POLITECNICO DI TORINO

Master's Degree Course in Mechatronic Engineering

Master's degree Final Thesis

Virtual sensor design for payload estimation in excavators

Thesis Advisor
Prof. Carlo Novara

Candidate
Alessandro Possetti

A.A.2017/2018

Acknowledgements

I would first like to thank my thesis advisor prof. Carlo Novara. I have been extremely lucky to have a supervisor who cared so much about my work and who responded to my questions so promptly.

I would also like to thank Modelway which had given me the possibility to work at their project. In particular, prof. Mario Milanese, engr. Stefano Mosca and engr. Ilario Gerlero were always disposed to clarify every question about my research.

I would also like to acknowledge every professor of Politecnico di Torino for the teachings and interests transmitted during the lessons.

I must express my very profound gratitude to my mother and to my girlfriend for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis.

Finally, I would like to thank Rosa Mosso for the English tips, my course mates for any given help and to all my friends who have given me the best memories I have.

This accomplishment would not have been possible without them. Thank you.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Excavators are among the most used mobile construction machines. They are primarily adopted to move material and perform excavations. Due to their ability to collect and lift material, the payload is often required to be known. The term "payload" refers to the weight of material loaded at the operating end of the dipper arm. The knowledge of payload is very important both for commercial purposes, not to overcome excavator mechanical limits at given positions of arms and to respect laws restriction during the loading of trucks. Unfortunately, payload may not be measured directly, so virtual sensors are used to estimate it from noisy measurements. Literature offers many methods allowing to design performer virtual sensors under different noise assumptions and accuracy measures. For example, the Kalman filter [16, 17] is an iterative algorithm able to estimate quantities difficult or impossible to measure, comparing statistically noisy measurements against values provided by model equations. The Kalman filter requires to work correctly that system model is linear and exactly known, and noises are considered Gaussian distributed with 0 mean. Possible estimator alternatives are: $\mathcal{H}_2$ filters [10, 36], $\mathcal{H}_\infty$ filters [10, 36, 24] and ℓ_1 filters [23, 39].
Non-linear observers are technically difficult to design since they are highly unstable. The most common approach to design non-linear observers is probably the Extended Kalman Filter (EKF) [14, 2] which firstly linearises the system around the actual state conditions and finally applies the Kalman Filter. Other possible methods to design non-linear state observers are: Unscented Kalman Filters (UKF) [15], Particle Filters (PF) [33] and Ensamble Kalman Filters (EnKF) [8].
As previously said, virtual sensors for non-linear systems rarely assure the required stability of solutions. Another recurrent problem in virtual sensor design is that system equations are often unknown. A common approach consists in identifying the system model from a set of data and in finally designing the filter using the identified model equations. The approximations introduced by the two-step procedure are often incompatible with the required precision of the estimates, because this procedure may offer only an approximative knowledge of the system model.

New approaches have been developed in last years to increase the quality of solutions. Excellent solutions are given by the use of Direct Filtering techniques (DF) [30, 29], which allow to design DVSs (Direct Virtual Sensors) directly from set of data.
DF approach allows to avoid the study of a system model and provides much better solutions than any other methods.

Two estimation strategies are possible for payload estimators in excavators. Payload estimators working in quasi-static conditions are able to provide the estimates when all excavator arms are moving at low constant angular speed. Payload estimators working in dynamic conditions are able to provide the estimates at any motion condition of arms.
Virtual sensors which estimate the payload under quasi-static conditions are a consolidated reality and are all widely based on patent EP0736752A1 [35]. Observers based on this patent, estimate payloads interpolating actual measured values with respect to a characteristic map. The considered inputs are the boom angle (ϕ_1), the angle formed by the boom and the dipper arm (ϕ_2) and the hydraulic pressure difference across the main cylinder (dP). The characteristic map is acquired at calibration phase. The calibration phase consists in measuring the quantities ϕ_1, ϕ_2 and dP, for two payload references and over a pre-defined range of motion at same angular speeds. Estimators based on this patent should not estimate payload at different angular speeds than ones used to calibrate them, in order to preserve precision of estimates.
The method described in "Payload Estimation in Excavators: Model-Based Evaluation of Current Payload Estimation Systems" [4] tries to increase the estimation robustness at different angular speeds in two-piece boom excavators, by adding to the characteristic map another set of data measured at different arms angular speeds. This approach allows to improve the robustness of estimations at different angular speeds but it still doesn't allow to correctly estimate when angular speeds are varying in time.
Great improvements have been obtained by approach described in "A Method for Payload Estimation in Excavators" [40] where authors developed a model-based observer able to dynamically estimate payload, in two-piece boom excavators, as function of joint torques and link accelerations. Solutions provided by its simulations show much better precision than ones estimated by other methods, but this estimator is still far from being applied on reality since measurement noises are not considered by its internal structures.
An artificial neural network approach has been investigated in paper "An Artificial Neural Network Approach to Payload Estimation in Four Wheel Drive Loaders" [11] to design dynamic payload estimators for four-wheel-drive loaders, but test results did not pass the minimum requirements acceptable for the industry.

The aim of this thesis is to provide valid solutions to estimate correctly the payload in dynamic conditions for two-arms excavators, analysing several methodologies and keeping in consideration real problems as well.
The state of the art of actual dynamic payload estimators doesn't allows to deal with noisy measurements without violating the 1% of accuracy required by industry. The purpose of the developed virtual sensor will be to firstly guarantee the quality of estimates in presence of noisy data and in dynamic conditions of arms. In order to allow developed virtual sensor to provide accurate payload estimates, the rotating platform of the excavator has to not move or move at constant speed with respect to the ground and the excavator has to be level.

Another important requirement of the dynamic estimator project is to keep the economic costs low as much as possible. Due to this reason, the sensors have to be a compromise between performances and costs, and have to be low in number.
Due to the high non-linearity of the model, two design approaches have been used to develop observers and to derive the best estimator.
The first design approach has been called "Classical Observers Approach" and allows to design virtual sensors as Extended Kalman Filters. The classical observers approach may be seen as the evolution of the approach used in "A Method for Payload Estimation in Excavators" [40], since it uses extended Kalman filters which are based on model equations too.
The second design approach has been called "DVS Approach" and allows to design DVSs using Neural Network approach [1, 25]. The DVS approach takes inspiration from "An Artificial Neural Network Approach to Payload Estimation in Four Wheel Drive Loaders" [11], but it's focused on completely different structures of neural networks.
Unfortunately, it hasn't been possible to measure data from real field, so all experiments are simulated by a simulator made ad hoc and a set of data has been collected from it.

Chapter 2 starts by presenting a general overview about excavators. Section 2.2 describes briefly CAN network and communication protocols, which allow interaction between sensors, actuators and ECUs. The aim of chapter 2 is to provide a general overview about the physical system and to give a brief description about electrical connections and protocols which make exploitable algorithms implemented in ECUs.
Chapter 3 presents virtual sensors and classical observers, focusing attention on Kalman filter and its extended version.
Chapter 4 introduces DVSs and gives a brief description of the direct filtering technique. Many approaches allow to design powerful DVSs and neural networks are indubitably one of them. Neural networks are deeply described in section 4.2, analysing both main features and training algorithms.
Chapter 5 starts with a brief presentation of the Lagrangian mechanics and then, main passages to the model formulation are described. Section 5.3 presents features of the simulator and describes simulations.
Chapter 6 describes the main development phases of virtual sensors, while Chapter 7 shows the results obtained from the simulations of the previously designed virtual sensors.
Finally, Chapter 8 gives conclusions and reflections on possible future works.

Chapter 2

System Description

2.1 Excavators

Excavators are heavy machines used primarily to move material not particularly coherent. They may be equipped with particular tools, allowing excavators to perform other kinds of tasks.
An excavator may execute: digging operations, building demolition, forestry work, material handling, etc.
Excavator may be divided into three main sections, which are:

- The undercarriage;
- The house;
- The arms.

The undercarriage is the lowest excavator section. It includes tracks, gears and the hydraulic motor, which is used to convert the oil power into tracks movement. The undercarriage is connected to the house through a central pin, which allows the rotation of the house respect the undercarriage.

Figure 2.1: CAT 345C L [5]

The house is the upper part of the excavator and it includes the operator cab, the counterweight, the engine, oil pumps and other minor parts. In operator cab are located all the excavator commands. The engine convert fuel power into mechanical one. Mechanical power is then converted into fluid power by oil pumps, which are connected with the engine through a shaft.
The arms allow the excavator to move the tool. They are moved respect each other by means of hydraulic cylinders, commanded through spool valves placed in the operator cab. Boom is the arm connected directly to the house, while the dipper is the middle arm which provides force to the tool. The tool is the device used to carry out a particular function. The bucket is the most versatile tool, because it allows excavations and other main tasks. Other possible excavator tools are: rippers, multi processors, wood grapples and hydraulic breakers (see picture 2.2).

(a) *Ripper*

(b) *Multi-processor*

(c) *Wood-grapple*

(d) *hydraulic-breaker*

Figure 2.2: Excavator attachments [6]

2.2 ECU and CAN Protocol

ECUs are embedded systems which control one or more electrical systems or subsystems in vehicle field. An ECU is composed by:

- RAM memory: It's volatile memory and stores running data;
- FLASH memory: It's non-volatile memory and stores persistent data;
- CPU: It's the processor which runs code;
- I/O: It manages input data coming from sensors or output signal to transducers.

ECUs communications are allowed by a CAN network which connects together every sensors, actuators and ECUs.

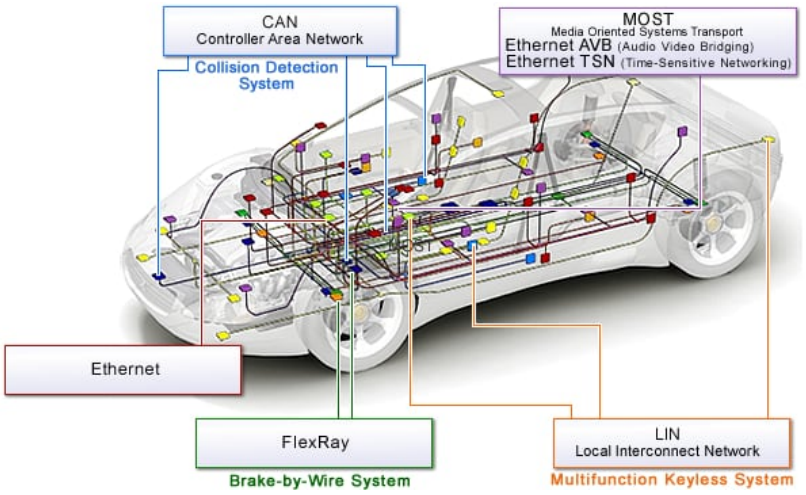


Figure 2.3: CAN network [32]

CAN is a communication protocol developed by BOSH in 1985 for in-vehicle networks. The name means Controller Area Network (CAN) and it has been developed to replace the antiquated hard-wires network.

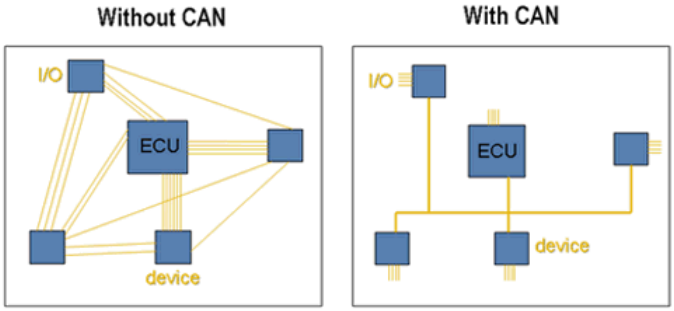


Figure 2.4: Invehicle networks [13]

CAN bus delivers messages from one sending node to one or numerous receiving nodes, packaging informations in bit sequences called data frames.
Data frames are composed by:

- SOF (start-of-frame) bit: It identifies the starting of a new frame;
- Arbitration ID: It identifies the type of message and the message priority;
- SRR (remote transmission request) bit;
- IDE (identifier extension) bit: It identifies the message format (standard or extended);
- RTR (remote transmission request) bit: It identifies the frame type of the message (data frame or remote frame);
- r0 (reserved) bit
- DLC (data length code): It identifies the number of byte of Data Field;
- Data Field: Series of bits representing the message;
- CRC (cyclic redundancy check): It is used to detect errors;
- ACK (ACKnowledgement) slot: Bits used by receivers to confirm the message has been received;
- EOF (end-of-frame) bit: It identifies the end of the frame.

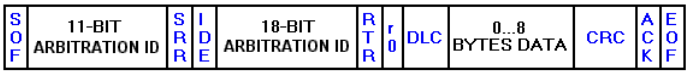


Figure 2.5: Data frame [13]

Other types of frames are:

- Remote frame: Frame created by a node and used to request data transmission from one another node;

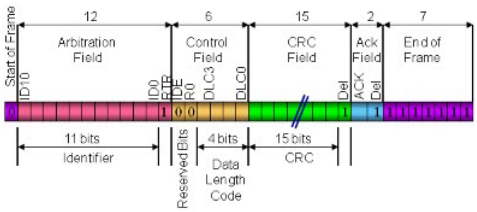


Figure 2.6: Remote frame [18]

- Error frame: Frame used to report errors;

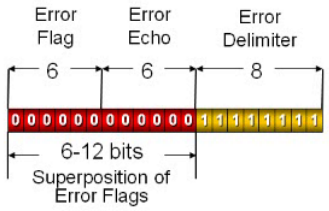


Figure 2.7: Error frame [18]

- Overload frame: Frame used to delay data frames and/or remote frames.

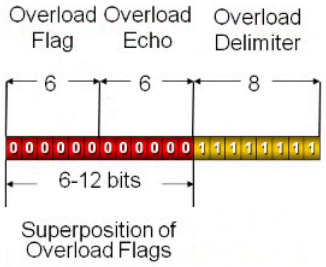


Figure 2.8: Overload frame [18]

Chapter 3

Classical Observers

3.1 What Is a Virtual Sensor?

Systems are logical entities having one or more inputs and outputs, which are governed by mathematical laws (mathematical model). They may be classified into two categories:

- Static systems;
- Dynamic systems.

In static systems the current outputs depend only on the current inputs. On the other hand, in dynamic systems the current outputs depend on the current inputs and on the current internal states. A dynamic system may be represented as:

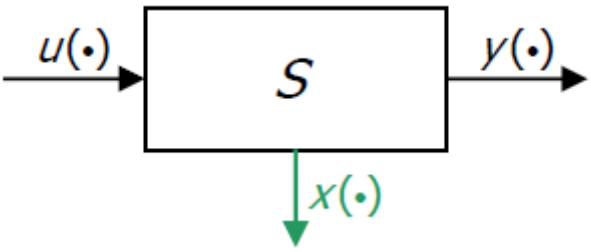


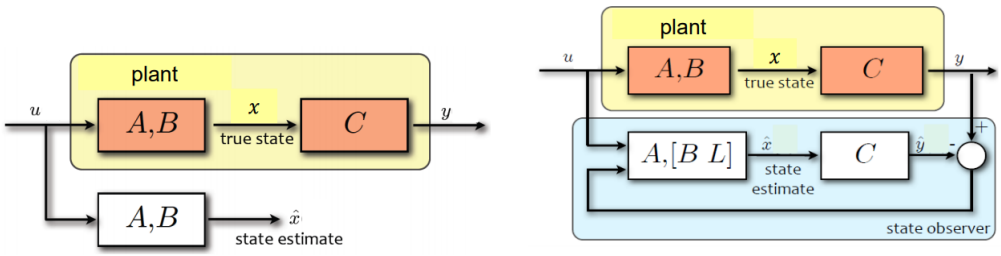
Figure 3.1: Dynamic system [37]

where u are the inputs, y are the outputs and x are the states.
Although inputs and outputs may be always measured, the states often aren't measurable. When states aren't measurable and the system is observable, an observer may be adopted to have access to internal states.

Observers may be classified into:

- Open-loop observers;
- Closed-loop observers.

Open-loop observers estimate internal states only by the current inputs, while closed-loop observers use also past outputs for the internal states estimation. Generally, closed-loop observers are more performer respect the open-loop ones.



(a) *Open-loop observer* (b) *Closed-loop observer*

Figure 3.2: Open/closed loop observers [27]

Due to their capacity to estimate quantities impossible/difficulty to measure, observers may be used to design virtual sensors.
The term "Virtual Sensors" includes sensors which cannot be physically implemented, so they estimate quantities from system equations and other measures.
In most practical situations, measures provided by sensors are affected by noises. To overcome this problems, a Kalman filter (or its extended version) may be used if noises are considered Gaussian distributed with 0 mean.

3.2 Kalman Filter

A Kalman filter [16, 17] is a recursive algorithm able to estimate internal states of a system, starting from a series of noisy measurements.
Rudolf Emil Kálmán officially discovered it, although Thorvald Nicolai Thiele and Peter Swerling developed a similar algorithm earlier.
For its versatility, Kalman filter is used for states observations, noise filtering, parameter estimation and sensors fusion.

Suppose to have a linear system affected by processing noise (w) and measurement noise (v):

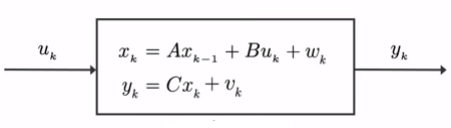


Figure 3.3: Discrete system [20]

A Kalman filter may be designed to estimate the system internal states from u and y measurements, if noise v and w are considered Gaussian distributed with 0 mean. Knowing the covariance Q of noise w and the covariance R of noise v, the Kalman filter estimates the system internal states $\hat{x}_k$.

Estimates provided by Kalman filter are computed in two phases:

- The prediction phase;
- The update phase.

During prediction, Kalman filter calculates:

$$\hat{x}_k^- = A\hat{x}_{k-1}^- + Bu$$
$$P_k{}^- = AP_{k-1}A^T + Q$$

where $\hat{x}_k$ is the predicted state and $P_k{}^-$ is the variance of the a priori estimate.
Once they have been calculated, the feedback gain K is found as:

$$K_k = \frac{P_k{}^- C^T}{CP_k{}^- C^T + R}$$

and the state estimate is updated:

$$\hat{x}_k = \hat{x}_k^- + K_k(y_k - C\hat{x}_k^-)$$

as also the error covariance:

$$P_k = (1 - K_k C)P_k{}^-$$

This procedure is repeated at every clock.

Kalman filter may be adopted only if the system to observe is linear. Instead, an Extended Kalman filter [14, 2] may be used if the system is non-linear.

Extended Kalman filter is a non-linear state observer, which linearises the system around the mean of the current state estimate.
Given the non-linear system:

$$\begin{cases} x_k = f(x_{k-1}, u_k) + w_k \\ y_k = g(x_k) + v_k \end{cases}$$

Every time step, a local linearisation has to be performed by means of Jacobian calculation:

$$F = \frac{\delta f}{\delta x}|_{\hat{x}_{k-1}, u_k}$$
$$G = \frac{\delta g}{\delta x}|_{\hat{x}_k,}$$

Obtaining the linearised system:

$$\begin{cases} \Delta x_k \approx F \Delta x_{k-1} + w_k \\ \Delta y \approx G \Delta x_k + v_k \end{cases} \tag{3.1}$$

After the calculation of the linearised system (3.1), the classical Kalman filter is used.

Extended Kalman filter is a powerful method to study non-linear systems. However, it has the following drawback:

- Jacobians are difficult to calculate;
- It requires high computational power;
- It works only on systems having a differentiable model;
- It doesn't work optimally if systems are highly non-linear.

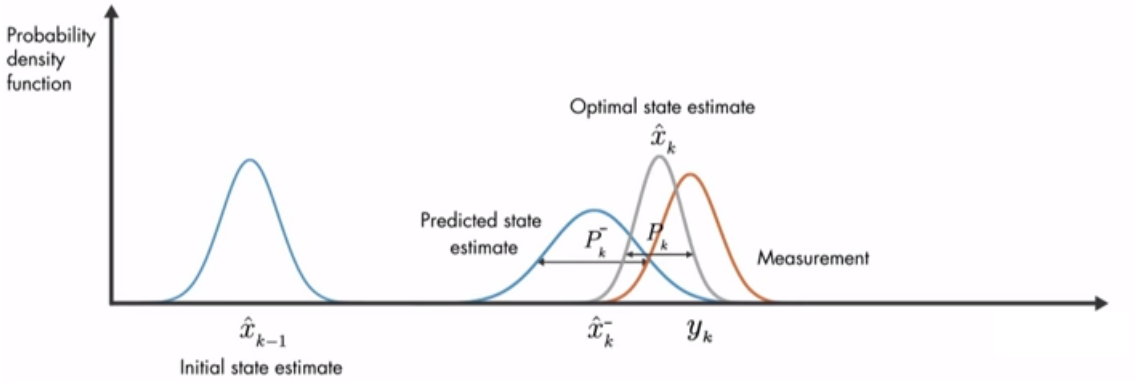


Figure 3.4: Kalman filter estimation [20]

Chapter 4

Direct Virtual Sensors

4.1 General Overview about DVS

Virtual sensors for non-linear systems are in general difficult to design since they rarely ensure the required stability of solutions.
Another relevant problem is that often system equations aren't known and the following two-step procedure has to be adopted:

- A model of the system is identified from data;
- A filter is designed from the identified model equations.

The approximation introduced by the two-step procedure might not ensure the stability of the estimation. Possible alternatives are given by the use of direct filtering techniques (DF) [30, 29], which allow to design observers directly from set of data, avoiding the study of a system model. Virtual sensors designed through direct filtering techniques are called direct virtual sensors (DVSs). Many techniques allow to design performer DVSs, and neural network approach is one of them.

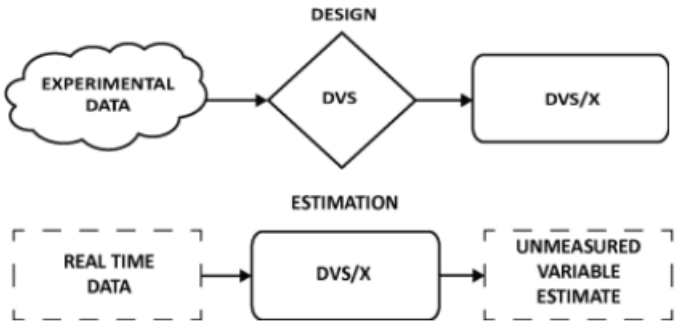


Figure 4.1: Design and estimation of a DVS [22]

4.2 Neural Networks

A neural network is a logical network able to find autonomously the relation between inputs and output by means of automatic training.

In 1943, Warren S. McCulloch, a neuroscientist, and Walter Pitts, a logician, developed the first conceptual model of an artificial neural network.
In their paper, "A Logical Calculus of Ideas Immanent in Nervous Activity" [21], they describe the concept of neural network starting from the analysis of the hidden rules behind the nervous activity.

As we know from biology, the central processing unit of nervous activity is placed in the brain and it's entrusted by particular cells called neurons. Every neuron is wired together to perform a dense network able to modify the internal state of target neurons (or other targets) through electrical signals.
The main body of a neuron is composed by soma, where the nucleo and other organs are placed. Dentrides and a long axon are placed attached to the soma. Dentrites are constituted by thin branches able to receive chemical messages from outside, while the axon is a thin protoplasmic fiber able to transmit electrical signal at speeds of 100 meters per second from the nucleo to terminal buttons.
Terminal buttons are directly interfaced with dentrites of other neurons. The gap between them is called sinapsy and it may be considered the link wiring neurons together.

The action mechanism of neurons may be described as follows:
When a message carried by electrical signal reaches the soma, the nucleo elaborates the information and if necessary a new electrical signal is created. The new information runs over the axon and reaches terminal buttons. Terminal buttons transform the electrical message in a chemical one and a neurotransmitter is released in the synapsy, ready to be captured by a dentrite or another target. If a neurotrasmitter reaches a dentrites of another neuron, the information is transformed newly in an electrical message and it's ready to be carried to the soma.

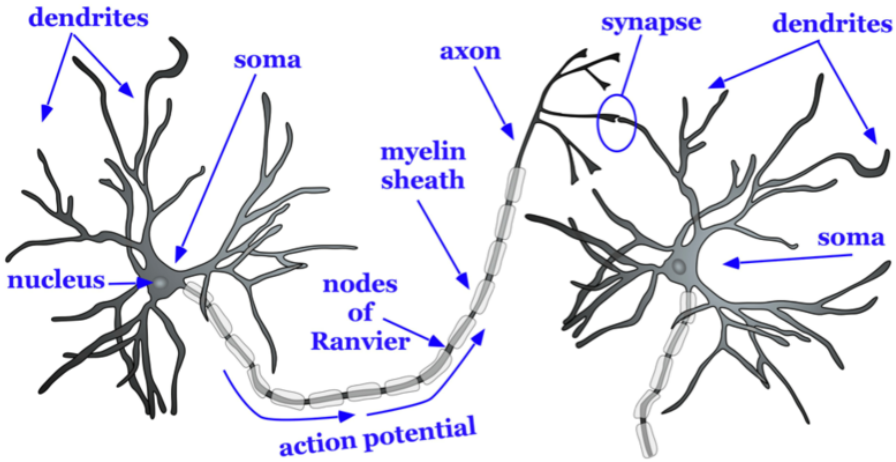


Figure 4.2: Neurons [9]

4.2.1 Feedforward Neural Networks

The structure of an artificial neural network follows similar rules of nervous activity. It's composed by neurons able to modify the state of other neurons directly connected with them.
Neurons are organized in virtual structures called layers which have the following defined rules:

- Neurons of the same layer aren't connected together and they have all the same inputs;
- Inputs of hidden layers are the outputs of the previous layer;
- Inputs of the input layer are the inputs of the network.

The first layer is called input layer and it hasn't neurons since it's just used to distribute inputs correctly to the first hidden layer. The last layer is called output layer and it has only a neuron. All other layers are called hidden layers.

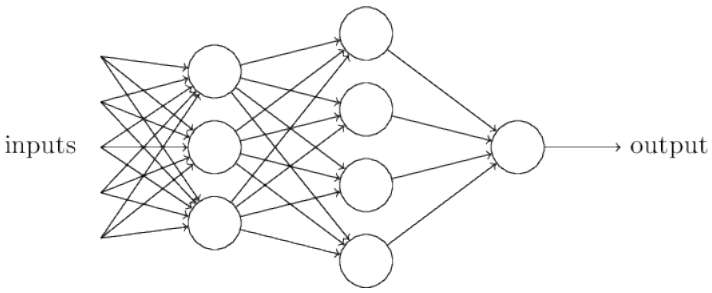


Figure 4.3: Neural network [26]

An artificial neuron is composed by a preparatory part of input signals, by a summatory and by a transfer function.
In the preparatory part, inputs are multiplied by weights. Then, the processed inputs are summed together and a bias is added to the sum. Finally, the result is passed to the transfer function.

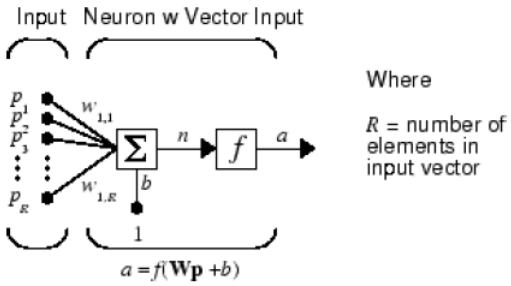


Figure 4.4: Artificial neuron [3]

Many transfer functions have been developed in last year.
The most relevant are represented below:

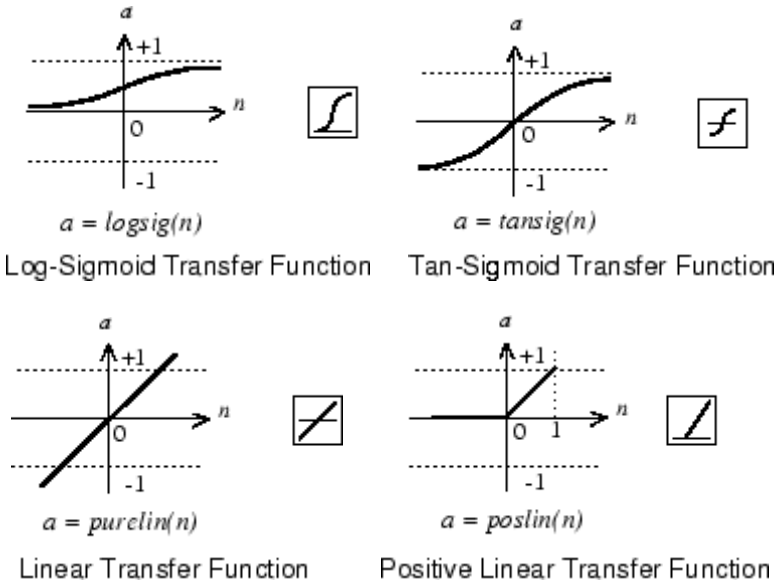


Figure 4.5: Transfer functions [3]

4.2.2 Dynamic Neural Networks

The structure of neural network described in the last section allows to manage only current inputs, without the possibility to consider past inputs and feedbacks from other layers.
Dynamic neural networks are a class of neural networks able to consider also other kinds of inputs than the current ones. The most relevant dynamic neural networks are represented below:

Time-delay Neural Networks

Time-delay neural networks are a class of dynamic neural networks, which use tapped delay lines to consider past input values. Two types of Time-delay neural networks are possible.

The first type is the "Focused Time Delay Neural Network" (also called FTDNN), which have a tapped delay line on inputs of the first hidden layer.

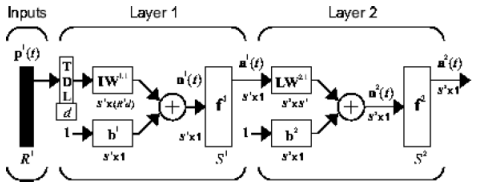


Figure 4.6: Focused Time Delay Neural Network [3]

The second type is the "Time Series Distributed Delay Neural Networks" (also called DTDNN), which have tapped delay lines on inputs of every hidden layer.

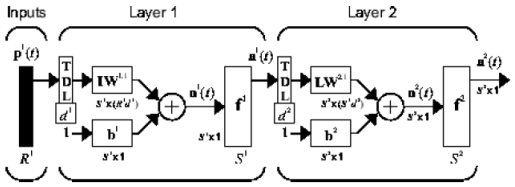


Figure 4.7: Time Series Distributed Delay Neural Network [3]

Time Series NARX Feedback Neural Networks

The "Nonlinear autoregressive networks with exogenous inputs" (also called NARXs) are autoregressive dynamic neural networks, which have tapped delay lines both on inputs than on feedback. In NARX, the feedback connection is considered from the last layer output to inputs of the first hidden layer.

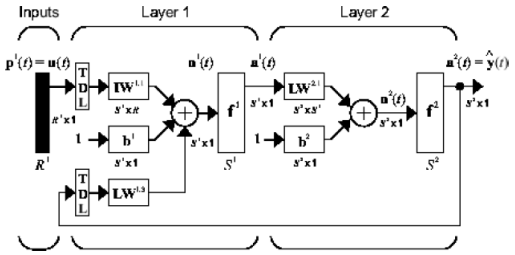


Figure 4.8: Time Series NARX Feedback Neural Network [3]

Layer-Recurrent Neural Networks

The first Layer-Recurrent Neural Network was introduced by Elman, J.L in his paper “Finding structure in time” [7]. The original Elman network has only two layers which use hyperbolic tangent sigmoids for hidden layers and a linear transfer function for the output layer.
Nowadays, Layer-Recurrent Neural Networks are referred as neural network having a closed loop structure on every layer except the last.

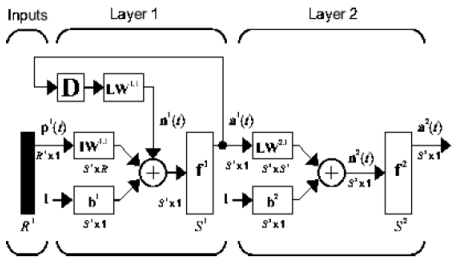


Figure 4.9: Layer-Recurrent Neural Network [3]

4.2.3 Neural Network Training

Gradient Descent Algorithm

Training is the phase in neural network design that allows to set optimal weights and biases.
There are several training algorithms able to perform an efficient parameters research. Some algorithms perform a better research than others but they require high power calculation.

One of the most relevant training algorithm is the "Gradient Descent algorithm", which is used to find weights and biases able to minimize the error between the calculated output and the desired one.
Errors are represented by means of a cost function. Probably, the most used cost function is the mean square error, which is defined as:

$$C(w,b) = \frac{1}{2n}\sum_{x} \|y(x) - a\|^2 \tag{4.1}$$

where $y(x)$ is the network output and a is the desired one.

How does gradient descent algorithm work?
As previously said, the gradient descent algorithm searches weights and biases able to minimize the cost function value.
Considering a cost function defined in two variables, the variation of its values may be defined approximately as:

$$\Delta C \approx \frac{\delta C}{\delta v_1}\Delta v_1 + \frac{\delta C}{\delta v_2}\Delta v_2 \tag{4.2}$$

Which can be rewrote as:

$$\Delta C \approx \nabla C \cdot \Delta v \tag{4.3}$$

where ∇C is the cost function gradient, defined as:

$$\nabla C = (\frac{\delta C}{\delta v_1}, \frac{\delta C}{\delta v_2})^T \tag{4.4}$$

Suppose now to relate ΔC with Δv, writing the equation:

$$\Delta v = -\eta \cdot \Delta C \tag{4.5}$$

where η is called learning rate. Equations (4.3) and (4.5) may be combined together, obtaining:

$$\Delta C \approx -\eta \cdot \|\nabla C\|^2 \tag{4.6}$$

It's possible to demonstrate from equation (4.6), that C always decrease if Δv change according:

$$v_f = v_i - \eta \cdot \Delta C \tag{4.7}$$

The term $\|\nabla C\|^2$ is always positive, so ΔC is always negative; It means that C always decrease.

Rewriting equation (4.7) in terms of weights and biases, it's possible to obtain:

$$\begin{cases} w_f = w_i - \eta \cdot \Delta C \\ b_f = b_i - \eta \cdot \Delta C \end{cases} \tag{4.8}$$

Through equations (4.8) every weight and bias may be found. Gradient descend works iteratively, so for every iteration an optimized set of weights and biases is found.
Learning rate has to be chosen carefully. Learning rate too small takes a lot of time to find a good solution, while learning rate too big might rise the gradient instead of descend it.

Backpropagation Algorithm

As explained in the subsection before, gradient descent allows to find iteratively weights and biases from the cost function gradient.
The calculation of the cost function gradient in many training algorithms is entrusted by the "Backpropagation Algorithm".
Backpropagation is an algorithm originally introduced in the 1970s, which is able to calculate gradients using an iteratively procedure. It began to be fully appreciated since 1986 after the publication wrote by David Rumelhart, Geoffrey Hinton, and Ronald Williams [34], where they demonstrate the improvement by the use of backpropagation algorithm in neural network training.
Backpropagation algorithm is used by several training algorithms which use different kinds of cost function. There are two conditions which are needed by cost functions to allow the use of backpropagation:

- It has to be an average function ($C = \frac{1}{N}\sum_{x=1}^{N} C_x$) over cost functions C_x for individual training experiment x.
- It has to be a function of the neural network output.

Suppose to have a neural network composed by L number of layers, where parameters are represented as follows:

- $w_j k^l$ is the weight connecting neuron k of layer $(l-1)$ with the neuron j of layer l.
- b_j is the bias associated with the neuron j of layer l.
- ${a_j}^l = \vartheta(z)$ is the activation associated with neuron j of layer l and ϑ is the transfer function.
- ${z_j}^l = \sum_k w_{jk}{}^l {a_k}^{l-1} + {b_j}^l$ is the weighted input of neuron j of layer l.

Imagine to add a small change $\Delta {z_j}^l$ to the weight of neuron j^{th} of layer l, so that the neuron output becomes $\sigma(z + \Delta {z_j}^l)$ instead of $\sigma(z)$; the change propagates over the

whole network so that the cost function becomes $\frac{\delta C}{\delta {z_j}^l}\Delta {z_j}^l$.
Error ${\delta_j}^l$ of neuron j of layer l is expressed as:

$$ {\delta_j}^l = \frac{\delta C}{\delta {z_j}^l} \tag{4.9} $$

Which may be adapted for the last layer and modified using the chain rule as:

$$ {\delta_j}^L = \sum_k \frac{\delta C}{\delta {a_k}^L} \frac{\delta {a_k}^L}{\delta {z_j}^L} \tag{4.10} $$

The derivative $\frac{\delta {a_k}^L}{\delta {z_j}^L}$ vanishing for all $k \neq j$, so equation (4.10) may be simplified as:

$$ {\delta_j}^L = \frac{\delta C}{\delta {a_j}^L} \frac{\delta {a_j}^L}{\delta {z_j}^L} = \frac{\delta C}{\delta {a_j}^L} \sigma'({z_j}^L) \tag{4.11} $$

Equation (4.11) may be written as:

$$ {\delta_j}^l = \sum_k \frac{\delta C}{\delta {z_k}^{l+1}} \frac{\delta {z_k}^{l+1}}{\delta {z_j}^l} = \sum_k {\delta_j}^{l+1} \frac{\delta {z_k}^{l+1}}{\delta {z_j}^l} \tag{4.12} $$

Equations (4.11) and (4.12) are the workhorse of backpropagation algorithm. Equation (4.11) allows to find the errors associated with neurons of last layer. Then, the remaining errors are calculated iterating equation (4.12).

From errors, it's possible to calculate all weights and biases as:

$$ \begin{cases} \frac{\delta C}{\delta b_j} = {\delta_j}^l \\ \frac{\delta C}{\delta {w_{jk}}^l} = {a_k}^{l-1} {\delta_j}^l \end{cases} \tag{4.13} $$

Levenberg-Marquardt Algorithm and Bayesian Regularization

Although gradient descend algorithm has good performance in weights/biases search, the use of second-order training algorithms allows to speed up the process. Probably the most performer second-order training algorithm is the "Levenberg-Marquardt Approach", but it requires much more memory than other same order algorithms.

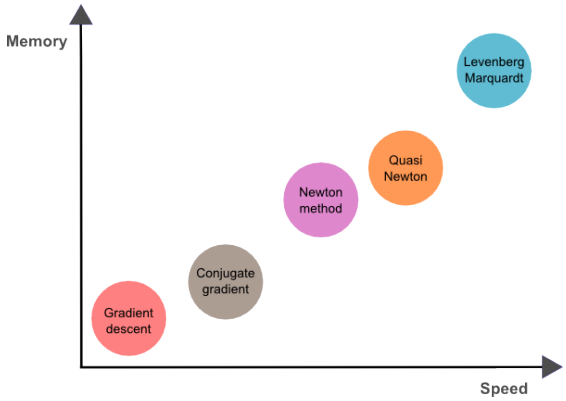


Figure 4.10: Comparison between training algorithms [31]

Given a cost function which can be expressed as a sum of squared errors e, the Levenberg-Marquardt approach update weights/biases accordingly with formula:

$$x_{k+1} = x_k - [H - \mu I]^{-1} g \tag{4.14}$$

Where the Hessian matrix H is approximated as:

$$H = J^T J$$

and gradient g is expressed as:

$$g = J^T e$$

Jacobian calculation is entrusted by backpropagation algorithm.

When μ parameter is zero, equation (4.14) is just the Newton method with approximated Hessian matrix; when μ parameter is big enough, equation (4.14) becomes gradient descent with a small step size. Newton method is faster than gradient descent, so μ should be chosen as much small as possible. Integrating "Bayesian Regularization" in cost function of Levenberg-Marquardt algorithm allows to decrease the possibility to overfit. Bayesian regularization expands the cost function to search, not only for the minimal error, but also for the minimal error using the minimal weights.
The cost function with Bayesian regularization is:

$$C(k) = \alpha E_d + \beta E_w$$

Where E_d is the sum of squared errors, while E_w is the sum of squared weights. α and β are called Bayesian hyper-parameters.

Chapter 5

System Model

5.1 Lagrangian Mechanics

Lagrangian and Newtonian mechanics are the most used approaches to model mechanical systems.
The main difference between the two methods is the use of different physical entities:

- The Newtonian mechanics considers the exchange of forces between bodies.
- The Lagrangian mechanics considers the exchange of energy between bodies.

The Newtonian approach considers vectors which are strictly dependent on the reference system chosen and some problems could be generated if the reference system would be modified. On the contrary, the Lagrangian mechanics doesn't have these problems because it considers scalars which are independent from the reference system.

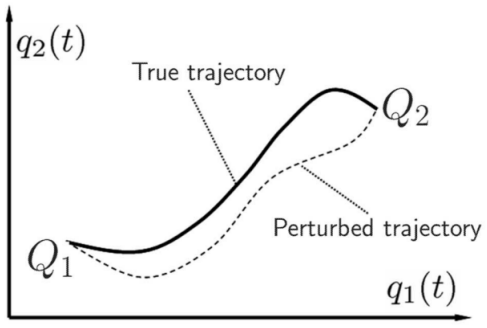


Figure 5.1: The trajectory paths [12]

The concept at the base of Lagrangian mechanics is "The principle of the least action".
Suppose to have a particle of mass m in the space $q \in Q$, subjected to a vectorial conservative field.
Consider the particle starts its motion at time t_1 in Q_1 and ends its motion at time t_2 in Q_2. The natural path described by the particle is called "True trajectory", while all other possible trajectories connecting Q_1 and Q_2 are called "Perturbed trajectories".

We may define the Action as:

$$S = \int_{t_1}^{t_2} \mathcal{L} dt \tag{5.1}$$

where $\mathcal{L} = K - P$ is the Lagrangian Functional and:

- K is the Kinetic energy
- P is the Potential energy (Gravitational+Elastic)

The principle of the least action affirms that the trajectory which minimizes the Action is the "True trajectory". On this path, the following identity is valid:

$$\frac{d}{dt}\frac{d\mathcal{L}}{d\dot{q}_i} = \frac{d\mathcal{L}}{dq_i}$$

Replacing the definition of $\mathcal{L}$, it's possible to obtain:

$$\frac{d}{dt}(\frac{dK}{d\dot{q}_i}) - \frac{d}{dt}(\frac{dP}{d\dot{q}_i}) - \frac{dK}{dq_i} + \frac{dP}{dq_i} = 0$$

The term $\frac{d}{dt}(\frac{dP}{d\dot{q}_i})$ is equal zero because the potential energy is independent from the velocity, so it may be deleted:

$$\frac{d}{dt}(\frac{dK}{d\dot{q}_i}) - \frac{d}{dt}(\frac{dP}{d\dot{q}_i}) - \frac{dK}{dq_i} + \frac{dP}{dq_i} = 0$$

If vector field is considered as non-conservative, the Lagrange equation should be modified introducing the virtual work done by non-conservative forces:

$$\frac{d}{dt}(\frac{dK}{d\dot{q}_i}) - \frac{dK}{dq_i} + \frac{dP}{dq_i} = F^{nc}$$

The non-conservative forces are composed by friction forces, plus external forces acting on the system. Friction forces may be introduced by the Rayleigh function:

$$D = \sum_{i=1}^{N} \frac{1}{2}\beta_i \dot{q}_i$$

So, the updated Lagrange equation for translating systems is:

$$\frac{d}{dt}(\frac{dK}{d\dot{q}_i}) - \frac{dK}{dq_i} + \frac{dP}{dq_i} + \frac{dD}{d\dot{q}_i} = F_i \tag{5.2}$$

While, for rotating systems is:

$$\frac{d}{dt}(\frac{dK}{d\dot{q}_i}) - \frac{dK}{dq_i} + \frac{dP}{dq_i} + \frac{dD}{d\dot{q}_i} = T_i \tag{5.3}$$

5.2 Excavator Model

Excavator has been modelled using the Lagrange approach. The Lagrangian mechanics is the most recommended method for designing multi-bodies systems.

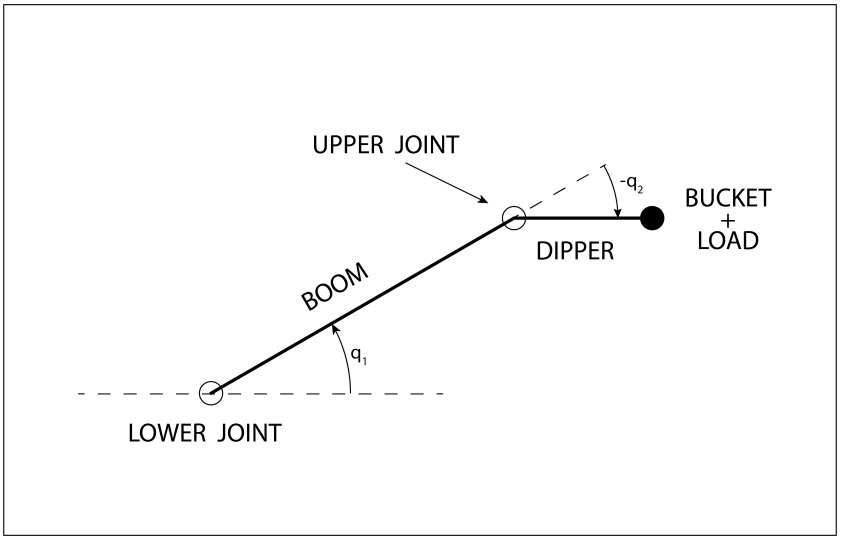


Figure 5.2: Excavator model

The model has been designed placing a reference system in every arm's barycenter. Dipper and boom have been considered as long bars and their barycentres have been placed into geometric centres of arms. Bucket and load are smaller than dipper and boom, so it's possible to consider them as point masses with barycentres placed on the dipper top end. Considering these settings, all the reference systems have:

- X-axes parallel to the axes of arms.
- Z-axes outgoing from the plain.
- Y-axes following the right-hand rule

The poses of all the reference systems respect $\mathcal{R}$ have been found using the roto-translation matrix:

$$T = \begin{vmatrix} \cos\alpha & -\sin\alpha & 0 & \Delta x \\ \sin\alpha & \cos\alpha & 0 & \Delta y \\ 0 & 0 & 1 & \Delta z \\ 0 & 0 & 0 & 1 \end{vmatrix} \tag{5.4}$$

So, the pose of every reference system is:

$$T_1 = \begin{vmatrix} \cos q_1 & -\sin q_1 & 0 & \frac{l_1}{2}\cos q_1 \\ \sin q_1 & cosq_1 & 0 & \frac{l_1}{2}\sin q_1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

$$T_2 = \begin{vmatrix} cosq_{12} & -sinq_{12} & 0 & \frac{l_2}{2}\cos q_{12} + l_1 \cos q_1 \\ sinq_{12} & cosq_{12} & 0 & \frac{l_2}{2}\sin q_{12} + l_1 \sin q_1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

$$T_3 = \begin{vmatrix} cosq_{12} & -sinq_{12} & 0 & l_2 \cos q_{12} + l_1 \cos q_1 \\ sinq_{12} & cosq_{12} & 0 & l_2 \sin q_{12} + l_1 \sin q_1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

with $q_{12} = q_1 + q_2$

From poses, the barycentre positions of both arms have been calculated as:

$$p_1 = \begin{vmatrix} \frac{l_1}{2}\cos q_1 \\ \frac{l_1}{2}\sin q_1 \\ 0 \end{vmatrix}$$

$$p_2 = \begin{vmatrix} \frac{l_2}{2}\cos q_{12} + l_1 \cos q_1 \\ \frac{l_2}{2}\sin q_{12} + l_1 \sin q_1 \\ 0 \end{vmatrix}$$

$$p_3 = \begin{vmatrix} l_2 \cos q_{12} + l_1 \cos q_1 \\ l_2 \sin q_{12} + l_1 \sin q_1 \\ 0 \end{vmatrix}$$

And the barycentre velocities as:

$$\dot{p_1} = \begin{vmatrix} -\frac{l_1}{2}\sin q_1 \dot{q_1} \\ \frac{l_1}{2}\cos q_1 \dot{q_1} \\ 0 \end{vmatrix}$$

$$\dot{p_2} = \begin{vmatrix} -\frac{l_2}{2}\sin q_{12}\dot{q_{12}} - l_1 \sin q_1 \dot{q_1} \\ \frac{l_2}{2}\cos q_{12}\dot{q_{12}} + l_1 \cos q_1 \dot{q_1} \\ 0 \end{vmatrix}$$

$$\dot{p_3} = \begin{vmatrix} -l_2 \sin q_{12}\dot{q_{12}} - l_1 \sin q_1 \dot{q_1} \\ l_2 \cos q_{12}\dot{q_{12}} + l_1 \cos q_1 \dot{q_1} \\ 0 \end{vmatrix}$$

So, the total kinetic energy associated with the system is:

$$K = \frac{1}{2}\sum_{i=1}^{2}(m_i\|\dot{p_i}\|^2 + J_i\dot{q_i}^2) + \frac{1}{2}(m_3 + m_{load})\|\dot{p_3}\|^2 \tag{5.5}$$

And the total potential energy is:

$$P = m_1 g(\frac{l_1}{2}\cos q_1) + m_2 g(\frac{l_2}{2}\sin q_{12} + l_1 \sin q_1) + (m_3 + m_{load})g(l_2 \sin q_{12} + l_1 \sin q_1)$$

The energy dissipated by joint frictions has been calculated through the Rayleigh function, as:

$$D = \frac{1}{2}\sum_{i=1}^{2} \beta \dot{q_i}^2$$

All arms are subjected by torques impressed by the hydraulic cylinders. These torques generate virtual works representable as:

$$\mathcal{W}_1 = \int T_1 \, dq_1$$
$$\mathcal{W}_2 = \int T_2 \, dq_2$$

The Lagrange equations are:

$$\frac{d}{dt}(\frac{dK_{tot}}{d\dot{q_1}}) - \frac{dK_{tot}}{dq_1} + \frac{dP_{tot}}{dq_1} + \frac{dD_{tot}}{d\dot{q_1}} = T_1$$
$$\frac{d}{dt}(\frac{dK_{tot}}{d\dot{q_2}}) - \frac{dK_{tot}}{dq_2} + \frac{dP_{tot}}{dq_2} + \frac{dD_{tot}}{d\dot{q_2}} = T_2$$

Replacing the following terms in the above equations:

$T_1 = u_1$	$T_2 = u_2$
$q_1 = x_1$	$q_2 = x_2$
$\dot{q_1} = x_3$	$\dot{q_2} = x_4$
$\ddot{q_1} = \dot{x_3}$	$\ddot{q_2} = \dot{x_4}$

and adding the identities:

$$\dot{x_1} = x_3$$
$$\dot{x_2} = x_4$$

It's possible to represent Lagrange equations in the matrix form:

$$A\dot{x} - B = 0$$

So, the model is representable as:

$$\dot{x} = A^{-1}B \tag{5.6}$$

The tricky point of these passages is the calculation of the inverse matrix A in the equation (5.6). To overcome this difficulty, the model has been represented in Matlab code as "Model.mat" (for code details, see Appendix A.1).

5.3 Simulator and Simulations

Simulator has been implemented in Simulink, considering the model designed on section 5.2. The physical parameters have been set up to simulate the excavator CAT 345C [5], which its arms measure:

	Length (m)	Weight (kg)
Boom	6.55	4600
Dipper	3.00	2410

Several sizes of bucket may be mounted onto CAT 345C. In simulator, the bucket dimensions have been set up as follows:

	Capacity (m^3)	Weight (kg)
Bucket	2.6	2900

Arms have been considered as rigid bars in inertias calculation.
The damping factors of joints have been set up with value of 3×10^5 J·s^2 to obtain a desired free evolution of the model.

Simulations have to follow trajectories coherent with ones performed by real excavators during excavations.
Due to the high non-linearity of the model, a "Non-linear predictive model controller"[28] has been used to obtain the right command activity.
Every T_s seconds, the NMPC performs a prediction over a period of T_p seconds and the command activity which minimize the error $\tilde{x}_p(\tau) \doteq r(\tau) - \tilde{x}(\tau)$ has to be set. Error $\tilde{x}_p(\tau)$ is minimized searching command activities which minimize the objective function:

$$J(u(t:t+T_p)) \doteq \int_t^{t+T_p} [||\tilde{x}_p(\tau)||^2{}_Q + ||u_p(\tau)||^2{}_R] d\tau + ||\tilde{x}_p(t+T_p)||^2{}_P \qquad (5.7)$$

The sampling and prediction time of the controller have been set up as:

- T_s=0.1 s;
- T_p=2 s.

Instead, the weight matrices have been set up as:

$$Q = \begin{vmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{vmatrix} \qquad R = \begin{vmatrix} 0 & 0 \\ 0 & 0 \end{vmatrix} \qquad P = \begin{vmatrix} 2 \times 10^{11} & 0 & 0 & 0 \\ 0 & 2 \times 10^{11} & 0 & 0 \\ 0 & 0 & 1 \times 10^3 & 0 \\ 0 & 0 & 0 & 1 \times 10^3 \end{vmatrix}$$

Simulations have been performed changing in every experiment the payload, the starting point and the arrival one.
Considering the bucket size and the earth density, the payload has been randomly picked up from 0 kg to 5000 kg.

The starting states and the arrival ones have been randomly picked up too.
The starting states are spaced from:

$$\left|-30\ deg \quad -110\ deg \quad 0\ \frac{deg}{s} \quad 0\ \frac{deg}{s}\right|$$

to:

$$\left|10\ deg \quad -70\ deg \quad 0\ \frac{deg}{s} \quad 0\ \frac{deg}{s}\right|$$

The arrival states are spaced from:

$$\left|30\ deg \quad -55\ deg \quad 0\ \frac{deg}{s} \quad 0\ \frac{deg}{s}\right|$$

to:

$$\left|50\ deg \quad -35\ deg \quad 0\ \frac{deg}{s} \quad 0\ \frac{deg}{s}\right|$$

Every measure has been discretized at 20 Hz with zero order hold technique and additives white noises have been added to the measures. White noises have been set up with sampling time equal to 0.05 s and power spectral densities equals to:

- 5×10^{4} for torque signals;
- 5×10^{-6} for angular position signals;
- 5×10^{-7} for angular speed signals;
- 5×10^{-6} for angular acceleration signals.

The dimensions of white noises have been chosen looking the datasheets of hypothetical sensors.

Every experiment has been initialized automatically by the Matlab script "Simulation.mat", which defines also the instructions for save experiments in a correct format (for code details, see Appendix A.2).

Chapter 6

Virtual Sensors Design

6.1 Classical Observers Approach

Classical virtual sensors use measures and model equations to derive the system internal states.
Unfortunately, measurement noises don't allow to find internal states directly from model equations, so the use of a filter is required to derive internal states.
Probably, the most used non-linear filter is the extended Kalman filter, which estimates internal states using an iterative algorithm. Every time step, the solution calculated from the system model is compared with noisy system measurements provided by sensors and the estimate is updated. The EKF spends some time steps to reach optimal estimates.
Sometimes, the quantities to estimate aren't system states, so it's required to increase the state vector size considering also them. The updated state vector is said to be extended.

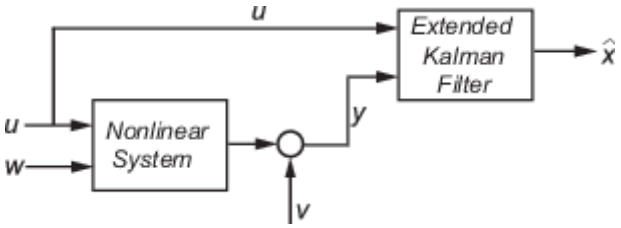


Figure 6.1: Extended Kalman filter [19]

The EKF for payload estimation problem has been designed in Simulink by means of the Extended Kalman Filter block.
Designing EKFs directly on Simulink allows to speed up both the design phase and simulations. Simulations are faster using the EKF block because it uses an optimized algorithm for the Jacobian calculation.
Data coming from sensors have been discretized at 20 Hz with the zero order hold technique.

Given the model equations:

$$\dot{x} = f(x, u)$$
$$y = g(x)$$

The functions f and g are passed to EKF block as Matlab functions. Function f has been represented in Matlab code as "myStateTransitionFcn(x,u)" (see Appendix A.4), while function g has been implemented as "myMeasurementFcn(x)" (see Appendix A.5).

Covariance matrices Q and R have been chosen observing the time behaviour of the estimations. They are:

$$Q = \begin{vmatrix} 1 \times 10^6 & 0 & 0 & 0 & 0 \\ 0 & 1 \times 10^6 & 0 & 0 & 0 \\ 0 & 0 & 1 \times 10^6 & 0 & 0 \\ 0 & 0 & 0 & 1 \times 10^6 & 0 \\ 0 & 0 & 0 & 0 & 1 \times 10^6 \end{vmatrix}$$

$$R = \begin{vmatrix} 1 \times 10^{-4} & 0 & 0 & 0 \\ 0 & 1 \times 10^{-5} & 0 & 0 \\ 0 & 0 & 1 \times 10^{-4} & 0 \\ 0 & 0 & 0 & 1 \times 10^{-5} \end{vmatrix}$$

The covariance associated with the initial conditions has been chosen equal to 5×10^{15}, while the initial conditions have been chosen as:

$$\begin{vmatrix} -0.17\, rad & -1.57\, rad & 0\, \frac{rad}{s} & 0\, \frac{rad}{s} & 5000\, kg \end{vmatrix}$$

In real applications, virtual sensors may access only a limited knowledge of the system model. Due to this reason, other two EKFs have been implemented with wrong model parameters. $EKF_{1\%}$ has model parameters different about 1% of the ones used in the system equations. Instead, $EKF_{3\%}$ has model parameters different about 3%.
Below details are reported:

	Model	$EKF_{1\%}$	$EKF_{3\%}$
L1 (m)	6.55	6.61	6.74
L2 (m)	3.00	3.03	3.09
M1 (kg)	4600	4554	4462
M2 (kg)	2410	2385.9	2337.7
Mbucket (kg)	2900	2871	2813
Beta1 ($J \cdot s^2$)	3×10^5	3.03×10^5	3.09×10^5
Beta2 ($J \cdot s^2$)	3×10^5	2.97×10^5	2.91×10^5

Observing the time evolution of the estimations, parameters of filters have been set up as follows:

	$EKF_{1\%}$
Initial covariance	5×10^{15}
Q	$\begin{bmatrix} 1.005 \times 10^6 & 0 & 0 & 0 & 0 \\ 0 & 1.005 \times 10^6 & 0 & 0 & 0 \\ 0 & 0 & 1.005 \times 10^6 & 0 & 0 \\ 0 & 0 & 0 & 1.005 \times 10^6 & 0 \\ 0 & 0 & 0 & 0 & 1.005 \times 10^6 \end{bmatrix}$
R	$\begin{bmatrix} 1 \times 10^{-4} & 0 & 0 & 0 \\ 0 & 1 \times 10^{-5} & 0 & 0 \\ 0 & 0 & 1 \times 10^{-4} & 0 \\ 0 & 0 & 0 & 1 \times 10^{-5} \end{bmatrix}$

	$EKF_{3\%}$
Initial covariance	5×10^{15}
Q	$\begin{bmatrix} 1.01 \times 10^6 & 0 & 0 & 0 & 0 \\ 0 & 1.01 \times 10^6 & 0 & 0 & 0 \\ 0 & 0 & 1.01 \times 10^6 & 0 & 0 \\ 0 & 0 & 0 & 1.01 \times 10^6 & 0 \\ 0 & 0 & 0 & 0 & 1.01 \times 10^6 \end{bmatrix}$
R	$\begin{bmatrix} 1 \times 10^{-4} & 0 & 0 & 0 \\ 0 & 1 \times 10^{-5} & 0 & 0 \\ 0 & 0 & 1 \times 10^{-4} & 0 \\ 0 & 0 & 0 & 1 \times 10^{-5} \end{bmatrix}$

6.2 DVS Approach

DVSs have been designed through neural network approach. This approach is the best one to accomplish the payload estimation requirements, because neural networks are robust in noise conditions and suit very well the regression problems. Late studies demonstrate dynamic neural networks are the most performer type of neural networks for regression problems. In particular, the best neurons combination is:

- Linear neuron for the output layer;
- Hyperbolic tangent sigmoid neurons for all other layers.

The number of layers and the number of neurons per layer are related to the problem complexity.
Usually, neurons distributed onto many layers are more performer than same number of neurons onto less layers.
Neural networks with high numbers of neurons are very good at fitting every problem, but they require an high number of experiments not to overfit.
Overfitting is a real problem in neural network design. A neural network is said it overfits when it fits very well on training experiments but it doesn't fit other experiments of the same type.

Neural network requires a pre/post processing stadium to work properly, which modify data to make them more suited for neural network.
Example of pre/post processing functions are standardization and normalization.
Standardization transforms data as:

$$x_{new} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

Normalization modifies data as:

$$x_{new} = \frac{x - \mu}{\theta}$$

No method is better than the other, so both have been tried.

Neural networks have been designed, trained and validated using the Matlab script "NNdesign.mat" (see Appendix: A.3).
Before starting to design neural networks, it should be clear that DVSs would be really implemented on excavators, so the neural networks should be simple and robust. Simple means neural networks use low number of experiments in training phase and low number of inputs to keep low the total costs. Robust means neural networks work also in noisy conditions.

Every type of neural network has been tested trying several combinations of hyperparameters, training algorithms and cost functions.

Net_1

Net_1 is a Focused Time Delay Neural Network composed by three hidden layers with 10 neurons each. The output layer is composed by a linear transfer functions, while others are composed by hyperbolic tangent sigmoid transfer functions.
Both standardization and normalization have been tested on inputs and output, but standardization performs better.
Inputs are considered at time steps t, (t-1), (t-2), (t-3), (t-4), (t-5). They are:

- Angular position and angular speed of the boom arm;
- Angular position and angular speed of the dipper arm;
- Torque impressed by the lower joint;
- Torque impressed by the upper joint.

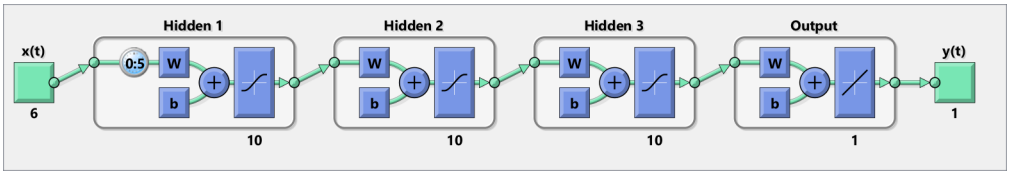


Figure 6.2: Net_1 structure

Net_1 has to be trained on 100 experiments not to overfit. The training algorithm used is the Levenberg-Marquardt with Bayesian regularization and the cost function is the mean square error.

The results obtained during the training phase are:

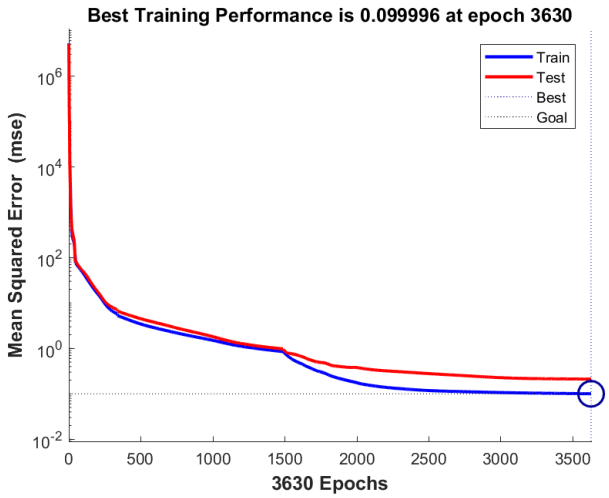


Figure 6.3: Net_1 training graph

It is possible to notice from the training graph that no overfit has occurred during the training phase. The best performance reached during training is 0.0999 Kg.

Net_2

Net_2 is a NARX neural network composed by three hidden layers with 10 neurons each. The output layer is composed by a linear transfer function, while others are composed by hyperbolic tangent sigmoid transfer functions.
For this type of neural networks as well, the standardization performs better than normalization.
Inputs used to obtain good estimates are:

- Angular position and angular speed of the boom arm;
- Angular position and angular speed of the dipper arm;
- Torque impressed by the lower joint;
- Torque impressed by the upper joint.

They are considered at time step t, (t-1), (t-2), (t-3), (t-4), (t-5) and the feedback is considered at time step (t-1).

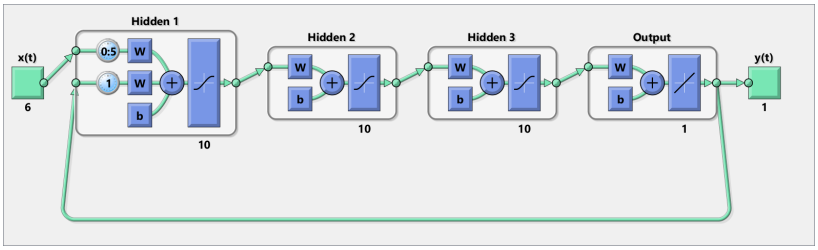


Figure 6.4: Net_2 structure

Net_2 has to require 100 experiments not to overfit. The training algorithm used was the Levenberg-Marquardt with Bayesian regularization and the training performance was the mean square error.

The results obtained during the training phase are:

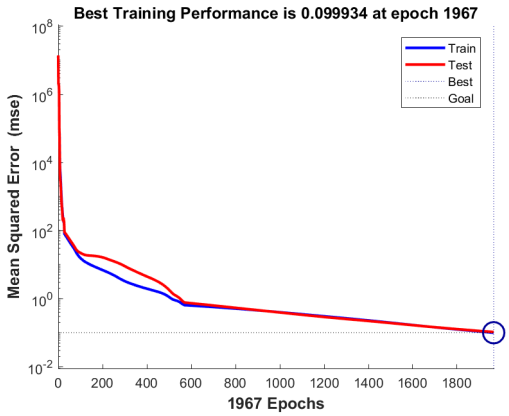


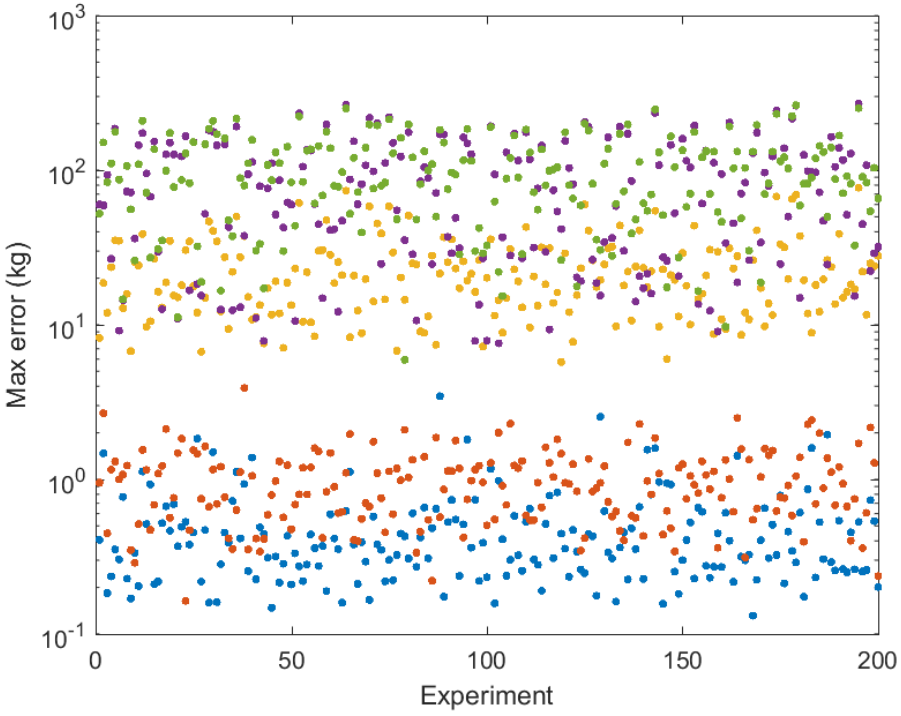
Figure 6.5: Net_2 training graph

As it's possible to see from the training graph, no overfit occurs during the training phase. The best performance reach during training is 0.0999 Kg.

Chapter 7

Results

Five virtual sensors have been obtained by the design phase. In order to test and validate them, a subsystem containing virtual sensors has been created into simulator and 200 simulations have been performed. The use of simulations is an efficient method to test and validate virtual sensors. Through the use of simulations, it's possible to evaluate the evolution of estimations and compare virtual sensors between each other.
Virtual sensors with open loop structures are able to provide estimates immediately, while estimators which use feedbacks require a convergence time. In order to compare both typologies of estimators, simulations are analysed after some time steps so that closed loop observers overcome the convergence time.
Analysing the last estimation samples of every experiment used to test virtual sensors, it's possible to have a general overview about the performance of estimators.



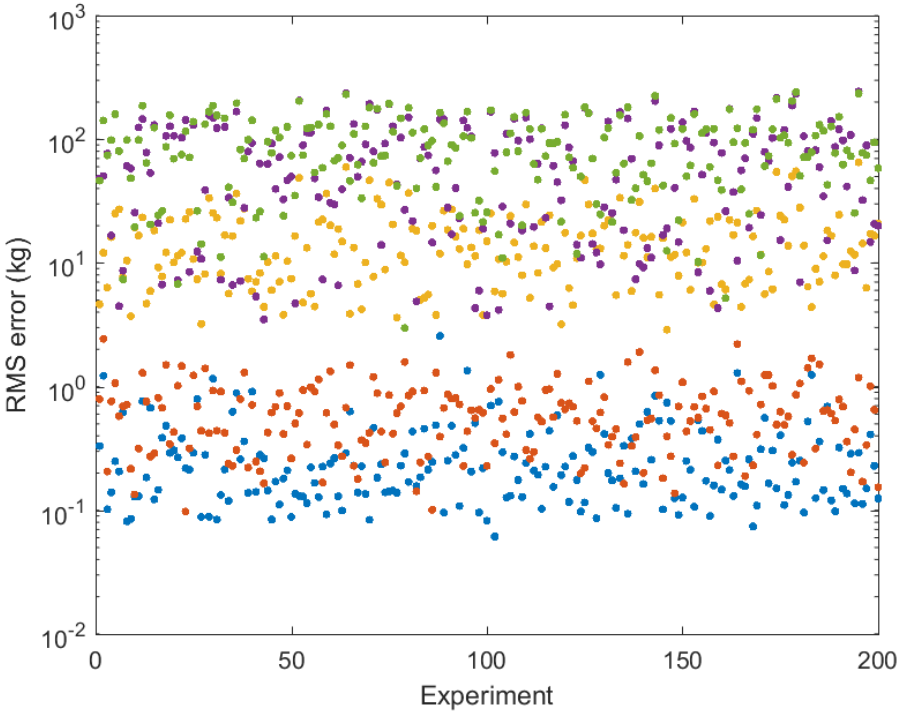


Figure 7.0: Errors evaluations discarding the first 81 steps of every experiment
Legend: Net_1(●), Net_2(●), EKF_{ideal} (●), $EKF_{1\%}$(●), $EKF_{3\%}$(●).

	Mean of MAX errors (kg)	Mean of RMS errors (kg)
Net_1	0.4998	0.296
Net_2	0.998	0.656
EKF_{ideal}	22.978	15.845
$EKF_{1\%}$	86.954	72.141
$EKF_{3\%}$	100.485	89.880

Table 7.1: Mean values of MAX and RMS errors (the first 81 steps of every experiment are discarded)

	Standard deviation of MAX errors (kg)	Standard deviation of RMS errors (kg)
Net_1	0.425	0.293
Net_2	0.545	0.415
EKF_{ideal}	13.952	11.439
$EKF_{1\%}$	65.614	58.665
$EKF_{3\%}$	58.683	54.728

Table 7.2: Standard deviations of MAX and RMS errors (the first 81 steps of every experiment are discarded)

Analysing the graphs and tables above, it's possible to observe that DVSs estimate much better than classical observers. Probably, the worst behaviour of classical observers should be attributed to the high non-linearity of the system model, which doesn't allow the correct linearisation by means of Jacobian calculation.
Analysing the tables above, it's also possible to observe better estimates provided by Net_1 in comparison to Net_2.
Robustness of both DVSs may be explored analysing the graphs in figure 7.0. Analysing these graphs, it's possible to observe no differences in terms of robustness among Net_1 and Net_2.
From the observations written above, I may affirm Net_1 is the most recommended observer for payload estimation because it allows more precision and less convergence time than other estimators.

Following are represented the most relevant simulations.

7.0.1 Simulation n°1

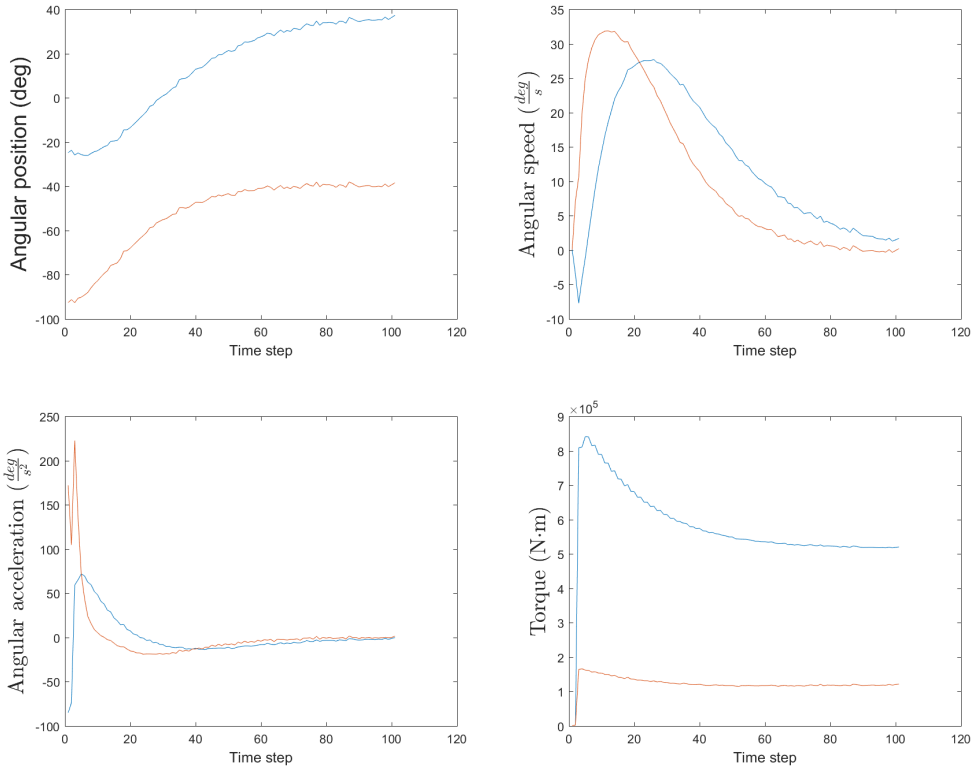


Figure 7.1: Measurements at joints (simulation n°1)
Legend: Lower joint measurements (—), Upper joint measurements (—)

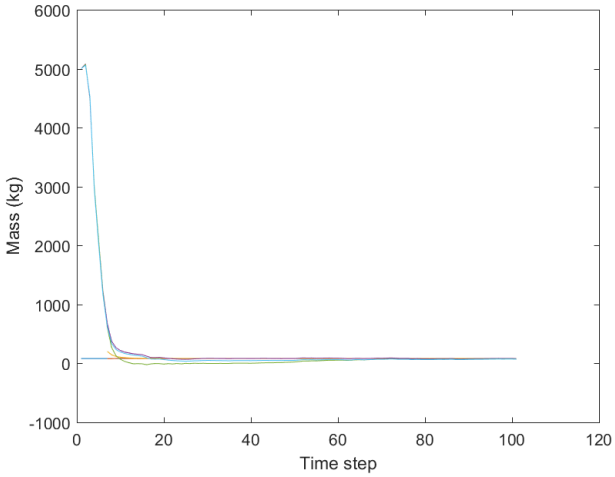


Figure 7.2: Payload estimations (simulation n°1)
Legend: Payload (—), Net_1 estimation (—), Net_2 estimation (—), EKF_{ideal} estimation(—), $EKF_{1‰}$ estimation (—), $EKF_{3‰}$ estimation (—).

7.0.2 Simulation n°2

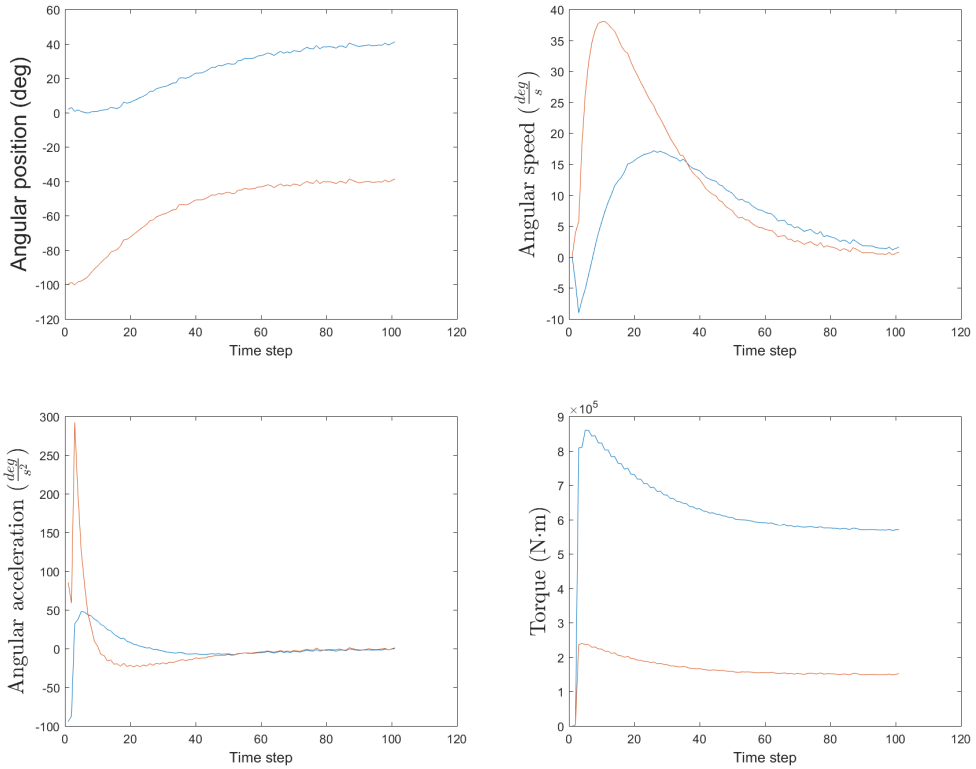


Figure 7.3: Measurements at joints (simulation n°2)
Legend: Lower joint measurements (—), Upper joint measurements (—)

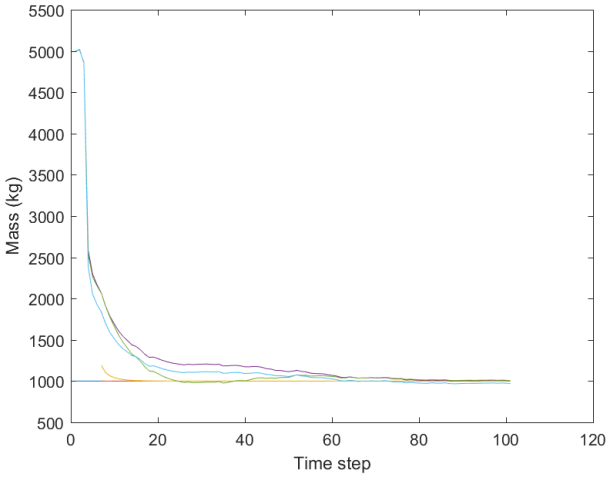


Figure 7.4: Payload estimations (simulation n°2)
Legend: Payload (—), Net_1 estimation (—), Net_2 estimation (—), EKF_{ideal} estimation(—), $EKF_{1\%}$ estimation (—), $EKF_{3\%}$ estimation (—).

7.0.3 Simulation n°3

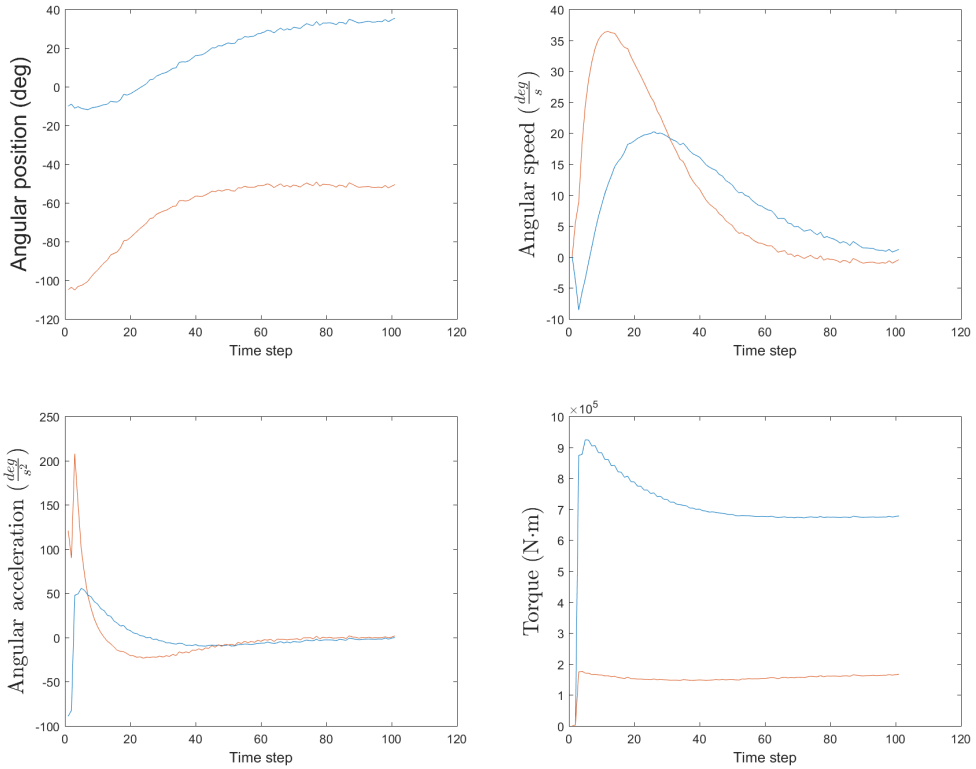


Figure 7.5: Measurements at joints (simulation n°3)
Legend: Lower joint measurements (—), Upper joint measurements (—)

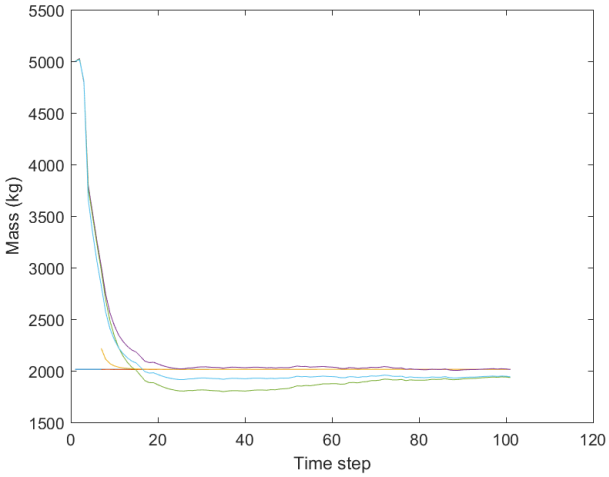


Figure 7.6: Payload estimations (simulation n°3)
Legend: Payload (—), Net_1 estimation (—), Net_2 estimation (—), EKF_{ideal} estimation(—), $EKF_{1\%}$ estimation (—), $EKF_{3\%}$ estimation (—).

7.0.4 Simulation n°4

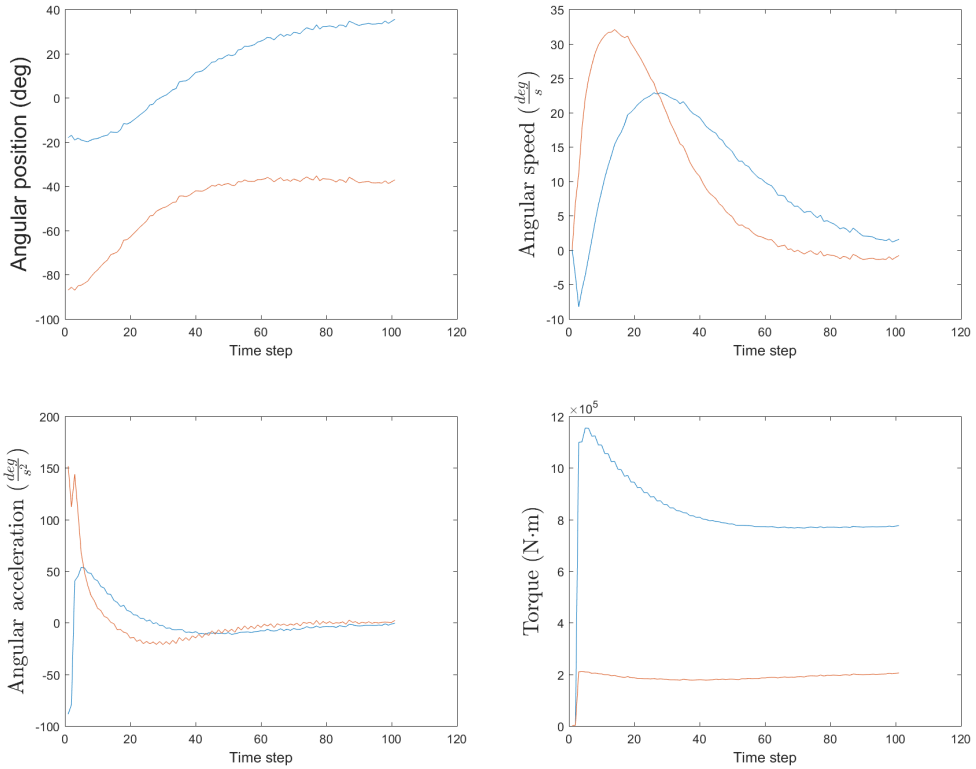


Figure 7.7: Measurements at joints (simulation n°4)
Legend: Lower joint measurements (—), Upper joint measurements (—)

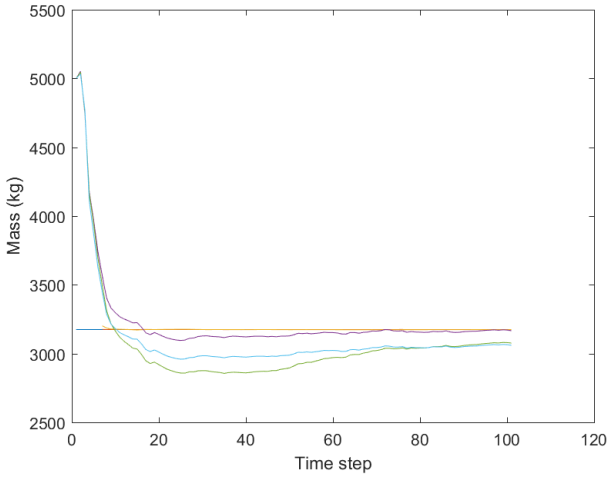


Figure 7.8: Payload estimations (simulation n°4)
Legend: Payload (—), Net_1 estimation (—), Net_2 estimation (—), EKF_{ideal} estimation(—), $EKF_{1‰}$ estimation (—), $EKF_{3‰}$ estimation (—).

7.0.5 Simulation n°5

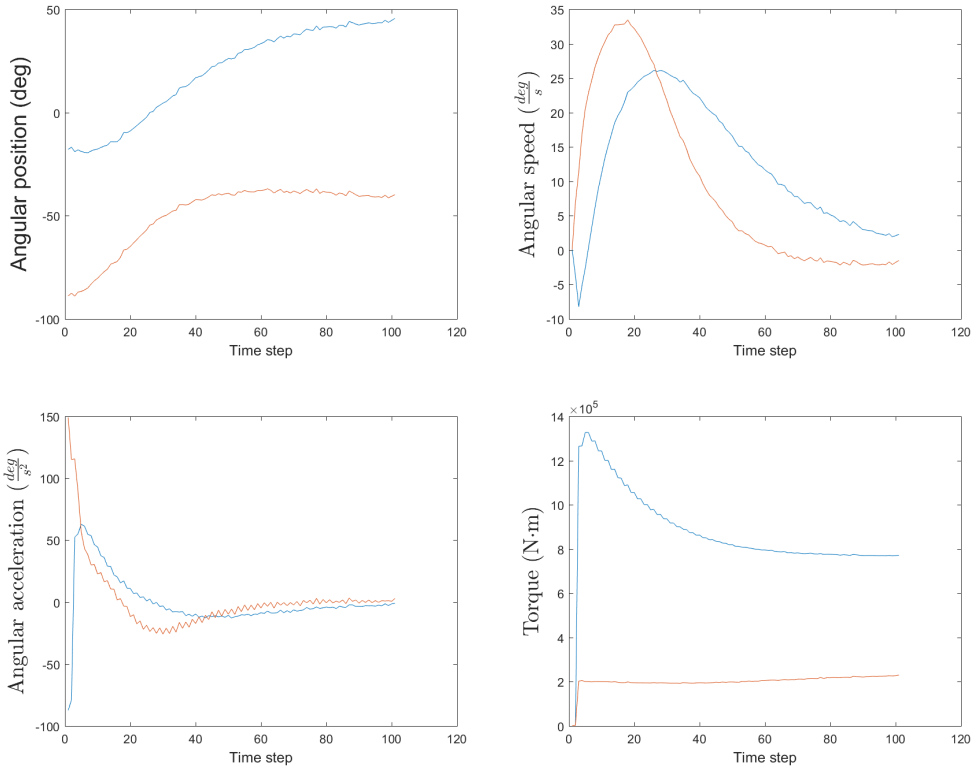


Figure 7.9: Measurements at joints (simulation n°5)
Legend: Lower joint measurements (—), Upper joint measurements (—)

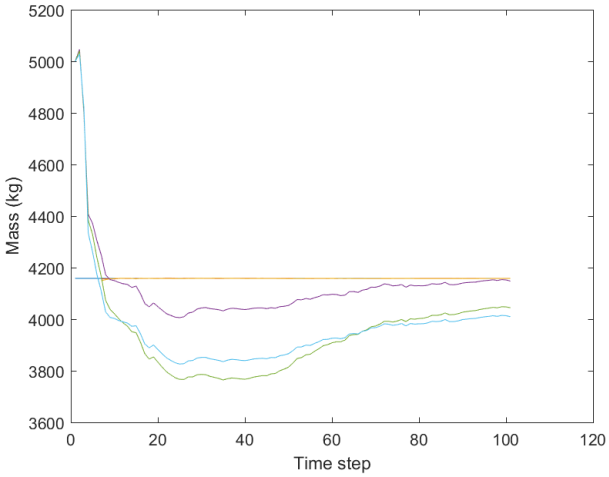


Figure 7.10: Payload estimations (simulation n°5)
Legend: Payload (—), Net_1 estimation (—), Net_2 estimation (—), EKF_{ideal} estimation(—), $EKF_{1\%}$ estimation (—), $EKF_{3\%}$ estimation (—).

7.0.6 Simulation n°6

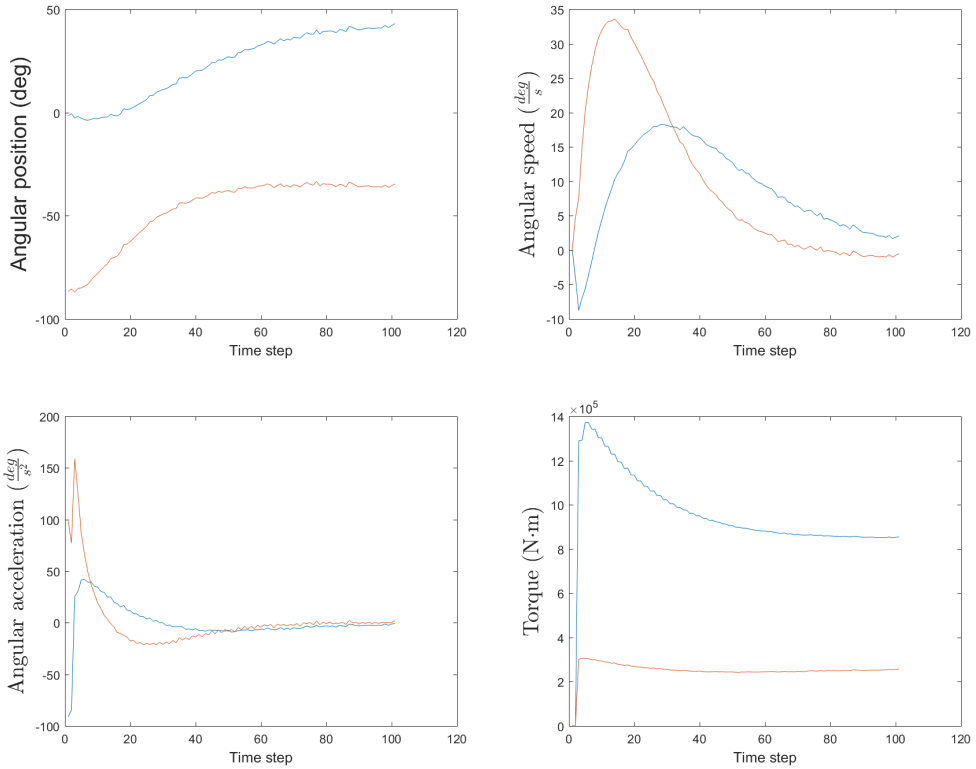


Figure 7.11: Measurements at joints (simulation n°6)
Legend: Lower joint measurements (—), Upper joint measurements (—)

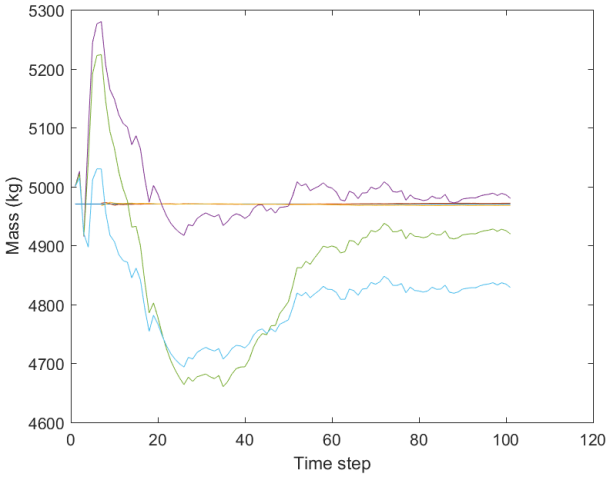


Figure 7.12: Payload estimations (simulation n°6)
Legend: Payload (—), Net_1 estimation (—), Net_2 estimation (—), EKF_{ideal} estimation(—), $EKF_{1\%}$ estimation (—), $EKF_{3\%}$ estimation (—).

Chapter 8

Conclusions

Payload evaluation in excavators is a real necessity. Unfortunately, payload is impossible to be directly measured by sensors, so mathematical approaches are required to estimate it from other measures.
Two strategies of estimation may be adopted for this purpose. The first strategy is called "Quasi-static estimation" and provides the estimates when all excavator arms are moving at low constant angular speed. The second strategy provides the estimates at any motion condition of arms and is called "Dynamic estimation". Quasi-static estimators are a consolidated reality and are implemented in numerous excavator models. Instead, the dynamic payload estimation is an opened problem since frictions and inertias complicate the payload estimation by not allowing the required stability of estimates.
The finality of this thesis is to develop an efficient dynamic payload estimator from simulated set of data, keeping in consideration also real problems like noises, uncertain knowledge of the model and economical costs.

In previous chapters, a simulator has been implemented (chapter 5) and five virtual sensors have been designed through several methodologies (chapter 6). After the design phase, the virtual sensors designed have been tested through simulations. The results of simulations are shown in chapter 7.

The results of simulations demonstrate the performance and the high quality of estimates provided by Net_1.
Net_1 overcomes the minimum required precision in industry (1% of accuracy), since it estimates with a mean value about 0.01% of MAX error and 0.006% of RMS error. Percentages are referred to the maximum permitted payload (5000 kg).
In addition, Net_1 is easy to be adapt to any model of two-arms excavators, because it doesn't require the study of any mathematical model at the design phase. Due to this appreciate quality, Net_1 could also be implemented on external units sold as aftermarket parts and trained on field. At the best of author's knowledge, no payload observers with these qualities are available in the literature, so the results obtained from this thesis may be considered a useful contribute to the study of payload estimators in two-arms excavators.

Unfortunately, I have to conclude my thesis here. In the future it would be interesting to apply these estimators' laws on real field, translating mathematical equations in embedded

code, and implement those on a real excavator ECU. I'm pretty sure virtual sensors would work in reality because the simulator has been carefully designed, but I'm also aware that the simulator approximations are many compared with reality.

I really hope my work will be of help to someone.

Appendices

Appendix A

Matlab Codes

A.1 Model.mat

```
%% Excavator model

clc
clear all
close all

syms t real

syms q1(t) real
syms q2(t) real

syms dq1(t) real
syms dq2(t) real

syms ddq1(t) real
syms ddq2(t) real

beta=sym('beta', [2 1], 'real');      %Friction coefficient

syms Mload real;                      %Payload

l=sym('l', [2 1], 'real');            %Lenght of the arms
M=sym('M', [2 1], 'real');            %Mass of the arms
J=sym('J', [2 1], 'real');            %Inertia of  the arms respect z

x=sym('x', [4 1], 'real');
dx=sym('dx', [4 1], 'real');

T1=simplify([[cos(q1),-sin(q1),0;sin(q1),cos(q1),0;0,0,1],[0;0;0]; ...
    zeros(1,3),1]*[eye(3),[l(1)/2;0;0];zeros(1,3),1])
T2=simplify(T1*[eye(3),[l(1)/2;0;0];zeros(1,3),1]*[[cos(q2),-sin(q2),0;...
    sin(q2),cos(q2),0;0,0,1],[0;0;0];zeros(1,3),1]*[eye(3),[l(2)/2;0;0];...
    zeros(1,3),1])
T3=simplify(T2*[eye(3),[l(2)/2;0;0];zeros(1,3),1])
```

```
pose1=formula(T1);
v1=simplify([diff(pose1(1,4),t)
             diff(pose1(2,4),t)
             diff(pose1(3,4),t)])
v1mq=simplify(v1(1)^2+v1(2)^2+v1(3)^2)

pose2=formula(T2);
v2=simplify([diff(pose2(1,4),t)
             diff(pose2(2,4),t)
             diff(pose2(3,4),t)])
v2mq=simplify(v2(1)^2+v2(2)^2+v2(3)^2)


pose3=formula(T3);
v3=simplify([diff(pose3(1,4),t)
             diff(pose3(2,4),t)
             diff(pose3(3,4),t)])
v3mq=simplify(v3(1)^2+v3(2)^2+v3(3)^2)


Kin(1)=0.5*(M(1)*v1mq+J(1)*diff(q1(t), t)^2);
Kin(2)=0.5*(M(2)*v2mq+J(2)*(diff(q1(t), t)+diff(q2(t), t))^2);
Kin(3)=0.5*(Mload*v3mq);

Ktot=simplify(sum(Kin)) %Total kinetic co-energy


g=[0;-9.81;0];
Ptot=simplify(-M(1)*g'*pose1(1:3,4)-M(2)*g'*pose2(1:3,4)-...
    (Mload)*g'*pose3(1:3,4)) %Total potential energy
Dtot=simplify(0.5*(beta(1)*(diff(q1(t), t))^2+...
    beta(2)*(diff(q2(t), t))^2))


Ktot=subs(Ktot,diff(q1(t), t),dq1);
Ktot=subs(Ktot,diff(q2(t), t),dq2);

Dtot=subs(Dtot,diff(q1(t), t),dq1);
Dtot=subs(Dtot,diff(q2(t), t),dq2);

equations(1,1)=diff(functionalDerivative(Ktot,dq1),t)-...
    functionalDerivative(Ktot,q1)+functionalDerivative(Ptot,q1)+...
    functionalDerivative(Dtot,dq1);
equations(2,1)=diff(functionalDerivative(Ktot,dq2),t)-...
    functionalDerivative(Ktot,q2)+functionalDerivative(Ptot,q2)+...
    functionalDerivative(Dtot,dq2);

equations=formula(simplify(equations));

equations=subs(equations,diff(dq1(t), t),dx(3));
equations=subs(equations,diff(dq2(t), t),dx(4));

equations=subs(equations,q1,x(1));
equations=subs(equations,q2,x(2));
```

```
equations=subs(equations,dq1,x(3));
equations=subs(equations,dq2,x(4));

for i=1:2
stateEq(i,1)=dx(i)-x(2+i);
stateEq(2+i,1)=equations(i,1);
end

stateEq

u=sym('u',[2 1],'real');
[A,b] = equationsToMatrix(stateEq==[0;0;u], dx) % A*dx=b
sol=inv(A)*b;
sol=simplify(sol);

[g,f] = equationsToMatrix(sol==zeros(4,1), u) % ff=f+g*u
sprintf('Equation: ff=f+g*u')
f=-f;
if simplify(sol-(f+g*u))==zeros(4,1)
    ff=f+g*u;
    save('model_variables.mat','ff','f','g')
    sprintf('Matrix ff,f,g saved in "model_variables.mat"')
else
    sprintf('It is not possible to save the model as matrix ff,f,g')
end
matlabFunction(sol,'file','ExcavatorModel');
```

A.2 Simulation.mat

```
clear all
close all
clc

x=sym('x', [4 1], 'real');
u=sym('u', [2 1], 'real');

Tf=5;
Ts=0.05;

l1=6.55;
l2=3.0;

M1=4600;
M2=2410;
Mbucket=2900;

beta1=3e5;
beta2=beta1;

J1=M1*(l1^2)/12;
J2=M2*(l2^2)/12;

U_noiseVar=1e6;
Acc_noiseVar=1e-4;
Vel_noiseVar=1e-5;
Pos_noiseVar=1e-4;

open('mymodel_nmpc.slx')

for iii=1:2000
r=(pi/180)*[40+10*(2*rand-1);-45+10*(2*rand-1);zeros(2,1)];
x0=(pi/180)*[-10+20*(2*rand-1);-90+20*(2*rand-1);zeros(2,1)];

Mload=5000*rand+Mbucket;

f=ExcavatorModel(J1,J2,M1,M2,Mload,beta1,beta2,l1,l2,u(1),u(2), ...
    x(1),x(2),x(3),x(4))

% om: optimization model
om.Ts=0.1; %sampling time at which the states are measured
om.Tp=2; %prediction horizon
om.Q=zeros(4);
om.R=0*eye(2);
om.P=diag([ones(1,2)*2e11,ones(1,2)*1e3]);
om.lb=[];
om.ub=[];
om.nlc=0;
om=nmpc_design(f,om);

sim('mymodel_nmpc.slx')
```

```
exp = size(dataset,2)+1;
init = 1;
stop =size(Lower_Arm_Kin,1);
step = 1;

dataset{exp}.input = [Lower_Arm_Kin(init:step:stop,:) ...
     Upper_Arm_Kin(init:step:stop,:) Torque(init:step:stop,:)];

dataset{exp}.output = (Mload-Mbucket)*ones(length(time(init:step:stop,:)),1);
dataset{exp}.time = time(init:step:stop,:);
dataset{exp}.input_names = {'Pos_LA' 'Vel_LA' 'Accel_LA' 'Pos_UA'...
    'Vel_UA' 'Accel_UA' 'Torque_LA' 'Torque_uA'};
dataset{exp}.input_unit = {'deg' 'deg/s' 'deg/s^2' 'deg' 'deg/s'...
    'deg/s^2' 'Nm' 'Nm'};
dataset{exp}.name = 'dataset_excavator';
dataset{exp}.note = '...';

end

save('dataset_excavator','dataset')
```

A.3 NNdesign.mat

```
clc
clear all
close all

load dataset_excavator    %Dataset used to train the net

N='[10 10 10]';          % Each element of the vector is a layer,
                         % the value of the element is the number of neurons
                         % in the layer


Input='[1 2 4 5 7 8]';      %Neural network's input

Sim='[101:151]';       %Experiments used for the validation phase
Train='[1:100]';       %Experiments used for the training phase

% Merge of data used for the training phase.
% Data should be merged to be recognized by
% the training program as sequential inputs.
% If data are passed as vector, the training
% program considers them as concurrent inputs.

sequence=str2num(Train);

for i=1:size(sequence,2)

a=0;
for iii=str2num(Input)
a=a+1;
inputArray(a,:)=dataset{sequence(i)}.input(:,iii);

end

nnn=0;
for n=1:size(dataset{sequence(i)}.output,1)
nnn=nnn+1;
uuu{nnn}(:,1) = inputArray(:,n);
ttt{nnn} = (dataset{sequence(i)}.output(n))';
end

if(i==1)
inT=uuu;
outT=ttt;
else
inT=catsamples(uuu,inT);
outT=catsamples(ttt,outT);
end
end


% Feedforward_neural_network ==> NNtype=1;
```

```
% Timed_delay_network ==> NNtype=2;
% TDNN_network ==> NNtype=3;
% Narx_network ==> NNtype=4;
% Layer_recurrent_network ==> NNtype=5;

NNtype=2;     %Defines the neural network type

pool=parpool  %Starts the parallel calculation


%% Feedforward Neural network
if NNtype==1

net = feedforwardnet(str2num(N));
net.trainFcn = 'trainbr';
net.trainParam.epochs = 1e10

% net.outputs{3}.processFcns{2}='mapstd';
% net.inputs{1}.processFcns{2}='mapstd';


[In,InI,Ai,Out] = preparets(net,inT,outT);
net=init(net)
net.trainParam.lr =Lr;
[net,tr]=train(net,inT,outT,'useParallel','yes','showResources','yes');

description=strcat('Feedforward Neural network - Network: ',N,...
    ' -Project: ',Train,' - Sim: ',Sim,' - Input: ',Input);
end


% Timed delay network
if NNtype==2
R='0:5';

net=timedelaynet(str2num(R),str2num(N));

net.trainFcn = 'trainbr';
net.trainParam.epochs = 1e10;

% net.outputs{3}.processFcns{2}='mapstd';
% net.inputs{1}.processFcns{2}='mapstd';

[In,InI,Ai,Out] = preparets(net,inT,outT);
net.trainParam.lr =Lr;
[net,tr]=train(net,In,Out,InI,'useParallel','yes','showResources','yes');

description=strcat('Timed delay network - Network: ',N,' - R: ',R,...
    ' -Project: ',Train, '- Sim: ',Sim,' - Input: ',Input);
end
% TDNN network
if NNtype==3
R='0:3';
R2='0:3';      %Regressors of the layer 2
R3='0:3';      %Regressors of the layer 3
```

```
net=distdelaynet({str2num(R),str2num(R2),str2num(R3)},str2num(N));

net.trainFcn = 'trainbr';
net.trainParam.epochs = 1e10;

[In,InI,Ai,Out] = preparets(net,inT,outT);
[net,tr]=train(net,In,Out,InI,'useParallel','yes','showResources','yes');

description=strcat('TDNN network - Network: ',N,' - R: ',R,' - R2: ',R2,...
    ' -Project: ',Train,' - Sim: ',Sim,' - Input: ',Input);
end

% Narx network
if NNtype==4
R1='0:5';
Rf='1';    %Regressors of the feedback
net=narxnet(str2num(R1),str2num(Rf),str2num(N));

net.trainFcn = 'trainbr';
net.trainParam.epochs = 1e10;

net=closeloop(net);

[In,InI,Ai,Out] = preparets(net,inT,{},outT);
[net,tr]=train(net,In,Out,InI,'useParallel','yes','showResources','yes');

description=strcat('Narx network - Network: ',N,' - R1: ',R1,' - Rf: ',...
    Rf,' -Project: ',Train,' - Sim: ',Sim,' - Input: ',Input);
end
% Layer-Recurrent Network
if NNtype==5
R='1:2';
net=layrecnet(str2num(R),str2num(N));

net.trainFcn = 'trainbr';
net.trainParam.epochs = 1e10;

[In,InI,Ai,Out] = preparets(net,inT,outT);
[net,tr]=train(net,In,Out,InI,'useParallel','yes','showResources','yes');

description=strcat('Layer Recurrent Network - Network: ',N,' - R: ',R,...
    ' -Project: ',Train,' - Sim: ',Sim,' - Input: ',Input);
end


%% Network's simulation
a=0;
for i=str2num(Sim)
aaa=0;
for iii=str2num(Input)
aaa=aaa+1;
inputArray(aaa,:)=dataset{i}.input(:,iii);
end
```

```
for n=1:size(dataset{i}.output,1)
inS{n}(:,1) = inputArray(:,n);
outS{n} = (dataset{i}.output(n))';
end


 if NNtype==4
     [In,InI,Ai,Out] = preparets(net,inS,{},outS);
 else
     [In,InI,Ai,Out] = preparets(net,inS,outS);
 end

 a=a+1;
 sim = net(In,InI,Ai,'useParallel','yes','showResources','yes')

 TestSim{NNtype}.simulation{a}.desired=cell2mat(Out);
 TestSim{NNtype}.simulation{a}.estimated=cell2mat(sim);
 TestSim{NNtype}.simulation{a}.err = cell2mat(sim)-cell2mat(Out);
 TestSim{NNtype}.simulation{a}.CI = prctile(...
     abs(TestSim{NNtype}.simulation{a}.err),[90 95 99]);
 TestSim{NNtype}.simulation{a}.N_test=i;
 TestSim{NNtype}.net=net;
 TestSim{NNtype}.description=description;
 TestSim{NNtype}.results(:,a) = TestSim{NNtype}.simulation{a}.CI;
 TestSim{NNtype}.MaxErr(1,a)=max(abs(TestSim{NNtype}.simulation{a}.err));
end
delete(pool)
```

A.4 myStateTransitionFcn.mat

```
function xk = myStateTransitionFcn(x,u)
l1=6.55;
l2=3;

M1=4600;
M2=2410;
Mbucket=2900;
Mload=abs(x(5))+Mbucket;

beta1=3e5;
beta2=beta1;

J1=M1*(l1^2)/12;
J2=M2*(l2^2)/12;

Ts=0.05;
xk=zeros(5,1);
dx=ExcavatorModel(J1,J2,M1,M2,Mload,beta1,beta2,l1,l2,u(1),u(2), ...
    x(1),x(2),x(3),x(4));

for i=1:4
xk(i)=x(i)+Ts*dx(i);
end
xk(5)=Mload-Mbucket;
end
```

A.5 myMeasurementFcn.mat

```
function y = myMeasurementFcn(x)
y=x(1:4);
end
```

Bibliography

Articles and Books

[1] Levin A. and Narendra K. "Control of nonlinear dynamical systems using neural networks. II. observability, identification, and control". In: *IEEE Trans. Neural Networks* NN-7.1 (1996), pp. 477–482.

[2] B. Anderson and J. Moore. "Optimal Filtering". In: Englewood Cliffs, NJ, 1979.

[4] Nureddin Bennett, Ashwin Walawalkar, and Christian Schindler. "Payload Estimation in Excavators: Model-Based Evaluation of Current Payload Estimation Systems". In: (2014).

[7] J.L. Elman. "Finding structure in time". In: *Cognitive Science* 14 (1990), pp. 179–211.

[8] G. Evensen. "Data Assimilation: The Ensamble Kalman Filter". In: *New York: Springer* (2007).

[9] Sharon Furtak. "Neurons". In: *Noba textbook series: Psychology*. 2018. URL: http://www.nobaproject.com/.

[10] J. C. Geromel and M. C. Oliveira. "$\mathcal{H}_2$ and $\mathcal{H}_\infty$ robust filtering for convex bounded uncertain systems". In: *IEEE Trans. Autom. Control* 46.1 (Jan. 2001), pp. 100–107.

[11] Jahmy Hindman, Richard Burton, and Greg Schoenau. "An Artificial Neural Network Approach to Payload Estimation in Four Wheel Drive Loaders". In: *ASME International Mechanical Engineering Congress and Exposition, Proceedings*. Vol. 9. Jan. 2007.

[14] A. Jazwinski. "Stochastic Processes and Filtering Theory". In: *Academic*. NewYork, 1970.

[15] S. Julier, J. Uhlmann, and H. Durrant-Whyte. "A new method for the nonlinear transformation of mean and covariances in filters and estimators". In: *IEEE Trans. Autom. Control* 45 (2002), pp. 477–482.

[16] R. Kalman and R. Bucy. "New results in linear filtering and prediction theory". In: *J . Basic Eng.* 83D (1961). Ed. by Trans. ASME, pp. 95–108.

[17] A. Kolmogorov. "Stationary sequences in hilbert space". In: *Bull. Moscow Univ.* 2 (1941), pp. 1–40.

[21] Warren McCulloch and Pitts Walter. "A Logical Calculus of Ideas Immanent in Nervous Activity". In: *Bulletin of Mathematical Biophysics* 5 (1943), pp. 115–133.

[23] K. Nagpal, J. Abedor, and K. Poolla. "A linear matrix inequality approach to peak-to-peak gain minimization". In: *IEEE. Trans. Autom. Control* 41.1 (1996), pp. 43–48.

[24] K. Nagpal and P. Khargonekar. "Filtering and smoothing in an $\mathcal{H}_\infty$ setting". In: *IEEE. Trans. Autom. Control* 36 (1991), pp. 152–166.

[25] O. Nelles. "Nonlinear System Identification: From Classical Approaches to Neural Networks and Fuzzy Models". In: *Springer-Verlag*. Berlin, Germany, 2001.

[26] Michael Nielsen. *Neural Networks and Deep Learning*. 2017. URL: `http://neuralnetworksanddeeplearning.com/index.html`.

[29] Carlo Novara, Mario Milanese, and Fredy Ruiz. "Direct Filtering: A New Approach to Optimal Filter Design for Nonlinear Systems". In: *IEEE TRANSACTIONS ON AUTOMATIC CONTROL* (2013).

[30] Carlo Novara et al. "The filter design from data (FD2) problem: parametric-statistical approach". In: *INTERNATIONAL JOURNAL OF ROBUST AND NONLINEAR CONTROL* (2011).

[33] B. Ristic, S. Arulampalam, and N. Gordon. "Beyond Kalman Filter: Particle Filters and Tracking Applications". In: *Artech House*. Boston, MA, 2004.

[34] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. "Learning representations by back-propagating errors". In: *Nature* 323 (1986), pp. 533–536.

[35] B. Silvy-Leligois and J. Giroud. *Wägeverfahren und Hubfahrzeug zur Durchführung des Verfahrens*. EP Patent App. EP19,950,104,951. Oct. 1996. URL: `https://encrypted.google.com/patents/EP0736752A1?cl=ar`.

[36] K. Sun and Packardv A. "Robust and filters for uncertain LFT systems". In: *IEEE Trans. Autom. Control* 50.5 (May 2005), pp. 715–720.

[39] P. G. Voulgaris. "On optimal to filtering". In: *Automatica* 31.3 (1995), pp. 489–495.

[40] Ashwin Walawalkar et al. "A Method for Payload Estimation in Excavators". In: (2016).

Lecture notes

[27] Carlo Novara. *Automatic control*. 2016–2017.

[28] Carlo Novara. *Nonlinear control and aerospace applications*. 2016–2017.

[37] Michele Taragna. *Controlli automatici*. 2016–2017.

[38] Massimo Violante. *Operating systems for embedded systems*. 2016–2017.

Videos

[20] Matlab. *Understanding Kalman Filters*. 2017. URL: https://it.mathworks.com/videos/series/understanding-kalman-filters.html.

Websites

[12] Physics Insights. *LagrangiaMechanics*. 2006. URL: http://www.physicsinsights.org/lagrange_1.html#eqn-4.

[13] National Instruments. *Controller Area Network (CAN) Overview*. URL: http://www.ni.com/white-paper/2732/en/.

[18] Bilal Malikuet. *Controller area network protocol, features history and working*. 2017. URL: http://microcontrollerslab.com/can-protocol-history-features/.

[19] Matlab. *Extended Kalman Filter*. URL: https://it.mathworks.com/help/control/ref/ekf_block.html.

[22] Modelway. *DVS®*. URL: http://modelway.it/technologies/dvs/.

[31] Alberto Quesada. *5 algorithms to train a neural network*. 2017. URL: https://www.neuraldesigner.com/blog/5_algorithms_to_train_a_neural_network.

[32] Renesas. *In-Vehicle Networking Solutions*. URL: https://www.renesas.com/en-in/media/solutions/automotive/technology/networking/lan-clickablemap.jpg.

Manuals

[3] Mark Hudson Beale, Martin T. Hagan, and Howard B. Demuth. *Neural Network Toolbox™ User's Guide*. 2016.

[5] Caterpillar. *345C L Hydraulic Excavator*. 2007.

[6] Doosan. *Excavator attachments*. 2015.