rule, the higher the sampling rate, the better the accuracy.

The second important remark is that when the best performing continuous time differentiators are converted to their discrete time version, they cannot estimate derivatives of a polynomial input. This is demonstrated in chapter 5, where the state-of-the-art differentiator is converted to discrete time. This problem occurs in TSEBD and AEAD, other methods have not been tested if they have the same problem. However, it is most likely that the same issue occurs in the other continuous time differentiators mentioned in this chapter as well.

3-3 Classical Differentiator (CD)

Differentiation of a digital signal is not trivial. The noise gets extremely amplified when a signal is differentiated in a classical sense as: $\dot{y}_k = \frac{1}{\Delta T}(y_k - y_{k-1})$. To fight this amplification, in classical control theory a differentiator is designed from an inertial system with a small time constant τ. This filter is as follows

$$Y = \frac{s}{\tau s + 1} U = \frac{1}{\tau}\left(1 - \frac{1}{\tau s + 1}\right) U. \tag{3-1}$$

This reduces the amplification of the noise with a proper chosen τ. However, this also introduces a time lag in the system equal to τ. Another downside is that when τ becomes very small equation 3-1 approaches the naive way of differentiating so that it becomes: $y(t) \approx \frac{1}{\tau}\left(u(t) - u(t-\tau)\right)) \approx \dot{u}(t)$. Resulting in amplification of the noise. These types of differentiators are very underperforming when dealing with high frequency signals and noise. There are numerous methods that combat these kind of effects as will be shown in the following sections.

3-4 Global robust exact differentiator (GRED)

The global robust exact differentiator (GRED) [35] [38] [46] [52] combines a sliding mode differentiator with a high gain differentiator and takes the weighted average between them. Although [35] proofs the existence of an arbitrary order robust differentiator, GRED is the realization for estimating the first derivative. For a first order derivative estimator, the GRED is defined as follows [52].

$$\begin{aligned} y_1 &= \alpha x_{11} + (1-\alpha) x_{21} \\ y_2 &= \beta x_{12} + (1-\beta) x_{22} \end{aligned} \tag{3-2}$$

where y_1 tracks the input $u(t)$ and y_2 represents the estimated derivative of $u(t)$. The high gain differentiator parameters x_{11} and x_{12} are defined as

$$\begin{aligned} \dot{x}_{11} &= x_{12} - \frac{a_1}{\tau}(x_{11} - u(t)) \\ \dot{x}_{12} &= -\frac{a_2}{\tau^2}(x_{11} - u(t)) \end{aligned} \tag{3-3}$$

And the sliding mode differentiator parameters are defined as:

$$\begin{aligned} \dot{x}_{21} &= x_{22} - \lambda_1 |x_{21} - u(t)| \operatorname{sgn}(x_{21} - u(t)) \\ \dot{x}_{22} &= -\lambda_2 \operatorname{sgn}(x_{21} - u(t)) \end{aligned} \tag{3-4}$$

In equation 3-2, α and β have special definitions and are used to switch between the two differentiators. They are defined as follows.

$$\alpha = \begin{cases} 0, & |e_1| < \varepsilon_1 - c_1 \\ \frac{|e_1| - \varepsilon_1 + c_1}{c_1}, & \varepsilon_1 - c_1 \leq |e_1| < \varepsilon_1 \\ 1, & |e_1| \geq \varepsilon_1 \end{cases}$$

and

$$\beta = \begin{cases} 0, & |e_2| < \varepsilon_2 - c_2 \\ \frac{|e_2| - \varepsilon_2 + c_2}{c_2}, & \varepsilon_2 - c_2 \leq |e_2| < \varepsilon_2 \\ 1, & |e_2| \geq \varepsilon_2 \end{cases}$$

where $e_1 = x_{21} - x_{11}$, $e_2 = x_{22} - x_{12}$. The positive tuning parameters are $\varepsilon_1 = \lambda_1 \tau$, $\varepsilon_2 = \lambda_2 \tau$, c_1 and c_2.

This differentiator can only estimate the first order derivative. It can estimate the derivative tolerable for low frequency signals (equal or lower than 1 rad/s, although this is not a formal definition) with and without noise as can be seen from Figure 3-2 and for the polynomial signal of 3-3. It has the highest tracking error when no noise is present of the differentiators in this chapter, surprisingly it seems to performs better than TSEBD when a polynomial signal is tracked with Gaussian noise (although the root mean square error (RMSE) of TSEBD could be lower). It can't track the signals for high frequency signals (10 rad/s) as can be seen in Figure 3-2c. Another issue is that GRED has chatter at the steady state. The reason for this is the use of the sign and a switching function.

3-5 Hybrid continuous nonlinear differentiator (HCND)

The hybrid continuous nonlinear differentiator (HCND) [53] is based on the GRED and reduces the chatter problem. It also has slightly better noise attenuation. HCND is based on a combination of a linear and a nonlinear differentiator. A second order hybrid system is as follows.

$$\begin{aligned} \dot{x}_1 &= x_2 - k_1 |x_1 - u(t)|^{\frac{\alpha+1}{2}} \operatorname{sgn}(x_1 - u(t)) - k_2(x_1 - u(t)) \\ \dot{x}_2 &= -k_3 |x_1 - u(t)|^{\alpha} \operatorname{sgn}(x_1 - u(t)) - k_4(x_1 - u(t)) \end{aligned} \tag{3-5}$$

where α, k_1, k_2, k_3 and k_4 are positive design parameters. α has to be in the range of $[0, 1]$. It must be chosen carefully to obtain high tracking performance. The k parameters are used to adjust the weight between the linear and non linear differentiator.

This differentiator works better than the GRED as can be seen in Figure 3-2 and 3-3, the combination of a nonlinear part makes it deal effectively with a large tracking error. It also has a significantly reduced chatter effect in the steady state error and it does not have a switching function.

However, it performs very poorly at high frequency derivative tracking as can be seen in 3-2c. The error is large and oscillates at the same frequency of the input signal.

3-6 Robust exact uniformly convergent arbitrary order differentiator (REUCOAD)

The robust exact convergent arbitrary order differentiator (REUCOAD) [4] [52] can estimate any order of derivative exact within finite time. When it is used as a first order estimator, REUCOAD is defined as follows.

$$\begin{aligned}\dot{x}_1 &= x_2 - \kappa_1\theta\,|x_1-u(t)|^{\frac{1}{2}}\,\mathrm{sgn}(x_1-u(t)) - k_1(1-\theta)|x_1-u(t)|^{\frac{1+\alpha}{2}}\,\mathrm{sgn}(x_1-u(t))\\ \dot{x}_2 &= \kappa_2\theta\,|x_1-u(t)|\,\mathrm{sgn}(x_1-u(t)) - k_2(1-\theta)|x_1-u(t)|^{1+\alpha}\,\mathrm{sgn}(x_1-u(t))\end{aligned} \tag{3-6}$$

where κ_1, κ_1, k_1, k_2, θ and α are the design/tuning parameters. It is tuned based on rules, which can be found in [4].

REUCOAD performs slightly better than HCND on high frequency signals, which can be seen from Figure 3-2c and the polynomial in Figure 3-3 This method also has the chattering phenomena due to the sign function as can be seen in the aforementioned figures. The only upside REUCOAD has is that it has the shortest convergence time [52].

3-7 Taylor series expansion based differentiator (TSEBD)

As the name says, the TSEBD is based on a truncated Taylor series expansion and is defined as follows [13].

$$r + \varepsilon\dot{r} + \frac{1}{2!}\varepsilon^2\ddot{r} + \frac{1}{3!}\varepsilon^3\dddot{r} = u(t) \tag{3-7}$$

This can be rewritten to extract the derivative estimations of $u(t)$:

$$\begin{aligned}\dddot{r} &+ \frac{3}{\varepsilon}\ddot{r} + \frac{6}{\varepsilon^2}\dot{r} + \frac{6}{\varepsilon^3}r = \frac{6}{\varepsilon^3}u(t)\\ \hat{u}^{(0)}(t) &= r(t+\varepsilon) \approx u(t)\\ \hat{u}^{(1)}(t) &= \frac{3}{\varepsilon}u(t) - \frac{3}{\varepsilon}r(t) - 2\dot{r}(t) - \frac{\varepsilon}{2}\ddot{r}(t)\\ \hat{u}^{(2)}(t) &= \frac{6}{\varepsilon^2}u(t) - \frac{6}{\varepsilon^2}r(t) - \frac{6}{\varepsilon}\dot{r}(t) - 2\ddot{r}(t)\end{aligned} \tag{3-8}$$

where ε is a small positive parameter. The smaller ε the higher the gain becomes. Which makes the filter faster and better on continuous noise free signals. This can be seen in Figure 3-1, where $u(t) = \sin(2t)$ is tracked with $\varepsilon = 0.005$. The high gain has the down side that it is very sensitive to noise, which can be seen in Figure 3-2b and 3-3b. The input terms of $\hat{u}^{(1)}(t)$ $\hat{u}^{(2)}(t)$ are scaled by $\frac{3}{\varepsilon}$ and $\frac{6}{\varepsilon^2}$ respectively, and thus will amplify noise that is on the input. The noise amplification can be reduced by tuning ε as was done by [32] in Figure 3-8 at the cost of accuracy and lag. In Figure 3-8 it can be seen that the zeroth derivative (top middle) lags behind the true signal.

Figure 3-2c shows that TSEBD cannot track the first derivative properly, although it follows the trend of it. This is due to a too high tuning parameter ε. A lower value of ε can track the first derivative of an even higher frequency signal as is shown in the middle plot Figure 3-8.

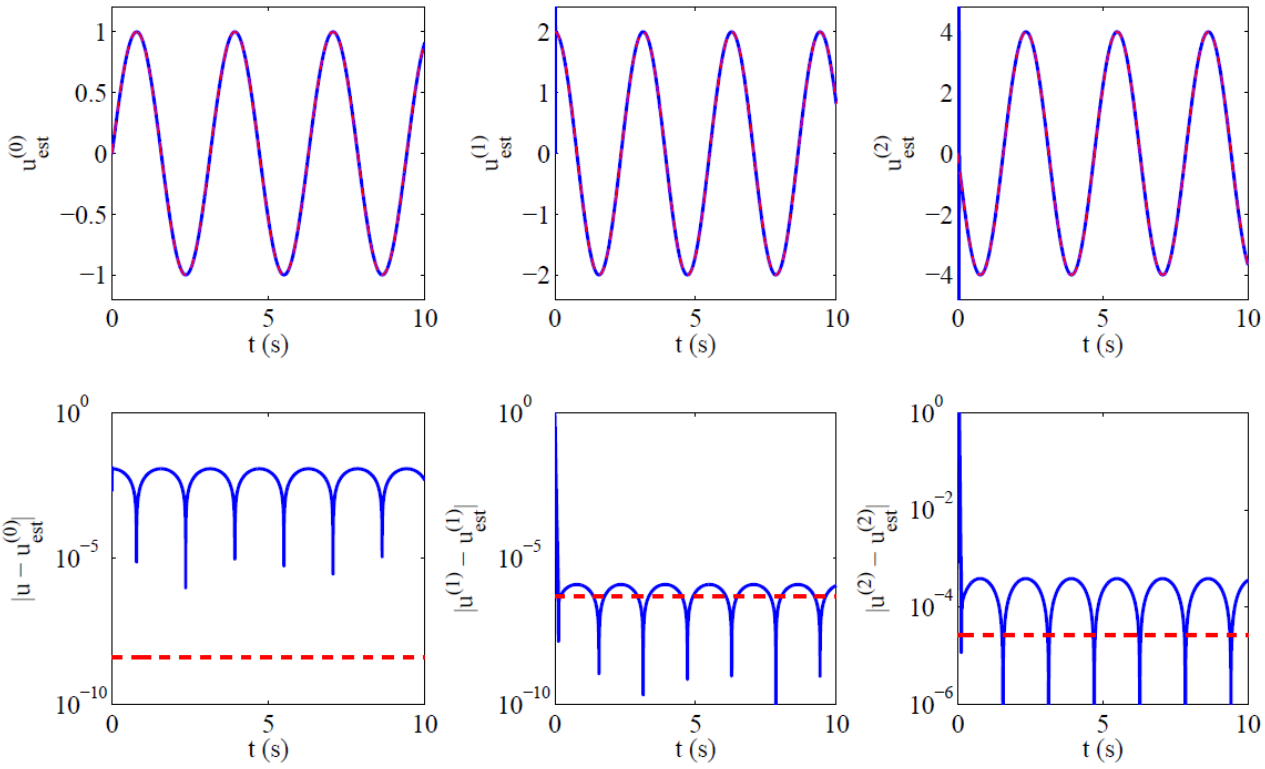


Figure 3-1: This figure is copied from [32] and shows the performance of TSEBD for the signal $u(t) = \sin(2t)$. The error bounds (red dashed lines) are the ones from AEAD in Figure 3-6. It can be seen that it has a very low error for the second order derivative for a low $\varepsilon = 0.005$.

3-8 Performance plots of GRED, HCND, REUCAOD and TSEBD

This section functions as support for the differentiators of the previous sections. It covers the plots of the GRED, HCND, REUCAOD and TSEBD. All plots and figures are copied from [52]. The following parameters were used for the differentiators [52].

1. GRED: $a_1 = 0.14$, $a_2 = 0.2$, $\tau = 0.1$, $\lambda_1 = 6$, $\lambda_2 = 28$, $\varepsilon_1 = 1$, $c_1 = 0.05$, $\varepsilon_2 = 0.5$ and $c_2 = 0.05$.
2. HCND: $\alpha = 0.2$ $k_1 = 1$, $k_3 = 8$. $k_2 = 7$ and $k_4 = 25$ when $t \leq 0$, else $k_2 = 1$ and $k_4 = 8$.
3. REUCAOD: $\kappa_1 = 6.2144$, $\kappa_2 = 20.48$, $k_1 = 7$, $k_2 = 2.1429$, $\theta = 0$ when $t \leq 1$ else $\theta = 1$ and $\alpha = 0.06$.
4. TSEBD: $\varepsilon = 0.1$, see section 3-7 for more details.

The follow signals are used:

1. $u(t) = \sin(t)$, which is the low frequency signal.
2. $u(t) = \sin(10t)$, which is the high frequency signal.
3. $u(t) = t^2 - t$, which is a polynomial and can be considered to have a zero frequency.

These signals are corrupted by a bounded Gaussian normal distributed noise that is bounded by 0.05, denoted as z_g. This means that the standard deviation of the noise is 0.05.

From the plots in Figure 3-2 and Figure 3-3 and the reasoning in the previous sections, TSEBD performs best with respect to the GRED, HCND and REUCAOD. The main reason is that it is

capable of tracking the higher order derivatives in the absence of noise, while the others can't track high frequency signals at all (without retuning the parameters). The capability of tracking higher order derivatives and high frequency signals is a very important requirement if a differentiator is used to construct the generalized measurements for use in Active Inference.

There are two downsides to TSEBD. The first is that it is only capable of estimating up to the second order derivative. The second is that it is very sensitive to noise, however, TSEBD can be tuned in such a way that the derivatives are still accurate at the cost of lag in the zeroth order derivative as was done in Figure 3-8.

3-9 Overlapping algebraic derivatives estimator (OADE)

The definition of the overlapping algebraic derivatives estimator (OADE) for the first three derivatives is as follows [32].

$$\hat{u}^{(j)}(t) = \begin{cases} \hat{u}_2^{(j)}(t), & (0 \leq \text{mod } T_r < T_r/2) \text{ AND } (t \geq \epsilon) \\ \hat{u}_1^{(j)}(t), & (T_r/2 \leq \text{mod } T_r < T_r) \text{ AND } (t \geq \epsilon) \end{cases} \tag{3-9}$$

where

$$\hat{u}_i^{(1)}(t) = \begin{cases} \frac{1}{t_i^7}[42t_i^6 u(t) + z_{i1}(t)], & \text{for } (t_i \geq \epsilon) \\ \hat{u}_i^{(1)}(t_i^-), & \text{for } 0 \leq t_i < \epsilon \end{cases} \tag{3-10}$$

$$\hat{u}_i^{(2)}(t) = \begin{cases} \frac{1}{t_i^8}[840t_i^6 u(t) + 35z_{i1}(t) + t_i z_{i2}(t)], & \text{for } (t_i \geq \epsilon) \\ \hat{u}_i^{(2)}(t_i^-), & \text{for } 0 \leq t_i < \epsilon \end{cases} \tag{3-11}$$

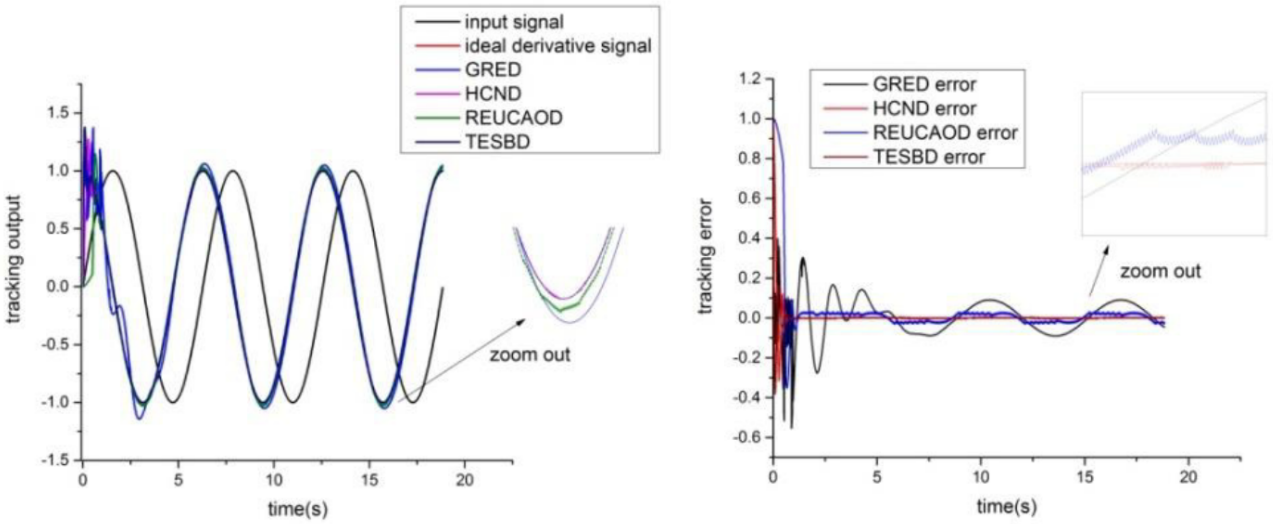
$$\hat{u}_i^{(3)}(t) = \begin{cases} \frac{1}{t_i^9}[10080t_i^6 u(t) + 560z_{i1}(t) + 28t_i z_{i2}(t) + t_i^3 z_{i3(t)}], & \text{for } (t_i \geq \epsilon) \\ \hat{u}_i^{(2)}(t_i^-), & \text{for } 0 \leq t_i < \epsilon \end{cases} \tag{3-12}$$

and the corresponding filter equations are

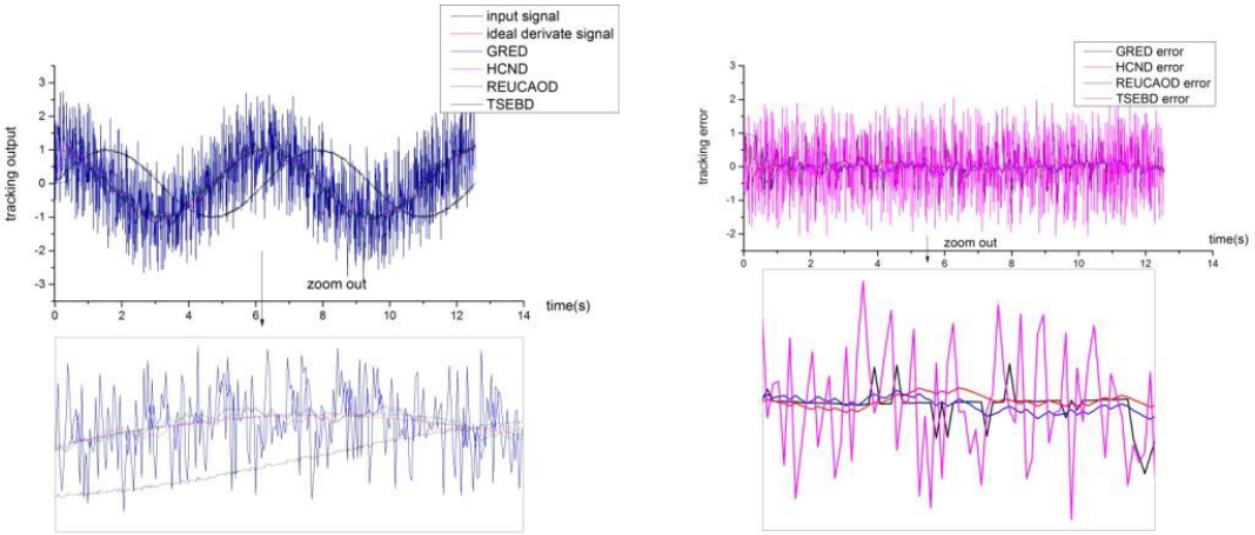
$$\begin{aligned} \dot{z}_{i1} &= -882t_i^5 u(t) + z_{i2} \\ \dot{z}_{i2} &= 7350t_i^4 u(t) + z_{i3} \\ \dot{z}_{i3} &= -29400t_i^3 u(t) + z_{i4} \\ \dot{z}_{i4} &= 52920t_i^2 u(t) + z_{i5} \\ \dot{z}_{i5} &= -35280t_i^1 u(t) + z_{i6} \\ \dot{z}_{i6} &= 5040u(t) \end{aligned} \tag{3-13}$$

This differentiator is also based on the Taylor series. The parameter T_r determines for how long the current estimated Taylor series is active for. This differentiator is then weighted with another one that runs concurrent with the first one, but has a time offset of $T_r/2$, see the modulo operation in equation (3-9).

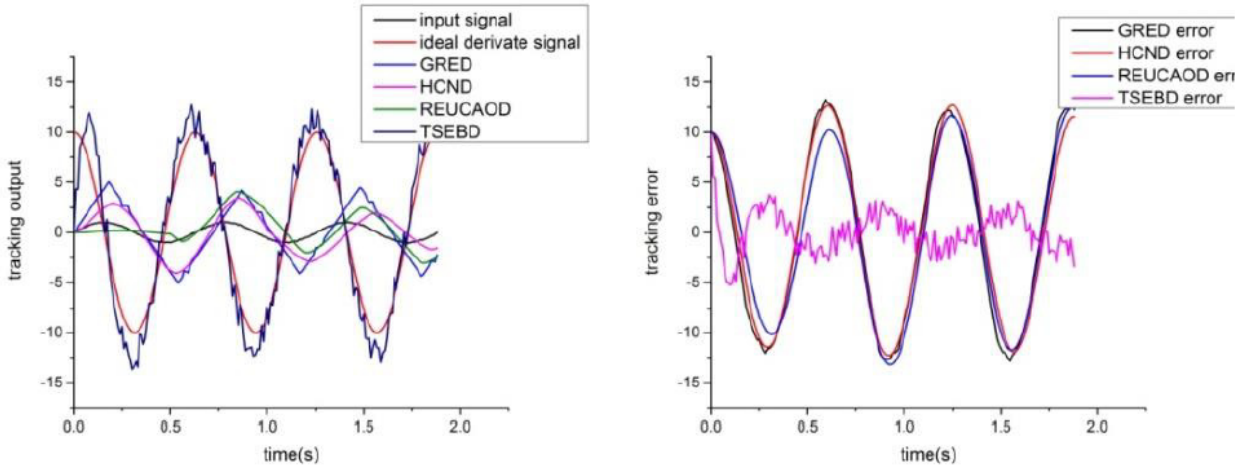
Figure 3-4 shows that it is capable of tracking the derivatives with a relatively low error. It outperforms the GRED, HCDN and REUCAOD in that regard. It can also track high frequency signals with noise as shown in Figure 3-8, however, the noise attenuation is not that strong. Another downside is that is has to be periodically reinitialized to maintain accuracy during the estimation process. For that reason OADE is rated just behind TSEBD.



(a) $u(t) = \sin(t)$

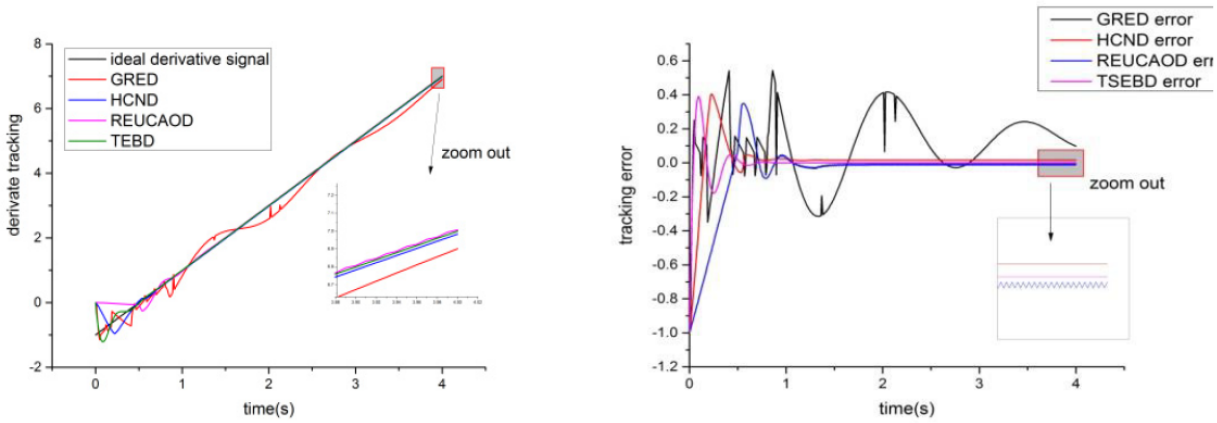


(b) $u(t) = \sin(t) + z_g$

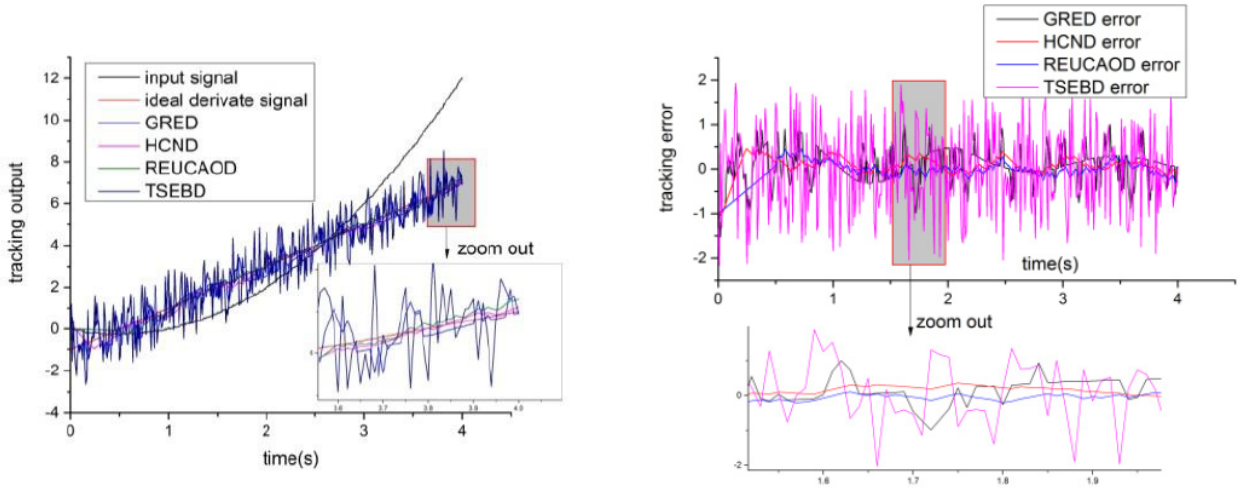


(c) $u(t) = \sin(10t) + z_g$

Figure 3-2: These figures are copied from [52] and show how the differentiators track the first derivative of sinusoid inputs. The errors are plotted at the right hand side. In (a) and (b) the input is $\sin(t)$, but in (b) this input is corrupted by additive Gaussian noise (z_g). The same goes for (c), but the frequency of the sine is increased so that the input is $\sin(10t) + z_g$. What stands out is that TSEBD can track the trend of the derivative in (c) and has the lowest error. However, (b) shows that the TSEBD is very sensitive to the noise.



(a)



(b)

Figure 3-3: These figures are copied from [52] and show how the differentiators track the first derivative of a polynomial input. The errors are plotted at the right hand side. In (a) and (b) the input is $u(t) = t^2 - t$, but in (b) this input is corrupted by additive Gaussian noise (z_g). TSEBD performs the best in (a), as it has a zero steady state error. What stands out in (b) is that the tracking error of TSEBD has a very large variance around zero. It is a lot more sensitive to noise than the others differentiators, although this could be reduced by having a larger ε.

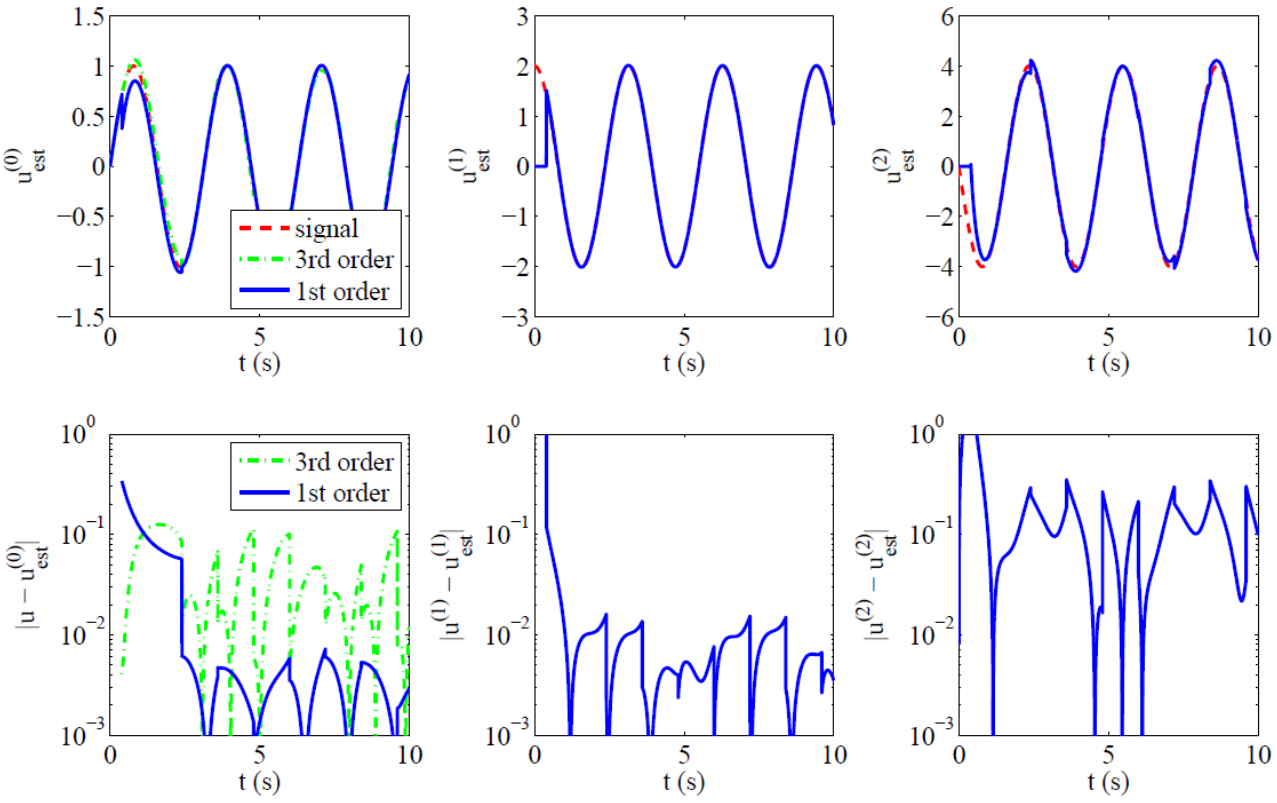


Figure 3-4: This figure is copied from [32]. This shows that the OADE is capable of tracking higher order derivatives with a relatively low error. However, OADE performs worse than TSEBD for the tracking error in Figure 3-1. The flat line at the start of the derivatives are due to initialization of the first estimates in a small time window ϵ. After that the main filter activates.

3-10 Algebraic estimation approach differentiator (AEAD)

The algebraic estimation approach differentiator (AEAD) [32] is based on a truncated Taylor series expansion. It makes use of a differentiable function that has the same properties of the Dirac delta function to create the estimator. That way the transfer function can be represented by a serial-parallel connections of the basic transfer functions $G(s) = a/(s+a)$ that have low pass characteristics. It is formulated conveniently in a continuous time state space format and is a follows.

$$\dot{\mathbf{x}}(t) = A\mathbf{x}(t) + Bu(t) \tag{3-14}$$
$$\hat{\mathbf{u}}(t) = \bar{M}^{-1}C\mathbf{x}(t) \tag{3-15}$$

The matrices A and B are

$$A = \begin{bmatrix} -a & 0 & 0 & \dots & 0 \\ a & -a & 0 & \dots & 0 \\ 0 & a & -a & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & -a \end{bmatrix}, \quad B = \begin{bmatrix} a \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{3-16}$$

with cutoff frequency $a > 0$ as tunable parameter. It is recommended to set the cutoff frequency slightly higher than the fastest system frequency. Although the transient response gets faster with higher cutoff frequency, it does not necessarily mean that the estimates gets better, unlike TSEBD. A too high value for a and the system estimates get worse, the same goes for a too low cutoff frequency. This tunable parameters also determines the eigenvalues of the A matrix, because all are $-a$. This means that the system is always stable.

The matrix C is defined as

$$C = \begin{bmatrix} 0 & \dots & 0 & 0 & \mu_{10} \\ 0 & \dots & 0 & \mu_{20} & \mu_{21} \\ 0 & \dots & \mu_{30} & \mu_{31} & \mu_{32} \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ \mu_{n,0} & \dots & \mu_{n,n-3} & \mu_{n,n-2} & \mu_{n,n-1} \end{bmatrix}, \tag{3-17}$$

where the indexed elements of μ are defined as binomial expansion

$$\mu_{ik} = (-1)^k a^{(i-1)} \begin{pmatrix} i-1 \\ k \end{pmatrix}, \quad i = 1, 2, ..., n, \quad k = 0, 1, 2, ..., n-1 \tag{3-18}$$

Finally the matrix $\bar{M}$ is defined as

$$\bar{M} = \begin{bmatrix} 1 & \frac{\hat{m}_1}{a} & \frac{\hat{m}_2}{a^2} & \frac{\hat{m}_3}{a^3} & \dots & \frac{\hat{m}_{n-1}}{a^{n-1}} \\ 0 & 1 & \frac{\hat{m}_1}{a} & \frac{\hat{m}_2}{a^2} & \dots & \frac{\hat{m}_{n-2}}{a^{n-2}} \\ 0 & 0 & 1 & \frac{\hat{m}_1}{a} & \dots & \frac{\hat{m}_{n-3}}{a^{n-3}} \\ 0 & 0 & 0 & 1 & \dots & \frac{\hat{m}_{n-4}}{a^{n-4}} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 1 \end{bmatrix}, \tag{3-19}$$

and here are the values of $\hat{m}$ are also defined as binomial expansion as

$$\hat{m}_k = (-1)^k \begin{pmatrix} n+k-1 \\ k \end{pmatrix}, \quad k = 1, 2, ..., n-1 \tag{3-20}$$

For efficiency, the inverse of $\bar{M}$ can be calculated in a similar fashion without the need for calculating the inverse after constructing $\bar{M}$. This inverse $W = \bar{M}^{-1}$ is defined in [32].

Figure 3-5 and 3-6 show the performance of AEAD without noise on a signal $u(t) = \sin(2t)$ for the first five derivatives. It can be seen that it can track the higher order derivatives extremely well. The tracking error increases over each successive derivative, but that is expected from a truncated Taylor series expansion. This is also seen more clearly in Figure 3-7, where the error is plotted with respect to the tuning parameters. Adding more derivatives to the estimator increases the accuracy.

Figure 3-8 shows the first two derivatives estimation of AEAD, TSEBD and OADE. The input signal used is $u(t) = \sin(20t) + 0.01\mathcal{N}(0, 1)$, where $\mathcal{N}(0, 1)$ is a Gaussian noise normal distribution. The parameters used for the AEAD are $a = 30$ and $n = 10$. For TSEBD the parameter is $\varepsilon = 0.01$. For OADE the parameters are $T_r = 0.3$ and $\epsilon = 0.08$. What stands out is that the AEAD has the best noise attenuation and it also tracks the signal extremely well, although it has a slower transient than the other two. TSEBD still performs quite well, however, the noise amplification is clearly visible in the bottom plot. TSEBD has some lag on the zeroth order derivative due to the relatively high ε parameter, but it gives better noise attenuation and the lag has hardly effect on the first and second derivative. OADE performs the worst of the three when noise is present. AEAD outperforms these two and is therefore considered the state-of-the-art differentiator.

Although AEAD is claimed to be the best differentiator, it turns out that the discrete time version of AEAD can't track a polynomial input. The same is true for TSEBD and most likely for the other differentiators as well, although this has not been validated. The derivative estimates of AEAD diverge if a second degree or higher polynomial is used. This is shown later in Chapter 5. This divergence sounds like AEAD is unstable in discrete time, but that is not the case. The derivative estimates are diverging, even though the internal system stable.

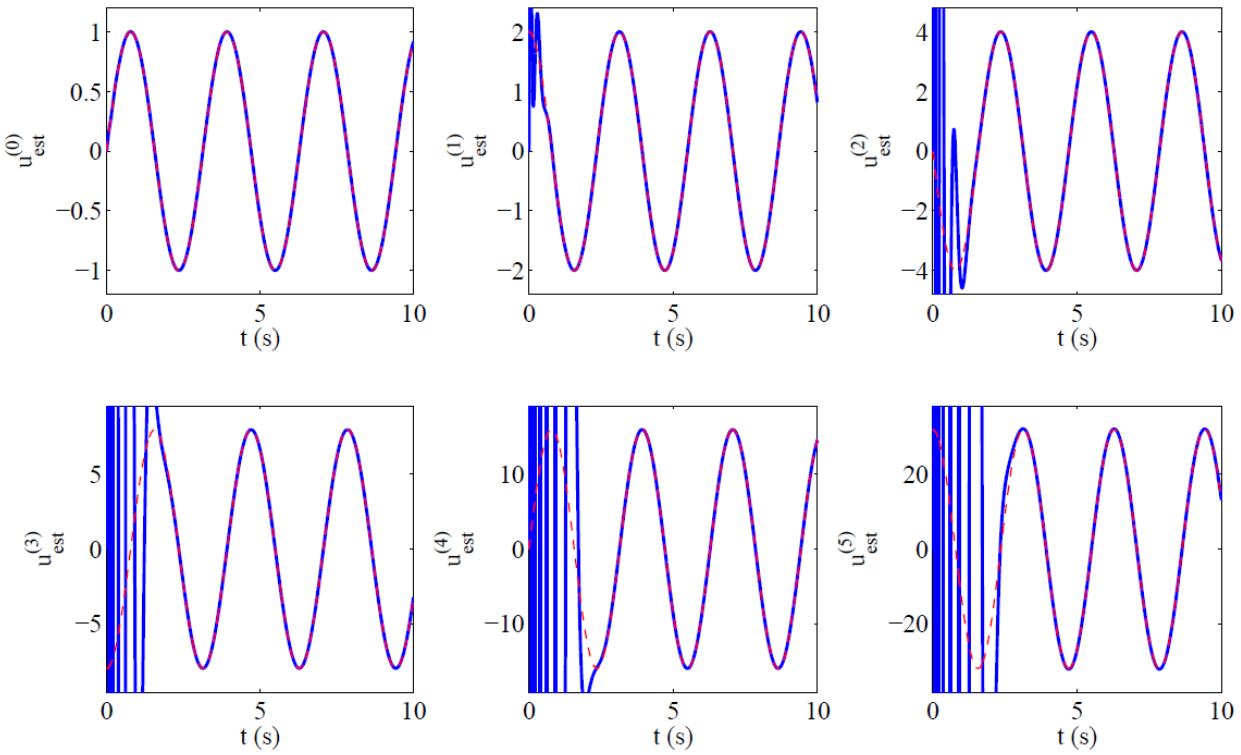


Figure 3-5: This figure is copied from [32] and shows the tracking performance of the AEAD. The red dashed line is the analytical solution and the blue line is the estimated derivative of the input signal $u(t) = \sin(2t)$. The estimations are denoted by $u_{est}^{(i)}$, where i denotes the i-th derivative. It tracks the true derivative very good with the absence of noise. It has sharply peaked transients for the higher order derivatives. The corresponding errors are plotted in Figure 3-6. The parameters that were used are $a = 10$ and $n = 12$.

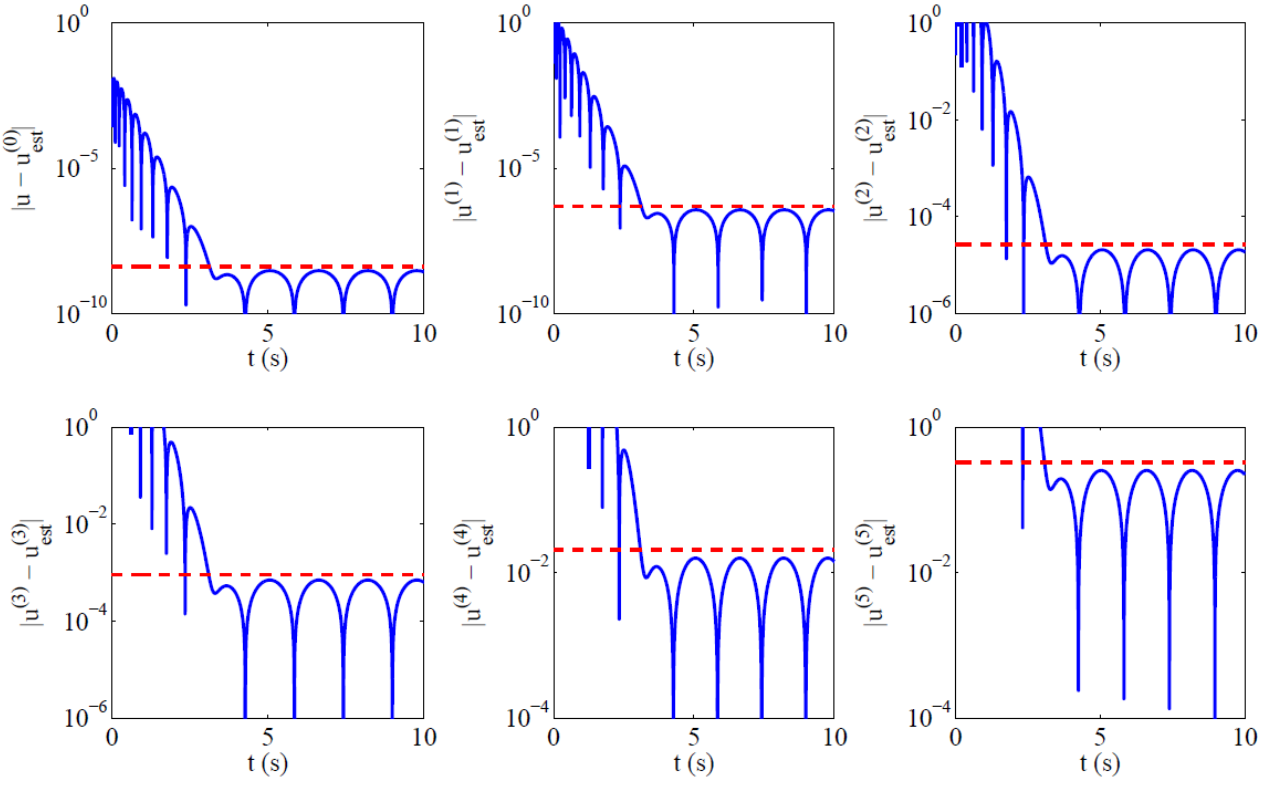


Figure 3-6: This figure is copied from [32] and shows the tracking error of Figure 3-5. The red dashed line shows the bound of the estimation errors. It can be seen that the error is low at first and increases over the amount of derivatives estimated.

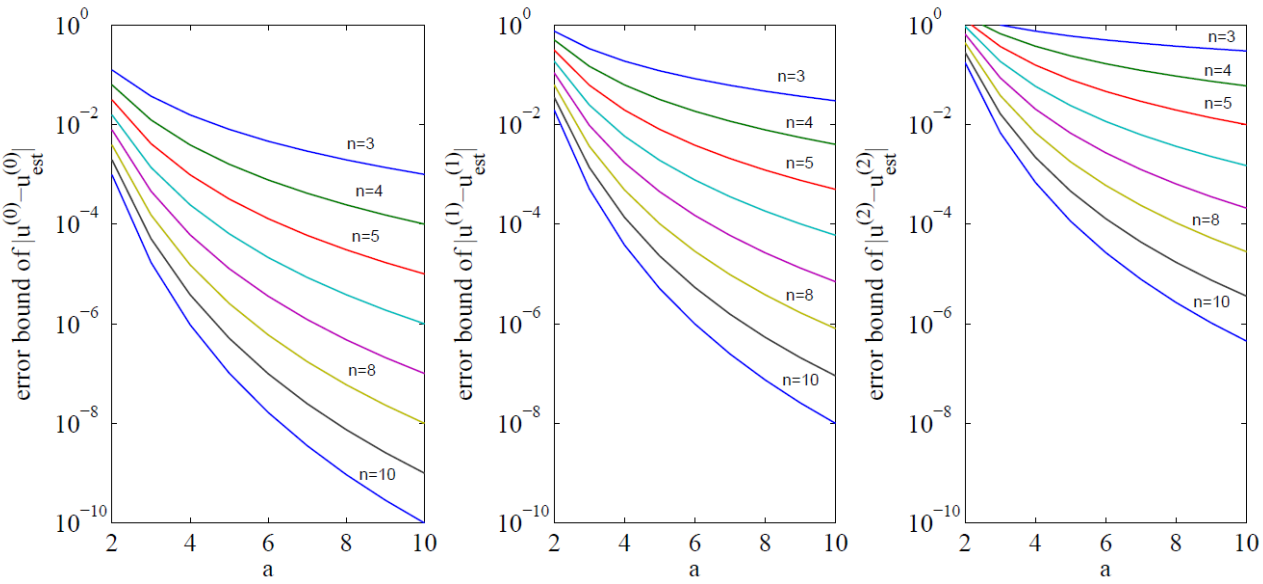


Figure 3-7: This figure is copied from [32] and shows how the AEAD tracking improves by increasing the amount of states n and is plotted over the cutoff frequency a. These upper bound errors are computed based on the limit expression when time goes to infinity [32]. It can be clearly seen by adding more states, the differentiator improves its accuracy. However, the accuracy reduces for every step of higher order derivative with respect to the base signal.

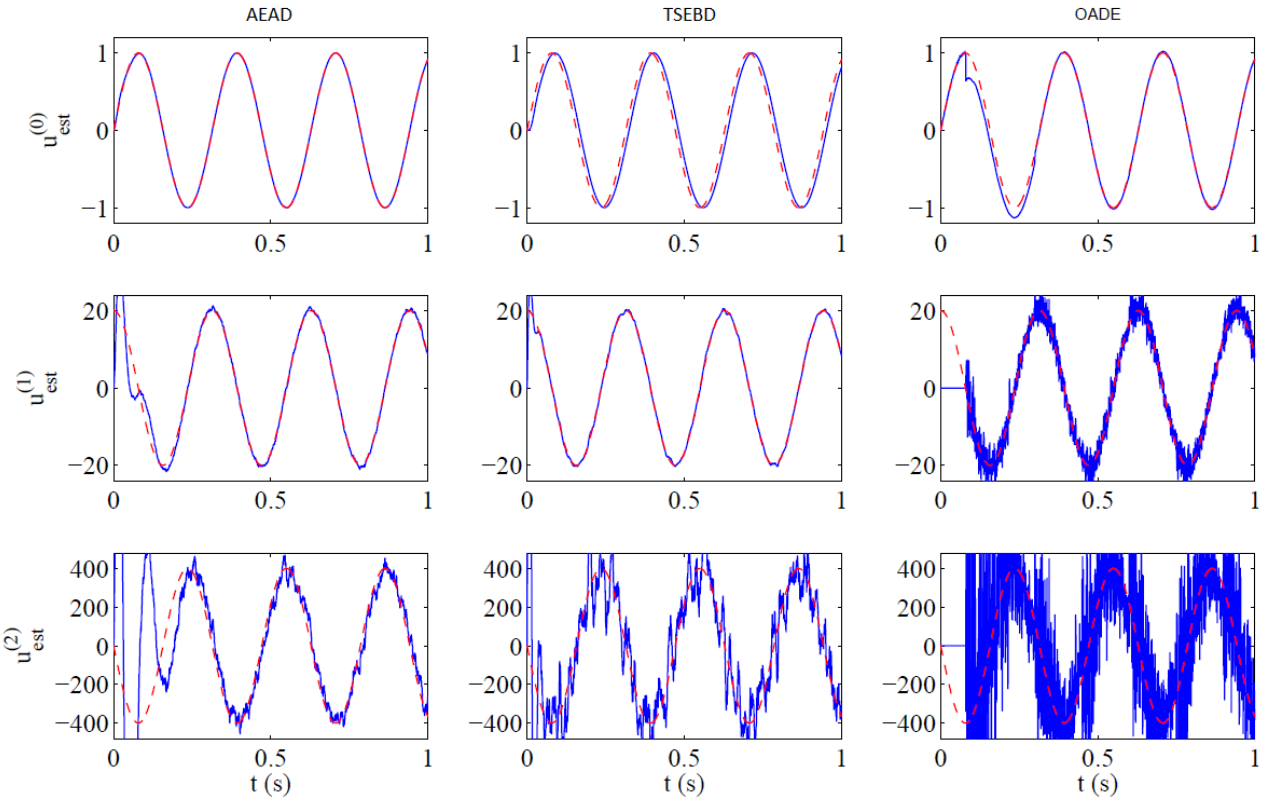


Figure 3-8: This figure is copied from [32] and shows the performance of AEAD, TSEBD and OADE on a high frequency signal with noise. The function to track is $u(t) = \sin(20t) + 0.01\mathcal{N}(0,1)$, and the dashed red lines are the analytical derivatives. When looking at the bottom three plots, it can be seen that AEAD outperforms the other two significantly in noise attenuation and estimation at the cost of a slower transient.

3-11 Categorizing the differentiators

There are two important categories for real time differentiators based on the available literature. The first category is their capability to track derivatives of a signal, which can be subdivided into low and high frequency signals. The second category is how well they attenuate noisy signals. Table 3-1 shows the scores of each differentiator mentioned in this chapter. The scores range from 0 to 10 and are mainly based on the figures and literature throughout this chapter.

The table consist of the following categories. The most important category is the performance metric, this covers derivative tracking of low and high frequency signals and noise attenuation. A less, but still important metric is how many derivatives can be estimated (the more the better) and how many tunable parameters the differentiator has (the lower the better). The last category covers the underlying technique that the differentiator uses, a Taylor series based approach (TSB) or a sliding mode technique (SMT). A backwards finite difference scheme (BFD) and the classical differentiator (CD) from equation 3-1 are also included for reference. Note that all differentiators apart from the BFD are defined in continuous time.

AEAD scores the best out of the differentiators mentioned in this chapter. This was also clearly shown by the convincing plot of Figure 3-8. It shows that it outperforms TSEBD and OADE, which are second and third respectively. AEAD is capable of tracking high frequency signals as well as estimate any order of derivatives. This is reflected in the scores of AEAD in Table 3-1. TSEBD scores second best if tuned properly. The effects of a good tuned TSEBD can be seen in Figure 3-8 vs the one of 3-2c. Due to this conflict the score of this differentiator has been given a range. What also stands out is that every good performing differentiator makes use of a Taylor series based approach. It shows that these methods have at most two tunable parameters that are easily understandable. This makes the TSB methods a lot easier to tune with respect to the SMT based methods, which have more than four tunable parameters. And those parameters do not make physically sense of how they affect the performance and will require lots of trial and error to get right.

	Performance of eight types of differentiators					
Method	LF derivative tracking	HF derivative tracking	noise attenuation	max derivatives	tunable parameters	underlying technique
BFD	0.0	0.0	0.0	> 1	0	TSB
CD	1.0 to 3.0	1.0	1.0 to 5.0	1	1	-
GRED	4.0	2.0	5.0	1	6	SMT
HCND	7.0	2.0	4.0	1	5	SMT
REUCAOD	6.0	3.0	4.0	> 1	6	SMT
OADE	7.0	4.0	5.0	> 1	2	TSB
TSEBD	4.0 to 7.0	2.0 to 5.0	1.0 to 6.0	2	1	TSB
AEAD	9.0	7.0	8.0	> 1	2	TSB

Table 3-1: This table lists one discrete time (BFD) and seven continuous time differentiators to show how they perform in each category. LF and HF denotes low frequency and high frequency respectively. A greater number indicates a better score, except for the tunable parameters catagory. SMT denotes the sliding mode technique and TSB is a Taylor series based technique. The CD (classical differentiator) and TSEBD have a range of scores, because their tuning parameter can have a lot of effect on the noise attenuation and thus the tracking accuracy at the cost of lag. From this table its easy to see that the top performing differentiators are the last three. They have in common that they are all Taylor series based (TSB), but the best one is clearly the AEAD.

Chapter 4

Proposed method for finding Generalized Measurements

The previous chapter shows that it is possible to extract higher order derivatives from a single signal. The best performing differentiators are the ones that make use of a truncated Taylor series. The proposed method in this chapter uses a Taylor series and first order solution to a differential equation at its base, which can be exploited by differentiation to get higher order derivatives. Section 4-1 explains the proposed method thoroughly.

The inspiration for finding higher order derivatives are ordinary differential equations (ODE), because their solutions are usually exponential functions and can be differentiated multiple times. These are the backbone of the proposed method and are explained and exploited in section 4-2 to 4-4.

The main concept of finding higher order derivatives of a single signal is that it is possible to create a dynamic system that tracks the signal in real time. When a system is dynamic, it can be differentiated and therefore allows for extraction of higher order derivatives. The proof of concept of the proposed differentiator is shown in 4-5. The system is not always stable, therefore the stability of the system is analyzed in section 4-6. Finally this chapter ends with a noise analysis in section 4-8 to show how the standard deviation of noise propagates per extra derivative estimated.

4-1 Proposed method: Recurrent Low Pass Algebraic Differentiator (RL-PAD)

The proposed causal differentiator that can estimate higher derivatives and track a polynomial signal is defined in discrete time by the following two recurrence equations.

$$y_{k+1}^{(i)} = \left(y_k^{(i)} - u_k^{(i)}\right) e^{-\omega_i h} + u_k^{(i)} + \sum_{j=1}^{n-i-1} \frac{h^j}{j!} y_k^{(j)} \quad \text{for } i = 0, 1, ..., n-1 \tag{4-1}$$

$$u_{k+1}^{(i+1)} = -\omega_i \left(y_k^{(i)} - u_k^{(i)}\right) e^{-\omega_i h} + \sum_{j=0}^{n-i-2} \frac{h^j}{j!} y_k^{(j)} \quad \text{for } i = 0, 1, ..., n-2 \tag{4-2}$$

In a nutshell, equation (4-1) and (4-2) continuously estimates the Taylor series expansion coefficients of the external input signal $u_k^{(0)}$ at any k by using low pass filtered derivative terms of itself. The analytic definition of a low pass filter makes it differentiable and is used as an a priori value of the derivatives $u_k^{(i+1)}$ that is used in the $k+1$ step. At that step these derivatives are filtered again in

(4-1) by using them as inputs to steer the system in the proper direction until it converges towards the true values of the Taylor expansion of the true input.

Equation (4-1) outputs the derivative estimate $y_{k+1}^{(i)}$, where the superscript (i) denotes the order of derivative estimate. The amount of derivatives that are estimated go up to n, which also includes the 0th derivative. This means that for $n = 2$, the zeroth ($y_{k+1}^{(0)}$) and first derivative ($y_{k+1}^{(1)}$) are estimated. The input to this equation is represented by $u_k^{(i)}$. The most important input is at $i = 0$. This is the external signal sample whose derivatives are estimated and is represented by $u_k^{(0)}$.

Equation (4-2) determines the inputs $u_{k+1}^{(i+1)}$ for the higher order derivative estimates of $y_{k+1}^{(i)}$. These updated inputs are in fact the derivative with respect to h of equation (4-1), with the assumption that $y_k^{(i)}$ and $u_k^{(i)}$ are constant. This equation adds two types of dynamics to the estimations of $y_{k+1}^{(i)}$. The first is due to the low pass filter and the second is that the output $u_{k+1}^{(i)}$ is used at the next step of equation (4-1) that adds a lag. Thus the system order of (4-1) is $2n - 1$.

The term h is in both equations the time step and the cutoff frequencies are ω_i, which are tunable parameters. The summation term represents a truncated Taylor series expansion, but note that in equation (4-1) the series start at the second term.

When these equations work in unison, they do the following. The input is fed through a first order low pass filter term and is then compensated by the truncated Taylor series expansion. This significantly reduces lag for the output $y_{k+1}^{(i+1)}$. Note that the Taylor series expansion requires the terms $y_k^{(j)}$ which are based on $y^{(0)}$ of previous samples. To determine the higher order derivative estimates, equation (4-2) calculates the derivative of $y_k^{(i)}$ with respect to the time step h. This derivative term $u_k^{(i+1)}$, will approach the real derivative of a signal over time. However, this term is always larger when there is a mismatch between the true and real derivative. Therefore it is filtered through the low pass filter in equation (4-1) at the next step to attenuate this high value and be compensated for lag by the summation term. These low pass filters at every step are the inspiration of the name for the proposed method. In essence the two steps that are repeated each time are: Firstly update $y_{k+1}^{(i)}$ by using the derivatives of the previous state $u_k^{(i)}$, but remember that $u_k^{(0)}$ is always the raw input of the external signal. Secondly use this $y_{k+1}^{(i)}$ to calculate $u_{k+1}^{(i)}$ and set $u_k^{(0)}$ properly.

An intuitive interpretation why the algorithm can estimate coefficients of the Taylor series expansion of the input is as follows. Suppose we have a signal $r(t)$ that can be expanded as a Taylor series around $t = 0$ as

$$r(t) = r_0 + \dot{r}_0 t + \frac{1}{2}\ddot{r}_0 t^2 + ...$$

Converting it to discrete time with time step h results in

$$r_{k+1} = r_k + \dot{r}_k h + \frac{1}{2}\ddot{r}_k h^2 + ...$$

Expanding equation (4-1) for the zeroth derivative ($i = 0$) and using r_k as input we get

$$y_{k+1}^{(0)} = \left(y_k^{(0)} - r_k\right) e^{-\omega_0 h} + r_k + y_k^{(1)} h + \frac{1}{2} y_k^{(2)} h^2 + ...$$

Assuming that the algorithm is stable and converges over time, then when the term $\left(y_k^{(0)} - r_k\right) \approx 0$ for all k steps after that (this means that also $\left(y_{k+1}^{(0)} - r_{k+1}\right) \approx 0$), the remainder of equation (4-1) will approximately be

$$y_{k+1}^{(0)} \approx r_{k+1} \approx r_k + y_k^{(1)} h + \frac{1}{2} y_k^{(2)} h^2 + ...$$

This means that

$$r_k + y_k^{(1)} h + \frac{1}{2} y_k^{(2)} h^2 + ... \quad \approx \quad r_k + \dot{r}_k h + \frac{1}{2}\ddot{r}_k h^2 + ... \tag{4-3}$$

and therefore

$$\begin{aligned} y_k^{(1)} &\approx \dot{r}_k \\ y_k^{(2)} &\approx \ddot{r}_k \\ &\vdots \\ y_k^{(n-1)} &\approx r_k^{(n-1)} \end{aligned}$$

Thus the derivatives of r_k are found by estimating the coefficients of the Taylor series expansion. It is important to note that the higher order derivatives become progressively less accurate, because each order is based on the derivative estimate of one order lower. The reason for this is that $y_k^{(1)}$ is based on $u_k^{(0)}$ and derivative estimate $y_k^{(2)}$ is based on the input estimate of $y_k^{(1)}$. This dependency of estimations based on estimations causes the accuracy of higher order derivatives estimates to decrease.

Important tuning parameters are the cutoff frequencies ω_i for $i = 0, 1, ..., n-1$ where n represents the amount of derivative estimates in the system (including the 0th) and i denotes the order of term for each cutoff frequency. This means that $i = 0$ is for the 0th derivative, $i = 1$ for the first, $i = 2$ for the second, etc. The main transient (of the zeroth derivative) is determined by ω_0. It is necessary that $\omega_{i+1} < \omega_i$, because it is important to have the lowest derivative converge properly before orders above it do. This means that ω_i for $i \neq 0$ controls the damping of the response. The consecutive decreasing values of ω_i have the downside that the bandwidth for higher order derivatives gets lower for each consecutive derivative estimation step and therefore, estimations of higher derivatives decrease in accuracy rapidly. This phenomenon is also observed in simulations of the AEAD.

Tuning every ω_i individually is not an easy task, therefore to reduce the amount of tuning parameters to three ω_i is defined as

$$\omega_{i+1} = \frac{1}{f_{red}}\omega_i, \quad \text{for } i = 0, 1, ..., n-2, \quad \text{and adviced to have } f_{red} >= 2.0. \tag{4-4}$$

It can also be described in terms of ω_0 as

$$\omega_{i+1} = \frac{1}{(f_{red})^{i+1}}\omega_0$$

The tuning parameters are now ω_0, f_{red} and n and are all positively valued. The rise time is determined by ω_0. The parameter f_{red} controls the damping of the response, a higher value adds more damping. Note that the time step h is not considered a tuning parameter, because this is determined by the sample rate of a system. This makes tuning a lot easier for large n, although the values of ω_i can be individually tuned for better performance. The reduction factor $f_{red} >= 2.0$ is empirically chosen so that it gives a good damped response for the zero-th derivative. The advised lower limit of $f_{red} = 2.0$ gives a good balance between ω_0, h and n before the system becomes unstable. But note that the stability is based on a combination of ω_i, n and h together.

The transient response is determined by the parameters ω_0, f_{red}, n and also the time step h. A smaller h allows for a faster response, because it increases the bandwidth of ω_0. This means that ω_0 can be higher so that lowest cutoff frequency $\omega_{n-1} = \omega_0 / (f_{red})^{n-1}$ is also higher. For the best estimation results, the cutoff frequency ω_{n-1} associated with the highest derivative should have a cutoff frequency that is higher than the input its highest frequency. The amount of derivatives (n) also affects the response, it becomes slower when more derivatives are estimated. The reason for this is that higher derivatives are progressively adding slower dynamics to the system and thus they take longer to converge. This slow converging is then fed back through all lower order derivatives resulting in them taking more time to settle. As mentioned before, this behavior is also observed in AEAD.

This method can be seen as hybrid between TSEBD and AEAD, the proposed method has a similar formulation to TSEBD, but with the noise attenuation of AEAD. The clear appearance of the truncated Taylor series is what the proposed method has in common with TSEBD. AEAD is also based on a truncated Taylor series, but it is convoluted in the matrix equations. The response based on the tuning

parameter of TSEBD and the proposed method are also similar, the higher the value the faster the response and the more accurate the estimates become. However, TSEBD estimates the derivatives without filtering them resulting in a high noise sensitivity, in contrast the proposed method does filter the signals before estimating the derivatives and is therefore less sensitive to noise. TSEBD estimates the derivatives without filtering them, in contrast the proposed method does filter the signals before estimating the derivatives. This is similar to the AEAD, that filters higher order derivatives of its internal system through serial-parallel connected low pass filters and estimates the derivative based on a high gain measurement matrix. The proposed method does not have a high gain measurement matrix as is shown in the state space formulation of section 4-1-1. The proposed method has stronger noise attenuation for the higher order derivatives, because the recurrent low pass filter makes sure that noise and derivative estimates are contained within finite bounds or go to zero depending on the filter parameters used.

To improve the readability of this chapter, the outputs of equation (4-1) and (4-2) are split up with a different notation and are as follows. The state estimate $y_k^{(i)}$ is notated by

$$\begin{aligned} y_k &= y_k^{(0)} \\ \hat{y}_k^{(i+1)} &= y_k^{(i+1)}, \end{aligned}$$

where y_k denotes the zeroth order derivative estimate and $\hat{y}_k^{(1+i)}$ the higher order derivatives. These are also denoted as dot notation extensively. The input estimates $u_k^{(i)}$ is split up and notated as

$$\begin{aligned} r_k &= u_k^{(0)} \\ \frac{d}{dh} y_k &= u_k^{(1)} \\ \frac{d}{dh} \hat{y}_k^{(i+1)} &= u_k^{(i+1)} \end{aligned}$$

The true input is represented by r_k that comes from an external source, such as sensor samples. The input estimates that are used to estimate higher order derivatives are $\frac{d}{dh} y_k$ and $\frac{d}{dh} \hat{y}_k^{(i+1)}$. They are denoted by $\frac{d}{dh}$ of the state estimate, because that is a better understandable notation for the derivation of the proposed method. The terms with prefix $\frac{d}{dh}$ are also called raw derivatives throughout this chapter.

4-1-1 RLPAD discrete time state space representation

The equations (4-1) and (4-2) have no way of directly telling that the system is stable if used with a set of parameters ω_i, h and n. A solution to this would be to convert them to a discrete time state space so that the system matrix can be analyzed for stability. The RLPAD can be written as a linear system of equations as

$$\begin{aligned} \mathbf{x}_{k+1} &= A\mathbf{x}_k + Bu_k \\ \mathbf{y}_k &= C\mathbf{x}_k \end{aligned} \tag{4-5}$$

The matrix A is constructed by the following conditional expression.

$$A_{ij} = \begin{cases} e_\alpha \mathcal{S}_{j+1}, & \text{if } i = j \\ -\omega_\alpha e_\alpha \mathcal{S}_{j+1} + (1 - e_\beta)\mathcal{S}_j, & \text{if } i = j+1 \\ \omega_\beta e_\beta \mathcal{S}_j, & \text{if } i = j+2 \\ \dfrac{1}{(\beta-\gamma)!} h^{(\beta-\gamma)!} \mathcal{S}_{j+1}, & \text{if } i < j \\ 0, & \text{otherwise} \end{cases} \tag{4-6}$$

Here is $e_\alpha = e^{-\omega_\alpha h}$ and the same goes for e_β. The integer values α, β and γ are defined as the floored values of $\alpha = \left\lfloor \frac{j-1}{2} \right\rfloor$, $\beta = \left\lfloor \frac{j}{2} \right\rfloor$ and $\gamma = \left\lfloor \frac{i}{2} \right\rfloor$. $\mathcal{S}_j$ is an altering series that outputs 0 if j is odd and 1 if even. It is defined as

$$\mathcal{S}_j = \frac{(-1)^j + 1}{2}, \quad j \in \mathbb{Z}$$

This is also used in the construction of the non square C matrix. This C matrix has a size of $(n \times 2n-1)$ and is defined as

$$C_{ij} = \begin{cases} \mathcal{S}_j, & \text{if } 2i-1 = j \\ 0, & \text{otherwise} \end{cases} \tag{4-7}$$

The matrix B is very trivial and is

$$B = [1 - e_0, \omega_0 e_0, 0, ..., 0]^\top \tag{4-8}$$

and has a size of $(2n - 1 \times 1)$.

As an example, the matrices A, B and C are expanded below for $n = 5$ to show what the matrices should look like if done properly. To avoid confusion with u in equation (4-5), the outputs of equations (4-1) and (4-2) use the expanded notation of the output terms that was mentioned at the end of the previous section. The raw derivatives terms ($\frac{d}{dh} y_k^{(i)}$) increase the matrix size by $2n-1$, where n is the amount of derivatives to be estimated including the 0th. This means that for the case of $n = 5$ the discrete time state space system estimates the 0th to 4th derivative and is

$$\underbrace{\begin{bmatrix} y^{(0)} \\ \frac{d}{dh} y^{(0)} \\ \hat{y}^{(1)} \\ \frac{d}{dh} \hat{y}^{(1)} \\ \hat{y}^{(2)} \\ \frac{d}{dh} \hat{y}^{(2)} \\ \hat{y}^{(3)} \\ \frac{d}{dh} \hat{y}^{(3)} \\ \hat{y}^{(4)} \end{bmatrix}_{k+1}}_{\mathbf{x}_{k+1}} = \underbrace{\begin{bmatrix} e_0 & 0 & h & 0 & \frac{1}{2}h^2 & 0 & \frac{1}{6}h^3 & 0 & \frac{1}{24}h^4 \\ -\omega_0 e_0 & 0 & 1 & 0 & h & 0 & \frac{1}{2}h^2 & 0 & \frac{1}{6}h^3 \\ 0 & 1-e_1 & e_1 & 0 & h & 0 & \frac{1}{2}h^2 & 0 & \frac{1}{6}h^3 \\ 0 & \omega_1 e_1 & -\omega_1 e_1 & 0 & 1 & 0 & h & 0 & \frac{1}{2}h^2 \\ 0 & 0 & 0 & 1-e_2 & e_2 & 0 & h & 0 & \frac{1}{2}h^2 \\ 0 & 0 & 0 & \omega_2 e_2 & -\omega_2 e_2 & 0 & 1 & 0 & h \\ 0 & 0 & 0 & 0 & 0 & 1-e_3 & e_3 & 0 & h \\ 0 & 0 & 0 & 0 & 0 & \omega_3 e_3 & -\omega_3 e_3 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1-e_4 & e_4 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} y^{(0)} \\ \frac{d}{dh} y^{(0)} \\ \hat{y}^{(1)} \\ \frac{d}{dh} \hat{y}^{(1)} \\ \hat{y}^{(2)} \\ \frac{d}{dh} \hat{y}^{(2)} \\ \hat{y}^{(3)} \\ \frac{d}{dh} \hat{y}^{(3)} \\ \hat{y}^{(4)} \end{bmatrix}_{k}}_{\mathbf{x}_k} + \underbrace{\begin{bmatrix} 1-e_0 \\ \omega_0 e_0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}}_{B} u$$

with $e_i = e^{-\omega_i h}$ for $i = 0, 1, ..., n-1$. The output $\mathbf{y}_k$ is

$$\underbrace{\begin{bmatrix} y^{(0)} \\ \hat{y}^{(1)} \\ \hat{y}^{(2)} \\ \hat{y}^{(3)} \\ \hat{y}^{(4)} \end{bmatrix}_{k}}_{\mathbf{y}_k} = \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}}_{C} \mathbf{x}_k$$

The matrix that is most important is the A matrix. The stability of the system is determined by calculating the eigenvalues of A. Since this system is defined in discrete time, the system is stable when the absolute value of the eigenvalues fall within the unit circle, i.e. are less than one. However, due to the size of the A matrix, analytical methods are not feasible for determining the stability for $n > 2$. Therefore numerical methods must be used to find the eigenvalues. The software MATLAB for example can do this easily by using the `eig(A)` function. Combining this with the tuning based on the parameters n, ω_0 and f_{red} (see equation (4-4), it is not hard to find a stable A matrix rapidly.

The only disadvantage of this state space representation is that matrices are sparse due to the large state vector $\mathbf{x}_k$ that also contains the raw derivative terms next to the estimated derivatives of u. This sparsity wastes computational resources and increases computational complexity, which is $O((2n-1)^2)$.

4-1-2 RLPAD algorithm

Equation (4-1) and (4-2) can be captured in a single algorithm, see Algorithm 1. The algorithm takes as input a set of ordered $\omega \in \Omega$ ($\Omega = [\omega_0, \omega_1, ..., \omega_{n-1}]$), a time step size h, an input u, the raw derivative states M and the state of the current dynamic system X. The state X contains the state estimates of the current time step so that $X = [y_k, \hat{\dot{y}}_k, \hat{\ddot{y}}_k, ...]$. These are the estimated derivatives including the zero-th derivative. The raw derivatives states are structured as $M = \left[0, \frac{d}{dh}y_k, \frac{d}{dh}\hat{\dot{y}}_k, \frac{d}{dh}\hat{\ddot{y}}_k, ...\right]$. This algorithm has to run every time step and has to be updated with the new state, the new raw derivatives and input each time. Note that this algorithm makes use of the truncated Taylor series expansion of Algorithm 2.

The complexity of this algorithm is $O(n^2)$, for the reason that computing a factorial has $O(n)$ complexity and that has to be computed n times. This results in a computational complexity of $O(n^2)$. The complexity can be improved, because the Taylor series doesn't have to be computed for all n terms. When the time step is small enough, the accuracy does not improve much after three terms, because four or higher terms will be approximately zero ($h^3 \approx 0$). In that case the total complexity would be $O(3n)$.

Algorithm 1 Algorithm for extracting higher order derivatives

Input
    a set of cutoff frequencies Ω
    time step h
    input u
    a set of raw derivative states M of X
    a set of states X which also have the estimated terms in the Taylor series
Output
    Y, a set of new states and estimates of the derivatives of the measurement. P, a set of raw derivative terms required for updating the next step
procedure RLPAD_Step(Ω, h, u, M, X)
    Y.init(X.size()) ▷ A new list to contain the updated states
    P.init(X.size()) ▷ A new list to contain the raw derivative states, with 0th index 0
    $M[0] = u$ ▷ Assign the input to the 0th index
    for ($i \leftarrow 0$ **to** X.size() -1) **do**
        $x_c \leftarrow X[0]$ ▷ the first entry is used to update the state
        $X[0] \leftarrow 0$ ▷ Force this to 0 so it has no effect in the Taylor Series
        $y \leftarrow (x_c - M[i])\exp(-\Omega[i]h) + M[i] + \text{TaylorSeriesStep}(X, h)$ ▷ Update
        X.pop(0); ▷ Remove the first entry of X
        $Y[i] \leftarrow y$ ▷ Assign the new state estimate to Y
        $\dot{y} \leftarrow -\Omega[i](x_c - M[i])\exp(-\Omega[i]h) + \text{TaylorSeriesStep}(X, h)$ ▷ Derivative of y
        $P[i+1] \leftarrow \dot{y}$ ▷ Assign $\dot{y}$ to the list for a new "measurement" for a future step
    end for
    return Y, P ▷ The set of new states of X and the set of raw derivatives of X
end procedure

4-2 Second order differential equation for tracking

This section goes through the steps of how the proposed differentiator came to be. Let's first take a look at how a tracking system behaves in a physical sense. A very good example is a critically damped

Algorithm 2 Algorithm for calculating the result of a Taylor series

```
1: procedure TaylorSeriesStep(T, h)        ▷ An array of terms T and step size h
2:     y ← 0
3:     for (i ← 0 to T.size() − 1) do
4:         y ← y + T[i]h^i/i!              ▷ Sum each individual term of the Taylor Series
5:     end for
6:     return y                            ▷ The output for the Taylor series
7: end procedure
```

mass spring damper system that follows a forced input x_r as shown in Figure 4-1. A damped system also has the nice property of being less sensitive to noise at higher frequencies than that of the system itself. This will prove quite useful in designing a method for extracting derivatives later on.

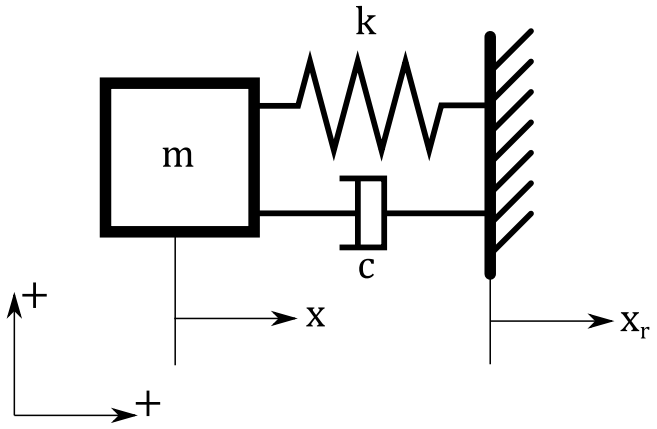


Figure 4-1: A model of a mass spring damper system with a prescribed motion x_r. In this image k is the spring constant, c the damping coefficient and m the mass. The direction of the displacement is indicated by x.

From the free body diagram the differential equation can be extracted as follows.

$$m\ddot{x} + c\dot{x} + kx = m\ddot{x}_r + c\dot{x}_r + kx_r \tag{4-9}$$

$$\ddot{x} + \frac{c}{m}\dot{x} + \frac{k}{m}x = m\ddot{x}_r + \frac{c}{m}\dot{x}_r + \frac{k}{m}x_r \tag{4-10}$$

$$\tag{4-11}$$

Let $m = 1$, then the second order differential equation becomes:

$$\ddot{x} + c\dot{x} + kx = \ddot{x}_r + c\dot{x}_r + kx_r \tag{4-12}$$

Assume that x_r, $\dot{x}_r$ and $\ddot{x}_r$ are known and that they can be represented as:

$$\begin{aligned} x_r &= \frac{1}{2}\ddot{x}_{r0}t^2 + \dot{x}_{r0}t + x_{r0} \\ \dot{x}_r &= \ddot{x}_{r0}t + \dot{x}_{r0} \\ \ddot{x}_r &= \ddot{x}_{r0} \end{aligned} \tag{4-13}$$

which is in essence a truncated Taylor series. This is an very important concept throughout this chapter.

The system is critically damped when $c = 2\sqrt{k}$. This comes from the famous differential equation:

$$\ddot{x} + 2\zeta\omega_n\dot{x} + \omega_n^2 x = 0 \tag{4-14}$$

when $\zeta = 1$ and $k = \omega_n^2$. This well known differential equation has the solution in the form of:

$$x(t) = (C_1 + C_2 t)e^{\lambda t} \tag{4-15}$$

with $\lambda = -\omega_n$ as a double root. So for equation (4-12) the double root is $\lambda_{1,2} = -\sqrt{k}$ and plugging this into (4-15) gives the homogeneous solution x_h.

The general solution of this second order system is:

$$x = x_h + x_p \tag{4-16}$$

To find the particular solution x_p, an ansatz is needed. Let's say that

$$x_p = At^2 + Bt + C. \tag{4-17}$$

The term x and its derivatives are substituted for x_p in equation (4-12). The full expressions of x_r of equation 4-13 also gets substituted in that same equation. Then equation (4-12) becomes:

$$2A + c(2At + B) + k(At^2 + Bt + C) = \ddot{x}_{r0} + c(\ddot{x}_{r0} + \dot{x}_{r0}) + k(\frac{1}{2}\ddot{x}_{r0}t^2 + \dot{x}_{r0}t + x_{r0}) \tag{4-18}$$

Collecting the terms give:

$$kAt^2 + (2CA + kB)t + 2A + cB + kC = \frac{1}{2}k\ddot{x}_{r0}t^2 + (c\ddot{x}_{r0} + k\dot{x}_{r0})t + \ddot{x}_{r0} + c\dot{x}_{r0} + kx_{r0} \tag{4-19}$$

From this equation A, B and C can be solved:

$$\begin{aligned} A &= \frac{1}{2}\ddot{x}_{r0} \\ \\ 2cA + kB &= c\ddot{x}_{r0} + k\dot{x}_{r0} \\ B &= \dot{x}_{r0} \\ \\ 2A + cB + kC &= \ddot{x}_{r0} + c\dot{x}_{r0} + kx_{r0} \\ C &= x_{r0} \end{aligned}$$

It can be seen that the coefficients A, B and C have the same values as the coefficients of the truncated Taylor series of x_r. Now that the coefficients of the particular solution are known, the initial values can be used to solve the constants C_1 and C_2 to get the general solution of equation (4-12). Plugging it into equation 4-16 and taking the derivative:

$$x(t) = (C_1 + C_2 t)e^{\lambda t} + \frac{1}{2}\ddot{x}_{r0}t^2 + \dot{x}_{r0}t + x_{r0} \tag{4-20}$$

$$\dot{x}(t) = \lambda(C_1 + C_2 t)e^{\lambda t} + \ddot{x}_{r0}t + \dot{x}_{r0} \tag{4-21}$$

The initial conditions at $t = 0$ are $x(0) = x_i$ and $\dot{x}(0) = \dot{x}_i$ then by evaluating the equations above at $t = 0$:

$$\begin{aligned} x(0) &= C_1 + x_{r0} = x_i \\ C_1 &= x_i - x_{r0} \end{aligned} \tag{4-22}$$

$$\begin{aligned} \dot{x}(0) &= \lambda C_1 + C_2 + \dot{x}_{r0} = \dot{x}_i \\ C_2 &= \dot{x}_i - \dot{x}_{r0} - \lambda(x_i - x_{r0}) \end{aligned} \tag{4-23}$$

This gives the general solution

$$x(t) = (x_i - x_{r0})e^{\lambda t} + (\dot{x}_i - \dot{x}_{r0} - \lambda(x_i - x_{r0}))te^{\lambda t} + \frac{1}{2}\ddot{x}_{r0}t^2 + \dot{x}_{r0}t + x_{r0} \tag{4-24}$$

This solution tracks the target x_r perfectly with no error and no lag, because the exponential terms go to zero when $t \to \infty$ and the remainder is the truncated Taylor series which happens to be equal to x_r in equation 4-13. Note that this can only track x_r perfectly if x_r and its time derivatives are known and have no noise on the signal. If the truncated Taylor series is omitted, the system will lag the target when it accelerates. This is due to the nature of the second order system, a third order dynamical system can track constant acceleration for example.

The continuous time domain solution is not useful for target tracking, because the target to be tracked can change its states, such as position and velocity, over time. This requires to update the initial conditions for every sample, therefore it has to be discretized. The solution can be easily discretized in time with time step $h = T_s$:

$$\begin{aligned} x_{k+1} &= (C_{1,k} + C_{2,k}h)e^{\lambda h} + \frac{1}{2}\ddot{x}_{r,k}h^2 + \dot{x}_{r,k}h + x_{r,k} \\ \dot{x}_{k+1} &= \lambda(C_{1,k} + C_{2,k}h)e^{\lambda h} + C_{2,k}e^{\lambda h} + \ddot{x}_{r,k}h + \dot{x}_{r,k} \\ &\text{with} \\ C_{1,k} &= x_k - x_{r,k} \\ C_{2,k} &= \dot{x}_k - \dot{x}_{r,k} - \lambda(x_k - x_{r,k}) \end{aligned} \tag{4-25}$$

In this set of equations k represents the current sample, x_r the measurement of the target as the input and x_k the current state of the system tracking the target. λ is the double root of the characteristics equation of the system and can be seen as a cutoff frequency.

This set of equations has two advantages for the problem of finding higher order derivatives. The first is that it is possible to differentiate equation (4-25) multiple times with respect to h as it is represented by an exponential. Therefore, higher order derivatives can be found. The second advantage is that a damped system attenuates noise, which will be required when the input is real noisy sensor data.

On the other hand the system of equations (4-25) has two problems. The first problem is that the equations require knowledge about $\dot{x}_{r,k}$ and $\ddot{x}_{r,k}$. However, these derivatives of the measurement are unknown. In fact, finding these derivatives is the main goal of the thesis. The second problem is that $C_{2,k}$ needs the derivative $\dot{x}_k$ of the system itself as well as the measurement derivative $\dot{x}_{r,k}$. This means that it requires two unknown derivatives where the latter depends on the first. This makes the estimations inaccurate, because the estimated "measurement" derivative depends on another estimation. In order to eliminate this problem, the system can be reduced by one order as will be explained in the next section.

There are two important concepts to be learned from this differential equation. The first is that it allows for extracting of higher order derivatives of an input signal due to the exponential term. The second is that if the measurement is noise free and the derivatives are known, a dynamic system can be used to track the signal perfectly. However, the derivatives of inputs/measurements are considered unknown in this thesis and are usually not readily available in real systems.

4-3 First order ODE, a simpler model

As mentioned before, the second order ODE has as problem that it requires known derivatives of input x_r and derivatives of it self for $\dot{x}_{k+1}$. In order to avoid that, the system is reduced by one order. A first order differential equation has the form of:

$$\tau_0\dot{y} + y = r \tag{4-26}$$

In this equation τ is the time constant, r is the measurement that is tracked and y is the state of the tracker. A physical interpretation of this system is a differential equations for the velocity of a falling mass under constant acceleration with linear air resistance ($\dot{v} + kv = g$).

The solution to this differential equations is as follows. First solve the homogeneous solution:

$$\begin{aligned} \tau_0 \dot{y} + y &= 0 \\ \dot{y} + \omega_0 y &= 0 \\ \text{with: } \omega_0 &= \frac{1}{\tau_0} \end{aligned}$$

The root of the characteristic equation is

$$\lambda = -\omega_0 \tag{4-27}$$

The homogeneous solution has the form of

$$y_h(t) = C_1 e^{-\omega_0 t} \tag{4-28}$$

For the particular solution an ansatz is

$$y_p = A$$

Plugging y_p into the differential equation for y and solve for A:

$$\begin{aligned} \tau_0 \dot{y}_p + y_p &= r \\ A &= r \\ y_p &= r \end{aligned}$$

The general solution is:

$$\begin{aligned} y &= y_h + y_p \\ y &= C_1 e^{-\omega_0 t} + r \end{aligned}$$

The general solution constant term C_1 can now be solved using the initial conditions at $t = 0$ so that $y(0) = y_0$, where y_0 is the initial condition.

$$\begin{aligned} y(t) &= C_1 e^{-\omega_0 t} + r \\ y(0) &= C_1 + r = y_0 \\ C_1 &= y_0 - r \end{aligned}$$

The solution to the differential equation is:

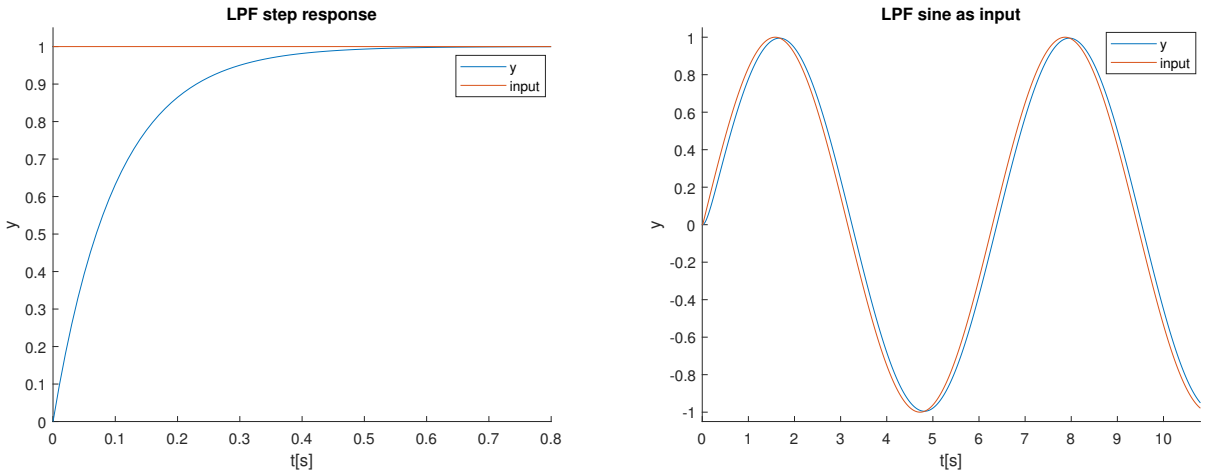
$$y(t) = (y_0 - r)e^{-\omega_0 t} + r \tag{4-29}$$

For the same reasons mentioned in the previous section, it has to be in discrete time with $h = T_s$:

$$y_{k+1} = (y_k - r_k)e^{-\omega_0 h} + r_k \tag{4-30}$$

The input term r is the measurement and the cutoff frequency is ω_0. Just as the second order solution to the ODE in equation (4-16) it has low pass characteristics. In fact, the solution above is a first order low pass filter and can be represented by a resistor-capacitor circuit (RC filter). This means that if it is differentiated with respect to time, it is exactly equal to the classical differentiator (CD) of equation (3-1).

The solution of equation (4-30) has two advantages. The first advantage is that it is possible to take derivatives multiple times, which allows for extracting higher order derivatives for noise free signals.



(a) (b)

Figure 4-2: The step response in (a) and a sine input $r = \sin(t)$ in (b) of the solution to the first order differential equation of (4-30) (a low pass filter). The parameters used are $\omega_0 = 10$ and a time step of $h = 0.01$. It tracks the constant input perfectly in (a). It follows the trend of the input with a lag when a time varying input is used in (b)

The second advantage is that the equation does not depend on higher order terms of itself as was the case with the second order ODE in equation (4-16).

When r is constant, it can track the input perfectly (see Figure 4-2a). However, when r has a constant velocity $\dot{r}$, the tracker will lag behind the target. This can be clearly seen in Figure 4-2b, where y lags the input by some amount. To be precise, it will lag $\tau_0 \dot{y}$ in distance and note that $\dot{y} = \dot{r}$ when $t \to \infty$. Although one could make τ_0 very small to reduce lag, but this also has the side effect that the cutoff frequency $\omega_0 = \frac{1}{\tau_0}$ is very high and thus will be more sensitive to noise. The reason for this is that it is exactly equal to the classical differentiator in (3-1. So this solution comes with all the problems the classical differentiator has, i.e. noise amplification for high gains and lag for low gains. This is not desired for tracking, the tracker must track the input as accurate as possible. The next section covers a method to reduce lag significantly.

4-4 Compensator

The second order ODE in section 4-2 shows that with prior knowledge of the truncated Taylor series of the input helps to track the target with zero error. However, in this thesis the derivatives are unknown and therefore a truncated Taylor series cannot be constructed directly. Despite that, it is possible to estimate the coefficients of the Taylor series over time instead, which in turn are the estimates of the derivatives of the input. This section shows how to do that by extending the first order ODE of the previous section with a compensator that helps to reduce lag and allows for extracting higher order derivative.

4-4-1 Input as Taylor series

A continuous signal can be represented by an infinite Taylor series around a point t_0. This means that signal $r(t)$ can be represented as an infinite polynomial:

$$r(t) = r(t_0) + \frac{\dot{r}(t_0)}{1!}(t - t_0) + \frac{\ddot{r}(t_0)}{2!}(t - t_0)^2 + \frac{\dddot{r}(t_0)}{3!}(t - t_0)^3 + ... \tag{4-31}$$

When $r(t)$ is substituted in equation (4-26) and set $t_0 = 0$ for convenience it becomes:

$$\tau_0 \dot{y} + y = r(0) + \dot{r}(0)t + \frac{\ddot{r}(0)}{2}t^2 + \frac{\dddot{r}(0)}{6}t^3 + ... \tag{4-32}$$

This first order differential equation has the solution in the form of:

$$y(t) = C_1 e^{-\omega_0 t} + A + Bt + Ct^2 + Dt^3 + ... \tag{4-33}$$

One might expect that the coefficients of A, B, C and D are the same as the ones of the differential equation of (4-19). Although it would be very convenient to have, this is not case. When the differential equation of (4-33) is solved for a second degree polynomial input, the coefficients are:

$$\begin{aligned} A &= r(0) - \tau_0 \dot{r}(0) + \tau_0^2 \ddot{r}(0) \\ B &= \dot{r}(0) - \tau_0 \ddot{r}(0) \\ C &= \frac{1}{2}\ddot{r}(0) \\ C_1 &= y_0 - r(0) + \tau_0 \dot{r}(0) - \tau_0^2 \ddot{r}(0) \end{aligned}$$

This shows that the coefficients depend on multiple derivative terms of r which did not solve the problem of having derivatives of r in the coefficients. The derivative terms of the input r are required, which are unknown. The goal is to find these derivatives of signals and eventually also for noisy signals.

A solution to this is to set the coefficients $(A, B, C, ...)$ with the same values of the truncated Taylor series and keep the first order low pass filter as is. The solution of equation (4-29) is augmented so that it can follow the Taylor series of (4-31). Then the solution becomes:

$$y(t) = (y(t_0) - r(t_0))e^{-\omega_0(t-t_0)} + r(t_0) + \frac{\dot{r}(t_0)}{1!}(t-t_0) + \frac{\ddot{r}(t_0)}{2!}(t-t_0)^2 + ... \tag{4-34}$$

The main idea of this is that when $t \to \infty$ the exponential term is equal to zero, and thus the Taylor series remain. However, a truncated Taylor series can only estimate the input accurate in a small range around an evaluating point. So in order to improve the estimation, this equation has to be evaluated at several evaluating points of the input. When this is done for a set of evaluating points $\mathbf{t_e}$ the solution of $y(t)$ will converge to the Taylor series around that given t_i, where i denotes the i-th evaluation point. Thus it will match the input signal more accurately.

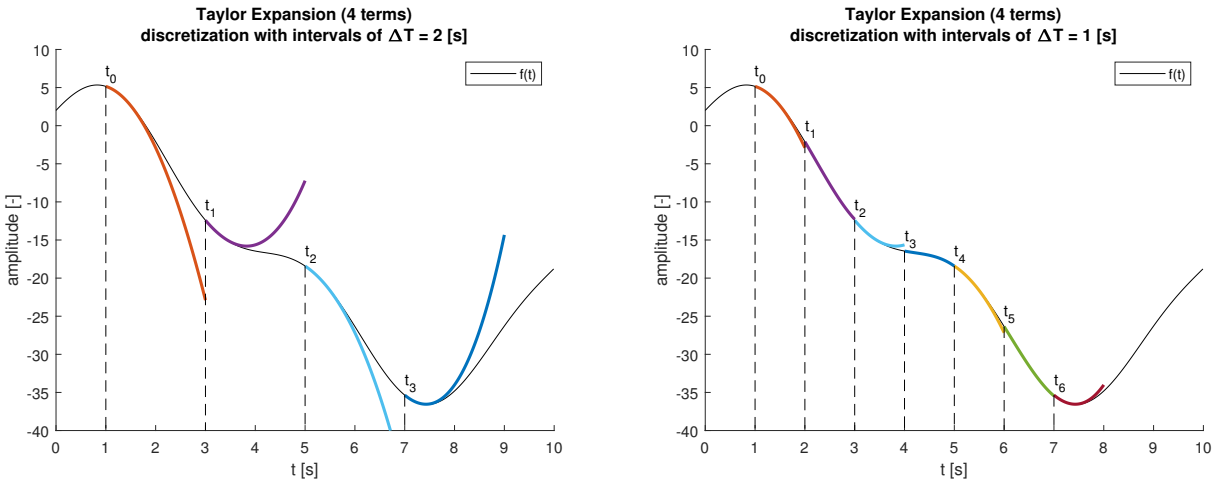
Evaluating equation (4-34) for a set of evaluating points $\mathbf{t_e}$ turn this continuous equation into a discrete problem. The reason for that is that the equation its initial conditions has to be set at every newly evaluated t_i, and those initial conditions depend on the previous evaluating point at t_{i-1} just before $t = t_i$. This causes discontinuities in equation (4-34). This is shown in Figure 4-3, where the evaluating points of the set $\mathbf{t_e}$ are shown. By adding more evaluating points, the equation stays more close to the reference input. This means that when making the steps between evaluating points so small, so that a small increment h of time t ends up at a new evaluating point, a recurrence equation will appear. This transforms equation (4-34) to its discrete time variant and is:

$$y_{k+1} = (y_k - r_k)e^{-\omega_0 h} + r_k + \dot{r}_k h + \frac{1}{2}\ddot{r}_k h^2 + \frac{1}{6}\ddot{r}_k h^3 + ..., \tag{4-35}$$

where k denotes the k-th sample at time $t = kh$ and h is the time step.

Note that equation (4-34) and (4-35) are **not** the solution to the differential equation. This augmentation must be seen as a compensator to reduce lag. This formulation is very similar to the dynamic system of TSEBD in equation (3-7), but with a low pass filter at the start and is defined in discrete time.

The problem of the equation above is that, as mentioned before, the derivatives of the input signal are unknown. A way to solve this issue is to estimate the derivatives of r_k by having the derivatives of r_k be a function of y_k and r_k. The next section explains how these input derivatives can be estimated.



(a) (b)

Figure 4-3: These plots show the truncated Taylor series expansion of 4 terms of the function $f(t)$ at several evaluation points. It shows that when more evaluation points are added, the Taylor series of 4 terms can estimate the function more accurately. The continuous change of evaluating points creates discontinuities in equation (4-34) and is, therefore, turned into a discrete time equation (4-35).

4-4-2 Estimating the Taylor series coefficients

There are three properties that can be exploited from equation (4-30). The first is that it has a steep gradient or velocity at the start of a step response, see Figure 4-2a, which can be used to estimate derivatives. The second is that it can follow the trend of a time varying signal with some lag (Figure 4-2b) and the third is that it can attenuate noise, since it is a low pass filter at its core. The key idea is to combine these properties in a feedback loop to compensate for the lag, attenuate noise and give a better tracking accuracy than without a compensator. Rewriting equation (4-35) to work with estimates of derivate terms of r_k denoted by $\hat{y}$ gives:

$$y_{k+1} = (y_k - r_k)e^{-\omega_0 h} + r_k + \hat{\dot{y}}_k h + \frac{1}{2}\hat{\ddot{y}}_k h^2 + \frac{1}{6}\hat{\dddot{y}}_k h^3 + ... \tag{4-36}$$

For simplicity, the higher order terms are omitted for a first analysis. So the equation reduces to:

$$y_{k+1} = (y_k - r_k)e^{-\omega_0 h} + r_k + \hat{\dot{y}}_k h \tag{4-37}$$

The next question is, what would $\hat{\dot{y}}$ be? One might try to estimate the velocity as the direct derivative of the equation above so that

$$\hat{\dot{y}}_{k+1} = \frac{d}{dh} y_{k+1} = -\omega_0 (y_k - r_k)e^{-\omega_0 h} + \hat{\dot{y}}_k \tag{4-38}$$

Note that all the y and $\hat{\dot{y}}$ terms are assumed constant during differentiation. This has to be done, because these are also constant in the continuous time variant of the Taylor series. This way it preserves the interchangeability between a continuous and a discrete time Taylor series. Or to put in a signal processing way, the terms y and $\hat{\dot{y}}$ are constant during a step k, because the samples are modeled as zero order hold.

At first glance derivative feedback could work, but when transformed to the Z-domain it will turn out that this method induces oscillations for a wide range of combinations for $\omega_0 h < 1$, which can be seen in Figure 4-4. This is bad as the oscillations will be induced for low values of ω_0, essentially removing low pass characteristics. The following analysis makes this this clear.

The system that has feedback with a velocity term of itself is

$$\begin{aligned} y_{k+1} &= (y_k + r_k)e_0 + r_k + \hat{\dot{y}}_k h, \quad e_0 = e^{-\omega_0 h} \\ \hat{\dot{y}}_{k+1} &= -\omega_0 (y_k + r_k)e_0 + \hat{\dot{y}}_k \end{aligned} \tag{4-39}$$

The Z-domain transformations are as follows.

$$\begin{aligned}\mathcal{Z}\{y_k\} &= Y(z)\\ \mathcal{Z}\{r_k\} &= R(z)\\ \mathcal{Z}\{\hat{y}_k\} &= \hat{Y}(z)\end{aligned}$$

Then the system of equations of (4-39) can then be transformed to the Z-domain as:

$$\begin{aligned}zY(z) &= e_0Y(z) + (1-e_0)R(z) + h\hat{\hat{Y}}(z)\\ z\hat{\hat{Y}}(z) &= -\omega_0 e_0 Y(z) + \omega_0 e_0 R(z) + \hat{\hat{Y}}(z)\end{aligned}$$

This can be written as a discrete transfer function by substituting $\hat{\hat{Y}}(z)$ and rearranging the terms so that the transfer function becomes

$$\frac{Y(z)}{R(z)} = \frac{(1-e_0)z + \mathrm{e}_0 - 1 + \omega_0 e_0 h}{z^2 - (e_0+1)z + e_0 + \omega_0 e_0 h}, \quad \text{with } e_0 = e^{-\omega_0 h} \tag{4-40}$$

It is interesting to see that the system turned into a second order system by adding a velocity term that depends on itself to the base function. This is not that strange because the velocity feedback added additional dynamics to the system. This transfer function is stable, as it has its roots of the denominator always within the unit circle. However, it will be an oscillating system with low damping for a wide range of values of ω_0 and h. This can be shown by the analysis of poles. The characteristic equation is $z^2 - (e_0+1)z + e_0 + \omega_0 e_0 h$, rewriting this to

$$\begin{aligned}\mathcal{C}_{eq} &= z^2 + \beta z + \gamma\\ \beta &= -(e_0+1)\\ \gamma &= e_0 + \omega_0 e_0 h\end{aligned} \tag{4-41}$$

allows us to find the roots easily. The range of values of β and γ are limited by the nature of the exponent for negative values. The ranges are found as follows. Set $e_0 = e^{-\kappa}$ with $\kappa = \omega_0 h$ and $\kappa \geq 0$. The limits of β can be directly found from the fact that $0 \leq e_0 \leq 1$ and therefore the limits are

$$\begin{aligned}\lim_{\kappa\to 0}\beta &= -2\\ \lim_{\kappa\to\infty}\beta &= -1\\ \{\beta \in \mathbb{R} | -2 \leq \beta \leq -1\}\end{aligned} \tag{4-42}$$

The interval of γ takes a bit more effort to find, the limits of γ are as follows. By writing γ as

$$\gamma = e^{-\kappa} + \kappa e^{-\kappa} \tag{4-43}$$

the limit of $\kappa \to 0$ is found easily.

$$\lim_{\kappa\to 0}\gamma = 1 \tag{4-44}$$

The limit of $\kappa \to \infty$ is found by using L'Hospital's rule:

$$\lim_{\kappa\to\infty}\gamma = \lim_{\kappa\to\infty} e^{-\kappa} + \lim_{\kappa\to\infty}\frac{\kappa}{e^{\kappa}} \tag{4-45}$$

$$\lim_{\kappa\to\infty}\gamma = 0 + \lim_{\kappa\to\infty}\frac{\frac{d}{d\kappa}\kappa}{\frac{d}{d\kappa}e^{\kappa}} = \lim_{\kappa\to\infty}\frac{1}{e^{\kappa}} = 0 \tag{4-46}$$

The limits do not give a definitive answer of the interval of γ, as it does not tell if $\gamma \geq 1$ for a value of $\kappa \geq 0$. This can be determined by taking the derivative of γ with respect to κ which is:

$$\frac{d}{d\kappa}\gamma = -\kappa e^{-\kappa} \tag{4-47}$$

This shows that the derivative of γ is strictly negative, which means that $\gamma < 1$ for $\kappa > 0$. Thus the interval of γ is $\{\gamma \in \mathbb{R} | 0 \leq \gamma \leq 1\}$.

The poles of $\mathcal{C}_{eq} = z^2 + \beta z + \gamma$ are found by the quadratic formula:

$$z_{1,2} = -\frac{\beta}{2} \pm \frac{1}{2}\sqrt{\beta^2 - 4\gamma} \tag{4-48}$$

The system will have oscillation when $\sqrt{\beta^2 - 4\gamma} < 0$. These oscillations can reach almost pure oscillatory behavior depending on the value κ. This is shown by the following analysis when $\sqrt{\beta^2 - 4\gamma} < 0$.

The absolute value is defined as $||z|| = \sqrt{z\bar{z}}$, where $\bar{z}$ is the complex conjugate. The absolute value of the poles is found by taking the absolute value of the quadratic formula. When the poles are complex the absolute value is

$$||z_{1,2}|| = \sqrt{\gamma} \tag{4-49}$$

This also shows that $||z_{1,2}|| <= 1$ and thus the system of equation (4-40) is always stable or marginally stable ($||z_{1,2}|| = 1$) for $\sqrt{\beta^2 - 4\gamma} < 0$. These poles in the Z-domain, are not as intuitive to interpret as poles in the S-domain, therefore insight in the oscillations are easier in the S-domain for this case. The exact relation of $s \to z$ is defined as $z = e^{sh}$, where h is the time step. So rearranging this expression in terms of s is

$$s = \frac{1}{h} \ln z \tag{4-50}$$

The value of z can be complex so the logarithm can be expanded and the expression becomes

$$s = \frac{\ln(||z||) + i \arg(z)}{h} \tag{4-51}$$

In the case of $\sqrt{\beta^2 - 4\gamma} < 0$, the pole pair $z_{1,2}$ in the S-domain are

$$s_{1,2} = \frac{1}{h}\left(\ln(\sqrt{\gamma}) \pm i \arctan\left(\frac{\sqrt{||\beta^2 - 4\gamma||}}{-\beta}\right)\right) \tag{4-52}$$

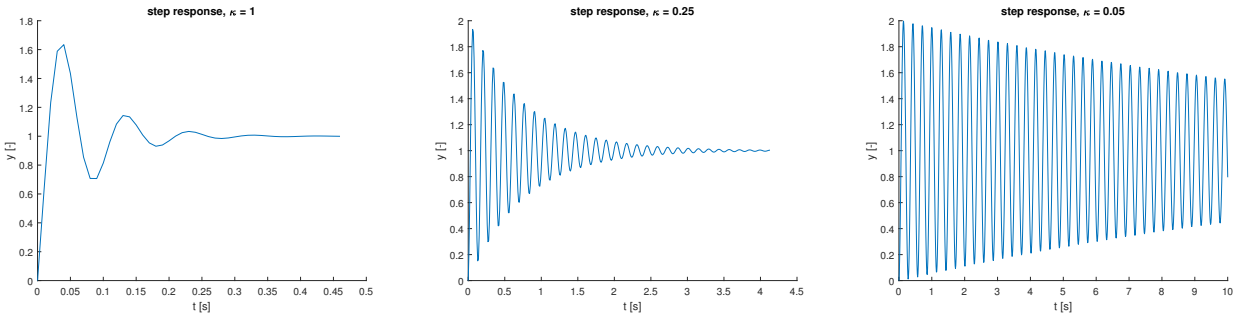
When $\gamma \to 1$ the real part of $s_{1,2} \to 0$, while the imaginary part does not go to zero as fast. This results in complex conjugate pole pairs where the imaginary part is much greater than the real part and thus the system has slow decaying oscillations.

In the extreme case it will have a pole pair that is close to $s_{1,2} = 0 \pm \alpha i$. The characteristic equation of this is then $(s + \alpha i)(s - \alpha i) = s^2 + \alpha^2$. This has the same characteristics equation of a sine with $\sin(\alpha t)$. From that it can be concluded that the response of the system in equation (4-40) can be close to a pure oscillator. This is also confirmed by Figure 4-4bc. It shows that for a step response the system oscillates with very low damping (b) and these effects are increased for lower values of κ (c).

The most important aspect of Figure 4-4 is that it shows that the transfer functions of equation (4-40) has really bad low pass properties. When κ is low valued and the time step is constant ($h = 0.01$), $\omega_0 = \kappa h^{-1}$ will be relatively low valued as well. A low pass filter with a low ω would damp the response, but with this system a low ω_0 induces oscillations and they get worse the lower ω_0 gets. This is the reason why the filter of equation (4-39) would not work for a differentiator.

In order to reduce the effects of oscillations, a tunable damping term is added. The derivative of equation (4-39) can be fed through a low pass filter similar to equation (4-36) as that has a dampening effect. This process is as follows for a first degree polynomial:

$$\begin{aligned} y_{k+1} &= (y_k - r_k)e^{-\omega_0 h} + r_k + \hat{\dot{y}}_k h \\ \frac{d}{dh}y_{k+1} &= -\omega_0(y_k - r_k)e^{-\omega_0 h} + \hat{\dot{y}}_k \\ \hat{\dot{y}}_{k+1} &= (\hat{\dot{y}}_k - \frac{d}{dh}y_k)e^{-\omega_1 h} + \frac{d}{dh}y_k \end{aligned} \tag{4-53}$$



(a) (b) (c)

Figure 4-4: Step response of the discrete transfer function of equation (4-40) for varying $\kappa = \omega_0 h$, with constant time step $h = 0.01$. The left image (a) shows the characteristic response of a second order system with $\kappa = 1$. It has high over shoot, rapid decaying oscillations and a fast settling time. Lowering κ to 0.25 (b) shows that the oscillations are decaying slower and that the settling time is longer with respect to (a). The frequency of the oscillations are slightly lower than of (a). Lowering the value of κ to 0.05 (c) show even slower decaying oscillation and a settling time that is significantly longer than (a). Also note that the frequency of the oscillations in (c) is reduced with respect to (b). Having $\kappa > 1$ results in a more damped and faster transient. On the other hand, when $0 < \kappa < 1$, the system is slower and behaves more as pure oscillator. This is unwanted behavior for an differentiator.

The last expression can be interpreted as a new "measurement" $\frac{d}{dh}y_k$ is created to update the estimated velocity $\hat{\dot{y}}_{k+1}$. This has as result that the derivative $\frac{d}{dh}y_k$ is smoothed by a first order low pass filter with ω_1 as the cutoff frequency. Note that $\frac{d}{dh}y_{k+1}$ is not used for calculating $\hat{\dot{y}}_{k+1}$, but the previous calculated sample is used. This way there is no mismatch between the samples at k. If $\frac{d}{dh}y_{k+1}$ was used instead, a lead would be introduced.

The effect of filtering the derivative of $\frac{d}{dh}y_k$ with a low pass filter has the net effect that it is damped. This can be shown by transforming the set of equations of (4-53) to the Z-domain. The transformations are

$$\begin{aligned} \mathcal{Z}\{y_k\} &= Y(z) \\ \mathcal{Z}\{r_k\} &= R(z) \\ \mathcal{Z}\{\frac{d}{dh}y_k\} &= D(z) \\ \mathcal{Z}\{\hat{\dot{y}}_k\} &= \hat{\dot{Y}}(z) \end{aligned}$$

and the transformed equations of (4-53) become

$$\begin{aligned} zY(z) &= (Y(z) - R(z))e_0 + R(z) + \hat{\dot{Y}}(z)h \\ zD(z) &= -\omega_0(Y(z) - R(z))e_0 + \hat{\dot{Y}}(z) \\ z\hat{\dot{Y}}(z) &= (\hat{\dot{Y}}(z) - D(z))e_1 + D(z) \end{aligned} \tag{4-54}$$

where $e_0 = e^{-\omega_0 h}$ and $e_1 = e^{-\omega_1 h}$. By solving these equations in terms of $Y(z)$ and $R(z)$ the following discrete time transfer functions is found:

$$\frac{Y(z)}{R(z)} = \frac{(1-e_0)z^2 - (1-e_0)e_1 z - (1-e_1)(h\omega_0 e_0 + e_0 - 1)}{z^3 - (e_0+e_1)z^2 + (e_0 e_1 + e_1 - 1)z + (1-e_1)(h\omega_0 e_0 + e_0)} \tag{4-55}$$

It can be seen that this system is one order higher than the transfer function of equation (4-40) due to the additional filter pass. The part of interest is $(1-e_1)(h\omega_0 e_0 + e_0)$ in the denominator. The term $e_1 = e^{-\omega_1 h}$ can take the values between 0 and 1 by tuning ω_i. This results in controlling how much

the last part has effect on the response. In turn this will affect the oscillations of the response, while keeping the main transient response mainly determined by e_0.

The effect of a constant e_0 and varying e_1 is shown in Figure 4-5. The time step is kept constant and the value of ω_1 is changed in $\kappa_1 = \omega_1 h$. The damping of the response increases from high to low values for κ_1, which can be clearly seen in (a-b) and over damping in (c). Notice that in these plots the opposite of Figure 4-4 happens, where it shows that for lower κ the oscillations increased. The transient response is also affected by the value of κ_1 with the shortest one in (b). However, it is a lot less affected compared to the transient response in Figure 4-4. This shows that the transient can be tuned individually along with the damping.

Concluding, by filtering the raw derivative through a second low pass filter with cutoff frequency ω_1, the problems that were encountered in equation (4-40) are solved. This way the transient response based on e_0 and damping term (e_1) can be tuned individually. Another important aspect is that the filter restored its low pass characteristics, which can be seen in the bode plot of Figure 4-5d. The magnitude decreases by 20dB/decade after the cutoff frequency $\omega_0 = 20$ rad/s.

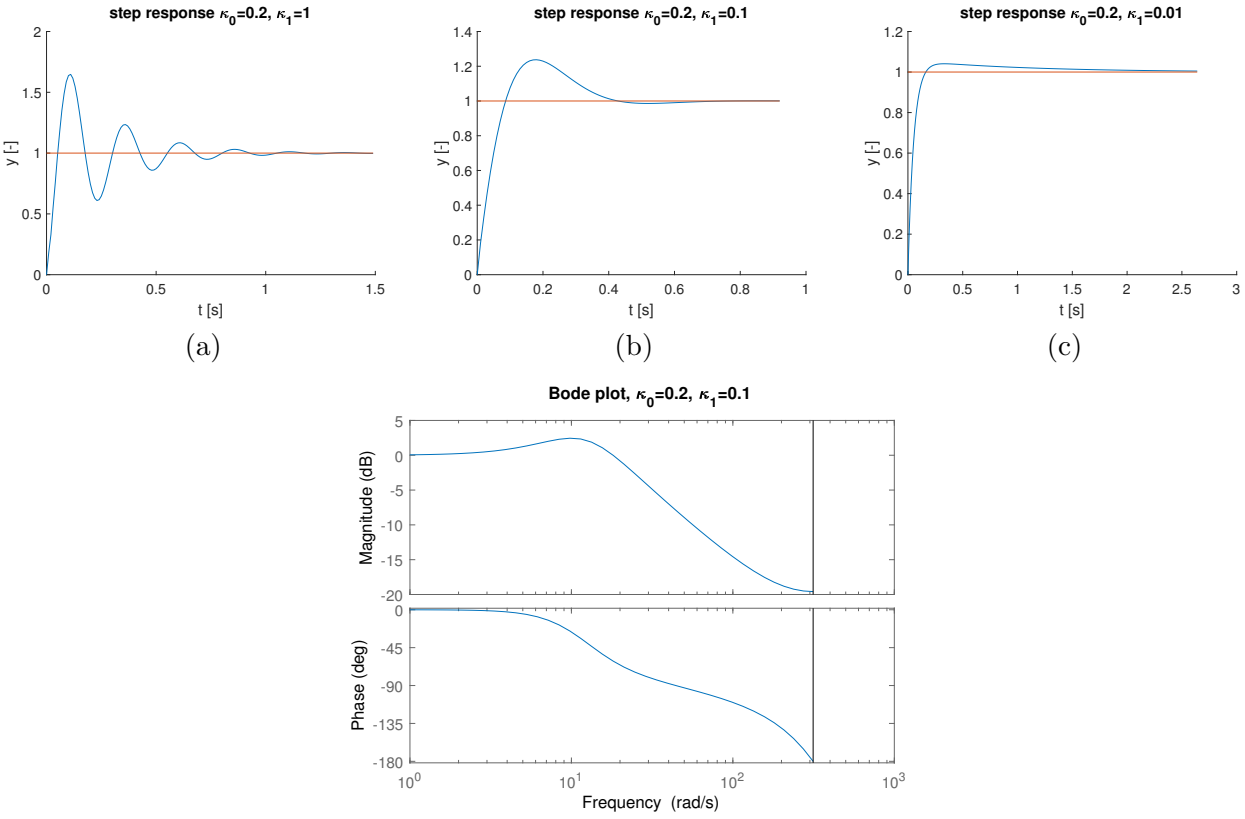


Figure 4-5: Step responses of the discrete transfer function of equation (4-55) for constant $e_0 = e^{-\kappa_0}$ and varying $e_1 = e^{-\kappa_1}$, where $\kappa_0 = \omega_0 h$ and $\kappa_1 = -\omega_1 h$ in (a-c) and the bode plot of (b) shown in (d). The transfer function has a time step of $h = 0.01$ seconds. The left image (a) shows that there are fast oscillations for $\kappa_1 = 1$ that reduce in amplitude over time. Lowering κ to 0.1 (b) shows that the oscillations are damped rapidly with some overshoot. Having $\kappa_1 = 0.01$ in (c) results in more damping than (b). However, it is over damped so much that it takes a long time to settle. All figures show a similar rise time, that is because the transfer function of equation (4-55) has a term to control the damping and rise time which is something that the transfer function of equation (4-40) could not. The low-pass characteristics of (b) are shown clearly in the Bode plot (d)

Adding a low pass filter creates the problem that the derivative estimate will lag behind the true value. In that case the derivative estimate will lag $\frac{1}{\omega_1}\frac{d}{dh}\hat{y}_k$ in distance. This is the same problem that was encountered in equation (4-30). So a solution to this is to compensate for this lag as was done for the lower order derivative by using another truncated Taylor series. This means that it is possible to generalize this method for up to any amount of terms in the Taylor series by taking the derivative

and then feed it through a low pass filter that is also compensated by a Taylor series expansion. The generalization of this process is

$$\begin{aligned} y_{k+1}^{(i)} &= \left(y_k^{(i)} - u_k^{(i)}\right) e^{-\omega_i h} + u_k^{(i)} + \sum_{j=1}^{n-i-1} \frac{h^j}{j!} y_k^{(j)} \quad \text{for } i = 0, 1, ..., n-1 \\ u_{k+1}^{(i+1)} &= -\omega_i \left(y_k^{(i)} - u_k^{(i)}\right) e^{-\omega_i h} + \sum_{j=0}^{n-i-2} \frac{h^j}{j!} y_k^{(j)} \quad \text{for } i = 0, 1, ..., n-2 \end{aligned}$$

These are the same recurrence equations of (4-1) and (4-2) of the proposed method (RLPAD) in section 4-1. The important input is $u_k^{(0)}$, which is the external input to the system. The positive tuning parameters are ω_0, f_{red}, n and h. The parameter f_{red} determines the values of $\omega_{i+1} = f_{red}^{-(i+1)}\omega_0$, with the condition that $\omega_{i+1} < \omega_i$. It is also possible to tune all ω_i individually, but that is a tedious process for a large n.

These two equations capture the whole proposed method for estimating higher order derivatives. To show that this method can do that, a proof of concept is shown in the next section. The proposed method is compared to the state of the art differentiator AEAD in chapter 5 and 6 to see how they perform with different types of inputs. Chapter 5 tests both methods for analytical inputs such as a polynomial, a sine and a combination of the two. Chapter 6 shows how they perform when the input is real sensor data.

4-5 A proof of concept

This section shows the proof of concept of the proposed method. The algorithm is tested on two types of inputs, a polynomial and a sine. The polynomial function is

$$\begin{aligned} u(t) &= x_0 + v_0 t + a_0 \frac{1}{2} t^2 + j_0 \frac{1}{6} t^3 + s_0 \frac{1}{20} t^4 + c_0 \frac{1}{120} t^5 \\ &\quad \text{with } x_0 = 0, \;\; v_0 = 2, \;\; a_0 = 1, \;\; j_0 = -0.5, \;\; s_0 = 0.25, \;\; c_0 = -0.125. \end{aligned} \tag{4-56}$$

and the sine function is

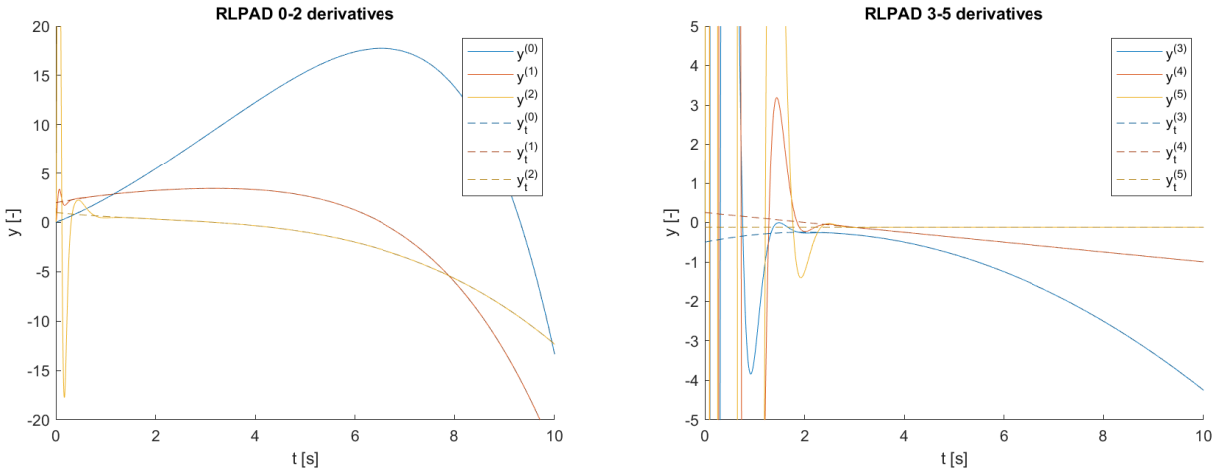
$$r(t) = \sin(t) \tag{4-57}$$

The polynomial function can be interpreted as a constant "crackle" function ("crackle" is the 5th order derivative of position). This means that is has has non zero derivatives up to the 5th order. The constant terms are chosen so that the output values are contained within a small bound between $t = [0, 10]$ for clear figures. This function will also be used later to compare the performance of AEAD versus the proposed method in Chapter 5.

It is important that these functions can be differentiated multiple times so that the root mean square error (RMSE) can be evaluated of the differentiator with respect to the true derivatives. The polynomial can be differentiated five times before it is zero. The degree of the polynomial is limited to five, because five derivatives can be extracted from it and that amount is assumed to be sufficient for testing purposes. The sine function can be differentiated an infinite amount of times, and will prove to be hard to approximate with a finite set of derivatives in contrast to the polynomial.

The goal of this proof of concept is to verify that the proposed method is capable of extracting higher order derivatives from a polynomial and a sine. Therefore the tuning of the proposed method to get the RMSE as low as possible is not considered in this section. The parameters used are $n = 6$ (estimates 5 derivatives), $\omega_0 = 100$ and $f_{red} = 2.0$. These parameters are the same for the sine and polynomial inputs.

The polynomial and its derivatives are tracked with very low errors in Figure 4-6. Table 4-1 shows that this is indeed the case. Although the errors increase over each consecutive derivative, the RMSE



(a) (b)

Figure 4-6: These plots show the derivative tracking of the proposed method for the polynomial signal of equation (4-56). The parameters used are $n = 6$, $\omega_0 = 100$ and a time step of $h = 0.001$. The i in $y^{(i)}$ denotes the order of derivative. The true derivatives are denoted by $y_t^{(i)}$. The polynomial and its estimated derivatives are tracked with very low errors in (a) and (b) with the RMSE shown in Table 4-1.

of the 5th derivative is still extremely low. The proposed method has a big advantage here, because the input polynomial has a degree of six. This results in a constant 5th order derivative, which can be tracked by the algorithm with low error because its filter size is $n = 6$. If it was a higher degree polynomial or $n = 5$, the estimated derivatives RMSE would have been higher as the proposed method would have not been able to track those without lag for the higher order derivative estimates. This advantage is shown in Figure 4-9a, where it can be seen that adding more derivative estimations do not improve the error, because the 5th derivative is constant at $n = 6$. It also shows that the RMSE of $e^{(0)}$ decreases linear in log scale by adding more derivative terms.

The sine input and its first two derivatives in Figure 4-7 are tracked with very low errors, however, higher order estimates decrease in accuracy rapidly. It shows in (b) that the 4th derivative starts to overshoot and the 5th derivative lags behind the true signal. These effects results in higher RMSE in Table 4-1 for the sine input. It shows that the RMSE is increased for each higher order derivative. The 5th order derivative lags the true one, because the amount of states in the algorithm is limited to $n = 6$. This means that there is no lag compensation for this estimate. Increasing the filter size to higher orders and retuning solves this issue. Figure 4-9b shows that by adding more derivative estimation terms the error for the sine decreases, however, at $n = 8$ the error starts to increase again. This is due to the bandwidth reduction by of ω_i by f_{red} for each higher order derivative estimated. This eventually leads to a too low ω_i that is unable to track the true higher order derivative.

A sinusoidal input has the property that it has an infinite amount of derivatives. Therefore at some point the estimated derivatives starts to lag the true ones, since the proposed method can only estimate a finite amount of derivatives. The effect of increasing the amount of states to $n = 10$ and retuning the filter to $\omega_0 = 500$ can be seen in Figure 4-8. It tracks the 5th order derivative with significantly reduced lag and the 4th order derivative has hardly any overshoot. However, this higher gain value has the downside that it makes the system very sensitives to noise.

This proof of concept concludes that the proposed method is capable of tracking higher order derivatives with low errors when its not limited by noise and large time steps. It excels at tracking and estimating derivatives of polynomial signals, which was one of the sub goals in this thesis. It can track sinusoidal inputs and estimate the derivatives of it, although they decrease in accuracy more rapidly than for polynomial inputs. These effects can be reduced with the right tuning.

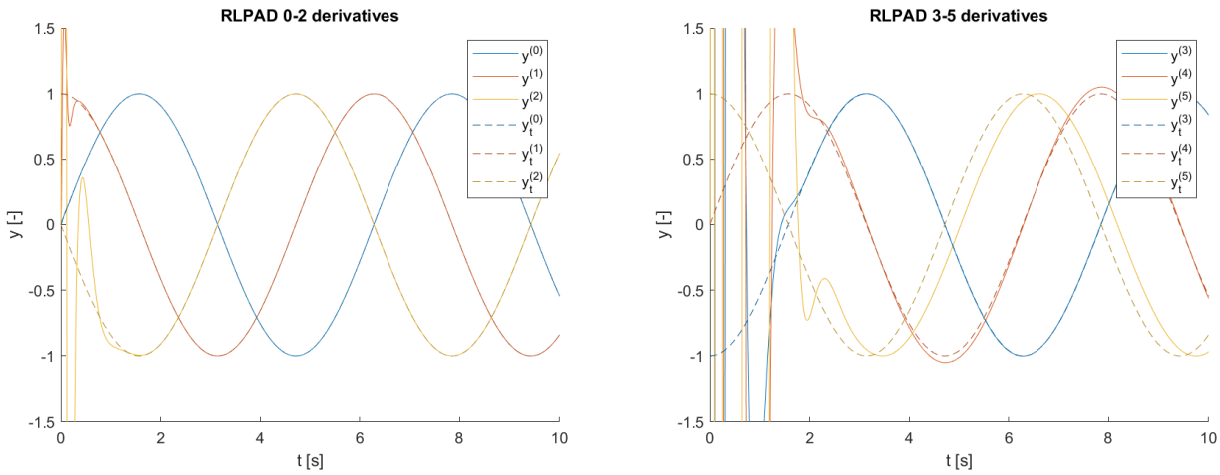


Figure 4-7: These plots show the derivative tracking of the proposed method for $\sin(t)$. The parameters used are $n = 6$, $\omega_0 = 100$, $f_{red} = 2.0$ and a time step of $h = 0.001$. The i in $y^{(i)}$ denotes the order of derivative. The true derivatives are denoted by $y_t^{(i)}$. The sine is tracked with low errors in (a). In (b) it shows some overshoot for the 4th derivative and the 5th derivative starts to lag the true derivative. The RMSE of these plots are shown in Table 4-1.

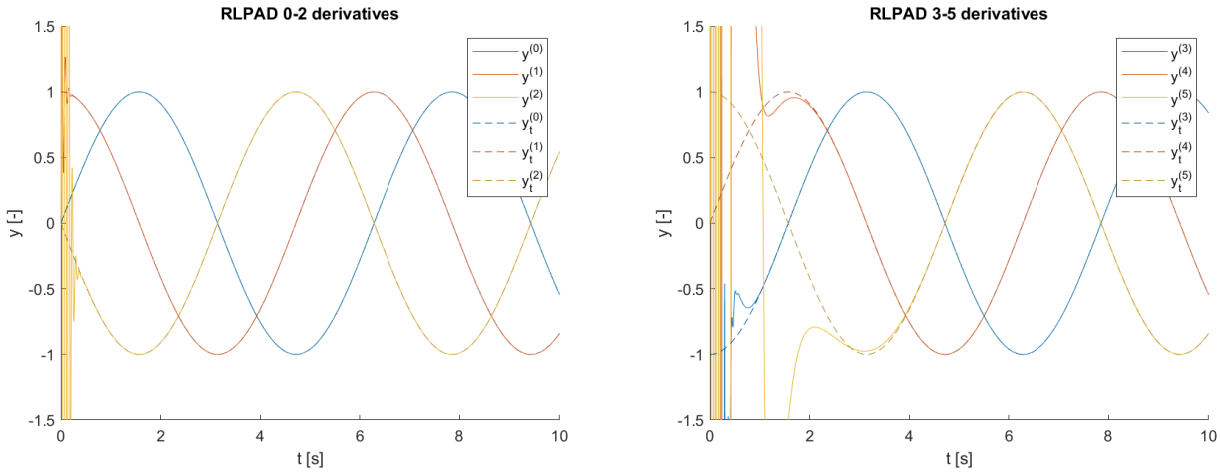
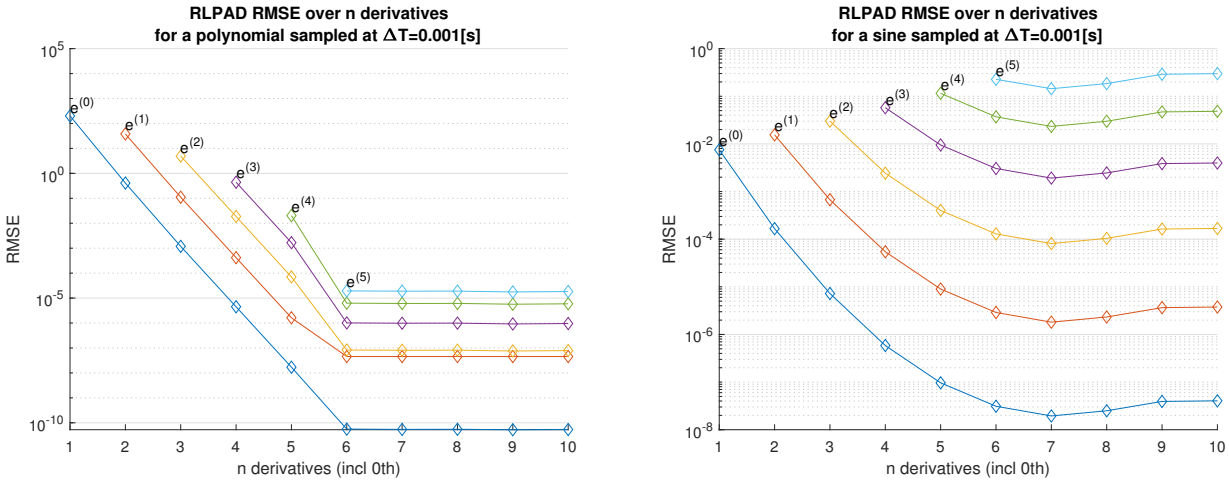


Figure 4-8: These plots show the derivative tracking of the proposed method for $\sin(t)$ with different parameters. The parameters used are $n = 10$, $\omega_0 = 500$, $f_{red} = 2.0$ and a time step of $h = 0.001$. The i in $y^{(i)}$ denotes the order of derivative. The true derivatives are denoted by $y_t^{(i)}$. This figure shows that by adding more states and having a higher gain, it can track the derivatives of a sine a lot better than the one in Figure 4-7.

	RMSE of RLPAD with time step $\Delta T = 0.001$ and $n = 6$					
input	$e^{(0)}$	$e^{(1)}$	$e^{(2)}$	$e^{(3)}$	$e^{(4)}$	$e^{(5)}$
polynomial	$1.4 \cdot 10^{-12}$	$2.9 \cdot 10^{-10}$	$4.9 \cdot 10^{-9}$	$9.7 \cdot 10^{-8}$	$9.2 \cdot 10^{-7}$	$4.6 \cdot 10^{-6}$
sine	$3.1 \cdot 10^{-8}$	$2.9 \cdot 10^{-6}$	$1.3 \cdot 10^{-4}$	$3.0 \cdot 10^{-3}$	$3.7 \cdot 10^{-2}$	$2.3 \cdot 10^{-1}$

Table 4-1: The root mean square errors of Figure 4-6 and 4-7 are tabulated here. The parameters used are $n = 6$, $\omega_0 = 100$ and $f_{red} = 2.0$. The RMSE is recorded over the interval $t = [4.0, 20]$ after the transient response has settled. The error of the i-th derivative is denoted by $e^{(i)}$. This table shows that the proposed method is capable of tracking derivatives accurately with very low errors, especially for the polynomial. The sine input also has low errors, but it increases over the amount of derivatives added.



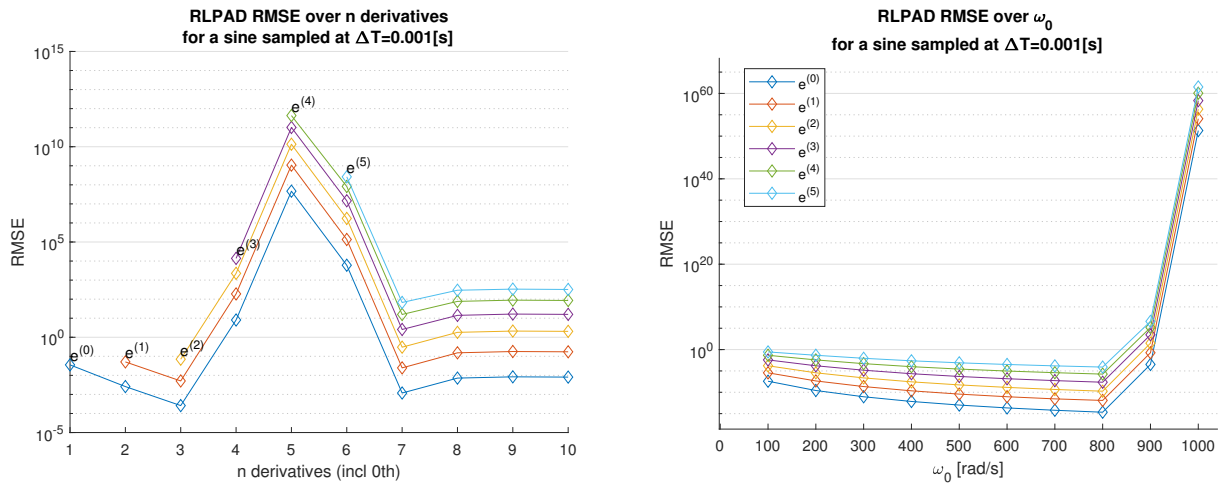
(a) (b)

Figure 4-9: These plots show the RMSE (in log scale) over an increasing amount of derivatives estimates from $n = 1$ to 10 calculated by the proposed method. The RMSE is recorded over the interval $t = [40, 50]$ to make sure that all transients have settled. The parameters used are $\omega_0 = 100$, $f_{red} = 2.0$ and a time step of $h = 0.001$. The i in $e^{(i)}$ denotes the root mean square error of the ith derivative. Plot (a) shows the error of the polynomial (4-56) over the amount of derivatives and in (b) the same, but for a sine function (4-57). It can be seen that in (a), the errors drop rapidly and linear in log scale for a polynomial. It also shows that the error does not decrease more if it exceeds the degree of polynomial. The reason that the error is very high at $n = 1$ is, because, the polynomial has a very high velocity at $t = 40$ and this results in a large lag for the position. Plot (b) shows that the RMSE of a sine input also improves by adding more states, however, it also worsen by adding more states than $n = 7$ for this particular case. The reason for this is the bandwidth reduction per derivative estimated, therefore at some point the bandwidth is too low to estimate derivative accurately.

4-6 Stability Analysis

The definition of the proposed method in discrete time has the downside that it is not always stable. The main culprit of that is the sample time h. The sampling time is a limiting factor for the tunable parameters ω_0, f_{red} and n. This means that a large range of combinations can make the algorithm unstable. This is shown in Figure 4-10a, where the plot shows that when constant parameters are used except for n, the system can behave differently. It shows that the RMSE is increased drastically for $n = 4$ to 6. This indicates that the system is unstable for these n. On the other hand it is stable again for higher values of n. Note that these used parameters have a terrible settling time due to slow decaying oscillations because of a low value of $f_{red} = 1.4$. Nevertheless it illustrates that the system stability is sensitive to the parameters used.

Figure 4-10b, shows a similar effect when only ω_0 is varied and a constant $n = 10$ and $f_{red} = 2.0$ are used. Now the system gets unstable at the higher values of ω_0. This eventually happens because the damping is not scaled with ω_0. This means that the systems will start to oscillate more with higher values of ω_0 until the oscillations are eventually not damped anymore, resulting in an unstable system.



(a) (b)

Figure 4-10: These plots show the stability in terms of increasing RMSE of the proposed method with a constant set of parameters, but with a varying $n = 1$ to 10 in (a) and varying $\omega_0 = 100$ to 900 in (b). The input is $u = \sin(t)$ and the RMSE is recorded over the interval $t = [10, 20]$, where not all transients haves settled. This range is chosen so that the stability can be detected based on the increasing RMSE. The parameters used are $\omega_0 = 20$, $f_{red} = 1.4$ for (a) and $n = 10$, $f_{red} = 2.0$ in (b). Both have a time step of $h = 0.001$. The i in $e^{(i)}$ denotes the root mean square error of the ith derivative. In (a) something unexpected happens, the system is stable for $n = 1$ to 3 and from $n = 7$ to 10. The RMSE is high for the latter, because the system has not settled yet. In fact, the parameters that are used cause it to have slow decaying oscillations and those take long to settle. Most noticeable is that the system is unstable for $n = 4$, 5 and 6. This shows that the range of stable parameters can vary quite a lot even when its stable for others. Plot (b) shows that the system eventually gets unstable (at $\omega_0 = 1000$) by increasing ω_0. At $\omega_0 = 900$ the system seems unstable, but it is actually stable. It has very slow decaying oscillations with high amplitudes, but these have not settled yet within the RMSE interval.

The easiest way to check if the system is stable for a set of parameters is to confirm that all eigenvalues of the state transitions matrix $\mathbf{A}$ of equation (4-6) fall within the unit circle. The size of $\mathbf{A}$ scales by $2n-1$. This means that for $n = 3$ this matrix is (5×5), which results in a 5th degree polynomial if one tries to find the eigenvalues. There is no general solution to find the roots of a 5th degree polynomial or higher, therefore numerical methods provide the solution.

For example, MATLAB can find the eigenvalues easily with the function `eig(A)`. Then these eigenvalues have to be checked if their absolute values are less then one. If that is the case, the system is asymptotically stable. If at least one eigenvalue is exactly 1, the system is marginally stable and unstable if greater than one. This procedure was used extensively to tell if the parameters used caused the system to be stable.

However, numerical methods have the downside that they can be inaccurate for eigenvalues very close to 1. This means that a system can be falsely considered unstable if the rounding errors caused an eigenvalue to be just slightly greater than one. This can also happen the other way around, so that a system is falsely considered stable.

Another way to look at the stability is to see what happens in the limit case when the time step $h \to 0$. Then by setting $h = 0$ in $\mathbf{A}$ of (4-6) a lot of terms become one or zero, because $e^0 = 1$. Then by calculating the eigenvalues as $|\mathbf{A} - \lambda \mathbf{I}| = 0$ and solving for λ the expression

$$\lambda^{n-1}(1 - \lambda)^n = 0$$

is found. This shows that $n - 1$ eigenvalues are 0 and the other n eigenvalues are 1. This means that in the limit case $h \to 0$ the system is marginally stable, since it has poles exactly on the unit circle. Analyzing the limit case of $h \to 0$ does not give more information about the stability.

Concluding, the stability is determined by the tuning parameters n, f_{red}, ω_0 and h. This results in a wide range of parameters that are stable or unstable. The next section goes into depth on how

to quickly find good tuning parameters that are stable. The stability cannot be checked analytically therefore numerical methods must be used that can determine the stability. For example MATLAB can check this by verifying that the eigenvalues of the state transition matrix are within the unit circle.

4-7 Tuning rules

Section 4-6 mentioned that the stability of the proposed method is affected by all tuning parameters ω_0, f_{red} and n. The time step is not considered a direct tuning parameter, because the sample rate will be determined by external systems. Since the system depends on a wide range of parameters, it is impossible to find analytically proper tuning parameters, therefore a rule set is given to find a stable and good set of tuning parameters rapidly. When h is preset and the input signal has known derivatives the tuning rules are as follows.

1. Set n to the desired amount of derivatives to be estimated.
2. Set $f_{red} = 2.0$ for starters. This gives a good damping for a range of stable ω_0.
3. Set ω_0 high enough so that it has still enough bandwidth for higher order derivatives, because $\omega_{i+1} = \omega_0 / f_{red}^{i+1}$ have subsequently a lower cutoff frequency. For best results, the cutoff frequency ω_{n-1} associated with the highest derivative should have a cutoff frequency that is higher than the input its highest frequency. The upper bound of ω_0 is roughly limited by $\frac{1}{h}$ when $f_{red} = 2.0$ before the system becomes unstable.
4. Test the response on an given input with known derivatives and verify that the response and the estimations errors are satisfactory.
5. If the response is unstable, reduce ω_0 or increase f_{red}. A final option is to reduce n, to have less derivatives estimated in favor of stability.
6. If the response is slow, try increasing ω_0.
7. If the derivative estimation errors are too high, try increasing ω_0 or lowering f_{red}.
8. If the response has slow decaying oscillations, try increasing f_{red}. This term has the most effect on damping the oscillations.
9. Repeat step 4 to 8 until satisfactory.
10. If still not satisfactory, try reducing or increasing n, since there is an optimal n that give the lowest errors.

These tuning steps are also similar if there is noise on a signal. The only part that must be considered is that a high ω_0 also amplifies noise on the higher order derivatives. So there must be a good trade off between ω_0 and f_{red} to get good noise attenuation.

4-8 Noise analysis

The proposed method can attenuate noise better than AEAD if tuned properly. However, it also can attenuate it too much. This happens because the proposed method has low pass filter characteristics with consecutive decreasing cutoff frequencies for higher order derivatives. This eventually results in a fully damped signal that is zero and has no more information. This is demonstrated by the following analysis.

Let the input u be a steady state signal corrupted by noise so that

$$u = 0 + \sigma_n \mathcal{N}(0, 1),$$

where $\mathcal{N}(0,1)$ represents a normalized Gaussian distributed random variable with zero mean and is scaled by σ_n. This σ_n is then also the standard deviation of the input u. This input is then used for the proposed method, RLPAD, to gain insights in the noise attenuation capabilities. In additions this will also show how the noise variance is propagated over its higher order derivatives.

Figure 4-11 and Figure 4-12 shows the standard deviation of the higher order derivatives when the input is u with $\sigma_n = 0.01$. In both figures it shows that the zeroth derivative is attenuated well below the standard deviation of the noise input. Figure 4-12b shows that by increasing the gain (ω_0), the standard deviation of the zeroth derivative remains below the input standard deviation $\sigma_n = 0.01$. Concluding that it has good noise attenuation for the zeroth derivative.

The first derivative and up have all higher standard deviations than the input σ_n. This is expected, because the noise gets amplified by ω_{i-1} for every ith derivative term, but is also reduced in amplification by ω_i which have a consecutive lower bandwidth for the next i. Eventually this bandwidth is so low that the noise gets reduced even more and starts to go to zero, which can be clearly seen in Figure 4-11b. The damping term f_{red} is here so high that the higher order noise standard deviation starts to decrease. However, when it starts to decrease, it does not necessarily mean that the higher order derivative estimates have a lower variance than the input. Generally speaking, the ω_i of those have too low bandwidth to track an input signal properly. An exception to this is a constant high degree polynomial, in that case it can estimate the highest (constant) derivative with an standard deviation below the input its standard deviation. The downside of this is that the settling time is extremely slow with respect to the zeroth derivative for those cases.

The parameter ω_0 determines the rise time of the system. This means that a higher ω_0 tracks the input closer to a 1:1 ratio, making it more sensitive to noise. In turn this amplifies the noise of the higher order derivatives by a lot. Figure 4-12b shows that increasing ω_0 has a strong effect on the standard deviation of each consecutive higher order derivative estimate.

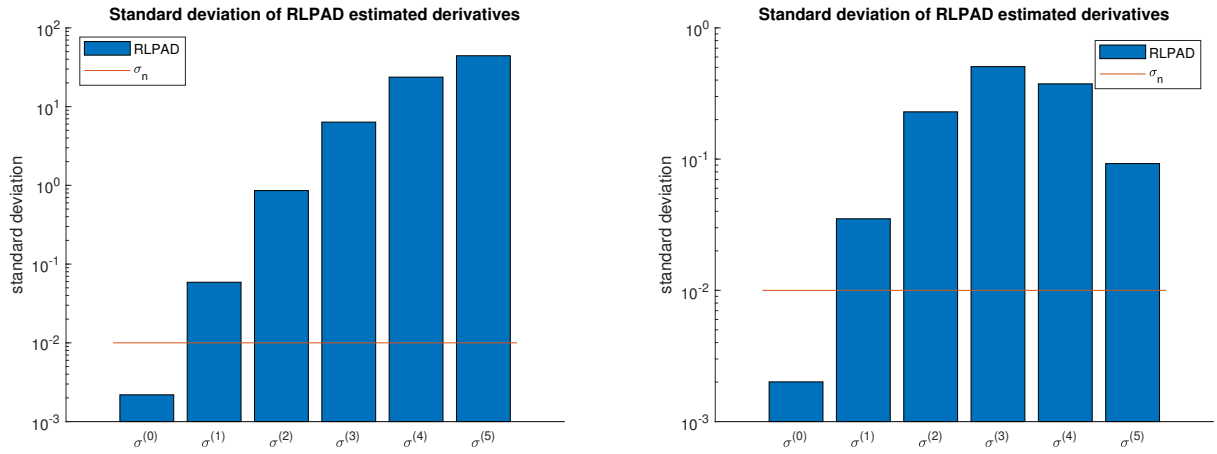
The effect of varying the parameter f_{red} on the standard deviation is shown in Figure 4-13. It shows that increasing f_{red} damps the noise and eventually damps it too much. This is indicated by the crossing of $\sigma^{(5)}$ with $\sigma^{(4)}$ and $\sigma^{(3)}$. The suggested starting value of $f_{red} = 2.0$ for tuning the filter is based on this figure, because it shows that the fifth order derivative is not yet over damped.

Summarizing, the noise is attenuated properly for the zeroth order derivative estimate so that the standard deviation of the output is always lower than that of the input. However, the higher order derivatives amplify this standard deviation and have consecutively less attenuation for each higher order derivative. The system can be over damped to force noise attenuation on the higher order derivatives, but this will also result in removing any useful information about them. The parameters that affect noise attenuation mostly are ω_0 and f_{red}, where the first determines the gain and the latter the damping.

4-8-1 Knowledge about the higher order variance is required in Active Inference

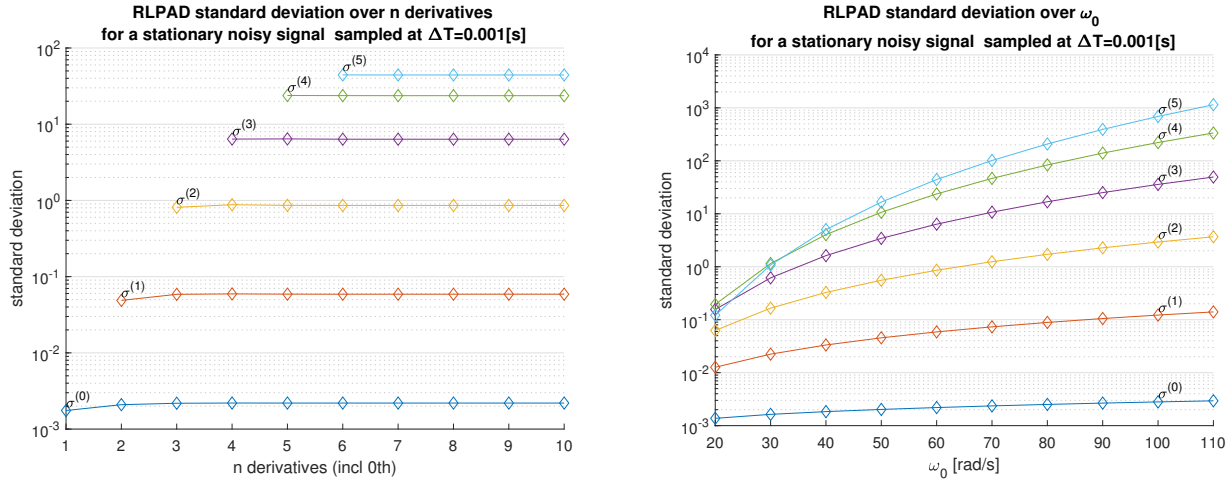
Active inference requires not only the generalized measurements, but also the precisions of the measurements for Π_z as was mentioned in section 2-2. This means that the variance of the noisy input and also the variances of the estimated derivatives are required for Active Inference. Finding these dependent variances is not an easy task. For example, one could use the error propagation formula, because the variance should get propagated to higher order derivatives. However, this will not work, since error propagation assumes every variable independent. That is clearly not the case for the proposed method, because all higher order derivatives are a function of the lower ones. In fact the noise of all estimated derivatives is dependent on the noise of the input signal. Solving this analytically is out of the scope of this thesis.

A solution to find the variances of the higher order derivatives is to simulate the system with white Gaussian noise for a large number of steps. Then from these simulations, the standard deviations can be estimated and thus the variances. However, this numeric method for finding variances of higher order derivatives can also be used for all other existing differentiators. These variances would then be



(a) (b)

Figure 4-11: These plots show the standard deviation of the estimated derivatives for a stationary signal with $\sigma_n = 0.01$. The parameters used are $n = 10$, $\omega_0 = 60$, $h = 0.001$, $f_{red} = 2.0$ in (a) and $f_{red} = 3.0$ in (b). The standard deviation is calculated over 20001 samples and the red line indicates the standard deviation of the input noise. It can be seen that in both plots the zeroth derivative is attenuated so that it falls below the original standard deviation. This means that it can improve a noisy signal. The standard deviation increase over the amount of higher order derivatives estimated. However, (b) shows that it starts to get lower again. This is due to the higher damping term $f_{red} = 3.0$. This reduction in standard deviation does not mean that the system can track this derivative estimate more accurate, because it might not track it at all because the bandwidth was too low.



(a) (b)

Figure 4-12: These plots show how the standard deviation of the estimated derivatives change over the states $n = 1$ to 10 in (a) and over $\omega_0 = 20$ to 110 in (b). In both plots the parameters are $f_{red} = 2.0$, $h = 0.001$ and the input is a stationary signal with $\sigma_n = 0.01$. Plot (a) has $\omega_0 = 60$ and (b) has $n = 10$ and are recorded over 20001 samples. The standard deviation of the derivative estimates are denoted by $\sigma^{(i)}$, where i is the ith derivative. Plot (a) shows that increasing the number of estimated derivatives does not have much effect on the standard deviation. It can be seen that there is a slight bump at each second data point, but after that the standard deviation is almost constant. Plot (b) shows the effect of an increasing ω_0 on the standard deviation. It can be seen that a higher value of omega increases the derivative errors and they seem to asymptotically converge to a constant standard deviation. However, they do not, and besides that the system becomes unstable before the standard deviations can converge. It also shows that $\sigma^0 < \sigma_n$, which indicates that the 0th order derivative estimate is better than a raw noisy input.

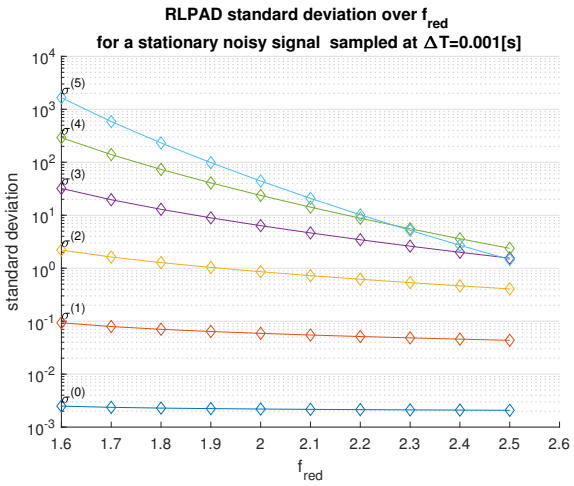


Figure 4-13: These plots show how the standard deviation of the estimated derivatives change over the parameter f_{red}, where $f_{red} = 1.6$ to 2.5. The parameters used are $n = 10.0$, $\omega_0 = 60$, $h = 0.001$ and the input is a stationary signal with $\sigma_n = 0.01$. The standard deviation of the derivative estimates are recorded over 20001 samples and are denoted by $\sigma^{(i)}$, where i is the ith derivative. This figure shows that by increasing f_{red} the standard deviation is also reduced for each order derivative. It affects lower order less with respect to the higher ones. It can be seen that $\sigma^{(5)}$ eventually crosses $\sigma^{(4)}$. This is due to the fact that at some point f_{red} damps the higher derivative estimates too much and go to zero. Note that this plot is similar to Figure 4-11, but with more samples of f_{red}.

assumed independent (but are not) and used for the precision matrix in Active Inference. However, there is another problem, the noise that remains on the output of a differentiator is smoothed and becomes therefore smooth noise or colored noise. This is something that Active Inference can take care of, because it is build around the idea that real world noises are generated by dynamic processes [17] and are therefore smooth. However, taking care of smooth noise properly is a complex task and is out of the scope of this thesis.

Chapter 5

Numerical Experiments

In this chapter the performance of the algebraic estimation approach differentiator (AEAD) is compared to the proposed method, the recurrent low pass algebraic differentiator (RLPAD). The methods are tested with a known (causal) input and derivatives of that input. The tracking functions are a polynomial, a sinusoid and a combination of the two. Then they are evaluated based on the root mean square error (RMSE) with respect to the true values. The most important experiment is that the differentiators are also evaluated when the tracking signal has additive noise on it. The capability of determining the unknown derivatives of the input signal as close to the ground truth as possible determines what the better differentiator is.

AEAD and the proposed method have as input a polynomial, a sinusoid and a combination of the two. The reason for using a polynomial is that a second degree polynomial can model a free falling object with no drag. This means, that polynomials are encountered in physics and are therefore realistic to test the differentiators on. The same is true for a sinusoid. A mass spring system has for example a sinusoidal oscillation. These facts makes the combination of the two also a realistic input. The proposed method and AEAD are tested with the following input functions.

1. The differentiators are tested with a 5th degree polynomial as input which is

$$\begin{aligned} u(t) &= x_0 + v_0 t + a_0 \frac{1}{2} t^2 + j_0 \frac{1}{6} t^3 + s_0 \frac{1}{20} t^4 + c_0 \frac{1}{120} t^5 \\ &\text{with } x_0 = 0, \;\; v_0 = 2, \;\; a_0 = 1, \;\; j_0 = -0.5, \;\; s_0 = 0.25, \;\; c_0 = -0.125. \end{aligned} \tag{5-1}$$

   This function can be interpreted as a constant "crackle" function ("crackle" is the 5th order derivative of position). This means that is has has non zero derivatives up to the 5th order.

2. A sinusoid input

$$u(t) = \sin(2t), \tag{5-2}$$

   which has an infinite order of derivatives. The frequency of 2 rad/s is used to match a test case in the works of Kasac et al (2018) [32]. This is the same reason for why the cutoff frequency of 5.0 rad/s is given to the AEAD.

3. A polynomial and sinusoid combined to make the input function:

$$\begin{aligned} u(t) &= \sin(t) + 3\cos(at) + bt^3 \\ \text{with } a &= \frac{1}{10} \\ b &= \frac{1}{100} \end{aligned} \tag{5-3}$$

   which is the same function as in the Master's thesis of I. Heijne [28]. This function also has an infinite number of derivatives due to the periodicity.

The derivatives are known of the input functions, this means that the performance of the differentiators is evaluated by a root mean square error over a time interval.

All functions can be corrupted by additive white noise that is based on a normalized Gaussian distribution $\sigma\mathcal{N}(0,1)$. The values picked from the distribution are multiplied by a constant σ, which is also the value of the standard deviation of the resulting distribution.

To give the proposed method an equal playing field, the continuous time state space of AEAD is converted to a discrete time state space. This is done in MATLAB by using the `c2d(SS, dt)` function, that uses the most accurate conversion method. Both differentiators are set to $n = 10$ to estimate 10 derivatives (including the 0th). This high amount of estimates $n = 10$ has the most advantage for AEAD, the zeroth estimates $y^{(0)}$ is improved significantly by the amount of n terms. However, more terms do not necessarily lead to better derivative estimations especially in the presence of noise. A good balance for AEAD is $n = 10$ and this value is also set for the proposed method so that they estimate an equal number of derivatives. This is a disadvantage for the proposed method, because adding that many states do not necessarily improve its estimates and can even make them worse.

The performance of the zeroth to the fifth order derivative are evaluated. This is done because according to Friston, the generalized states in Active Inference stop containing useful information after an embedding order of six [24]. This is reduced to the fifth order for readability in the figures and also because the differentiators start to perform worse for even higher derivatives than five.

All plots in this chapter are created with an input and differentiator sampled at $h = \Delta T = 0.01$ seconds (100Hz) unless stated otherwise. This sampling rate is chosen with the thought that it is not considered a too high or too low frequency. This means that increasing the sample frequency results in better estimates and the other way around for a lower sample frequency. The following sections evaluate the performance of the differentiators per input function stated above.

5-1 The proposed method outperforms AEAD for polynomial inputs

The main reason why RLPAD outperforms AEAD is that the latter cannot track the polynomial signal. The estimated derivatives diverge over time which can be clearly seen in Figure 5-1b. Reducing the step time does not prevent this problem, but it does make the estimates diverge slower as can be seen in Table 5-2.

For the case without noise the following tuning parameters are used. AEAD has $\omega_c = 5$ rad/s, this gives a fast enough transient. RLPAD has the parameters $\omega_0 = 50$ rad/s and $f_{red} = 2.0$, this also gives a fast transient while keeping enough bandwidth for higher order derivatives. The performance of the differentiators with a noise free polynomial as input is plotted in Figure 5-1.

Without noise, RLPAD tracks the polynomial and its derivatives with very low errors, which can be seen from Table 5-1 and Figure 5-1cd. The figure also shows that the higher order derivatives settle slower consecutively, the reason for that is of every $\omega_{0,1\ldots,n-1}$ is decreasing for each extra derivative. AEAD is no real competition here, because it can't track the derivatives and give high errors.

The differentiators are retuned for the input with noise to have a lower RMSE and better noise attenuation. Retuning AEAD did not have much effect, therefore $\omega_c = 5$ rad/s. A lower ω_c would attenuate the noise more, but the rise time also becomes slower which did not help in this case. RLPAD is retuned by setting $f_{red} = 3.0$ to have better noise attenuation.

The tracking performance of the differentiators for a polynomial input with noise is plotted in Figure 5-2 and have three important aspects to it. The first is that AEAD behaves the same as before and the noise does not make a real difference. Although it can be seen that the noise has significantly less attenuation than RLPAD. The second is that RLPAD in (c) shows almost no visible difference compared to the one without noise in Figure 5-2c. The errors are still far below one which can be seen in Table 5-3. The errors of the 0th to 2nd derivative have increased by a factor of roughly 10 to 100 compared to Table 5-1. The third is that the attenuation of the noise is very good in (d), but has a slow settling time due to a higher f_{red} value.

Concluding, the RLPAD outperforms AEAD significantly, because AEAD cannot track a polynomial signal. RLPAD on itself tracks a polynomial signal with very small errors and can estimate the higher order derivatives, even in the presence of white noise.

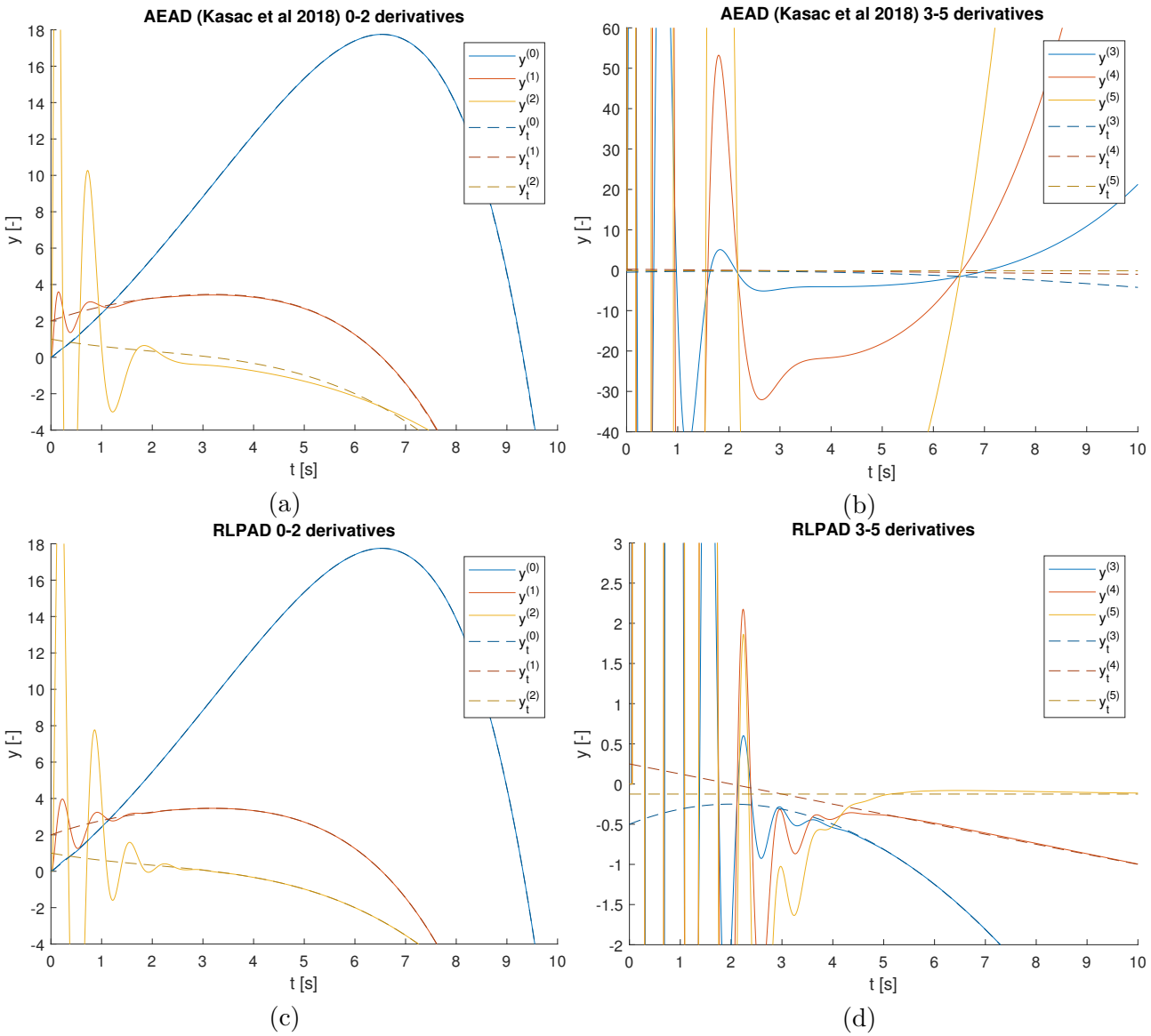


Figure 5-1: Plot of the trajectories of AEAD (a-b) and RLPAD (c-d) for a polynomial (5-1) input function sampled at $h = 0.01$ seconds. The parameters used are $\omega_c = 5$ rad/s for AEAD, for RLPAD $\omega_0 = 50$ rad/s and $f_{red} = 2.0$. Both differentiators use $n = 10$ for the amount of estimated derivatives and their sample time is $h = 0.01$ seconds. The solid lines are the trajectories estimated and dashed lines are the true derivatives. The superscript y^i denotes the i-th derivative. The top images show clearly that AEAD can't track this polynomial function, especially in (b). The error of $y^{(0)}$ also increases over time but its not that apparent from the plot itself. RLPAD on the other hand tracks the polynomial with very low error for all derivatives. It shows that its converging on (d) the higher order derivative. It outperforms AEAD significantly for a polynomial input.

5-2 The proposed method is slightly better for a sinusoid input

The input used for this section is the sinusoidal function of equation (5-2) which is $u(t) = \sin(2t)$. For the case without noise the following tuning parameters are used. AEAD has $\omega_c = 5$ rad/s, because this was also used in Kasac et al (2018) [32] for the same input. RLPAD has the parameters $\omega_0 = 100$ rad/s and $f_{red} = 2.4$, which gives a fast response similar to AEAD, but with more oscillations. The performance of the differentiators with a noise free sinusoidal input is plotted in Figure 5-3 and the

	RMSE Polynomial with time step $\Delta T = 0.01$ and $n = 10$					
Method	$e^{(0)}$	$e^{(1)}$	$e^{(2)}$	$e^{(3)}$	$e^{(4)}$	$e^{(5)}$
AEAD	1.118	2.186	25.8	$2.25 \cdot 10^2$	$1.35 \cdot 10^3$	$5.63 \cdot 10^3$
RLPAD	$1.20 \cdot 10^{-6}$	$4.22 \cdot 10^{-5}$	$5.49 \cdot 10^{-4}$	$4.63 \cdot 10^{-3}$	$2.14 \cdot 10^{-2}$	$4.740 \cdot 10^{-2}$

Table 5-1: This table shows the root mean square errors of the derivative estimations with the true ones of the polynomial input. The RMSE is recorded over the interval $t = [4.0, 20.0]$ seconds after the main transient. The i-th derivative error is denoted by $e^{(i)}$ .The parameters used are $\omega_c = 5$ rad/s for AEAD, for RLPAD $\omega_0 = 50$ rad/s and $f_{red} = 2.0$. Both differentiators use $n = 10$ for the amount of estimated derivatives and their sample time is $h = 0.01$ seconds. It is clear that RLPAD performs alot better with all errors far below one. The RMSE of AEAD is really high in comparison, but this is due to the fact that AEAD can't track polynomial signals, because the derivative estimates diverge over time.

	RMSE Polynomial with time step $\Delta T = 0.001$ and $n = 10$					
Method	$e^{(0)}$	$e^{(1)}$	$e^{(2)}$	$e^{(3)}$	$e^{(4)}$	$e^{(5)}$
AEAD	$1.04 \cdot 10^{-1}$	$4.48 \cdot 10^{-2}$	$2.63 \cdot 10^{-1}$	2.26	$1.36 \cdot 10^1$	$5.65 \cdot 10^1$
RLPAD	$2.71 \cdot 10^{-7}$	$1.26 \cdot 10^{-5}$	$2.88 \cdot 10^{-4}$	$3.28 \cdot 10^{-3}$	$1.76 \cdot 10^{-2}$	$4.172 \cdot 10^{-2}$

Table 5-2: This table shows the root mean square errors of the derivative estimations with the true ones of the polynomial input. The RMSE is recorded over the interval $t = [4.0, 20.0]$ seconds after the main transient. The parameters for the differentiators are the same as in Table 5-1, but now the sample time is set to $h = 0.001$. The conclusion did not change, but it shows that by making the time step smaller, the errors also become smaller with constant differentiator parameters. The derivative estimates of AEAD still diverge, but slower when the sample time is smaller. RLPAD could even get lower estimation errors if a higher ω_0 was used.

	RMSE Noisy ($\sigma = 0.001$) Polynomial with time step $\Delta T = 0.01$ and $n = 10$					
Method	$e^{(0)}$	$e^{(1)}$	$e^{(2)}$	$e^{(3)}$	$e^{(4)}$	$e^{(5)}$
AEAD	1.12	2.19	25.8	$2.25 \cdot 10^2$	$1.35 \cdot 10^3$	$5.63 \cdot 10^3$
RLPAD	$5.36 \cdot 10^{-4}$	$6.04 \cdot 10^{-4}$	$3.17 \cdot 10^{-2}$	$7.77 \cdot 10^{-2}$	$1.02 \cdot 10^{-1}$	$4.98 \cdot 10^{-2}$

Table 5-3: This table shows the root mean square errors of the derivative estimations with the true ones of the noisy polynomial input. The RMSE is recorded over the interval $t = [4.0, 20.0]$ seconds after the main transient. The i-th derivative error is denoted by $e^{(i)}$. The parameters used are $\omega_c = 5$ rad/s for AEAD, for RLPAD $\omega_0 = 50$ rad/s and $f_{red} = 3.0$. Both differentiators use $n = 10$ for the amount of estimated derivatives and their sample time is $h = 0.01$ seconds. The errors of AEAD are not affected by the noise that much, because the divergence error is leading. RLPAD shows that with the presence of noise, the errors of the higher order derivatives are very low. The error $e^{(0)}$ is affected most compared to Table 5-1. This shows that RLPAD has good noise attenuation.

	RMSE Noisy ($\sigma = 0.01$) Polynomial with time step $\Delta T = 0.01$ and $n = 10$					
Method	$e^{(0)}$	$e^{(1)}$	$e^{(2)}$	$e^{(3)}$	$e^{(4)}$	$e^{(5)}$
AEAD	1.12	2.19	25.8	$2.26 \cdot 10^2$	$1.36 \cdot 10^3$	$5.65 \cdot 10^3$
RLPAD	$5.36 \cdot 10^{-3}$	$5.93 \cdot 10^{-2}$	$2.93 \cdot 10^{-1}$	$5.27 \cdot 10^{-1}$	$3.34 \cdot 10^{-1}$	$8.19 \cdot 10^{-2}$

Table 5-4: This table shows the root mean square errors of the derivative estimations with the true ones of the noisy polynomial input. The RMSE is recorded over the interval $t = [4.0, 20.0]$ seconds after the main transient. The i-th derivative error is denoted by $e^{(i)}$. The same parameters are used as in Table 5-3, but now the noise standard devation is increased by a factor 10 ($\sigma = 0.01$). It shows that increasing the standard deviation by a factor 10 scales the errors of RLPAD roughly by a factor 10 as well. Especially for the first three derivatives.

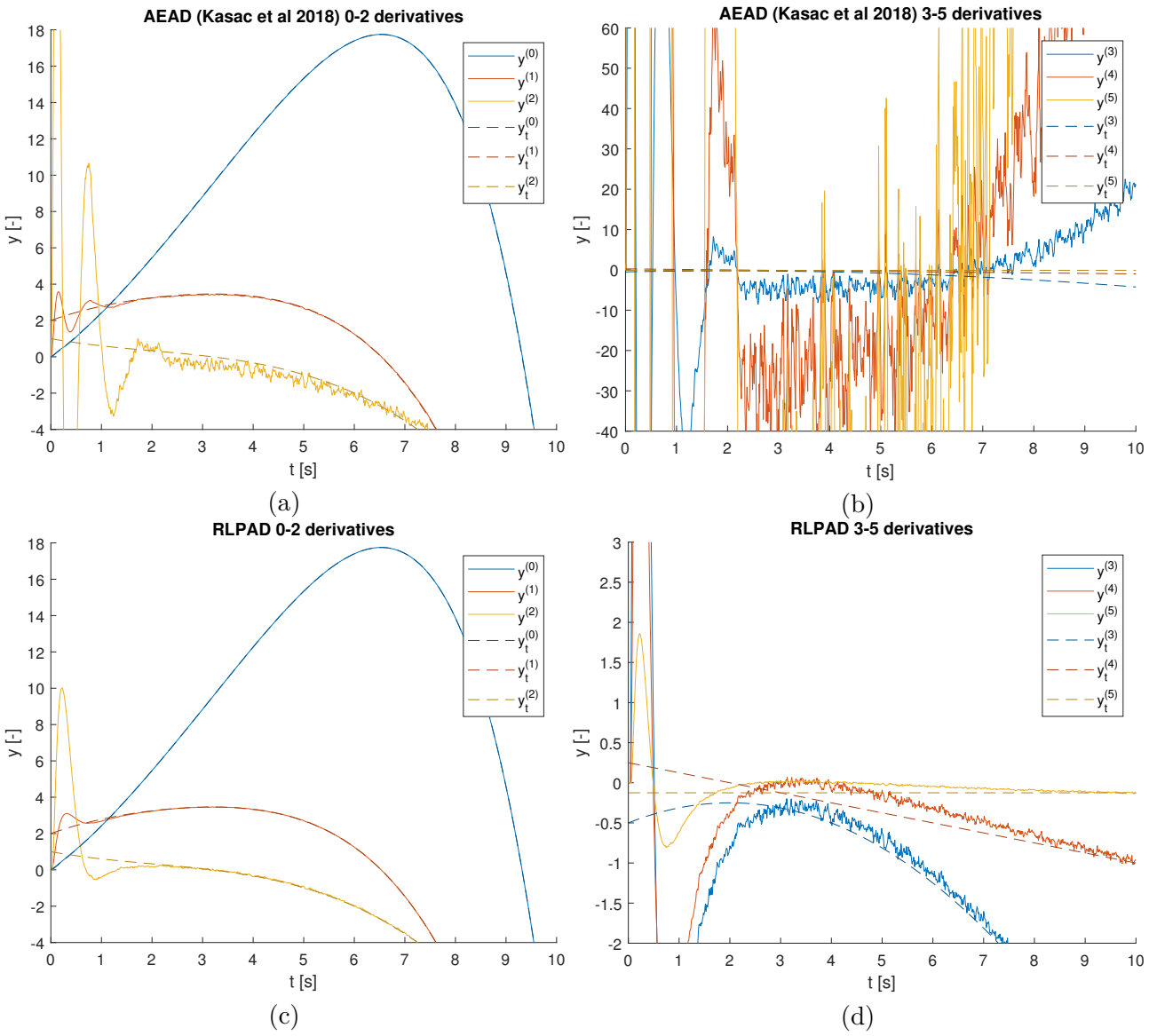


Figure 5-2: Plot of the trajectories of AEAD (a-b) and RLPAD (c-d) for a polynomial input function (5-1) with additive noise with standard deviation $\sigma = 0.001$ sampled at $h = 0.01$ seconds. The parameters used are $\omega_c = 5$ rad/s for AEAD, for RLPAD $\omega_0 = 50$ rad/s and $f_{red} = 3.0$. Both differentiators use $n = 10$ for the amount of estimated derivatives and their sample time is $h = 0.01$ seconds. The solid lines are the trajectories estimated and dashed lines are the true derivatives. The superscript y^i denotes the i-th derivative. The conclusion is still the same for AEAD, it cannot track polynomials. Besides that, it can be seen that the noise is amplified considerably for the higher order derivatives in (a-b). RLPAD tracks the polynomial and attenuates the noise significantly better.

corresponding root mean square errors of the derivative estimates are tabulated in Table 5-5 and tell the following.

Both AEAD and the proposed method can track a sinusoidal signal. Figure 5-3 shows that both differentiators are capable of estimating the zeroth to second order derivative accurately, because they both have errors well below one in Table 5-5. RLPAD has a lower RMSE for $e^{(0)}$ and $e^{(1)}$, however, AEAD has a lower RMSE for the second up to the fourth derivative estimation. Both $y^{(2)}$ estimates lead slightly the true value, but the lead is less for AEAD in Figure 5-3a. Figure 5-3b show clearly why the third and fourth derivative estimate have low errors with respect to (d). The overshoots of the third derivative estimate $y^{(3)}$ in (b) is slightly lower than the one in (d). The fourth derivative lags for both differentiators but the amplitude is a lot higher for RLPAD compared to AEAD.

A more interesting case will be a sinusoid with noise, because now it is possible to compare both differentiators on a function that they can track. The parameters of AEAD remain unchanged, but

RLPAD is retuned with $\omega_0 = 60$ and $f_{red} = 2.0$. The standard deviation of the noise is $\sigma = 0.001$. The performance of the differentiators with a noisy sinusoidal input are plotted in Figure 5-4 with the corresponding root mean square errors tabulated in Table 5-6 and concludes the following three points.

The first point is that both differentiators can estimate the zeroth up to the second derivative with a low RMSE, as can be seen in Table 5-6 without amplifying the noise too much. Also in this case RLPAD has a better performance for the zeroth derivative, it is a factor 11 lower than AEAD. This table also shows that AEAD did not improve the noisy input, because the RMSE of $e^{(0)}$ is greater than the standard deviation of the noise ($e^{(0)} > \sigma$), RLPAD on the other hand did improve it. The second point is that both differentiators cannot estimate derivatives higher than the second order with the parameters used. Although both can estimate the trend of the third order derivative with some overshoot and not too much lag. The trends of the estimated higher order derivatives are still very apparent with respect to Figure 5-3b and (d), but it is somewhat harder to spot for AEAD as the noise is amplified quite a lot. The third point is that the the proposed method has better noise attenuation for the third and higher derivative estimates. This can be clearly seen in Figure 5-3d for the fourth and fifth derivative with respect to Figure 5-3b. AEAD has a far larger variance compared to RLPAD.

Summarizing, both differentiators can track the low frequency sinusoid input $\sin(2t)$ with and without noise, but they fail to track the fourth derivative. However, it is important to note that higher order derivatives of the noiseless input get more accurate when the time step becomes smaller and the gains higher, especially for RLPAD. Which has a lower RMSE for the zeroth derivative estimate and has roughly equal errors for the higher order derivative estimates compared to AEAD. However, AEAD has a slightly lower error for estimating the second derivative and higher. For the fact that RLPAD estimates the zeroth order derivative a factor ≈ 10 better in the presence of noise, it can be concluded that the proposed method is slightly better than AEAD for a low frequency sinusoid.

	RMSE Sinusoid with time step $\Delta T = 0.01$ and $n = 10$					
Method	$e^{(0)}$	$e^{(1)}$	$e^{(2)}$	$e^{(3)}$	$e^{(4)}$	$e^{(5)}$
AEAD	$7.59 \cdot 10^{-3}$	$1.77 \cdot 10^{-2}$	$1.30 \cdot 10^{-1}$	1.73	8.46	$3.90 \cdot 10^{1}$
RLPAD	$2.20 \cdot 10^{-4}$	$1.29 \cdot 10^{-2}$	$1.96 \cdot 10^{-1}$	2.19	$1.34 \cdot 10^{1}$	$3.84 \cdot 10^{1}$

Table 5-5: This table shows the root mean square errors of the derivative estimations with the true ones of the sinusoidal input. The RMSE is recorded over the interval $t = [4.0, 20.0]$ seconds after the main transient. The i-th derivative error is denoted by $e^{(i)}$. The parameters used are $\omega_c = 5$ rad/s for AEAD, for RLPAD $\omega_0 = 100$ rad/s and $f_{red} = 2.4$. Both differentiators use $n = 10$ for the amount of estimated derivatives and their sample time is $h = 0.01$ seconds. Both differentiators perform similar, the only large difference in error is in the zeroth derivative. RLPAD has a $\approx 35\times$ lower error than AEAD for $e^{(0)}$, the others have a difference of a factor 2 at max.

	RMSE Noisy ($\sigma = 0.001$) sinusoid with time step $\Delta T = 0.01$ and $n = 10$					
Method	$e^{(0)}$	$e^{(1)}$	$e^{(2)}$	$e^{(3)}$	$e^{(4)}$	$e^{(5)}$
AEAD	$7.62 \cdot 10^{-3}$	$2.25 \cdot 10^{-2}$	$2.29 \cdot 10^{-1}$	2.17	$1.32 \cdot 10^{1}$	$5.75 \cdot 10^{1}$
RLPAD	$6.69 \cdot 10^{-4}$	$1.75 \cdot 10^{-2}$	$2.46 \cdot 10^{-1}$	2.37	$1.42 \cdot 10^{1}$	$4.77 \cdot 10^{1}$

Table 5-6: This table shows the root mean square errors of the derivative estimations with the true ones of the sinusoidal input. The RMSE is recorded over the interval $t = [4.0, 20.0]$ seconds after the main transient. The i-th derivative error is denoted by $e^{(i)}$. The parameters used are $\omega_c = 5$ rad/s for AEAD, for RLPAD $\omega_0 = 60$ rad/s and $f_{red} = 2.0$. Both differentiators use $n = 10$ for the amount of estimated derivatives and their sample time is $h = 0.01$ seconds. Both differentiators perform similar, the only large difference in error is in the zeroth derivative. RLPAD has a $\approx 11\times$ lower error than AEAD for the zeroth.

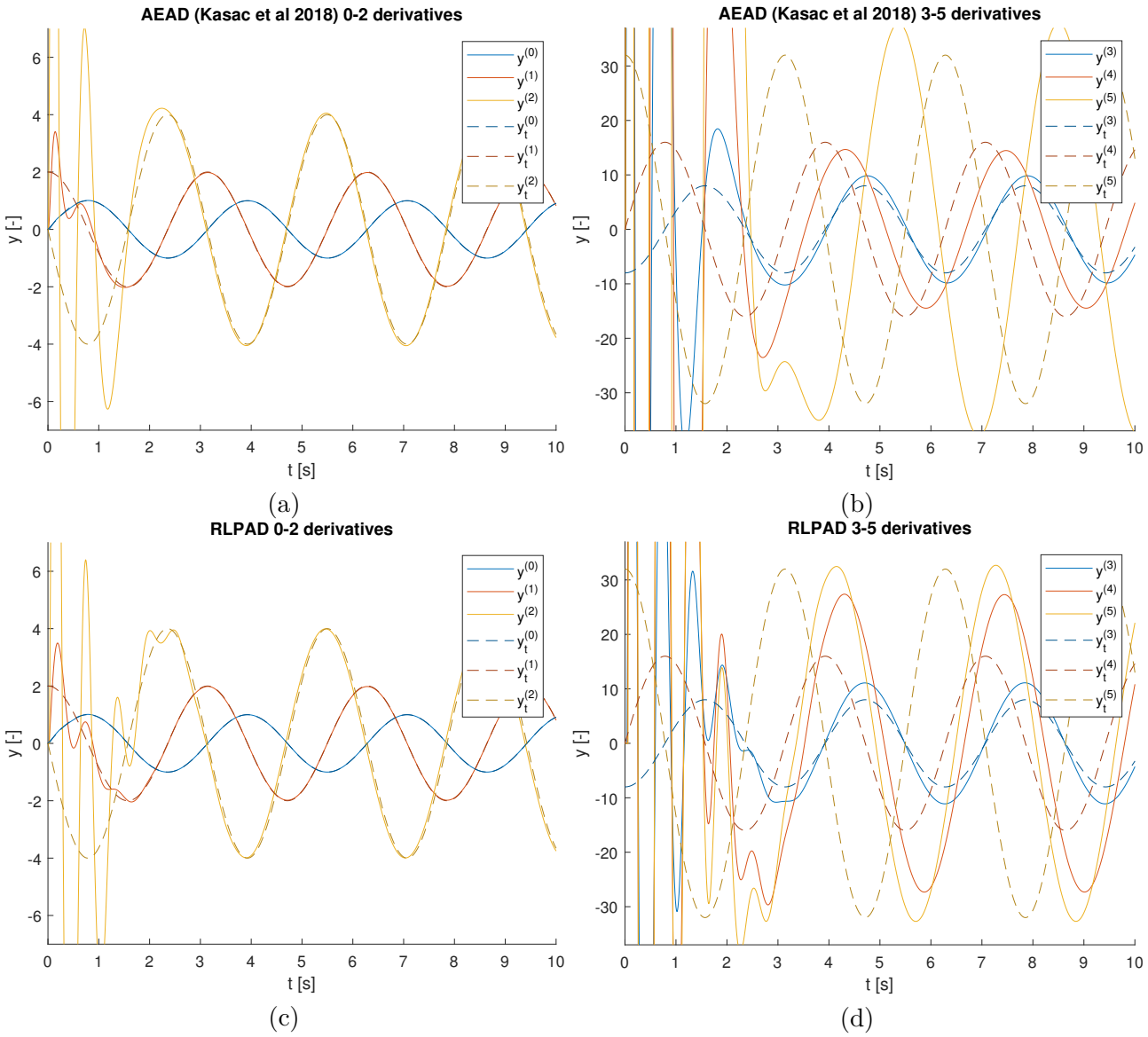


Figure 5-3: Plot of the trajectories of AEAD (a-b) and RLPAD (c-d) for a sinusoidal input function (5-2) sampled at $h = 0.01$ seconds. The parameters used are $\omega_c = 5$ rad/s for AEAD, for RLPAD $\omega_0 = 100$ rad/s and $f_{red} = 2.4$. Both differentiators use $n = 10$ for the amount of estimated derivatives and their sample time is $h = 0.01$ seconds. The solid lines are the trajectories estimated and dashed lines are the true derivatives. The superscript y^i denotes the i-th derivative. Both differentiators track up to the second derivative with low errors (a) and (c). Although the second order derivative is leading the true one and leads slightly more for RLPAD. The right images shows that they both cannot track the higher order derivatives properly. The third order derivative estimate follows the trend, but overshoots for both differentiators. RLPAD has more overshoot in it than AEAD.

5-2-1 High frequency sinusoid tracking

Differentiators are very sensitive to their time step when a periodic function is used as input. The previous section shows that for tracking the derivatives of a low frequency signal ($\sin(2t)$ has a frequency of 0.3 Hz), the sample rate has to be relatively very high (100 Hz). This results in accurate derivatives estimations up to the second derivative.

A general downside of differentiators is that they need a relatively high bandwidth compared to the system that is being tracked. It must be said that RLPAD is more sensitive to sampling rates than AEAD, however, in the end both cannot track the derivatives properly when the sampling rate is too low.

The following figure (Figure 5-5) demonstrates that differentiators are capable of tracking a periodic

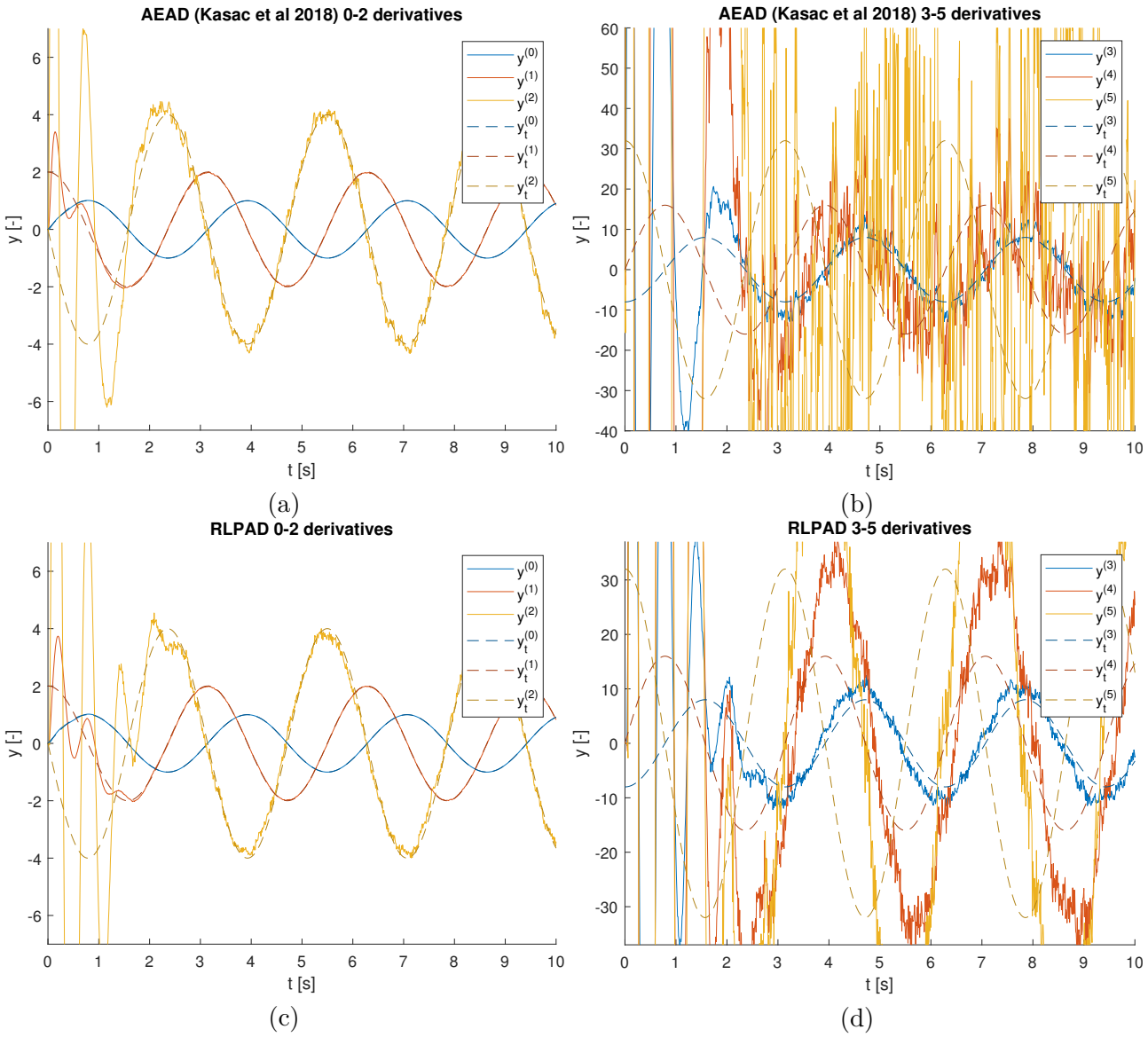


Figure 5-4: Plot of the trajectories of AEAD (a-b) and RLPAD (c-d) for a sinusoidal input function (5-2) with additive noise with standard deviation $\sigma = 0.001$ sampled at $h = 0.01$ seconds. The parameters used are $\omega_c = 5$ rad/s for AEAD, for RLPAD $\omega_0 = 60$ rad/s and $f_{red} = 2.0$. Both differentiators use $n = 10$ for the amount of estimated derivatives and their sample time is $h = 0.01$ seconds. The solid lines are the trajectories estimated and dashed lines are the true derivatives. The superscript y^i denotes the i-th derivative. Both differentiators perform well at estimating the first two derivatives in (a) and (c). The noise is attenuated, but it can be clearly seen that AEAD has worse noise attenuation in (b) with respect to (d). RLPAD has better noise attenuation, but both struggle with estimating the fourth and higher derivative.

function with a higher frequency when the sampling rate is sufficient. The input to the differentiator is $u = \sin(20t)$ corrupted by a Gaussian noise with a standard deviation of $\sigma = 0.01$.

This figure shows that the performance is very similar for both RLPAD and AEAD. This is also shown by the RMSE in Table 5-6. However, AEAD is slightly better at estimating the first and second derivatives. Figure 5-6b shows that the errors lie very close to each other when a log scale is used, but it can be seen that RLPAD has clearly a better zeroth order derivative estimate when $n = 10$. Considering that they perform mostly similar, the tracking performance of a high frequency sinusoid is considered to be equal. However, the conclusion that RLPAD performs slightly better than AEAD for a sinusoidal input does not change. The reason for this can also be found in Figure 5-6a, RLPAD settles faster to an optimal zeroth order derivative compared to AEAD when fewer derivatives are to be estimated (lower n).

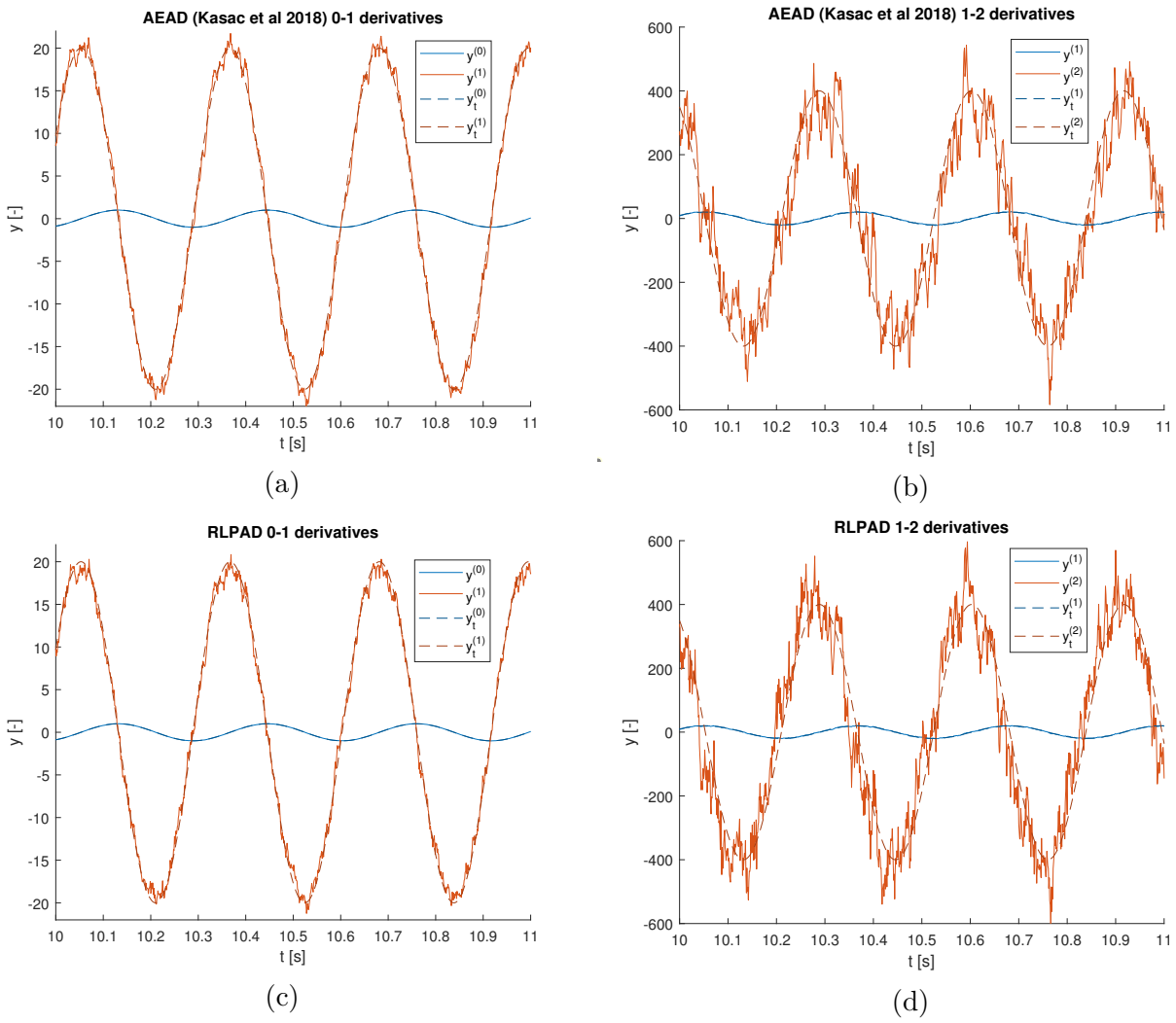


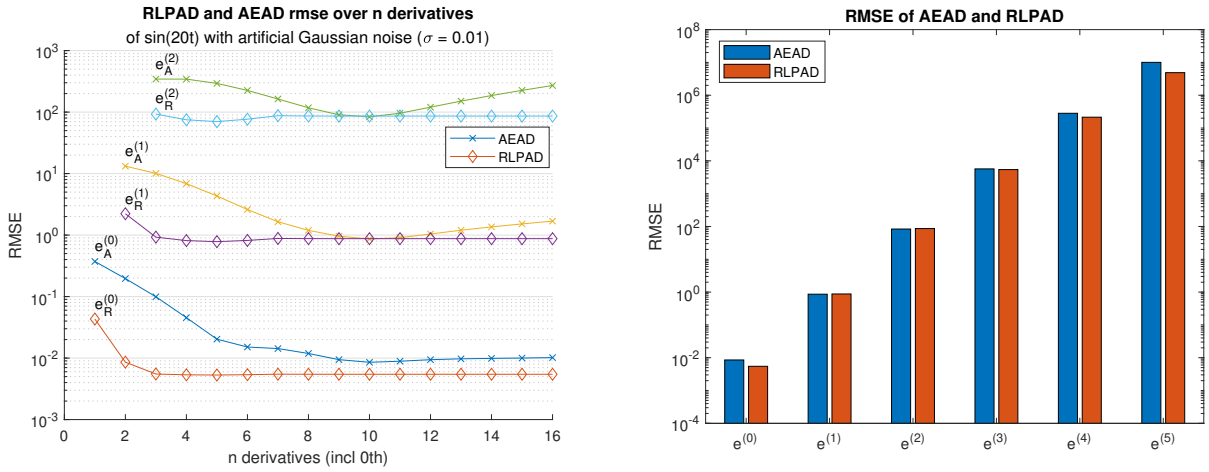
Figure 5-5: Plot of the trajectories of AEAD (a-b) and RLPAD (c-d) for a sinusoidal input function $\sin(20t)$ with additive noise with standard deviation $\sigma = 0.01$ sampled at $h = 0.001$ seconds. The parameters used are $\omega_c = 33$ rad/s for AEAD, for RLPAD $\omega_0 = 400$ rad/s and $f_{red} = 2.0$. Both differentiators use $n = 10$ for the amount of estimated derivatives and their sample time is $h = 0.001$ seconds. The solid lines are the trajectories estimated and dashed lines are the true derivatives. The superscript $y^{(i)}$ denotes the i-th derivative. The figures only show the zeroth, first and second derivative, because higher derivatives are not tracked properly (they lag) and are too hard to make out due to the increasing amplitude of the derivatives. Both differentiators seems to have the same behavior. They track all derivatives similar and attenuate the noise roughly by the same amount.

	RMSE Noisy ($\sigma = 0.01$) input $\sin(20t)$ with time step $\Delta T = 0.001$ and $n = 10$					
Method	$e^{(0)}$	$e^{(1)}$	$e^{(2)}$	$e^{(3)}$	$e^{(4)}$	$e^{(5)}$
AEAD	$8.54 \cdot 10^{-3}$	$8.63 \cdot 10^{-1}$	$8.36 \cdot 10^{1}$	$5.69 \cdot 10^{3}$	$2.82 \cdot 10^{5}$	$1.01 \cdot 10^{7}$
RLPAD	$5.46 \cdot 10^{-3}$	$8.75 \cdot 10^{-1}$	$8.62 \cdot 10^{1}$	$5.45 \cdot 10^{3}$	$2.15 \cdot 10^{5}$	$4.83 \cdot 10^{6}$

Table 5-7: This table shows the root mean square errors of the derivative estimations with the true ones of the input $\sin(20t)$ with noise. The RMSE is recorded over the interval $t = [4.0, 20.0]$ seconds after the main transient. The i-th derivative error is denoted by $e^{(i)}$ .The parameters used are $\omega_c = 33$ rad/s for AEAD, for RLPAD $\omega_0 = 400$ rad/s and $f_{red} = 2.0$. Both differentiators use $n = 10$ for the amount of estimated derivatives and their sample time is $h = 0.001$ seconds. Both differentiators perform very similar and both can attenuate the noise enough so that the zeroth derivative has a lower error than the standard deivation.

Additional insights

Generally more states make the differentiators derivative estimates more accurate, but this is not always the case. Especially when noise is present on an input. This is shown in Figure 5-6a. The RMSE of the zeroth order derivative estimate of RLPAD caps out very rapidly at about $n = 4$. Which makes sense, because the Taylor series expansion has almost no influence on the zeroth derivative when $n > 4$. On the other hand, AEAD continuous to improve its zeroth order estimate, coincidently its best estimate is at $n = 10$. This figure can change a lot depending on the type of signal and noise as is shown later in chapter 6 when real sensor data is used. However, this figure indicates that there is an optimal set of parameters for an arbitrary input that give the lowest RMSE for both differentiators.



(a) (b)

Figure 5-6: RMSE of AEAD vs RLPAD over n in (a) and at $n = 10$ in (b) for a sinusoidal input function $\sin(20t)$ with additive noise with standard deviation $\sigma = 0.01$ sampled at $h = 0.001$ seconds. The parameters used are $\omega_c = 33$ rad/s for AEAD, for RLPAD $\omega_0 = 400$ rad/s and $f_{red} = 2.0$. It shows that RLPAD reaches its constant RMSE very rapidly over n states, while AEAD improves the estimates slowly over n, until it eventually increases again. Coincidently, that point is at $n = 10$ for the input used.

5-3 RLPAD outperforms AEAD for a sinusoid combined with polynomial as input

The input function that is used next is the sinusoid combined with a polynomial of equation (5-3) and written below for reference.

$$u(t) \quad = \quad \sin(t) + 3\cos(\frac{1}{10}t) + \frac{1}{100}t^3$$

The parameters used for the noise free cases are for AEAD $\omega_c = 10$ rad/s and for RLPAD $\omega_0 = 50$ and $f_{red} = 2.2$. These gives a nice damped and fast response for RLPAD similar to the initial transient of AEAD. The plots are shown in Figure 5-7.

Figure 5-7 concludes the same as was the case for a polynomial input for AEAD in section 5-1. The derivatives diverge in (a-b) and thus AEAD cannot track a polynomial and therefore also not the additive sinusoid within it.

The proposed method can track a polynomial and sinusoid separately, therefore it is expected that it can track also a combination of the two. This is confirmed by Figure 5-7cd and Table 5-8. The errors are well below one for the zeroth up to the second derivative. However, RLPAD does start to have increased errors starting from the third order derivative, which is shown by the overshoots at the peaks of $y_t^{(3)}$ in Figure 5-7d. The fourth and fifth order estimates lag and have high overshoots resulting in large errors.

The differentiators are retuned for the case with additive noise ($\sigma = 0.001$) in Figure 5-4. AEAD has $\omega_c = 5$ rad/s and for RLPAD ω_0 is set to $\omega_0 = 40$. Their RMSE are logged in Table 5-6.

The conclusion about Figure 5-4 is similar to the noise free case. AEAD can't track the input, and it shows less noise attenuation compared to RLPAD. RLPAD on the other hand does follow the input and attenuates the noise. The output of $y^{(0)}$ has lower RMSE than the standard deviation, thus it shows that the differentiator can improve the noisy input. The noise gets amplified over each consecutive derivative estimation, but that is expected. It still manages to keep the RMSE of the estimates with noise fairly low in Table 5-9 compared to Table 5-8. The zeroth order is only increased by roughly a factor 10 which gets lower for the consecutive derivative errors.

Summarizing, AEAD cannot track the derivatives of an input that is a combination of a polynomial and sinusoid. The reason for this is that it can't track a polynomial input as was shown in section 5-1. RLPAD on the other can track this type of input and estimates derivative with low errors in the case with a without noise. RLPAD also improves the input by estimating the zeroth order derivative with a lower RMSE than it would have if the input was used with noise.

	RMSE polynomial + sinusoid input with time step $\Delta T = 0.01$ and $n = 10$					
Method	$e^{(0)}$	$e^{(1)}$	$e^{(2)}$	$e^{(3)}$	$e^{(4)}$	$e^{(5)}$
AEAD	$3.21 \cdot 10^{-2}$	$5.95 \cdot 10^{-2}$	$7.40 \cdot 10^{-1}$	6.48	$3.89 \cdot 10^{1}$	$1.62 \cdot 10^{2}$
RLPAD	$3.32 \cdot 10^{-5}$	$1.22 \cdot 10^{-3}$	$1.64 \cdot 10^{-2}$	$1.32 \cdot 10^{-1}$	$5.58 \cdot 10^{-1}$	1.14

Table 5-8: This table shows the root mean square errors of the derivative estimations with the true ones of the combination of the polynomial and sine input. The RMSE is recorded over the interval $t = [4.0, 20.0]$ seconds after the main transient. The i-th derivative error is denoted by $e^{(i)}$. The parameters used are $\omega_c = 5$ rad/s for AEAD, for RLPAD $\omega_0 = 50$ rad/s and $f_{red} = 2.2$. Both differentiators use $n = 10$ for the amount of estimated derivatives and their sample time is $h = 0.01$ seconds. The errors of AEAD increase rapidly, because it cannot track a function that consists of a polynomial signal. RLPAD can track this type and shows that the error is far below one for up to the second order derivative.

	RMSE Noisy ($\sigma = 0.001$) polynomial+sinusoid with time step $\Delta T = 0.01$ and $n = 10$					
Method	$e^{(0)}$	$e^{(1)}$	$e^{(2)}$	$e^{(3)}$	$e^{(4)}$	$e^{(5)}$
AEAD	$3.22 \cdot 10^{-2}$	$6.12 \cdot 10^{-2}$	$7.65 \cdot 10^{-1}$	6.71	$4.03 \cdot 10^{1}$	$1.68 \cdot 10^{2}$
RLPAD	$5.28 \cdot 10^{-4}$	$7.15 \cdot 10^{-3}$	$5.96 \cdot 10^{-2}$	$2.73 \cdot 10^{-1}$	$7.92 \cdot 10^{-1}$	1.25

Table 5-9: This table shows the root mean square errors of the derivative estimations with the true ones of the combination of the polynomial and sine input with additive noise ($\sigma = 0.001$). The RMSE is recorded over the interval $t = [4.0, 20.0]$ seconds after the main transient. The i-th derivative error is denoted by $e^{(i)}$. The parameters used are $\omega_c = 5$ rad/s for AEAD, for RLPAD $\omega_0 = 40$ rad/s and $f_{red} = 2.2$. Both differentiators use $n = 10$ for the amount of estimated derivatives and their sample time is $h = 0.01$ seconds. The error of AEAD increases rapidly because it cannot track a polynomial signal. RLPAD shows that it performs with low errors for up to the second order derivative. It also improves the estimate of the zeroth order derivative, because the RMSE of it is lower than the standard deviation of the noise ($e^{(0)} < \sigma$).

5-4 Summary

The performance of AEAD and the proposed method (RLPAD) are evaluated by using three type of input functions with known higher order derivatives. The inputs are a polynomial, a sinusoid and a combination of the two, see equation (5-1) to (5-3) respectively. The performance of these inputs conclude the following four points. The first is that AEAD cannot track a polynomial signal, because the derivative estimates diverge over time. This eventually leads to all estimates diverging including the zeroth order estimate, albeit a lot slower. The proposed method on the other hand can track a

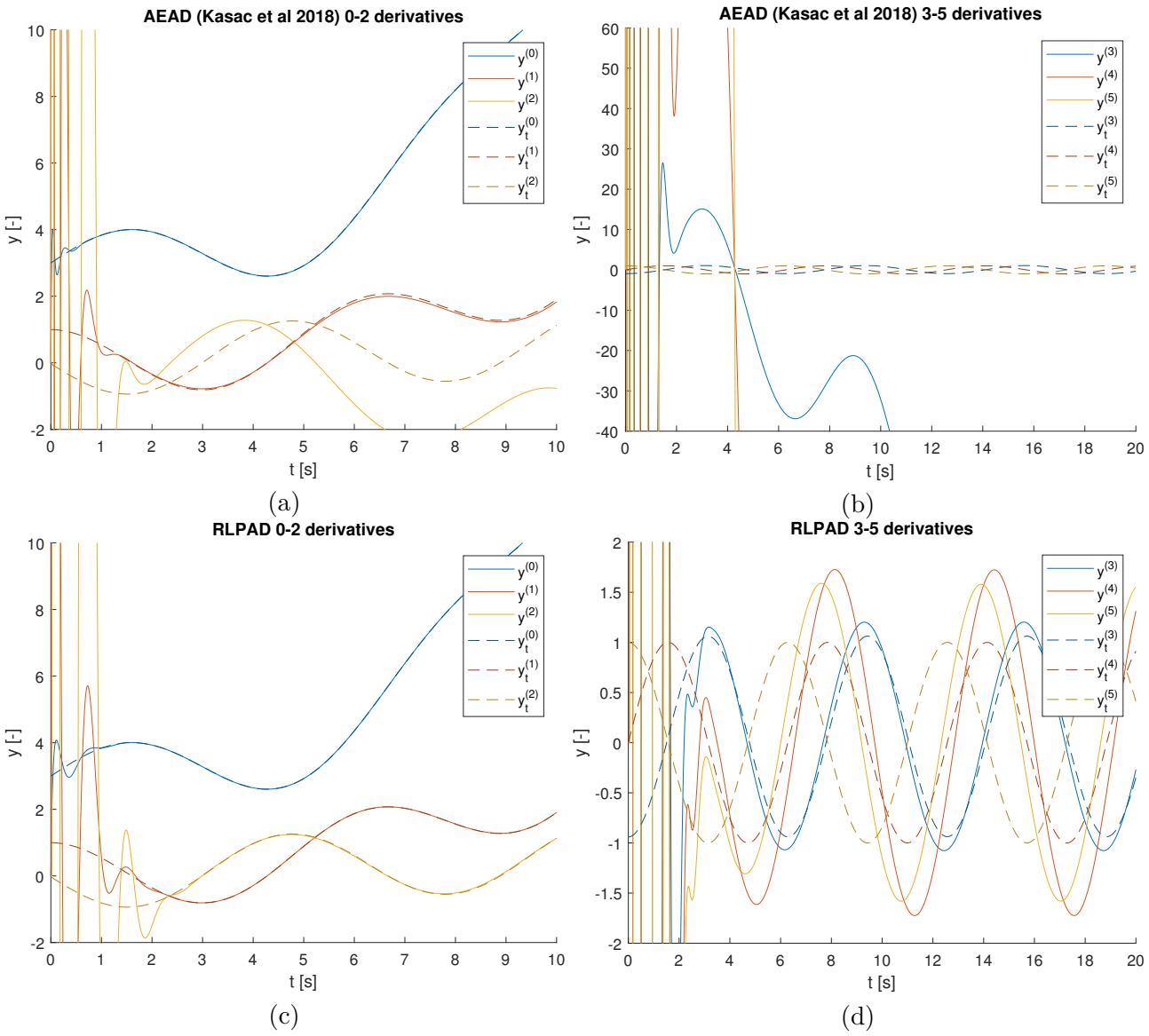


Figure 5-7: Plot of the trajectories of AEAD (a-b) and RLPAD (c-d) for a combination of a polynomial and sinusoidal input function (5-3) sampled at $h = 0.01$ seconds. The parameters used are $\omega_c = 10$ rad/s for AEAD, for RLPAD $\omega_0 = 50$ rad/s and $f_{red} = 2.2$. Both differentiators use $n = 10$ for the amount of estimated derivatives and their sample time is $h = 0.01$ seconds. The solid lines are the trajectories estimated and dashed lines are the true derivatives. The superscript $y^{(i)}$ denotes the i-th derivative. Since AEAD cannot track a polynomial signal, it is no surprise that it also cannot track a combination of it with a sine. The divergence is already getting out of hand for $y^{(2)}$. RLPAD on the other hand, can track this signal with low error. It tracks the zeroth to second derivative good, but struggles with the third and higher.

polynomial very accurately also in the presence of noise. It can track all derivatives of a polynomial when the amount of estimated derivatives is equal to the degree of the polynomial. And in the case with noise, RLPAD attenuates it properly for polynomial signals.

The second is that both differentiators can track a sinusoid input with and without noise. With the right tuning their performance is very similar for both the RMSE of the derivatives and transient responses. The proposed method has a better estimate for the zeroth order derivative and is therefore considered slightly better than AEAD.

The third point is that the proposed method also functions properly with an input that is a combination of a sinusoid and a polynomial, whereas AEAD cannot track this signal. The reason for that is that AEAD cannot track polynomial signals.

The fourth point is that the performance of both differentiators is highly affected by the time step,

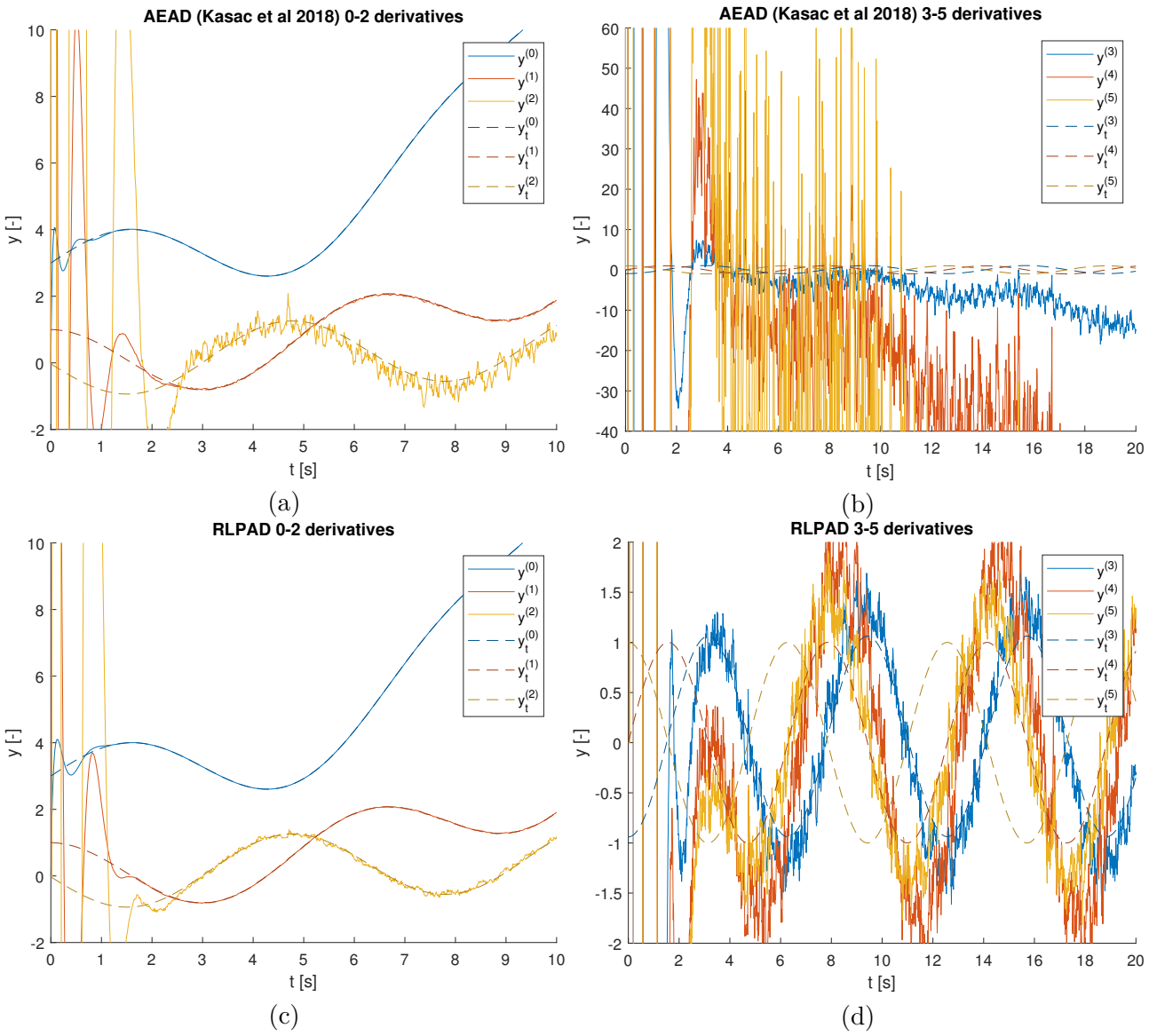


Figure 5-8: Plot of the trajectories of AEAD (a-b) and RLPAD (c-d) for a combination of a polynomial and sinusoidal input function (5-3) with additive noise ($\sigma = 0.001$) sampled at $h = 0.01$ seconds. The parameters used are $\omega_c = 5$ rad/s for AEAD, for RLPAD $\omega_0 = 40$ rad/s and $f_{red} = 2.2$. Both differentiators use $n = 10$ for the amount of estimated derivatives and their sample time is $h = 0.01$ seconds. The solid lines are the trajectories estimated and dashed lines are the true derivatives. The superscript $y^{(i)}$ denotes the i-th derivative. The conclusion is for AEAD the same, it still cannot track an input that consists of a polynomial. RLPAD can track the zeroth to second order derivative still pretty good with noise present. It also tracks the trend of the third derivative, but it overshoots slightly. Higher order derivatives start to lag behind.

the amount of derivatives estimated and the noise that is present on the input.

Chapter 6

Real data experiment

The previous chapter evaluates the performance of the recurrent low pass algebraic differentiator (RLPAD) and the algebraic estimation approach differentiator (AEAD) for nice mathematical inputs. It is also interesting to see how well the differentiators perform on real sensor data. This chapter evaluates the RLPAD and AEAD when real sensor data is used as input. The sensor data that is used is recorded by an OptiTrack system [1] and is made available by D. Benders. For in depth information about the data acquisition setup and experiments conducted, see his Master's thesis [6]. The data consist of flight data from a Parrot AR.Drone 2.0 (a quad-copter) [2] that flies stationary while it is affected by wind from a fan, which is described in more detail in section 6-1. Section 6-2 covers offline smoothing of the raw data so that a ground truth of the higher order derivatives can be determined. The performance of AEAD and RLPAD can then be evaluated, which is covered in section 6-3. Finally, the findings of the real data experiment are summarized in section 6-4.

6-1 Experimental setup for data acquisition

The setup that was used to get sensor data is shown in Figure 6-1 and is as follows. The lab room is rigged with 10 OptiTrack system cameras [1]. The drone is placed in the middle of the lab, with its axis and center of mass aligned with the calibrated ones of the OptiTrack system. The calibrated ("world") axes are as follows. The y-axis points towards the fan, x-axis points to the right when facing the fan and the z-axis point up. The drone has reflective markers on it to allow it to be tracked by the OptiTrack system, they are placed in such a way so that the center of mass of the markers align with the center of mass of the drone. The Robot Operating System (ROS) is used for the data acquisition of the OptiTrack system and the drone. ROS is also used to control the drone, which commands the drone to maintain a steady position during flight.

A wind source is used to disturb the drone, this means that the unmodeled noise of the drone has smooth characteristics. To create this wind, a fan is placed 6 meters in front of the drone. The fan is initially turned off while the drone takes off and maintains a steady flight. After 10 seconds of liftoff the fan is turned on and set to a low wind mode. After 45 seconds of liftoff, the fan is set to a high wind mode and creates a stronger wind stream.

The OptiTrack systems records positions and rotations very accurately with its proprietary software and has a very high signal to noise ratio. The drone can pitch, yaw and roll, but for the differentiator experiment the only input used is the "world" x-position recorded by the OptiTrack system. The world positions can be transformed to body coordinates of the drone, but this is not necessary for derivative estimation as a differentiator should be able to handle any type signal as they are model free. Therefore it is decided to use the "world" x-position recorded by the OptiTrack system as input for the comparison of the two differentiators.

The raw data and the time line of events when the fan is turned on are shown in Figure 6-2. Notice that the drone does not respond instantly to the wind produced by the fan at the 10 seconds mark. The winds takes some time until it reaches the drone six meters away. The transition point at $t = 26$ seconds denotes the point where the wind effects the drone. This can be seen by looking at the relatively low amplitude oscillations before the transition point and higher and faster oscillations after. When the fan is at high speed at $t = 45$ seconds, the drone starts to drift away from its set point.

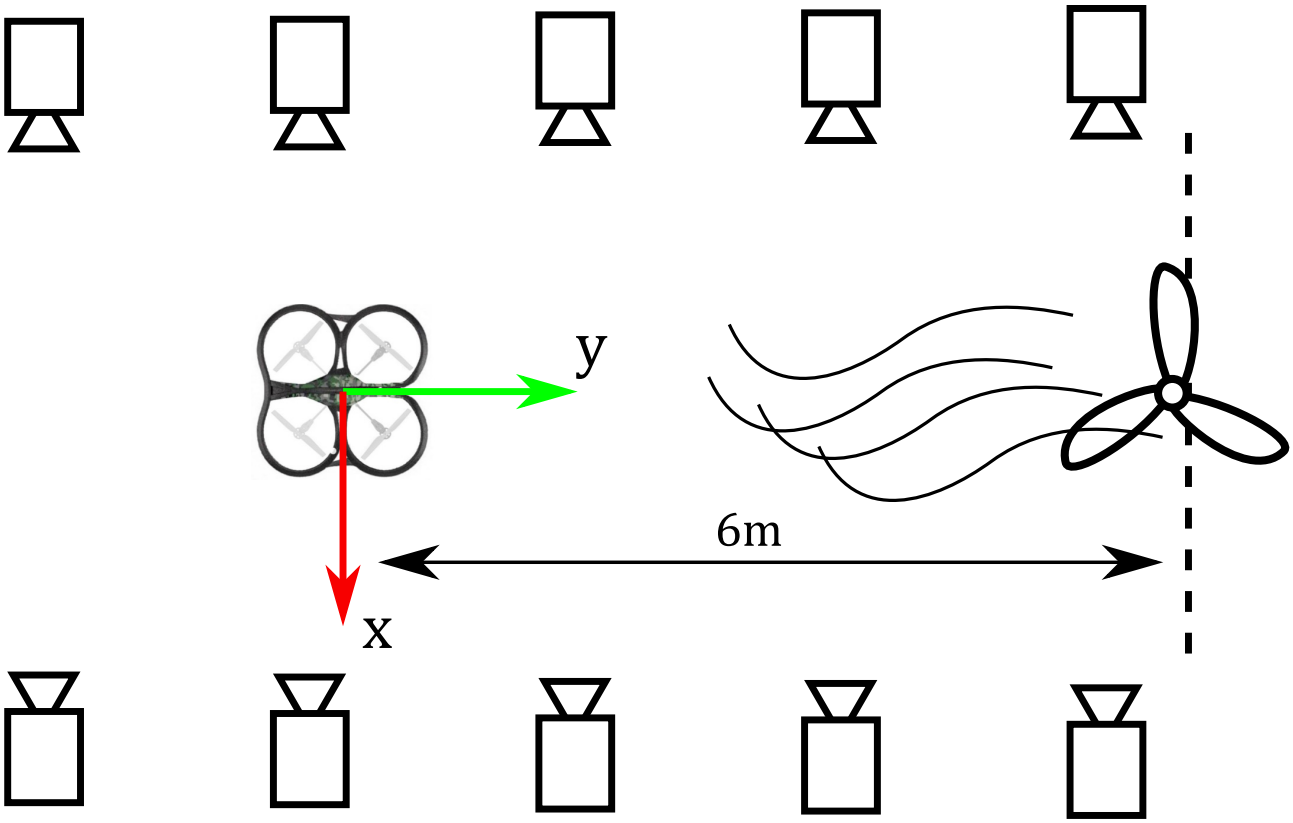


Figure 6-1: Schematic of the experiment setup. The drone is placed and aligned with the origin of the calibrated OptiTrack system. The wind source is placed about 6 meters in front of the drone. The drone its position and attitude is recorded by 10 OptriTrack cameras at 120 Hz. The image of the drone is copied from [3].

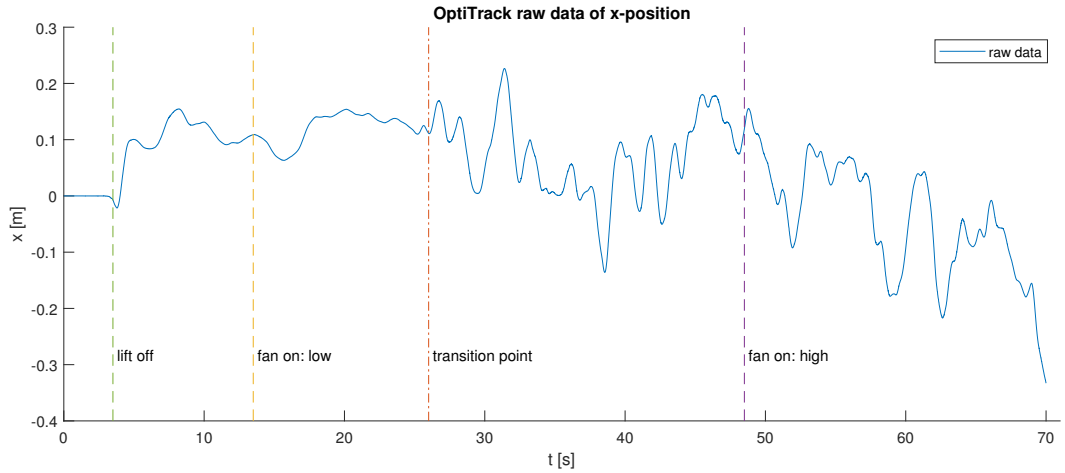
6-2 Experiment data analysis and post processing

A ground truth of the higher order derivatives of the raw data are necessary to compare the proposed method (RLPAD) with the state of the art differentiator (AEAD). There are no sensors that measured the derivatives, therefore they must be determined offline.

The OptiTrack system has a very low standard deviation of $\sigma = 6.0 \cdot 10^{-5}$ [m] for the first 3 seconds when the drone is not moving (see Figure 6-3a). Because the noise is so low, one could think that the derivatives would be accurate if a finite difference scheme is used. This is not true, even though the standard deviation of the noise is very low, the estimated derivatives will be amplified too much, even by this noise (see the saw tooth shapes in Figure 6-3b). Therefore the data has to be smoothed to create a reasonable ground truth for the higher order derivatives.

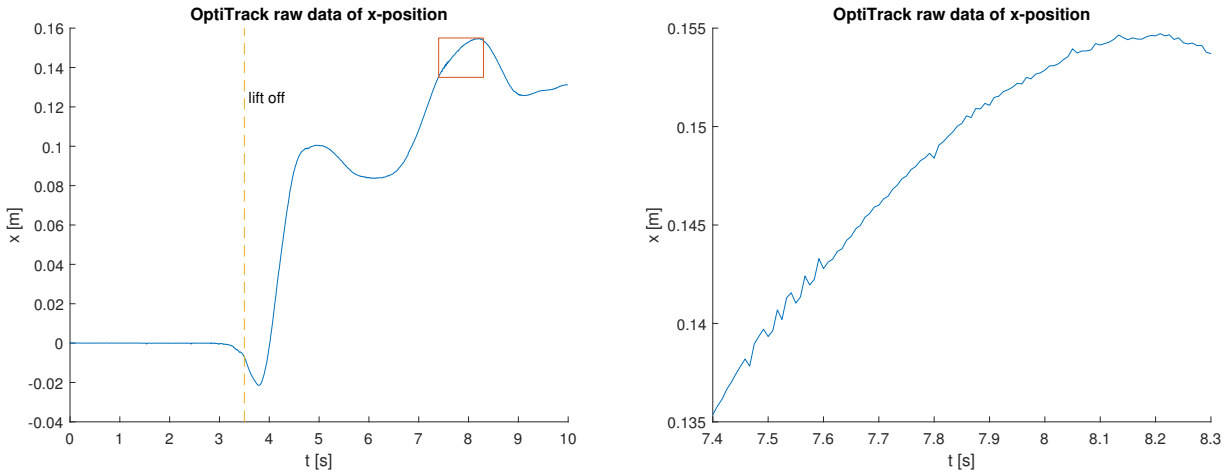
There are numerous methods that can smooth data, such as a moving average, exponential smoothing [45] and a Gaussian filter (also known as Gaussian blur) to name a few. Each have their downsides, but the main disadvantage of these is that they cannot determine the derivatives of the smoothed data directly from their formulation. A smoothing filter that can smooth and estimate the derivatives at the same time, is a Savitzky-Golay filter [48]. It can do this, because the smoothing is based on a n-th

degree local polynomial that is fitted through data points around a central point by a given window size in a least-squares sense. The derivatives are then estimated by taking the derivative of this n-th degree polynomial up to $n-1$ derivatives. This type of filter is a form of local polynomial regression or sometimes called a local polynomial smoothing filter [49]. The Savitzky-Golay filter is used in the next section to smooth the raw data.



(a)

Figure 6-2: Plot of the raw sensor data acquired by the OptiTrack system at 120Hz. The full data set is plotted over 70 seconds. The vertical dashed lines indicated the events of the experiment. Lift off happens at $t = 3.5$ seconds. The fan is turned on 10 seconds after lift off and set to low at $t = 13.5$ seconds. The fan is set to high at $t = 48.5$ seconds. The transition point, indicated by the dashed dot line is when the drone is clearly affected by the wind disturbance. The wind source does not disturb the drone instantly after it is turned on, because the wind needs time to travel a distance of about 6 meters first.



(a) (b)

Figure 6-3: Plot of the raw sensor data acquired by the OptiTrack system at 120Hz. The data of Figure 6-2 is zoomed in to a range of $t = 0$ to $t = 10$ seconds in (a). The box indicated the close up region of (b). It shows clean data in (a), however, viewing the data from that resolution is deceiving. Because a close up in (b) shows that there are relatively high saw tooths present in the data. These are the reason why backwards differentiation will not give accurate results.

6-2-1 Offline smoothing and derivative estimation for a ground truth

The Savitzky-Golay filter [48] requires two parameters to be tuned for smoothing the data properly. The first parameter is the degree of the polynomial used for fitting and the second parameter deter-

mines the window size around a central point. These parameters have to be set properly for the data to still have meaningful information.

There are two important factors to keep in mind when using this filter. The first is that the window size determines how many samples are considered. This essentially functions as a low pass filter when the window size becomes larger. This means that the smoothed output starts to lose information by increasing the window size. The second is that the amount of derivatives that can be extracted depends on the polynomial degree minus one. Furthermore increasing the polynomial degree does not result in better smoothed data, especially when a constant window size is used [33].

The goal of this filter is to smooth the raw data in such a way so that the sensor noise is reduced significantly, while still following the trend of the data points. A helpful tool for finding good parameters is by making use of the residual noise of the raw data. It is defined as

$$\eta_r = x_{smooth} - x_{raw}. \tag{6-1}$$

This allows for calculating the mean, standard deviation and autocorrelation. The standard deviation and autocorrelation are important for determining the proper tuning parameters. The standard deviation is important, because for properly smoothed data it is expected to have a standard deviation that is slightly greater or equal to the standard deviation of the stationary case (between $t = [0, 3]$ in Fig 6-3). The autocorrelation is important, because it can determine the whiteness of the residual noise. The parameters of the Savitzky-Golay filter determine the autocorrelation plot. When the filter does not smooth the raw data as much, it will show correlation with itself and when the filter smooths too much it will show the same behavior. There is a set of parameters that smooth the raw data in an uncorrelated way and at that point the residual noise can be considered uncorrelated white noise.

The sensor noise of the OptiTrack system is considered white in this thesis. Therefore the parameters of the Savitzky-Golay filter can be tuned in such a way so that the autocorrelation plot of the residual noise is as uncorrelated as possible. This is achieved by the following steps.

1. Set a desired polynomial degree and keep it constant. The amount of derivatives that can be extracted is the degree minus one.
2. Set an odd window size that is greater than the polynomial degree.
3. Smooth the raw data with the Savitzky-Golay filter [48]
4. Calculate the residual noise by subtracting the raw data from the smoothed data.
5. Calculate the mean, standard deviation and autocorrelation of the residual noise.
6. Evaluate the mean, it should be close to zero.
7. Evaluate the standard deviation, it should be only a few factors greater than the stationery noise.
8. Evaluate the autocorrelation function, it should be highly uncorrelated.
9. If the evaluation is not sufficient enough, go back to step 2 and repeat.

These steps were evaluated manually, while having a constant polynomial degree of 4. This gave the best results for higher order derivatives. A higher degree polynomial resulted in rapid oscillations of derivatives and could not get the data smoothed based on the conditions stated above. This could be solved by multi filtering the smoothed data, but this is not considered in this thesis. A degree of 4 allows for the extraction of up to the third order derivative. This is considered sufficient for differentiator testing purposes, because they show very large errors when they estimate derivatives higher than the second order when noise is present on the signal.

By following the steps above, the best window size found is 31 samples. This gave the following statistical properties of the residual noise. The mean is $\mu_r = 2.8 \cdot 10^{-8}$ and the standard deviation is $\sigma_r = 2.1 \cdot 10^{-4}$. The mean shows that it is very close to zero, this indicates together with the standard deviation that it follows the raw data accurately. The standard deviation is only a factor $\frac{2.1 \cdot 10^{-4}}{6.0 \cdot 10^{-5}} = 3.5$ larger than the standard deviation of the stationary noise. This is considered good, because one can expect that the standard deviation of the noise is larger during flight. Factors like vibrations and velocity of the drone can affect the accuracy of the OptiTrack system. The decisive factor of how good the noisy data is filtered is shown by the autocorrelation plot in Figure 6-5b. It shows that the residual noise is highly uncorrelated, when the raw data is smoothed with the parameters: a polynomial degree of 4 and a window size of 31. Note that instead of manually finding the optimal windows size, [47] proposes a recursive method that can find the optimal window size based on the given polynomial degree and known noise properties of the signal. However, this method was not considered due to time constraints and it is not known how optimal the window size is when the variance of the noise varies over time.

The result of smoothing the raw data by a Savitzky-Golay filter [48] is shown in Figure 6-4. It shows the smoothed signal and its first derivative in (a). The derivative looks smooth, although it still has some small bumps of short duration. Figure 6-4b shows the same close up of Figure 6-3b, but has the smoothed signal plotted over it. It shows that the saw tooths of the raw data are effectively filtered, resulting in smoother derivatives.

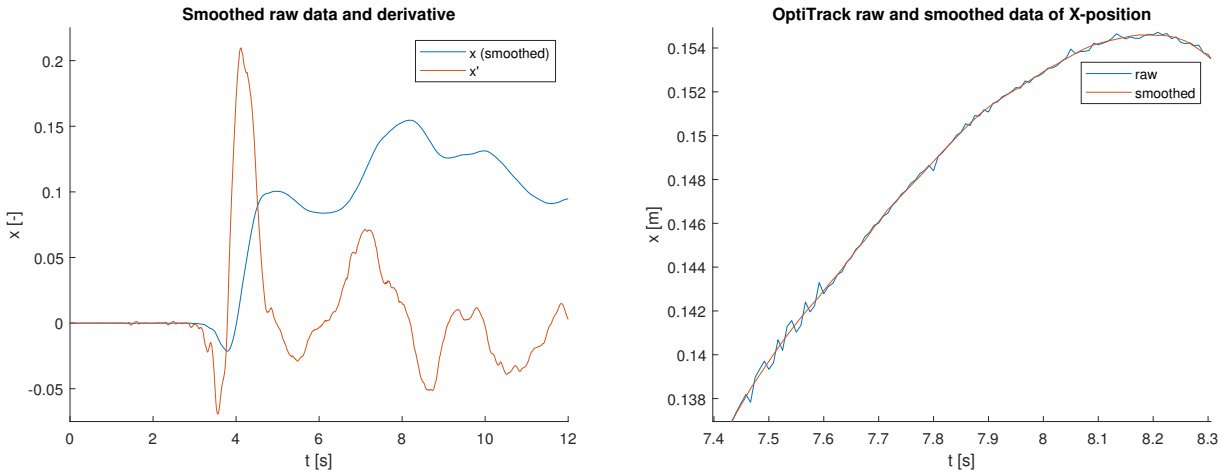
An interesting fact about the residual noise in Figure 6-5a is that it has a non constant standard deviation. The noise clearly has higher amplitudes past the transition point at $t = 26$ seconds and thus has higher standard deviations. This varies over time, because the interval of $t = [50, 60]$ shows that the noise has lower amplitudes, resulting in a lower standard deviation locally. This change of the noise properties results in a non Gaussian distributed normalized histogram, which can be seen in Figure 6-6. This figure does not show all the histogram bins, because there are several outliers. To show that this distribution is non Gaussian, three pure Gaussian probability density functions are plotted over it. The reason why this distribution is non Gaussian, is because the standard deviation of the noise residual ($\sigma = 2.14 \cdot 10^{-4}$) does not line up with the peaks of the histogram plot at all. The main reason for this is that the standard deviation varies over the whole data set and there are relatively many outliers. The Gaussian distribution of the stationary standard deviation is also plotted for reference, and also this does not match the peaks of the histogram. A better Gaussian distribution fit for this histogram is when the standard deviation is $\sigma = 9.0 \cdot 10^{-5}$. The change of variance is actually a beneficial aspect in the comparison of the differentiators, because it also tests the capability of how they handle variable noise.

Summarizing, the raw data is smoothed by a Savitzky-Golay filter [48]. The parameters used for it are a 4th degree polynomial and a window size of 31 samples. The 4th degree polynomial allows for the extraction up to the third order derivative. Although the filter can extract more derivatives, the degree of 4 is used, because the noise affects higher derivative too much. The smoothed data and its corresponding derivatives are the ground truth for the comparison between the proposed method and AEAD in the next section.

6-3 AEAD vs RLPAD

The experiment is as follows. The input to the differentiators is the smoothed raw sensor data that is created by the Savitzky-Golay filter [48]. The ground truths for the derivatives are also determined by this filter, up to the third order for the reasons stated in the previous section. The performance of both differentiators is evaluated based on the RMSE between the outputs and ground truths over a specified time interval.

The ground truths determine the tuning of both differentiators. They are manually tuned to get the lowest root mean squared error for the zero-th and first order derivative. Initially the following parameters are used:



(a) (b)

Figure 6-4: The smoothed raw data and the derivative is calculated by using the Savitzky-Golay filter (a). A close up of the raw and smoothed data is plotted in (b). The data is now cleaner as it filtered out the saw tooths in the close up in (b). The derivative is smooth but still has rapid dips and bumps along its signal. This derivative is used as ground truth for the differentiator experiment.

■ **Parameter set A**

- AEAD: $a = 12$ and $n = 10$.
- RLPAD: $\omega_0 = 120$, $f_{red} = 3.0$ and $n = 10$.

For this set of parameters n is set to 10, this means that the differentiators will estimate up to 9 derivatives plus the zeroth. However, from Figure 6-7 it can be seen that the performance of the RLPAD does not necessarily get better by estimating more derivatives. It shows that it has the lowest RMSE at $n = 3$ for the zeroth derivative. The same is true for AEAD, its higher order estimations get worse for an input with a high noise variance and those influence the lower order estimations. This means that adding more derivative estimations does not always improve accuracy. For aforementioned reasons, both differentiators are also retuned and tested with the following parameters.

■ **Parameter set B**

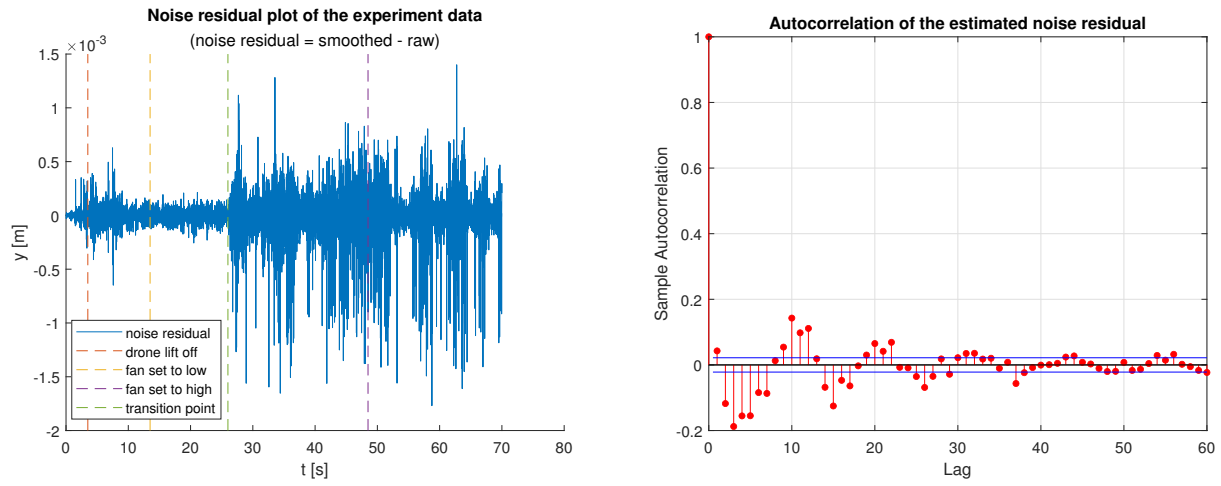
- AEAD: $a = 12$ and $n = 3$.
- RLPAD: $\omega_0 = 100$, $f_{red} = 1.8$ and $n = 3$.

The AEAD is converted to a discrete state space with MATLAB's `c2d` function with a sample rate of $T_s = 1/120$. This function converts it with the highest accuracy possible. Both differentiators are evaluated for the parameter set A and B in the following two sections.

6-3-1 Performance of parameter set A ($n = 10$)

The performance of the proposed method, RLPAD, is compared with AEAD for parameter set A and the results are as follows. AEAD performs better at estimating the first derivative, but performs similar as the proposed method for the rest of the derivatives. However, the proposed method damps higher order derivatives better. Generally speaking, both differentiators struggle to estimate the 3rd order derivative due to high errors.

Table 6-1 shows that this is the case. AEAD performs about 2.9× better at estimating the first derivative. They perform very similar for the zeroth and second order derivative. The errors increase



(a) (b)

Figure 6-5: The residual noise is plotted in (a) and its autocorrelation in (b). The residual noise plot also shows two distinct phases. The noise has lower amplitudes at the first 25 seconds. The autocorrelation plot shows that the samples are highly uncorrelated with past samples over the full duration of 70 seconds.

rapidly for the third order derivative, but RLPAD keeps it lower than AEAD.

	RMSE of the full data set and $n = 10$ at 120Hz			
Method	$e^{(0)}$	$e^{(1)}$	$e^{(2)}$	$e^{(3)}$
AEAD	$5.50 \cdot 10^{-4}$	$1.37 \cdot 10^{-2}$	0.68	19.3
RLPAD	$5.61 \cdot 10^{-4}$	$4.0 \cdot 10^{-2}$	0.69	5.65

Table 6-1: This RMSE is recorded after the main transient response in the interval of $t = [4.0, 70.0]$[s]. Both differentiators perform similar for the zeroth and second derivative when $n = 10$. However, AEAD is 2.9 times better at estimating the first derivative. RLPAD has the advantage that it does not let a higher derivative explode as fast which can be seen at $e^{(3)}$. It supresses the noise (and signal) of higher derivatives a lot more.

The performance is visualized in Figure 6-8, which shows the zeroth, first and second derivative plot in the interval $t = [30, 40]$. It shows clearly that AEAD (a) has lower errors for the first derivative compared to RLPAD in (b). This is shown more clearly in Figure 6-9. RLPAD shows a smoother estimation of the second derivative than AEAD. It overshoots less, but it is slower and lags somewhat compared to (c). The reason for this is that RLPAD has a stronger low pass filter for each consecutive higher order derivative that is estimated.

6-3-2 Performance of parameter set B ($n = 3$)

In this section the proposed method, RLPAD, is compared with AEAD for parameter set B. This means that for this parameter set the maximum amount of derivatives estimated is up to the second order. In contrast to the previous section, RLPAD outperforms AEAD significantly for all derivatives estimated. RLPAD estimates the zeroth order derivative a factor 3.5 better than AEAD and the first derivative by a factor of 2.1.

Table 6-2 shows that this is the case. RLPAD performs about 2.1× better at estimating the first derivative and it is coincidently 2.1 times better than the estimate in Table 6-1. However, the errors get closer to each other for the second derivative, but RLPAD is still the better one as it can follow the trend of the second derivative.

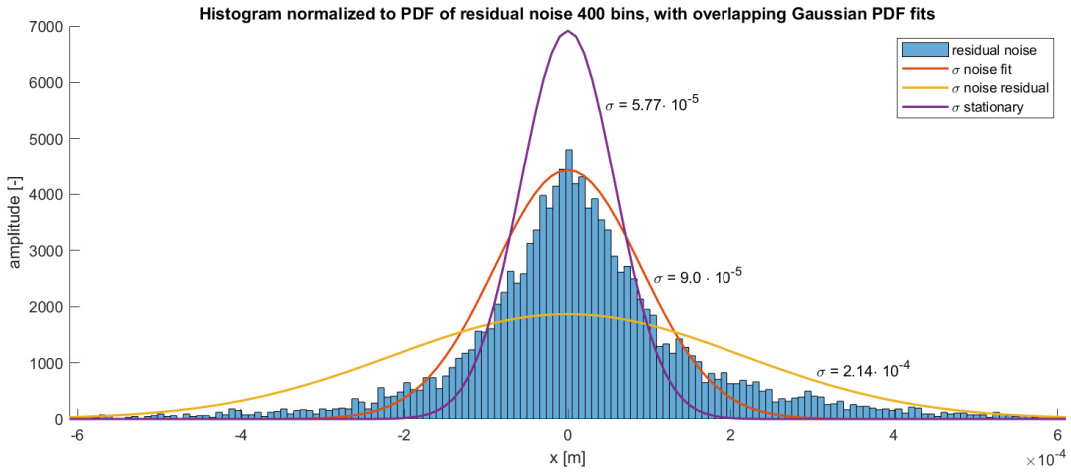


Figure 6-6: This histogram shows that the residual noise does not follow a Gaussian normal distribution and therefore the noise is not Gaussian distributed (globally). Figure 6-5a shows that the variation changes a lot over time, so locally the noise can be, and likely is, Gaussian distributed. This results in that the total distribution of the residual noise is a sum of several Gaussian distributed probability density functions. The yellow bell curve is the shape that the bins of the residual noise should have if the noise was Gaussian with a constant variance.

	RMSE of the full data set and $n = 3$ at 120Hz		
Method	$e^{(0)}$	$e^{(1)}$	$e^{(2)}$
AEAD	$1.09 \cdot 10^{-3}$	$4.11 \cdot 10^{-2}$	$5.26 \cdot 10^{-1}$
RLPAD	$3.08 \cdot 10^{-4}$	$1.95 \cdot 10^{-2}$	$4.29 \cdot 10^{-1}$

Table 6-2: This RMSE is recorded after the main transient response in the interval of $t = [4.0, 70.0]$[s] with parameter set B. In contrast to the case where $n = 10$, RLPAD outperforms AEAD for all estimated derivatives. RLPAD performs $3.5\times$ better at estimating the zeroth derivative and $2.1\times$ for the first derivative estimate.

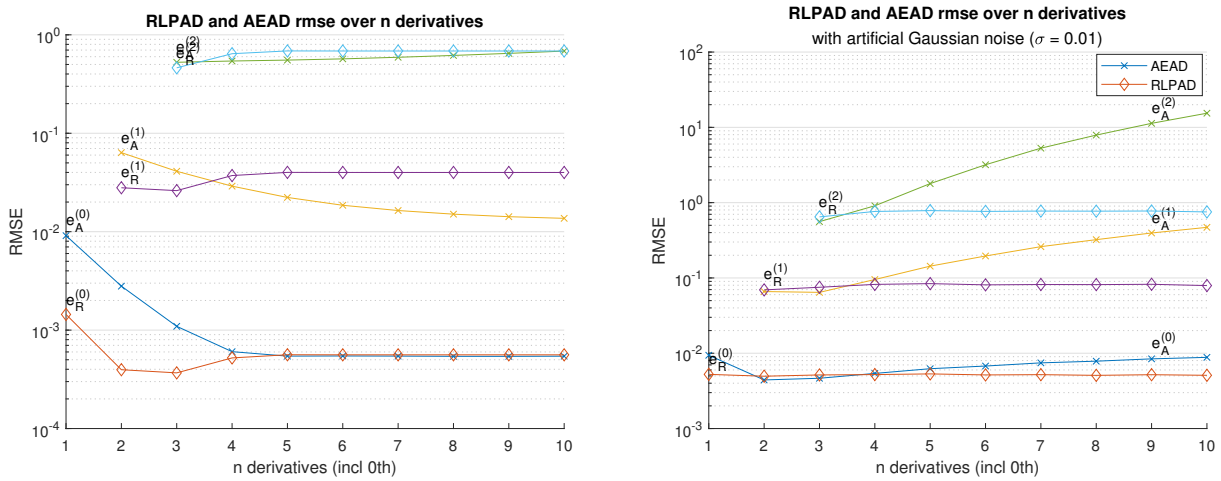
The performance is visualized in Figure 6-10, which shows the zeroth, first and second derivative plot in the interval $t = [30, 40]$. It shows clearly that AEAD (a) has higher errors for the first derivative compared to RLPAD in (b). It follows the trend but it lags behind the true signal. This is shown more clearly in Figure 6-11. AEAD fails to track the second derivative (c), it shows that it is to slow to follow the trend properly and lags even more than the first derivative estimate. RLPAD on the other hand follows the trend very good, but it also suffers from a small lag that is induced by the low pass filter characteristics.

6-3-3 Performance of parameter set B ($n = 3$) on raw data

The previous two subsections compared the performance with the smoothed data as input and ground truth. It is also interesting to see how the differentiators perform with raw data as input, but now with the smoothed data as ground truth. This way, the backwards differentiation scheme can also be compared. The reason for comparing a backwards finite difference scheme is that it is used in current implementations of Active Inference for finding generalized measurements [7].

A finite difference scheme was not compared in the previous subsections, because it will outperform the differentiators when the smoothed data is used as input. There is no noise on the input, and locally it is based on a polynomial, this means that a backwards differentiator scheme can estimate the first derivative a lot better than the differentiators can. The differentiators are designed to work with noisy inputs. Therefore it is more realistic to compare a finite difference scheme on a noisy input.

The differentiators use parameter set B and the results are as follows. Table 6-3 shows that the



(a) (b)

Figure 6-7: This figure shows the RMSE over $n = 1$ to 10 of the smoothed real sensor data for the parameters of parameter set A in (a). The errors are indicated by $e^{(i)}$, where i denotes the i-th derivative. The diamonds and crosses indicate the RMSE of RLPAD and AEAD respectively. This figure clearly indicates that RLPAD has a lowest error for all estimates at $n = 3$. The lowest error for AEAD is at $n = 5$ and does not change much over more states. In contrast, the first order derivative estimation decreases until $n = 10$. Plot (b) shows that this figure is highly affected by noise. The smoothed input is corrupted with a Gaussian noise with a standard deviation of 0.01. It shows that the RMSE of AEAD is highly affected by the noise. The more derivatives are added the worse all estimates become. In contrast, RLPAD is not sensitive to that, but this does not necessarily mean that the estimations are accurate. The reason for this is that RLPAD suppresses the higher order derivatives so much that they eventually go towards zero. Note that RLPAD was retuned in plot (b) to give a better plot ($\omega_0 = 60.0$).

backwards finite difference scheme (BFD) outperforms the differentiators for the 0th derivative. This is somewhat misleading, because the backwards differentiators 0th derivative is just the raw input. This means that the differentiators fail to filter the noise of this input so that the zeroth derivative is more accurate. Improving the signal is hard for differentiators when the raw data is very accurate already.

An important result is that RLPAD estimates the first order derivative better than BFD. BFD outperforms AEAD, because the full data set has a standard deviation of $2.11 \cdot 10^{-4}$. If the standard deviation is higher, the derivative estimations of BFD gets worse rapidly. This can already be seen for the second order derivative estimation which is already 17 times higher than RLPAD.

Figure 6-12 shows the plots of the derivative estimation in the interval $t = [30, 40]$. The conclusion for RLPAD and AEAD is the same as Figure 6-10. BFD shows that it is very noisy, but it follows the trend of the velocity good on average. This is due to the fact that the raw input its noise has a very low standard deviation. Although BFD works for the first derivative in this case, it will not for higher orders because the noise will be amplified for each additional derivative. Resulting in extremely high errors.

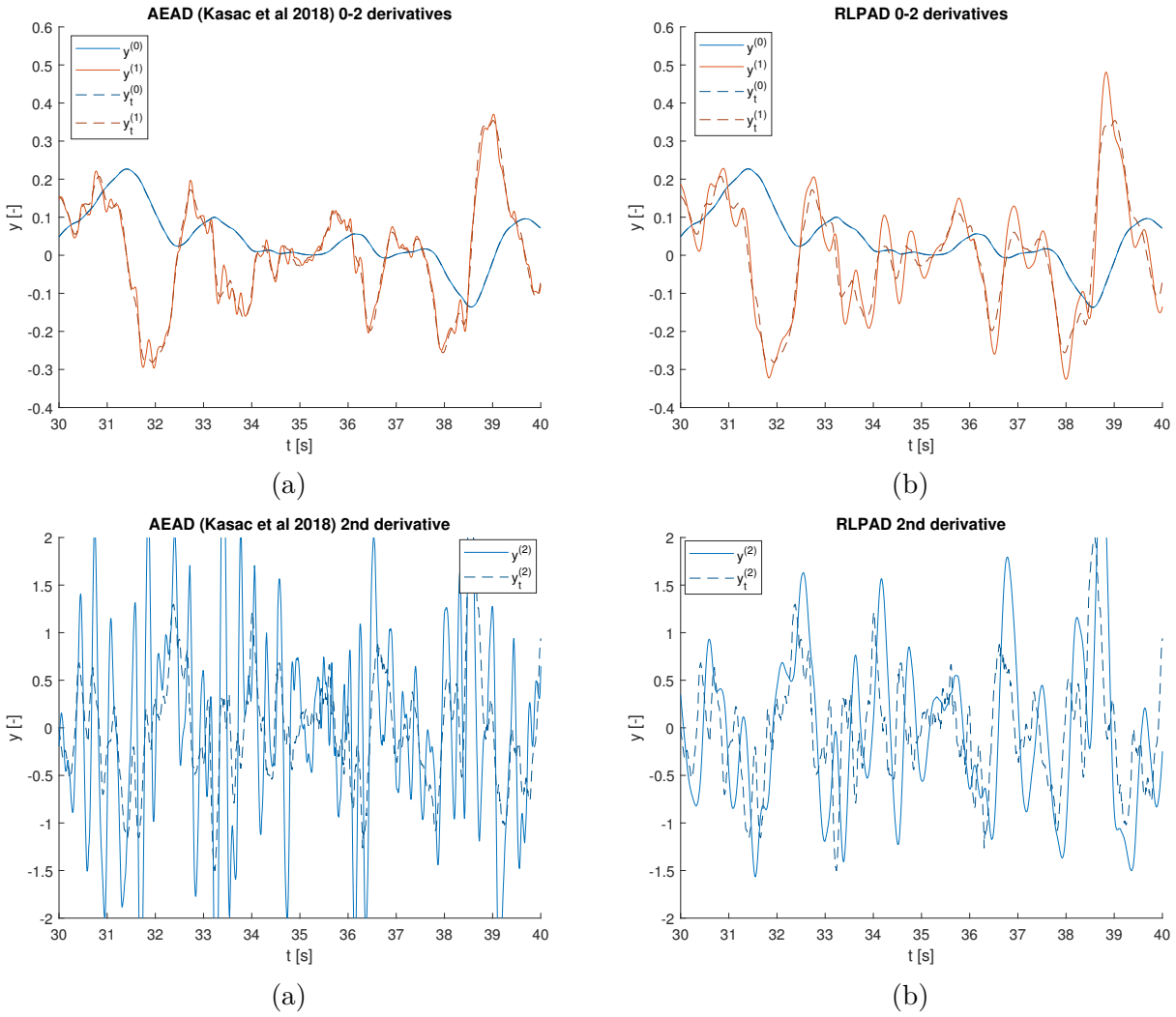
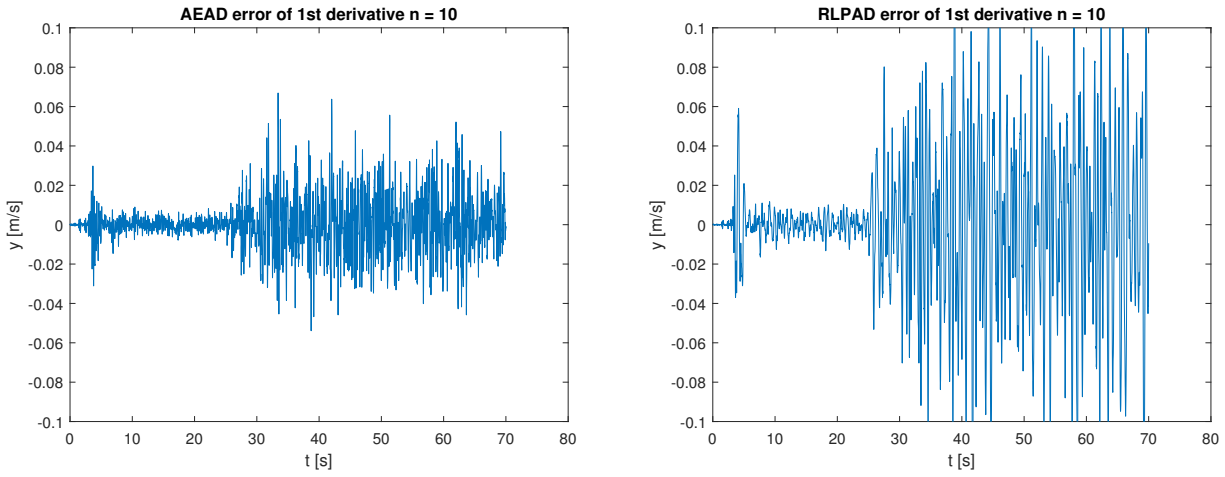


Figure 6-8: The figures show the tracking performance of the zeroth, first and second derivative for $n = 10$ in the interval $t = [30, 40]$. The solid lines are the estimations and the dashed lines are the true values. It shows that the AEAD (a) has frequent oscillations in the estimations. This is amplified a lot for the second derivative estimation in (c). RLPAD (b) has a smoother first derivative estimation, but overshoots more as can be clearly seen in the highest peak around $t = 39$. The estimated second derivative in (d) is therefore also more smooth than (c). It overshoots relatively less than (c) and lags behind the true signal, but it follows the general trend.

	RMSE of the full data set and $n = 3$ at 120Hz		
Method	$e^{(0)}$	$e^{(1)}$	$e^{(2)}$
BFD	$2.19 \cdot 10^{-4}$	$3.65 \cdot 10^{-2}$	7.35
AEAD	$1.09 \cdot 10^{-3}$	$4.11 \cdot 10^{-2}$	$5.26 \cdot 10^{-1}$
RLPAD	$3.41 \cdot 10^{-4}$	$2.02 \cdot 10^{-2}$	$4.43 \cdot 10^{-1}$

Table 6-3: This RMSE is recorded after the main transient response in the interval of $t = [4.0, 70.0]$[s] with parameter set B. In contrast to the case where $n = 10$, RLPAD outperform AEAD in all estimated derivatives. RLPAD performs $3.5\times$ better at estimating the zeroth derivative and $2.1\times$ for the first derivative estimate. BFD shows the lowest error for the zeroth derivative, and this error is equal to the standard deviation of the raw data with respect to the smoothed data. This means that the other differentiators struggle with low standard deviations as they could not improve the error of the zeroth derivative.



(a) (b)

Figure 6-9: The figures show the error of the first derivative of the smoothed data set. It shows that the AEAD (a) estimates the velocity better than RLPAD (b) when $n = 10$. It is also interesting to see that the noise seems more smooth in (b).

6-4 Summary

In this chapter the proposed method (RLPAD) is compared with AEAD by using real sensor data as input to the differentiators. The real sensor data is not usable as input directly, because it does not have a ground truth about its derivatives. Therefore the sensor data is smoothed by a Savitzky-Golay filter [48]. This filter smooths and estimates the derivatives of the input by fitting a polynomial through the data points.

The smoothed data is used as input to the differentiators and they are compared with two sets of parameters. The reason for two parameter sets is that the performance of the differentiators are highly affected by the amount of derivatives they estimate. The first parameter set estimates 9 derivatives (parameter set A) and the second set 2 derivatives (parameter set B). When the differentiators use parameter set A, the AEAD and RLPAD perform similar except for estimating the first derivative. AEAD estimates it 2.9 times better than RLPAD.

On the other hand, when parameter set B is used, RLPAD outperforms AEAD on every aspect. It estimates the zeroth derivative better by a factor of 3.5 and the first derivative by a factor of 2.1. Its estimates are better than when RLPAD uses parameter set A. It also shows that a backward finite difference scheme is bad for finding higher order derivatives.

A final insight is that the performance of the differentiators are highly affected by noise. AEAD shows that this is the case when the noise variance gets higher. Instead of improving the derivative estimates by increasing n, they become worse instead. However, RLPAD is not sensitive to this effect when adding more derivative terms, because it has a stronger noise attenuation for estimating the higher order derivatives. But this does not mean that the derivative estimates are accurate.

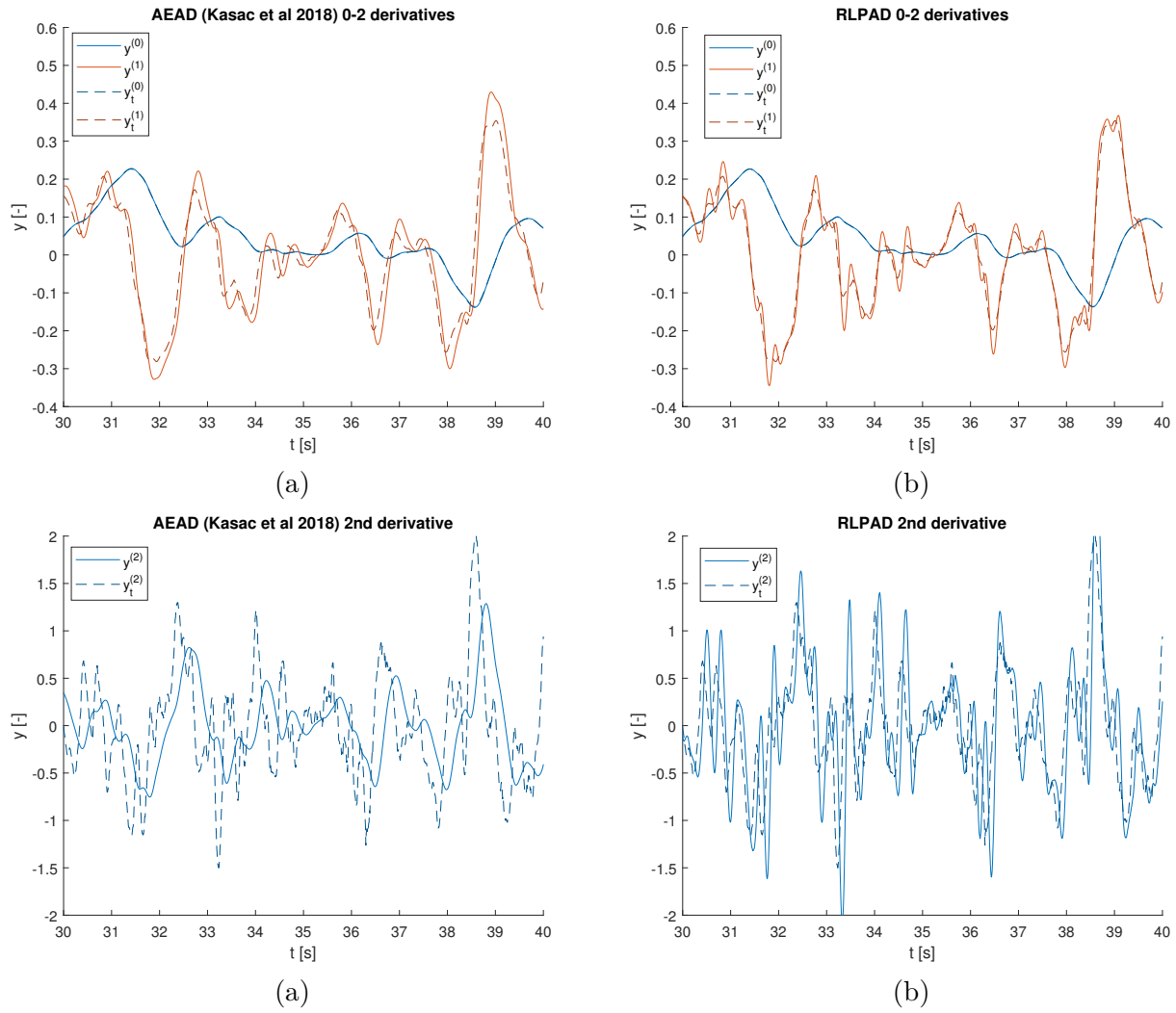


Figure 6-10: The figures show the tracking performance of the zeroth, first and second derivative for $n = 3$ in the interval $t = [30, 40]$. The solid lines are the estimations and the dashed lines are the true values. It shows that the AEAD (a) has lag the estimations of the first derivative, but it follows the trend. In (c) it shows that the estimated second derivative lags more and shows that it is too slow to track the trend of it. On the other hand RLPAD (b and d) can track the velocity without lag and follows the trend of the second derivative with some lag.

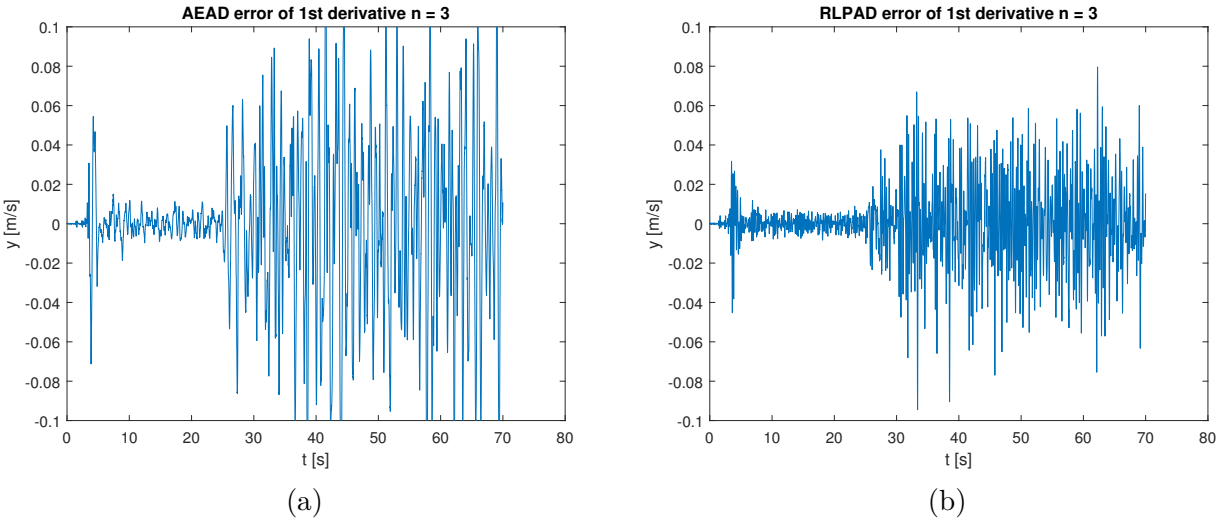


Figure 6-11: The figures show the error of the first derivative of the smoothed data set. It shows the opposite of Figure 6-9. AEAD (a) estimates the velocity worse than RLPAD (b) when $n = 3$. It now shows that AEAD seems to be more smooth, because of the larger gaps between the lines.

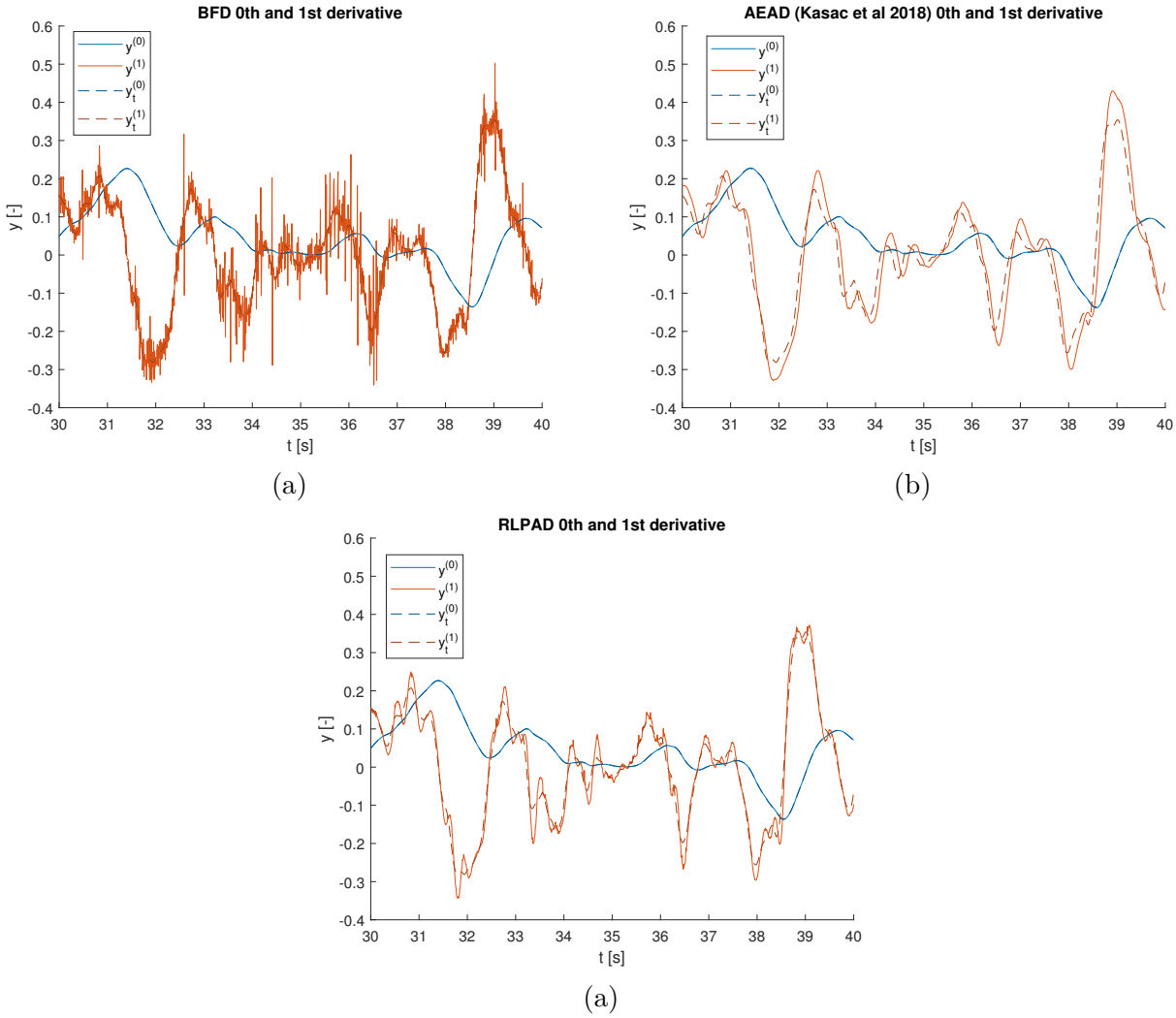


Figure 6-12: The figures show the tracking performance in the interval $t = [30, 40]$ when the raw input is used for a backwards finite difference scheme (BFD), AEAD and RLPAD. The differentiators used parameter set B and the solid lines are the estimations and the dashed lines are the true values. It shows that the BFD (a) has a noisy first derivative estimation. Albeit relatively accurate due to the accuracy of the OptiTrack system [1]. However, the second derivative amplifies this noise even more and becomes useless due to a bad signal to noise ratio. AEAD (b) and RLPAD (c) shows the same performance as in Figure 6-10. RLPAD tracks the velocity good with small overshoots and without any noise compared to (a).

Chapter 7

Conclusion

The initial goal of this thesis is to investigate if there are existing methods that can generate generalized measurements from a single input for Active Inference in real time. Real time differentiators are the solution to this, because those are able to estimate higher order derivatives of a single input signal. Chapter 3 conducts a short literature study about differentiators and tabulates eight existing causal differentiators that are scored based on performance metrics. The state of the art differentiator is the algebraic estimation approach differentiator (AEAD), which can extract an arbitrary order of derivatives. However, it turns out that the state of the art differentiator and others cannot track a polynomial input when converted to a discrete time formulation.

This leads to the main goal of this thesis, which is:

Create a custom differentiator that can track a polynomial signal and any other type of input, while performing similar or better than the state of the art differentiator AEAD.

The proposed method, the recurrent low pass algebraic differentiator (RLPAD), has achieved this goal. The RLPAD is based on a truncated Taylor series expansion and a low pass filter at its core. In contrast to the state of the art differentiator, this one is defined in discrete time. This definition makes it able to track polynomial signals, thereby solving the first part of the main goal.

The next part of the goal is to evaluate if the proposed method performs better or similar compared to the AEAD. Two experiments are conducted to numerically evaluate both differentiators. The first experiment in chapter 5 evaluates the performance based on three analytical inputs, which can be differentiated multiple times. The three input functions are a polynomial, a sinusoid and a combination of the two. Additionally, the analytical inputs are corrupted by a Gaussian noise, to determine how well the differentiators attenuate noise.

The second experiment in chapter 6 evaluates how well the differentiators perform when real sensor data is used. It does not matter what kind of system generates the data, because a differentiator is capable of handling any type of signal. In order to evaluate the performance properly, the sensor data is smoothed and filtered by a Savitizky-Golay filter [48] to create a ground truth about the derivatives of the signal. The Savitizky-Golay filter is tuned iteratively so that the noise residual between the smoothed and raw data is highly uncorrelated. This way, the derivative estimations of the differentiators can be evaluated up to the third order derivative.

The first experiment concludes the following. The proposed method performs very similar compared to AEAD for sinusoidal inputs with and without noise, but it estimates the zeroth order derivative better. The proposed method outperforms AEAD significantly when polynomial inputs are used. This also includes a combination of a sinusoid with a polynomial. The proposed differentiator performs exceptionally well when pure polynomial signals are used, even when they are corrupted by noise. The AEAD cannot track these types of signals.

The second experiment concludes the following two points. The first is that the performance of the differentiators are highly affected by the amount of derivatives they estimate. This results in a range for each differentiator, where either the proposed method, or AEAD is better. Therefore, in chapter 6 the differentiators are tested at their optimal parameter n. The proposed method outperforms AEAD when $n = 3$ and performs better than when more derivatives are estimated ($n = 10$). However, when $n = 10$, AEAD outperforms the proposed method and has the lowest absolute error for the first derivative for the specific data set used. Other than that they perform roughly similar for the other derivatives.

The second point is that the performance of the two differentiators, and differentiators in general, is highly affected by the strength of the noise. By artificially increasing the noise of the smoothed data, AEAD shows that it gets less accurate by adding more derivative estimations terms. In contrast, the proposed method is not very sensitive to that and the errors remain almost constant at some point when adding more derivative estimation terms. This is due to the fact that the proposed method has a stronger noise attenuation than AEAD.

Based on the conclusions of these two experiments, the performance metrics of RLPAD are tabulated in Table 7-1 together with the scores of AEAD that were determined in chapter 3. This table shows that RLPAD outperforms AEAD at derivative tracking for low frequency signals. The main reason for this is that the AEAD could not track a polynomial function which can be considered to have zero frequency. With the right tuning and a small enough time step, they perform roughly similar for high frequency signals. As said before, RLPAD is slightly better than AEAD for the zero-th up to the second derivative in the numerical simulations. On the other hand AEAD performs slightly better for real sensor data as input, although this varies a lot by the amount of derivatives estimated and noise of the input. In terms of noise attenuation RLPAD and AEAD perform very similar, however, RLPAD is allowed to have a stronger noise attenuation due to an extra damping parameter. Additionally it is capable of tracking polynomials with noise, therefore the noise attenuation of RLPAD is rated slightly higher. Both differentiators can estimate any order of derivative, however, the accuracy of higher order derivatives is highly affected by the sample rate and noise. The category where AEAD is definitely better at is the amount of tunable parameters, which are two compared to RLPAD that has three.

	Performance of AEAD vs the proposed method (RLPAD)					
Method	LF derivative tracking	HF derivative tracking	noise Attenuation	max derivatives	tunable parameters	underlying technique
AEAD	9.0	7.0	8.0	> 1	2	TSB
RLPAD	10.0	7.0	9.0	> 1	3	TSB

Table 7-1: This table shows how the current state of the art differentiator (AEAD) performs versus the proposed method (RLPAD). LF and HF denotes low frequency and high frequency respectively. The values of AEAD are copied from Table 3-1. AEAD is outperformed by RLPAD for low frequency signals, because AEAD can't track polynomial signals properly. RLPAD can track polynomial inputs and for high frequency signals it performs about the same as AEAD. The noise attenuation is slightly better for RLPAD. The downside of RLPAD is that it has an extra tunable parameter and that it is defined in discrete time.

The following sub goal is partially answered: *Quantify the variance propagation of the higher order derivatives from the input.*

Knowing the variances of the estimated derivatives would simplify the construction of the precision matrix in Active Inference that corresponds with the generalized measurements. The premise is that the variances of the outputs of RLPAD can be determined analytically when the input noise variance is known. Determining the variances analytically turned out to be a difficult problem and is out of the scope of this thesis, thus the premise was not evaluated as true nor false. Instead, the variances of the output can be determined with numerical simulations by having as input a stationary signal with additive Gaussian noise. However, this is a general method that works for all differentiators.

Now the final question can be answered: *which method is best suited for creating generalized measurements in Active Inference?*

RLPAD is more suitable than AEAD for creating generalized measurements in Active Inference for three reasons. The first is that RLPAD can estimate derivatives of polynomial signals properly which is something AEAD cannot. However, when real sensor data is used, AEAD performs slightly better than RLPAD. The main reason for this is that AEAD has the lowest absolute error for the first order derivative for the data set used. However, the performance of both differentiators are highly affected by noise and the amount of derivatives estimated.

The second reason is that RLPAD has superior noise attenuation. The way RLPAD is defined makes sure that the higher order derivatives eventually go to zero. AEAD, on the other hand, amplifies the noise considerably for each consecutive derivative estimate.

The third reason is that RLPAD performs better when a lower number of derivatives are estimated. This is advantageous, because it reduces computational time at the cost of some accuracy.

Chapter 8

Recommendations and future work

This research came with its own difficulties. Some ideas could not be implemented or could not be solved due to time constraints. These difficulties gave rise to new ideas and questions. Therefore, this chapter covers three recommendations of the recurrent low pass algebraic differentiator (RLPAD), also known as the proposed method. Furthermore, it mentions an inspiration for a potential new filter. And finally, states two new research question for Active Inference as future work.

Recommendations

The first thing that stands out in chapter 4 is that the proposed method is defined in discrete time. Therefore, the first recommendation is that it can be beneficial if RLPAD can be defined in continuous time. Attempts have been made to transform it to continuous time, but were unsuccessful. A continuous time definition would make the tuning of the filter easier. The reason for this is that once a set of parameters work in continuous time, they behave the same when converted to discrete time when the most precise conversion is used under the assumption that the filter is stable. On the other hand, the discrete time variant requires retuning as soon as the time step becomes larger, because it can become unstable. However, a danger is that when the proposed method is converted to continuous time, it might not be able to track polynomial signals when converted to discrete time. This was also the case for the algebraic estimation approach differentiator (AEAD) and Taylor series expansion based differentiator (TSEBD).

The second recommendation is to find a better expression to tune the cutoff frequencies ω_i. In this work, they are defined as $\omega_{i+1} = \frac{\omega_i}{f_{\text{red}}}$ for $i = 0, 1, ..., n-1$. This expression helps to reduce the amount of tunable parameters of the filter to three, however, it is not proven that each subsequent ω is optimal. Therefore a suggestion is to find an optimal set of ω_i if there exists one.

The third recommendation is that the performance of the proposed differentiator can be improved by multi filtering. Multi filtering is not considered in this thesis, however, the authors of AEAD [32] did for their differentiator. They show that multi filtering can improve noise attenuation significantly, therefore multi filtering can potentially improve the performance of RLPAD.

Inspiration for a new filter

A Savitzky-Golay filter is used in chapter 6 to smooth the data and extract the derivatives at the same time. The ability to smooth and extract derivatives is a very interesting property of this filter. It would be very beneficial if this filter could work in real time. Sadly this filter is non-causal as it takes future and past samples into account. Therefore, this gave rise to the idea that the underlying technique of the Savitzky-Golay filter (fitting a local polynomial through data points in the least

square sense) can be an inspiration to develop a causal filter that can smooth and estimate derivatives at the same time.

A filter like this can overcome the high bandwidth requirements for real time differentiators, because a polynomial fit works with a lower bandwidths. Although this comes with its own set of problems [12], because a too low bandwidth results in noisy estimates and a too high bandwidth creates a bias. This causes loss of information of the original data.

Future work

This thesis gives rise to the first research question: How does Active Inference perform when it uses differentiators to construct the generalized coordinates? Current implementations of Active Inference [10] or other methods that make use of generalized coordinates [37] make use of a simple backward difference scheme to estimate higher order derivatives, which is very inaccurate when dealing with noisy signals. This thesis shows that there are better methods that can estimate higher order derivatives in real time. This means that higher order derivatives of sensor data can be estimated to improve the performance of Active Inference. Therefore, it would be very interesting to evaluate how Active Inference performs when differentiators are used to estimate higher order derivatives. Also note that not only the derivatives of measurements can be estimated, but also those of the inputs.

In turn this also gives directly rise to the second research question: How many embedding orders are necessary in practical implementation of Active Inference? Recall that the embedding order indicates the amount of higher order derivatives Active Inference uses. Friston mentions that signals can still have useful information up to an embedding order of six [24]. However, even with low noise it is difficult to estimate the second order derivative accurately as this thesis shows. Let alone the sixth order derivative. It is therefore interesting to investigate to which embedding order Active Inference improves its performance in practical implementations.

These last two stated questions are very interesting and can bring the full potential of Active Inference one step closer in robotics.

Bibliography

[1] Optitrack system. https://www.optitrack.com. Accessed: 25-8-2021.

[2] Parot ar 2.0 drone. https://www.drones.nl/. Accessed: 6-11-2021.

[3] Parot ar 2.0 drone image. https://droneshop.nl. Accessed: 27-10-2021.

[4] Marco Angulo, Jaime Moreno, and Leonid Fridman. Robust exact uniformly convergent arbitrary order differentiator. *Automatica*, 49:V, 08 2013.

[5] Mohamed Baioumy, Paul Duckworth, Bruno Lacerda, and Nick Hawes. Active inference for integrated state-estimation, control, and learning, 2021.

[6] D Benders. Ar.drone 2.0 state estimation using dynamic expectation maximization. *Master of science thesis*, 12 2020.

[7] Fred Bos, Ajith Anil Meera, Dennis Benders, and Martijn Wisse. Free energy principle for state and input estimation of a quadcopter flying in wind. 2021.

[8] Christopher L. Buckley, Chang Sub Kim, Simon McGregor, and Anil K. Seth. The free energy principle for action and perception: A mathematical review. *Journal of Mathematical Psychology*, 81:55–79, 2017.

[9] J. Coehorn. A comparison of active inference and linear-quadratic gaussian control. *Master of science thesis*, 8 2021.

[10] M. Deken. Applied hierarchical active inference on a skid steering mobile robot. *Master of science thesis*, 8 2021.

[11] L. Drolet, F. Michaud, and J. Cote. Adaptable sensor fusion using multiple kalman filters. In *Proceedings. 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2000) (Cat. No.00CH37113)*, volume 2, pages 1434–1439 vol.2, 2000.

[12] Jianqing Fan and Irène Gijbels. Adaptive order polynomial fitting: Bandwidth robustification and bias reduction. *Journal of Computational and Graphical Statistics*, 4(3):213–227, 1995.

[13] Hongyinping Feng and Shengjia Li. A tracking differentiator based on taylor expansion. *Applied Mathematics Letters*, 26:735–740, 07 2013.

[14] Richard. P Feynman. *Statistical Mechanics*. W.A. Benjamin, 1 edition, 1977.

[15] Ragnar Fjelland. Why general artificial intelligence will not be realized. *Humanities and Social Sciences Communications*, 7:10, 06 2020.

[16] Bruce Francis and Walter Wonham. The internal model principle of control theory. *Automatica*, 12:457–465, 09 1976.

[17] Karl Friston. Hierarchical models in the brain. *PLOS Computational Biology*, 4(11):1–24, 11 2008.

[18] Karl Friston. Friston, k.j.: The free-energy principle: a unified brain theory? nat. rev. neurosci. 11, 127-138. *Nature reviews. Neuroscience*, 11:127–38, 02 2010.

[19] Karl Friston. What is optimal about motor control? *Neuron*, 72:488–98, 11 2011.

[20] Karl Friston, Jérémie Mattout, and James Kilner. Action understanding and active inference. *Biological cybernetics*, 104:137–60, 02 2011.

[21] Karl Friston, Spyridon Samothrakis, and Read Montague. Active inference and agency: optimal control without cost functions. *Biological Cybernetics*, 106(106):523–541, 10 2012.

[22] Karl Friston and Klaas Stephan. Hierarchical models in the brain. *Synthese*, 159(3):417–458, 12 2007.

[23] Karl J. Friston, Jean Daunizeau, J Kilner, and Stefan J. Kiebel. Action and behavior: a free-energy formulation. *Biological Cybernetics*, 102(3):227–260, 03 2010.

[24] KJ Friston, N Trujillo-Barreto, and J Daunizeau. Dem: a variational treatment of dynamic systems. *NeuroImage*, 41(3):849—885, July 2008.

[25] Crispin Gardiner. *Stochastic Methods*, volume 4. 2009.

[26] Matthieu Geist and Olivier Pietquin. Kalman filtering & colored noises: the (autoregressive) moving-average case. page 4 pages, 12 2011.

[27] S Grimbergen. The state space formulation of active inference. *Master of science thesis*, 01 2019.

[28] I.L. Hijne. Generalised motions in active inference by finite differences. *Master of science thesis*, 8 2020.

[29] Geoffrey E. Hinton and D. Camp. Keeping the neural networks simple by minimizing the description length of the weights. In *COLT '93*, 1993.

[30] Lazaros S. Iliadis, Vera Kurkova, and Barbara Hammer. Brain-inspired computing and machine learning. *Neural Computing and Applications*, 32, 2020.

[31] R. E. Kalman. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 82(1):35–45, 03 1960.

[32] Josip Kasac, Dubravko Majetic, and Danko Brezak. An algebraic approach to on-line signal denoising and derivatives estimation. *Journal of the Franklin Institute*, 355, 08 2018.

[33] Hugh Kennedy. Recursive digital filters with tunable lag and lead characteristics for proportional–differential control. *IEEE Transactions on Control Systems Technology*, 23:2369–2374, 11 2015.

[34] Pablo Lanillos and G. Cheng. Active inference with function learning for robot body perception. 2018.

[35] Arie Levant. Robust exact differentiation via sliding mode technique**this paper was recommended for publication in final form by associate editor hassan khalil under the direction of editor tamer basar. *Automatica*, 34(3):379–384, 1998.

[36] D. J. C. MacKay. Free energy minimisation algorithm for decoding and cryptanalysis. *Electronics Letters*, 31(6):446–447, 1995.

[37] Ajith Anil Meera and Martijn Wisse. A brain inspired learning algorithm for the perception of a quadrotor in wind. 2021.

[38] Eduardo Vieira Leao Nunes, Liu Hsu, and Fernando Lizarralde. Global exact tracking for uncertain systems using output-feedback sliding mode control. *IEEE Transactions on Automatic Control*, 54(5):1141–1147, 2009.

[39] Keshab K. Parhi and Nanda K. Unnikrishnan. Brain-inspired computing: Models and architectures. *IEEE Open Journal of Circuits and Systems*, 1:185–204, 2020.

[40] Corrado Pezzato, Riccardo M. G. Ferrari, and Carlos Hernandez. A novel adaptive controller for robot manipulators based on active inference. *CoRR*, abs/1909.12768, 2019.

[41] Léo Pio-Lopez, Ange Nizard, Karl Friston, and Giovanni Pezzulo. Active inference and robot control: A case study. *Journal of the Royal Society, Interface*, 13, 09 2016.

[42] Bernd Porr and Florentin Wörgötter. Isotropic sequence order learning. *Neural computation*, 15:831–64, 05 2003.

[43] K. V. Ramachandra. *Kalman filtering techniques for radar tracking*. CRC Press, an imprint of Taylor and Francis, 2000.

[44] Rajesh Rao and Dana Ballard. Predictive coding in the visual cortex: a functional interpretation of some extra-classical receptive-field effects. *Nature neuroscience*, 2:79–87, 02 1999.

[45] Brown R.G. *Smoothing, Forecasting and Prediciton of Discrete Time Series*. Dover Publication, Inc., 1963.

[46] Tiago Roux Oliveira, Alessandro Peixoto, Eduardo Nunes, and Liu Hsu. Control of uncertain nonlinear systems with arbitrary relative degree and unknown control direction using sliding modes. *International Journal of Adaptive Control and Signal Processing*, 21:692 – 707, 10 2007.

[47] Mohammad Sadeghi and Fereidoon Behnia. Optimum window length of savitzky-golay filters with arbitrary order, 2018.

[48] Abraham. Savitzky and M. J. E. Golay. Smoothing and differentiation of data by simplified least squares procedures. *Analytical Chemistry*, 36(8):1627–1639, 1964.

[49] Ivan W. Selesnick, Stephen Arnold, and Venkata R. Dantham. Polynomial smoothing of time series with additive step discontinuities. *IEEE Transactions on Signal Processing*, 60(12):6305–6318, 2012.

[50] Nazmul H. Siddique and Hojjat Adeli. Nature inspired computing: An overview and some future directions. *Cognitive Computation*, 7:706 – 714, 2015.

[51] Teng-Tiow Tay, John Moore, and Iven Mareels. *High Performance Control*. 01 1997.

[52] Hongwei Wang and Wang Heping. A comparison study of advanced tracking differentiator design techniques. *Procedia Engineering*, 99:1005–1013, 12 2015.

[53] Xinhua Wang and Hai Lin. Design and analysis of continuous hybrid differentiator. *Computing Research Repository - CORR*, 5, 03 2011.

[54] Zebo Zhou, Jin Wu, Yong Li, Chen Fu, and Hassen Fourati. Critical issues on kalman filter with colored and correlated system noises. *Asian Journal of Control*, 19(6):1905–1919, 2017.