Filtragem de Kalman Neuronal para Aplicações Aeroespaciais

(Versão Final Após Defesa)

Felippe Ferreira da Silva

Dissertação para obtenção do Grau de Mestre em
Engenharia Aeronáutica
(mestrado integrado)

Orientador: Prof. Doutor Kouamana Bousson

Dezembro de 2024

Declaração de Integridade

Eu, Felippe Ferreira da Silva, que abaixo assino, estudante com o número de inscrição 40689 de Engenharia Aeronáutica da Faculdade de Engenharia, declaro ter desenvolvido o presente trabalho e elaborado o presente texto em total consonância com o **Código de Integridades da Universidade da Beira Interior**.

Mais concretamente afirmo não ter incorrido em qualquer das variedades de Fraude Académica, e que aqui declaro conhecer, que em particular atendi à exigida referenciação de frases, extratos, imagens e outras formas de trabalho intelectual, e assumindo assim na íntegra as responsabilidades da autoria.

Universidade da Beira Interior, Covilhã 04 /12 /2024

Assinado por: **FELIPPE FERREIRA DA SILVA**
Num. de Identificação: 32279547
Data: 2024.12.04 10:50:11+00'00'

CARTÃO DE CIDADÃO

Dedicatória

Dedico este trabalho a toda minha família, que apesar de estar tão longe nunca deixou de me apoiar. Também dedico a todos aqueles que sempre buscaram por mais conhecimento e, com este, tornar este mundo um lugar um pouco melhor.

Agradecimentos

Primeiramente, gostaria de agradecer à minha família a qual sempre me apoiou, apesar de estar distante, e tornou este caminho mais fácil de ser trilhado, principalmente à minha mãe, que mudou de vida para que eu pudesse estar onde estou e conquistar os meus sonhos.

Gostaria também de agradecer a todo o corpo docente desta universidade, em particular ao do Departamento de Ciências Aeroespaciais, por tornar esta jornada pelo conhecimento possível. Em especial, agradecer ao meu orientador, Professor Doutor Kouamana Bousson, pelo trabalho incrível de acompanhamento e orientação, e pela sua paciência e disponibilidade, os quais em conjunto tornaram possível a realização deste trabalho. Afinal, "se eu vi mais longe, foi porque estava sobre os ombros de gigantes". Sir Isaac Newton – 1675.

Por fim, gostaria de agradecer àqueles que estiveram ao meu lado e batalharam durante esta caminhada, seja nos momentos felizes, como nos tristes, e tornaram estes seis anos os melhores da minha vida, os meus amigos. Agradeço ao Marco, ao Bruno, ao Samuel, ao Miguel e ao Lucas, por serem quem são e espero que continuem sendo as pessoas incríveis que sempre estiveram ao meu lado. A busca pelo conhecimento é uma jornada longa e sem fim, mas a felicidade encontra-se ao nosso lado.

Resumo

A estimação em tempo real é um dos tópicos mais importantes na engenharia, principalmente em sistemas não-lineares, presentes em praticamente todas as aplicações aeroespaciais. Na indústria aeroespacial, há cada vez mais a tendência de utilizar instrumentos, sensores e métodos de estimação que sejam precisos, leves e com baixos custos computacional e financeiro.

O termo filtragem ou estimação é utilizado para o conjunto de técnicas e algoritmo que permitem obter uma redução de ruídos ou erros de medição além de estimar estados não mensuráveis diretamente. A grande dificuldade encontra-se justamente na complexidade dos sistemas observados, que podem apresentar dinâmica altamente não-linear, além de ruídos e baixa disponibilidade de dados, o que torna a seleção do método de filtragem um fator muito importante para que haja o máximo de redução do erro de estimação.

Um dos métodos mais importantes desenvolvidos foi o filtro de Kalman linear, considerado filtro ótimo para sistemas lineares. Mas, visto que a maioria dos sistemas reais envolve dinâmica não-linear, outros métodos foram desenvolvidos com vista a superar este impasse. Métodos como o famoso filtro de Kalman Estendido (EKF – *Extended Kalman Filter*), o filtro de Kalman Unscented (UKF – *Unscented Kalman Filter*) e o filtro de Kalman Pseudo-linear (PSELIKA – *Pseudo Linear Kalman Filter*) foram algumas das soluções implementadas que tiveram sucesso em aplicações reais, apresentando vantagens e limitações particulares.

Para o filtro EKF, a sua principal limitação é o seu comportamento errático e possivelmente divergente em sistemas altamente não-lineares, ou com estimações iniciais *imprecisas*. Já para o UKF, a dimensão do estado pode aumentar significativamente a complexidade temporal do algoritmo, o que pode torná-la uma solução inviável para tempo real. O filtro PSELIKA apresenta limitação parecida onde a complexidade das operações efetuadas é a principal causa do custo computacional elevado.

Destes métodos, o PSELIKA chama atenção pelo facto de ser aplicável em praticamente qualquer sistema não-linear, dentro das condições necessárias, mas sendo possível utilizar os métodos de filtragem linear ao modelar a dinâmica do sistema numa estrutura pseudo-linear. A limitação é dada pela necessidade de resolver a equação Algébrica de Riccati associada e realizar inversões matriciais para o cálculo da matriz de ganho do sistema, necessitando bastantes recursos computacionais.

Assim, com o objetivo de reduzir o custo computacional e melhorar o desempenho, as redes neuronais têm sido integradas aos métodos de filtragem. Estas são ferramentas extraordinárias que atuam como funções não-lineares capazes de obter conhecimento experimental, e neste caso, podem atuar para prever os cálculos efetuados durante o método de filtragem PSELIKA.

De forma a reduzir a principal limitação do método de estimação PSELIKA, aproveitando do facto de apresentar uma estrutura pseudo-linear, esta dissertação propôs a integração de uma rede neuronal do tipo *Perceptron* de Múltiplas Camadas (MLP- *Multilayer Perceptron*) ao algoritmo PSELIKA, tendo como principal objetivo a redução do custo computacional original, implementando, por fim, um algoritmo denominado PSELIKA-MLP.

Com vista a validar a utilização do algoritmo, nesta dissertação são abordados dois casos de estudo: a estimação da taxa angular e atitude de um veículo aeroespacial, além da estimação da órbita de um satélite artificial. Nestes, são comparados a exatidão, complexidade temporal e espacial entre o algoritmo implementado, o filtro EKF e o filtro PSELIKA.

Os resultados demonstram que o algoritmo implementado foi capaz de reduzir significativamente o tempo computacional do método PSELIKA, sem um grande detrimento da exatidão da estimação, mas apresentando complexidade espacial superior. Assim, é uma possível aplicação para a redução do custo computacional servindo de complemento para o algoritmo original.

Palavras-chave

Filtragem Não-Linear; Filtro de Kalman Pseudo-Linear; Redução do Custo Computacional; Redes Neuronais Artificiais; Aplicações aeroespaciais.

Abstract

Real-time estimation is one of the most important topics in engineering, especially in non-linear systems, present in practically all aerospace applications. In the aerospace industry, there is an increasing tendency to use instruments, sensors and estimation methods that are precise, lightweight and with low computational and financial costs.

The term filtering or estimation is used for the set of techniques and algorithms that allow reducing noise or measurement errors in addition to estimating states that cannot be measured directly. The great difficulty lies precisely in the complexity of the observed systems, which can present very non-linear dynamics in addition to noise and low data availability, which makes the selection of the filtering method a very important factor for maximum reduction of the estimation error.

One of the most important methods developed was the linear Kalman filter, considered an optimal filter for linear systems. But since most real systems involve non-linear dynamics, other methods have been developed to overcome this impasse. Methods such as the famous Extended Kalman filter (EKF), the Unscented Kalman filter (UKF) and the Pseudo-linear Kalman filter (PSELIKA) were some of the solutions implemented that have been successful in real applications, presenting advantages and limitations.

For the EKF filter, its main limitation is its erratic and possibly divergent behavior in highly non-linear systems, or with *imprecise* initial estimations. As for UKF, the dimension of the state can significantly increase the temporal complexity of the algorithm, which can make it an unfeasible solution for real time. The PSELIKA filter presents a similar limitation where the complexity of the operations performed is the main cause of the high computational cost.

Of these methods, PSELIKA draws attention because it is applicable to practically any non-linear system, within the necessary conditions, but it is possible to use linear filtering methods when modeling the system dynamics in a pseudo-linear structure. The limitation is given by the need to solve the associated Algebraic Riccati equation and perform matrix inversions to calculate the system's gain matrix, requiring considerable computational resources.

Thus, with the aim of reducing computational cost and improving performance, neural networks have been integrated into filtering methods. These are extraordinary tools that act as non-linear functions capable of obtaining experimental knowledge, and in this case, can act to predict the calculations carried out during the PSELIKA filtering method.

To reduce the main limitation of the PSELIKA estimation method, taking advantage of the fact that it presents a pseudo-linear structure, this dissertation proposed the integration of a Multilayer Perceptron (MLP- Multilayer Perceptron) neural network into the PSELIKA algorithm, with the main objective of reducing the original computational cost, finally implementing an algorithm called PSELIKA-MLP.

With a view to validate the use of the algorithm, this dissertation addresses two case studies: the estimation of the angular rate and attitude of an aerospace vehicle, in addition to the estimation of the orbit of an artificial satellite. In these, the accuracy, temporal and spatial complexity between the implemented algorithm, the EKF filter and the PSELIKA filter are compared.

The results demonstrate that the implemented algorithm was able to significantly reduce the computational time of the PSELIKA method, without a major detriment to estimation accuracy, but presenting superior spatial complexity. Thus, it is a possible application to reduce computational cost by serving as a complement to the original algorithm.

Keywords

Non-Linear Filtering; Pseudo-Linear Kalman Filter; Reduction of Computational Cost; Artificial Neural Networks; Aerospace applications.

Índice

Lista de Figuras

Lista de Tabelas

Lista de Acrónimos

AdaGrad	-	*Adaptive Gradient*
ADALINE	-	*Adaptive LInear NEuron*
ADAM	-	*Adaptive Moment Estimation*
BFGS	-	Broyden-Fletcher-Goldfarb-Shanno
CKF	-	*Cubature Kalman Filter*
DARE	-	*Discrete Algebraic Riccati Equation*
EKF	-	*Extended Kalman Filter*
GPT	-	*Generative Pre-Trained Transformer*
KF	-	*Kalman Filter*
L-BFGS	-	*Limited-Memory* Broyden-Fletcher-Goldfarb-Shanno
LLM	-	*Large Language Model*
LQR	-	*Linear Quadratic Regulator*
LSTM	-	*Long Short-Term Memory*
MADALINE	-	*Multiple Adaptive Linear Neuron*
MLP	-	*Multilayer Perceptron*
MLPRegressor	-	*Multilayer Perceptron Regressor*
MLFFNN	-	*Multilayer Feedforward Neural Network*
MSE	-	*Mean Squarred Error*
NN	-	*Neural Network*
PDP	-	*Parallel Distributed Processing*
PSELIKA	-	*Pseudo-Linear Kalman Filter*
PSELIKA-MLP	-	Filtro de Kalman Pseudo-linear com *Perceptron* de Múltiplas Camadas

RBF	-	*Radial Basis Function*
RK	-	Runge - Kutta
RKN	-	*Recurrent Kalman Network*
RL	-	*Reinforcement Learning*
RMSE	-	*Root Mean Squarred Error*
RMSProp	-	*Root Mean Square Propagation*
RNN	-	*Recurrent Neural Network*
SDC	-	*State-Dependent Coefficient*
SDRE	-	*State-Dependent Riccati Equation*
SGD	-	*Stochastic Gradient Descent*
SLP	-	*Single-Layer Perceptron*
SSE	-	*Sum of Squarred Errors*
UKF	-	*Unscented Kalman Filter*
UT	-	*Unscented Transformation*
XOR	-	*Exclusive - Or*

Lista de Símbolos

Símbolo	Descrição	Unidades
α	Parâmetro de factorização de um sistema não-linear	[-]
α_i	Vetor definido para o método L-BFGS	[-]
β_i	Vetor definido para o método L-BFGS	[-]
γ_t	Parâmetro para estimativa inicial de H_t^0	[-]
δ_{ij}	Kronecker delta com índices i e j	[-]
δ_i^l	Erro no neurónio i da camada l	[-]
$\Delta\zeta_i^l$	Erro associado a mudança do valor de ζ_i^l	[-]
ϵ	Taxa de aprendizagem de uma rede neuronal	[-]
ζ	Resultado da soma ponderada de um *perceptron*	[-]
η	Valor de entrada de um *perceptron*	[-]
θ_k	Azimute no sistema de coordenadas esféricas no instante t_k	[rad]
ϑ	Conjunto de parâmetros de uma rede neuronal	[-]
ϑ_0	Valor inicial para o conjunto de parâmetros de uma rede neuronal	[-]
κ	Parâmetro real utilizado na filtragem UKF	[-]
λ	Valor real positivo	[-]
μ	Parâmetro gravitacional terrestre	[N.m^2/kg]
$\bar{\mu}$	Média de uma população	[-]
Ξ_k	Conjunto de valores do vetor quaternião para o instante t_k	[-]
ρ_i	Escalar definido para o método L-BFGS	[-]
σ_{var}	Desvio padrão da variável var	[-]
ϕ_k	Zénite no sistema de coordenadas esféricas no instante t_k	[rad]

Φ_k	Matriz de transição do instante t_k	[-]
χ_1, χ_2	Argumentos para condição de convexidade de uma função	[-]
ω	Taxa angular do veículo aeroespacial	[rad/s]
$\widehat{\omega}$	Estimativa do vetor de taxa angular do veículo aeroespacial	[rad/s]
Ω_k	Conjunto de valores do vetor taxa angular para o instante t_k	[rad/s]
a	Valor de saída da função de ativação de um *perceptron*	[-]
a_j^l	Valor de saída da função de ativação do neurónio j da camada l	[-]
A	Matriz que descreve a linearização de um espaço de estados em torno de um estado x	[-]
A_d	Matriz discretizada do modelo do sistema	[-]
b	Viés de um *perceptron*/modelo dinâmico	[-]
B	Matriz que relaciona a variável u ao estado x, em torno de um estado x	[-]
B_d	Matriz discretizada do modelo de controlo do sistema	[-]
cte	Constante utilizada para função bipolar sigmoide	[-]
D	Matriz simétrica genérica	[-]
$E[\cdot]$	Função esperança matemática	[-]
E_{var}	Margem atribuída a variável var durante o treinamento de rede neuronal	[-]
$F_A(\cdot)$	Função de ativação	[-]
$f(\cdot)$	Função genérica/não-linear	[-]
$f_{UKF}[\cdot]$	Função de transformação não-linear da filtragem UKF	[-]
$F(\cdot)$	Matriz Jacobiana da função f	[-]
$\boldsymbol{g}$	Gradiente de uma função genérica f	[-]
$g_{in}(\cdot)$	Função genérica interna	[-]
$g_{out}(\cdot)$	Função genérica externa	[-]
G	Constante universal gravitacional	$[\mathrm{N.m^2/kg^2}]$

$h(\cdot)$	Função não-linear que descreve a relação entre medição z_k e estado x_k	[-]
h_{SC}	Momento angular das rodas de momento do veículo aerospacial	[kg.m^2/s]
$h_{UKF}[\cdot]$	Função que descreve o modelo de medição na filtragem UKF	[-]
H	Matriz que relaciona o estado x a medição z	[-]
$H_k(\cdot)$	Matriz Jacobiana da função h	[-]
Hd	Resultado do produto Hadamard	[-]
$\boldsymbol{H_s}$	Matriz hessiana de uma função genérica f	[-]
H_t^0	Primeira aproximação da matriz Hessiana para o método L-BFGS	[-]
I	Matriz identidade	[-]
I_{SC}	Matriz de inércia do veículo aeroespacial	[N.m^2]
J	Função de custo para minimização	[-]
J_2	Segundo harmónico zonal	[-]
J_k	Função de custo para filtragem de Kalman	[-]
$k(\cdot)$	Lei de controlo do método LQR	[-]
K_k	Matriz de ganho de Kalman para o instante t_k	[-]
l	Numeração de camadas escondidas	[-]
L	Função de custo de uma rede neuronal	[-]
$max(\cdot)$	Função valor máximo de um conjunto	[-]
M_{Terra}	Massa terrestre	[kg]
n	Dimensão da variável aleatória/ função genérica	[-]
$N(\cdot)$	Função de distribuição de probabilidades gaussiana	[-]
$N_{corr}(\cdot)$	Função de correção não-linear dada por uma rede neuronal	[-]
N_{passos}	Número de passos de uma simulação	[-]
N_{sim}	Número de simulações de Monte Carlo	[-]

N_l	Número de neurónios na camada l	[-]
O_b	Matriz de observabilidade	[-]
$p(\cdot)$	Função de probabilidade de densidade condicional	[-]
p	Número de vetores do conjunto de pontos sigma menos 1	[-]
$Posição_k$	Conjunto de valores do vetor posição para o instante t_k	[km]
P_0	Matriz de covariância do erro de estimação inicial	[-]
$P_k(+)$	Matriz de covariância a posteriori da medição no instante t_k	[-]
$P_k(-)$	Matriz de covariância a priori da medição no instante t_k	[-]
P_{UKF}	Matriz de covariância dos pontos de transformação $z_{UKF}^{(i)}$	[-]
$\hat{P}_k^{xy}$	Matriz de covariância cruzada da filtragem UKF	[-]
q	Vetor quaternião do veículo aeroespacial	[-]
$\vec{q}$	Gradiente da função f na etapa $\boldsymbol{t}$ do método L-BFGS	[-]
$\hat{q}$	Estimativa do vetor quaternião	[-]
Q_k	Matriz de covariância do erro de processamento para w_k	[-]
$rank(\cdot)$	Característica matricial	[-]
Q	Matriz da dinâmica dos quaterniões	[-]
r	Distância no sistema de coordenadas esféricas	[km]
r_a	Distância do apogeu de uma órbita	[km]
r_p	Distância do perigeu de uma órbita	[km]
r_{Terra}	Raio equatorial terrestre	[km]
R	Matriz de covariância do erro de medição	[-]
R^2	Coeficiente de desempenho de modelo de previsão	[-]
$\hat{R}_k$	Matriz de covariância da inovação na filtragem UKF	[-]
R_k	Matriz de covariância do erro de medição para v_k	[-]
$ReLU(\cdot)$	Função Unidade Linear Retificada	[-]

s	Variável real independente	[-]
$s_{\boldsymbol{t}}$	Diferença do parâmetro objetivo para etapa $\boldsymbol{t}+1$ e $\boldsymbol{t}$ do método L-BFGS	[-]
$sig_1(\cdot)$	Função de polo único sigmoide	[-]
$sig_2(\cdot)$	Função bipolar sigmoide	[-]
$step(\cdot)$	Função Heaviside	[-]
S	Matriz semi-definida positiva arbitrária	[-]
t	Tempo	[s]
$\boldsymbol{t}$	Etapa dos métodos L-BFGS	[-]
T_s	Passo de discretização/simulação	[s]
T_{SC}	Somatório dos momentos externos aplicados ao veículo aeroespacial	[N.m]
$tr\,[\cdot]$	Função traço matricial	[-]
u	Vetor de entrada de função/de controlo	[-]
U_1, U_2	Matrizes genéricas de mesma dimensão	[-]
v	Vetor de ruído presente numa medição	[-]
v_k	Vetor de ruído presente numa medição no instante t_k	[-]
$v_{r,k}$	Velocidade radial orbital no instante t_k	[km/s]
$v_{\phi,k}$	Velocidade zenital orbital no instante t_k	[km/s]
$v_{\theta,k}$	Velocidade Azimutal orbital no instante t_k	[km/s]
$\hat{v}_{r,i}$	Estimativa da velocidade radial orbital no instante t_k	[km/s]
$\hat{v}_{\phi,i}$	Estimativa da velocidade zenital orbital no instante t_k	[km/s]
$\hat{v}_{\theta,i}$	Estimativa da velocidade Azimutal orbital no instante t_k	[km/s]
V_k	Conjunto de valores do vetor velocidade para o instante t_k	[km/s]
w_k	Vetor de ruído de processamento	[-]
W	Peso atribuído as conexões de um *perceptron*	[-]

W_{ij}^{l}	Peso atribuído a conexão do neurónio i da camada $l-1$ para o neurónio j da camada l	[-]
$W_{UKF}^{(i)}$	Peso associado aos pontos sigma da filtragem UKF	[-]
x	Vetor de estado	[-]
x_0	Vetor de estado inicial	[-]
x_k	Estado no instante t_k	[-]
$\hat{x}$	Estimativa do vetor estado	[-]
$\hat{x}_0$	Estimativa inicial do vetor de estado	[-]
$\hat{x}_k(+)$	Estimativa a posteriori da medição no instante t_k	[-]
$\hat{x}_k(-)$	Estimativa a priori da medição no instante t_k	[-]
$\tilde{x}_k(+)$	Erro de estimativa a posteriori da medição no instante t_k	[-]
$\tilde{x}_k(-)$	Erro de estimativa a priori da medição no instante t_k	[-]
$x_1, x_2, \ldots, x_n$	Elementos do vetor estado	[-]
$x_{UKF}^{(i)}$	Ponto sigma i da filtragem UKF	[-]
$x_{UKF,a}^{i}$	Ponto sigma i do modelo aumentado da filtragem UKF	[-]
X	Conjunto de variáveis independentes	[-]
$X_{p,k}$	Coordenada no eixo x da posição no instante t_k	[-]
$\hat{X}_{p,k}$	Estimativa da coordenada no eixo x da posição no instante t_k	[-]
X_i	Elemento i do conjunto X	[-]
$y_{\boldsymbol{t}}$	Diferença de gradientes para etapa $\boldsymbol{t}+1$ e $\boldsymbol{t}$ do método L-BFGS	[-]
$y_{UKF}^{(i)}$	Ponto i de transformação não-linear na filtragem UKF	[-]
$y_{UKF,a}^{(i)}$	Ponto i de transformação não-linear do modelo aumentado na filtragem UKF	[-]
$\bar{y}_{UKF}$	Média dos pontos de transformação na filtragem UKF	[-]
$\bar{y}_{UKF,a}$	Média dos pontos de transformação do modelo aumentado na filtragem UKF	[-]

Y	Conjunto de observação	[-]
Y_i	Elemento i do conjunto Y	[-]
$\hat{Y}_i$	Estimativa do elemento i do conjunto de observação do treinamento por uma rede neuronal	[-]
$Y_{p,k}$	Coordenada no eixo y da posição no instante t_k	[-]
$\hat{Y}_{p,k}$	Estimativa da coordenada no eixo y da posição no instante t_k	[-]
$\bar{Y}$	Média dos elementos do conjunto Y	[-]
z	Conjunto de medições efetuadas	[-]
z_k	Conjunto de medições efetuadas no instante t_k	[-]
$\hat{z}_k$	Observação prevista no modelo de observação da filtragem UKF	[-]
$\hat{z}_k^{(i)}$	Ponto de predição i instanciado no modelo de observação da filtragem UKF	[-]
$z_{1k}, z_{2k}, \dots, z_{lk}$	Elementos do vetor de medição no instante t_k	[-]
$Z_{p,k}$	Coordenada no eixo z da posição no instante t_k	[-]
$\hat{Z}_{p,k}$	Estimativa da coordenada no eixo z da posição no instante t_k	[-]

Capítulo 1 – Introdução

1.1 Contexto e Motivação

O avanço tecnológico, assim como a compreensão do mundo, só foi possível através das observações que são feitas do meio em que se vive. No campo aerospacial, para que seja possível uma aeronave realizar a sua navegação com segurança, é necessário que haja conhecimento de diversos parâmetros ou estados durante sua fase de voo, como por exemplo a posição, velocidade e atitude.

Diversos instrumentos e sensores foram desenvolvidos com o objetivo de tentar obter os valores acerca destes estados, mas não é possível mensurá-los com perfeita exatidão uma vez que existem ruídos e imperfeições associadas a medição. Este ruído deve ser eliminado ou pelo menos reduzido para que os dados obtidos a partir dos instrumentos sejam utilizáveis. Perturbações de diversas fontes e o próprio sistema de medição podem causar distúrbios na medição e provocar os tais dados ruidosos.

O termo filtragem ou estimação abrange o conjunto de técnicas e algoritmos com o intuito de obter dados acerca de estados que não podem ser obtidos por medição direta, ou que apresentem medições ruidosas, reduzindo ao máximo o seu erro. Estas técnicas utilizam modelos matemáticos do sistema e algoritmos sofisticados para inferir o estado real do sistema a partir de dados imperfeitos [1]. A escolha da técnica adequada depende de diversos fatores, como a complexidade do sistema, a natureza do ruído e a disponibilidade de dados.

Diversos métodos foram desenvolvidos ao longo dos anos devido a sua extrema importância em várias áreas da ciência. Um dos mais famosos é o filtro de Kalman linear (KF – *Kalman Filter*) sendo considerado o filtro ótimo para sistemas lineares. Entretanto, a grande maioria dos sistemas reais são não lineares, principalmente na área aeroespacial, o que limita a utilização deste filtro ótimo. Assim, versões foram desenvolvidas, baseadas neste filtro, para que pudessem ser aplicáveis aos sistemas não-lineares. Destes métodos a filtragem de Kalman Estendida (EKF – *Extended Kalman Filter*) é a mais famosa e a mais utilizada para a estimação não-linear.

Entretanto, também apresenta limitações que agravam a sua utilização uma vez que apresenta comportamento errático e possivelmente divergente em sistemas bastante não-lineares ou em estimações iniciais demasiado imprecisas [2], [1], [3]. Assim, outros métodos foram desenvolvidos, como, por exemplo, a filtragem de Kalman *Unscented* (UKF –

Unscented Kalman Filter) e a filtragem de Kalman Pseudo-linear (PSELIKA – *Pseudo-linear Kalman Filter*).

Destes, a filtragem de Kalman Pseudo-linear chama bastante atenção pelo facto de ser aplicável para sistemas não-lineares, mas é modelado em uma estrutura pseudo-linear o que permite a utilização dos métodos de filtragem de Kalman linear, mas capturando as não-linearidades do sistema [4], [5].

1.2 Limitações dos trabalhos anteriores

No âmbito aeroespacial, a limitação do custo computacional e sua exatidão é de extrema importância pois há uma tendência cada vez maior de se utilizar instrumentos, sensores e métodos de estimação que sejam precisos, leves e com baixos custos computacional e financeiro [4]. Assim, caso seja obtido alguma forma de tornar a filtragem de Kalman Pseudo-linear menos custosa, a sua aplicação poderia ser considerada uma boa alternativa ao método EKF.

A limitação deste método é justamente a necessidade de resolver a equação Algébrica de Riccati associada ao modelo do sistema e realizar inversões matriciais para o cálculo da matriz de ganho do sistema para todos os instantes de tempo. Estes dois processos aumentam significativamente o custo computacional do método de filtragem.

Desta forma, uma ferramenta que tem sido integrada aos métodos de filtragem com o objetivo de melhorar o seu desempenho e reduzir o custo computacional é a rede neuronal artificial. Esta é uma função não-linear poderosa que tem sido estudada e desenvolvida ao longo dos anos sendo aplicada em diversas áreas que envolvem a utilização direta de dados. E por conta disto poderá auxiliar na redução do custo computacional do método de filtragem de Kalman Pseudo-linear.

As redes neuronais artificiais podem ser descritas como um conjunto de funções obtidas por um processo de aprendizagem obtendo conhecimento experimental e permitem calcular um valor de saída com base em um valor de entrada [6], podendo funcionar como aproximadores de funções não-lineares.

De forma a tentar reduzir a principal limitação do método de estimação PSELIKA, aproveitando a sua vantagem de apresentar uma estrutura pseudo-linear, esta dissertação propõe a integração de uma rede neuronal do tipo *Perceptron* de Múltiplas Camadas (MLP-*Multilayer Perceptron*) ao algoritmo PSELIKA, tendo como principal objetivo a redução do custo computacional original, implementando, por fim, um algoritmo denominado PSELIKA-MLP.

1.3 Objetivos

O objetivo principal desta dissertação é desenvolver um método de integração de uma rede neuronal à filtragem de Kalman pseudo-linear permitindo que seja aplicado à estimação em sistemas não-lineares, que representam as aplicações aeroespaciais, com a intenção de reduzir o custo computacional do método original de filtragem e manter a exatidão semelhante da estimação.

Este método é baseado no facto de que é possível aplicar os métodos lineares de estimação aos sistemas pseudo-lineares por apresentarem uma quase-linear com parametrização dependente do estado. Assim, possibilita o cálculo antecipado das matrizes de covariância do erro de estimação e consequentemente a de ganho, por meio da resolução da equação algébrica de Riccati associada ao sistema. Logo, por ser possível calcular estas matrizes a priori, é praticável utilizá-las como dados de treinamento para uma rede neuronal que terá como objetivo estimar o valor da matriz de ganho com base no vetor estado estimado da etapa anterior.

Para que este objetivo seja cumprido será necessário realizar as seguintes etapas fundamentais de análise:

Para o primeiro passo desse estudo, será realizada uma análise aprofundada de diversos métodos de filtragem não-linear baseados na filtragem de Kalman linear, como o EKF, UKF e o PSELIKA, com foco em suas principais vantagens, limitações e aplicações. Essa análise criteriosa permitirá identificar o algoritmo PSELIKA como a base para o desenvolvimento do algoritmo final.

Na segunda etapa, será realizado um exame aprofundado do funcionamento de diversas arquiteturas de redes neuronais, com o objetivo de identificar suas principais aplicações, vantagens e limitações. Essa análise abrangente permitirá escolher a rede do tipo MLP como uma boa solução para o trabalho em questão.

A seguir, será apresentada uma análise breve dos métodos existentes para integração de redes neuronais (NN – *Neural Network*) com algoritmos de filtragem, seguida da descrição detalhada das técnicas e do algoritmo implementado para integrar uma rede neural do tipo MLP ao algoritmo de filtragem PSELIKA, resultando no PSELIKA-MLP.

Para que esse algoritmo seja possível de ser implementado serão apresentadas algumas técnicas que incluem:

-Método de factorização de uma função não-linear para uma estrutura pseudo-linear;

-Definição do domínio do sistema para o treinamento de rede;

-Estrutura da rede neuronal aplicada a etapa de correção da filtragem;

-Treinamento e validação da rede neuronal.

A utilização do método proposto será validada nos seguintes casos de estudo de âmbito aeroespacial:

-Caso 1: Estimação da atitude e taxa angular de um veículo aeroespacial;

-Caso 2: Estimação da órbita de um satélite artificial.

1.4 Estrutura

Esta dissertação encontra-se dividida em cinco capítulos que estão organizados no seguinte formato:

Capítulo 1 – contempla a motivação e contexto desta dissertação assim como seus principais objetivos.

Capítulo 2 – está divido em duas partes essenciais. A primeira inclui os principais métodos de filtragem não-linear, assim como suas vantagens e desvantagens, e explicará o porquê da utilização do algoritmo PSELIKA como base para o trabalho. A segunda realizará uma introdução acerca das redes neuronais, assim como principais arquiteturas com vantagens e desvantagens, também justifica a utilização de MLPs no contexto de filtragem não-linear.

Capítulo 3 - introduz a união dos conceitos apresentados no Capítulo 2 e apresenta as técnicas principais e o algoritmo proposto nesta dissertação.

Capítulo 4 – são realizadas as simulações e seus resultados com vista a validar a utilização do algoritmo proposto em dois casos: estimação de atitude e taxa angular de um veículo aeroespacial e estimação da órbita de um satélite artificial.

Capítulo 5 – tem como objetivo realizar uma apresentação das conclusões acerca desta dissertação, assim como sugerir trabalhos futuros.

Capítulo 2 – Fundamentos

Para que haja a compreensão acerca de qualquer assunto, é necessário que o conhecimento adquirido pelos pesquisadores e cientistas ao longo história seja reconhecido, de forma a não cometer os mesmos erros do passado e caminhar pela trilha do conhecimento de forma eficiente e fundamentada.

Neste capítulo, serão introduzidos os conceitos fundamentais acerca de métodos de filtragem, em especial os de Kalman, e redes neuronais. Estes conceitos são cruciais para que o objetivo deste trabalho seja cumprido, permitindo a compreensão da escolha dos métodos selecionados para o capítulo 3.

2.1 Métodos de Filtragem de Kalman

A estimação precisa do estado de um sistema dinâmico a partir de medições ruidosas é um problema fundamental em diversas áreas, desde a engenharia de controle até o processamento de sinais abrangendo também a engenharia aeroespacial.

Em cenários reais, as medições obtidas através de sensores e instrumentos estão inevitavelmente sujeitas a erros e ruídos. Essa imprecisão pode ter diversas origens, como flutuações aleatórias, erros de calibração e limitações instrumentais. A presença de ruído nos dados de medição torna difícil a obtenção de uma estimativa precisa do estado real do sistema, afetando diretamente o desempenho e a confiabilidade de sistemas críticos.

Tendo em vista esse problema, diversos métodos de filtragem ou estimação foram desenvolvidos ao longo da história, implementando formas de reduzir o inevitável ruído e também estimar estados que não são observados diretamente.

Nesta secção, são apresentados alguns conceitos introdutórios à estimação de estados incluindo a introdução histórica de alguns dos principais métodos de filtragem, assim como seus algoritmos, vantagens e limitações, sendo estes o filtro de Kalman linear, o filtro de Kalman Estendido, o filtro de Kalman *Unscented* e o filtro de Kalman Pseudo-linear. Por fim, é feita uma introdução acerca das simulações de Monte Carlo e como estas podem ser utilizadas em conjunto com métodos para mensurar o desempenho de filtros.

2.1.1 Desenvolvimento sobre a filtragem de Kalman

O termo filtragem ou estimação é uma forma de tentar obter dados acerca de estados que não podem ser obtidos por medição direta, ou que não apresentem medições com precisão suficiente. O primeiro método foi proposto por Gauss e Legendre [7], em 1809 e 1806, de

forma independente, a partir do método dos mínimos quadrados, onde tentavam determinar a órbita de um corpo celestial a partir de medições. O objetivo não era encontrar o valor correto do estado, mas sim obter a estimativa mais provável, com base nas medições.

Com o objetivo de tentar obter a estimativa mais provável, Gauss antecipou o método da máxima verossimilhança, introduzido por Fisher em 1912 [8] sendo investigado continuamente. O método de Fisher permitiu a evolução da área de estimação servindo de base para os trabalhos Kolmogorov [9] em 1941 e Wiener em 1942 [10]. Estes desenvolveram independentemente métodos acerca da estimação média linear dos mínimos quadrados que eram muito semelhantes. Em seus trabalhos, os filtros necessitavam que ambos o estado e a medição estivessem de acordo com um processo estocástico estacionário.

O trabalho de Kolmogorov obteve solução para o tempo-discreto e o de Wiener para o tempo-contínuo [7]. A principal limitação destes filtros é a necessidade de resolver a equação de Wiener-Hopf durante o processo de filtragem, sendo necessários elevados recursos computacionais e espaço de armazenamento. Ainda assim, este filtro permitiu um grande avanço nos métodos de estimação estocástica, servindo de base para futuros trabalhos.

Com o objetivo de ultrapassar os limites existentes nos filtros de Wiener-Kolmogorov, em 1960, Kalman [11] propôs uma teoria de filtragem moderna em tempo-discreto e em 1961, com apoio de Bucy [12], a teoria para o tempo-contínuo. Em seu trabalho Kalman introduziu o conceito de espaço de estados na teoria de estimação estocástica, onde descreve as relações entre modelos de estado e medição, além da relação entre o estado estimado e as medições através de predições e correções.

O grande avanço de desenvolvimento é justamente o facto de que a filtragem de Kalman é um tipo de filtragem recursiva, ou seja, não requer informação de todas as etapas anteriores para calcular a etapa atual, apenas da anterior, reduzindo o problema de espaço de armazenamento e aumentando a velocidade de processamento. Além disso, a filtragem de Kalman pode ser aplicada a processos estocásticos não-estacionários [11], sendo considerado o estimador linear ótimo.

Ainda assim, a grande limitação deste método é justamente a sua limitação para sistemas lineares, além da necessidade de um modelo preciso do sistema e conhecimento das características estatísticas dos ruídos de processamento e medição, sendo assumidas gaussianas. Apesar das limitações, o Filtro de Kalman tem sido utilizado amplamente justamente pela fácil implementação, e diversas alternativas foram desenvolvidas de forma a tentar complementar e tornar o filtro mais robusto.

Na década de 1960, durante as missões Apollo, Kalman e Schmidt [13] desenvolveram um método de utilizar o conceito da filtragem de Kalman, porém em sistemas não-lineares, utilizando técnicas de linearização das funções que descreviam os modelos de sistema e medição [14]. Este método conhecido como Filtragem de Kalman Estendida (EKF – *Extended Kalman Filter*) foi bastante estudada sendo utilizada em diversas aplicações, incluindo no meio aeroespacial, tendo como alguns exemplos em [13], [15], [16]. Ainda assim, apesar de ter muito sucesso, este método de filtragem apresenta algumas limitações, uma vez que ocorre a linearização das matrizes, este processo introduz erros no sistema podendo levar a divergência do filtro.

Com a tentativa de contornar este problema, em 2000, Uhlmann, Julier e Durrant-Whyte [17] publicaram um novo método de estimação que não recorre a linearizações ou derivadas, conhecido hoje por Filtro de Kalman *Unscented* (UKF - *Unscented Kalman Filter*). Utilizando um novo conceito de pontos sigma os quais permitem a determinação da média e covariância do sistema, obtiveram resultados melhores que o EKF em diversas aplicações aeroespaciais, como em [17], [2], [18].

Em 1997, Bar-Itzhack e Harman [19], [20] desenvolveram o algoritmo da filtragem Pseudo-Linear de Kalman (PSELIKA - *Pseudo-Linear Kalman Filter*), com o objetivo de estimar a atitude e taxa angular de um veículo aeroespacial. A base deste algoritmo é a transformação de um sistema de equações não-lineares para uma estrutura pseudo-linear a partir de uma técnica de factorização utilizando a conversão para equações pseudo-lineares dependentes do estado. Esta técnica permitiu utilizar os conceitos de Filtragem de Kalman Linear para um sistema não-linear, o que também reduz os problemas enfrentados pelo EKF.

Há poucos anos, outros algoritmos de estimação foram desenvolvidos tentando superar o desempenho dos já mencionados. Em 2009 e 2010, Arasaratnam e Haykin [21], [22], desenvolveram o método de Filtragem de Kalman de Cubatura (CKF – Cubature Kalman Filter), baseado na análise Bayesiana em condições gaussianas e num modelo não-linear, sendo próximo ao filtro Bayesiano. Este método também foi aplicado na área aeroespacial, como em [23], reentrada na atmosfera, e [22], seguimento radar, onde o CKF superou os outros filtros utilizados.

Hoje, a utilização de métodos de filtragem tem utilizado como ferramenta de auxílio as redes neuronais. Em 2022, Revach, Ni, Escoriza, van Sloun e Eldar publicaram o seu trabalho sobre a KalmanNet [24], um filtro recursivo que utiliza uma rede neuronal recursiva (RNN – Recurrent Neural Network), com o objetivo de calcular a matriz de ganho de Kalman com pouco conhecimento acerca da estatística dos ruídos e do modelo do sistema.

Diversos outros métodos e variações foram desenvolvidos, entretanto, a bibliografia acerca é bastante extensa o que tornaria esta secção bastante alongada.

2.1.2 Filtragem Linear

A filtragem linear tem como objetivo estimar o valor de um estado a partir de uma medição corrompida por ruído de forma a reduzir o erro de estimação [1].

O termo filtragem refere a estimação do vetor estado no instante atual com base em todas as medições anteriores. O termo predição ou propagação refere-se à estimação do estado num instante futuro. Estes termos serão utilizados na demonstração a seguir do funcionamento do filtro de Kalman linear ou apenas filtro de Kalman (KF – *Kalman Filter*) [1].

Primeiramente, é preciso definir alguns termos de relevância para o entendimento do funcionamento do KF. A estimativa $\hat{x}$ é o valor computado de um estado x com base em um conjunto de medições z. Uma estimativa imparcial ou sem viés é aquela que seu valor esperado, ou esperança, apresenta o mesmo valor daquele do estado.

Um modelo de espaço de estados é aquele que pode ser descrito a partir de um conjunto de n equações diferenciais ordinárias de primeira ordem conhecidas por equações de estado [14], onde:

$$\dot{x}(t) = f(x, u, t) \tag{2.1}$$

Onde x é o vetor de estado de dimensão $n \times 1$, u é um vetor de entrada do modelo de dimensão $r \times 1$, onde o estado é função do tempo t, e f é a função que descreve o modelo podendo ou não ser linear.

Se o sistema for linear ou linearizado em relação a x e u então pode ser descrito pela seguinte equação:

$$\dot{x}(t) = Ax(t) + Bu(t) \tag{2.2}$$

Onde A e B são matrizes constantes que descrevem o sistema em torno de um ponto de linearização no instante t.

Partindo deste princípio, assuma que um conjunto de l medições, z, pode ser expresso como a combinação linear dos n elementos de um vetor constante x mais um erro aditivo de medição v. O processo de medição pode ser modelado por:

$$z = Hx + v \tag{2.3}$$

Onde $z, v \in \mathbb{R}^l$ e $x \in \mathbb{R}^n$, $H \in \mathbb{R}^{l \times n}$. Caso $l > n$ então o conjunto de medição apresenta informação redundante.

Uma possível abordagem é a estimação Bayesiana, onde modelos estatísticos estão disponíveis para x e z, e o objetivo é a obtenção de uma função de densidade condicional a posteriori, $p(x|z)$ já que contém todas as informações estatísticas de interesse. Partindo do teorema de Bayes:

$$p(x|z) = \frac{p(z|x)\, p(x)}{p(z)} \tag{2.4}$$

Onde $p(x)$ é a função de probabilidade de densidade a priori de x, e $p(z)$ é a função de densidade de probabilidade das medições. O termo $p(x|z)$ representa a função de probabilidade condicional de x em relação a z, ou seja, representa a probabilidade de x acontecer dado que z acontece. O objetivo é encontrar uma estimativa geral de variância mínima de Bayes, ou seja, minimizar a seguinte função de custo:

$$J = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \dots \int_{-\infty}^{\infty} (\hat{x} - x)^T S(\hat{x} - x)\, p(x|z) dx_1\, dx_2 \dots dx_n \tag{2.5}$$

Onde S é uma matriz semi-definida positiva arbitrária, caso seja feita a derivada parcial de J em relação a $\hat{x}$ e igualar a zero, é possível obter, independentemente de S que:

$$\hat{x} = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \dots \int_{-\infty}^{\infty} x\, p(x|z) dx_1\, dx_2 \dots dx_n = E[x|z] \tag{2.6}$$

Que se configura como a estimativa condicional média. Esta é conhecida como a estimativa ótima de Bayes [25]. Caso sejam assumidas distribuições gaussianas para x e v, o resultado de $E[x|z]$ em Eq. (2.6) é [1]:

$$\hat{x} = (P_0^{-1} + H^T R^{-1} H)^{-1} H^T R^{-1} z \tag{2.7}$$

Onde P_0 é a matriz de covariância inicial de x_0 dada por:

$$P_0 = E[(\hat{x}_0 - x_0)(\hat{x}_0 - x_0)^T] \tag{2.8}$$

E R é a matriz de covariância do erro associado ao vetor de medição dada pela expressão de esperança matemática $E[\cdot]$ por:

$$R = E[vv^T] \tag{2.9}$$

Comparando estes diferentes casos, se todos os erros de medição forem não correlacionados e apresentarem variância igual, para variáveis aleatórias gaussianas, todos esses métodos obterão os mesmos resultados [26].

Além disso, a variável $\hat{x}$ na Eq. (2.7) é uma operação linear dos dados de medição. É provado em [27] que para um sinal gaussiano variável ao longo do tempo, o preditor ótimo para erro mínimo quadrado médio é um preditor linear.

O KF, como será visto a seguir, acaba por ser um filtro recursivo do tipo linear, uma vez que se as variáveis consideradas apresentam distribuições gaussianas.

2.1.2.1 Filtros Recursivos

O chamado filtro de Kalman tem como objetivo principal a estimação de um vetor de estado corrompido por ruído. Este filtro é do tipo recursivo, ou seja, não necessita armazenar medições anteriores com o objetivo de calcular a estimativa atual, baseado na estimação Bayesiana. A descrição e equações que derivam o filtro estão escritas a seguir.

Considere um sistema linear contínuo, dado pelas seguintes equações:

$$\begin{aligned} \dot{x} &= Ax + w(t) \\ z &= Hx + v(t) \end{aligned} \tag{2.10}$$

O seu correspondente sistema linear discreto, com matriz de transição Φ, cujo estado no instante t_k é dado por $x(t_k)$ ou x_k , onde w_k é um vetor de ruído branco de média nula e distribuição gaussiana, de matriz de covariância Q_k, é tal que:

$$x_k = \Phi_{k-1}x_{k-1} + w_{k-1} \tag{2.11}$$

Apresentando como matriz de transição Φ_{k-1} do sistema do instante inicial t_{k-1} ao instante t_k, sendo T_s=t_k - t_{k-1},onde:

$$\Phi_{k-1} = e^{AT_s} \tag{2.12}$$

Neste caso A corresponde a matriz do modelo dinâmico contínuo do sistema, e Φ_{k-1} seria a sua representação no modelo discreto, para o instante t_{k-1}.

As medições são feitas como combinação linear das variáveis de estado do sistema, corrompido por ruído sem correlação. A equação da medição pode ser descrita em formato vetor-matriz como:

$$z_k = H_k x_k + v_k \tag{2.13}$$

Neste caso, z_k é o conjunto de l medições no instante t_k. H_k é a matriz de medição no instante t_k que descreve as combinações lineares das variáveis de estado que compreendem z_k na ausência de ruído. A dimensão desta matriz é $l \times n$, o que corresponde ao vetor de dimensão l das medições e o vetor de dimensão n do estado. v_k é um vetor de ruído branco de média nula e distribuição gaussiana, de matriz de covariância R_k, que corrompe as medições.

Para que seja possível realizar o processo de estimação é preciso que o sistema seja observável [28], ou seja, que para qualquer estado inicial x_0 e para qualquer tempo final finito $t > 0$, o estado inicial pode ser estimado exclusivamente a partir do conhecimento das entradas do sistema $u(t)$ e as saídas $z(t)$. Assim, considerando um sistema linear de ordem n, a matriz de observabilidade O_b é dada por:

$$O_b = \begin{bmatrix} H \\ HA \\ \vdots \\ HA^{n-1} \end{bmatrix} \tag{2.14}$$

O sistema só é observável se a seguinte condição for satisfeita:

$$rank(O_b) = n \tag{2.15}$$

Onde $rank(O_b)$ corresponde a característica matricial de uma matriz O_b, ou seja, ao número de linhas linearmente independentes da matriz.

Partindo do princípio de que o sistema é observável, o objetivo então é encontrar, para o instante t_k, o valor da estimativa $\hat{x}_k(+)$ (imediatamente depois da medição discreta) com base em uma estimativa anterior $\hat{x}_k(-)$ (imediatamente antes da medição discreta), e no valor da medição atual z_k. Em outras palavras, é feita uma combinação linear da previsão do estado $\hat{x}_k(-)$ com a medição atual z_k para obter a estimativa do estado final $\hat{x}_k(+)$, como pode ser visto na equação 2.16. De forma a não haver necessidade de armazenar os valores de todas as medições, o que levaria à uma carga computacional elevada, recorre-se a utilização de uma forma recursiva do estimador linear:

$$\hat{x}_k(+) = K_k' \hat{x}_k(-) + K_k z_k \tag{2.16}$$

Neste caso as duas matrizes $K_k{}'$ e K_k, são matrizes de ponderação que podem variar com tempo. Se w_k e v_k forem gaussianos, então o filtro será o filtro ótimo, e um filtro não-linear não será capaz de fazer melhor [11].

2.1.2.2 Filtro de Kalman Discreto

É possível derivar o filtro de Kalman realizando a otimização da forma assumida de estimador linear. O objetivo é simplesmente encontrar o valor da matriz de ganho K tal que o erro de estimação seja mínimo.

Por definição $E[v_k] = 0$. Se $E[\tilde{x}_k(-)] = 0$ (neste caso, $\tilde{x}_k(-)$ corresponde ao erro de estimação a priori da medição), então o estimador será sem viés. Se isto for válido, então obtém-se o seguinte [1]:

$$K_k' = I - K_k H_k \tag{2.17}$$

Assim, após manipulação algébrica o estimador assume o seguinte formato:

$$\hat{x}_k(+) = \hat{x}_k(-) + K_k[z_k - H_k\hat{x}_k(-)] \tag{2.18}$$

Nesta expressão, é possível perceber que a estimação se resume à soma entre a previsão do estado $\hat{x}_k(-)$ e o termo de correção $K_k[z_k - H_k\hat{x}_k(-)]$.

E, assim, o valor de estimação do erro pode ser obtido [1].

$$\tilde{x}_k(+) = (I - K_k H_k)\tilde{x}_k(-) + K_k v_k \tag{2.19}$$

Atualização da matriz de covariância dos erros ($\mathbf{P_k}$)

Sabendo o novo valor do erro de estimação é possível obter a expressão do novo valor da matriz de covariância dos erros. Pela definição:

$$P_k(+) = E[\tilde{x}_k(+)\ \tilde{x}_k(+)^T] \tag{2.20}$$

Da equação (2.19) tem-se, após manipulações algébricas [1], obtém-se:

$$P_k(+) = (I - K_k H_k)\ P_k(-)\ (I - K_k H_k)^T + K_k R_k K_k^T \tag{2.21}$$

Valor ótimo da matriz de ganho ($\mathbf{K_k}$)

O critério para escolher o valor de K_k é minimizar uma soma escalar ponderada dos elementos diagonais da matriz de covariância dos erros $P_k(+)$. Assim, para a função custo escolhe-se a seguinte função quadrática:

$$J_k = E[\tilde{x}_k(+)^T S \tilde{x}_k(+)] \tag{2.22}$$

Neste caso, S é qualquer matriz semi-definida positiva. Como demonstrado na equação (2.6), a estimativa ótima independe do valor de S, assim, é possível utilizar a própria matriz identidade, $S = I$, o que resulta em:

$$J_k = tr[P_k(+)] \tag{2.23}$$

A função $tr[\cdot]$ corresponde a função traço de uma matriz. Isto corresponde a minimizar o comprimento do vetor de estimação do erro. Para encontrar o valor de K_k que minimiza esta função, é necessário realizar a derivada parcial de J_k em relação a K_k, e igualar a zero. Tendo como base a seguinte expressão verdadeira [29], caso D seja uma matriz simétrica:

$$\frac{\partial}{\partial G}\left[tr(GDG^T)\right] = 2GD \tag{2.24}$$

Então das equações (2.21) e (2.22) o resultado é:

$$-2(I - K_k H_k)P_k(-)H_k^T + 2\,K_k R_k = 0 \tag{2.25}$$

E resolvendo para K_k, obtém-se:

$$K_k = P_k(-)\,H_k^T\left[H_k P_k(-) H_k^T + R_k\right]^{-1} \tag{2.26}$$

K_k é referido como matriz de ganho de Kalman.

Ao realizar manipulação da junção das equações (2.26) em (2.21) obtém-se:

$$P_k(+) \quad = \quad [I - K_k H_k]\,P_k(-) \tag{2.27}$$

O que corresponde ao valor otimizado da matriz de covariância dos erros atualizada.

Até então, o que foi descrito é todo o processo de estimação de estado descontínuo assim como o comportamento da matriz de covariância do erro ao longo de uma medição. A previsão ou propagação destas quantidades entre medições é dada por:

$$\hat{x}_k(-) = \Phi_{k-1}\hat{x}_{k-1}(+) \tag{2.28}$$

$$P_k(-) = \Phi_{k-1}\, P_{k-1}\,(+)\Phi_{k-1}^T + Q_{k-1} \tag{2.29}$$

Devido ao valor de P_k ser justamente a matriz de covariância dos erros de estimação, acaba por se tornar uma medição da precisão da estimativa.

Assim, resumindo todo o processo das equações da filtragem de Kalman linear, obtém-se a Tabela 2.1.

Modelo do Sistema	$x_k = \Phi_{k-1}x_{k-1} + w_{k-1},\ \ w_k \sim N(0, Q_k)$
Modelo de Medição	$z_k = H_k x_k + v_k, \qquad v_k \sim N(0, R_k)$
Condições Iniciais	$E[x(0)] = \hat{x}_0, E[(x(0) - \hat{x}_0)(x(0) - \hat{x}_0)^T] = P_0$
Outras Assunções	$E\left[w_k v_j^T\right] = 0\ para\ todos\ j, k$
Previsão da estimativa de estado	$\hat{x}_k(-) = \Phi_{k-1}\hat{x}_{k-1}(+)$
Previsão da Covariância do Erro	$P_k(-) = \Phi_{k-1}P_{k-1}(+)\Phi_{k-1}^T + Q_{k-1}$
Correção da Estimativa do Estado	$\hat{x}_k(+) = \hat{x}_k(-) + K_k[z_k - H_k\hat{x}_k(-)]$
Correção da Covariância do Erro	$P_k(+) = [I - K_k H_k]P_k(-)$
Matriz de Ganho de Kalman	$K_k = P_k(-)H_k^T\left[H_k P_k(-)H_k^T + R_k\right]^{-1}$

Tabela 2.1: Resumo das equações da filtragem de Kalman linear discreta.

2.1.3 Filtragem de Kalman Estendida

A filtragem de Kalman linear é considerada o estimador ótimo para sistemas lineares [11]. Entretanto, a maioria dos sistemas dinâmicos existentes tem caráter não linear. Como este método não pode ser implementado em sistemas lineares, diversos métodos foram desenvolvidos de forma a tentar contornar esse difícil problema de estimação não-linear.

Uma vez que a solução ótima Bayesiana do problema requer conhecimento completo da função da densidade de probabilidade das variáveis, resulta por não ser viável na prática [2]. Assim, diversos métodos foram desenvolvidos de forma a poder aproximar este estimador, como por exemplo o filtro de Kalman estendido que, como o próprio nome diz, é uma extensão do filtro já apresentado para problemas não-lineares [1], [14], [30].

Este filtro consiste na linearização das funções não lineares que representam o modelo dinâmico do sistema e o modelo de medição, partindo de aproximações da série de Taylor que define as funções não-lineares [14], [30], [31].

Sendo utilizado em diversas aplicações e devido justamente a sua facilidade de implementação e baixo custo computacional, este método de filtragem é indicado como primeiro tipo de estimador para diversos tipos de sistemas [1], [3]. Entretanto, este filtro deve ser utilizado com cuidado uma vez que estimações iniciais que se afastem muito do estado atual podem acarretar em divergência do filtro [2], [1], [3].

2.1.3.1 Algoritmo EKF

A estimativa de variância mínima é sempre a média condicional do vetor de estado, independentemente da sua função de densidade.

Suponha um sistema não-linear definido pela seguinte equação diferencial estocástica:

$$\dot{x}(t) = f(x(t), t) + w(t) \tag{2.30}$$

O problema a ser investigado desta vez é de estimar $x(t)$ a partir de medições não-lineares de formato:

$$z_k = h\big(x(t_k)\big) + v_k, k = 1, 2, \dots \tag{2.31}$$

A derivação das equações deste método de filtragem é um pouco extensa, mas pode ser observada em detalhe em [1].

Para que seja utilizado de forma prática em algoritmos de estimação são necessários métodos de cálculo da média do estado e matriz de covariância que não dependam da função de densidade de probabilidade. Um dos métodos é aplicar a série de Taylor em um estado $x(t)$ conhecido, com a truncagem efetuada nos dois primeiros termos, ou seja, são calculadas as matrizes jacobianas das funções não-lineares que descrevem o modelo do sistema e da medição.

Ao ser linearizada no ponto do vetor de estimativa do estado, estas equações apresentam estrutura similar as do Filtro de Kalman, para sistemas lineares, sendo assim, consideradas as equações de propagação do EKF.

Para a implementação deste filtro em um algoritmo a partir de medições efetuadas, pode-se utilizar o mesmo conceito implementado no Filtro de Kalman linear, onde era preciso que a correção da estimativa fosse uma função linear da medição.

As equações que constituem o algoritmo do Filtro de Kalman estendido para sistemas não lineares com medições discretas podem ser encontradas na Tabela 2.2 [1].

Modelo do Sistema	$\dot{x}(t) = f(x(t), t) + w(t); \quad w(t) \sim N(0, Q(t))$
Modelo de Medição	$z_k = h_k\big(x(t_k)\big) + v_k, \qquad k = 1,2, \ldots; \; v_k \sim N(0, R_k)$
Condições Iniciais	$x(0) \sim N(\hat{x}_0, P_0)$
Outras Assunções	$E\big[w(t)v_k^T\big] = 0 \; para \; todos \; k \; e \; t$
Previsão da estimativa de estado	$\dot{\hat{x}}(t) = f(\hat{x}(t), t)$
Previsão da Covariância do Erro	$\dot{P}(t) = F(\hat{x}(t), t)P(t) + P(t)F^T(\hat{x}(t), t) + Q(t)$
Correção da Estimativa do Estado	$\hat{x}_k(+) = \hat{x}_k(-) + K_k\big[z_k - h_k\big(\hat{x}_k(-)\big)\big]$
Matriz de Ganho de Kalman	$K_k = P_k(-)H_k^T\,\big(\hat{x}_k(-)\big)\big[H_k\big(\hat{x}_k(-)\big)P_k(-)H_k^T\big(\hat{x}_k(-)\big) + R_k\big]^{-1}$
Correção da Covariância do Erro	$P_k(+) = \big[I - K_k H_k\big(\hat{x}_k(-)\big)\big]P_k(-)$
Definições	$F(\hat{x}(t), t) = \dfrac{\partial f(x(t), t)}{\partial x(t)}\Big\vert_{x(t)=\hat{x}(t)}$ $H_k\big(\hat{x}_k(-)\big) = \dfrac{\partial h_k(x)}{\partial x}\Big\vert_{x=\hat{x}_k(-)}$

Tabela 2.2: Resumo das equações do filtro de Kalman Estendido contínuo-discreto.

A principal diferença entre o filtro convencional e a versão estendida é que a matriz de ganho K_k é na realidade uma variável aleatória que depende da estimativa $\hat{x}(t)$ através das matrizes $F(\hat{x}(t), t)$ e $H_k\big(\hat{x}_k(-)\big)$. Isso resulta do facto de que foi escolhido linearizar as matrizes f e h_k a partir da estimativa de $x(t)$. Assim, os valores de K_k devem ser calculados em tempo real não podendo ser pré-computados, sendo a precisão da estimação dependente da trajetória [1].

No caso da filtragem de Kalman Linear, é possível realizar o pré-cálculo das matrizes de ganho K_k quando as funções f e h_k são linearizadas em torno de um vetor de estado que é especificado antes do processamento dos dados mensurados [1].

Já que a matriz P_k nas equações da Tabela 2.2 é apenas uma aproximação da verdadeira matriz de covariância, não há garantia que a estimativa obtida será próxima da verdadeira estimativa ótima. Contudo, o EKF consegue realizar boas aproximações em várias aplicações práticas importantes [1]. Além disso, devido a similaridade ao próprio filtro de Kalman linear, e também em termos computacionais, este método tende a ser utilizado em primeira instância para os casos de filtragem não linear [1].

2.1.4 Filtragem de Kalman *Unscented*

O grande problema do EKF é justamente a sua imprevisibilidade na estimação, o que significa que apesar de ter sido desenvolvido para a estimação em sistemas não-lineares, poderá não apresentar bom desempenho nos mesmos [2]. Visto que conceito principal da filtragem EKF é a linearização do sistema em torno do estado estimado anterior, caso o estado se afaste demasiado do valor real, a previsão dos valores seguintes poderá divergir em casos onde a não-linearidade do sistema for elevada, podendo este problema ocorrer também no caso de uma escolha muito imprecisa para o estado inicial. [2]

Como visto anteriormente, a solução ótima para o problema de estimação é justamente a Bayesiana [32] através da descrição completa da função de densidade de probabilidade. Entretanto, não é possível na prática utilizar este modelo com base em um número finito de parâmetros. Desta forma, foram desenvolvidos diversos métodos de aproximação que permitem o cálculo prático de um estimador, porém não ótimo.

Além disso, sistemas que apresentam descontinuidades, ou sistemas com características discretas não podem desfrutar deste estimador [2], pois necessitam que a matriz Jacobiana exista.

Ainda assim, existe a possibilidade de ocorrerem erros no cálculo da própria matriz Jacobiana. Este processo gera muitos dados e é de dificuldade para depurar em termos de programação e validação, podendo também ser interferido por erro humano [2].

Portanto, o método de Transformação *Unscented* [17], [2] foi desenvolvido de forma a contornar os principais problemas enfrentados pelo EKF, provendo um mecanismo de cálculo mais explícito para tratar os dados de média e covariância presentes nestes estimadores.

2.1.4.1 Transformação *Unscented*

O conceito principal por trás da transformação *Unscented* (UT – *Unscented Transform*) é justamente ser mais fácil aproximar uma distribuição de probabilidade do que aproximar uma função ou transformação não-linear [33]. Neste caso, um conjunto de pontos (pontos sigma) são escolhidos de forma que sua média e covariância sejam $\bar{x}$ e P_x. A função não-

linear é então aplicada a cada ponto formando um conjunto. As estatísticas deste conjunto podem então ser calculadas formando uma estimativa da média e covariância desta transformação.

Os pontos sigma são selecionados de forma determinística, e não aleatória, de maneira que seu conjunto apresente propriedades estatísticas específicas. Assim, informação de elevada ordem acerca de uma distribuição pode ser obtida a partir de um número pequeno de pontos fixos.

Um conjunto de pontos sigma $\bar{S}$ consiste em $p + 1$ vetores e seus pesos (W) associados $\bar{S} = \{i = 0, 1, \dots, p: x_{UKF}^{(i)}, W_{UKF}^{(i)}\}$. Os pesos podem ser positivos ou negativos, mas devem obedecer a seguinte condição para fornecerem uma estimativa sem viés:

$$\sum_{i=0}^{p} W_{UKF}^{(i)} = 1. \tag{2.32}$$

Assim, tendo estes pontos, $\bar{z}$ e P_z são calculados da seguinte forma:

1) Primeiramente, é preciso instanciar cada ponto $x_{UKF}^{(i)}$ através da função de transformação não-linear f_{UKF}.

$$y_{UKF}^{(i)} = f_{UKF}\left[x_{UKF}^{(i)}\right]. \tag{2.33}$$

2) A seguir, a média é obtida através de uma média ponderada dos pontos transformados.

$$\bar{y}_{UKF} = \sum_{i=0}^{p} W_{UKF}^{(i)} y_{UKF}^{(i)}. \tag{2.34}$$

3) A covariância é então a soma do produto externo ponderado dos pontos transformados.

$$P_{UKF} = \sum_{i=0}^{p} W_{UKF}^{(i)} \left\{y_{UKF}^{(i)} - \bar{y}_{UKF}\right\}\left\{y_{UKF}^{(i)} - \bar{y}_{UKF}\right\}^T. \tag{2.35}$$

Os pontos sigma podem ser selecionados a partir do seguinte algoritmo, contendo $2n + 1$ pontos (ou $p = 2n$), sendo comum utilizar o seguinte algoritmo [17]:

$$\begin{aligned} x_{UKF}^0(t_k) &= \hat{x}_k(+) \\ W_{UKF}^0 &= \kappa/(n + \kappa) \end{aligned} \tag{2.36}$$

$$x_{UKF}^{i}(t_k) = \hat{x}_k(+) + \left(\sqrt{(n+\kappa)P_k(+)}\right)_i$$

$$W_{UKF}^{i} = \frac{1}{\{2(n+\kappa)\}}$$

$$x_{UKF}^{i+n}(t_k) = \hat{x}_k(+) - \left(\sqrt{(n+\kappa)P_k(+)}\right)_i$$

$$W_{UKF}^{i+n} = \frac{1}{\{2(n+\kappa)\}}.$$

Onde $\kappa \in R$ tal que $(n+\kappa) \neq 0$, $\left(\sqrt{(n+\kappa)P_k(+)}\right)_i$ é a i-ésima linha ou coluna da raiz quadrada da matriz de $(n+\kappa)P_k(+)$, e n é a dimensão do vetor de estado.

Uma representação da aplicação da UT pode ser visualizada na Figura 2.1 a seguir:

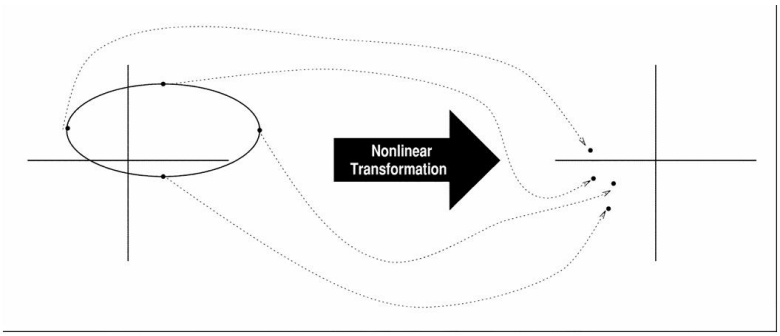

Figura 2.1: Representação da UT (imagem retirada de [2]).

Apesar de sua simplicidade, a UT apresenta algumas propriedades importantes [2]. Este método pode ser aplicado em praticamente quaisquer sistemas onde as entradas e saídas sejam bem definidas, até mesmo em transformações descontínuas [2]. O custo computacional pode ser da mesma ordem do EKF caso o número de pontos sigma projetados não seja muito grande [2].

Por fim, qualquer conjunto de pontos sigma que apresentem a média e covariância corretamente traduzem num valor projetado correto de média e covariância, até a segunda ordem. A estimativa introduz também termos de correção de viés de forma implícita sem a necessidade de calcular primeiras ou segundas derivadas [2].

Apesar de apresentar os valores corretos de primeiro e segundo momentos, existem diversas formas de escolher os valores dos pesos, alterando os momentos de ordem mais alta sem afetar os menores, até mesmo através da introdução de um novo ponto e peso no conjunto de pontos sigma. Nos anexos presentes em [17], [2], estão descritas diversas formas de se calcular e selecionar os pesos de forma a alterar os diversos momentos da distribuição destes pontos.

2.1.4.2 Algoritmo UKF

Para utilizar este algoritmo em conjunto com os métodos de filtragem, é preciso primeiro realizar um aumento do sistema a ser estudado [2], ou seja, acrescentar novos estados os quais se deseja observar de maneira a reestruturar os modelos de processamento e observação.

A formulação geral do Filtro de Kalman *Unscented*, no sistema aumentado pode ser visualizada abaixo [2].

1) O conjunto de pontos sigma é criado ao aplicar um algoritmo de seleção de p pontos sigma ao sistema aumentado dado em [2]. Em $x^{i}_{UKF,a}$, o índice a indica que o estado apresentado tem em conta o modelo aumentado do sistema que inclui os ruídos de processamento e de medição.
2) O conjunto transformado é dado ao instanciar-se cada ponto pelo modelo de propagação:

$$y^{(i)}_{UKF,a} = f_{UKF}[x^{i}_{UKF,a}(t_k), u_n] \tag{2.37}$$

3) A média prevista é calculada por:

$$\bar{y}_{UKF,a} = \sum_{i=0}^{p} W^{(i)}_{UKF} y^{(i)}_{UKF,a}. \tag{2.38}$$

4) E a covariância prevista é calculada por:

$$P_{UKF} = \sum_{i=0}^{p} W^{(i)} \left\{ y^{(i)}_{UKF,a} - \bar{y}_{UKF,a} \right\} \left\{ y^{(i)}_{UKF,a} - \bar{y}_{UKF,a} \right\}^{T}. \tag{2.39}$$

5) Instanciar cada um dos pontos de predição através do modelo de medição,

$$\hat{z}^{(i)}_{k} = h_{UKF}[y^{(i)}_{UKF,a}, u_k] \tag{2.40}$$

6) A observação prevista é calculada por:

$$\hat{z}_k = \sum_{i=0}^{p} W^{(i)}_{UKF} \hat{z}^{(i)}_{k} \tag{2.41}$$

7) A matriz de covariância da inovação é dada por:

$$\hat{R}_k = \sum_{i=0}^{p} W_{UKF}^{(i)} \left\{ \hat{z}_k^{(i)} - \hat{z}_k \right\} \left\{ \hat{z}_k^{(i)} - \hat{z}_k \right\}^T . \tag{2.42}$$

8) A matriz de covariância cruzada é dada por:

$$\hat{P}_k^{xy} = \sum_{i=0}^{p} W^{(i)} \left\{ y_{UKF}^{(i)} - \bar{y}_{UKF} \right\} \left\{ y_{UKF}^{(i)} - \bar{y}_{UKF} \right\}^T . \tag{2.43}$$

9) Finalmente, a correção pode ser dada pelas equações normais do filtro de Kalman:

$$\begin{aligned} x_k &= \bar{y}_{UKF} + K_k(z_k - \hat{z}_k) \\ P_k &= P_{UKF} - K_k \, \hat{R}_k K_k^T \\ K_k &= \hat{P}_k^{xy} \, \hat{R}_k^{-1} \end{aligned} \tag{2.44}$$

Ao observar o algoritmo de UKF é possível perceber que o custo computacional é proporcional ao número de pontos sigma [2].

O número mínimo de pontos sigma é justamente o requerido para obter a média e covariância. Desta forma, um espaço n-dimensional pode ser representado por um conjunto de $n + 1$ vértices [2]. Entretanto, um número maior de pontos pode ser selecionado para adicionar restrições as propriedades estatísticas deste conjunto.

O algoritmo UT apresenta uma simplicidade conceitual e facilidade de aplicação que o tornam atrativo no ponto de vista de aplicação. Similarmente à linearização no EKF, o UT se destaca por sua praticidade. No entanto, ao contrário da linearização, o UT oferece um nível de precisão superior, suficiente para ser aplicado em diversas áreas de filtragem e controle, mesmo quando a não linearidade se apresenta como um desafio significativo.

Ainda assim, com a demanda cada vez maior por métodos de estimação mais eficientes e rápidos utilizando meios de medição mais simples, o UKF acaba por não ser a melhor escolha em aplicações onde o esforço computacional não pode ser elevado em casos de utilização em tempo real, principalmente se o número de pontos sigma for necessariamente elevado (dimensão do estado elevada) para o mapeamento da distribuição das variáveis de estimação, visto a necessidade de realizar a raiz quadrada de uma matriz para obtê-los. Em casos onde isso não é problema, o UKF apresenta desempenho de estimação superior ao EKF sendo a melhor escolha entre os dois em diversas aplicações [17], [2].

2.1.5 Filtro de Kalman Pseudo-linear

O filtro de Kalman Pseudo-Linear (PSELIKA) é um filtro de Kalman linear ordinário onde os coeficientes dependentes do estado (SDC - *State-Dependent Coefficient*) são função da

melhor estimativa disponível. Este filtro tem como base fundamental o problema de estabilização ou regulação de um sistema não-linear utilizando a solução da Equação de Riccati dependente do estado [4], [34].

2.1.5.1 Regulação de Sistemas Pseudo-lineares

Este problema tem em sua formulação o seguinte sistema totalmente observável, autónomo, não linear nos estados e afim na saída, representado por [5]:

$$\dot{x}(t) = f(x) + B(x)u(t), x(0) = x_0 \qquad (2.45)$$

Onde $x \in \mathbb{R}^n$ é o vetor de estado, $u \in \mathbb{R}^m$ é o vetor de entrada do regulador, e $t \in [0, \infty)$, com funções $C_1(\mathbb{R}^n)$ $f: \mathbb{R}^n \to \mathbb{R}^n$ e $B: \mathbb{R}^n \to \mathbb{R}^{n\times m}, B(x) \neq 0\ \forall x$. Sem perda de generalidade, a origem $x = 0$ é assumida por ser um ponto de equilíbrio, tal que $f(0) = 0$. Nesse contexto, minimização do critério de desempenho para um tempo infinito [5]:

$$J(x_0, u(\cdot)) = \frac{1}{2}\int_0^\infty \{x^T(t)Q(x)x(t) + u^T(t)R(x)u(t)\}dt \qquad (2.46)$$

é considerado, sendo não quadrático em x mas quadrático em u. Neste caso, as matrizes de ponderação do estado e entrada são assumidas dependentes do estado tal que $Q(x): \mathbb{R}^n \to \mathbb{R}^{n\times n}$ e $R(x)$: $\mathbb{R}^n \to \mathbb{R}^{m\times m}$. Estas são escolhidas de forma que $Q(x) \geq 0$ e $R(x) \geq 0$ (semi-definidas positivas) para todos os x [5]. Assim, dentro das condições especificadas, a lei de controlo

$$u(x) = k(x) = -K(x)x, \qquad k(0) = 0, \qquad (2.47)$$

onde $k(\cdot) \in C_1(\mathbb{R}^n)$, é procurado de forma a tentar minimizar (2.46) com base na restrição (2.45) além de tentar aproximar o sistema para a origem para qualquer x, tal que $\lim_{t\to\infty} x(t) = 0$.

A linearização estendida [35], ou linearização aparente [36] ou parametrização SDC [37], [38] é o processo de factorização de um sistema não-linear para uma estrutura do tipo linear que contém matrizes SDC. Assumindo que $f(0) = 0$ e $f(\cdot) \in C_1(\mathbb{R}^n)$, uma função matricial contínua não linear $A(x)$ sempre existe, tal que

$$f(x) = A(x)x\,. \qquad (2.48)$$

Onde $A: \mathbb{R}^n \to \mathbb{R}^{n\times n}$ é encontrada a partir de factorização matemática e não única quando $n > 1$.

Assim, a linearização estendida da função afim do sistema não linear (1) torna-se:

$$\dot{x}(t) = A(x)x(t) + B(x)u(t), x(0) = x_0, \tag{2.49}$$

que apresenta uma estrutura pseudo-linear com matrizes $A(x)$ e $B(x)$ SDC.

A aplicação de qualquer método de síntese de controlo linear para a estrutura pseudo-linear (5), onde $A(x)$ e $B(x)$ são tratadas como matrizes constantes formam um método de controlo com linearização estendida. [5]

Ao imitar a formulação presente em nos reguladores lineares quadráticos (LQR- *Linear Quadratic Regulator*), o controlador de feedback de estados é obtido no formato:

$$u(x) = -R^{-1}(x)B^T(x)P(x)x \tag{2.50}$$

Onde $P(x)$ é solução única, simétrica e definida positiva da equação de Riccati Dependente do Estado (SDRE - *State-Dependent Riccati Equation*) [5]:

$$P(x)A(x) + A^T(x)P(x) - P(x)B(x)R^{-1}(x)B^T(x)P(x) + Q(x) = 0, \tag{2.51}$$

Ou em sua versão discreta [39], descrita a seguir, onde todas as matrizes dependem de x, o qual será omitido para concisão, e os termos com índice d referem-se as matrizes originais discretizadas, que são obtidas a partir do modelo da dinâmica contínua, que no caso seriam $A(x)$ e $B(x)$.

$$A_d^T P A_d - P - \left(A_d^T P B_d\right)\left(R + B_d^T P B_d\right)^{-1}(B_d^T P A_d) = 0 \tag{2.52}$$

Isto leva a terminologia controlo SDRE.

O benefício mais claro do algoritmo SDRE é a sua simplicidade e efetividade aparente. Quando o coeficiente e as matrizes ponderadoras são constantes, o regulador não linear colapsa para o problema de LQR, e o método SDRE de controlo torna-se um regulador linear estacionário. [5]

Devido a dualidade existente entre o LQR e o Filtro de Kalman, é possível estender o conceito de SDRE para a filtragem de Kalman [40], [31]. Há, em ambos os casos, a necessidade de resolver a Equação Algébrica de Riccati, sendo esta a solução ótima para o problema, e ao aplicar o conceito de factorização é possível transformar um sistema não-linear em uma estrutura pseudo-linear, podendo, desta forma, aplicar os conceitos de Filtragem de Kalman Linear. Assim, este método apresenta terminologia de Filtro de Kalman Pseudo-linear.

2.1.5.2 Algoritmo PSELIKA

Uma vez que é possível utilizar os métodos lineares em sistemas de estrutura pseudo-linear, então considere o seguinte sistema não linear contínuo com variável estocástica:

$$\dot{x} = f(x) + B(x)u(t) + w(t) \tag{2.53}$$

Neste caso, x representa o estado do sistema, $f(x,t)$ representa uma função matricial não linear, $u(t)$ o vetor do valor de entrada de um possível controlador no sistema e $w(t)$ o vetor do ruído de processamento, sendo neste caso branco do tipo gaussiano, e $Q(x)$ a sua respetiva matriz de covariância.

As medições desse sistema são feitas através de uma função não-linear da seguinte forma:

$$z = H(x) + v(t) \tag{2.54}$$

Onde z representa o vetor das medições, $H(\cdot)$ é uma função matricial não linear e v o vetor dos ruídos em relação a medição, sendo neste caso também branco do tipo gaussiano, e $R(x)$ a sua respetiva matriz de covariância.

Como mencionado anteriormente, é possível atribuir a esse sistema uma estrutura pseudo-linear, desde que $f(0) = 0$ e $H(0) = 0$, tal que:

$$f(x) = A(x)x \tag{2.55}$$

$$H(x) = C(x)x \tag{2.56}$$

Assim, ao substituir (2.55) e (2.56) em (2.53) e (2.54), obtém-se o seguinte sistema com estrutura pseudo-linear:

$$\begin{cases} \dot{x} = A(x)x + B(x)u(t) + w(t) \\ z = C(x)x + v(t) \end{cases} \tag{2.57}$$

Ao assumirmos a condição que em cada etapa de medição as matrizes A, B, C, Q e R serão tratadas como constantes, e que os ruídos de processamento e de medição são não correlacionados, este deixa de ser um problema de estimação não-linear e passa a se tornar o filtro de Kalman Linear. Realizando a discretização do sistema e utilizando os conceitos mencionados na secção de filtragem de Kalman Linear, obtém-se as equações para o PSELIKA discreto na Tabela 2.3.

Modelo do Sistema	$x_k = A_{d\,k-1}(x)x_{k-1} + B_{d\,k-1}(x)u_{k-1} + w_{k-1},\ \ w_k \sim N(0, Q_k)$
Modelo de Medição	$z_k = C_k(x)x_k + v_k, \qquad v_k \sim N(0, R_k)$
Condições Iniciais	$E[x_0] = \hat{x}_0, E[(x_0 - \hat{x}_0)(x_0 - \hat{x}_0)^T] = P_0$
Outras Assunções	$E\left[w_k v_j^T\right] = 0\ \forall\ j, k$
Previsão da estimativa de estado	$\hat{x}_k(-) = A_{d\,k-1}(x)\hat{x}_{k-1}(+) + B_{d\,k-1}(x)u_{k-1}$
Previsão da Covariância do Erro	$P_k(-) = A_{d\,k-1}(x)P_{k-1}(+)A_{d\,k-1}^T(x) + Q_{k-1}(x)$
Correção da Estimativa do Estado	$\hat{x}_k(+) = \hat{x}_k(-) + K_k[z_k - C_k(x)\hat{x}_k(-)]$
Correção da Covariância do Erro	$P_k(+) = [I - K_k C_k(x)]P_k(-)$
Matriz de Ganho de Kalman	$K_k = P_k(-)C_k^T(x)\left[C_k(x)P_k(-)C_k^T(x) + R_k(x)\right]^{-1}$

Tabela 2.3: Resumo das equações da filtragem de Kalman Pseudo-linear discreta.

A grande vantagem de utilizar este filtro, em relação ao EKF, é a sua robustez a erros grandes na determinação da estimativa inicial e disparidade na definição do modelo [19], sendo a sua contraparte o possível aumento do custo computacional [3], devido a necessidade da solução das equações de Riccati. Ainda assim, pelo facto de apresentar estrutura pseudo-linear é possível utilizar os métodos da filtragem de Kalman linear em sua formulação, o que permite um cálculo *off-line* das matrizes de ganho do sistema.

Assim, o PSELIKA pode ser um bom método para aplicações onde o tempo de cálculo e recursos devem ser reduzidos, mas mantendo boa precisão na estimação, uma vez que há o compromisso entre capturar as não-linearidades do sistema sem a necessidade de realizar cálculos extensos, resumindo-se ao esforço computacional presente na filtragem de Kalman Linear [19].

2.1.6 Simulação de Monte Carlo

Para a formulação e teste dos diferentes tipos de filtros, é preciso utilizar algum meio que sirva como base para avaliação do desempenho. É possível realizar esta avaliação a partir

de principalmente dois meios, utilizando simulações numéricas ou a partir de dados recolhidos de experimentos, que serviriam como referência. De maneira a comparar o desempenho de dois ou mais filtros é necessário equiparar as condições iniciais para todos, principalmente em relação a estimativa inicial, para a análise de convergência [3].

Tendo como principal possível origem o trabalho dos matemáticos J. Neyman e S. Ulam em 1949 [41], o método de Monte Carlo tem seu nome oriundo da famosa cidade de Monte Carlo, no Principado de Mônaco, pela presença de casas de aposta.

A base teórica do método tem como principal conceito a seleção de quantidades aleatórias. A grande dificuldade enfrentada era justamente a realização desta seleção que sem o auxílio de computadores eletrónicos tornava-se uma tarefa extremamente extenuante. Assim, a sua aplicação prática só foi possível após o aparecimento de computadores [42].

O principal objetivo de uma simulação de Monte Carlo é justamente simular numericamente as condições representadas por algum cenário composto por variáveis aleatórias que seguem algum tipo de distribuição de probabilidade. Desta forma, é possível determinar o resultado de algum sistema ou cenário que contenha variáveis aleatórias [43].

As simulações presentes no caso de filtragens apresentados são ótimos exemplos para a aplicação da simulação de Monte-Carlo. Nestas, considera-se a presença de ruídos de processamento e medição onde se assume uma distribuição normal ou gaussiana (ruído branco), uma vez que esta aproxima-se de diversas situações da natureza, pelo teorema do limite Central, presente na Teoria da Probabilidade, desenvolvida inicialmente por Laplace [42]. Em conjunto com estes ruídos, a estimativa inicial pode ter bastante influência no comportamento e desempenho do filtro, como já foi mencionado nas secções anteriores.

Supondo a necessidade de apenas comparar os modelos em termos de convergência para o máximo de diferentes condições iniciais, deve-se avaliá-las de igual maneira, ou seja, supondo que todas possam acontecer com a mesma probabilidade. Assim, neste caso, a escolha de uma distribuição uniforme para a função de densidade de probabilidade é adequada para o estado inicial, visto a sua simplicidade e a necessidade de avaliar as diferentes possibilidades dentro de um intervalo. Um exemplo da utilização deste método para a comparação de filtros pode ser visualizado em [3].

No caso de os modelos apresentarem a mesma estimativa inicial, a comparação pode ser efetuada tendo como principal causa os ruídos introduzidos no sistema, verificando o desempenho dos filtros em reduzir ao máximo o erro de estimação. Desta forma, pode-se efetuar diversas simulações que poderão apresentar diferentes valores de ruídos, com base em estados de referência, o que leva a mensurar o valor do erro médio através de uma função

de desempenho com maior precisão. Este método de comparação pode ser visualizado em [17].

2.1.7 Desempenho de um Estimador

O desempenho de um estimador pode ser calculado a partir da utilização de diversas funções. As principais são a partir do erro quadrado médio (MSE- *Mean Squarred Error*) e da raiz do erro quadrado médio (RMSE – *Root Mean Squarred Error*), calculados a partir de valores de referência que são comparados com os valores estimados. Estes métodos são derivados da distância Euclidiana, sendo sempre um valor positivo onde seu valor se aproxima a zero quando o erro diminui.

Tem como objetivo medir a média do quadrado da diferença entre uma estimativa e o valor real. Se o vetor de m predições é gerado a partir de um conjunto de m pontos e Y_i é o vetor dos valores observados da variável sendo prevista, com $\hat{Y}_i$ sendo os valores previstos então definida pela seguinte equação:

$$MSE = \frac{1}{m}\sum_{i=1}^{m}\left(Y_i - \hat{Y}_i\right)^2. \tag{2.58}$$

O $RMSE$ é justamente a raiz do MSE, sendo análogo ao desvio padrão:

$$RMSE = \sqrt{\frac{1}{m}\sum_{i=1}^{m}\left(Y_i - \hat{Y}_i\right)^2} \tag{2.59}$$

Ambos métodos são utilizados atualmente como comparadores de desempenho entre estimadores [44], [3], [17]. Entretanto, para o caso deste trabalho será selecionado o RMSE justamente pela sua interpretação facilitada já que apresentaria as mesmas unidades que o valor previsto [45].

2.2 Redes Neuronais Artificiais

A tendência da utilização das redes neuronais artificiais tem aumentado cada vez mais ao longo das últimas décadas, observando-se cada vez mais a utilização destas como ferramenta de auxílio para a resolução de problemas que antes pareciam impossíveis em diversas esferas da ciência.

Nesta secção, são apresentados alguns conceitos introdutórios às redes neuronais artificiais que incluem uma breve história por trás do seu desenvolvimento, as principais arquiteturas

com as suas vantagens e limitações e o porquê da seleção da rede neuronal do tipo *Perceptron* de Múltiplas Camadas para este trabalho, assim como seu algoritmo. Por fim, serão apresentados os principais métodos de treinamento de redes neuronais e aquele selecionado. Todos estes conceitos são de extrema importância para o entendimento do algoritmo final implementado para este trabalho.

2.2.1 Conceito da Rede Neuronal Artificial

Rede neuronal artificial, ou simplesmente rede neuronal (NN), como descrita em [46], é um processador constituído por diversas unidades de processamento mais simples distribuídas em paralelo, sendo capaz de armazenar e retornar conhecimento experimental. Em outras palavras, é um conjunto de funções obtidas por um processo de aprendizagem que permitem calcular um valor de saída com base em um valor de entrada [6].

O termo rede neuronal deve-se ao facto de ser constituída por unidades, também conhecidas como neurónios, que podem conectar-se aos valores de entrada, aos valores de saída ou até mesmo a outros neurónios. De forma análoga ao funcionamento dos neurónios num cérebro, os quais funcionam a partir de sinais elétricos, utilizam a força ou o peso dessas conexões para guardar o conhecimento adquirido de um ambiente de trabalho a partir de um processo de aprendizagem [46].

Na Figura 2.2 é possível observar a representação de um neurónio biológico. Um conjunto de neurónios biológicos formam uma rede neuronal biológica.

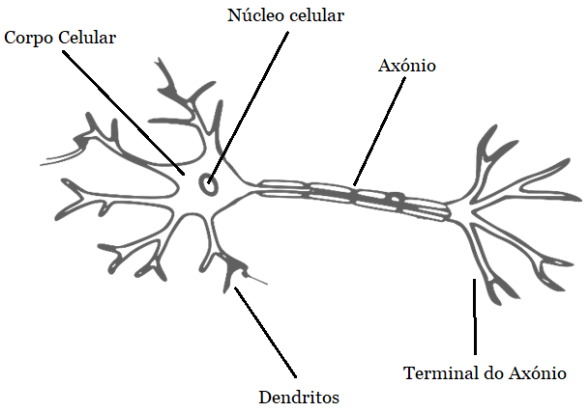

Figura 2.2: Representação de um neurónio biológico e principais elementos constituintes.

Os principais elementos de um neurónio biológico são o corpo celular, o núcleo celular, os dendritos e o axónio [47] como pode ser visto na Figura 2.2. O corpo celular abriga o núcleo celular, que por sua vez contêm os demais componentes principais celulares como a mitocôndria e o Complexo de Golgi [47]. Os dendritos permitem a receção de sinais elétricos de entrada através de conexões sinápticas com outros neurónios [47]. O axónio funciona

como o canal de saída do neurónio enviando sinais elétricos para os dendritos de outros neurónios [47].

A Figura 2.3 mostra a representação de um elemento básico de uma rede neuronal artificial, o *perceptron*, que apresenta funcionamento e estrutura análoga a de um neurónio biológico.

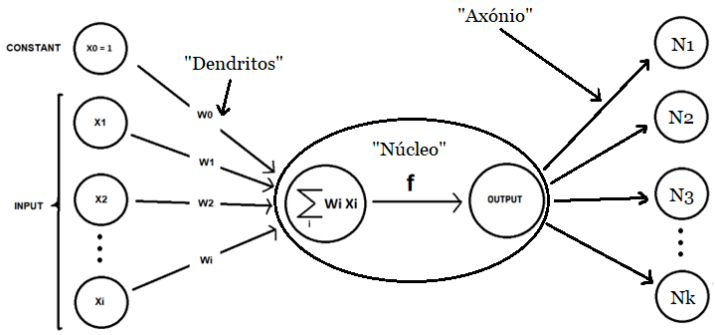

Figura 2.3: Representação da estrutura de um neurónio artificial.

Os "dendritos" se referem aos pesos de conexão, ou "sinapses", com todos os outros neurónios que se conectam a este, representando a entrada do neurónio. O núcleo se refere a operação realizada ao receber todos os sinais dos "dendritos", que seria a soma ponderada dos sinais seguida da aplicação de uma função de ativação. Por fim, o resultado da função de ativação é enviado pelo "axónio" através de "sinapses", que também apresentam um peso de conexão e este representa a saída do neurónio. Esta analogia é muito interessante pois o ser humano tem buscado entender e replicar o funcionamento de um neurónio ao longo da história. Esta história assim como o funcionamento de uma rede neuronal artificial serão mais bem explicados nas subsecções a seguir.

2.2.1.1 Desenvolvimento sobre as Redes Neuronais Artificiais

O ser humano sempre foi fascinado pelo entendimento e compreensão acerca de tudo ao seu redor e também sobre si mesmo, principalmente sobre o funcionamento do seu próprio cérebro. Com a evolução da tecnologia dos computadores durante o começo do século XX diversos pesquisadores desenvolveram sistemas com o intuito de simular as interações que acontecem entre os neurónios, baseados na similaridade entre estes e as unidades do tipo *on-off* [48], presentes em todos os computadores.

Assim, diversos pesquisadores desenvolveram modelos de cérebros que eram capazes de realizar algoritmos simples em resposta a uma sequência de estímulos. Um desses exemplos foi os princípios e modelos desenvolvidos por Rashevsky (1938) [49] com o objetivo de simplificar processos complexos presentes no corpo humano.

McCulloch e Pitts (1943) [50] são conhecidos pelo seu pioneirismo nas bases das NNs utilizando modelagem lógica e circuitos elétricos. Culbertson (1950) [51], Kleene (1956) [52]

e Minsky (1956) [53] também fizeram contribuições significativas para os primeiros modelos neurais.

Outros pesquisadores estavam mais preocupados no entendimento do funcionamento do sistema nervoso biológico e a sua atividade num ambiente natural. Alguns nomes se destacam como Hebb [54] e Hayek [55].

Destes, Hebb (1949), em seu livro "*The Organization of Behavior*" [54], propôs o famoso Postulado de aprendizagem de Hebb, o conceito base para os processos de aprendizagem em redes neuronais [46]. Neste postulado ele afirma que a ativação simultânea de dois neurónios aumenta a força de conexão entre eles. Este conceito foi fundamental para o desenvolvimento da neuromodelagem e, consequentemente, da compreensão do comportamento humano.

Inspirado no livro de design cerebral de Ashby (1952) [56] e em diversos outros estudos acerca do sistema nervoso, Rosenblatt (1958) [48] propôs o *perceptron*, um modelo teórico para um neurónio artificial [48]. Este deu origem ao *perceptron* de múltiplas camadas (MLP) [48], um modelo de rede neuronal utilizado até hoje. Rosenblatt também formulou a "regra de aprendizagem do *perceptron*" para treinar o modelo [57]. Para o MLP, ele introduziu a noção de erros retropropagados, mas não conseguiu formular um algoritmo de treinamento completo baseando-se na tentativa e erro para o treinamento.

Outros pesquisadores chegaram a conclusões similares com circuitos parecidos aos *perceptrons* [58]. Widrow e Hoff (1960), com os circuitos ADALINE e MADALINE [59], capazes de reconhecer padrões visuais e palavras faladas, e Steinbuch (1961) [60], utilizou matrizes em circuitos para aprender padrões.

Em 1969, Minsky e Papert [61] publicaram um livro que criticava severamente os *perceptrons*, que eram considerados na época como as máquinas de aprendizado e reconhecimento de padrões [58]. A principal crítica era a incapacidade dos *perceptrons* de realizar algumas tarefas simples como, por exemplo, o *Exclusive-Or* (XOR) o que levou a dúvida acerca da capacidade de aprendizagem.

Combinando este trabalho com as falhas em experimentos de redes neurais (como tradução automática), o trabalho de Minsky e Papert levou a uma estagnação da pesquisa sobre o tema nos Estados Unidos da América [58]. Mesmo assim, esse campo da ciência manteve-se bastante ativo em países europeus e asiáticos [62].

Contudo, durante este período, descobertas permitiram que as NNs retornassem nos âmbitos de pesquisas. A adição de um elemento *perceptron* entre as unidades de entrada e

saída (MLP) permitia resolver problemas como a operação XOR [58]. Neste os valores dos pesos eram obtidos pelo método ineficiente de tentativa e erro [57], [63].

Ivakhnenko e Lapa (1965) desenvolveram o algoritmo de aprendizagem para MLPs [64], conhecido por aprendizado profundo ou *Deep Learning*. Em 1967, Amari sugeriu a implementação do método gradiente descendente estocástico (SGD - *Stochastic Gradient Descent*) [65] para o aprendizado profundo. O método SGD foi desenvolvido em 1951 por Robbins and Monro [66].

Em 1970, Linnainmaa [67] desenvolveu o algoritmo de retropropagação em seu trabalho de dissertação, sendo a seguir aplicada por Werbos (1974) [68] em NNs, tornando-a uma peça fundamental para a etapa de treinamento. Este algoritmo é um dos métodos mais famosos de treinamento de MLPs [58].

No início dos anos 80, dois eventos em paralelo ocorreram de forma a fomentar novamente o campo de pesquisa das NNs, além do avanço da computação que se tornou muito mais acessível se comparada com a década anterior.

Em 1982, o físico John Hopfield publicou um artigo [69] mostrando como que uma rede de neurónios simples poderia ter a capacidade de calcular utilizando teoria matemática próxima de termodinâmica.

Em 1986, Rumelhart, em conjunto com o famoso grupo de estudo PDP (PDP- *Parallel Distributed Processing*), publicou dois artigos revolucionários sobre NNs [70], o que levou a impulsionar esta área de estudo nos EUA e abriu o caminho para mais investimentos e diversas outras aplicações.

Assim, após estas publicações, as NNs tiveram sucesso em tarefas como a geração de fala artificial [71] e sendo capaz de conduzir veículos [72], o que estimulou ainda mais o desenvolvimento de novos modelos e aplicações, e isso se mantém até hoje.

Em 1991, a geração de texto mais próxima à linguagem humana deu um passo a frente a partir dos *Transformers* [73]. Este método utiliza autoatenção linearizada [74], [75] que permite os *Transformers* processarem dados e ajustar pesos autonomamente sendo subsequentemente mais eficientes. Assim, abriu as portas para o desenvolvimento de modelos de linguagem avançados, como os *Transformers* Generativos Pré-treinados (GPT - *Generative Pre-Trained Transformer*) [76].

Também em 1991, foram desenvolvidas técnicas de aprendizagem para redes neuronais recorrentes (RNN - *Recurrent Neural Network*) profundas de forma a aprenderem autonomamente [77], [78].

Considerado o maior problema em relação ao treinamento de NNs profundas, o Problema fundamental do aprendizado profundo, conhecido por *Vanishing Gradients* ou desvanecimento de gradientes, foi analisado e resolvido em 1991 [79]. Esse problema ocorre em NNs profundas ou RNNs quando os erros retropropagados acabam se reduzindo muito rapidamente ou crescem fora dos limites, o que acarreta na falha do processo de aprendizagem [80].

O problema foi resolvido com o desenvolvimento das RNNs do tipo memória de longo prazo (LSTM - *Long Short-Term Memory*) [81]. Este modelo tem sido aplicado como base para diversas arquiteturas até hoje, como o *Tensorflow* de Google para o reconhecimento de fala [62]. É capaz também de aprender tarefas simbólicas, como tradução entre linguagens, servindo como peça fundamental para tradução em tempo real pelo Google *Translate* [82]. Também tem apelo na medicina, principalmente na área de diagnósticos [77].

Hoje, a técnica mais utilizada para aprendizagem é o Aprendizado Reforçado (RL-*Reinforcement Learning*) [83], onde a NN aprende a interagir com um ambiente dinâmico parcialmente observável, sem nenhum conhecimento a priori, com base em um sistema de recompensas. Este método permitiu o desenvolvimento das NNs DeepMind [84] e OpenAI Five [85], capazes de vencer jogadores profissionais em jogos eletrónicos.

O principal desenvolvimento mais recente na área das redes neuronais [86] é o ChatGPT, desenvolvido no final de 2022 com base em modelos base GPT, sendo responsável pela atual profusão de modelos de Inteligências Artificiais. É um modelo de linguagem grande (LLM-*Large Language Model*), que permite o seu utilizador conduzir uma conversa natural e refiná-la com base em comprimento, formato, estilo, nível de detalhe e linguagem. [87].

2.2.1.1.1 Classificação de Redes Neuronais

Depois do sucesso, as NNs começaram a ser implementadas em diversas aplicações, sendo cada arquitetura utilizada em diferentes aplicações, cada uma com suas vantagens e limitações.

Atualmente, os principais modelos de redes neuronais desenvolvidos e os mais utilizados são as de alimentação direta de multicamadas (MLFFNN - *Multilayer Feedforward Neural Network*) e as recorrentes (RNN) [88].

É importante mencionar, assim como na secção acerca da história, que existem diversos outros modelos de redes neuronais os quais não serão mencionados, mas podem ser encontradas a partir das referências bibliográficas utilizadas.

As redes MLFFNN têm como principais vantagens a sua elevada gama de aplicações [89], [90], com uma estrutura simples e de fácil implementação, se comparada com outros

modelos [46], [77], sendo robusta e com bom desempenho em vários casos [91], [92], [93]. No caso de problemas de aproximação de funções são preferidas em casos onde não haja picos e vales regulares [94]. A sua grande limitação é justamente a necessidade de um grande número de dados de pares de entrada e saída para o processo de treinamento [95], [96].

Redes neuronais recorrentes são um tipo poderoso de rede neural que pode lidar com dados sequenciais [97]. Eles são computacionalmente eficientes para tarefas que envolvem processamento temporal e podem ser usados em diversas aplicações, como interação humano-robô [98], [99], [100], [101].

As RNNs têm a principal vantagem de serem capazes de aproximar relações complexas entre sequências de entrada e saída, tendo a propriedade de serem o aproximador universal, capazes de aproximarem sistemas dinâmicos não-lineares arbitrários [102]. No entanto, o treinamento de RNNs pode ser desafiador devido a problemas de nulificação ou divergência de gradientes [80], e sua eficácia pode ser limitada pela função de ativação usada [90].

2.2.1.1.2 Aplicações de Redes Neuronais

As redes neuronais podem ser separadas em dois grupos os quais apresentam ou não um processo de aprendizado supervisionado. O aprendizado supervisionado refere-se a todas as NNs em que se quer modelar um conjunto de dados com base em atributos descritivos adicionais [103], enquanto o não supervisionado utiliza dados de entrada sem uma função de saída, com o objetivo de agrupar dados com base em similaridades sem atributos adicionais [104].

Dentro do aprendizado supervisionado dois grandes grupos se destacam com base no tipo de aplicação, os quais são o de regressão e de classificação. Para os problemas de classificação o objetivo é aproximar a variável de saída a um conjunto finito de possibilidades, ou valores discretos, como num classificador binário onde a saída é 0 ou 1. Já nos problemas de regressão, a variável de saída tem domínio contínuo, com um número infinito de possibilidades. Um exemplo é a previsão do tamanho de um salmão com base em sua idade e peso [104].

Tendo em vista minimizar o custo computacional e complexidade, tanto de implementação como treinamento, para o objetivo deste trabalho que é a predição ou modelação da matriz de ganho da filtragem de Kalman com o mínimo custo computacional, este torna-se um problema de regressão e, por isso, o modelo de rede escolhido para ser estudado e aplicado em foco é o MLFFNN, apesar das RNNs apresentarem o potencial de melhores resultados em sistemas não-lineares.

2.2.2 Redes Neuronais *Feedforward*

O principal objetivo das redes *feedforward* é aproximar uma função que pode ou não ser linear, com base em um conjunto de valores de entrada utilizados para treinamento. Neste caso, as conexões realizadas são feitas de modo que a saída do neurónio de uma camada sempre se conecte com neurónios da camada seguinte, ou seja, a jusante do fluxo de dados.

No modelo *feedforward*, os valores de entrada passam por um processo computacional intermédio, obtendo valores de saída, sendo o fluxo de informação nesta ordem direta. Além disso, a rede apresenta uma estrutura neuronal estratificada, ou seja, é composta por camadas que contém os neurónios. Os parâmetros que definem a estrutura e alteram o treinamento de uma rede são chamados de hiper-parâmetros. A Figura 2.4 representa um exemplo de uma NN do tipo *feedforward* e seus principais componentes, e é possível perceber os principais elementos da Figura 2.3 para cada neurónio.

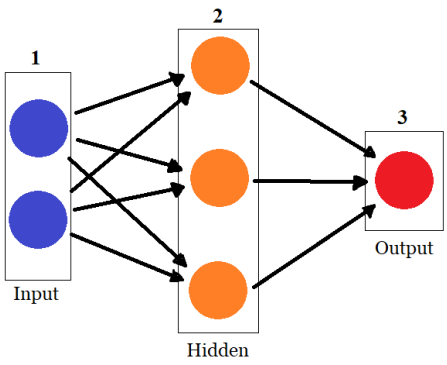

Figura 2.4: Representação de rede neuronal artificial.

A rede da Figura 2.4 apresenta três camadas numeradas: camada 1 de entrada (em azul), camada 2 escondida (em laranja) e camada 3 de saída (em vermelho). O número de camadas define a profundidade da rede. Cada camada apresenta um número diferentes de neurónios sendo 2, 3 e 1 respetivamente. As setas e o sentido representam as conexões e sentido de fluxo de dados dentro da rede.

2.2.2.1 *Perceptron* de Múltiplas Camadas

A aplicação da rede no contexto deste trabalho será de obter uma função capaz de realizar uma regressão não-linear a partir de um conjunto de dados obtidos a priori. O modelo de rede que será implementado será o *perceptron* de múltiplas camadas, ou *MultiLayer Perceptron* (MLP). Este é um caso específico das MLFFNN onde todas as camadas escondidas apresentam a mesma função de ativação [105], simplificando o processo de aprendizagem.

Um dos principais constituintes do modelo MLP é o *perceptron*, ou *perceptron* de camada única (SLP - *Single-Layer Perceptron*). A estrutura de um *perceptron* pode ser observada

na Figura 2.5. O *perceptron* é um algoritmo para aprendizado supervisionado, análogo a um neurónio que realiza duas etapas de cálculo [46], [61].

Primeiro, recebe dados de entrada e um valor constante e realiza a soma ponderada destes valores utilizando pesos que compõem as conexões entre os neurónios. A seguir, o resultado é utilizado como argumento de uma função de ativação não-linear F_A para obter a saída. A principal aplicação de um SLP é ser utilizado como um classificador binário [61]. A operação realizada por um *perceptron* é dada a seguir por:

$$\zeta = \sum_i (W_i \, \eta_i) + b \tag{2.60}$$

Onde ζ representa o valor de saída antes da aplicação da função de ativação, W_i corresponde ao valor do peso atribuído ao valor de entrada η_i, e b corresponde ao valor constante que será tratado como o viés (*bias*) do perceptron. Este valor pode ser observado como W_0 na Figura 2.5. A saída a encontra-se após a aplicação da função de ativação F_A(dado na Figura 2.5 apenas por f).

$$a = F_A(\zeta) \tag{2.61}$$

Para ser utilizado como classificador binário é preciso utilizar funções de ativação específicas que permitem a seleção de uma classe, como 0 ou 1, neste caso, a partir de um limite que separa as condições 0 ou 1. As funções mais comummente utilizadas, pois apresentam imagem limitada por limites assintóticos, são a de polo-único sigmoide e a bipolar sigmoide [106].

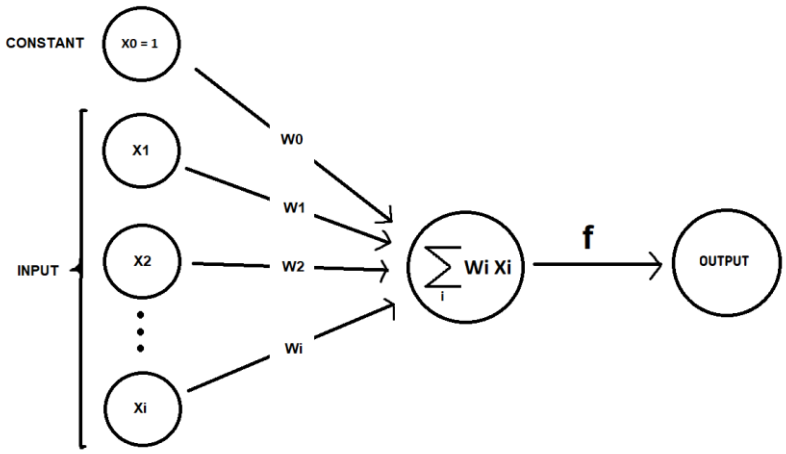

Figura 2.5: Estrutura de um *Perceptron*.

A utilização destas duas funções de ativação mencionadas não apresentaria bons resultados para este trabalho, já que são funções com imagens limitadas. Além disso, a não utilização

de funções de ativação faria com que a saída da rede fosse apenas uma combinação linear dos valores de entrada, tornando esta um regressor linear.

O *perceptron* é um tipo de rede neuronal com apenas duas camadas: a de entrada e a de saída. Geralmente, para efeitos de classificação, enumeram-se apenas as camadas em que há transformação dos dados, por isso esta é chamada de *perceptron* de camada única. Se o valor de saída deste for utilizado como valor de entrada para outro *perceptron* então obtém-se uma NN de três camadas, com uma camada de entrada, uma de saída e uma escondida intermédia, de largura $l = 1$, ou seja, seria um *perceptron* de duas camadas.

O principal problema da utilização de SLPs é que estes são apenas capazes de aprender padrões linearmente separáveis [107]. Isto significa que só seria capaz de fazer a separação de dados por um plano, ou hiperplano, no caso de uma classificação.

O termo *perceptron* de múltiplas camadas é utilizado justamente para os casos em que o número de camadas totais, incluindo a de entrada, é de pelo menos três. Além disso, pode-se utilizar o termo Rede Neuronal Profunda (*Deep Neural Network*) para estes casos, e seu processo de aprendizado como "Aprendizado profundo", ou *Deep Learning*.

O termo aprendizado é utilizado para a atualização dos pesos presentes nas conexões entre neurónios com base numa função de desempenho e dados, de forma que a rede tente se aproximar ao máximo da função objetivo. No caso das MLPs, o algoritmo de aprendizado mais famoso é o de retropropagação ou *backpropagation*.

2.2.2.1.1 Algoritmo do MLP

O algoritmo do *Perceptron* de Múltiplas Camadas é bem simples e descrito da seguinte forma:

Se for seguido a seguinte notação o valor do peso entre o neurónio i da camada l e o neurónio j na camada $l-1$ por W_{ij}^{l} e o correspondente viés por b_i^l, então o cálculo da soma ponderada como entrada para o neurónio i na camada l é dado por:

Passo 1 – Os outputs dos neurónios da camada escondida são calculados, a camada zero corresponde a camada de entrada da rede:

$$a_j^l = F_A\left(\sum_i W_{ij}^l \, a_i^{l-1} + b_j^l\right)$$
$$a_i^{l-1} = \eta_i^l \tag{2.62}$$

Passo 2 – Os valores reais da saída da rede são calculados tendo em vista que na saída a função de ativação das camadas escondidas não é aplicada:

$$\zeta_j^D = \sum_i W_{ij}^D \eta_i^D + b_j^D \tag{2.63}$$

Neste caso, a camada de entrada é representada por $l = 0$, e a dimensão da profundidade de rede por D representa o número de camadas total menos a inicial.

Assim, o vetor de parâmetros que representa os pesos e vieses da rede é dado por ϑ e N_D corresponde ao número de neurónios da última camada:

$$\vartheta = \left(W_{11}^1, W_{12}^1, \dots, W_{11}^D, \dots, W_{N_{D-1}N_D}^D, b_1^1, \dots, b_{N_D}^D\right)^T \tag{2.64}$$

2.2.2.1.2 Função de Ativação

As funções de ativação também conhecidas como função de transferência são tipicamente funções não lineares que transformam a soma ponderada dos valores de entrada nos valores de saída. A escolha da função de ativação é importante para a performance do algoritmo de treinamento. Para a realização de treinamento por *backpropagation* a função deve ser contínua, diferenciável e monótona não-decrescente [108], [109].

As funções mais comummente utilizadas são a de polo-único (sig_1)(ou logística), a bipolar sigmoide (sig_2)(ou tangente hiperbólica) e a Unidade Linear Retificada, ou *Rectified Linear Unit* (ReLU). Estas funções encontram-se na Tabela 2.4.

$sig_1(s) = \frac{1}{1+e^{-s}}$	(2.65)
$sig_2(s) = \frac{1-e^{-cte \cdot s}}{1+e^{-cte \cdot s}}$	(2.66)
$ReLU(s) = \frac{s+\lvert s\rvert}{2} = \begin{cases} \eta & se\ \eta > 0, \\ 0 & caso\ contrário. \end{cases}$	(2.67)

Tabela 2.4: Funções de ativação comummente utilizadas.

As duas primeiras são mais utilizadas em problemas de classificação [110]. Entretanto, a tendência nas redes neuronais mais modernas é a de aplicação das funções de ativação do tipo ReLU.

A função ReLU apresenta a vantagem de ser mais rápida em termos computacionais e durante o processo de aprendizagem [110] sendo uma das melhores opções em redes para regressão não-linear. Assim, para atingir o objetivo deste trabalho será utilizada a função de ativação ReLU. Será aplicada apenas nas camadas escondidas visto que a saída não deve ser limitada.

Assim, pode-se definir duas funções de ativação para a última camada D e para as escondidas l, respetivamente, para o algoritmo a ser implementado:

$$a_k^D = \zeta_k^D \tag{2.68}$$

$$a_k^l = ReLU(\zeta_k^l) = \begin{cases} \zeta_k^l & se\ \zeta_k^l > 0, \\ 0 & se\ \zeta_k^l \leq 0. \end{cases} \tag{2.69}$$

A derivada desta função será representada por uma função step do tipo função de Heaviside, ou função degrau. Esta foi desenvolvida por Oliver Heaviside, sendo singular e descontínua que assume valor nulo para argumento negativo, valor unitário para argumento positivo [111]. Além disso, apresenta como imagem para $x = 0$ o valor 0.5, como pode ser visto na Figura 2.6.

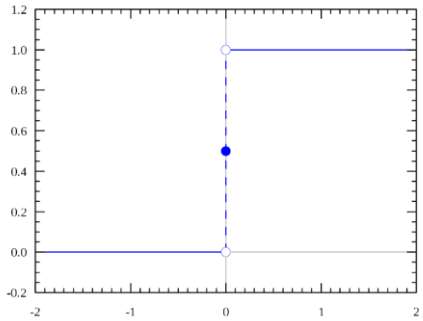

Figura 2.6: Função Heaviside (imagem obtida em [112])

Entretanto, para a derivada desta função, no caso das redes neuronais, é indiferente ser nula ou 0.5. Desta forma, esta função é diferenciável com exceção a origem, mas isso pode ser contornado atribuindo o valor nulo ou 0.5 durante o cálculo, caso seja necessário. Assim, a derivada da função será descrita da seguinte forma:

$$\frac{\partial a_k^l}{\partial \zeta_k^l} = \begin{cases} 1 & se\ \zeta_k^l > 0, \\ 0 & se\ \zeta_k^l \leq 0. \end{cases} \tag{2.70}$$

O facto de não ativar todos os neurónios ao mesmo tempo faz com que métodos de treinamento como *backpropagation* sejam mais rápidos se comparados as outras funções de ativação.

2.2.3 Aprendizado de Rede Neuronal

Após a compreensão do funcionamento do MLP é possível avançar para o método de treinamento que torna este algoritmo funcional. A maneira com que o aprendizado é efetuado permite a atualização dos valores dos pesos e vieses, utilizados para a soma ponderada, a partir de uma função de custo que deve ser selecionada a priori.

O objetivo do processo de aprendizagem é minimizar a função de custo ao longo da distribuição de dados gerados a partir da alteração dos pesos e vieses utilizados pela rede neuronal [113].

No problema de regressão um conjunto de n exemplos de treinamento x_i com os valores correspondentes y_i serão dados. A rede então utilizará os valores de x_i para fazer um cálculo preditivo do valor de y_i que será dado por $\hat{y}_i$. Para mensurar o desempenho pode-se utilizar uma função de custo que atinge valores elevados caso a rede desempenhe mal e vice-versa.

Visto que o problema é de regressão, onde os valores de saída são contínuos, a escolha de uma função de custo do tipo contínua é a mais adequada. A função contínua mais utilizada para este caso é a soma dos quadrados dos erros (SSE - *Sum of Squared Errors*), sendo utilizada na publicação principal do método de *Backpropagation* [114]. Sendo Y_i o vetor dos valores observados e $\hat{Y}_i$ o vetor dos valores previstos, então a função custo L dada pelo SSE do preditor é calculado pela seguinte equação [114]:

$$L = SSE = \frac{1}{2}\sum_{i=1}^{n}\left(Y_i - \hat{Y}_i\right)^2 \tag{2.71}$$

Tendo como objetivo minimizar o valor de L, a partir da alteração dos pesos e vieses, um dos métodos mais conhecidos é o de retropropagação, ou *Backpropagation*.

2.2.3.1 Algoritmo de *Backpropagation*

O algoritmo de *backpropagation* tem como conceito base a regra da cadeia de cálculo desenvolvida por Leibniz (1676) [115], [116] e L'Hopital (1696), sendo uma maneira eficiente de aplicar esta regra para NNs com funções de neurónios diferenciáveis. A regra da cadeia pode ser visualizada a seguir, onde $g_{in}(\cdot)$ e $g_{out}(\cdot)$ representam funções genéricas, e todas as derivadas são feitas em relação a variável $s \in \mathbb{R}$.

$$\frac{d}{ds}\Big(g_{out}\big(g_{in}(s)\big)\Big) = g'_{out}\big(g_{in}(s)\big) \cdot g'_{in}(s) \tag{2.72}$$

O principal objetivo do algoritmo de *Backpropagation* é alterar os valores dos pesos e vieses presentes na rede neuronal com o objetivo de reduzir a função custo L até que atinja um mínimo. A partir da regra da cadeia e a própria estrutura da NN, o algoritmo utiliza a derivada da função custo calculada na camada de saída para calcular a derivada da mesma função em todas as outras camadas, propagando o erro para trás na rede até a camada de entrada, o que explica a terminologia *backpropagation*.

O algoritmo será descrito a seguir, e permitirá observar a regra da cadeia em ação [117], [114].

Passo 1: Inicialização

Todos os valores de pesos e vieses são inicializados com valores aleatórios distribuídos uniformemente num pequeno intervalo de valores [109]. Os valores devem ser diferentes de zero, uma vez que se forem nulos os valores de gradiente serão também nulos. É indicado realizar vários treinamentos com valores de inicialização diferentes de forma a possibilitar encontrar diferentes valores mínimos para a função custo.

Passo 2: Nova era de treinamento

Uma era de treinamento corresponde a utilização de todos os exemplos do conjunto de treinamento. Os valores de atualização de pesos e vieses são normalmente atualizados apenas ao final de uma era. Assim, todos os gradientes dos pesos e vieses, além do atual valor de erro devem ser armazenados para esta era. Todos os gradientes dos pesos e vieses, além do erro, são inicializados com valor inicial nulo.

Passo 3: Propagação direta do sinal

Passo 3.1: Um dos elementos presentes no conjunto de treinamento é aplicado aos valores da camada de entrada da rede.

Passo 3.2: Os valores de saída das camadas escondidas são calculados segundo o algoritmo das MLPs.

Passo 3.3: As saídas da rede são então calculadas:

$$\hat{Y}_{ij} = \zeta_j^D = \sum_{i=1}^{N_{D-1}} (W_{ij}^D \eta_i^D) + b_j^D \tag{2.73}$$

Neste caso, $\hat{Y}_{ij}$ corresponde ao elemento j da saída prevista $\hat{Y}_i$.

Passo 3.4: A função custo por era é atualizada:

$$L = SSE = \frac{1}{2}\sum_{i=1}^{m}\left(Y_i - \hat{Y}_i\right)^2 \tag{2.74}$$

Passo 4: A propagação para trás dos erros e ajuste dos pesos

O objetivo neste passo é determinar os gradientes da função custo em relação aos parâmetros utilizados. A descrição em detalhe do método de *Backpropagation* encontra-se no Anexo A – Algoritmo de *Backpropagation*.

Para este passo, define-se δ_i^l como o erro no neurónio i da camada l por [114]:

$$\delta_i^l = \frac{\partial L}{\partial \zeta_i^l} \tag{2.75}$$

A derivada da função de ativação ReLU na camada escondida pode ser descrita por uma função Heaviside com argumento ζ_i^l, que neste caso está representada por $step(\zeta_i^l)$.

Representado os resultados do algoritmo em notação vetorial obtém-se [113]:

$$\begin{cases} \delta^D = \nabla_{a^D} L \\ \delta^l = \left(W^{l+1}\right)^T \delta^{l+1} \odot step(z^l) \end{cases} \tag{2.76}$$

$$\begin{cases} \dfrac{\partial L}{\partial b^l} = \delta^l \\ \dfrac{\partial L}{\partial W^l} = \delta^l \cdot \left(a^{l-1}\right)^T \end{cases} \tag{2.77}$$

Ao utilizar as equações (2.76) e (2.77) é possível determinar o gradiente em relação aos parâmetros, que é exatamente o que faz o algoritmo de *Backpropagation*.

Na equação (2.76), $\odot$ representa o operador do produto Hadamard (ou produto elemento a elemento) entre matrizes, ou seja, para duas matrizes de mesma dimensão é feito o produto dos elementos das mesmas coordenadas de cada matriz, resultando numa matriz de mesma dimensão das matrizes iniciais. Isso pode ser observado a seguir, onde Hd é a matriz do resultado do produto Hadamard entre duas matrizes U_1 e U_2 de mesma dimensão:

$$\begin{gathered} Hd = U_1 \odot U_2 \\ (Hd)_{\text{ij}} = (U_1)_{\text{ij}} \cdot (U_2)_{\text{ij}} \end{gathered} \tag{2.78}$$

Onde $(Hd)_{\text{ij}}$ representa o elemento da linha i e coluna j da matriz Hd.

Passo 5: Uma nova iteração

Enquanto houver vetores teste a serem aplicados na atual era, basta retornar para o passo 3. Caso contrário, é então feita a soma de todos os gradientes em relação a função de perda e a seguir é multiplicado por um fator ϵ chamado taxa de aprendizado.

$$\vartheta = \vartheta - \frac{\epsilon}{m}\sum_{i=1}^{m} \nabla_{\vartheta} L_i \tag{2.79}$$

Na equação (2.79), m é igual ao número de elementos presentes no conjunto de treinamento, e ϑ corresponde ao vetor contendo todos os valores de pesos e vieses da rede neuronal novamente. $\nabla_{\vartheta} L_i$ corresponde ao gradiente da função de custo em relação aos parâmetros ϑ, aplicados ao elemento i do conjunto Y.

Se uma era for completa, é feito o teste de verificação do critério para finalização do algoritmo (tolerância mínima é atingida ou o máximo número de eras de treino foram atingidas). Se não, retorna-se ao passo 2. Se sim, o algoritmo termina.

2.2.3.2 Métodos de Otimização da Função de Custo

O algoritmo de *Backpropagation* permite a otimização da função custo, a partir do cálculo do gradiente desta função em relação aos parâmetros da NN. Contudo, o número de parâmetros pode ser muito grande, assim como conjunto de treino, o que tornaria o processo deste cálculo um grande esforço computacional. Desta forma, foram desenvolvidos vários métodos que permitem a redução deste esforço.

O conceito principal para esta redução é a utilização de lotes, ou *batches*, em que a ideia é simplesmente dividir o conjunto de treinamento em *batches* de um certo tamanho *m*, avaliando o gradiente com apenas m exemplos de uma vez só, atualizando os parâmetros de acordo. A maioria dos algoritmos converge muito mais rapidamente, em relação ao tempo computacional total, caso tenha sido feita a aproximação dos gradientes se comparado com o cálculo exato usando todo o conjunto de treinamento [113].

Este trabalho foi realizado com o auxílio da linguagem de programação *Python*, utilizando uma das bibliotecas disponíveis em domínio público chamada *Scikit Learn*. A partir desta biblioteca foi possível realizar o treinamento da rede neuronal utilizando um dos métodos de otimização disponíveis, dentre eles: SGD, ADAM e L-BFGS [118], os quais serão descritos a seguir. Dentre estes, aquele que foi implementado com maior facilidade, em relação a tempo de computação e resultado, durante o treinamento foi o L-BFGS, o que será o foco da descrição a seguir.

2.2.3.2.1 Algoritmo SGD – *Stochastic Gradient Descent*

O algoritmo SGD, ou gradiente descendente estocástico, é um método iterativo utilizado para encontrar o mínimo de uma função objetivo. É um método de primeira ordem, ou seja, utiliza somente as primeiras derivadas de elementos da função. É baseada no facto que o gradiente de uma função L sempre aponta na direção do aumento máximo, então ao mover na direção oposta do gradiente é possível reduzir o valor da função objetivo [113].

Este algoritmo é baseado no método de gradiente descendente onde é calculado o gradiente da soma da função de perda de cada um dos elementos do conjunto de treinamento, avaliando-as todas naquele passo [113]. Quando o número de elementos é muito alto ocorre uma carga computacional enorme.

Assim, o método SGD, utilizando lotes ou *batches*, resolve este problema computacional ao selecionar aleatoriamente um número m de elementos do conjunto de treinamento. Então, o gradiente é aproximado e aplicado nos parâmetros. Selecionar um número muito baixo de amostras, como por exemplo $m = 1$, pode levar a um comportamento instável da função objetivo [113].

Taxa de Aprendizagem

Um parâmetro importante no processo de treinamento, senão o mais importante [119], assim como para os demais métodos de *Deep Learning* é a taxa de aprendizagem ϵ, também conhecido por *Learning Rate*. Este parâmetro determina o tamanho do passo aplicado em cada iteração. Um valor elevado resulta em passos grandes e vice-versa. Se for muito grande, pode apresentar um comportamento ruidoso do algoritmo levando a incapacidade de convergir para valores razoáveis da função objetivo [120]. Se for muito pequeno, o processo de obtenção do valor mínimo global acaba por ser ineficiente, podendo levar muito tempo para o alcançar ou até mesmo podendo ficar preso em mínimos locais [119]. A Figura 2.7 [119] mostra um exemplo da seleção incorreta de taxa de aprendizagem.

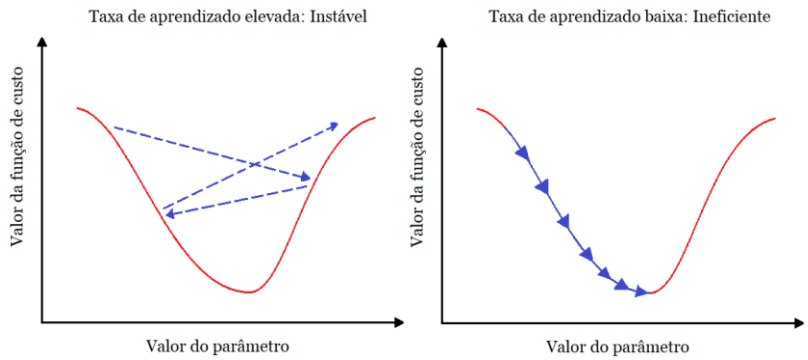

Figura 2.7: Representação do comportamento da função custo baseado no valor da taxa de aprendizagem. [119]

Apesar de se poder tentar utilizar valores constantes para a taxa de aprendizado, a escolha incorreta do valor pode levar aos problemas citados anteriormente. Vários métodos foram desenvolvidos permitindo alterar o valor da taxa ao longo do processo de aprendizagem, aumentando a eficiência do processo. Idealmente, o valor da taxa deve ser maior ao início do processo e ir reduzindo ao longo do tempo enquanto se aproxima da convergência [119]. Dentre estes alguns recomendados [119] são ADAM (*Adaptive Moment Estimation*) [121], Nesterov *Momentum* [122], RMSProp (*Root Mean Square Propagation*) [123] e AdaGrad (*Adaptive Gradient*) [124].

O algoritmo de SGD está descrito na Tabela 2.5.

Algoritmo 2.1 - SGD
Ponto inicial $\vartheta = \vartheta_0$
Taxa de aprendizagem ϵ
while (Critério de paragem não for atingido) **do**
Escolher um conjunto dos elementos de treinamento $\{X_{(i)}, Y_{(i)}\}_{i=1}^{m}$ aleatoriamente
$\vartheta \leftarrow \vartheta - \frac{\epsilon}{m}\sum_{i=1}^{m} \nabla_{\vartheta} L_i$
end while

Tabela 2.5: Algoritmo de SGD.

Apesar de ser simples, o SGD teve sucesso em diversos problemas de *Machine Learning* [113]. Entretanto, o algoritmo pode apresentar problemas de ficar estável em locais onde o gradiente é baixo, se a taxa de aprendizado também for baixa, assim como outros problemas também podem ocorrer em regiões de elevada curvatura [113]. Alguns destes problemas tentam ser resolvidos pelo algoritmo ADAM.

2.2.3.2.2 Algoritmo ADAM

O algoritmo ADAM apresenta este nome devido ao método utilizado: *Adaptive Moments Estimation*. É um algoritmo de primeira ordem para funções objetivas estocásticas sendo baseado em estimativas adaptadas dos momentos de primeira e segunda ordem. A regra de atualização dos parâmetros é baseada em médias móveis do gradiente em relação aos parâmetros que decaem exponencialmente e ao quadrado deste gradiente.

O algoritmo foi desenvolvido em 2014 [121], aplicando as abordagens de adaptação do método RMSprop para o momento. O momento é definido pela capacidade do algoritmo de se lembrar o valor do gradiente da última iteração para aplicação na próxima, realizando uma combinação linear dos gradientes da iteração atual e passada.

O algoritmo é bastante popular para o treinamento de redes neuronais e tem mostrado bons resultados numa variedade de problemas, entretanto é mais complexo que o algoritmo anterior SGD. [121]

2.2.3.2.3 Algoritmo *Limited-Memory* Broyden-Fletcher-Goldfarb-Shanno (L-BFGS)

Antes de perceber o funcionamento do algoritmo BFGS de memória limitada (L-BFGS – *Limited-Memory* Broyden-Fletcher-Goldfarb-Shanno) é preciso compreender a sua origem. O L-BFGS utiliza, para a sua estratégia de otimização, métodos de segunda ordem, ou seja, inclui além do gradiente, a matriz Hessiana H_s ou alguma aproximação desta de forma a ter em conta a curvatura da função objetivo.

O conceito de matriz Hessiana foi desenvolvido pelo matemático alemão Ludwig Otto Hesse, no séc. XIX, sendo a matriz H_s dada por todas as combinações de derivada parcial de segunda ordem, se existente, de uma função f genérica, em formato matricial, ou seja:

$$\left(H_s(f(x_1, \dots, x_n)\right)_{ij} = \frac{\partial^2 f}{\partial x_i \partial x_j}. \tag{2.80}$$

Um algoritmo que trabalha quando a função objetivo é quadrática ou convexa é chamada de método de Newton, mas uma vez que redes neuronais são muitas vezes não convexas então a convergência não é garantida [6].

Uma função é dada por convexa, se todos os pontos de um segmento de reta entre dois pontos distintos de um gráfico desta função sempre estiverem acima desta, ou seja, em terminologia matemática na seguinte condição:

$$f: \mathbb{X} \to \mathbb{R}: f(s \cdot \chi_1 + (1-s) \cdot \chi_2) \leq \ s \cdot f(\chi_1) + (1-s) \cdot \ f(\chi_2) \ \forall \ s \in [0,1], \chi_1, \chi_2 \in \ \mathbb{X}$$

O método de Newton tem como princípio a utilização da série de Taylor de segunda ordem para otimização da função f e pode ser descrito na Tabela 2.6 [113]:

Algoritmo 2.2 – Método de Newton

Ponto inicial $x = x_0$

while (Critério de paragem não for atingido) **do**

$\boldsymbol{g} \leftarrow \nabla f$

$\boldsymbol{H_s} \leftarrow \nabla^2 f$

$\vartheta \leftarrow \vartheta - \boldsymbol{H_s^{-1}} \boldsymbol{g}$

end while

Tabela 2.6: Algoritmo do método de minimização de Newton.

O método de Newton requer cálculo correto da matriz Hessiana e gradiente, o que pode trazer cargas computacionais elevadas como no caso de a função ter n variáveis então a matriz Hessiana teria dimensão $n \times n$. Essas condições fazem com que o método de Newton não seja adequado para treinamento de redes neuronais [6].

Métodos Quase-Newton

Os algoritmos são chamados método Quase-Newton quando buscam algumas vantagens do método de Newton sem a complexidade computacional do cálculo da matriz Hessiana. A ideia é aproximar a matriz Hessiana, neste caso a sua inversa, de forma a não precisar calculá-la explicitamente. O problema de métodos de segunda ordem é que em geral podem apresentar carga computacional maior se comparada com métodos de primeira ordem.

O algoritmo BFGS (Broyden-Fletcher-Goldfarb-Shanno) tem como origem o trabalho independente de quatro pesquisadores: Broyden [125], Fletcher [126], Goldfarb [127] e Shanno [128]. Os principais teoremas base para este método podem ser vistos em [129]. Este algoritmo tem como objetivo calcular a matriz M_t que aproxima a inversa da matriz Hessiana verdadeira da função objetivo com n variáveis. Essa matriz é simétrica, logo são necessários $n(n+1)/2$ locais de armazenamento. Em problemas de dimensões grandes é uma difícil tarefa reter essas matrizes no armazenamento de alta velocidade, então isto acaba por ser um problema. Ainda assim, neste caso a matriz M_t deve satisfazer a condição Quase-Newton [130]:

$$M_{t+1} \cdot (\vartheta_{t+1} - \vartheta_t) = \nabla f(\vartheta_{t+1}) - \nabla f(\vartheta_t) \,. \tag{2.81}$$

Por esse algoritmo, muito tempo pode ser salvo, já que a matriz Hessiana é aproximada utilizando o valor dos gradientes calculados.

O grande problema deste método é a necessidade de armazenar o valor da matriz passada M_t para a próxima iteração. Muitas vezes, o treinamento da rede neuronal pode abranger milhões de parâmetros, sendo inviável armazenar o valor desta matriz para cada iteração.

Assim, o algoritmo de memória limitada BFGS (L-BFGS) desenvolvido em 1980 por Nocedal [131] é capaz de realizar a aproximação da matriz Hessiana sem a necessidade de alocar memória para os valores anteriores desta matriz, sendo capaz de, em todos os passos de atualização, substituir a informação antiga pela nova, estando continuamente atualizando esta matriz [131].

Neste caso, também se assume que as últimas m atualizações destes valores são armazenadas. É então feita a primeira aproximação da matriz Hessiana na iteração $\boldsymbol{t}$ dada por $H_{\boldsymbol{t}}^{0} = \gamma_{\boldsymbol{t}} I$ onde I é a matriz identidade e [113]:

$$\gamma_{\boldsymbol{t}} = \frac{s_{\boldsymbol{t}-1}^{T} y_{\boldsymbol{t}-1}}{y_{\boldsymbol{t}-1}^{T} y_{\boldsymbol{t}-1}} \tag{2.82}$$

Onde

$$\begin{cases} y_{\boldsymbol{t}} = \nabla f(\vartheta_{\boldsymbol{t}+1}) - \nabla f(\vartheta_{\boldsymbol{t}}) \\ s_{\boldsymbol{t}} = \vartheta_{\boldsymbol{t}+1} - \vartheta_{\boldsymbol{t}} \end{cases} \tag{2.83}$$

Também é preciso definir a sequência de m escalares ρ_i como [113]:

$$\rho_i = \frac{1}{y_i^{T} s_i} \tag{2.84}$$

Onde $i = t - m, \dots, t$. O algoritmo finalizado pode ser visualizado na Tabela 2.7. Neste caso, é necessário apenas armazenar as m últimas atualizações dos valores de $s_{\boldsymbol{t}}$ e $y_{\boldsymbol{t}}$, invés de guardar toda a matriz aproximada, entretanto esta torna-se mais imprecisa.

Algoritmo 2.3 – L-BFGS [131]

Ponto inicial ϑ_0

$\boldsymbol{t} \leftarrow 1$

while (Critério de paragem não for atingido) **do**

$\vec{q} \leftarrow \nabla f(\vartheta_{\boldsymbol{t}})$

for i= **t** -1, ..., **t** -m **do**

$\alpha_i \leftarrow \rho_i s_i^{T} \vec{q}$

$\vec{q} \leftarrow \vec{q} - \alpha_i y_i$

end for

Calcular $\gamma_{\boldsymbol{t}}$

$H_{\boldsymbol{t}}^{0} \leftarrow \gamma_{\boldsymbol{t}} I$

$\psi \leftarrow H_{\boldsymbol{t}}^{0} \vec{q}$

for i= **t** -m, ..., **t** -1 **do**

$\beta_i \leftarrow \rho_i y_i^{T} \psi$

$\psi \leftarrow (\alpha_i - \beta_i) s_i$

end for

$\vartheta_{t+1} \leftarrow \vartheta_t - \psi$

$t \leftarrow t + 1$

end while

Tabela 2.7: Algoritmo L-BFGS.

O algoritmo L-BFGS não desempenha tão bem utilizando o método de *mini-batches*, sendo bastante sensível a alteração dos parâmetros que definem a rede e os dados de entrada. Suspeita-se que seja necessário utilizar uma grande parte, senão toda, dos elementos de treinamento para conseguir uma boa aproximação da matriz Hessiana [130]. Assim como para o algoritmo BFGS, este não necessita da taxa de aprendizagem como hiper-parâmetro, já que o valor de atualização dos parâmetros é ajustado a partir do valor de α_i e β_i, na variável ψ do algoritmo.

Capítulo 3 – Aplicação de Redes Neuronais MLP à Filtragem de Kalman Pseudo-linear

A combinação de redes neuronais à filtragem de Kalman é um tema que tem sido abordado há mais de duas décadas e essa combinação tem sido implementada principalmente de duas diferentes formas. Os métodos de filtragem têm sido utilizados como ferramenta de auxílio para o treinamento das redes neuronais em diversas aplicações e as redes neuronais têm sido utilizadas com o intuito de melhorar o desempenho do filtro ou reduzir o seu custo computacional, baseando-se em modelos com diferentes níveis de conhecimento: totalmente conhecidos, parcialmente conhecidos ou totalmente desconhecidos.

Nesta secção, será abordada justamente a segunda aplicação onde as redes neuronais servirão como ferramenta para melhorar o desempenho de um filtro. Primeiramente, será feita uma introdução histórica acerca da união dos conceitos explicados nas secções anteriores neste âmbito. A seguir, será reunido um conjunto de técnicas que permitem a constituição de um algoritmo onde há a aplicação de uma rede do tipo MLP para a filtragem de Kalman pseudo-linear.

3.1 Desenvolvimento sobre a integração de NNs e métodos de filtragem

Nos anos recentes foi possível identificar diversos trabalhos que com sucesso conseguiram combinar as redes neuronais aos processos de filtragem.

Em 1997, Horton [132] investigou o problema de identificação em tempo real das derivativas aerodinâmicas na aplicação de um míssil guiado. Um estimador híbrido foi formado a partir de um filtro de Kalman Linearizado que é assistido por uma NN, produzindo resultados superiores àqueles dos quais é derivado.

Um outro exemplo de aplicação é na previsão da vida útil restante na aeronáutica, onde combinam o modelo de previsão do final do tempo de vida com uma NN e filtragem final com um filtro de Kalman, exemplo que pode ser visto como exemplo em [133] e [134].

Em [135] é possível ver um exemplo da aplicação de uma rede do tipo RBF (*Radial Basis Function*) aplicada a filtragem EKF adaptativa sendo capaz de estimar o estado e a matriz de covariância dos erros em uma aplicação de navegação relativa entre veículos aeroespaciais.

Em [136] demonstrou-se a aplicabilidade de uma rede do tipo MLP *feed-forward* em conjunto com um EKF para a previsão do estado do modelo meteorológico DYNAMO.

Para KFs auxiliados por redes neuronais profundas, várias maneiras foram sugeridas de forma a estimar a matriz de covariância dos erros e a matriz de ganho. Por exemplo, a rede recorrente de Kalman (RKN – *Recurrent Kalman Filter*) proposta em [137] foi treinada para produzir uma estimativa de erro além da estimativa de estado utilizando uma RNN.

A KalmanNet desenvolvida por Revach, Shlezinger, Ni, Escoriza, van Sloun e Eldar [24] onde uma RNN é aplicada para a estimação da matriz de ganho K com base em modelos estatísticos parcial ou totalmente desconhecidos em aplicações onde o modelo dinâmico é incompatível ou não-linear.

Em [138] foi demonstrado que ao aplicar KalmanNet em modelos lineares com matriz de observação completa e ruído de medição gaussiano, a matriz de covariância do erro pode ser extraída de recursos internos da arquitetura.

Em 2023, Dahan, Revach, Dunik, e Shlezinger [138] estudaram diferentes métodos de extrair a matriz de covariância do erro para o auxílio em rastreio a partir de redes neuronais profundas em modelos não-lineares comparando os resultados ao filtro EKF.

3.2 Técnicas implementadas e propostas

Visto que os métodos de filtragem de Kalman linear podem ser aplicados a modelos que apresentem estrutura pseudo-linear, é possível aproveitar-se da estrutura de filtros de Kalman Pseudo-lineares e utilizar uma rede neuronal MLP para calcular a matriz de ganho do sistema em função da estimativa de estado atual e das matrizes de covariância do erro de processamento Q e de medição R.

O processo de filtragem é composto por duas etapas: a previsão e a correção. A previsão consiste em realizar a extrapolação do estado atual do sistema para o próximo instante, a partir das equações que descrevem o modelo do sistema. Quanto mais preciso for o modelo, melhor será a precisão do filtro. Já em relação a etapa da correção, o filtro baseado no modelo estatístico do sistema, a partir das matrizes de covariância dos ruídos de processamento e medição, é capaz de calcular uma matriz de covariância do erro do estado e subsequentemente uma matriz de ganho que tem como objetivo corrigir a previsão do passo anterior e estimar o estado atual a partir da medição atual efetuada.

O processo de resolução da equação de Riccati para a determinação da matriz de covariância do erro do estado estimado é extenuante do ponto de vista computacional, sendo necessário realizar inversões matriciais que aumentam a sua complexidade em proporção ao número de dimensões presentes no modelo do sistema. Assim, a ideia seria utilizar uma função não-linear para aproximar a matriz de ganho K utilizando como argumento o estado estimado atual.

A seguir, serão apresentados os métodos que foram utilizados para a estruturação de um sistema não-linear para o formato pseudo-linear, além do método de treinamento e validação das redes neuronais para a determinação da matriz de ganho.

3.2.1 Factorização de uma função não-linear

Para que a técnica de filtragem pseudo-linear seja aplicada, é necessário realizar a factorização ou parametrização da função $f(x)$ não-linear que descreve o modelo do sistema ou o modelo de medição em uma estrutura pseudo-linear onde $f(x) = A(x)\,x$, sendo a única condição necessária $f(0) = 0$. O modelo do sistema está descrito a seguir:

$$\dot{x}(t) = f(x, u, t) \tag{3.1}$$

Contudo, no caso em que a dimensão do vetor estado é maior que o caso escalar, a factorização não é única, e pode ser feita não só para melhorar o desempenho, otimização, estabilidade, robustez entre outros aspetos do filtro como também para evitar singularidades no cálculo matricial, trazendo maior flexibilidade no design do filtro [5].

Neste caso de dimensões superiores a um, a própria matriz fatorizada pode ser parametrizada da seguinte forma. Suponha que $A_1(x)$ e $A_2(x)$ são duas factorizações possíveis da função $f(x)$ onde $f(x) = A_1(x)x = A_2(x)x$. Então: [5]

$$A(x, \alpha) = \alpha A_1(x) + (1 - \alpha)A_2(x) \tag{3.2}$$

é também uma possível parametrização dependente do estado para qualquer valor de α. Assim, $A(x, \alpha)$ representa uma família infinita de parametrizações dependentes do estado para a função $f(x)$ [5]. Este conceito pode ser estendido para um número ainda maior de factorizações $A_i(x)$. Os graus de liberdade disponíveis através de alfa trazem flexibilidade de design que pode ser utilizada para alterar os diversos aspetos do filtro mencionados anteriormente.

Em [5], Çimen descreve algumas das principais diretrizes que são efetuadas de forma a contornar os principais problemas ocorridos durante o processo de factorização, além de permitir um possível aumento de desempenho do filtro. Os principais problemas são a presença de termos independentes do estado e a presença de termos dependentes do estado que excluem a origem, no caso da factorização da matriz do modelo do sistema.

A primeira sugestão ao selecionar a factorização dependente do estado é aplicar um valor não nulo ao elemento $\{i, j\}$ da matriz $A(x, \alpha)$ se a i-ésima derivada do estado depender do j-ésimo estado [5].

Conformar com estrutura correta e condições

Para que o método PSELIKA seja aplicado diretamente, é necessário que a dinâmica do sistema descrita por (3.1) seja tal que $f(0) = 0$. Assim, as situações que devem ser contornadas são a presença de termos independentes do estado ou vieses e a presença de termos dependentes do estado que excluem a origem.

Presença de termos independentes do estado

Na presença de termos independentes do estado, existem três principais formas de lidar com este termo dado por $b(t)$. Primeiramente, se o viés é constante ou varia lentamente, então pode ser modelado como um estado estável [139]

$$\dot{b}(t) = -\lambda b(t) \tag{3.3}$$

Onde λ é um valor positivo pequeno. Então, em cada etapa de filtragem, o valor real de $b(t)$ é utilizado ao calcular o valor da matriz do modelo do sistema agora aumentado por esta variável.

A segunda solução [140] é multiplicar e dividir o viés por um estado x que é sabido que não será nulo, como por exemplo o módulo do vetor velocidade ou a temperatura em Kelvin de algum sistema, sendo fatorizado como

$$b(t) = \left[\frac{b(t)}{x}\right] x \tag{3.4}$$

Onde x, neste caso, não será nulo durante o processo de filtragem.

A terceira maneira é realizar um aumento do sistema utilizando uma variável estável x_a [141], tal que

$$\dot{x}_a(t) = -\lambda x_a(t) \tag{3.5}$$

com $\lambda > 0$. O viés pode então ser fatorizado por

$$b(t) = \left[\frac{b(t)}{x_a}\right] x_a \tag{3.6}$$

Neste caso, toda vez que a matriz do sistema for calculada, o valor inicial $x_a(0)$ será utilizado na estimação.

Presença de termos dependentes do estado que excluem a origem

No caso da presença de termos que são dependentes do estado, mas não passam pela origem quando o estado é nulo a abordagem pode ser um pouco diferente para manter a condição $f(0) = 0$.

É possível utilizar a segunda e terceira abordagens do item anterior, onde multiplica-se e divide-se por uma variável de estado que se sabe que não terá valor nulo durante o funcionamento. Entretanto, é mais desejável capturar a dependência do estado destes termos no próprio elemento da matriz $A(x, \alpha)$ [5]. Por exemplo [5], suponha que $\dot{x}_2 = cos\, x_1$. Como é possível perceber $f(0) = 1$ neste caso, então será necessário alterar essa função, mantendo o facto que $\dot{x}_2$ depende de x_1. E isso pode ser feito da seguinte forma:

$$\dot{x}_2 = \left[\frac{\cos x_1 - 1}{x_1}\right] x_1 + 1$$

Como o limite de $\left[\frac{cos\, x_1 - 1}{x_1}\right]$ quando x_1 tende para 0 é nulo, a condição $f(0) = 0$ se mantém. O termo "1" criado é um viés e é um problema que pode ser resolvido pelos métodos apresentados anteriormente.

3.2.2 Definição do domínio do sistema para treinamento

Para que seja possível realizar um bom treinamento de uma rede neuronal, é necessário adquirir dados que sejam fiáveis. No caso de um modelo de sistema, é preciso definir quais dados são importantes para o treinamento, e isso é feito selecionando primeiramente o domínio do espaço de estados de forma eficiente. Como é necessário aplicar esse conceito para ambas etapas de previsão e correção, então o domínio para ambas deve ser o mesmo.

O primeiro passo é definir os limites do domínio do sistema para cada dimensão, assim como para o controlador, caso exista um. Um exemplo seria a magnitude do vetor velocidade de uma aeronave de pequeno porte, que apresentaria um limite inferior de 0 e um superior de V_{max}, enquanto o outro exemplo seria o ângulo de ataque da asa poderia variar de -17 graus a +20 graus, num funcionamento normal, sendo estes valores normalmente obtidos a partir de dados experimentais.

A seguir, a seleção dos pontos que serão utilizados para o treinamento da rede pode ser feita através diversos métodos. É possível dividir os intervalos do domínio selecionado em várias partes e criar uma malha de pontos, sendo a precisão maior quanto maior for a divisão efetuada. Também é possível selecionar os pontos de forma aleatória, utilizando uma função de distribuição uniforme dos pontos dentro do domínio selecionado.

3.2.3 Rede neuronal aplicada a etapa de correção

A matriz de ganho de Kalman apresenta um outro procedimento de obtenção dos valores alvos. O argumento utilizado será apenas o vetor estado, já que pelas formulações descritas na secção 2.1.5, o cálculo da matriz de ganho de Kalman não depende do vetor de controlo, caso exista um controlador, apenas dependendo do estado anterior e da matriz de covariância do erro. Para a etapa de correção, o conjunto selecionado de p pontos a partir da definição do domínio é utilizado para o treinamento da rede neuronal. Os dados utilizados como alvos são definidos no seguinte procedimento.

Uma vez que o sistema apresenta uma estrutura do tipo linear, é possível aplicar métodos de filtragem linear a esta estrutura. Com isso em mente, a matriz de ganho de Kalman K pode ser pré-calculada para um certo valor de vetor estado resolvendo a Equação Algébrica de Riccati Discreta (DARE - *Discrete Algebraic Riccati Equation*) associada àquele sistema naquele estado, como descrito na secção (Regulação não-linear SDRE). Essa propriedade dos sistemas de estrutura pseudo-linear permite que os valores das matrizes K sejam utilizados como valores alvo para o treinamento de uma rede neuronal.

Considere as seguintes equações que descreve o modelo do sistema e de medição já fatorizados e discretizados:

$$\begin{aligned} x_k &= A_d(x_{k-1})x_{k-1} + B_d(x_{k-1})u_{k-1} + w_{k-1} \\ z_k &= C(x_k)x_k + v_k \end{aligned} \tag{3.7}$$

A DARE associada ao vetor de estado x_k, permite encontrar o valor da matriz de covariância P_k, o que permite então o cálculo da matriz de ganho K_k, o alvo Y_k da rede neuronal, a partir da equação:

$$Y_k^T = K_k^T = \left(C_k P_k C_k^T + R\right)^{-1} C_k P_k A_d^T(x_{k-1}) \tag{3.8}$$

Para encontrar então a matriz P_k é preciso resolver a seguinte DARE

$$\begin{aligned} &A_d(x_{k-1})P_k A_d^T(x_{k-1}) - P_k \\ &\quad - \left(A_d(x_{k-1})P_k C_k^T\right)\left(R_k + C_k P_k C_k^T\right)^{-1}\left(C_k^T P_k A_d^T(x_{k-1})\right) + Q_k = 0 \end{aligned} \tag{3.9}$$

Como pode ser visto, para a solução da DARE é necessário obter os valores de $A_d(x_{k-1})$, C_k, Q_k e R_k. Para a solução da DARE, existem diversos métodos. O método que será utilizado será a partir do auxílio da função dlqr(*) do software matlab [142], a qual apresenta como

saída os valores de K_k e P_k para a equação dada. Assim, a matriz K_k será calculada em relação ao valor x_k.

É importante mencionar que as NN do tipo MLP retornam apenas vetores, sendo necessário para o caso na $N_{corr}(\cdot)$ remodelar o vetor de saída de dimensão $m \cdot r \times 1$, para uma matriz de dimensão $m \times r$. O processo inverso deve ser feito durante a etapa de treinamento, onde o alvo matriz K_k deve ser transformado em vetor.

Para as matrizes de transição $A_{d_k}(x)$ e $B_{d_k}(x)$, o índice k representa que as matrizes são obtidas utilizando o estado x_k, ou seja, para o instante t_k, assim como para as matrizes de medição $C_k(x)$, covariância do erro de processamento $Q_k(x)$ e covariância do erro de medição $R_k(x)$. As matrizes de transição são obtidas através de métodos de discretização de modelos contínuos [11]. O método que será utilizado neste trabalho será o de aproximação integral. Este método assume que os valores de entrada no sistema são constantes durante o período em que são aplicados, também conhecido por *zero-order hold* [143].

Neste caso, T é o período total desde o instante inicial:

$$\begin{aligned} x(T) &= e^{AT}x(0) + \int_0^T e^{A(T-\tau)}B(0)d\tau \\ &= e^{AT}x(0) + e^{AT}\int_0^T e^{-A\tau}d\tau\, B(0) = F(T)x(0) + \int_0^T F(T-\tau)d\tau\, B(0) \end{aligned} \tag{3.10}$$

Assim, conclui-se que:

$$\begin{aligned} A_d &= e^{AT} \\ B_d &= \int_0^T e^{-A\tau}d\tau\, B \end{aligned} \tag{3.11}$$

A partir da definição de exponencial matricial chega-se a seguinte expressão para a discretização, caso o passo de discretização seja de T_s:

$$\begin{aligned} A_d &= I + (A \cdot T_s) + \frac{(A \cdot T_s)^2}{2!} + \frac{(A \cdot T_s)^3}{3!} + \cdots \\ B_d &= \left(I \cdot T_s + A \cdot \frac{T_s^2}{2!} + A^2 \cdot \frac{T_s^3}{3!} + \cdots \right) \cdot B \end{aligned} \tag{3.12}$$

Essa expressão pode ser simplificada, com a precisão sendo maior quanto maior o número de termos utilizados ou menor o passo T_s selecionado. Para os demais algoritmos, a

quantidade de termos considerados na expressão de discretização de A_d foi determinada com base na norma de Frobenius dos termos. Somente os termos cuja norma fosse superior a um pequeno valor pré-definido, chamado EPSILON, foram incluídos. O número de termos selecionado para B_d é igual ao número de termos utilizado em A_d. O valor foi selecionado de forma a permitir que o número de termos fosse reduzido a no máximo quatro, de maneira a reduzir o custo computacional dos filtros.

3.2.4 Treinamento e validação da rede neuronal

A rede neuronal será treinada utilizando como ferramenta de auxílio a biblioteca *scikit-learn* (sklearn) disponível para a linguagem *Python*. Esta biblioteca apresenta diversos modelos de treinamento, e aquele que foi selecionado para este algoritmo foi o MLPRegressor (*Multilayer Perceptron Regressor*), com o método de solução 'lbfgs' [118] ou L-BFGS.

É importante mencionar que os valores de argumento em ambos os casos de treinamento devem ser padronizados, ou seja, devem ser alterados para que a média final dos valores seja nula, com desvio padrão unitário, para que o processo de treinamento apresente maior efetividade [106]. O modelo de padronização deve ser aplicado a qualquer valor de argumento utilizado na rede neuronal final. O mesmo processo não é necessário para o valor alvo da rede, podendo ser alterado de acordo com o desempenho necessário.

Para a padronização de um conjunto de elementos utiliza-se a seguinte equação:

$$z_i = \frac{Y_i - \bar{\mu}}{\sigma} \tag{3.13}$$

Onde Y_i representa um elemento i do conjunto, $\bar{\mu}$ representa a média da população, σ representa o desvio padrão médio da população e, por fim, z_i representa o valor padronizado de x_i segundo o conjunto de elementos.

Para ambos os casos de treinamento, é necessário utilizar algum método de avaliação da rede neuronal em relação aos dados de treinamento. Aquele que será utilizado é justamente o método existente para o modelo MLPRegressor, o coeficiente de determinação, ou R^2, descrito dentro das funções como *score*. Este coeficiente pode retornar um valor dentro do intervalo $(-\infty, 1]$, onde se retornar o valor 1 significa que a rede prevê corretamente o valor da função não-linear que se pretende aproximar. O valor nulo significa que a rede retorna sempre valor médio de todas as medições e valores negativos significam que a rede apresenta previsões piores do que a média. Existem diversas equações diferentes para o cálculo de R^2. Uma das equações mais geralmente utilizada para o cálculo de R^2 está descrita a seguir, a qual será utilizada para medir o desempenho das redes [144]:

$$R^2 = 1 - \frac{\sum_i (Y_i - \hat{Y}_i)^2}{\sum_i (Y_i - \bar{Y})^2} \tag{3.14}$$

Onde $\hat{Y}_i$ representa o valor estimado ou previsto da observação Y_i e $\bar{Y}_i$ representa a média dos m valores observados dados por:

$$\bar{Y} = \frac{1}{m} \sum_{i=1}^{m} Y_i \tag{3.15}$$

O coeficiente de determinação pode ser calculado utilizando o conjunto de treinamento e o conjunto de teste. O conjunto de treinamento é aquele que apresenta o argumento e o valor de retorno de uma função e sendo utilizado para o processo de treinamento. O conjunto de teste é utilizado apenas para a validação do processo de treinamento. É importante que o conjunto de elementos do teste não esteja contido no conjunto utilizado para o treinamento, para evitar a ocorrência um fenômeno chamado *overfitting*, onde a rede neuronal retorna valores precisos para os dados de treinamento, mas não para novos dados. Um exemplo disso seria a rede apresentar um *score* de R^2=0.95 na fase de treinamento, mas R^2=0.1 na fase de teste. Esta é uma forma bastante rápida e simples de realizar a avaliação da rede com resultados válidos, e por isso será utilizada.

Existem diversas maneiras de realizar o processo de avaliação, com métodos chamados de validação cruzada, podendo ser bastante complexos, mas com um detalhamento maior acerca do desempenho da NN utilizada [145].

3.3 Algoritmo Proposto – PSELIKA-MLP

Considere o seguinte sistema dinâmico não-linear dado por:

$$\dot{x} = f(x, u) + w(t) \tag{3.16}$$

Onde, $x \in \mathbb{R}^n$ representa o vetor de estados do sistema, $f(\cdot)$ a função não-linear que descreve a dinâmica do sistema; $u \in \mathbb{R}^m$ representa o vetor de entrada do controlo no sistema; w representa o ruído de processamento aplicado ao sistema que é caracterizado por ser branco e gaussiano, apresentando matriz de covariância $Q(t) \in \mathbb{R}^{n \times n}$.

Para o modelo de medições tem-se:

$$z = h(x) + v(t) \tag{3.17}$$

Neste caso, $z \in \mathbb{R}^r$ representa o vetor das medições efetuadas; $h(x)$ representa a função não-linear que descreve o modelo de medição do sistema; v descreve o ruído de medição aplicado ao modelo sendo caracterizado por ser também branco e gaussiano, mas apresentando matriz de covariância $R(t) \in \mathbb{R}^{r \times r}$.

Em ambos modelos, a matriz de covariância do ruído é assumida constante ao longo do processo de filtragem.

O primeiro passo é a factorização do sistema dinâmico não-linear e do modelo de medição, para o formato pseudo-linear, seguindo as demais diretrizes descritas na secção 3.2.1. Assim, o sistema completo pode ser rescrito por:

$$\begin{aligned} \dot{x} &= A(x)x + B(x)u(t) + w(t) \\ z &= C(x)x + v(t) \end{aligned} \tag{3.18}$$

Que, para este caso deve ser em formato discreto:

$$\begin{aligned} x_k &= A_d(x_{k-1})x_{k-1} + B_d(x_{k-1})u_{k-1} + w_{k-1} \\ z_k &= C(x_k)x_k + v_k \end{aligned} \tag{3.19}$$

O índice k representa o valor da variável no instante discreto t_k.

O segundo passo é a definição da rede neuronal para a etapa de correção da estimativa do estado segundo as equações descritas na seção 3.2.3. Para isso, deve-se definir o domínio sobre o qual a rede será treinada, selecionando os pontos que serão argumentos, no caso o vetor estado x_{k-1} e calculando os valores alvo, no caso a matriz de ganho de Kalman no instante t_k, $K_k \in \mathbb{R}^{m \times r}$, para o treinamento da MLP.

Destes valores, recomenda-se que uma fração n, chamada conjunto de teste, seja separada e que não participe do conjunto de treinamento, sendo utilizada a posteriori do treinamento para a validação da NN. Após a definição e cálculo destes valores, é preciso selecionar o número de camadas e neurónios por camada da NN. Após essa seleção, será utilizado o método L-BFGS para o treinamento sendo verificado o desempenho após o treinamento através de métodos de validação.

Caso o desempenho não seja o desejado, deve-se alterar ou o conjunto de dados de treinamento, ou o número de camadas ou neurónios por camada. Por fim, a rede neuronal gerará uma função não-linear chamada $N_{corr}(\cdot)$, que recebe por argumento o vetor x_{k-1}, segundo a seguinte equação.

$$K_k = N_{corr}(x_{k-1}) \tag{3.20}$$

Por fim, o valor da nova estimativa x_k do estado será dada então pela equação da correção da estimativa do estado presente na Tabela 2.3:

$$\hat{x}_k(+) = \hat{x}_k(-) + K_k[z_k - C_k(x)\hat{x}_k(-)] \tag{3.21}$$

Além disso, o processo de treinamento é bastante experimental sendo necessário realizar múltiplos experimentos de forma a obter o melhor desempenho, visto que o algoritmo de otimização pode ficar aprisionado em algum mínimo local. Os principais fatores que alteram o desempenho do filtro são a forma de factorização das funções não-lineares, as matrizes de covariância $Q(t)$ e $R(t)$, a definição do domínio, a distribuição dos pontos do domínio selecionados além do número de camadas e de neurónios por camada.

A Tabela 3.1 descreve o algoritmo de treinamento da NN e a Tabela 3.2 o algoritmo de PSELIKA-MLP (Filtro de Kalman Pseudo-linear com *Perceptron* de Múltiplas Camadas) de forma sucinta.

1 Inicialização do treinamento N_{corr}	
1.1 Seleção do conjunto de vetores de estado x_i para treinamento	Domínio=$\{x_i\}_p$
1.2 Seleção das matrizes de covariância do filtro	$Q_0 = Q_{inicial}, R_0 = R_{inicial}$
1.3 Solução da DARE associada	$[P_k] = dlqr(A_d^T, C_k^T, Q_k, R_k)$
1.4 Cálculo da matriz de ganho K_k	$Y_k^T = K_k^T = \left(C_k P_k C_k^T + R\right)^{-1} C_k P_k A_d^T(x_{k-1})$
1.5 Seleção do número de camadas escondidas	camadas=l
1.6 Seleção do número de neurónios N_l para cada camada	neurónios = N_l
2 Treinamento da N_{corr}	
2.1 Utilização do método L-BFGS	MLPRegressor-Método='lbfgs'
2.2 Valor de desempenho da N_{corr}	Coeficiente de determinação R^2

Tabela 3.1: Algoritmo de treinamento do filtro PSELIKA-MLP.

1 Inicialização do filtro	
Assumir o vetor de estado inicial	$\hat{x}_0(+) = x_0$
2 Propagação da estimativa de estado	
Equação de previsão do estado	$\hat{x}_k(-) = A_{d\,k-1}(x)\hat{x}_{k-1}(+) + B_{d\,k-1}(x)u_{k-1}$
3 Correção da estimativa de estado	
Cálculo da matriz de ganho K_k	$K_k=N_{corr}(\hat{x}_{k-1}(+))$
Equação de correção do estado	$\hat{x}_k(+)=\hat{x}_k(-)+K_k[z_k-C_k(x)\hat{x}_k(-)]$

Tabela 3.2: Algoritmo do filtro PSELIKA-MLP.

Capítulo 4 – Simulações Numéricas

Este capítulo tem como principal objetivo validar a utilização de redes neuronais do tipo MLP para o cálculo da matriz de ganho segundo os métodos propostos na secção 3.3. Os casos de estudo são a estimação da taxa angular e atitude de um veículo aeroespacial com base em medições de quaterniões e a estimação de uma órbita de um satélite artificial a partir de medições bastante imprecisas de um radar hipotético.

A validação será feita a partir da comparação do algoritmo PSELIKA-MLP com os algoritmos PSELIKA e EKF descritos na secção 2.1. As comparações serão feitas em termos de exatidão na estimação, além da complexidade temporal, ou seja, tempo computacional e da complexidade espacial, que significa, número e tamanho de parâmetros e variáveis utilizadas durante o algoritmo.

Para que a comparação entre os diferentes algoritmos fosse eficiente, foram considerados valores elevados de ruído. Além disso, para efeitos de comparação e visualização, foram assumidos os valores das matrizes de covariância do erro de processamento (Q) e de medição (R).

4.1 Caso 1: Atitude e taxa angular de um veículo aeroespacial

A medição da taxa angular de um veículo aeroespacial ou satélite de forma precisa é necessária para a determinação de sua atitude, assim como para a amortização do controlo de atitude. Há uma tendência em se construir veículos aeroespaciais cada vez menores mais leves e baratos, podendo comprometer a exatidão na determinação da atitude. Existem diversas maneiras de realizar a medição da taxa angular, sendo realizado anteriormente principalmente por meio de giroscópios. Entretanto, esse meio tem sido substituído por outros instrumentos mais leves, sendo necessário utilizar outros métodos de cálculo da taxa angular em veículos sem giroscópios. [20]

Existem diversas maneiras de obter a taxa angular. Se a atitude for conhecida é possível derivar o seu valor em relação aos parâmetros que conectam a atitude a taxa angular, a partir das equações da dinâmica angular de um satélite. Entretanto, este processo de diferenciação introduz um componente considerável de ruído no valor calculado do vetor de taxa angular. Para compensar este ruído é possível utilizar um filtro ativo como o filtro de Kalman. [20]

Esse método utiliza as derivadas dos parâmetros da atitude ou das direções medidas. Outra abordagem, é utilizar os próprios parâmetros de atitude ou direções medidas como

medições no filtro de Kalman. O modelo da dinâmica deste KF também inclui a equação da dinâmica angular do veículo aeroespacial que é uma equação diferencial não-linear de primeira ordem. Neste caso, a equação da cinemática que conecta os parâmetros da atitude ou as direções com as suas derivadas estão incluídas no modelo da dinâmica utilizados pelo filtro, então a necessidade de diferenciação é eliminada. [20]

Novos tipos de sensores que utilizam rastreadores de estrelas retornam o valor da atitude do veículo aeroespacial em termos de quaterniões. Assim, é possível utilizar estas medições em quaterniões para o filtro. Essas medições permitem que o modelo de medição seja efetuado de forma linear. Essa abordagem foi realizada em [20] onde a equação da dinâmica não-linear foi convertida para uma equação pseudo-linear dependente do estado, podendo assim ser utilizado o filtro PSELIKA. Esse modelo será então utilizado para a validação do PSELIKA-MLP.

A secção a seguir apresenta os modelos matemáticos que descrevem a dinâmica angular de um veículo aeroespacial, assim como em seu formato pseudo-linear com a factorização utilizada para o modelo utilizado no filtro PSELIKA. O mesmo é feito para o modelo de observação dos dados.

4.1.1 Modelo Dinâmico

A equação da dinâmica angular de um veículo aeroespacial pode ser descrita da seguinte forma [146]:

$$I_{SC}\dot{\omega} + \dot{h}_{SC} + \omega \times (I_{SC}\omega + h_{SC}) = T_{SC} \tag{4.1}$$

Neste caso, I_{SC} corresponde a matriz de inércia do veículo, ω o vetor da taxa angular do veículo, h_{SC} o vetor do momento angular das rodas de momento do veículo e T_{SC} o somatório dos momentos externos aplicados ao veículo.

Dado que I_{SC} é inversível, é possível reescrever esta equação da seguinte maneira no formato pseudo-linear:

$$\dot{\omega} = A'(\omega)\omega + u(t) \tag{4.2}$$

Onde

$$A'(\omega) = {I_{SC}}^{-1}[(I_{SC}\omega + h_{SC})\times] \tag{4.3}$$

$$u(t) = I_{SC}^{-1}(T_{SC} - \dot{h}_{SC}) \tag{4.4}$$

Onde $[(h)\ \times]$ representa a matriz de produto externo de um vetor $h = [h_1 \quad h_2 \quad h_3]^T$

Assim, a sua matriz de produto externo é dada por:

$$[(h)\times] = \begin{bmatrix} 0 & -h_3 & h_2 \\ h_3 & 0 & -h_1 \\ -h_2 & h_1 & 0 \end{bmatrix} \tag{4.5}$$

$F'(w)$ apresenta exatamente oito maneiras diferentes de expressar a equação não linear da dinâmica de um veículo aeroespacial [20].

Para a descrição da atitude o melhor a ser feito é utilizar quaterniões de rotação. A equação da dinâmica pode ser descrita por [146]:

$$\dot{q} = \frac{1}{2} Q\omega \tag{4.6}$$

Onde:

$$Q = \begin{bmatrix} q_4 & -q_3 & q_2 \\ q_3 & q_4 & -q_1 \\ -q_2 & q_1 & q_4 \\ -q_1 & -q_2 & -q_3 \end{bmatrix} \tag{4.7}$$

Assim ao reunir (4.2) e (4.6) tem-se:

$$\begin{bmatrix} \dot{\omega} \\ \dot{q} \end{bmatrix} = \begin{bmatrix} I^{-1}[(I\omega + h)\times] & 0 \\ \frac{1}{2}Q & 0 \end{bmatrix} \begin{bmatrix} \omega \\ q \end{bmatrix} + \begin{bmatrix} I^{-1}(T - \dot{h}) \\ 0 \end{bmatrix} \tag{4.8}$$

O modelo de medições a partir dos quaterniões é bastante simples, sendo linear e obtido a partir das medições dos quaterniões

$$z = [0_{3\times4} \quad I_4] \begin{bmatrix} \omega \\ q \end{bmatrix} \tag{4.9}$$

Onde $0_{3\times4}$ representa uma matriz nula de dimensões 3×4 e I_4 representa a matriz identidade de ordem 4.

Visto que as equações que descrevem o modelo da dinâmica angular e de medição são descritas por equações pseudo-lineares dependentes do estado, então podem ser utilizadas no filtro PSELIKA a partir das equações em (3.18).

Descrição do filtro PSELIKA

A partir das equações que descrevem o modelo da dinâmica e de medição é possível formar o filtro PSELIKA

Seja:

$$x = \begin{bmatrix} \omega \\ q \end{bmatrix} \tag{4.10}$$

Então é possível reescrever (4.8) e (4.9) como um modelo estocástico de acordo com as seguintes equações

$$\begin{aligned} \dot{x} &= A(x)x + u(t) + w(t) \\ z &= H(x)x + v(t) \end{aligned} \tag{4.11}$$

Onde

$$\begin{aligned} A(x) &= \begin{bmatrix} I^{-1}[(I\omega + h)\times] & 0 \\ \frac{1}{2}Q & 0 \end{bmatrix} \\ H(x) &= [0_{3\times 4} \quad I_4] \end{aligned} \tag{4.12}$$

A partir deste modelo de filtro PSELIKA, serão baseados as etapas de treinamento e os resultados numéricos.

4.1.2 Treinamento de Rede

Em relação ao treinamento da rede, após análise experimental deste caso, percebeu-se que para um valor semelhante de coeficiente de desempenho R^2, a rede que desempenhava melhor, em termos de custo computacional para o filtro, era aquela que apresentava menor número de camadas.

Assim, o treinamento de rede limitar-se-á a seleção do número de neurónios da única camada escondida, em função do coeficiente de desempenho. Esta decisão foi tomada de forma a facilitar e acelerar a análise de treinamento de rede, uma vez que o número de combinações de camadas e neurónios por camada é infinito.

O primeiro passo para o treinamento desta rede foi definir o domínio de funcionamento do sistema. Este valor dependerá do sistema e das possíveis condições que serão impostas ao sistema. Como método de validação selecionou-se para os elementos do vetor da taxa angular foi decidido que seu valor absoluto máximo será de 1 [rad/s]. Para os elementos dos

quaterniões o seu valor absoluto máximo é determinado pela definição dos quaterniões, onde o módulo do vetor quaternião tem valor máximo de 1. Sendo assim,

$$\begin{aligned} |\{\omega_i\}| \leq 1, \forall i \\ |\{q_i\}| \leq 1, \forall i \end{aligned} \tag{4.13}$$

Para este treinamento considerou-se que o momento angular das rodas de momento assim como a sua derivada são constantes e têm valor inicial:

$$\begin{aligned} h_{SC} = [1 \quad 1 \quad 1]^T kg \cdot \frac{m^2}{s} \\ \dot{h_{SC}} = 0 \end{aligned} \tag{4.14}$$

Visto que a matriz $A(x)$ é dependente do valor de h_{SC}, alguma alteração deste valor provocaria alteração na matriz $A(x)$, sendo necessário recalcular o seu valor e consequentemente resolver novamente a equação de Riccati dependente do estado. Para que fosse possível a utilização de uma função $h_{SC}(t)$ seria necessário adicionar esta variável como um dos estados, ou como parâmetro para o treinamento de rede. Contudo, este procedimento não será efetuado, tendo em vista que o objetivo é apenas validar o uso deste filtro. A mesma lógica se aplica para as matrizes de covariância Q e R.

Por fim, o número de vetores estado selecionados para a realização do treinamento neste caso foi de cinco mil pontos, os quais foram gerados de forma aleatória, segundo uma distribuição uniforme dentro dos domínios selecionados. Destes pontos, 70% do conjunto foi utilizado para treinamento enquanto o restante foi utilizado para a etapa de teste. Nos treinamentos efetuados para esta rede, para os diferentes valores de neurónios, foram utilizados os mesmos conjuntos de estados para treinamento, e os mesmos para o teste, sendo calculado o valor do coeficiente de desempenho R^2 para ambos.

A matriz de inércia utilizada para o veículo é:

$$I_{SC} = \begin{bmatrix} 20.26 & -1.37 & -2.17 \\ -1.37 & 23.25 & -0.37 \\ -2.17 & -0.37 & 27.83 \end{bmatrix}$$

As matrizes de covariância utilizadas para a etapa de treinamento foram:

$$Q_k = (0.01)^2 I_7 \quad R_k = (2)^2 I_4$$

Onde I_k representa a matriz identidade de ordem k, Q_k é a matriz de covariância do processamento e R_k a matriz de covariância da medição.

Resultados do treinamento

O treinamento foi efetuado com um número de neurónios dentro do intervalo de 20 a 300 e os resultados podem ser visualizados na Figura 4.1.

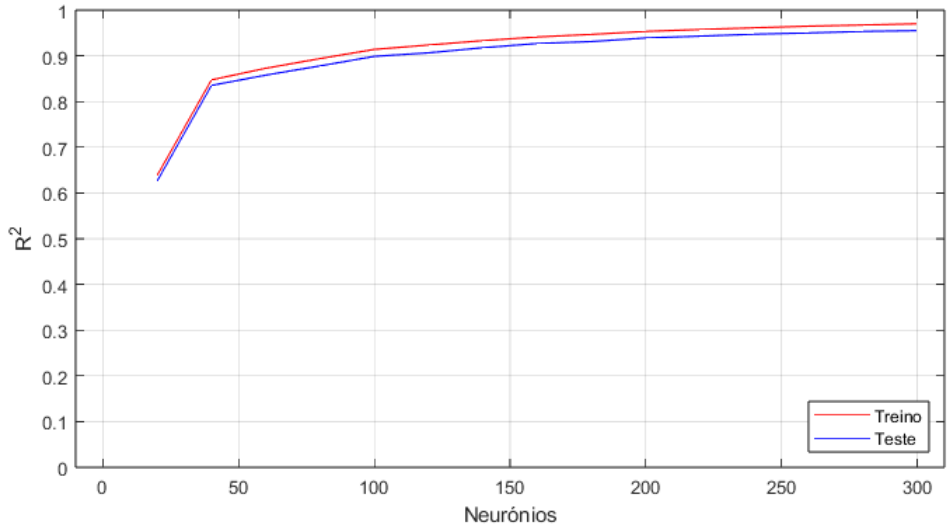

Figura 4.1: Coeficiente de desempenho do treinamento do caso de estudo 1 em função do número de neurónios.

Em relação ao desempenho do filtro, após uma análise experimental, verificou-se que o número de neurónios não influencia muito o resultado para valores acima de 200 neurónios. Além disso, a partir de 200 neurónios o valor do coeficiente R^2 não aumenta substancialmente. Tendo em vista também o custo computacional, foi escolhido o número de 200 neurónios para a camada escondida, já que um número maior de neurónios resulta num custo computacional superior. Foi obtido o valor de R^2=0.953 para o conjunto de treino e de R^2=0.939 para o conjunto de teste no caso de 200 neurónios.

É importante notar que os resultados obtidos são função direta do conjunto de treinamento assim como a fração do conjunto de pontos e seleção do domínio, além do método selecionado para avaliação da rede. Sendo assim, a escolha de novos parâmetros de treinamento podem levar a resultados bastante diferentes, entretanto esta análise está fora do âmbito deste trabalho.

4.1.3 Simulação e resultados

As equações em (4.8) são utilizadas para a simulação das atitudes e taxas angulares de referência. Estes parâmetros são simulados em Matlab e com o auxílio do método de Runge-Kutta (RK) de quarta ordem [147]. Assim, para esta simulação considerou-se um passo de simulação de T_s =0.01s e número de passos N_{passos}=3000. O seguinte valor de estado foi utilizado como valor inicial da simulação:

$$[\omega_1\ \omega_2\ \omega_3\ q_1\ q_2\ q_3\ q_4]^T = [0.300\ \ 0.200\ \ 0.500\ \ 1.000\ \ 0.000\ \ 0.000\ \ 0.000]^T$$

Considerou-se os seguintes valores para o desvio padrão do erro:

$$\sigma_{quat} = 0.1$$

$$\sigma_Q = 0.01 \quad \sigma_R = 2$$

Onde σ_{quat} representa o valor do desvio padrão do erro associado aos elementos do vetor de medição dos quaterniões. Além disso, σ_Q e σ_R representam os valores do desvio padrão utilizados para as matrizes Q_k e R_k respetivamente, em ambos filtros PSELIKA e EKF.

Considerou-se também que nenhum momento externo é aplicado ao veículo aeroespacial, tendo valor nulo ao longo da simulação.

$$T_{SC}(t) = 0\ [N.m]$$

A Figura 4.2 representa um exemplo do conjunto de simulações realizadas onde medições foram efetuadas por um sensor em vetor de quaterniões segundo os valores de desvio padrão do ruído mencionados e o estado inicial considerado.

Valores iniciais para o filtro $\hat{x}_0$ e P_o

O valor utilizado como estado inicial $\hat{x}_0$ para a inicialização dos 3 filtros será considerado como se o estado inicial não fosse conhecido, assim será:

$$\hat{x}_0 = [\omega_1\ \omega_2\ \omega_3\ q_1\ q_2\ q_3\ q_4]^T = [0.000\ \ 0.000\ \ 0.000\ \ 0.000\ \ 0.000\ \ 0.000\ \ 1.000]^T$$

O valor utilizado como matriz de covariância P_o para os filtros PSELIKA e EKF será a mesma, sendo assumido que é desconhecida.

$$P_o = I_7$$

Para comparar os diferentes filtros de forma direta em termos de exatidão será utilizado o índice RMSE, ou seja, a raiz quadrada do erro de estimação. Neste caso, serão efetuados N_{sim} = 10000 simulações de Monte Carlo. As simulações de Monte Carlo efetuadas serão justamente em relação ao ruído branco adicionado à medição, conforme uma distribuição

normal, com desvio padrão igual ao mencionado nesta secção para cada variável de observação. Neste caso, o RMSE será calculado para o vetor de taxa angular e o vetor quaternião estimados em cada instante t_k.

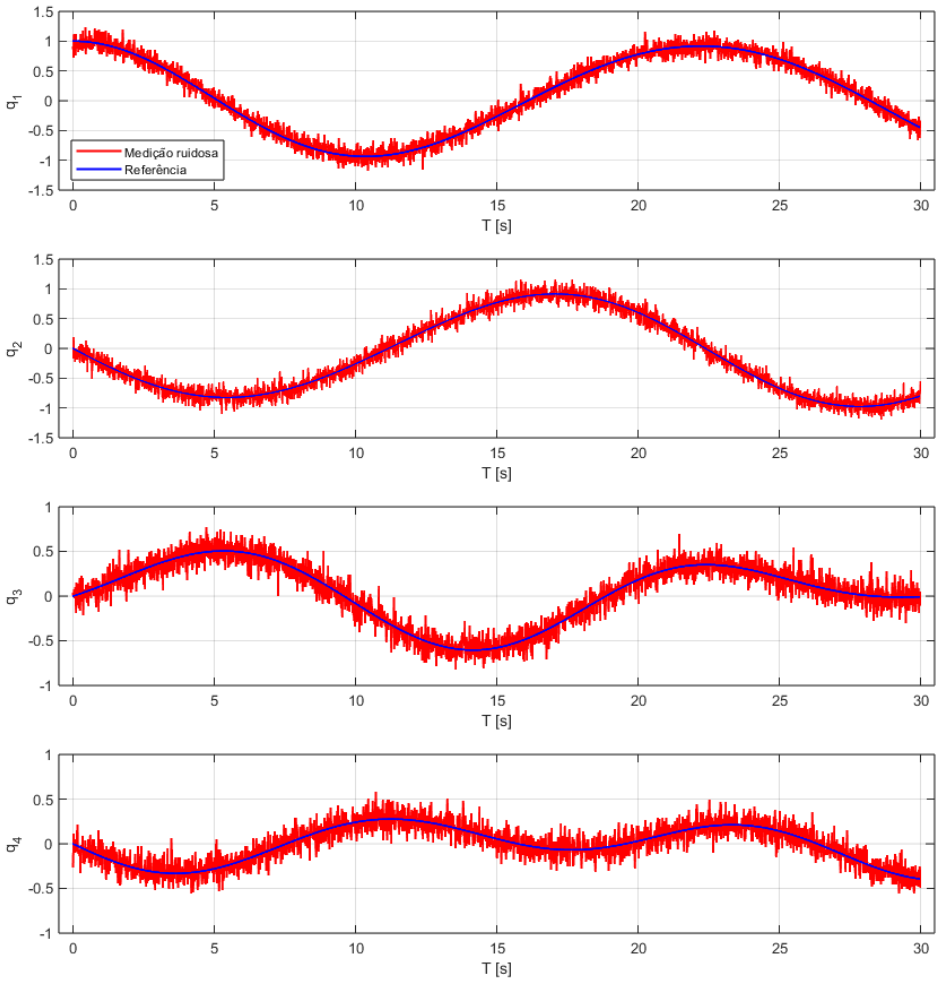

Figura 4.2: Exemplo de uma das simulações de Monte Carlo para o caso de estudo 1.

Para a taxa angular, são comparados diretamente os vetores estimados com os da simulação de referência sendo o mesmo realizado para o vetor quaternião. As expressões estão descritas nas equações a seguir:

$$RMSE_{\omega,k} = \sqrt{\frac{\sum_{i=1}^{N_{sim}} \left(\left|\left|(\omega_k - \hat{\omega}_i)\right|\right|_2^{\,2} \right)}{N_{sim}}} \tag{4.15}$$

$$RMSE_{q,k} = \sqrt{\frac{\sum_{i=1}^{N_{sim}} \left(\left|\left|(q_k - \hat{q}_i)\right|\right|_2^{\,2} \right)}{N_{sim}}} \tag{4.16}$$

Onde $||x||_2$ é a norma euclidiana do vetor x de dimensão n dada por $||x||_2 = \sqrt{x_1^2 + \cdots + x_n^2}$, e o índice k corresponde ao parâmetro dado para o instante t_k.

Outro índice de desempenho que será comparado será o erro máximo obtido para cada um dos instantes t_k dentre as N_{sim} simulações, também conhecido por norma máxima ou norma infinita. Esse índice permite perceber qual o comportamento dos filtros nos piores casos.

O valor máximo medido para os dois casos será dado por:

$$||\Omega_k||_\infty = max\left(||(\omega_k - \hat{\omega}_i)||_2\right)_{i=1}^{N_{sim}} \tag{4.17}$$

$$||\Xi_k||_\infty = max\left(||(q_k - \hat{q}_i)||_2\right)_{i=1}^{N_{sim}} \tag{4.18}$$

Onde Ω_k representa o conjunto de N_{sim} valores da norma euclidiana da diferença entre o valor obtido de ω numa simulação de Monte Carlo i e o valor de referência para um instante t_k, e Ξ_k representa o mesmo, mas em relação aos vetores de quaternião obtidos para o instante t_k. A função $max(\cdot)_{i=1}^{N_{sim}}$ representa o valor máximo do conjunto de N_{sim} elementos.

4.1.3.1 Exatidão

O desempenho dos filtros é então avaliado a partir dos resultados gráficos das equações (4.15) e (4.16) para o RMSE e (4.17) e (4.18) para o erro máximo, sendo assim possível fazer a sua comparação. Os resultados estão representados nas Figura 4.3, Figura 4.4, Figura 4.5 e Figura 4.6. Nestas figuras, as linhas em preto, verde e azul representam respetivamente os filtros EKF, PSELIKA e PSELIKA-MLP.

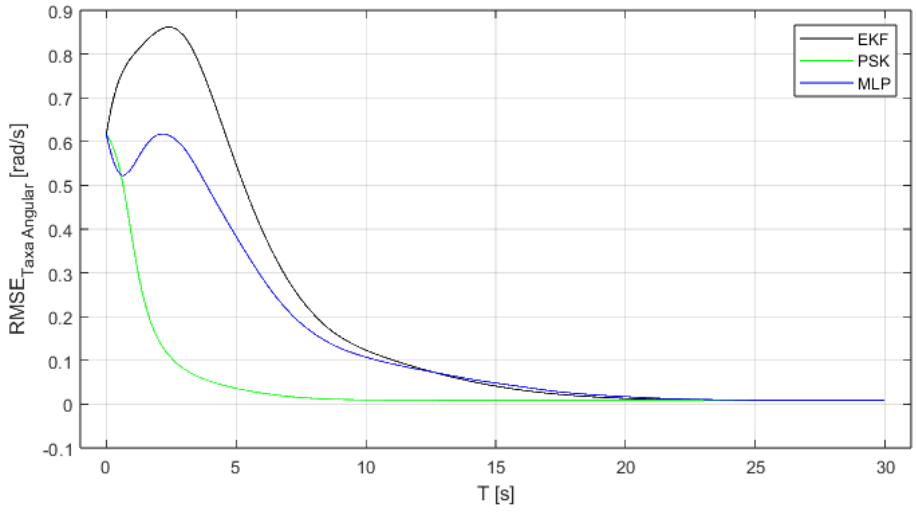

Figura 4.3: Comparação do RMSE obtido EKF, PSELIKA e PSELIKA – MLP para o vetor taxa angular.

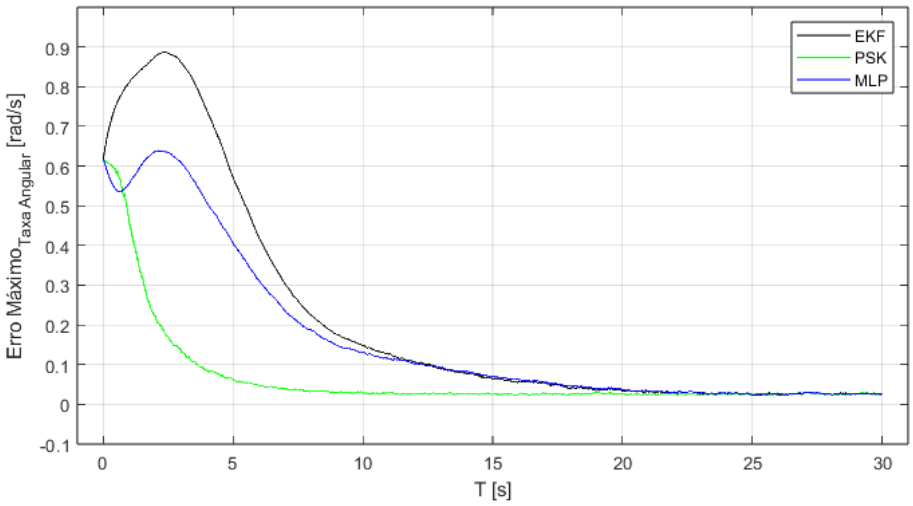

Figura 4.4: Comparação do erro máximo obtido do EKF, PSELIKA e PSELIKA – MLP para o vetor taxa angular.

É possível observar na Figura 4.3 e na Figura 4.4, respetivamente, o RMSE da taxa angular assim como seu valor do erro máximo em função do instante t_k em segundos. É possível perceber que o RMSE apresenta valor inicial elevado e que se reduz ao longo do tempo para os três filtros, e o mesmo ocorre para o erro máximo. A oscilação que ocorre ao início, tanto no filtro EKF, como no PSELIKA-MLP, deve-se ao seu possível comportamento errático que, neste caso, atribui-se ao erro relativamente grande de estimação inicial, já que o valor para estado inicial se encontra afastado do valor inicial verdadeiro.

O valor inicial elevado deve-se justamente pela estimativa inicial do estado de cada filtro ser bastante diferente do valor correto do estado inicial. Ainda assim, os três filtros foram capazes de convergir para um valor cada vez menor de RMSE e Erro máximo até atingir um valor estável. Dos três filtros, aquele que apresentou a convergência mais rápida foi o PSELIKA, seguido do EKF e do PSELIKA-MLP. Entretanto, o EKF apresentou o maior pico em relação a ambos os gráficos com um RMSE máximo de aproximadamente 0.86 rad/s e erro máximo de 0.89 rad/s, um comportamento esperado, justamente pelas limitações do filtro.

O PSELIKA-MLP não apresentou convergência mais rápida, mas demonstrou desempenho semelhante aos outros filtros após a estabilização. O desempenho do PSELIKA-MLP foi superior ao do EKF até aproximadamente 12.62 segundos e superior ao PSELIKA até 0.56 segundos, mas inferior em todos os outros instantes até estabilização. A estabilização ocorre para o RMSE de aproximadamente 0.009 rad/s e para o erro máximo de aproximadamente 0.026 rad/s, próximo dos 25 segundos.

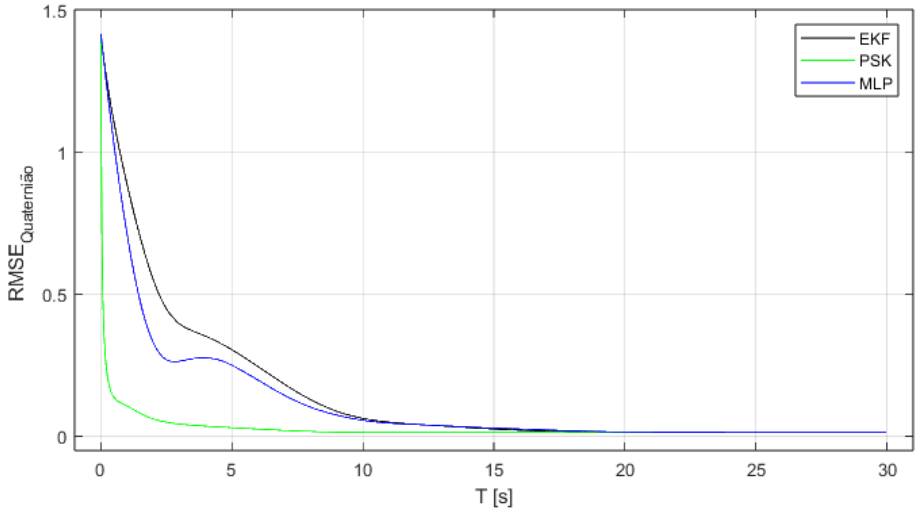

Figura 4.5: Comparação do RMSE obtido EKF, PSELIKA e PSELIKA – MLP para o vetor quaternião.

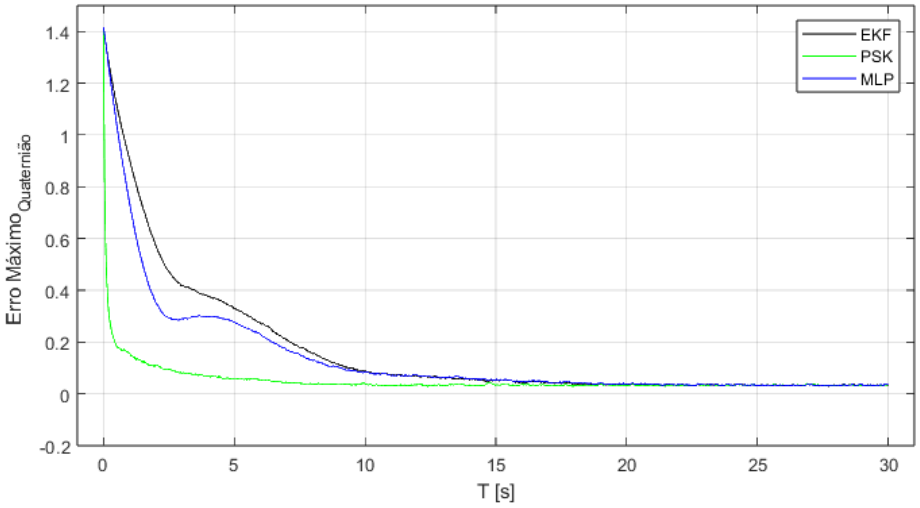

Figura 4.6: Comparação do erro máximo obtido do EKF, PSELIKA e PSELIKA – MLP para o vetor quaternião.

É possível observar na Figura 4.5 e na Figura 4.6, respetivamente, o RMSE dos quaterniões assim como seu valor do erro máximo em função do instante t_k em segundos. O desempenho em relação aos filtros para os quaterniões é semelhante ao da taxa angular. Entretanto, neste caso, o valor máximo de RMSE e de erro máximo é igual para os três filtros tendo valor de aproximadamente 1.41.

Em relação ao RMSE e erro máximo ao longo do tempo, apresenta quase o mesmo comportamento gráfico em relação a taxa angular anteriores onde o desempenho do PSELIKA-MLP foi superior ao do EKF até aproximadamente 12.5 segundos, mas inferior em todos os instantes em comparação ao PSELIKA até estabilização. A estabilização ocorre para o RMSE de aproximadamente 0.013 e para o erro máximo de aproximadamente 0.035, próximo dos 25 segundos.

4.1.3.2 Complexidade Temporal

A complexidade temporal é uma medida de quanto tempo o algoritmo utiliza para ser executado em função do tamanho da entrada.

Em relação a complexidade temporal dos filtros a comparação foi feita com base no tempo médio de execução dos algoritmos após 1000 simulações de Monte Carlo. A comparação será feita em relação ao algoritmo PSELIKA-MLP com o ganho ou perda relativa de tempo em relação aos outros filtros.

Os resultados obtidos foram um ganho de aproximadamente 23.50% de tempo em comparação ao algoritmo PSELIKA e um ganho de aproximadamente 60.23% de tempo em comparação ao algoritmo EKF. O algoritmo PSELIKA-MLP foi mais rápido em relação aos outros algoritmos, o que já era esperado, visto que a rede neuronal utilizada é bastante simples. Os desvios padrões associados obtidos são de aproximadamente 0.523% e 0.585% respetivamente.

O que torna os outros algoritmos mais complexos em termos computacionais é a necessidade de realizar diversas multiplicações matriciais para determinar os valores da matriz de ganho K, além de uma inversão matricial e a necessidade de realizar a propagação e correção da matriz de covariância P. Ainda assim, o algoritmo EKF foi bem mais custoso justamente por necessitar calcular a matriz Jacobiana discretizada do modelo do sistema e de a matriz Jacobiana do modelo de medição. No algoritmo PSELIKA a matriz discretizada do modelo do sistema A_d é utilizada para a propagação do estado estimado e a propagação da matriz de covariância, sendo necessário calculá-la apenas uma vez.

4.1.3.3 Complexidade Espacial

A complexidade espacial é uma medida de espaço de memória utilizado por um algoritmo até o fim de sua execução.

Para a comparação será utilizada a função *whos* do software Matlab que permite observar o espaço ocupado na memória por cada variável. Todas variáveis foram guardadas em formato *double*, ou seja, o espaço ocupado por um único elemento de um vetor ou matriz ocupará 8 bytes ou 64 bits.

O algoritmo PSELIKA-MLP foi o que apresentou maior uso de espaço de memória com um total de 59880 bytes utilizados sendo destes 58408 bytes utilizados apenas para a execução da rede neuronal implementada, que inclui as matrizes dos parâmetros da rede neuronal e os fatores utilizados para a padronização do estado, correspondendo a aproximadamente 97.54% do espaço total necessário. Em segundo lugar o algoritmo EKF necessitou de 2472

bytes seguido do PSELIKA com 2360 bytes, ocupando 95.77% e 95.96% a menos de espaço de memória respetivamente.

4.2 Caso 2: Órbita de satélite artificial

É de extrema importância a estimação e previsão acerca do estado de um satélite em órbita, para que haja constante monitorização acerca de seu funcionamento. Por isso, existem diversos radares ao longo do planeta com o objetivo de realizar medições e a partir destas estimar os parâmetros desta órbita.

A dinâmica de um satélite é afetada por diversos fenômenos físicos, que vão desde efeitos aerodinâmicos, gravitacionais e de radiação, assim como os próprios atuadores ou propulsores presentes no satélite. Estas fazem que o sistema de equações dinâmicas que descrevem o seu movimento seja bastante não-linear.

A secção a seguir apresenta os modelos matemáticos que descrevem a dinâmica orbital de um satélite, assim como em seu formato pseudo-linear com a factorização utilizada para o modelo utilizado no filtro PSELIKA. O mesmo é feito para o modelo de observação dos dados.

4.2.1 Modelo Dinâmico

Para o modelo dinâmico implementado, será assumido que o radar utilizado para a medição da posição encontra-se no centro da Terra em um ponto fixo, meramente para a demonstração da validade do filtro implementado.

As equações do modelo dinâmico serão utilizadas em coordenadas esféricas facilitando a sua implementação. Além disso, de todas as perturbações que poderiam afetar a órbita do satélite escolheu-se manter apenas a perturbação devido à variação do potencial gravitacional terrestre, uma vez que é a mais influente [148]. Assim, os demais efeitos, como os aerodinâmicos, gravitacionais de outros planetas e de pressão de radiação solar serão desprezados. O satélite será modelado como um ponto de massa para as equações dinâmicas.

O sistema de coordenadas esféricas a ser utilizado pode ser visualizado na Figura 4.7 a seguir, onde r corresponde à distância, ϕ corresponde ao zénite e θ ao azimute.

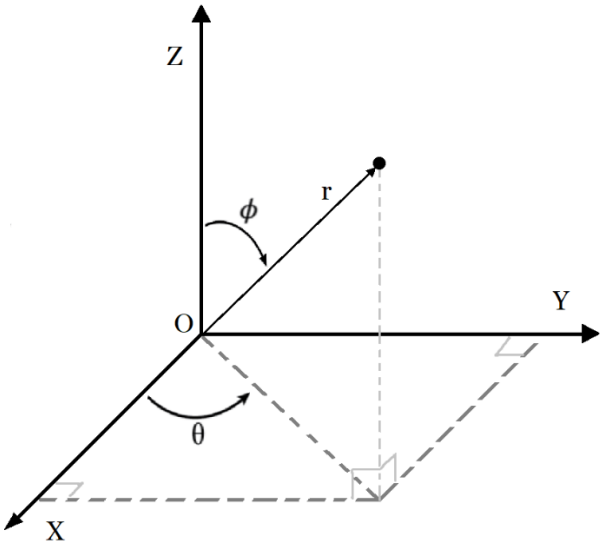

Figura 4.7: Sistema de coordenadas esféricas utilizado para o caso de estudo 2.

Os parâmetros universais que são utilizados para a computação são:

$$r_{Terra} = 6378.1370\ km$$

$$G = 6.67430 \times 10^{-11} \frac{m^3 \cdot s}{kg}$$

$$M_{Terra} = 5.9722 \times 10^{24} kg$$

$$\mu = G \cdot M_{Terra}$$

$$J_2 = 1.08264 \times 10^{-3}$$

Dentro destes parâmetros, r_{Terra} representa o raio equatorial terrestre, G a constante universal gravitacional, M_{Terra} representa a massa terrestre, μ representa o parâmetro gravitacional terrestre, por fim a constante J_2 representa o parâmetro de achatamento terrestre, também conhecido como segundo harmónico zonal.

As equações que descrevem a dinâmica da órbita de um satélite, em coordenadas esféricas, sujeita aos efeitos gravitacionais devidos ao achatamento da Terra são [149]:

$$\ddot{r} = r\dot{\theta}\, sin^2\, \phi + r\dot{\phi} - \frac{\mu}{r^2} + \frac{3}{2}\mu J_2 r_{Terra}^2 \frac{(3\, cos^2\, \phi - 1)}{r^4} + u_r \qquad (4.19)$$

$$\ddot{\theta} = \frac{-2\dot{r}\dot{\theta}}{r} - 2\dot{\theta}\dot{\phi}\, cot\, \phi + \frac{u_\theta}{r\, sin\, \phi} \qquad (4.20)$$

$$\ddot{\phi} = \frac{-2\dot{r}\dot{\phi}}{r} + \dot{\theta}^2 sin\phi cos\phi + 3\mu J_2 \frac{r_{Terra}^2}{r^5}\, cos\phi sin\phi + \frac{u_\phi}{r} \qquad (4.21)$$

Nestas equações u_r, u_θ e u_ϕ são, respetivamente, as acelerações devidas aos propulsores presentes no satélite na direção radial, na direção tangencial em relação ao azimute e na direção tangencial em relação ao zénite.

Modelo geral do PSELIKA

Uma vez que as equações que descrevem este modelo são do segundo grau, podemos reescrever o sistema de equações diferenciais de forma que o vetor de estado apresente as demais variáveis com derivativas. Isso pode ser feito aumentando o modelo e incluindo as variáveis $\dot{r}$, $\dot{\theta}$ e $\dot{\phi}$ para o conjunto de variáveis de estado. Assim, o vetor estado x será dado por:

$$x = \left[r\ \theta\ \phi\ \dot{r}\ \dot{\theta}\ \dot{\phi}\right]^T$$

Seguindo as diretrizes da secção 3.2.1 é possível realizar a transformação das equações que descrevem o modelo dinâmico para uma estrutura pseudo-linear. Uma vez que a variável r não será nula ao longo de uma órbita existente, é possível utilizar esta variável para escrever termos constantes em função da variável de estado.

Desta forma, as equações (4.19), (4.20) e (4.21) podem ser reescritas para o seguinte formato pseudo-linear matricial:

$$\ddot{r} = \left[-\frac{\mu}{r^3} + \frac{3}{2}\mu J_2 r_{terra}^2 \frac{3\cos^2\phi - 1}{r^5} \quad 0 \quad 0 \quad 0 \quad r\dot{\theta}\sin^2\phi \quad r\dot{\phi}\right]\left[r\ \theta\ \phi\ \dot{r}\ \dot{\theta}\ \dot{\phi}\right]^T + u_r \tag{4.22}$$

$$\ddot{\theta} = \left[0 \quad 0 \quad 0 \quad -\frac{2\dot{\theta}}{r} \quad 0 \quad -2\dot{\theta}\cot\phi\right]\left[r\ \theta\ \phi\ \dot{r}\ \dot{\theta}\ \dot{\phi}\right]^T + \frac{u_\theta}{r\sin\phi} \tag{4.23}$$

$$\ddot{\phi} = \left[\frac{3\mu J_2 r_{terra}^2}{r^6}\cos\phi\sin\phi \quad 0 \quad 0 \quad -\frac{2\dot{\phi}}{r} \quad \dot{\theta}\sin\phi\cos\phi \quad 0\right]\left[r\ \theta\ \phi\ \dot{r}\ \dot{\theta}\ \dot{\phi}\right]^T + \frac{u_\phi}{r} \tag{4.24}$$

Essas três equações em conjunto com a seguinte equação:

$$\begin{bmatrix}\dot{r}\\ \dot{\theta}\\ \dot{\phi}\end{bmatrix} = [0_{3x3}\ I_3]\left[r\ \theta\ \phi\ \dot{r}\ \dot{\theta}\ \dot{\phi}\right]^T \tag{4.25}$$

Formam as equações em formato pseudo-linear que descrevem o modelo dinâmico do sistema.

Em relação ao modelo de medição, esse será efetuado com base nas medições de um radar hipotético localizado no centro da Terra. O objetivo é estimar o vetor de estado completo

com base apenas nas observações de r, θ e ϕ. Assim, o modelo de medição é linear e dado por:

$$z_k = [I_3\ 0_{3\times3}\ ]\left[\mathrm{r}\ \theta\ \phi\ \dot{\mathrm{r}}\ \dot{\theta}\ \dot{\phi}\right]^{\mathrm{T}} \tag{4.26}$$

Por fim, as equações que caracterizam o filtro pseudo-linear são:

$$\begin{gathered}\dot{x} = A(x)x + B(x)u(t) + w(t) \\ z = H(x)x + v(t)\end{gathered} \tag{4.27}$$

Onde:

$$A(x) = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ -\frac{\mu}{r^3} + \frac{3}{2}\mu J_2 r_{terra}^2 \frac{3\cos^2\phi - 1}{r^5} & 0 & 0 & 0 & r\dot{\theta}\sin^2\phi & r\dot{\phi} \\ 0 & 0 & 0 & -\frac{2\dot{\theta}}{r} & 0 & -2\dot{\theta}\cot\phi \\ \frac{3\mu J_2 r_{terra}^2}{r^6}\cos\phi\sin\phi & 0 & 0 & \frac{2\dot{\phi}}{r} & \dot{\theta}\sin\phi\cos\phi & 0 \end{bmatrix} \tag{4.28}$$

$$B(x) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & \frac{1}{r\sin\phi} & 0 \\ 0 & 0 & \frac{1}{r} \end{bmatrix} \quad u(t) = \begin{bmatrix} u_r \\ u_\theta \\ u_\phi \end{bmatrix} \tag{4.29}$$

$$H(x) = [I_3 \quad 0_{3\times3}] \tag{4.30}$$

4.2.2 Treinamento de Rede

Para o treinamento desta rede foram utilizadas as mesmas considerações que a rede anterior acerca do número de camadas.

Assim, para a definição do domínio de funcionamento, utilizou-se como base o resultado da propagação realizada utilizando o método RK. Em geral, um objeto em uma órbita bem definida tende a se manter dentro de alguns limites em termos de r, θ, ϕ. Entretanto, como

as matrizes $A(x)$ e $H(x)$ não dependem de θ e $\dot{r}$, não é preciso definir o domínio destes valores nem utilizá-los como argumento durante o treinamento.

Além disso, é preciso que se inclua uma margem para o domínio do treinamento. A margem permite que os valores de estado utilizados como argumento da rede durante a utilização do filtro se mantivessem dentro do domínio dos valores utilizados para o treinamento sendo selecionada experimentalmente. Uma margem muito pequena fará com que haja maior possibilidade de divergência do filtro e uma muito grande não irá retornar os melhores resultados.

Para o caso 1, não foi preciso adicionar margem visto que não existiam problemas de divergência, entretanto para este caso já é necessário. Os possíveis valores de ϕ são definidos pelas coordenadas esféricas e a margem E_ϕ, ou seja:

$$\phi_{min} - E_\phi \leq \phi \leq \phi_{max} + E_\phi \tag{4.31}$$

Os valores de r dependerão dos valores da distância do perigeu (mínima) e apogeu (máxima) da órbita, dados por r_p e r_a, e a margem selecionada E_r:

$$r_p - E_r \leq r \leq r_a + E_r \tag{4.32}$$

Os valores de $\dot{\theta}$ e $\dot{\phi}$ também se encontram limitados pelos valores máximos e mínimos obtidos durante a propagação, assim os valores utilizados foram:

$$\begin{bmatrix} \dot{\theta}_{min} \\ \dot{\phi}_{min} \end{bmatrix} - \begin{bmatrix} E_{\dot{\theta}} \\ E_{\dot{\phi}} \end{bmatrix} \leq \begin{bmatrix} \dot{\theta} \\ \dot{\phi} \end{bmatrix} \leq \begin{bmatrix} \dot{\theta}_{max} \\ \dot{\phi}_{max} \end{bmatrix} + \begin{bmatrix} E_{\dot{\theta}} \\ E_{\dot{\phi}} \end{bmatrix} \tag{4.33}$$

Onde $E_{\dot{\theta}}$ e $E_{\dot{\phi}}$ são os valores de margem selecionados.

Os demais valores selecionados para os parâmetros que definem o domínio deste caso podem ser visualizados na Tabela 4.1. As unidades são as mesmas em relação aos respetivos estados.

Parâmetro	**Valor**
$\begin{bmatrix} r_p & r_a \\ \phi_{min} & \phi_{max} \\ \dot{\theta}_{min} & \dot{\theta}_{max} \\ \dot{\phi}_{min} & \dot{\phi}_{max} \end{bmatrix} \begin{matrix} [km] \\ [rad] \\ [rad/s] \\ [rad/s] \end{matrix}$	$\begin{bmatrix} 6919 & 6978 \\ 1.5533 & 1.5883 \\ 1.082E-3 & 1.100E-3 \\ -1.906E-5 & 1.906E-5 \end{bmatrix}$

$\begin{bmatrix} E_r \\ E_\phi \\ E_{\dot{\theta}} \\ E_{\dot{\phi}} \end{bmatrix} \begin{matrix} [km] \\ [rad] \\ [rad/s] \\ [rad/s] \end{matrix}$	$\begin{bmatrix} 20.5 \\ 1.5533 \\ 0.01 \\ 9.809E-4 \end{bmatrix}$

Tabela 4.1: Valores atribuídos aos parâmetros e margens durante o treinamento do caso de estudo 2.

Por fim, o número de vetores estado selecionados para a realização do treinamento em cada caso foi de dez mil pontos, os quais foram gerados de forma aleatória, segundo uma distribuição uniforme dentro dos domínios selecionados. Destes pontos, 90% foi utilizado para treinamento enquanto os demais foram utilizados para a etapa de teste. Em todos os treinamentos efetuados para esta rede foram utilizados os mesmos conjuntos de estados para treinamento, e os mesmos para o teste, sendo calculado o valor do coeficiente de desempenho R^2 para ambos.

As matrizes de covariância utilizadas para a etapa de treinamento foram:

$$Q_k = 0.0001 \cdot I_6$$

$$R_k = \begin{bmatrix} 1000 & 0 & 0 \\ 0 & 100 & 0 \\ 0 & 0 & 100 \end{bmatrix}$$

Onde I_k representa a matriz identidade de ordem k, Q_k é a matriz de covariância do processamento e R_k a matriz de covariância da medição.

O treinamento foi efetuado com um número de neurónios dentro do intervalo de 10 a 200 e os resultados podem ser visualizados na Figura 4.8.

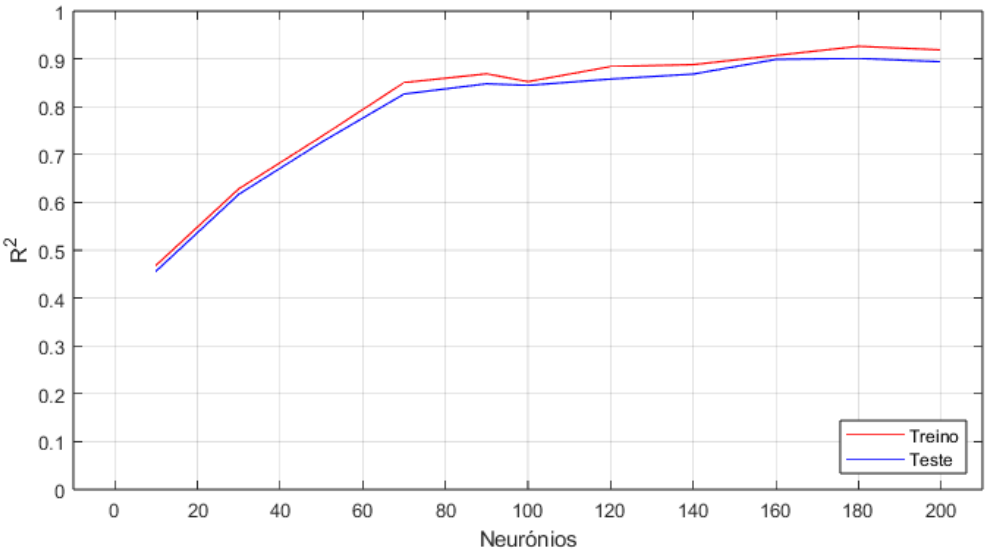

Figura 4.8: Coeficiente de desempenho do treinamento do caso de estudo 2 em função do número de neurónios.

Em relação ao desempenho do filtro, após uma análise experimental, verificou-se que o número de neurónios superior a 100 influenciou negativamente o funcionamento do filtro apesar de apresentarem um valor de R^2 superior, resultando em divergências ou valores de RMSE muito elevados. Tendo isso em conta e em conjunto ao custo computacional, foi escolhido o número de 100 neurónios para a camada escondida. Foi obtido o valor de R^2=0.852 para o conjunto de treino e de R^2=0.844 para o conjunto de teste no caso de 100 neurónios.

É importante notar que os resultados obtidos são, novamente, função direta do conjunto de treinamento assim como a fração do conjunto de pontos e seleção do domínio, além do método selecionado para avaliação da rede. Uma escolha descuidada do domínio ou do método de seleção do conjunto pode levar a maus resultados para o funcionamento do filtro como provavelmente ocorreu. Este processo é bastante experimental e custoso do ponto de vista computacional e temporal e não foi possível realizar uma análise mais aprofundada da fase de treinamento por conta destas limitações.

4.2.3 Simulação e resultados

Os elementos orbitais que definem a órbita estão descritos na Tabela 4.2 pelos elementos Keplerianos [150].

Parâmetro	**Valor**	**Unidade**
Excentricidade	0.00287709	[-]
Semi-eixo maior	6957.981	[km]
Inclinação	1.000000	[graus]
Longitude do nó ascendente	271.0000	[graus]
Argumento do perigeu	270.0000	[graus]
Anomalia verdadeira	180.0000	[graus]

Tabela 4.2: Parâmetros orbitais selecionados para o caso de estudo 2.

A partir dos valores da Tabela 4.2 é possível definir o ponto inicial para a simulação. As equações para conversão dos elementos orbitais para coordenadas esféricas são muito extensas, mas podem ser observadas em [150].

As equações (4.19), (4.20) e (4.21) são utilizadas para a simulação orbital de referência. Estes parâmetros são simulados em Matlab e com o auxílio do método de Runge-Kutta de quarta ordem [147]. Assim, para esta simulação considerou-se um passo de simulação de $T_s = 1$ segundo e número de passos $N_{passos} = 5750$. O seguinte valor de estado foi utilizado como valor inicial da simulação:

$$\left[r\ \theta\ \phi\ \dot{r}\ \dot{\theta}\ \dot{\phi}\right]^T = [6.978E+3\quad 1.7453E-2\quad 1.5533\quad 0.000\quad 1.0817E-3\quad 0.000]^T$$

É importante notar que r e $\dot{r}$ encontram-se em [km] e [km/s] nesta simulação, sendo todas as constantes convertidas para tal.

A Figura 4.9 permite visualizar a órbita, em azul, em torno da Terra, gerada pela simulação.

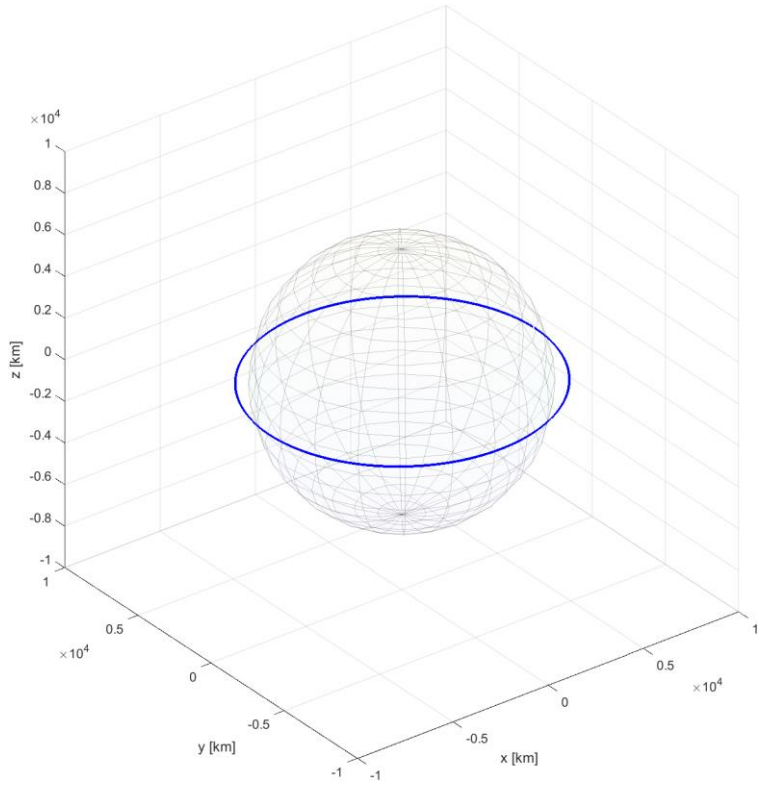

Figura 4.9: Órbita de referência utilizada para o caso de estudo 2 (Globo terrestre representado pelas linhas circulares claras cinzentas).

Considerou-se os seguintes valores para o desvio padrão utilizados para a simulação de cada variável de medição:

$$\sigma_r = 0.056234\ [km] \quad \sigma_\theta = 0.056234\ [rad]\ \ \sigma_\phi = 0.056234\ [rad]$$

As matrizes de covariância utilizadas para a filtragem em ambos os filtros PSELIKA e EKF foram:

$$Q_k = 0.0001 \cdot I_6$$

$$R_k = \begin{bmatrix} 1000 & 0 & 0 \\ 0 & 100 & 0 \\ 0 & 0 & 100 \end{bmatrix}$$

Onde σ_r, σ_θ e σ_ϕ representam, respetivamente, o valor do desvio padrão do erro associado à distância r em [km], ao ângulo θ em [rad] e ao ângulo ϕ em [rad].

Considerou-se também que nenhuma aceleração a partir de propulsores é aplicada ao satélite, tendo valor nulo ao longo da simulação.

$$|u(t)| = 0\ m/s^2$$

A Figura 4.10 representa um exemplo do conjunto de simulações realizadas onde medições foram efetuadas pelo radar hipotético segundo os valores de desvio padrão do ruído utilizados. A medição do raio apresenta ruído, entretanto, devido a sua dimensão, é difícil visualizá-lo.

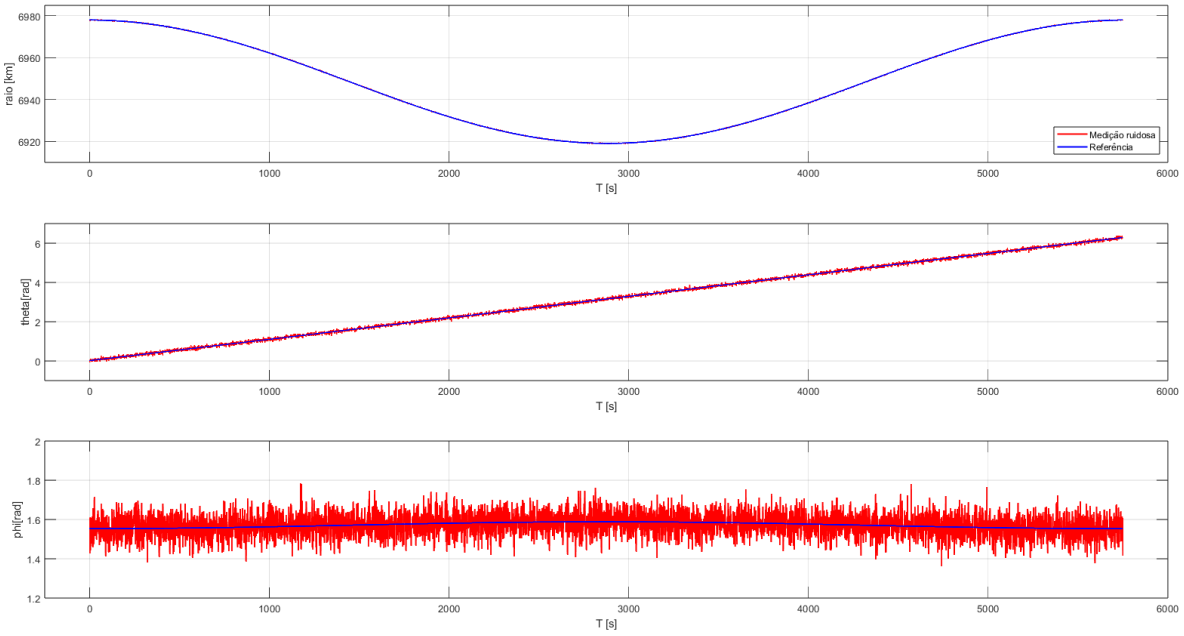

Figura 4.10: Exemplo de uma das simulações de Monte Carlo para o caso de estudo 2.

O valor utilizado como estado estimado inicial $\hat{x}_0$ para a inicialização dos 3 filtros será considerado como o mesmo para a propagação, assim será:

$$\hat{x}_0 = \left[\mathrm{r}\ \theta\ \phi\ \dot{\mathrm{r}}\ \dot{\theta}\ \dot{\phi}\right]^{\mathrm{T}} = [6.978E+3\ \ 1.7453E-2\ \ 1.5533\ 0.000\ \ 1.0817E-3\ \ 0.000]^T$$

O valor utilizado como matriz de covariância P_o para os filtros PSELIKA e EKF será a mesma, sendo assumido que é desconhecida.

$$P_o = I_6$$

Para comparar os diferentes filtros em termos de exatidão será utilizado mais uma vez o índice RMSE onde N_{sim}=10000 é o número de simulações de Monte Carlo efetuadas e k corresponde ao instante t_k. Neste caso o RMSE é calculado para a posição e a velocidade do satélite.

Para a posição é feita a transformação do estado para o sistema de coordenadas cartesiano:

$$\begin{aligned} X_{p,k} &= r_k \cdot sin\phi_k \cdot cos\theta_k \\ Y_{p,k} &= r_k \cdot sin\phi_k \cdot sin\theta_k \\ Z_{p,k} &= r_k \cdot cos\phi_k \end{aligned} \tag{4.34}$$

Para a velocidade são comparados os elementos radiais v_r e tangenciais v_ϕ e v_θ do vetor de velocidade $\vec{V}_k$ onde:

$$\vec{V}_k = v_{r,k}\vec{r}_k + v_{\phi,k}\vec{\phi} + v_{\theta,k}\vec{\theta} \tag{4.35}$$

Tal que:

$$\begin{aligned} v_{r,k} &= \dot{r}_k \\ v_{\phi,k} &= r_k\dot{\phi} \\ v_{\theta,k} &= r_k\dot{\theta}_k sin\phi_k \end{aligned}$$

Assim a expressão utilizada para o cálculo do RMSE da posição é:

$$RMSE_{posição,k} = \sqrt{\frac{\sum_{i=1}^{N_{sim}} \left(\left(X_{p,k} - \hat{X}_{p,i}\right)^2 + \left(Y_{p,k} - \hat{Y}_{p,i}\right)^2 + \left(Z_{p,k} - \hat{Z}_{p,i}\right)^2 \right)}{N_{sim}}} \tag{4.36}$$

E a expressão utilizada para o cálculo do RMSE da velocidade é:

$$RMSE_{velocidade,k} = \sqrt{\frac{\sum_{i=1}^{N_{sim}} \left(\left(v_{r,k} - \hat{v}_{r,i}\right)^2 + \left(v_{\phi,k} - \hat{v}_{\phi,i}\right)^2 + \left(v_{\theta,k} - \hat{v}_{\theta,i}\right)^2 \right)}{N_{sim}}} \tag{4.37}$$

Assim como para o caso anterior, outro índice de desempenho a ser utilizado será o valor máximo do erro, também conhecido como norma máxima ou infinita. Da mesma forma, será escolhido o valor máximo obtido para o erro no instante t_k dentre as N_{sim} simulações.

Assim, as seguintes equações descrevem o erro máximo:

O valor máximo medido para os dois casos será dado por:

$$||Posição_k||_\infty = \max\left(\sqrt{(X_k - \hat{X}_i)^2 + (Y_k - \hat{Y}_i)^2 + (Z_k - \hat{Z}_i)^2}\right)_{i=1}^{N_{sim}} \quad (4.38)$$

$$||V_k||_\infty = \max\left(\sqrt{(v_{r,k} - \hat{v}_{r,i})^2 + (v_{\phi,k} - \hat{v}_{\phi,i})^2 + (v_{\theta,k} - \hat{v}_{\theta,i})^2}\right)_{i=1}^{N_{sim}} \quad (4.39)$$

Onde $Posição_k$ representa o conjunto de N_{sim} valores da norma euclidiana da diferença entre as coordenadas (X, Y, Z) obtidas numa simulação i de Monte Carlo e o valor de referência para um instante t_k, e $\vec{V}_k$ representa o mesmo, mas em relação aos vetores de velocidade obtidos para o instante t_k.

4.2.3.1 Exatidão

O desempenho dos filtros é então avaliado a partir dos resultados gráficos das equações (4.36) e (4.37) para o RMSE e (4.38) e (4.39) para o erro máximo, sendo assim possível fazer a sua comparação. Os resultados estão representados na Figura 4.11, na Figura 4.12, na Figura 4.13 e na Figura 4.14. Nestas figuras, as linhas em preto, verde e azul representam respetivamente os filtros EKF, PSELIKA e PSELIKA-MLP.

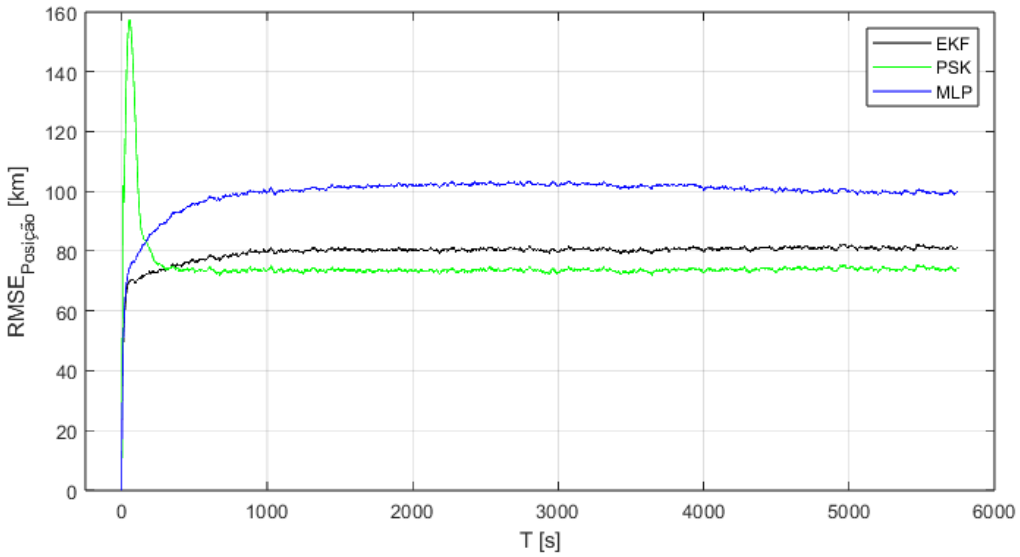

Figura 4.11: Comparação do RMSE obtido EKF, PSELIKA e PSELIKA – MLP para a posição.

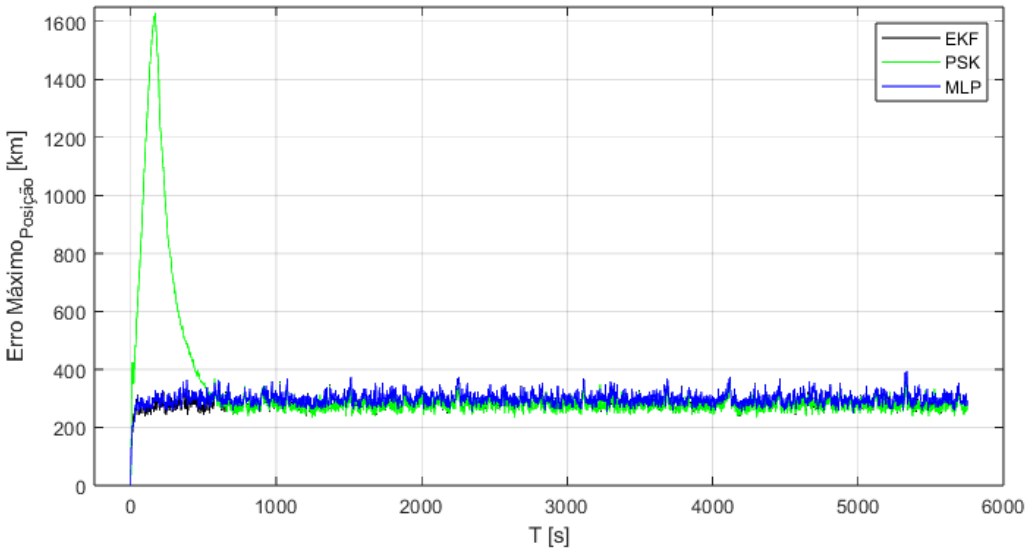

Figura 4.12: Comparação do erro máximo obtido do EKF, PSELIKA e PSELIKA – MLP para a posição.

É possível observar na Figura 4.11 e na Figura 4.12, respetivamente, o RMSE da Posição e seu valor do erro máximo em função do instante t_k em segundos. É possível perceber que o RMSE apresenta valor inicial nulo e que aumenta e atinge um intervalo estável para os filtros PSELIKA e EKF após aproximadamente 800 segundos e para o PSELIKA-MLP para aproximadamente 4500 segundos, e o mesmo ocorre para o erro máximo.

Como é possível perceber de ambos gráficos o desempenho do filtro PSELIKA é bastante reduzido no seu início de funcionamento pois atinge um pico de RMSE de aproximadamente 160 km e um erro máximo de pouco mais de 1600 km. Isso deve-se ao facto que a matriz inicial de covariância P_0 difere-se bastante do valor correto, e neste caso o filtro necessitou de diversos instantes de tempo até atingir um desempenho estável. O mesmo não ocorre para os outros filtros que gradualmente foram aumentando o valor de RMSE e erro máximo até atingir um intervalo de valores estável.

Ainda assim, apesar de apresentar o valor elevado de RMSE e erro máximo, após a estabilização o PSELIKA apresenta o melhor desempenho dos demais filtros com um valor médio de RMSE de aproximadamente 74 km, seguido do EKF com aproximadamente 81 km e por fim o PSELIKA-MLP com 100 km. É importante notar que o RMSE para o PSELIKA-MLP apresenta um comportamento quase estável aos 800 segundos, mas continua aumentando gradualmente até os 3000 segundos e depois retorna aos 4500 segundos para uma fase mais estável.

Em relação aos valores de erro máximo no período estável os três filtros apresentam valores relativamente próximos, mas o PSELIKA apresenta o melhor desempenho dos demais

filtros com um valor médio de erro máximo de aproximadamente 279 km, seguido do EKF com aproximadamente 281 km e por fim o PSELIKA-MLP com 299 km.

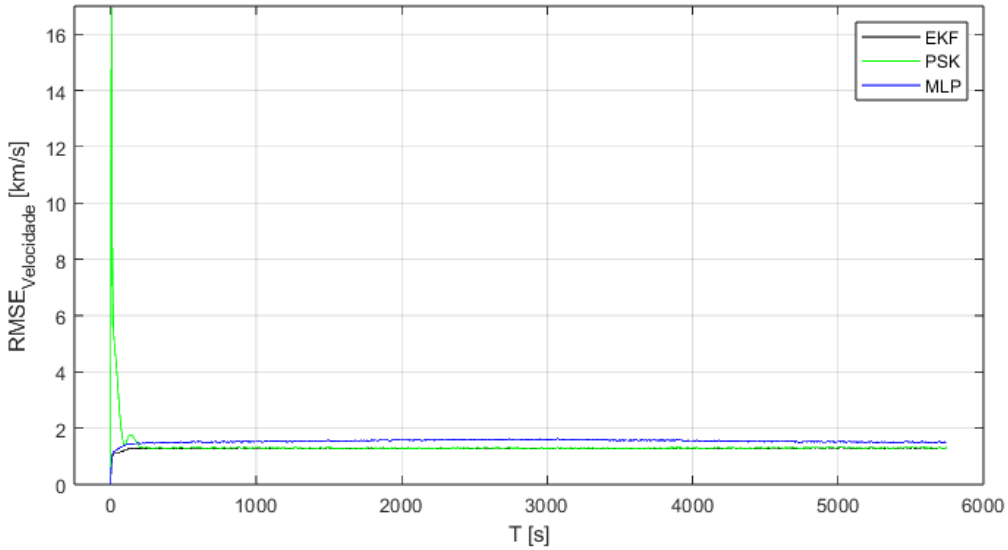

Figura 4.13: Comparação do RMSE obtido EKF, PSELIKA e PSELIKA – MLP para a velocidade.

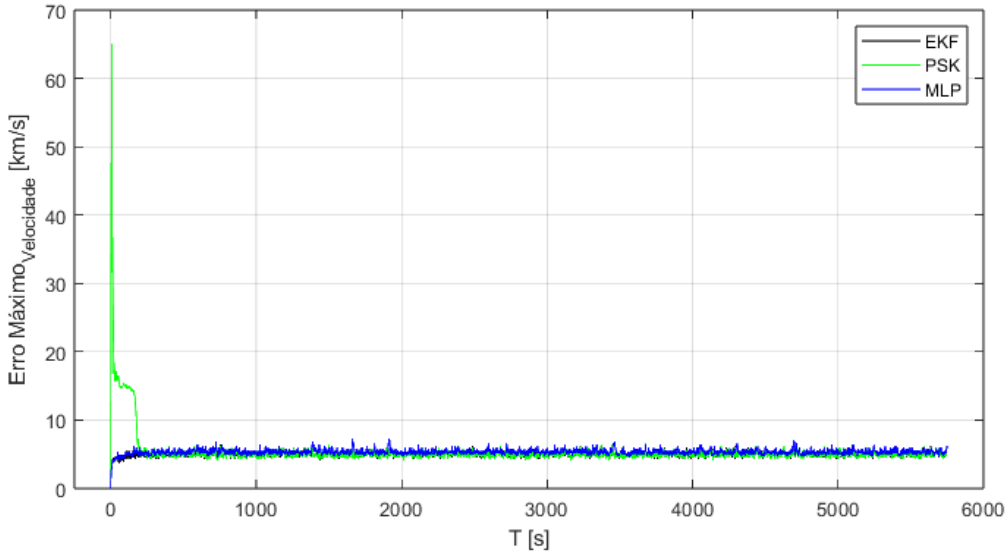

Figura 4.14: Comparação do erro máximo obtido do EKF, PSELIKA e PSELIKA – MLP para a velocidade.

É possível observar na Figura 4.13 e na Figura 4.14, respetivamente, o RMSE da velocidade e seu valor do erro máximo em função do instante t_k em segundos. É possível perceber que o RMSE apresenta valor inicial nulo e que aumenta e atinge um intervalo estável para os filtros PSELIKA e EKF após aproximadamente 800 segundos e para o PSELIKA-MLP para aproximadamente 4500 segundos, e o mesmo ocorre para o erro máximo.

Como é possível perceber de ambos gráficos o desempenho do filtro PSELIKA é bastante reduzido no seu início de funcionamento pois atinge um pico de RMSE de

aproximadamente 17 km/s e um erro máximo de aproximadamente 65 km/s. Isso deve-se pelos mesmos motivos da posição, com as mesmas observações.

Ainda assim, apesar de apresentar o valor elevado de RMSE e erro máximo, após a estabilização o PSELIKA apresenta o segundo melhor desempenho dos demais filtros com um valor médio de RMSE de aproximadamente 1.297 km/s. O melhor desempenho é do filtro EKF com aproximadamente 1.289 km/s e por fim o PSELIKA-MLP com 1.511 km/s. É importante notar que o RMSE para o PSELIKA-MLP apresenta um comportamento quase estável aos 800 segundos, mas continua aumentando gradualmente até os 3000 segundos e depois retorna aos 4500 segundos para uma fase mais estável.

Em relação aos valores de erro máximo no período estável os três filtros apresentam valores relativamente próximos, mas o PSELIKA apresenta o melhor desempenho dos demais filtros com um valor médio de erro máximo de aproximadamente 4.997 km/s, seguido do EKF com aproximadamente 5.005 km/s e por fim o PSELIKA-MLP com 5.361 km/s.

Em relação ao desempenho do filtro para este radar, é de se notar que os valores de erro apresentados em ambos os gráficos são bastante elevados se comparados com radares reais. Entretanto, neste caso trata-se de um radar hipotético em que o valor do desvio padrão da distância medida é muito superior aos existentes pois apresenta-se em quilômetros, servindo apenas como um exemplo do funcionamento e comparação entre filtros. Como termos de comparação o RMSE médio para a medição da posição é de aproximadamente 550 km enquanto o erro máximo médio é de aproximadamente 1250 km, o que mostra que os filtros são realmente capazes de reduzir os erros de medição de maneira eficaz.

4.2.3.2 Complexidade Temporal

Em relação a complexidade temporal dos filtros a comparação foi feita da mesma forma que o caso anterior com base no tempo médio de execução dos algoritmos após 1000 simulações de Monte Carlo. A comparação também será feita em relação ao algoritmo PSELIKA-MLP com o ganho ou perda relativa de tempo em relação aos outros filtros.

Os resultados obtidos foram um ganho de aproximadamente 23.20% de tempo em comparação ao algoritmo PSELIKA e um ganho de aproximadamente 46.34% de tempo em comparação ao algoritmo EKF. O algoritmo PSELIKA-MLP foi mais rápido em relação aos outros algoritmos, o que já era esperado, visto que a rede neuronal utilizada é bastante simples. Os desvios padrões associados são de aproximadamente 1.045% e 0.4471% respetivamente.

As razões para que os outros algoritmos apresentem tempo de processamento superior são os mesmos que para o caso anterior, visto que os algoritmos não se alteraram. Entretanto,

neste caso o algoritmo EKF pode-se aproximar em tempo computacional se comparado ao PSELIKA visto que a dimensão do problema foi reduzida de sete estados para o que reduz o número de cálculos. O mesmo aconteceu para a rede neuronal que apresenta 100 neurónios em comparação aos 200 neurónios do primeiro caso.

4.2.3.3 Complexidade Espacial

O método de comparação espacial será o mesmo apresentado para o caso anterior.

O algoritmo PSELIKA-MLP foi o que apresentou maior uso de espaço de memória com um total de 20336 bytes utilizados sendo destes 19056 bytes utilizados apenas para a execução da rede neuronal implementada, que inclui as matrizes dos parâmetros da rede neuronal e os fatores utilizados para a padronização do estado, correspondendo a aproximadamente 93.71% do espaço total necessário. Em segundo lugar o algoritmo EKF necessitou de 1904 bytes seguido do PSELIKA com 1808 bytes, ocupando 90.01% e 90.51% a menos de espaço de memória respetivamente.

4.3 Comentários

Os resultados obtidos para os dois casos demonstram a capacidade do filtro PSELIKA-MLP de reduzir consideravelmente os custos computacionais mantendo uma exatidão próxima dos algoritmos originais. Entretanto, em termos de complexidade espacial, mantém-se muito acima dos outros dois.

Em relação ao filtro UKF, não foi feita uma comparação com os outros filtros visto que a exatidão e a complexidade temporal são bastante distintas. Como mencionado no capítulo 2 a exatidão do UKF é muito superior aos outros dois métodos e tem como contraparte o aumento do tempo computacional devido às diversas etapas de cálculo que são efetuadas, sendo proporcionais à dimensão do estado.

Capítulo 5 – Conclusões e Trabalhos Futuros

5.1 Conclusões

Determinar o estado atual de um sistema não-linear em constante mudança com base em dados imperfeitos é um desafio fundamental em diversos campos, como engenharia de controle e processamento de sinais, sendo este processo chamado de filtragem ou estimação.

No âmbito aeroespacial apresenta importância eminente, uma vez que a aplicação prática dos métodos de estimação requer exatidão e rapidez com o menor custo possível em termos computacionais e financeiros. O grande desafio enfrentado é justamente a capacidade de estimar o estado de sistemas de elevada complexidade oriundo das não-linearidades que o descrevem ou que não foram capazes de ser modeladas, além das perturbações presentes nos dados adquiridos por conta dos instrumentos utilizados.

Tendo em vista essas necessidades, as redes neuronais têm muito potencial no contexto de filtragem, visto que são capazes de armazenar e retornar conhecimento experimental, permitindo calcular um valor de saída com base em um valor de entrada, aproximando um conjunto de dados a uma função não-linear. Se forem corretamente treinadas, estas podem servir como ferramentas essenciais que integrariam e tornariam os processos de filtragem ainda mais precisos e rápidos, podendo ser utilizadas em conjunto a diferentes métodos.

Neste contexto, diversos métodos de filtragem foram desenvolvidos nas últimas décadas, tentando solucionar este problema, sendo o filtro de Kalman e seus derivados as ferramentas mais utilizadas. O filtro de Kalman estendido (EKF) é o mais conhecido e utilizado, e é baseado no pressuposto de que a realização de uma linearização em torno do estado estimado atual é suficiente para lidar com as não-linearidades do sistema. Apesar de permitir uma solução simples para diversas aplicações, apresenta algumas limitações que incluem principalmente o seu comportamento errático e divergente quando o estado inicial é muito distinto do correto ou para sistemas muito não-lineares.

Outras soluções foram implementadas tentando suprir estas deficiências como, por exemplo, a implementação de métodos de filtragem adaptativa, construção de um modelo estatístico que se adeque ao sistema referido através de uma seleção cuidada de um conjunto de pontos durante a filtragem de Kalman *Unscented*, ou até mesmo a transformação do modelo não-linear para uma estrutura pseudo-linear com coeficientes dependentes do

estado, ou pseudo-linear, dando origem à filtragem de Kalman Pseudo-linear (PSELIKA - *Pseudo-Linear Kalman Filter*).

A vantagem do método PSELIKA é justamente o facto de apresentar uma estrutura pseudo-linear onde é possível aplicar métodos da filtragem Kalman linear, além de capturar bem as não-linearidades do sistema. A sua grande limitação é o facto de necessitar realizar inversões matriciais durante o seu algoritmo o que pode levar a processos de elevada complexidade computacional para sistemas com muitas dimensões.

Assim, aproveitando a sua vantagem de apresentar uma estrutura pseudo-linear, esta dissertação propõe a utilização de uma rede neuronal do tipo *Perceptron* de Múltiplas Camadas (MLP - *Multilayer Perceptron*) com o objetivo de estimar a matriz de ganho para um certo estado sem a necessidade de estimar a matriz de covariância associada, tendo como principal objetivo a redução do custo computacional original do algoritmo PSELIKA.

Primeiramente, foi feita a análise de alguns dos principais métodos de filtragem não-linear nomeadamente o EKF, UKF e PSELIKA, com as suas principais vantagens e limitações, identificando a principal razão do algoritmo PSELIKA ser utilizado como base. Segundamente, foi examinado o funcionamento de algumas arquiteturas de rede neuronal e suas as principais aplicações, assim como suas vantagens e limitações associadas que fizeram a escolha da rede do tipo MLP ser clara para esta aplicação. A seguir, foi feita uma breve análise dos métodos já desenvolvidos de integração entre NNs e métodos de filtragem, seguida da descrição das técnicas e algoritmo implementado para integração de uma rede neuronal MLP ao algoritmo de filtragem PSELIKA, que foi denominado PSELIKA-MLP.

O método PSELIKA necessita da estimação da matriz de covariância para que a matriz de ganho seja calculada na etapa de correção do filtro, e isso pode ser feito a partir da solução da equação de Riccati associada ao modelo. Visto que a equação depende apenas das matrizes que descrevem os modelos do sistema e de medição, ao assumir que as matrizes de covariância do erro de processamento e de medição são constantes é possível estimar a matriz de covariância do erro de estimação apenas a partir do valor do estado do passo anterior. Assim, permite que o cálculo da subsequente matriz de ganho associada seja feito a priori e possa ser estimada a partir de uma rede neuronal que descreva uma função não-linear recebendo como argumento apenas o vetor de estado.

Para que esse algoritmo fosse possível de ser implementado foram apresentadas algumas técnicas que incluem:

-Método de factorização de uma função não-linear para uma estrutura pseudo-linear;

-Definição do domínio do sistema para o treinamento de rede;

-Estrutura da rede neuronal aplicada a etapa de correção da filtragem;

-Treinamento e validação da rede neuronal.

Por fim, a utilização do filtro PSELIKA-MLP foi validada nos seguintes casos de estudo:

-Caso 1: Estimação da atitude e taxa angular de um veículo aeroespacial;

-Caso 2: Estimação da órbita de um satélite artificial.

O objetivo no primeiro caso era estimar a atitude e a taxa angular de um veículo aeroespacial com base em medições ruidosas do vetor de quaternião que descreve a atitude do satélite segundo um referencial fixo, para um valor de estado inicial desconhecido. O objetivo no segundo caso era estimar os valores de posição e velocidade de um satélite artificial em órbita a partir de medições extremamente ruidosas de um radar hipotético.

A validação foi feita a partir da comparação do algoritmo PSELIKA-MLP com os algoritmos PSELIKA e EKF. As comparações serão feitas em termos de exatidão na estimação, além da complexidade temporal e da complexidade espacial.

Para que houvesse relevância estatística dos resultados foram efetuadas dez mil simulações de Monte-Carlo onde o termo aleatório é o ruído branco adicionado às medições, que segue uma distribuição gaussiana e viés nulo.

Para a comparação da exatidão utilizou-se dois métodos, o RMSE (*Root Mean Squared Error*) e a norma infinita ou erro máximo de cada amostra. Em relação à complexidade computacional, mediu-se o tempo médio de execução dos algoritmos para mil simulações de Monte-Carlo. E em relação à complexidade computacional, foi medido o espaço ocupado pelas variáveis guardadas e utilizadas para todos os cálculos intermédios de cada algoritmo.

Para o primeiro caso, os três algoritmos apresentaram valores semelhantes de estabilização apresentando praticamente o mesmo desempenho em termos de exatidão. Entretanto, o PSELIKA converge para o valor estável mais rapidamente.

Em relação a taxa angular, ocorre estabilização para os três algoritmos dos valores de RMSE e norma infinita para valores aproximadamente 0.009 rad/s e 0.026 rad/s, próximo dos 25 segundos. Em relação aos vetores de quaternião a estabilização ocorre para o RMSE de aproximadamente 0.013 e para o erro máximo de aproximadamente 0.035, próximo dos 25 segundos.

Em relação a complexidade temporal, o algoritmo PSELIKA-MLP foi o que teve o menor tempo computacional sendo 23.50% mais rápido que o PSELIKA e 60.23% mais rápido que o EKF.

Já em relação a complexidade espacial, pelo facto de necessitar armazenar as matrizes da rede neuronal o algoritmo PSELIKA-MLP foi o que apresentou maior uso de espaço de memória com um total de 59880 bytes necessários principalmente por utilizar 200 neurónios em sua camada escondida. Em segundo lugar o algoritmo EKF necessitou de 2472 bytes seguido do PSELIKA com 2360 bytes, ocupando 95.77% e 95.96% a menos de espaço de memória respetivamente.

Para o segundo caso, os três algoritmos apresentaram valores diferentes de estabilização apresentando desempenhos distintos quanto ao comportamento e exatidão. O algoritmo PSELIKA, diferentemente dos outros, apresentou dificuldades em estimar corretamente o valor do estado nos instantes iniciais a partir dos valores de estado e matriz de covariância de erro de estimação iniciais. Contudo, após a convergência e estabilização dos valores de erro foi o algoritmo com os valores de erro menores.

Em relação a posição, ocorre estabilização para o PSELIKA, EKF e PSELIKA-MLP algoritmos em torno dos valores de RMSE de 74 km, 81 km e 100 km respetivamente. Para a norma infinita ocorre estabilização do valor médio de aproximadamente 279 km, 281 km e 299km respetivamente.

Em relação a velocidade, ocorre estabilização para o PSELIKA, EKF e PSELIKA-MLP algoritmos em torno dos valores de RMSE de 1.297 km/s, 1.289 km/s e 1.511 km/s respetivamente. Para a norma infinita ocorre estabilização do valor médio de aproximadamente 4.997 km/s, 5.005 km/s e 5.361 km/s respetivamente.

Em relação a complexidade temporal, o algoritmo PSELIKA-MLP foi o que teve o menor tempo computacional novamente sendo 23.20% mais rápido que o PSELIKA e 46.34% mais rápido que o EKF.

Já em relação a complexidade espacial, pelo facto de necessitar armazenar as matrizes da rede neuronal o algoritmo PSELIKA-MLP foi o que apresentou maior uso de espaço de memória com um total de 19056 bytes necessários principalmente por utilizar 100 neurónios em sua camada escondida. Em segundo lugar, o algoritmo EKF necessitou de 1904 bytes seguido do PSELIKA com 1808 bytes, ocupando 90.01% e 90.51% a menos de espaço de memória respetivamente.

Dos resultados, é possível perceber que o filtro PSELIKA-MLP apresenta resultados próximos em termos de exatidão, com uma redução considerável em termos de custo computacional se comparado com os outros dois métodos. Ainda assim, apresenta uma limitação em termos de espaço de memória necessário, sendo necessário muito mais memória que os demais, justamente pela necessidade de armazenar as matrizes dos parâmetros de rede.

Ainda assim, o PSELIKA-MLP é um método muito experimental em que a alteração de diversos parâmetros na etapa de treinamento da rede pode afetar drasticamente os resultados em termos de exatidão, complexidade temporal e espacial.

Na etapa de treinamento, a definição do domínio de treinamento e método de seleção do conjunto de treinamento é de extrema importância, uma vez que, por exemplo, um domínio muito grande com poucos pontos pode resultar em menor exatidão, mas uma maior robustez do filtro e vice-versa, mas seu impacto não foi avaliado neste trabalho.

Da mesma forma, selecionar um número muito baixo de neurónios por camada pode reduzir a exatidão do filtro, enquanto o caso contrário pode levar a treinamentos e aplicação da rede neuronal com elevados custos computacionais.

Além disso, o método de factorização selecionado também deve ser analisado de forma a otimizar a exatidão do algoritmo para cada caso, uma vez que há uma infinidade de parametrizações que podem ser implementadas.

Se métodos efetivos de otimização dos demais parâmetros e hiper-parâmetros de rede forem implementados de forma a garantir robustez e desempenhos aceitáveis consistentemente, o algoritmo PSELIKA-MLP pode se tornar um bom método alternativo ao PSELIKA e EKF, principalmente em aplicações onde o custo computacional deve ser reduzido ao máximo.

5.2 Trabalhos Futuros

Uma parte muito interessante da ciência é que a descoberta ou desenvolvimento de um campo de pesquisa pode abrir portas para a evolução de outro campo. O estudo e aperfeiçoamento das redes neuronais ao longo dos anos permitiu que diversas tarefas que eram antes muito difíceis ou custosas computacionalmente fossem implementadas de forma simples.

Esta dissertação tem como objetivo introduzir uma rede neuronal a filtragem de Kalman pseudo-linear para aplicações aeroespaciais, tentando superar a sua principal limitação, com o algoritmo PSELIKA-MLP. Os resultados são promissores, mas a sua implementação necessita de uma abordagem mais pragmática e otimizada, com uma análise mais aprofundada de como projetar a rede neuronal e o sistema pseudo-linear de forma a garantir robustez e exatidão do filtro, além de manter o custo computacional baixo que é o principal objetivo.

Além disso, os casos de estudo são apenas uma ínfima parte de tudo aquilo que pode aplicado a partir deste algoritmo, sendo necessário, a seguir, expandir esta análise para

outros casos que podem ou não estar incluídos na área aeroespacial e utilizar este algoritmo para dados reais.

Por fim, este algoritmo assume condições estatísticas constantes, então, após a aplicação das considerações anteriores, seria interessante experimentar uma abordagem adaptativa onde as matrizes de covariância do erro de processamento e medição se alteram ao longo do funcionamento do filtro.

Referências

[1] A. Gelb, J. Joseph F. Kasper, J. Raymond A. Nash, C. F. Price e J. Arthur A. Sutherland, "Optimal Linear Filtering," em *Applied Optimal Estimation*, Cambridge, Massachussetts, The MIT Press, 1974, pp. 102-155.

[2] S. J. Julier e J. K. Uhlmann, "Unscented filtering and nonlinear estimation," *Proceedings of the IEEE,* vol. 92, nº 3, pp. 401-422, 2004.

[3] W. Leung e C. Damaren, "A Comparison of the Pseudo-Linear and Extended Kalman Filters for Spacecraft Attitude Estimation," em *AIAA Guidance, Navigation, and Control Conference and Exhibit*, Providence, Rhode Island, 2004.

[4] R. Harman e I. Bar-Itzhack, "The use of pseudo-linear and SDARE filtering for satellite angular-rate estimation," em *AIAA Guidance, Navigation, and Control Conference and Exhibit*, Boston,MA,U.S.A., 1998.

[5] T. Çimen, "State-Dependent Riccati Equation (SDRE) Control: A Survey," *IFAC Proceedings Volumes,* vol. 41, nº 2, pp. 3761-3775, 2008.

[6] O. Bonde e L. Karlsson, "A Comparison of Selected Optimization Methods for Neural Networks," KTH, School of Engineering Sciences (SCI), Stockholm, Sweden, 2020.

[7] H. W. Sorenson, "Least-Squares Estimation: From Gauss to Kalman," *IEEE Spectrum,* vol. 7, nº 7, pp. 63-68, 1970.

[8] R. A. Fisher, "On an absolute criterion for fitting frequency curves," *Messenger Math,* vol. 41, p. 155–160, 1912.

[9] A. N. Shiryayev, "Interpolation and Extrapolation of Stationary Random Sequences," em *Selected Works of A. N. Kolmogorov: Volume II Probability Theory and Mathematical Statistics*, Dordrecht, Springer Netherlands, 1992, pp. 272-280.

[10] N. Wiener, Extrapolation, Interpolation, and Smoothing of Stationary Time Series, New York: Wiley, 1949.

[11] R. E. Kalman, "A New Approach to Linear Filtering and Prediction Problems," *Journal of Basic Engineering (ASME),* vol. 82D, pp. 35-45, 1960.

[12] R. E. Kalman e R. S. Bucy, "New Results in Linear Filtering and Prediction Theory," *J. Basic Eng.,* vol. 83, nº 1, pp. 95-108, 1961.

[13] L. A. McGee e S. F. Schmidt, "Discovery of the Kalman filter as a practical tool for aerospace and industry," NASA Technical report, NASA-TM-86847, 1985.

[14] H. W. Sorenson, Kalman filtering : theory and application, New York: IEEE Press, 1985.

[15] M. S. Grewal e A. P. Andrews, "Applications of Kalman Filtering in Aerospace: 1960 to the Present," *IEEE Control Systems Magazine,* vol. 30, nº 3, pp. 69-78, 2010.

[16] M. Coelho, K. Bousson e K. Ahmed, "Survey of Nonlinear State Estimation in Aerospace Systems with Gaussian Priors," *Advances in Aircraft and Spacecraft Science,* vol. 7, nº 6, p. 2020, 495-516.

[17] S. Julier, J. Uhlmann e H. F. Durrant-Whyte, "A new method for the nonlinear transformation of means and covariances in filters and estimators," *IEEE Transactions on Automatic Control,* vol. 45, nº 3, pp. 477-482, 2000.

[18] S.-W. Kim, M. Abdelrahman, S.-Y. Park e K.-H. Choi, "Unscented Kalman Filtering for Spacecraft Attitude and Rate Determination Using Magnetometer," *Space Sci.,* vol. 26, pp. 31-46, 2009.

[19] R. R. Harman e I. Y. Bar-Itzhack, "Pseudolinear and State-Dependent Riccati Equation Filters for Angular Rate Estimation," *Journal of Guidance,* vol. 22, nº 5: Engineering Notes, pp. 723-725, 1999.

[20] I. Bar-Itzhack, R. Harman e D. Choukroun, "State-Dependent Pseudo-Linear Filter for Spacecraft Attitude and Rate Estimation," em *AIAA Guidance, Navigation, and Control Conference and Exhibit*, Monterey, California, 2001.

[21] I. Arasaratnam, “Cubature Kalman Filtering: Theory & Applications,” Ph.D., Dep. Elect. Comput. Eng., McMaster Univ., Ontario, Canada, 2009.

[22] I. Arasaratnam, S. Haykin e T. R. Hurd, “Cubature Kalman Filtering for Continuous-Discrete Systems: Theory and Simulations,” *IEEE Transactions on Signal Processing,* vol. 58, nº 10, pp. 4977-4993, 2010.

[23] I. Arasaratnam e S. Haykin, “Cubature Kalman Filtering: A Powerful New Tool For Aerospace Applications,” em *Procedures of the International Radar Conference*, Guilin, China, 2009.

[24] G. Revach, N. Shlezinger, X. Ni, A. L. Escoriza, R. J. G. v. Sloun e Y. C. Eldar, “KalmanNet: Neural Network Aided Kalman Filtering for Partially Known Dynamics,” *IEEE Transactions on Signal Processing,* vol. 70, pp. 1532-1547, 2022.

[25] R. E. Kalman, “New Methods in Wiener Filtering,” em *Proc. of the First Symposium on Engineering Applications of Random Function Theory and Probability*, New York, John Wiley and Sons, Inc, 1963.

[26] G. L. Smith, “The scientific inferential relationships between statistical estimation, decision theory, and modern filter theory,” *Proc. JACC,* pp. 350-359, 1965.

[27] J. H. J. Laning e R. H. Battin, em *Random Processes in Automatic Control*, New York, McGraw-Hill Book Company, Inc., 1956, pp. 269-275.

[28] D. Simon, Optimal State Estimation: Kalman, H infinity, and Nonlinear Approaches, Hoboken, New Jersey: John Wiley & Sons Inc., 2006.

[29] M. Athans e F. C. Schweppe, “Gradient Matrix and Matrix Calculations,” Technical Note, Lincoln Laboratory, MIT, Lexington, Massachussetts, 1965.

[30] A. H. Jazwinski, Stochastic Processes and Filtering Theory, Cambridge: Academic Press, 1970.

[31] R. P. Wishner, J. A. Tabaczynski e M. Athans, “A comparison of three non-linear filters,” *Automatica,* vol. 5, pp. 487-496, 1969.

[32] H. J. Kushner, “Dynamical equations for optimum nonlinear filtering,” *Journal of Differential Equations,* vol. 3, p. 179–190, 1967.

[33] J. K. Uhlmann, “Simultaneous map building and localization for real time applications,” Technical report, Transfer thesis, University of Oxford, Oxford, UK, 1994.

[34] R. Azor, I. Y. Bar-Itzhack, J. K. Deutschmann e R. R. Harman, “Angular-Rate Estimation Using Quaternion Measurements,” NASA Technical Report 19990014340, Washington, D.C., 1998.

[35] B. Friedland, Advanced Control System Design, Englewood Cliffs: Prentice-Hall, 1996.

[36] A. Wernli e G. Cook, “Suboptimal control for the nonlinear quadratic regulator problem,” *Automatica,* vol. 11, pp. 75-84, 1975.

[37] J. Cloutier, C. D’Souza e C. Mracek, “Nonlinear regulation and nonlinear H∞ control via the state-dependent Riccati equation technique: Part 1, Theory; Part 2, Examples,” em *Proc. of the First International Conference on Nonlinear Problems in Aviation and Aerospace*, Daytona Beach, pp. 117-141, 1996.

[38] C. P. Mracek e J. R. Cloutier, “Control designs for the nonlinear benchmark problem via the state-dependent Riccati equation method,” *International Journal of Robust and Nonlinear Control,* vol. 8, pp. 401-433, 1998.

[39] A. A. Stoorvogel e A. Saberi, “The discrete algebraic Riccati equation and linear matrix inequality,” *Linear Algebra and its Applications,* vol. 274, nº 1, pp. 317-365, 1998.

[40] D.-H. Lee e J. Hu, “A Study of the Duality between Kalman Filters and LQR Problems,” Purdue University, Department of Electrical and Computer Engineering Technical Reports, Paper 476, West Lafayette, Indiana, 2016.

[41] N. Metropolis e S. Ulam, “The Monte Carlo Method,” *Journal of the American Statistical Association,* vol. 44, nº 247, pp. 335-341, 1949.

[42] I. M. Sobol, The Monte Carlo Method (Tradução e Adaptação), R. Messer, J. Slone e P. Fortini, Edits., Little Mathematics Library: Mir Publishers, 1974.

[43] J. Moshman, "Random number generation," *Mathematical methods for digital computers,* vol. 2, pp. 249-263, 1967.

[44] S. Akhlaghi, N. Zhou e Z. Huang, "Adaptive adjustment of noise covariance in Kalman filter for dynamic state estimation," em *IEEE Power & Energy Society General Meeting*, Chicago, IL, 2017.

[45] S. Olumide, "Root Mean Square Error (RMSE) In AI: What You Need To Know," 08 Agosto 2023. [Online]. Available: https://arize.com/blog-course/root-mean-square-error-rmse-what-you-need-to-know/. [Acedido em 02 Maio 2024].

[46] S. S. Haykin, Neural Networks and Learning Machines, 3ª ed., Hamilton, Ontario: Pearson Prentice Hall, 2009.

[47] H. Gray, "Anatomy of the Human Body," 1918. [Online]. Available: https://www.bartleby.com/lit-hub/anatomy-of-the-human-body/ix-neurology/. [Acedido em 20 Maio 2024].

[48] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain," *Psychological Review,* vol. 65, nº 6, pp. 386-408, 1958.

[49] N. Rashevsky, Mathematical Biophysics: Physicomathematical Foundations of Biology, Chicago: University of Chicago Press, 1938.

[50] W. S. McCulloch e W. A. Pitts, "Logical calculus of the ideas immanent in nervous activity," *Butt. math. Biophysics,* vol. S, pp. 115-133, 1943.

[51] J. T. Culbertson, Consciousness and behavior: A neural analysis of behavior and of consciousness, Dubuque, Iowa: Wm. C. Brown Company, 1950.

[52] S. C. Kleene, "Representation of events in nerve nets and finite automata," *Annals of Mathematics Studies 34,* pp. 3-41, 1956.

[53] M. L. Minsky, “Some universal elements for finite automata,” *Automata studies,* pp. 117-128, 1956.

[54] D. O. Hebb, The Organization of Behavior: A neuropsychological theory, New York: John Wiley & Sons Inc., 1949.

[55] F. A. Hayek, The Sensory Order, Chicago: University of Chicago Press, 1952.

[56] W. R. Ashby, Design for a brain, New York: John Wiley, 1952.

[57] F. Rosenblatt, Principles of Neurodynamics, New York: Spartan Books, 1962, pp. 595-596.

[58] D. Crevier, AI: The Tumultuous History of the Search for Artificial Intelligence, New York City: BasicBooks, 1993.

[59] B. Widrow e M. E. Hoff, “Adaptive Switching Circuits,” *1960 IRE WESCON Convention Record,* vol. 4, pp. 96-104, 1960.

[60] K. Steinbuch, “Die Lernmatrix,” *Kybernetik,* vol. 1, p. 36–45, 1961.

[61] S. Papert e M. Minsky, Perceptrons: An Introduction to Computational Geometry, The MIT Press, 1969.

[62] J. Schmidhuber, “Annotated History of Modern AI and Deep Learning,” Technical Report IDSIA-22-22, ArXiv, 2022.

[63] G. E. Hinton, “How Neural Networks Learn from Experience,” *Scientific American,* pp. 145-151, 1992.

[64] A. G. Ivakhnenko e V. G. Lapa, Cybernetic Predicting Devices, Purdue Univ. Lafayette ind School of Electrical Engineering: Joint Publications Research Service, 1965.

[65] S. Amari, “A Theory of Adaptive Pattern Classifiers,” *IEEE Transactions on Electronic Computers,* vol. EC16, nº 3, pp. 299-307, 1967.

[66] H. Robbins e S. Monro, “A Stochastic Approximation Method,” *The Annals of Mathematical Statistics,* vol. 22, nº 3, pp. 400-407, 1951.

[67] S. Linnainmaa, “The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors,” Dissertação de Mestrado (em finlandês), Univ. Helsinki, Helsink, Finlândia, 1970.

[68] P. J. Werbos, “Applications of advances in nonlinear sensitivity analysis,” em *Lecture Notes in Control and Information Sciences*, Berlin, Heidelberg, 1982.

[69] J. J. Hopfield, “Neural Networks and Physical Systems with Emergent Collective Computational Abilities,” *Proceedings of the National Academy of Sciences,* vol. 79, nº 8, pp. 2554-2558, 1982.

[70] D. E. Rumelhart, J. L. McClelland e PDP Research Group, Parallel Distributed Processing: Explorations into the Microstructure of Cognition, vol. 1 e 2, Cambridge, Massachussetts: The MIT Press, 1986.

[71] T. J. Sejnowski e C. R. Rosenberg, “Parallel Networks That Learn to Pronounce English Text,” *Complex Systems,* vol. 1, pp. 145-168, 1987.

[72] D. Pomerleau, “ALVINN: An Autonomous Land Vehicle in a Neural Network,” *Advances in Neural Information Processing Systems I,* pp. 305-313, 1989.

[73] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser e I. Polosukhin, “Attention is all you need,” *NIPS 2017,* pp. 5998-6008, 2017.

[74] A. Katharopoulos, A. Vyas, N. Pappas e F. Fleuret, “Transformers are {RNN}s: Fast Autoregressive Transformers with Linear Attention,” *Proceedings of the 37th International Conference on Machine Learning,* vol. 119, pp. 5156-5165, 2020.

[75] K. Choromanski, V. Likhosherstov, D. Dohan, X. Song, A. Gane, T. Sarlos, P. Hawkins, J. Davis, A. Mohiuddin, L. Kaiser, D. Belanger, L. Colwell e A. Weller, “Rethinking Attention with Performers,” *arXiv,* pp. 1-38, 2022.

[76] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G.

Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever e D. Amodei, "Language Models are Few-Shot Learners," *arXiv,* pp. 1-75, 2020.

[77] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks,* vol. 61, pp. 85-117, 2015.

[78] J. Schmidhuber, "Learning Complex, Extended Sequences Using the Principle of History Compression," *Neural Computation,* vol. 4, pp. 234-242, 1992.

[79] J. Hochreiter, "Untersuchungen zu dynamischen neuronalen Netzen," Tese de Diploma em Informática, Institut für Informatik Technische Universität München Arcisstr, Munique, 1991.

[80] Y. Bengio, P. Simard e P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Transactions on Neural Networks,* vol. 5, nº 2, pp. 157-166, 1994.

[81] S. Hochreiter e J. Schmidhuber, "Long Short-term Memory," *Neural computation,* vol. 9, nº 8, pp. 1735-1780, 1997.

[82] C. Metz, "An Infusion of AI Makes Google Translate More Powerful Than Ever," 27 Setembro 2016. [Online]. Available: https://www.wired.com/2016/09/google-claims-ai-breakthrough-machine-translation/. [Acedido em 16 Maio 2024].

[83] L. P. Kaelbling, M. L. Littman e A. W. Moore, "Reinforcement Learning: A Survey," *Journal of Artificial Intellingence,* vol. 4, pp. 237-285, 1996.

[84] The Alpha Star team, "AlphaStar: Mastering the real-time strategy game StarCraft II," 24 Janeiro 2019. [Online]. Available: https://deepmind.google/discover/blog/alphastar-mastering-the-real-time-strategy-game-starcraft-ii/. [Acedido em 13 Maio 2024].

[85] C. Berner, G. Brockman, B. Chan, V. Cheung, P. Dębiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, R. Józefowicz, S. Gray, C. Olsson, J. Pachocki, M. Petrov, H. P. d. Pinto, J. Raiman, T. Salimans, J. Schlatter, J. Schneider, S. Sidor, I.

Sutskever, J. Tang, F. Wolski e S. Zhang, “Dota 2 with Large Scale Deep Reinforcement Learning,” *arXiv,* pp. 1-66, 2019.

[86] K. Weise, C. Metz, N. Grant e M. Isaac, “Inside the A.I. Arms Race That Changed Silicon Valley Forever,” 5 Dezembro 2023. [Online]. Available: https://www.nytimes.com/2023/12/05/technology/ai-chatgpt-google-meta.html. [Acedido em 13 Maio 2024].

[87] OpenAI Developers, “Introduction,” 2024. [Online]. Available: https://platform.openai.com/docs/introduction. [Acedido em 10 Maio 2024].

[88] A.-N. Sharkawy, “Principle of Neural Network and Its Main Types: Review,” *Journal of Advances in Applied & Computational Mathematics,* pp. 8-19, 2020.

[89] C. Fung, V. Iyer, W. Brown e K. Wong, “Comparing the Performance of Different Neural Networks Architectures for the Prediction of Mineral Prospectivity,” *2005 International Conference on Machine Learning and Cybernetics, ICMLC 2005, Guangzhou, China,* pp. 394-398, Setembro 2005.

[90] Y.-M. Chiang, L.-C. Chang e F.-J. Chang, “Comparison of static-feedforward and dynamic-feedback neural networks for rainfall–runoff modeling,” *Journal of Hydrology,* vol. 290, nº 3-4, pp. 297-311, 2004.

[91] S.-C. Chen, S.-W. Lin, T.-Y. Tseng e H.-C. Lin, “Optimization of Back-Propagation Network Using Simulated Annealing Approach,” *2006 IEEE International Conference on Systems, Man and Cybernetics, Taipei, Taiwan,* vol. 4, pp. 2819-2824, 2006.

[92] M. A. Sassi, M. J.-D. Otis e A. Campeau-Lecours, “Active stability observer using artificial neural network for intuitive physical human–robot interaction,” *International Journal of Advanced Robotic Systems,* vol. 14, nº 4, pp. 1-16, 2017.

[93] E. De Momi, L. Kranendonk, M. Valenti, N. Enayati e G. Ferrigno, “A Neural Network-Based Approach for Trajectory Planning in Robot–Human Handover Tasks,” *Frontiers in Robotics and AI,* vol. 3, pp. 1-10, 2016.

[94] T. Xie, H. Yu e B. Wilamowski, "Comparison between traditional neural networks and radial basis function networks," *2011 IEEE International Symposium on Industrial Electronics,Gdansk, Poland,* pp. 1194-1199, 2011.

[95] P. Jeatrakul e K. W. Wong, "Comparing the Performance of Different Neural Networks for Binary Classification Problems," *2009 Eighth International Symposium on Natural Language Processing, Bangkok, Thailand,* pp. 111-115, 2009.

[96] D. Anderson e G. McNeill, "Artificial Neural Networks Technology: A DACS State-of-the-Art Report," Kaman Sciences Corporation, Utica, New York, 1992.

[97] K.-L. Du e M. N. S. Swamy, Neural Networks and Statistical Learning, 1ª ed., London: Springer, 2013.

[98] X. Zhao, S. Chumkamon, S. Duan, J. Rojas e J. Pan, "Collaborative Human-Robot Motion Generation using LSTM-RNN," *2018 IEEE-RAS 18th International Conference on Humanoid Robots (Humanoids), Beijing, China,* pp. 441-446, 2018.

[99] C. Torkar, S. Yahyanejad, H. Pichler, M. Hofbaur e B. Rinner, "RNN-based Human Pose Prediction for Human-Robot Interaction," *Proceedings of the ARW & OAGM Workshop 2019,* pp. 76-80, 2019.

[100] T. Yamada, S. Murata, H. Arie e T. Ogata, "Dynamical Integration of Language and Behavior in a Recurrent Neural Network for Human–Robot Interaction," *Frontiers in Neurorobotics,* vol. 10, nº 5, pp. 1-17, 2016.

[101] P. Schydlo, M. Rakovic, L. Jamone e J. Santos-Victor, "Anticipation in Human-Robot Cooperation: A Recurrent Neural Network Approach for Multiple Action Sequences Prediction," *2018 IEEE International Conference on Robotics and Automation (ICRA), Brisbane, QLD, Australia,* pp. 5909-5914, 2018.

[102] H. T. Siegelmann e E. D. Sontag, "Turing computability with neural nets," *Applied Mathematics Letters,* vol. 4, nº 6, pp. 77-80, 1991.

[103] M. Mohri, A. Rostamizadeh e A. Talwalkar, Foundations of Machine Learning, MIT Press, 2012.

[104] scikit-learn Developers, “An introduction to machine learning with scikit-learn,” 2024. [Online]. Available: https://scikit-learn.org/stable/tutorial/basic/tutorial.html. [Acedido em 5 Junho 2024].

[105] Alex, “Feedforward Neural Networks and Multilayer Perceptrons,” 08 Abril 2020. [Online]. Available: https://boostedml.com/2020/04/feed- forward-neural-networks-and-multilayer-perceptrons.html.. [Acedido em 26 Abril 2024].

[106] M.-C. Popescu, V. Balas, L. Perescu-Popescu e N. Mastorakis, “Multilayer Perceptron and Neural Networks,” *WSEAS Transactions on Circuits and Systems,* vol. 8, pp. 579-588, Julho 2009.

[107] T. J. Sejnowski, The Deep Learning Revolution, Cambridge, Massachusetts: The MIT Press, 2018, pp. 47-48.

[108] M. Rosen-Zvi, M. Biehl e I. Kanter, “Learnability of periodic activation functions: General results,” *Phys. Rev. E,* vol. 58, nº 3, pp. 3606-3609, Setembro 1998.

[109] I. Isa, S. Omar, Z. Saad e M. Osman, “Performance Comparison of Different Multilayer Perceptron Network Activation Functions in Automated Weather Classification,” *Asia International Conference on Modelling & Simulation, Kota Kinabalu, Malaysia,* vol. 0, pp. 71-75, Maio 2010.

[110] S. Sharma, S. Sharma e A. Athaiya, “ACTIVATION FUNCTIONS IN NEURAL NETWORKS,” *International Journal of Engineering Applied Sciences and Technology,* vol. 04, pp. 310-316, 05 2020.

[111] Universidade Federal do Rio Grande do Sul, “A função de Heaviside,” 26 Julho 2022. [Online]. Available: https://www.ufrgs.br/reamat/TransformadasIntegrais/livro-tl/apdtedtd-a_funx00e7x00e3o_de_heaviside.html. [Acedido em 07 Maio 2024].

[112] Omegatron, 9 Abril 2014. [Online]. Available: https://en.wikipedia.org/wiki/Step_function#/media/File:Dirac_distribution_CDF.svg. [Acedido em 09 Maio 2024].

[113] I. Goodfellow, Y. Bengio e A. Courville., Deep Learning, Cambridge, Massachusetts: The MIT Press, 2016.

[114] D. E. Rumelhart, G. E. Hinton e R. J. Williams, "Learning representations by back-propagating errors," *Nature 323,* p. 533–536, 1986.

[115] J. M. Child (Tradutor) e G. W. Leibniz (Autor), The Early Mathematical Manuscripts of Leibniz., Merchant Books, 2007.

[116] G. W. Leibniz, Nova Methodus pro Maximis et Minimis, 1684.

[117] M. Negnevitsky, Artificial Intelligence: A Guide to Intelligent Systems, 2ª ed., Harlow, Essex: Pearson Education Limited, 2002.

[118] scikit-learn Developers, "MLPRegressor," 2024. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html#sklearn.neural_network.MLPRegressor. [Acedido em 26 Abril 2024].

[119] J. Patterson e A. Gibson, Deep Learning: A Practitioner's Approach, Sebastopol: O'Reilly Media, Inc., 2017.

[120] S. Marsland, Machine Learning: An Algorithmic Perspective, 2ª ed., Boca Raton, Fla.: Chapman and Hall/CRC, 2014.

[121] D. P. Kingma e J. L. Ba, "Adam: A Method for Stochastic Optimization," *International Conference for Learning Representations,* vol. abs/1412.6980, 2015.

[122] T. Dozat, "Incorporating Nesterov Momentum into ADAM," *Proceedings of the 4th International Conference on Learning Representations,* pp. 1-4, 2016.

[123] G. Hinton, Neural Networks for Machine Learning, slides of Neural Networks for Machine Learning: Lecture 6e RMSprop: Divide the gradient by a running average of its recent magnitude, Coursera, 2016.

[124] J. Duchi, E. Hazan e Y. Singer, "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization," *Journal of Machine Learning Research,* vol. 12, pp. 2121-2159, 2012.

[125] C. G. Broyden, “The convergence of single-rank quasi-Newton methods,” *Mathematics of Computation,* vol. 24, pp. 365-382, 1970.

[126] R. Fletcher, “A New Approach to Variable Metric Algorithms,” *The Computer Journal,* vol. 13, pp. 317-322, 1970.

[127] D. Goldfarb, “A Family of Variable-Metric Methods Derived by Variational Means,” *Mathematics of Computation,* vol. 24, nº 109, pp. 23-26, 1970.

[128] D. F. Shanno, “Conditioning of Quasi-Newton Methods for Function Minimization,” *Mathematics of Computation,* vol. 24, nº 111, pp. 647-656, 1970.

[129] J. E. Dennis Jr. e J. J. Moré, “Quasi-Newton Methods, Motivation and Theory,” *SIAM Review,* vol. 19, nº 1, pp. 46-89, 1977.

[130] R. Bollapragada, D. Mudigere, J. Nocedal, H.-J. M. Shi e P. T. P. Tang, “A Progressive Batching L-BFGS Method for Machine Learning,” *Proceedings of the 35th International Conference on Machine Learning,* pp. 620-629, 2018.

[131] J. Nocedal, “Updating Quasi-Newton Matrices with Limited Storage,” *Mathematics of Computation,* vol. 35, nº 151, pp. 773-782, 1980.

[132] M. Horton, “Real-time identification of missile aerodynamics using a linearised Kalman filter aided by an artificial neural network,” *IEE Proceedings - Control Theory and Applications,* vol. 144, nº 4, pp. 299-308, 1997.

[133] M. Baptista, E. Henriques, I. P. de Medeiros, J. Malere, C. Nascimento e H. Prendinger, “Remaining useful life estimation in aeronautics: Combining data-driven and Kalman filtering,” *Reliability Engineering & System Safety,* vol. 184, pp. 228-239, 2019.

[134] M. Alberto-Olivares, A. Gonzalez-Gutierrez, S. Tovar-Arriaga e E. Gorrostieta-Hurtado, “Remaining Useful Life Prediction for Turbofan based on a Multilayer Perceptron and Kalman Filter,” *2019 16th International Conference on Electrical Engineering, Computing Science and Automatic Control (CCE), Mexico City, Mexico,* pp. 1-6, 2019.

[135] V. Pesce, S. Silvestrini e M. Lavagna, “Radial basis function neural network aided adaptive extended Kalman filter for spacecraft relative navigation,” *Aerospace Science and Technology,* vol. 96, p. 105527, 2020.

[136] F. P. Härter e H. F. d. C. Velho, “Multilayer Perceptron Neural Network in a Data Assimilation Scenario,” *Engineering Applications of Computational Fluid Mechanics,* vol. 4, nº 2, pp. 237-245, 2010.

[137] P. Becker, H. Pandya, G. Gebhardt, C. Zhao, J. Taylor e G. Neumann, “Recurrent Kalman Networks: Factorized Inference in High-Dimensional Deep Feature Spaces,” *arXiv,* pp. 1-15, 2019.

[138] Y. Dahan, G. Revach, J. Dunik e N. Shlezinger, “Uncertainty Quantification in Deep Learning Based Kalman Filters,” *arXiv,* pp. 1-5, 2023.

[139] J. R. Cloutier e D. T. Stansbery, “The capabilities and art of state-dependent Riccati equation-based design,” *Proceedings of the 2002 American Control Conference (IEEE Cat. No.CH37301), Anchorage, AK, USA,* vol. 1, pp. 86-91, 2002.

[140] D. T. Stansbery e J. R. Cloutier, “Position and attitude control of a spacecraft using the state-dependent Riccati equation technique,” *Proceedings of the 2000 American Control Conference. ACC (IEEE Cat. No.00CH36334), Chicago, IL, USA,* vol. 3, pp. 1867-1871, 2000.

[141] J. R. Cloutier e D. T. Stansbery, “Nonlinear, Hybrid Bank-to-Turn/Skid-to-Turn Missile Autopilot Design,” *AIAA Guidance, Navigation and Control Conference and Exhibit, Montreal, Canada ,* pp. 1-11, 2001.

[142] I. The MathWorks, “dlqr,” 2024. [Online]. Available: https://www.mathworks.com/help/control/ref/dlqr.html. [Acedido em 20 Maio 2024].

[143] R. DeCarlo, Linear Systems: A State Variable Approach with Numerical Implementation, Englewood Cliffs, New Jersey: Prentice Hall, 1989.

[144] T. O. Kvalseth, “Cautionary Note about R2,” *The American Statistician,* vol. 39, nº 4, p. 279–285, 1985.

[145] scikit-learn Developers, “Model selection and evaluation,” 2024. [Online]. Available: https://scikit-learn.org/stable/model_selection.html. [Acedido em 20 Maio 2024].

[146] J. R. Wertz, em *Spacecraft Attitude Dynamics and Control*, Dordrecht, Holland, Reidel Publishing Co., 1978, pp. 512-513.

[147] M. Hamisu, I. Saidu e M. Y. Waziri, “A Simplified Derivation and Analysis of Fourth Order,” *International Journal of Computer Applications,* vol. 9, nº 8, pp. 51-55, 2010.

[148] W. M. Kaula, Theory of Satellite Geodesy, Waltham, MA: Blaisdell Publishing Company, 1966.

[149] J.-U. Park, K.-H. Choi e S. Lee, “Orbital rendezvous using two-step sliding mode control,” *Aerospace Science and Technology,* vol. 3, nº 4, pp. 239-245, 1999.

[150] H. D. Curtis, Orbital Mechanics for Engineering Students, Daytona Beach, Florida: Elsevier The Boulevard, 2005.

Anexos

Anexo A – Algoritmo de *Backpropagation*

O algoritmo de *Backpropagation* encontra-se descrito a seguir. As notações utilizadas são as mesmas presentes no corpo do trabalho principal.

Primeiramente, define-se δ_i^l como o erro no neurónio i da camada l por [114]:

$$\delta_i^l = \frac{\partial L}{\partial \zeta_i^l} \tag{A.1}$$

Para saber quanto esta quantidade está relacionada com o erro, assuma uma mudança no valor de ζ_i^l de $\Delta\zeta_i^l$. O valor da função de custo então mudará numa quantidade de $\frac{\partial L}{\partial \zeta_i^l}\Delta\zeta_i^l = \delta_i^l \Delta\zeta_i^l$ [113]. Se δ_i^l for grande, então o neurónio está longe do seu valor ótimo e a alteração na função de custo será relativamente grande. Caso contrário, se δ_i^l for pequeno então o neurónio está próximo do valor ótimo e a função custo permanecerá praticamente constante. Então, δ_i^l é a medida de quão próximo um neurónio encontra-se de seu valor ótimo.

O erro na camada de saída δ_i^D pode ser facilmente calculado com a regra da cadeia presente em cálculo.

$$\delta_i^D = \frac{\partial L}{\partial \zeta_i^D} = \sum_{k=1}^{N_D} \frac{\partial L}{\partial a_k^D}\frac{\partial a_k^D}{\partial \zeta_i^D} = \sum_{k=1}^{N_D} \frac{\partial L}{\partial a_k^D}\frac{\partial (\zeta_k^D)}{\partial \zeta_i^D} = \frac{\partial L}{\partial a_i^D} \tag{A.2}$$

uma vez que a ativação $a_k^D = \zeta_k^D$ depende somente de ζ_i^D quando i=k. Utilizando (A.2), pode-se então propagar para trás através da rede para calcular outros δ_i^l. Estes podem ser calculados de uma forma similar a da camada de saída [113]:

$$\begin{aligned} \delta_i^l = \frac{\partial L}{\partial \zeta_i^l} &= \sum_{k=1}^{N_{l+1}} \frac{\partial L}{\partial \zeta_k^{l+1}}\frac{\partial \zeta_k^{l+1}}{\partial \zeta_i^l} = \sum_{k=1}^{N_{l+1}} \delta_k^{l+1}\frac{\partial}{\partial \zeta_k^{l+1}}\left(\sum_{j=1}^{N_l} W_{kj}^{l+1}a_k^l + b_k^{l+1}\right) = \\ &= \sum_{k=1}^{N_{l+1}} \delta_k^{l+1} W_{ki}^{l+1}\frac{\partial a_i^l}{\partial \zeta_i^l} = \sum_{k=1}^{N_{l+1}} \delta_k^{l+1} W_{ki}^{l+1} step(\zeta_i^l) \end{aligned} \tag{A.3}$$

A derivada da função de ativação ReLU na camada escondida pode ser descrita por uma função Heaviside com argumento ζ_i^l, que neste caso está representada por $step(\zeta_i^l)$ na equação (A.3).

Então, ao calcular o erro na camada de saída é possível utilizar (A.3) para calcular o erro em todas as outras camadas. Por esta razão, o algoritmo é chamado "*backpropagation*", já que a propagação é feita para trás através da rede. Após isso, é possível relacionar os erros com as derivadas em relação aos pesos e vieses. A derivada em relação aos vieses é dada por:

$$\frac{\partial L}{\partial b_i^l} = \sum_{k=1}^{N_l} \frac{\partial L}{\partial \zeta_k^l} \frac{\partial \zeta_k^l}{\partial b_i^l} = \frac{\partial L}{\partial \zeta_i^l} = \delta_i^l \tag{A.4}$$

Na equação (A.4), $\frac{\partial \zeta_k^l}{\partial b_i^l}$, dentro do somatório é numericamente igual ao Kronecker delta, o qual será representado por δ_{ij}. Isto acontece porque o termo ζ_k^l depende apenas de b_i^l quando $i = k$. O Kronecker delta representa o elemento da linha i e coluna j de uma matriz onde se $i = j$ então $\delta_{ij} = 1$, caso contrário $\delta_{ij} = 0$, o que representa uma matriz identidade. A derivada em relação aos pesos é dada por:

$$\begin{gathered} \frac{\partial L}{\partial W_{ij}} = \sum_{k=1}^{N_l} \frac{\partial L}{\partial \zeta_k^l} \frac{\partial \zeta_k^l}{\partial W_{ij}^l} = \sum_{k=1}^{N_l} \frac{\partial L}{\partial \zeta_k^l} \frac{\partial}{\partial W_{ij}^l} \left(\sum_{m=1}^{N_{l-1}} W_{km}^l a_m^{l-1} + b_k^l \right) \\ \frac{\partial L}{\partial W_{ij}} = \sum_{k=1}^{N_l} \sum_{m=1}^{N_{l-1}} \delta_{ik} \delta_{jm} \delta_k a_m^{l-1} = \delta_i^l a_j^{l-1} \end{gathered} \tag{A.5}$$

Representado os resultados em notação vetorial obtém-se [113]:

$$\begin{cases} \delta^D = \nabla_{a^D} L \\ \delta^l = \left(W^{l+1}\right)^T \delta^{l+1} \odot step(z^l) \end{cases} \tag{A.6}$$

$$\begin{cases} \dfrac{\partial L}{\partial b^l} = \delta^l \\ \dfrac{\partial L}{\partial W^l} = \delta^l \cdot \left(a^{l-1}\right)^T \end{cases} \tag{A.7}$$

Anexo B – Neural Kalman State Estimation with Aerospace Applications *(to be submitted, preliminary version)*

Authors: Felippe Ferreira da Silva and Kouamana Bousson

Abstract

The search for more efficient and lower-cost methods for estimation or filtering, mainly in the aerospace area, has promoted the development of several methods, each with its advantages and limitations, one of the main limitations being computational cost. This work investigates the integration of a Multilayer Perceptron (MLP) neural network with the Pseudo-linear Kalman filter (PSELIKA), named PSELIKA-MLP, with the aim of calculating the gain matrix and reducing the computational cost, while maintaining the filter's accuracy as much as possible. The validation of this method was done through the case study of estimating the angular rate and attitude of an aerospace vehicle based on quaternion measurements. Using numerical simulations, the PSELIKA-MLP algorithm was compared with PSELIKA and the Extended Kalman Filter (EKF), in terms of accuracy, in addition to temporal and spatial complexity. The results show a considerable reduction in the computational time of the proposed method, maintaining similar accuracy to the original algorithm, at the cost of greater storage space required for the process.

Keywords

Non-Linear Filtering; Pseudo-Linear Kalman Filter; Reduction of Computational Cost; Artificial Neural Networks; Aerospace applications.

1 - Introduction

In order to use the various inventions developed in engineering in a safe and planned way, it is necessary to have knowledge of the different parameters or states that describe the dynamics of the system in question. The instruments were developed to measure states, however it is not possible to measure them perfectly, since there are disturbances from different sources and even the instrument itself can generate noise during the measurement.

The term filtering or estimation covers the set of techniques and algorithms with the objective of obtaining better estimates of the states of a system based on imperfect measurements made [1]. The appropriate method depends on several factors such as the complexity of the system, the nature of the noise and the availability of data.

Several methods have been developed over the last few decades, one of the most famous, serving as a conceptual basis, is the linear Kalman filter, which is considered the optimal filter for linear systems [2]. However, most dynamic systems present in engineering are non-linear, which limits the use of this filter. Other methods have been developed so that they can be applied to non-linear systems, such as the famous Extended Kalman Filter (EKF) and the Pseudo-Linear Kalman Filter (PSELIKA), among many others.

The EKF is the most famous non-linear estimation filter, which uses as a basis the linearization of the system around the estimated state to calculate the various matrices that describe the dynamics of the system. Despite having been used in several applications, it presents serious limitations since it presents erratic and possibly divergent behavior in very non-linear systems or in very imprecise initial estimates [1], [3], [4].

The PSELIKA aims to reduce these limitations as it restructures the non-linear model into a pseudo-linear format, which allows the application of linear techniques, such as linear Kalman filtering for state estimation, but capturing the non-linearities of the system [5], [6].

Like the previous method, this one also has limitations, precisely due to the need to solve the Algebraic Riccati equation associated with the system model and perform matrix inversions to calculate the filter gain matrix in real time. These two processes significantly increase the computational cost of the filtering method, so a method of reducing this cost is necessary.

Therefore, a tool that has been integrated into filtering methods with the aim of improving their performance and reducing computational cost is the artificial neural network which appears as a solution to this problem. This can be described as a set of non-linear functions capable of obtaining experimental knowledge through a learning process. [7]

With the intention of reducing the main limitation of the PSELIKA estimation method, taking advantage of presenting a pseudo-linear structure, this work proposes the integration of a Multilayer Perceptron (MLP- Multilayer Perceptron) neural network into the PSELIKA algorithm, with the main objective of reducing the original computational cost, finally implementing and validating an algorithm called PSELIKA-MLP.

Since it is possible to apply linear estimation methods to pseudo-linear systems, as they present a quasi-linear structure with state-dependent parameterization, it enables the off-line calculation of the covariance matrices of the estimation error and consequently the gain, by solving the Algebraic Riccati equation associated with the system.

By training a neural network to obtain the value of the gain matrix from a non-linear regression, it is possible to reduce the computational cost of this step. This method will then be validated in the case study of estimating the attitude and angular rate of an aerospace vehicle.

This work is structured as follows: firstly, the problem will be presented more completely, then the algorithm will be presented with the aim of reducing the problem and finally an implementation example will be used to validate the implemented algorithm, as well as comments and conclusions about it.

2 - Problem Statement

Linear Kalman filtering is considered the optimal estimator for linear systems [2]. However, most existing dynamical systems are non-linear in nature. As this method cannot be implemented in non-linear systems, several methods have been developed to try to overcome this difficult non-linear estimation problem.

2.1 - EKF

Since the optimal Bayesian solution to the problem requires complete knowledge of the probability density function of the variables, it ends up not being viable in practice [3]. Therefore, several methods have been developed in order to approximate this estimator, such as the extended Kalman filter, which, as the name suggests, is an extension of the filter already presented for non-linear problems [1], [8], [9].

This filter consists of the linearization of the non-linear functions that represent the dynamic model of the system and the measurement model, based on approximations of the Taylor series that define the non-linear functions [8], [9], [10].

Being used in several applications and precisely due to its ease of implementation and low computational cost, this filtering method is recommended as the first type of estimator for different types of systems [1], [4]. However, this filter must be used with caution since initial estimates that differ too far from the current state may result in divergence of the filter [1], [3], [4].

The equations that constitute the extended Kalman Filter algorithm for nonlinear systems with discrete measurements can be found in Table 1 [1].

System Model	$\dot{x}(t) = f(x(t), t) + w(t); \quad w(t) \sim N(0, Q(t))$
Measurement Model	$z_k = h_k\big(x(t_k)\big) + v_k, \qquad k = 1,2, \dots; \; v_k \sim N(0, R_k)$

Initial Conditions	$x(0) \sim N(\hat{x}_0, P_0)$
Other Assumptions	$E[w(t)v_k^T] = 0 \; \forall k \; e \; t$
State Estimate Prediction	$\dot{\hat{x}}(t) = f(\hat{x}(t), t)$
Error Covariance Prediction	$\dot{P}(t) = F(\hat{x}(t), t)P(t) + P(t)F^T(\hat{x}(t), t) + Q(t)$
Correction of the State Estimate	$\hat{x}_k(+) = \hat{x}_k(-) + K_k\left[z_k - h_k\left(\hat{x}_k(-)\right)\right]$
Kalman Gain Matrix	$K_k = P_k(-)H_k^T\left(\hat{x}_k(-)\right)\left[H_k\left(\hat{x}_k(-)\right)P_k(-)H_k^T\left(\hat{x}_k(-)\right) + R_k\right]^{-1}$
Error Covariance Correction	$P_k(+) = \left[I - K_k H_k\left(\hat{x}_k(-)\right)\right]P_k(-)$
Definitions	$F(\hat{x}(t), t) = \frac{\partial f(x(t), t)}{\partial x(t)}\Big\|_{x(t)=\hat{x}(t)}$ $H_k\left(\hat{x}_k(-)\right) = \frac{\partial h_k(x)}{\partial x}\Big\|_{x=\hat{x}_k(-)}$

Table 1: Summary of Extended Continuous-Discrete Kalman Filter Equations.

The main difference between the conventional filter and the extended version is that the gain matrix K_k is a random variable that depends on the estimate $\hat{x}(t)$ through the matrices $F(\hat{x}(t), t)$and $H_k\left(\hat{x}_k(-)\right)$ [1]. This results from the fact that it was chosen to linearize the matrices f and h_k based on the estimate of $x(t)$. Therefore, the K_k values must be calculated in real time and cannot be pre-computed, with the estimation accuracy being dependent on the trajectory [1].

Since the P_k matrix in the equations in Table 1 is only an approximation of the true covariance matrix, there is no guarantee that the estimate obtained will be close to the true optimal estimate. However, EKF can perform good approximations in several important practical applications [1]. Furthermore, due to the similarity to the linear Kalman filter itself, and in computational terms, this method tends to be used in the first instance for cases of non-linear filtering [1].

2.2 - PSELIKA

The Pseudo-Linear Kalman Filter (PSELIKA) is an ordinary linear Kalman filter where the State-Dependent Coefficients (SDC) are a function of the best available estimate. This filter is fundamentally based on the problem of stabilization or regulation of a non-linear system, through a Linear-Quadratic Regulator (LQR), using the solution of the state-dependent Algebraic Riccati Equation [5], [11].

Due to the duality between the LQR and the Kalman Filter, it is possible to extend the SDRE concept to Kalman filtering [12]. In both cases, there is a need to solve the Algebraic Riccati equation, which is the optimal solution to the problem, and by applying the concept of factorization it is possible to transform a non-linear system into a pseudo-linear structure, which thus allows the application of the concepts of Linear Kalman Filtering, if the state values are considered constant throughout a filtering step. Thus, this method presents Pseudo-linear Kalman Filter terminology.

Performing the discretization of the system and using the concepts of Linear Kalman filtering [2], the equations for the discrete PSELIKA in Table 2 are obtained [13].

System Model	$x_k = A_{d_{k-1}}(x)x_{k-1} + B_{d_{k-1}}(x)u_{k-1} + w_{k-1}, \;\; w_k \sim N(0, Q_k)$
Measurement Model	$z_k = C_k(x)x_k + v_k, \qquad v_k \sim N(0, R_k)$
Initial Conditions	$E[x_0] = \hat{x}_0, E[(x_0 - \hat{x}_0)(x_0 - \hat{x}_0)^T] = P_0$
Other Assumptions	$E\left[w_k v_j^T\right] = 0 \; \forall \; j, k$
State Estimate Propagation	$\hat{x}_k(-) = A_{d_{k-1}}(x)\hat{x}_{k-1}(+) + B_{d_{k-1}}(x)u_{k-1}$
Error Covariance Propagation	$P_k(-) = A_{d_{k-1}}(x)P_{k-1}(+)A_{d_{k-1}}^T(x) + Q_{k-1}(x)$
Correction of the State Estimate	$\hat{x}_k(+) = \hat{x}_k(-) + K_k[z_k - C_k(x)\hat{x}_k(-)]$
Error Covariance Correction	$P_k(+) = [I - K_k C_k(x)]P_k(-)$
Kalman Gain Matrix	$K_k = P_k(-)C_k^T(x)[C_k(x)P_k(-)C_k^T(x) + R_k(x)]^{-1}$

Table 2: Summary of discrete pseudolinear Kalman filtering equations.

The great advantage of using this filter, in relation to EKF, is its robustness to large errors in determining the initial estimate and disparity in the model definition [14], with its counterpart being the possible increase in computational cost [4], due to the need to solve the Riccati equations. Even so, since it presents a pseudo-linear structure, it is possible to use linear Kalman filtering methods in its formulation, which allows an off-line calculation of the system's gain matrices.

However, the limitation is found during the process of solving the Riccati equation to determine the covariance matrix of the estimated state error, since it is exhausting from a computational point of view, requiring matrix inversions that increase its complexity in

proportion to the number of dimensions present in the system model. There is, therefore, a need for a method that reduces the computational cost of the PSELIKA filter, while maintaining its filtering accuracy as much as possible, which was possible with the PSELIKA-MLP filter.

3 - Neural Kalman Filtering

Combining neural networks with Kalman filtering is a topic that has been addressed for more than two decades [15], [16]. Filtering methods have been used as an aid tool for training neural networks in various applications and neural networks have been used with the aim of improving filter performance or reducing its computational cost.

Since linear Kalman filtering methods can be applied to models that have a pseudo-linear structure, it is possible to take advantage of the structure of Pseudo-linear Kalman filters and use an MLP neural network to calculate the system gain matrix in function of the current state estimate and the processing error covariance matrices Q and measurement error R. This can be done if the non-linear function that describes the dynamic system presents the condition $f(0) = 0$ [6]. Thus, the idea would be to use a non-linear function to approximate the K gain matrix using the current estimated state as an argument.

The Kalman gain matrix K can be pre-calculated for a certain state vector value by solving the Discrete Algebraic Riccati Equation (DARE) associated with that system in that state [6], [17]. This property of systems with a pseudo-linear structure allows the values of the K matrices to be used as target values for training a non-linear regressor type neural network depending on the respective estimated state.

The range of possible states must be selected before training to cover as much as possible the system's states during the algorithm's operation.

Consider the following equations that describe the system and measurement model already factorized and discretized:

$$\begin{aligned} x_k &= A_d(x_{k-1})x_{k-1} + B_d(x_{k-1})u_{k-1} + w_{k-1} \\ z_k &= C(x_k)x_k + v_k \end{aligned} \tag{3.22}$$

The DARE associated with the state vector x_k allows finding the value of the covariance matrix P_k, which then allows the calculation of the gain matrix K_k, the target Y_k of the neuronal network, based on equation [6]:

$$Y_k^T = K_k^T = (C_k P_k C_k^T + R)^{-1} C_k P_k A_d^T(x_{k-1}) \tag{3.23}$$

To find the matrix P_k it is necessary to solve the following DARE [17]:

$$A_d(x_{k-1})P_kA_d^T(x_{k-1}) - P_k - (A_d(x_{k-1})P_kC_k^T)(R_k + C_kP_kC_k^T)^{-1}\left(C_k^TP_kA_d^T(x_{k-1})\right) + Q_k = 0 \quad (3.24)$$

As can be seen, to solve DARE it is necessary to obtain the values of $A_d(x_{k-1})$, C_k, Q_k and R_k. The method that will be used to solve the DARE will be based on the dlqr(*) function of the Matlab software [18], which presents as output the values of K_k and P_k for the given equation. Thus, the K_k matrix will be calculated with respect to the x_{k-1} value.

The transition matrices are obtained through continuous model discretization methods [2]. The propagation method that will be used for the filters will be integral approximation. This method assumes that the input values in the system are constant during the period in which they are applied, also known as zero-order hold [19].

3.1 - Training and validation of the neural network

The neural network will be trained using the scikit-learn (sklearn) library available for the Python programming language as an aid tool. This library presents several training models, and the one that was selected for this algorithm was *MLPRegressor* (Multilayer Perceptron Regressor), with the 'lbfgs' [20] or L-BFGS [21] solution method.

It is important to mention that the argument values in both training cases must be standardized, that is, they must be changed so that the final mean of the values is zero, with unit standard deviation, so that the training process is more effective [22]. The standardization model must be applied to any argument value used in the final neural network. The same process is not necessary for the target network value and can be changed according to the required performance.

The coefficient of determination, or R^2, will be used as a method of evaluating the performance of the MLPRegressor model. One of the most used equations for calculating R^2 is described below, which will be used to measure network performance [23]:

$$R^2 = 1 - \frac{\sum_i(Y_i - \hat{Y}_i)^2}{\sum_i(Y_i - \bar{Y})^2} \quad , \quad \bar{Y} = \frac{1}{m}\sum_{i=1}^{m} Y_i \quad (3.25)$$

Where $\hat{Y}_i$ represents the estimated or predicted value of observation Y_i and $\bar{Y}_i$ represents the average of the m observed values.

It is important that the set of test elements is not contained in the set used for training, to avoid the occurrence of a phenomenon called overfitting, where the neural network returns accurate values for the training data, but not for new data.

3.2 - Proposed Algorithm – PSELIKA-MLP

The training process is quite experimental, and it is necessary to carry out multiple experiments to obtain the best performance, since the optimization algorithm can be trapped in some local minimum. The main factors that change the performance of the filter are the form of factorization of non-linear functions, the covariance matrices $Q(t)$ and $R(t)$, the definition of the domain, the distribution of selected domain points in addition to the number of layers and neurons per layer.

Considering all the points mentioned, Table 3 describes the NN (Neural Network) training algorithm and Table 4 the PSELIKA-MLP (Pseudo-linear Kalman Filter with Multi-Layer Perceptron) algorithm [24].

1 Training initialization N_{corr}	
1.1 Selection of the set of state vectors x_i for training	Domain=$\{x_i\}_p$
1.2 Selection of filter covariance matrices	$Q_0 = Q_{initial}, R_0 = R_{initial}$
1.3 Associated DARE solution	$[P_k] = dlqr(A_d^T, C_k^T, Q_k, R_k)$
1.4 Calculation of the K_k gain matrix	$Y_k^T = K_k^T = (C_k P_k C_k^T + R)^{-1} C_k P_k A_d^T(x_{k-1})$
1.5 Selecting the number of hidden layers	layers=l
1.6 Selection of the number of N_l neurons for each layer	neurons = N_l
2 N_{corr} training	
2.1 Use of the L-BFGS method	MLPRegressor-Method='lbfgs'
2.2 N_{corr} performance value	Coefficient of determination R^2

Table 3: PSELIKA-MLP filter training algorithm.

1 Filter initialization	
Assume initial state vector	$\hat{x}_0(+) = x_0$
2 State Estimate Propagation	
State propagation equation	$\hat{x}_k(-) = A_{d_{k-1}}(x)\hat{x}_{k-1}(+) + B_{d_{k-1}}(x)u_{k-1}$

3 State estimate correction	
Calculation of the gain matrix K_k	$K_k=N_{corr}(\hat{x}_{k-1}(+))$
State correction equation	$\hat{x}_k(+)=\hat{x}_k(-)+K_k[z_k-C_k(x)\hat{x}_k(-)]$

Table 4: PSELIKA-MLP filter algorithm.

4 - Application

This section's main objective is to validate the use of MLP-type neural networks to calculate the gain matrix according to the methods proposed in the previous section. The case study in question is the estimation of the angular rate and attitude of an aerospace vehicle based on quaternion measurements [13].

Validation will be done by comparing the PSELIKA-MLP algorithm with the PSELIKA and EKF algorithms. Comparisons will be made in terms of estimation accuracy, in addition to temporal complexity, that is, computational time, and spatial complexity, that is, number and size of parameters and variables used during the algorithm.

The following subsection presents the mathematical models that describe the angular dynamics of an aerospace vehicle, as well as in its pseudo-linear format with the factorization used for the model used in the PSELIKA and PSELIKA-MLP filter. The same is done for the data observation model.

4.1 - Dynamic Model

The angular dynamics equation of an aerospace vehicle can be described as follows [25]:

$$I_{SC}\dot{\omega} + \dot{h}_{SC} + \omega \times (I_{SC}\omega + h_{SC}) = T_{SC} \tag{4.40}$$

In this case, I_{SC} corresponds to the vehicle's inertia matrix, ω the vehicle's angular rate vector, h_{SC} the vector of the angular momentum of the vehicle's momentum wheels and T_{SC} the sum of the external moments applied to the vehicle.

Given that I_{SC} is invertible, it is possible to rewrite this equation as follows in pseudo-linear format [13]:

$$\dot{\omega} = A'(\omega)\omega + u(t) \tag{4.41}$$

Where:

$$A'(\omega) = I_{SC}{}^{-1}[(I_{SC}\omega + h_{SC}) \times] \tag{4.42}$$

$$u(t) = I_{SC}^{-1}(T_{SC} - \dot{h}_{SC}) \tag{4.43}$$

Where $[(h) \times]$ represents the outer product matrix of a vector $h = [h_1 \quad h_2 \quad h_3]^T$.

F'(w) presents exactly eight different ways of expressing the nonlinear equation of the dynamics of an aerospace vehicle [13].

To describe the attitude, the best thing to do is to use rotation quaternions. The dynamics equation can be described by [25]:

$$\dot{q} = \frac{1}{2} Q\omega, \tag{4.44}$$

Where:

$$Q = \begin{bmatrix} q_4 & -q_3 & q_2 \\ q_3 & q_4 & -q_1 \\ -q_2 & q_1 & q_4 \\ -q_1 & -q_2 & -q_3 \end{bmatrix} \tag{4.45}$$

So, when combining (4.2) and (4.5) we have:

$$\begin{bmatrix} \dot{\omega} \\ \dot{q} \end{bmatrix} = \begin{bmatrix} I^{-1}[(I\omega + h) \times] & 0 \\ \frac{1}{2} Q & 0 \end{bmatrix} \begin{bmatrix} \omega \\ q \end{bmatrix} + \begin{bmatrix} I^{-1}(T - \dot{h}) \\ 0 \end{bmatrix} \tag{4.46}$$

The measurement model based on quaternions is quite simple, being linear and obtained from measurements of quaternions

$$z = [0_{3\times4} \quad I_4] \begin{bmatrix} \omega \\ q \end{bmatrix} \tag{4.47}$$

Where $0_{3\times4}$ represents a null matrix of dimensions 3×4 and I_4 represents the identity matrix of order 4.

Since the equations describing the angular and measurement dynamics model are described by state-dependent pseudo-linear equations, they can be used in the PSELIKA filter.

4.2 - Description of the PSELIKA filter

From the equations that describe the dynamics and measurement model it is possible to form the PSELIKA filter [13]. Suppose:

$$x = \begin{bmatrix} \omega \\ q \end{bmatrix} \tag{4.48}$$

Then it is possible to rewrite (4.7) and (4.8) as a stochastic model according to the following equations

$$\begin{gathered} \dot{x} = A(x)x + u(t) + w(t) \\ z = H(x)x + v(t) \end{gathered} \tag{4.49}$$

Where:

$$A(x) = \begin{bmatrix} I^{-1}[(I\omega + h)\times] & 0 \\ \frac{1}{2}Q & 0 \end{bmatrix}, H(x) = [0_{3\times4} \quad I_4] \tag{4.50}$$

The training steps and numerical results will be based on this PSELIKA filter model.

4.3 - Neural Network Training

After experimental analysis of the NN training, it was noticed that for a similar value of performance coefficient R^2, the network with fewer hidden layers performed better in relation to computational cost. Thus, the training stage will be limited to selecting the number of neurons to the single hidden layer depending on the respective performance coefficient R^2.

The neural network training domain, in relation to the states of the system's dynamic model, was selected so that all elements of the angular rate vector have an absolute maximum value of 1 [rad/s]. Quaternion vectors have a natural limitation where the unit value is the absolute maximum they can reach. For this training it was considered that the angular momentum of the momentum wheels as well as its derivative are constant and have an initial value:

$$\begin{gathered} h = [1 \quad 1 \quad 1]^T kg \cdot \frac{m^2}{s} \\ \dot{h} = 0 \end{gathered} \tag{4.51}$$

Since the matrix $A(x)$ is dependent on the value of h, if the value of h changes it would be necessary to solve the state-dependent Riccati equation again. To make it possible to use a function $h(t)$, it would be necessary to add this variable as one of the states, or as a parameter for network training, which will not be done in this work. The same occurs for the covariance matrices Q and R.

Finally, the number of state vectors selected for training was five thousand points, which were generated randomly, according to a uniform distribution within the selected domains. Of these points, 70% of the set was used for training while the rest was used for the testing stage. In the training carried out for the network, the same sets of states were used for training, and the same for testing, and the value of the performance coefficient R^2 was calculated for both.

The inertia matrix used for the vehicle, and the covariance matrices are:

$$I_{SC} = \begin{bmatrix} 20.26 & -1.37 & -2.17 \\ -1.37 & 23.25 & -0.37 \\ -2.17 & -0.37 & 27.83 \end{bmatrix}, Q_k = (0.01)^2 I_7 , \qquad R_k = (2)^2 I_4$$

Where $I_k$represents the identity matrix of order k, Q_k is the processing covariance matrix and R_k the measurement covariance matrix.

Training Results

Training was carried out with a number of neurons within the range of 20 to 300 and the results can be seen in Figure 1.

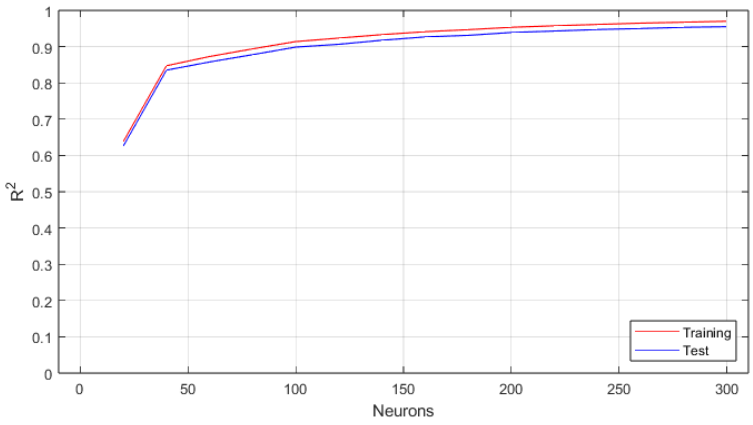

Figure 1: Training performance coefficient for case study as a function of the number of neurons.

Regarding the performance of the filter, after experimental analysis, it was found that there is no significant change in the value of the R^2 coefficient for a number greater than 200 neurons. As computational cost is the focus of this network, this quantity was selected for the network structure. The value of R^2=0.953 was obtained for the training set and R^2=0.939 for the test set in the case of 200 neurons.

It should be noted that the results obtained are a direct function of the selected training set, both training and testing. A different selection could lead to significantly different results, but this analysis is beyond the scope of this work.

4.4 – Simulation and Results

The equations in (4.7) are used to simulate the reference attitudes and angular rates. These parameters are simulated in Matlab software and with the aid of the fourth-order Runge-Kutta (RK) method [26]. Therefore, for this simulation, a simulation step of T_s =0.01s and number of steps N_{steps}=3000 was considered. The following state value was used as the initial value of the simulation:

$$x_0 = [\omega_1\ \omega_2\ \omega_3\ q_1\ q_2\ q_3\ q_4]^T = [0.300\ \ 0.200\ \ 0.500\ \ 1.000\ \ 0.000\ \ 0.000\ \ 0.000]^T$$

The following values were considered for the standard deviation of the error:

$$\sigma_{quat} = 0.1$$

$$\sigma_Q = 0.01 \quad \sigma_R = 2$$

Where σ_{quat} represents the value of the standard deviation of the error associated with the elements of the quaternion measurement vector. Furthermore, σ_Q and σ_R represent the standard deviation values used for the Q_k and R_k matrices respectively, in both PSELIKA and EKF filters.

It was also considered that no external moment is applied to the aerospace vehicle, having a zero value throughout the simulation.

$$T_{SC}(t) = 0\ [N.m]$$

The initial estimated state $\hat{x}_0$ for initializing the 3 filters will be considered as if the initial state were not known, as follows:

$$\hat{x}_0 = [\omega_1\ \omega_2\ \omega_3\ q_1\ q_2\ q_3\ q_4]^T = [0.000\ \ 0.000\ \ 0.000\ \ 0.000\ \ 0.000\ \ 0.000\ \ 1.000]^T$$

The value used as the covariance matrix P_o for the PSELIKA and EKF filters will be the same, assuming that it is unknown.

$$P_o = I_7$$

To compare the different filters directly in terms of accuracy, the RMSE index will be used, that is, the Root Mean Square Error. In this case, $N_{sim} = 10000$ Monte Carlo simulations will be performed. The Monte Carlo simulations carried out will be in relation to the white noise added to the measurement, according to a normal distribution, with a standard deviation equal to that mentioned in this section for each observation variable. In this case, the RMSE will be calculated for the angular rate vector and the quaternion vector estimated at each time t_k.

For the angular rate, the estimated vectors are directly compared with the reference simulation, the same being done for the quaternion vector. The expressions are described in the following equations:

$$RMSE_{\omega,k} = \sqrt{\frac{\sum_{i=1}^{N_{sim}} \left(\left|\left|(\omega_k - \widehat{\omega}_i)\right|\right|_2^{\,2} \right)}{N_{sim}}} \tag{4.52}$$

$$RMSE_{q,k} = \sqrt{\frac{\sum_{i=1}^{N_{sim}} \left(\left|\left|(q_k - \hat{q}_i)\right|\right|_2^{\,2} \right)}{N_{sim}}} \tag{4.53}$$

Where $\left|\left|x\right|\right|_2$ is the Euclidean norm of the vector x of dimension n given by $\left|\left|x\right|\right|_2 = \sqrt{x_1^2 + \cdots + x_n^2}$, and the index k corresponds to the parameter given for time t_k.

Another performance index that will be compared will be the maximum error obtained for each of the instants t_k among the N_{sim} simulations, also known as maximum norm or infinite norm. This index makes it possible to understand how the filters behave in the worst cases.

The maximum value measured for both cases will be given by:

$$\left|\left|\Omega_k\right|\right|_\infty = max\left(\left|\left|(\omega_k - \widehat{\omega}_i)\right|\right|_2 \right)_{i=1}^{N_{sim}} \tag{4.54}$$

$$\left|\left|\Xi_k\right|\right|_\infty = max\left(\left|\left|(q_k - \hat{q}_i)\right|\right|_2 \right)_{i=1}^{N_{sim}} \tag{4.55}$$

Where Ω_k represents the set of N_{sim} values of the Euclidean norm of the difference between the obtained value of ω in the i-th Monte Carlo simulation and the reference value for a time t_k, and Ξ_k represents the same, but in relation to the quaternion vectors obtained for the time t_k. The function $max(\cdot)_{i=1}^{N_{sim}}$ represents the maximum value of the set of N_{sim} elements.

4.4.1 - Accuracy

The performance of the filters is evaluated based on the graphical results of equations (4.13) and (4.14) for the RMSE and (4.15) and (4.16) for the maximum error, making it possible to compare them. The results are represented in figures 2, 3, 4 and 5. In these figures, the black, green and blue lines represent the EKF, PSELIKA and PSELIKA-MLP filters respectively.

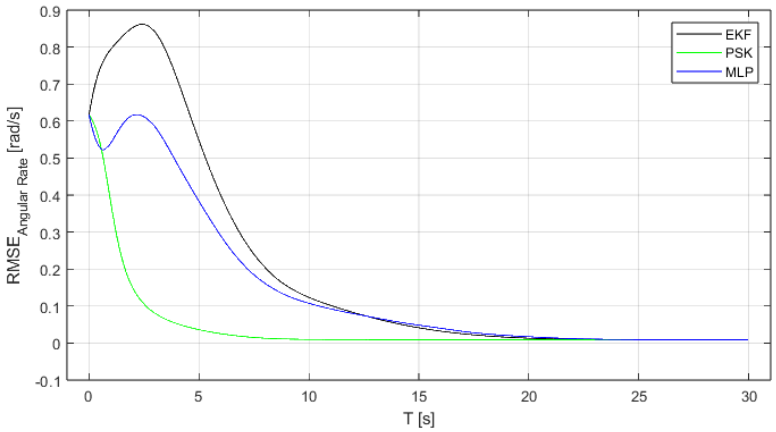

Figure 2: Comparison of the RMSE obtained from EKF, PSELIKA and PSELIKA – MLP for the angular rate vector.

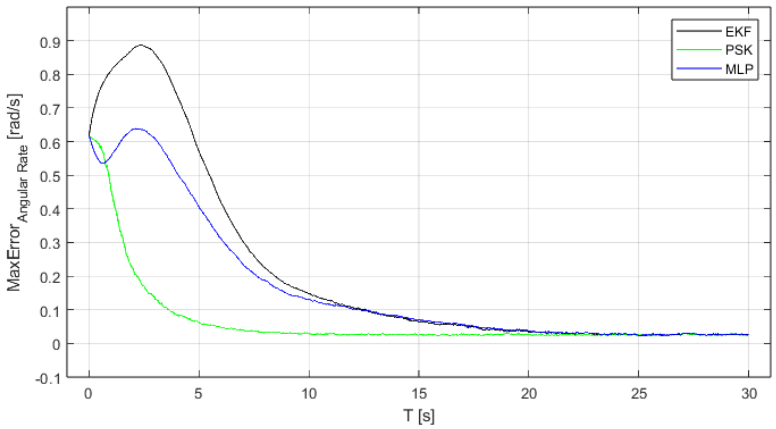

Figure 3: Comparison of the maximum error obtained from EKF, PSELIKA and PSELIKA – MLP for the angular rate vector.

It is possible to observe in figures 2 and 3, respectively, the RMSE of the angular rate as well as its maximum error value as a function of the instant t_k in seconds. It is possible to see that the RMSE has a high initial value and that it reduces overtime for the three filters, and the same occurs for the maximum error.

The high initial value is precisely because the initial estimate of the state of each filter is quite different from the correct value of the initial state. Even so, the three filters were able to converge to an increasingly lower value of RMSE and Maximum Error until reaching a stable value. Of the three filters, the one that showed the fastest convergence was PSELIKA, followed by EKF and PSELIKA-MLP. However, the EKF presented the largest peak in relation to both graphs with a maximum RMSE of approximately 0.86 rad/s and maximum error of 0.89 rad/s, an expected behavior, due to the limitations of the filter.

PSELIKA-MLP did not show faster convergence but demonstrated similar performance to the other filters after stabilization. The performance of PSELIKA-MLP was superior to that

of EKF up to approximately 12.62 seconds and superior to PSELIKA up to 0.56 seconds, but slightly inferior at all other times until stabilization. Stabilization occurs for the RMSE of approximately 0.009 rad/s and for the maximum error of approximately 0.026 rad/s, close to 25 seconds.

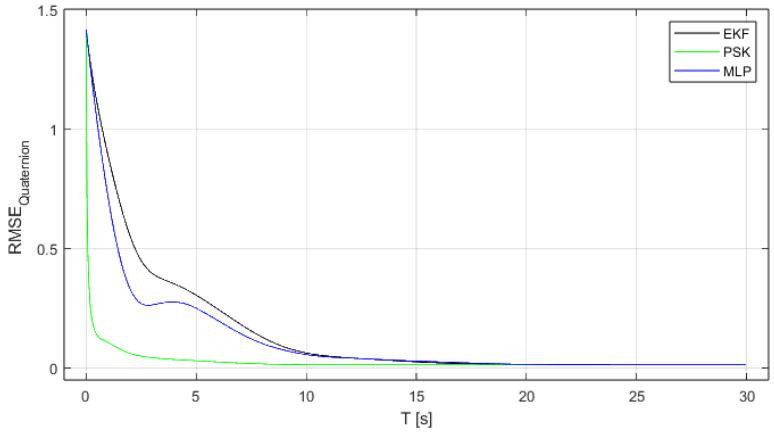

Figure 4: Comparison of the RMSE obtained from EKF, PSELIKA and PSELIKA – MLP for the quaternion vector.

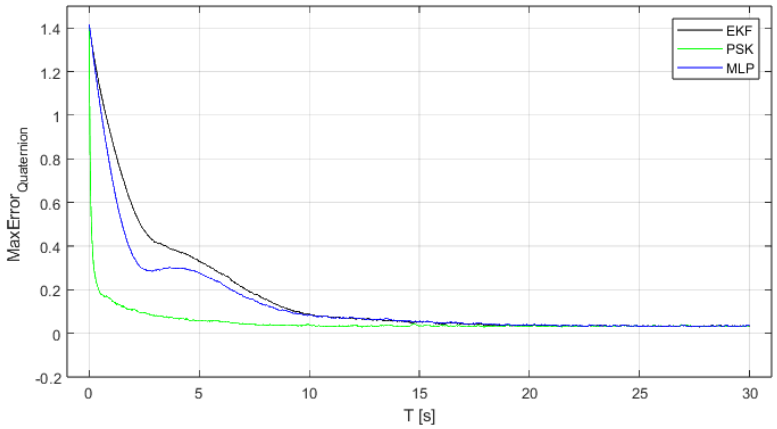

Figure 5: Comparison of the maximum error obtained from EKF, PSELIKA and PSELIKA – MLP for the quaternion vector.

It is possible to observe in figure 4 and 5, respectively, the RMSE of the quaternions as well as their maximum error value as a function of the instant t_k in seconds. The performance of these filters for quaternions is similar to that for angular rate. However, in this case, the maximum value of RMSE and maximum error is the same for the three filters, having a value of approximately 1.41.

Regarding RMSE and maximum error over time, it presents almost the same graphical behavior in relation to previous angular rates where the performance of PSELIKA-MLP was superior to that of EKF up to approximately 12.5 seconds, but inferior at all times compared

to PSELIKA until stabilization. Stabilization occurs for the RMSE of approximately 0.013 and for the maximum error of approximately 0.035, close to 25 seconds.

4.4.2 Temporal Complexity

Temporal complexity is a measure of how long the algorithm takes to execute as a function of the size of the input.

Regarding the temporal complexity of the filters, the comparison was made based on the average execution time of the algorithms after 1000 Monte Carlo simulations. The comparison will be made in relation to the PSELIKA-MLP algorithm with the relative gain or loss of time in relation to the other filters.

The results obtained were a faster execution time of approximately 23.50% and 60.23% compared to the PSELIKA and EKF algorithms respectively. The PSELIKA-MLP algorithm was faster compared to the other algorithms, which was expected, since the neural network used is quite simple. The associated standard deviations obtained are approximately 0.523% and 0.585% respectively.

What makes the other algorithms more complex in computational terms is the need to perform several matrix multiplications to determine the values of the gain matrix K, in addition to a matrix inversion and the need to carry out the propagation and correction of the covariance matrix P. Still thus, the EKF algorithm was much more costly because it needed to calculate the discretized Jacobian matrix of the system model and the Jacobian matrix of the measurement model. In the PSELIKA algorithm, the discretized matrix of the A_d system model is used to propagate the estimated state and the covariance matrix, requiring it to be calculated only once.

4.4.3- Spatial Complexity

Spatial complexity is a measure of the memory space used by an algorithm until the end of its execution.

For comparison, the *whos* function of the Matlab software will be used, which allows observing the space occupied in memory by each variable. All variables were saved in double format, that is, the space occupied by a single element of a vector or matrix will occupy 8 bytes.

The PSELIKA-MLP algorithm was the one that presented the greatest use of memory space with a total of 59880 bytes used, of which 58408 bytes were used only for the execution of the implemented neural network, which includes the matrices of the neural network parameters, and the factors used to state standardization, corresponding to approximately

97.54% of the total space required. In second place, the EKF algorithm required 2472 bytes followed by PSELIKA with 2360 bytes, occupying 95.77% and 95.96% less memory space respectively.

5 – Conclusions and Future Work

Several methods have been developed to perform non-linear filtering or estimation, based on the linear Kalman filter. Methods such as the Extended Kalman Filter (EKF) and the Pseudolinear Kalman Filter (PSELIKA), among many others, were developed with the aim of overcoming this problem. However, each method has its limitations.

With the aim of reducing the limitation inherent to the PSELIKA method, the computational cost, this work proposed the use of a Multi-Layer Perceptron (MLP) neural network as a non-linear regressor for calculating the filter gain matrix without the need to directly estimate the associated covariance matrix, implementing and validating an algorithm called PSELIKA-MLP.

The PSELIKA method requires the estimation of the covariance matrix of the estimation error to calculate the gain matrix, and this is carried out by solving the algebraic Riccati equation associated with the model. As this equation depends only on the matrices that describe the system and measurement models, by assuming that the processing and measurement error covariance matrices are constant, it is possible to estimate the estimation error covariance matrix from the estimated state value in the previous step. Thus, it allows the calculation of the associated subsequent gain matrix to be done a priori and can be estimated from a neural network acting as a non-linear regressor, taking only the state vector as an argument.

After describing the implemented algorithm, the use of the PSELIKA-MLP filter was validated in a case study.

The objective in the case study was to estimate the attitude and angular rate of an aerospace vehicle based on noisy measurements of the quaternion vector that describes the vehicle's attitude according to a fixed frame of reference, for an unknown initial state value. Validation was carried out by comparing the PSELIKA-MLP algorithm with the PSELIKA and EKF algorithms. Comparisons were made in terms of estimation accuracy, in addition to temporal complexity and spatial complexity.

From the results, it is possible to see that the PSELIKA-MLP filter presents close results in terms of accuracy, with a considerable reduction in terms of computational cost (23.50% faster than PSELIKA and 60.23% faster than EKF) compared with the other two methods. Even so, it presents a limitation in terms of required memory space, requiring much more

memory than the others, precisely because of the need to store the network parameter matrices.

Still, PSELIKA-MLP is a very experimental method in which changing several parameters in the network training stage can drastically affect the results in terms of accuracy, temporal and spatial complexity.

In the training stage, the definition of the training domain and training set selection method is extremely important, since, for example, a very large domain with few points can result in lower accuracy, but greater robustness of the filter and vice versa, but their impact was not evaluated in this work.

If effective methods of optimizing the other network parameters and hyper-parameters are implemented in order to guarantee robustness and consistently acceptable performances, the PSELIKA-MLP algorithm can become a good alternative method to PSELIKA and EKF, especially in applications where the computational cost must be reduced as much as possible.

5.1 Future Work

This work aims to introduce a neural network to pseudo-linear Kalman filtering, trying to overcome its main limitation, with the PSELIKA-MLP algorithm.

The results are promising, but their implementation requires a more pragmatic and optimized approach, with a more in-depth analysis of how to design the neural network and the pseudo-linear system to maximize robustness and accuracy of the filter, in addition to maintaining the computational cost low which is the main objective. It is also necessary to expand this analysis to other non-linear cases.

Finally, this algorithm assumes constant statistical conditions, so, after applying the previous considerations, it would be interesting to try an adaptive approach where the processing and measurement error covariance matrices change throughout the filter operation.

References

[1] A. Gelb, J. Joseph F. Kasper, J. Raymond A. Nash, C. F. Price and J. Arthur A. Sutherland, "Optimal Linear Filtering," in *Applied Optimal Estimation*, Cambridge, Massachussetts, The MIT Press, 1974, pp. 102-155.

[2] R. E. Kalman, "A New Approach to Linear Filtering and Prediction Problems," *Journal of Basic Engineering (ASME),* vol. 82D, pp. 35-45, 1960.

[3] S. J. Julier and J. K. Uhlmann, "Unscented filtering and nonlinear estimation," *Proceedings of the IEEE,* vol. 92, no. 3, pp. 401-422, 2004.

[4] W. Leung and C. Damaren, "A Comparison of the Pseudo-Linear and Extended Kalman Filters for Spacecraft Attitude Estimation," in *AIAA Guidance, Navigation, and Control Conference and Exhibit*, Providence, Rhode Island, 2004.

[5] R. Harman and I. Bar-Itzhack, "The use of pseudo-linear and SDARE filtering for satellite angular-rate estimation," in *AIAA Guidance, Navigation, and Control Conference and Exhibit*, Boston,MA,U.S.A., 1998.

[6] T. Çimen, "State-Dependent Riccati Equation (SDRE) Control: A Survey," *IFAC Proceedings Volumes,* vol. 41, no. 2, pp. 3761-3775, 2008.

[7] S. S. Haykin, *Neural Networks and Learning Machines*, 3ª ed., Hamilton, Pearson Prentice Hall, 2009.

[8] A. H. Jazwinski, *Stochastic Processes and Filtering Theory*, Academic Press, 1970.

[9] H. W. Sorenson, *Kalman filtering : theory and application*, IEEE Press, 1985.

[10] R. P. Wishner, J. A. Tabaczynski and M. Athans, "A comparison of three non-linear filters," *Automatica,* vol. 5, pp. 487-496, 1969.

[11] R. Azor, I. Y. Bar-Itzhack, J. K. Deutschmann and R. R. Harman, "Angular-Rate Estimation Using Quaternion Measurements," *NASA Technical Report* 19990014340, Washington, D.C., 1998.

[12] D.-H. Lee and J. Hu, "A Study of the Duality between Kalman Filters and LQR Problems," Purdue University, *Department of Electrical and Computer Engineering Technical Reports*, Paper 476, West Lafayette, Indiana, 2016.

[13] R. R. Harman and I. Y. Bar-Itzhack, "Pseudolinear and State-Dependent Riccati Equation Filters for Angular Rate Estimation," *Journal of Guidance,* vol. 22, no. 5: Engineering Notes, pp. 723-725, 1999.

[14] M. Horton, "Real-time identification of missile aerodynamics using a linearised Kalman filter aided by an artificial neural network," *IEE Proceedings - Control Theory and Applications,* vol. 144, no. 4, pp. 299-308, 1997.

[15] Y. Dahan, G. Revach, J. Dunik and N. Shlezinger, "Uncertainty Quantification in Deep Learning Based Kalman Filters," *arXiv,* https://doi.org/10.48550/arXiv.2309.03058, pp. 1-5, 2023.

[16] A. A. Stoorvogel and A. Saberi, "The discrete algebraic Riccati equation and linear matrix inequality," *Linear Algebra and its Applications,* vol. 274, no. 1, pp. 317-365, 1998.

[17] I. The MathWorks, "dlqr," 2024. [Online]. Available: https://www.mathworks.com/help/control/ref/dlqr.html. [Accessed 20 May 2024].

[18] R. DeCarlo, *Linear Systems: A State Variable Approach with Numerical Implementation*, Englewood Cliffs, Prentice Hall, 1989.

[19] scikit-learn Developers, "MLPRegressor," 2024. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html#sklearn.neural_network.MLPRegressor. [Accessed 26 April 2024].

[20] J. Nocedal, "Updating Quasi-Newton Matrices with Limited Storage," *Mathematics of Computation,* vol. 35, nº 151, pp. 773-782, 1980.

[21] M.-C. Popescu, V. Balas, L. Perescu-Popescu e N. Mastorakis, "Multilayer Perceptron and Neural Networks," *WSEAS Transactions on Circuits and Systems,* vol. 8, pp. 579-588, Julho 2009.

[22] T. O. Kvalseth, "Cautionary Note about R2," *The American Statistician,* vol. 39, no. 4, p. 279–285, 1985.

[23] F. F. d. Silva, "Filtragem de Kalman Neuronal para Aplicações Aeroespaciais," *MSc. Dissertation* (*to be defended*), University of Beira Interior, Department of Aerospace Sciences, Covilhã, Portugal, 2024.

[24] I. Bar-Itzhack, R. Harman and D. Choukroun, "State-Dependent Pseudo-Linear Filter for Spacecraft Attitude and Rate Estimation," in *AIAA Guidance, Navigation, and Control Conference and Exhibit*, Monterey, California, 2001.

[25] J. R. Wertz, in *Spacecraft Attitude Dynamics and Control*, Dordrecht, Holland, Reidel Publishing Co., 1978, pp. 512-513.

[26] M. Hamisu, I. Saidu and M. Y. Waziri, "A Simplified Derivation and Analysis of Fourth Order," *International Journal of Computer Applications,* vol. 9, no. 8, pp. 51-55, 2010.