Politecnico di Torino

Department of Control and Computer Engineering

Master of Science in Mechatronic Engineering

Virtual Sensors for Human Safety Monitoring in Factory 4.0 Applications

Academic Supervisor:
Prof. Carlo Novara
Company Supervisors:
Ilario Gerlero
Mario Bonansone

Candidate:
Alessio De Sangro
matr. 244372

Abstract

The safety of operators in Factories is a key problem since the Industrial Revolution. The growth of the technology let the development of new safety monitoring systems, often integrated with the industrial machinery itself. However, with the Factory 4.0 machines became gradually more autonomous and so the role of the worker has shifted towards the maintenance and the malfunctions management of these ones. The main goal of the future society should be to understand the meaning of this new interaction and to exploit the technology not only for improving performances, but also for protect the safety and the security of the operator. This Thesis work, developed in collaboration with Modelway s.r.l, starts from these considerations and aims to design a virtual sensor for human state monitoring during work activities. In includes two work packages (WP07, WP09) inside the Disloman project in which two different virtual sensors have been implemented. For both of them an Artificial Intelligence algorithm developed by Modelway and called DVS(Direct Virtual Sensor) has been used. The first one is able to detect the fall of the operator starting from acceleration measurements. They are acquired at a sampling time of $50ms$ by a wearable sensor placed in a bracelet in order to detect the wrist movement. The DVS/MDD (Man Down Detection) reads these variables and generates in output the estimation of the current state of the operator; after this a state machine has been implemented in order to elaborate the output of the virtual sensor, to provide the transitions between status and to return a feedback of the current one. In this project another DVS called DVS/R able to reset a wrong estimation of the DVS/MDD has been developed. The control unit chosen to manage these procedures is the $RaspberryPi3B+$.
The second virtual sensor is able to track the arm position of the operator that works near a mechanical press (at Honestamp Factory). In particular, it stops the action of the machinery each time the hand of the worker is inside the dangerous area. For this purpose two wearable sensors are used (placed in a bracelet and on top of the arm) which send measurements of accelerometer, gyroscope and magnetometer at a control unit ($RaspberryPi3B+$). Also in this case it analyses these signals and performs the Artificial Intelligence algorithm for the Arm Tracking estimation. Its output is received by another state machine that provides the transition of the states.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The safety of operators in Factories is a key problem since the Industrial Revolution. The growth of the technology let the development of new safety monitoring systems, often integrated with the industrial machinery itself. However, with the Factory 4.0, machines became gradually more autonomous and so the role of the worker has shifted towards the maintenance and the malfunctions management of these ones. The main problem in this scenario is to understand the meaning of this new interaction and to exploit the technology not only for improving performances, but also for protect the safety and the security of the operator. This Thesis work, developed in collaboration with Modelway s.r.l, starts from these considerations and aims to design a virtual sensor for human state monitoring during work activities. It includes two work packages (WP07, WP09) inside the Disloman project in which two different virtual sensors have been implemented. For both of them an Artificial Intelligence algorithm developed by Modelway and called DVS(Direct Virtual Sensor) has been used. It design a virtual sensor able to describe the dynamic behaviour(input/output) of systems, starting from data directly measured.
In the following paragraphs an overview of the historical context, Disloman project's scope and Machine Learning algorithms is proposed.

1.1 Historical Context-Industry 4.0

The term Industry 4.0 indicates a trend of industrial automation to integrates some new production technologies, to improve working conditions, create new business models and increase the productivity and production quality of plants. Moreover, its meaning coincides with the 4th Industrial Revolution. [1]
The previous three Industrial Revolutions are identified to:

- **1784** with the introduction of the steam engine and consequently with the exploitation of the power of water and steam to mechanize production;

- **1870** with the start of mass production through the widespread use of electricity, the advent of the internal combustion engine and the increase in the use of petroleum as a new energy source;

- **1970** with the birth of information technology, which increase automation levels by making use of electronic systems and IT (Information Technology).

The birth of the fourth Industrial Revolution is not yet established, probably because it is still in progress. The topic was discussed for the first time at the World Economic Forum 2016(Switzerland) and called "Mastering the Fourth Industrial Revolution".[1]
It is composed of machines completely interconnected, which communicate with each other and carry out self-diagnostics and preventive maintenance. In particular, as described into a report prepared by GE Digital with the independent research company Vanson Bourne, machinery maintenance will be performed by the machines themselves, thanks to the Internet of thing for example. These advances in technological evolution will bring factories to autonomously predict the degree of production failure, to adopt the best prevention measures and to implement self-repair actions. Furthermore, as explained in Industry 4.0, Robots will work in contact with man and will learn from man naturally. The workflow can be reproduced in a virtual way, therefore before physically preparing it in the workshop, in order to verify its abstract behavior and enhance its performance. The factory will be able to supply itself with energy without waste and at lowest cost, in a few words it will be smart. [1]
Despite this, there is still a crucial topic that should be considered: the role of the operator in these new factories. It is important to understand that the technological development can not follow only the direction of cost reduction and optimization process, but must also take into account the work scenario of actual society, focusing on the management of such of problem. In fact, it is clear that a new type of man-machine interaction must be found before the complete introduction of these new technologies, and a definition of the future role of the operator is mandatory for the collective growth of our community.

1.2 Disloman Project

Disloman project starts from previous consideration and its primary scope is the creation of an ICT platform for the integrated, dynamic and autonomous management of high-level automation production operations aimed to optimizing resources (people, materials, production systems). [2]

The DISLO-MAN platform will be composed of several horizontal hardware and software modules applicable to undifferentiated production environments, and verti-

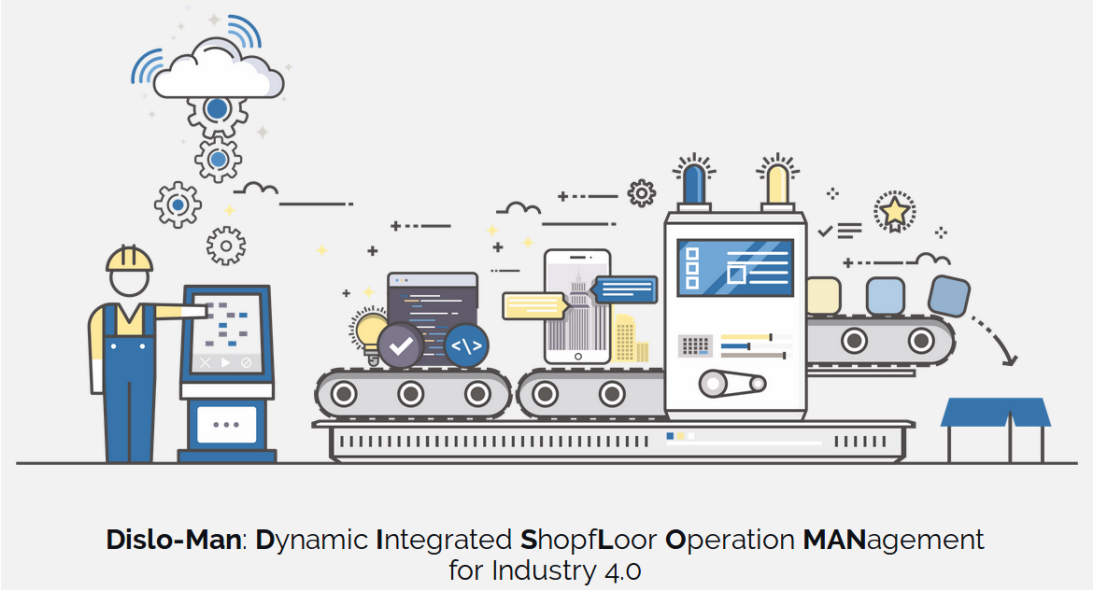

Figure 1.1: Dislo-Man Project [2]

cal modules applied to four high-impact demonstration chains in the area and with different volumes, dimensions and complexity: car and machine tools manufacturing, precision mechanics, pharmaceuticals and food processing. The DISLO-MAN platform is strongly based on Internet-of-Things (IoT) technologies in their various nuances; in particular it will consist of:

- Wearable sensors: wearable terminals (object of development and prototyping) and the like;
- Indoor location systems for operators and automated material handling systems (AGV, trolleys etc.);
- Modules for wireless communication and real-time data collection from human operators, machine tools, conveyor belts, goods in storage and on the move etc .;
- Software systems for Machine Learning/Big Data able to analyze complex scenarios and provide suggestions and alternatives aimed at optimizing human and material resources;

The main features of the DISLO-MAN platform are the following:

- Integration (of production systems) with sensor networks for continuous detection of production parameters installed on machine tools and aimed at optimizing material flows and minimizing downtime;

- Integration with wearables for movement recognition systems for optimizing ergonomics and reducing recovery activities;
- Integration with appropriately sensors for handling systems (AGV or chain belts) where present (vertical car manufacturing);
- Implementation of systems able to improve the energy efficiency of the production environment;
- Implementation of a data analytics system based on a combination of statistical methods, Big Data Analytics, Machine Learning and rule-based semantic reasoning techniques based on the most effective methods for real-time analysis of predictive modeling;
- Realization of protection and safety support systems alongside highly automated production lines;
- Realization of a reference architecture and its implementation in a cloud-based software platform for data acquisition, aggregation and homogenization and dynamic storage of data flows; [2]

1.3 Machine Learning

Machine Learning is a sub-field of artificial intelligence and, as a consequence, it is a category of algorithm that allows software applications to predicting outcomes without being explicitly programmed. But why this is important? There are many engineering reasons as well. For example it is possible to describe a behaviour of a system simply knowing its inputs and outputs, to find correlations and relationship of a big number of data (Data Mining), and to adapt the system model to a change of environment, reducing the need of redesign.[3]
There are different types of Machine Learning:

- **Supervised Learning**: At each input of the system corresponds the respective output in order to allow at the algorithm to find a correlation between them. Successively, it can be replicated with others data of the same statistical distribution of the previous one;
- **Unsupervised Learning**: These types of algorithms receives data input without any information of the desired result. Its main goal is to find a correlation between no labelled inputs;
- **Reinforcement Learning**: In this case, the system (computer, software, algorithm) must interact with a dynamic environment and perform output, also learning from errors. The software learns, for example, to beat an opponent in a game (or drive a vehicle) by concentrating their efforts on performing a certain task, aiming to reach the maximum value of the reward; in other words, the algorithm learns by playing (or driving) and by mistakes made by improving performance precisely based on the results achieved previously. [3]

1.3.1 Features Selection

In the previous paragraph a general overview of the machine learning principles has been done, now it is possible to understand in which way it can be built.
Basically the main goal of these methods is to minimize the loss function (the mean square error between the estimated output and the expected one) and, as a consequence, to find a correlation between inputs and outputs. In this scenario, the input dataset is fundamental for the good implementation of the algorithm, so the first step is to select important features that can describe properly the behaviour of the system. The procedure to transform the raw data into a dataset is called feature engineering.[5]

One-hot Encoding

Some Machine Learning algorithm are able to analyze only numerical feature vectors. If the input dataset is categorical, for example "animals" or "colors" it is possible to transform these types of vectors into numerical ones.

$$red = [1, 0, 0]; yellow = [0, 1, 0]; green = [0, 0, 1] \tag{1.1}$$

It is important to pay attention of the vector size. In particular, with this method the dimensionality of the input is increased.
This approach has been used in the first part of the project, for the design of DVS able to detect the fall of the operator implemented with LSTM neural networks.[5]

Normalization

Normalization is a mathematical process which converts the input range into a standard one, typically included between [-1,1] or [0,1].

$$xn^j = \frac{x^j - min^j}{max^j - min^j} \tag{1.2}$$

Where max and min are the maximum and the minimum value of the feature j in the dataset. This procedure is very useful in the learning phase because, as it standardizes all data, and ensures that inputs will be in the same small range. [5]

1.3.2 Learning Algorithm Selection

Input dataset is ready, now it is important to choose the correct machine learning algorithm. Basically the are four different approaches depending from the problem:

- **Classification**: Data are labeled according to some features given in input, and so labels are predicted during tests. For example in this project an algorithm is design in order to recognize the fall of the operator starting from accelerations. So, in training phase some of them are labeled 1 if they correspond to a fall movement, 0 otherwise. It is included into the Supervised learning with discrete output.

- **Regression**: It describes a behaviour in time of the system, so its output approximates the data mapping function. It is included into the Supervised learning with continuous output.

- **Clustering**: It groups objects with similar characteristics (called Features) into the same class. It is considered an unsupervised algorithm because its scope is to describe the hidden structure of objects and the relationship between them;

- **Dimensionality Reduction**: It reduces the number of features of the object. It is useful because simplifies computational problems.

Figure 1.2 summarizes this description.

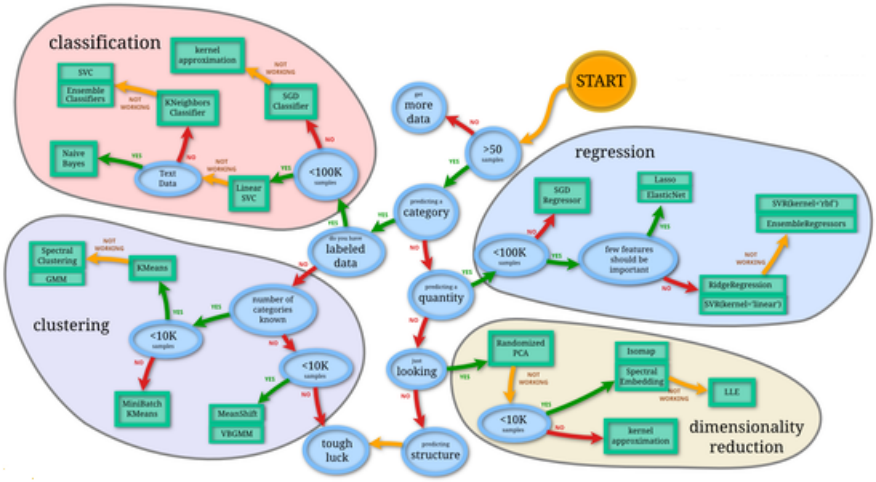

Figure 1.2: Machine Learning Algorithms [5]

The correct methodology inside these areas depends from the scope of the project, for example neural networks and ensemble models can be used if the problem is complex and high estimation accuracy must be reached, on the other hand linear regression and decision trees can be selected for problems that ask a simple but fast solution.
For both projects of this Thesis neural networks are implemented. Since they are described in the following chapter, only a general description of the other methods has been done.

1.3.3 Training

In this phase algorithm learns the correlation between inputs and outputs. It is divided into two steps: Training and Validation. In the first one data are used to fit the model, in the supervised approach for example, it read inputs and their respective labels; in the second one data are used to provide an unbiased evaluation of a fit on the training dataset while tuning model hyperparameters. [5] In the final step, the algorithm is tested with new real data.
So the entire dataset must be divided in training and validation set, the choice of these two sizes is crucial for a good design of the algorithm. Usually, a "k fold cross validation method" is used, in which if the total dataset has length equal to k, $(k-1)/k$ samples are destined in the training dataset, and $1/k$ in the validation one.
This is important because the difference in the performance of these two sets gives an

idea in which way it is possible to improve the accuracy of the algorithm. An example is given by the under and overfitting problem (Figure 1.3).

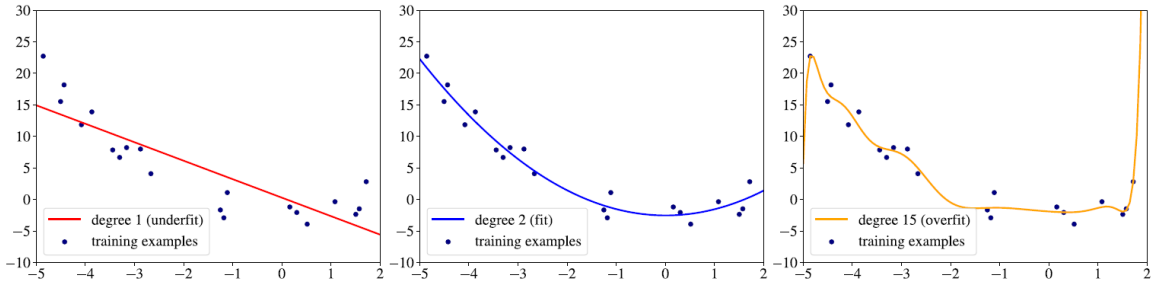

Figure 1.3: Underfitting (left), good fit (centre) and overfitting (right) [5]

In particular:

- **Underfitting Problem**: "Inability of the model to predict well the labels of the data it was trained on". The model is too simple for the data, or the features selected do not describe properly the behaviour of the system; [5]
- **Overfitting Problem**: "The model that overfits predicts very well the training data but poorly the data from at least one of the two hold-out sets". Probably it is more complex for the data, or there are many features with respect to training examples. [5]

Training and Validation performance can help to detect these two problems. Underfitting happens if the algorithm's accuracy is low for both sets, in this case number of input data must be increased. Overfitting happens if the accuracy is high in training, and low in validation, and so data must be changed in order to increase their heterogeneity.

1.3.4 Algorithm Performance

After the training and validation phases an evaluation of the algorithm accuracy is needed. For this purpose confusion matrix is used, in which each column represents the predicted values, while each row represents the real ones (Figure 1.4). It therefore describes the accuracy percentage in the validation phase which is obtained by dividing the sum of the values on the diagonal by the sum of all the values.Higher indexes along the diagonal indicate good accuracy. Obviously the higher these values are, the more precision increases.

$$Accuracy(\%) = \frac{T_p + T_n}{T_p + F_p + T_n + F_n} \tag{1.3}$$

		Condition Phase (Worst Case)		
		Condition Positive/ Shaded	Condition Negative/ Unshaded	
Testing Phase (Best Case)	Test Positive/ Shaded	True positive shaded T_p (*Correct*)	False positive shaded F_p (*Incorrect*)	Precision/Positive Predictive Value (PPV) $\frac{T_p}{T_p+F_p}$ x 100%
	Test Negative/ Unshaded	False negative unshaded F_n (*Incorrect*)	True negative unshaded T_n (*Correct*)	Negative Predictive Value (NPV) $\frac{T_n}{T_n+F_n}$ x 100
		Sensitivity/Recall Rate (RR) $\frac{T_p}{T_p+F_n}$ x 100%	Specificity Rate (SR) $\frac{T_n}{T_n+F_p}$ x 100%	

Figure 1.4: Confusion Matrix

1.4 Virtual Sensors and DVS

Virtual Sensor is a software algorithm able to estimate variables which are difficult to measure, starting from data that can be acquired easily. Its main feature is that it can be designed directly from the acquired data, and so it is able to substitute expensive physical sensors, or to perform "watchdog" activities of other sensors defined "mission critical".
Big Players as Facebook, Google and Amazon are making huge investments for the development of the most advanced algorithms for data-driven applications. Modelway in this context continuously adapt these software to implement virtual sensors for many sectors (automotive, aerospace, energy, environment, etc.).
Virtual Sensors can be classified in two main categories:

- **Model-Based Virtual Sensors**: They are based on two different steps: firstly, a mathematical model of the system must be identified, and after, a filter algorithm is designed (for example Kalmann Filter);

- **Direct Virtual Sensors**: It is a technology developed by Modelway and called with DVS acronym which allows to design a filter algorithm directly from the acquired data, without the necessity to identify the mathematical model of the system.

Sometimes, with DVS it is possible to solve complex non-linear problems, overcoming the physical limits of the Model-based approach. Its main benefits are:

- Reduction of development time and costs up to 60%-70%;
- Overall Time To Market Reduction;
- Significant accuracy improvement;
- Robustness versus not measurable operation conditions may be achieved;
- New applications possible, not feasible with conventional Virtual Sensor techniques;

Dvs is developed in MatLab or Python environment, moreover it can be integrated on different platforms. They can be installed on vehicle control unit, on servers, in the cloud or on a micro-controller platform. In the latter case the virtual sensor is not just a software, but it becomes an electronic device called EDVS (embedded direct virtual sensor). Taking the automotive field as a reference, an EDVS can be connected to the vehicle network, read the information of which it needs and return an estimate of the selected variable.

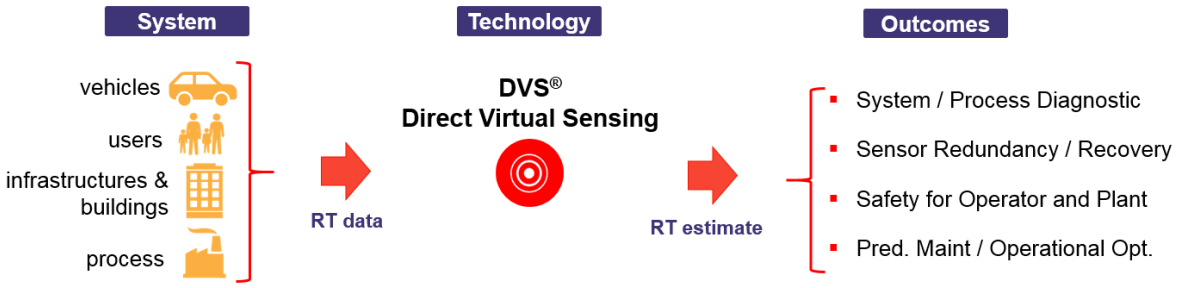

Figure 1.5: DVS

Chapter 2

Neural Networks

Artificial Neural Network (ANN) is a mathematical model similar to a black box, which receives one or more inputs, processing them, and perform one or more outputs. It is based on elementary units called "Neurons" grouped into a several layers with the rule that each "Neuron" of one layer is connected to all Neurons of the successive one through weighted connections (real numbers). The goal is to regulate this values in order to obtain the desired results.
A similar algorithm was proposed for the first time by Hebb in 1949 (Hebbian algorithm) which was based on the principle to reinforce the weight between two neurons if they are active at the same time. A classical example consists in a neuron able to recognize a circle and in an another one that checks the number six. When in the training phase a 6 is shown, the connection between these two neurons is reinforced in order to increase the activation of the neuron for the digit six the next time that a circle is visualized. Neural Networks are a consequence of this one, with the difference that they can learn online trough small updates on the connection weight. [4]
Moreover, Neural Networks are supervisor algorithms because they learn from a training set in which inputs are correlated at the corresponding output. For example in estimating the price of cars, it receives in input the main features and in output their prices. The main goal is to find the best combination of the weighted connections in order to decrease the loss function, i.e. the difference between the estimated output and real one. [4]
Basically, it works as a regression model:

$$y = f_{NN}(x) \tag{2.1}$$

$$f_{NN} = g_{NN}(W_{NN}z + b_{NN}) \tag{2.2}$$

So each unit takes as input the previous unit values multiplied by their respective connections with the addiction of another value called "BIAS value" (b_{NN}). After this

an Activation Function (g_{NN}) is applied which mathematically transform these values before propagating them. [5]

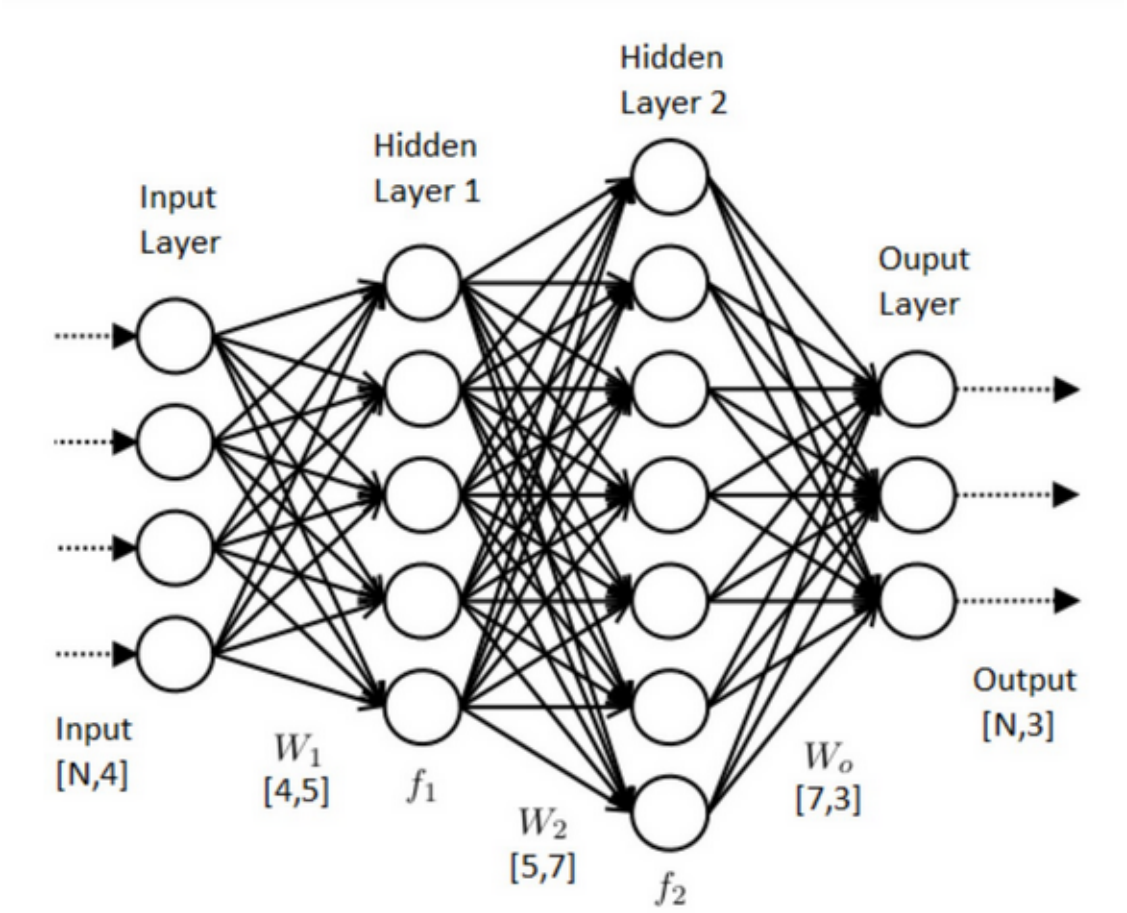

Figure 2.1: Typical Neural Network [11]

As shown in the figure there are three main layers:

- Input layer: It receives the input data of the Neural Network;
- Hidden layer: It elaborates the data computing the Activation Function;
- Output layer: It contains the final results;

2.1 Hidden Layer and Activation Functions

In order to understand better the role of the hidden layer, a definition of the main goal of the ANN is necessary, in particular they must be Universal Function Approximator, in other words they must be able to approximate every type of functions, mostly the non-linear ones. Obviously, to do this an introduction of non-linear factor is mandatory,

this is represented by the Activation Function. Without this function The Neural Network behaviour is simply equal to a regression model, that approximates the data distribution through a straight line, introducing a non-negligible error. [6]
An example is represented in the Figure 2.2 in which data are approximated with a linear and a non-linear function. As is possible to see, the non-linear one describes much better the distribution.

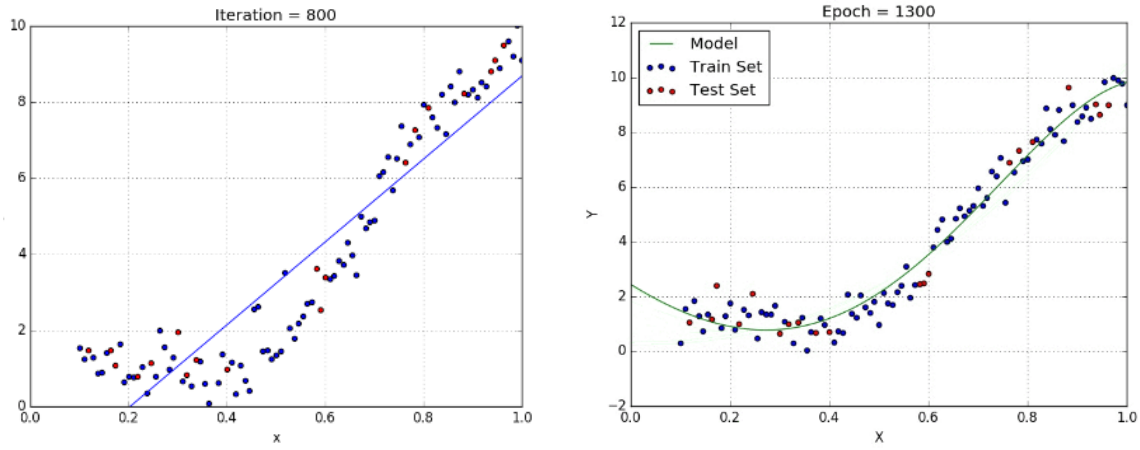

Figure 2.2: Linear (left) and Non-Linear (right) Distribution

Moreover in some cases, for example in a circular distribution, the linear regression is useless.

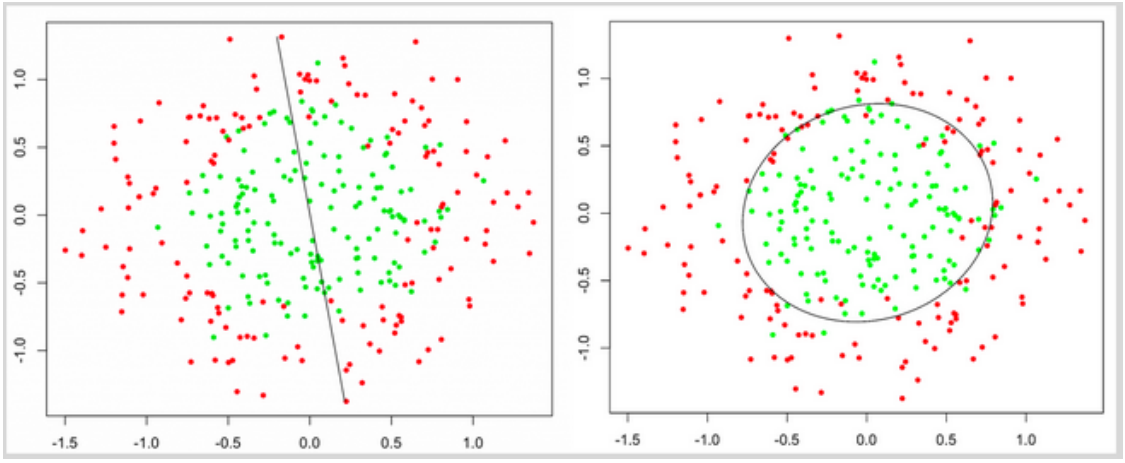

Figure 2.3: Circular Distribution [6]

The most important important activation functions are described below.

2.1.1 Step Function

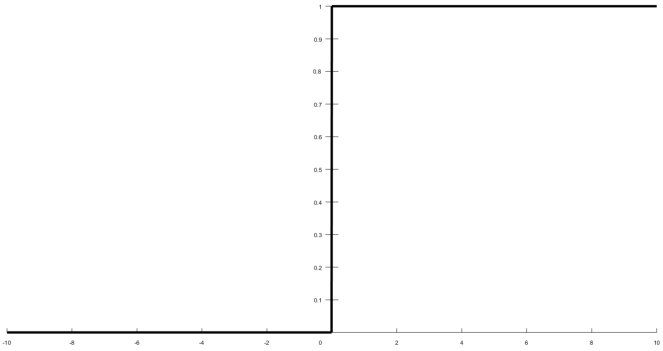

Figure 2.4: Step Function

The simplest function, that gives 1 in output for all positive inputs and zero for the negative ones. It has many advantages, for example it is very easy to compute and normalizes all output data in a range included between 1 and 0. On the other hand, it is not very stable and also its derivative is undefined in the point in which changes its value. The derivative represents the slope of the tangent line at that point, and it is fundamental in deep learning, as it determines the direction towards the correct values. If it is zero the behaviour of the network is uncontrollable, with the consequence that a tiny change in the parameters of the Network gives correct output for one input, but completely wrong for similar ones.[6]

2.1.2 Sigmoid function

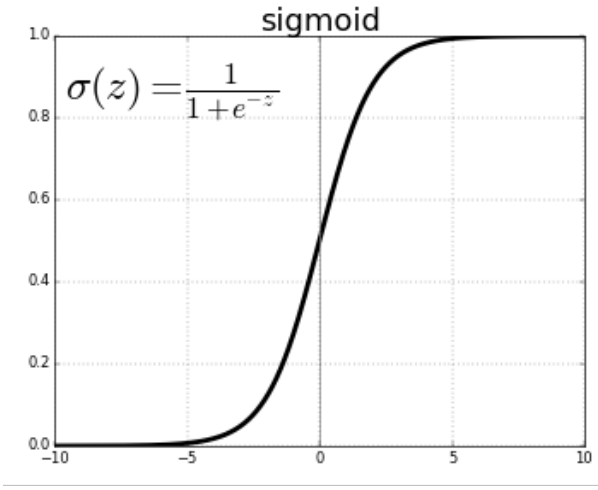

Figure 2.5: Sigmoid Function [7]

The sigmoid function is introduced in order to avoid the derivative problem of the step function, in fact its behaviour is equal to the previous one with the difference that it gradually changes its state from 0 to 1. The main advantage is that it compresses the values in this range and is therefore very stable even for large variations in values, on the other hand it has a very slow convergence (for very large input values the curve is almost flat) with the consequence that the derivative tends to zero. This poor responsiveness to the extremes of the curve tends to cause vanishing gradient problems. [6]
There is a more generalized activation function called Softmax.

$$Softmax(z_j) = \frac{e^{zj}}{\sum_{k=1}^{K} e^{zk}} \tag{2.3}$$

Where j is the single input and k the sum of all ones. As is possible to see, it is very similar to the Sigmoid Function, with the difference that all inputs are considered in the same time. In particular, with this feature the sum of all outputs is equal to 1, and so the probability of each class is obtained. This function is used in multi-class classification.

Vanishing Gradient

The Vanishing Gradient is a significant problem when the backpropagation method is used to upload weights. In particular, with this approach, partial derivatives (derivative with respect to weights) are multiplied together through previous layers in order to find the minimum of the loss function.

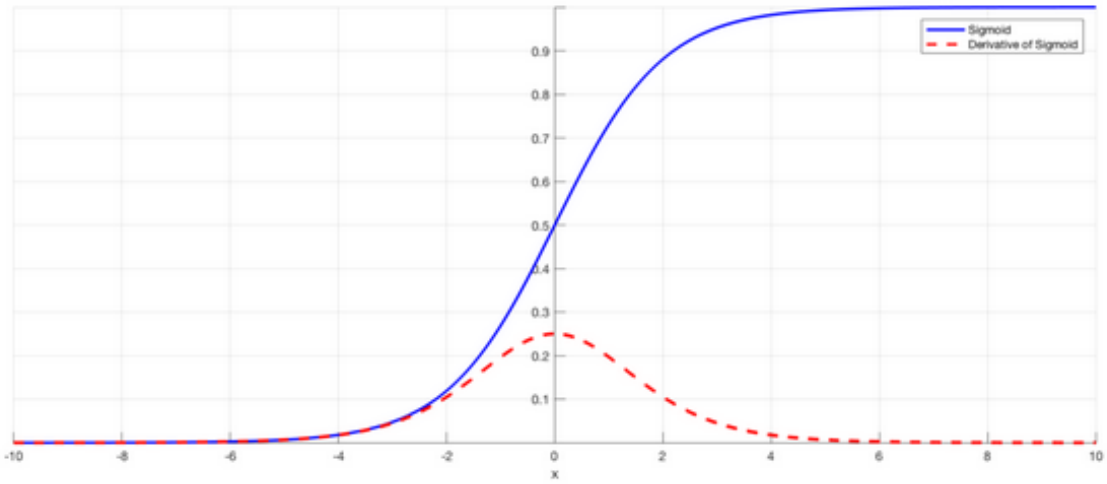

Figure 2.6: Sigmoid Function and its Derivative [8]

In the figure above, Sigmoid function and its derivative are shown. When activation

functions like this are used, for small or large inputs the derivative becomes close to 0. Since with backpropagation small derivatives are multiplied together, gradients of the loss function approaches zero, making the network hard to train. [8].
The solution of this problem is to use other types of activation functions such as ReLU.

2.1.3 ReLU

The ReLU function (linear rectifier unit) is a very useful function, especially in the intermediate layers. The reason is that it is simple to calculate: it outputs zero all negative inputs, while it leaves everything unchanged for values equal to or greater than zero.

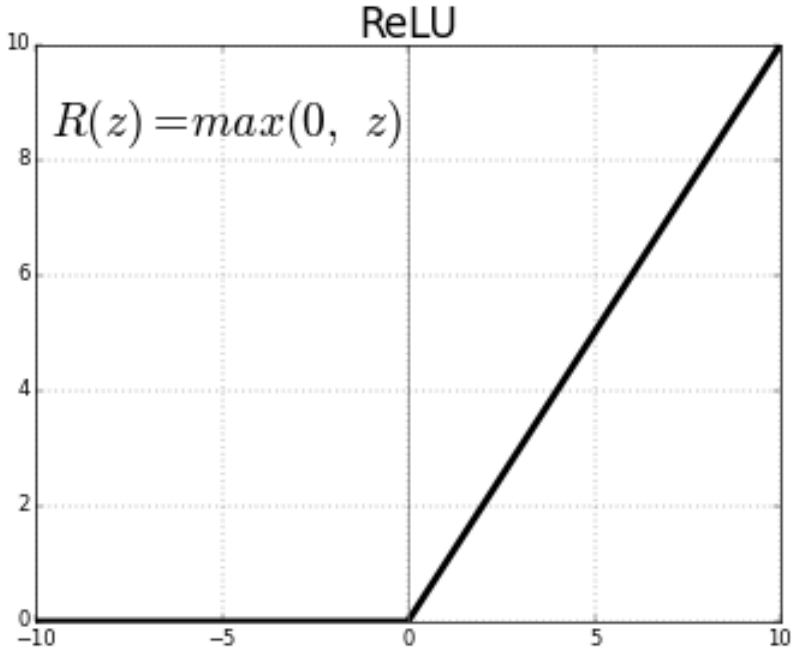

Figure 2.7: ReLU Function [7]

This simplicity, together with the fact of drastically reducing the problem of the vanishing gradient, makes this function very useful in the intermediate layer, where the quantity of steps and calculations is important.

2.1.4 Backpropagation Method

The backpropagation is a method to compute derivatives of the loss function with respect to weights. This is crucial for the Neural Network learning, since through derivatives it is possible to understand how much the predicted value is different from the expected one, and as a consequence, they can be used to update new weights.

The first step is to compute the derivative of the loss function

$$\frac{\partial E}{\partial w_{ij}} \tag{2.4}$$

Where w_{ij} is a generic weight between two successive Neurons. After weights can be uploaded for example through descent gradient:

$$w_{ij} = w_{ij} - \eta \cdot \frac{\partial E}{\partial w_{ij}} \tag{2.5}$$

η is called learning factor (value between 0 and 1).
This procedure is repeated until the minimum of the loss function is reached. [3]

2.2 Deep Learning

Deep learning is a sub-field of Machine Learning characterized by more consecutive hidden layers able to extract features from complex problems. An example of this algorithm is the LSTM (Long short term memory). It has been used in the first project of this Thesis in order to detect the fall of the operator.
Before going deep into LSTM, it is important to understand the need of this network which can be explained by the drawback of practical use of Recurrent Neural Network (RNN).

2.2.1 Recurrent Neural Network

A recurrent neural network is a class of artificial neural network in which the output values of a layer of a higher level are used as an input to a lower layer. This interconnection between layers allows the use of one of the layers as a state memory, and allows, by providing an input time sequence of values, to model a dynamic time behavior depending on the information received at the previous time instants.

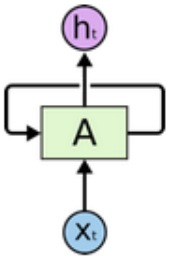

Figure 2.8: Recurrent Neural Network [9]

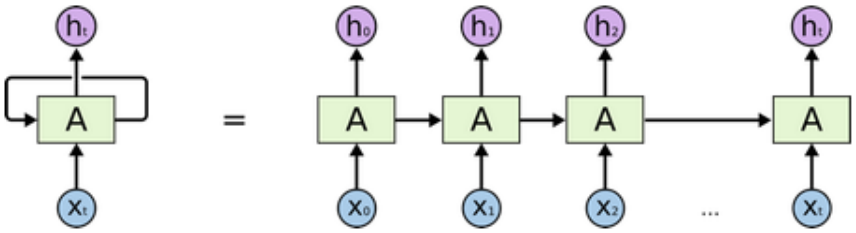

Figure 2.9: RNN-unrolled version [9]

The "unrolled" version is represented in the following figure for a clearer view of what has just been said.

A typical RNN takes in input information from the previous step in a loop. The output of one unit goes into the next one and the information is passed.
However in some application it should be wrong to learn from immediate past information. It is possible that the accumulation of the error gradient can generates an unstable network characterized by NaN values if the error is great or vanishing problem if it is too small.
From this disadvantages new variant of RNN model was created, called Long short term memory, with the advantages to use gates able to control the memorizing process. [9]

2.2.2 Long Short Term Memory

The following figure shows a typical LSTM architecture.

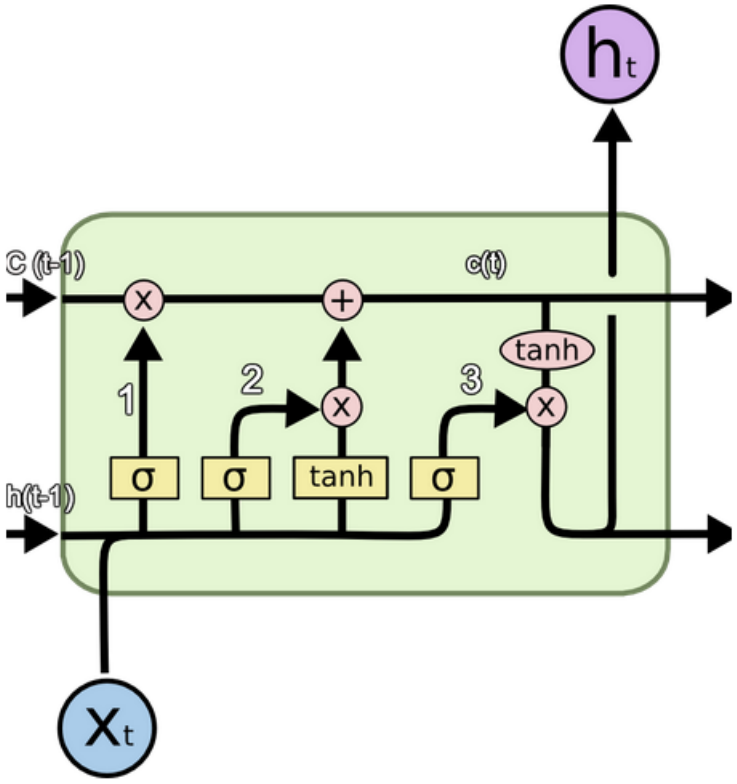

Figure 2.10: LSTM architecture [9]

Where:

- X: Scaling of information;
- $+$: Adding information;
- σ: Sigmoid Layer;
- $tanh$: Tanh Layer;
- $h(t-1)$: Output of last LSTM unit;
- $c(t-1)$: Memory from last LSTM unit;
- $X(t)$: Current input;
- $c(t)$: New updated memory;
- $h(t)$: Current output;

Now, a step-by-step analysis is presented in order to understand better the working principle of the LSTM.

Cell State

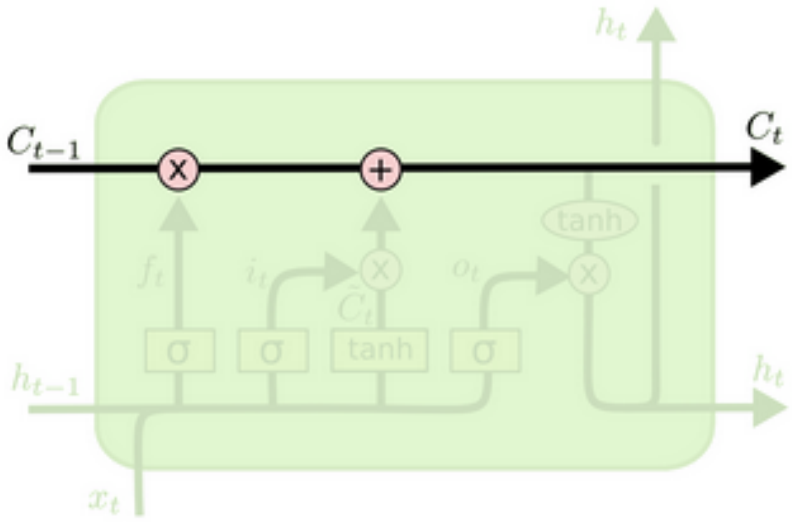

Figure 2.11: Cell State [10]

As said before, the main advantage of the LSTM network is the possibility to manage the memorizing process. This functionality is due thanks to the cell state. Similar to a conveyor belt, here the information is held and the LSTM has the ability to add or remove them from the cell through particular structures called gates. [10]

First Gate: Forget Gate Layer

With this gate the decision of which information of the old output should be uploaded to the cell state and which not is possible.

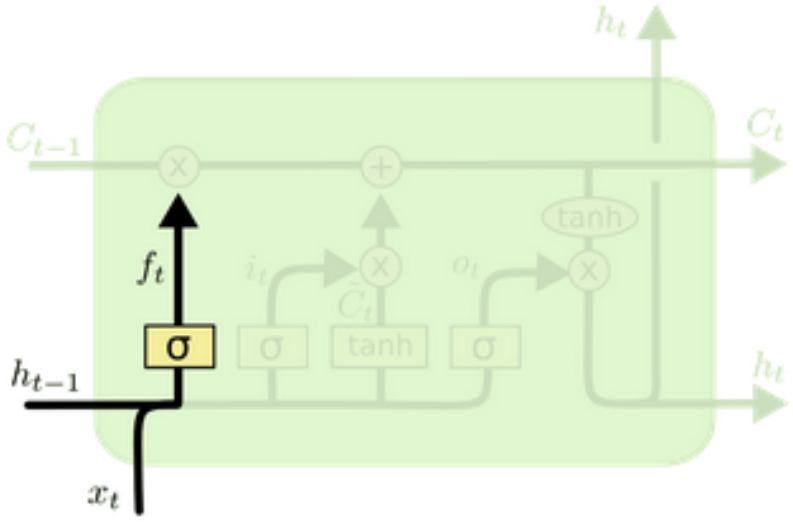

Figure 2.12: Forget Gate [10]

$$f_t = \sigma(W_f \cdot [h_{t-1}, X_t] + b_f) \tag{2.6}$$

Where:

- f_t: Binary value resulting from Sigmoid function;
- W_f: Value of the weighted connection between Neurons of two successive layers;
- b_f: Bias value;
- $h(t-1)$: Output of last LSTM unit;
- $X(t)$: Current Input;

This equation mathematically explains this functionality. In fact the first gate concatenates the two inputs (the current one and the output of the last time instant), obviously multiplied by the respective weighted connections.
[10] After this a sigmoid function conveniently designed is applied in order to decide which part of the old input should be removed (by outputting 0), and which should be confirmed (by outputting 1) [9]

Second Gate: Input Gate

This Gate is able to decide which information of the new input X_t store in the cell state.

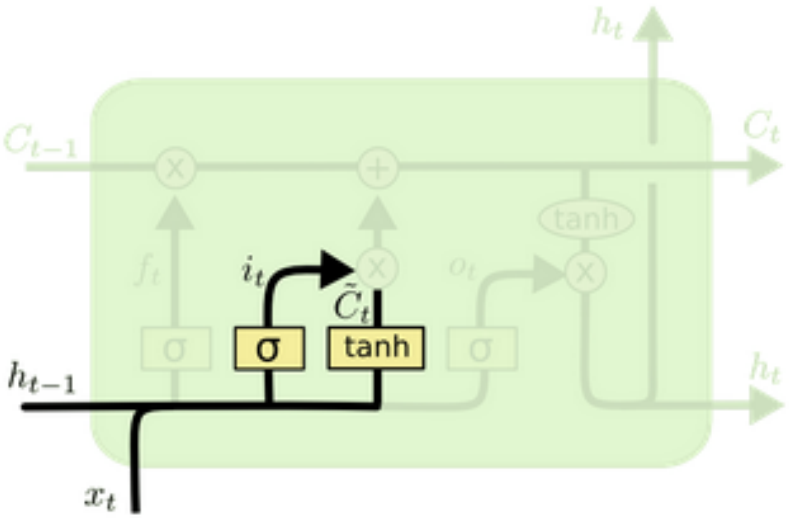

Figure 2.13: Input Gate [10]

$$i_t = \sigma(W_i \cdot [h_{t-1}, X_t] + b_i) \tag{2.7}$$

$$\tilde{C}_t = tanh(W_c \cdot [h_{t-1}, X_t] + b_c) \tag{2.8}$$

As shown in the equation above this Gate performs two function: The first one decides the information to upload with the sigmoid function; instead in the second one, a *tanh* layer creates a vector of all the possible values that could be added. [10]

Now it is possible to update this in the new cell State by adding it. Summarizing, there are two steps for updating the new cell State:

1. Multiplying the old cell State with the output of the forget gate (f_t) in order to reject the old useless information;

2. Adding the important information of the new input.

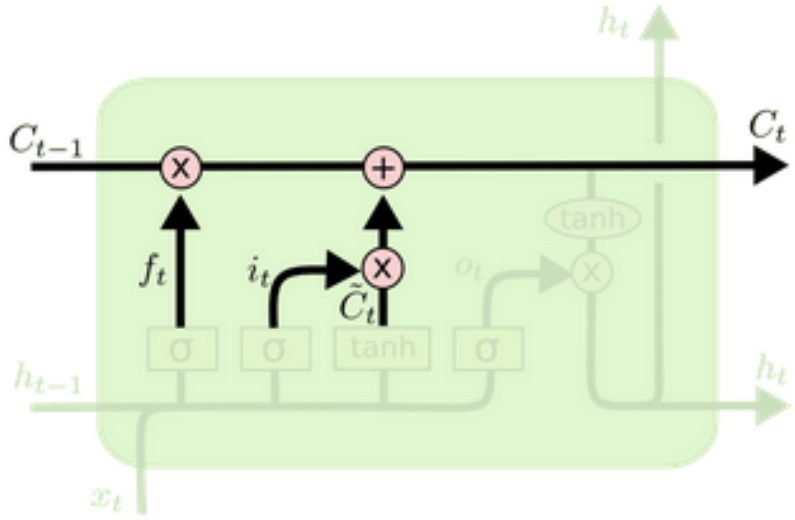

Figure 2.14: Update Cell State [10]

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t \tag{2.9}$$

Last Gate: Output Gate

The last step is to decide the final output. This is performed by the Output Gate which operates with two functions: A sigmoid layer decides which parts of the cell state will be the output. Then,a *tanh* generates all the possible values and multiplies them by the output of the sigmoid gate, so that it is possible to output the selected parts. [9] The following figure shows what just said.

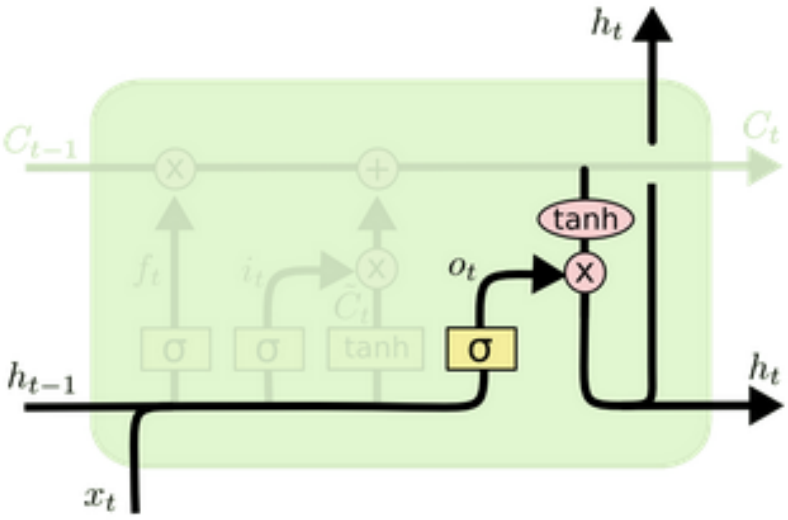

Figure 2.15: Output Gate [10]

$$o_t = \sigma(W_o \cdot [h_{t-1}, X_t] + b_o) \tag{2.10}$$

$$h_t = o_t \cdot tanh(C_t) \tag{2.11}$$

As just seen LSTM is very different to the classical RNN, since in the first one, it is possible to learn what information to store in long term memory and what to get rid of.

Chapter 3

Control Unit And Wearable Sensors

3.1 Control Unit

A Linux-based bridge system has been implemented. It is able to connect to the wearable Bluetooth Low Energy (BLE) devices for the operator monitoring, to save and collect the received data, and to send them outside (through socket TCP and WebSocket), so that they can be used by a remote client (for example by a $3D$ visualization instrument).

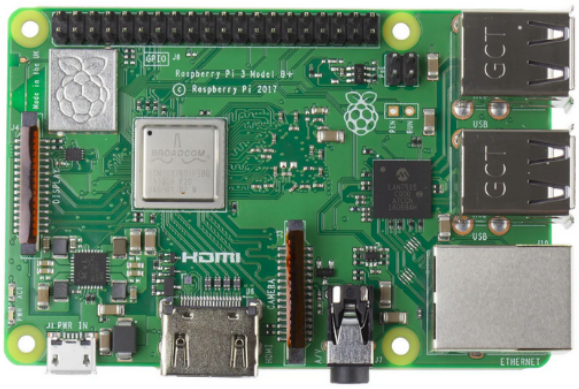

Figure 3.1: Control Unit

$RaspberryPi3B+$ has been chosen in order to make the whole system:

- Flexible: It is able to send data in Real-Time to any remote client (mobile app, visualization software, MatLab);
- Modular: It is possible to designate at the Control Unit the physical devices management (Bluetooth connections, synchronization and reception of the raw data). This approach simplifies the bridge system development;
- Reusable: This is because the Control Unit is independent by the elaboration system, this means that other wearable sensors can be used and so other application can be implemented with the same device.

The main hardware characteristics of the instrument are resumed in the following table:

Processor	Broadcom BCM2837B0 Cortex-A53 (ARMv8) 64-bit SoC @ 1.4GHz
Memory	1GB LPDDR2 SDRAM
Connectivity	LAN wireless IEEE 802.11.b/g/n/ac at 2.4GHz and 5GHz Bluetooth 4.2, BLE Gigabit Ethernet over USB 2.0
Access	40 pin GPIO
Sound and Video	HDMI full-size 4-pole stereo output and composite video port
SD Card Input	Micro SD for the Operating System and Data Storage
Power Supply	5V/2.5A DC

Table 3.1: Raspberry pi 3 B+ Datasheet

With the integration of a Bluetooth Dongle it is possible to associate the device with multiple BLE platforms. Each of it can link until 7 additional devices, with at least 20 simultaneous connections.

Figure 3.2: Bluetooth Dongle

A first version of the Extended Kalman Filter has been implemented, the device receives data from sensors (accelerometer, gyroscope, magnetometer) and computes

quaternions which give the absolute rotation of the rigid body.
Moreover, a touchscreen display has been integrated that allows the operator to activate/deactivate the wearable connection (in automatic or manual way) and to detects dangerous situations.

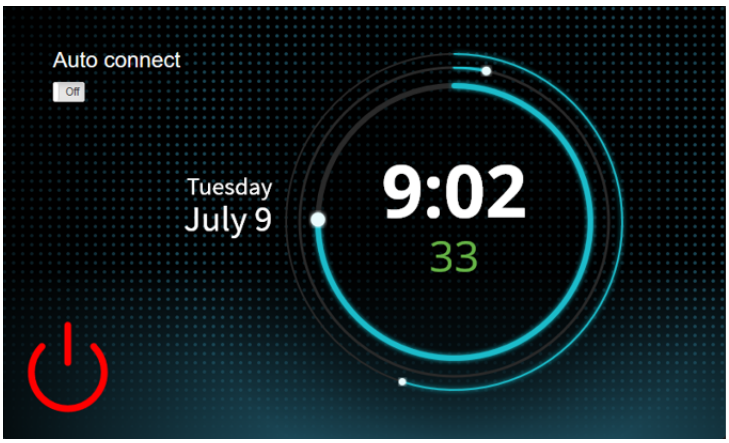

Figure 3.3: Display

The figure 3.3 shows the administration dashboard that allows to:

- Detect Bluetooth devices until an ideal range of 50 meters and connect with one or more of them
- Enable automatic connection with all available devices;
- Enable one or more sensors on the connected devices;
- Enable the BOSCH Sensor Fusion (only for the MbientLab boards);
- Receive raw data from sensors and computes, through the Extended Kalman Filter, quaternions
- Download (in CSV format) and delete the data stored on the DB;
- Start the communication service responsible for displaying the data on a specific TCP port;
- Start the communication service responsible for displaying the data on a WebSocket.

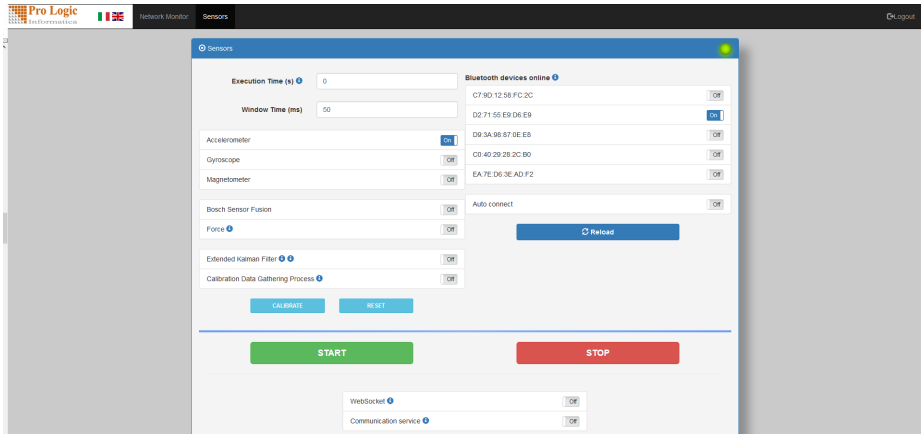

Figure 3.4: Graphical Interface

The main parameters are:

- **Execution time (s)**: Execution time (in seconds) of the acquisition process that it has been started. By default, it is set to 0 (i.e., the execution will be performed indefinitely);
- **Time window (ms)**: Sample time. By default, it is set to 125. For each window, the system will store in the database the average of each measurement belonging to the same type of sensor (accelerometer, gyroscope or magnetometer) of the same MetaWear device (identified by an id sensor in the database);
- **MetaWear devices switched on**: After pressing the 'Update' button, the devices available are displayed in this interface. At least one MetaWear card must be selected to start one new capture;
- **Modules**: it is possible to enable each combination of sensors on the connected devices. At least one sensor must be enabled in order to start a new capture.

3.2 Wearable Sensors

In the following table the main characteristics of the three possible solution for the experimental data acquisition are compared.

		MetaMotionC/R	CC2650 SensorTag
Accelerometro triassiale	Modello	BMI160	MPU9250
	Risoluzione [bits]	16	16
	Range di misura [g]	±2 -> ±16	±2 -> ±16
	Sensitività [LSB/g]	2048 -> 16384	2048 -> 16384
	Output data rate [Hz]	0.78 -> 1600	0.24 -> 500
Giroscopio triassiale	Modello	BMI160	MPU9250
	Risoluzione [bits]	16	16
	Range di misura [°/s]	±125 -> ±2000	±250 -> ±2000
	Sensitività [LSB/(°/s)]	16.4 -> 262.4	16.4 -> 131
	Output data rate [Hz]	25 -> 3200	4 -> 8000
Magnetometro triassiale	Modello	BMM150	MPU9250
	Risoluzione [bits]	13 (x, y-axis) 15 (z-axis)	14
	Range di misura [µT]	±1300 (x, y-axis) ±2500 (z-axis)	±4800
	Sensitività [µT/LSB]	0.3	0.6
	Output data rate [Hz]	25 ->300	-
Sensor Fusion		si	no

Figure 3.5: Sensors Features

The Three inertial platforms have the same range accuracy of the accelerometers and gyroscopes. MetaMotionC is the solution selected in the design phase (Figure 3.6).

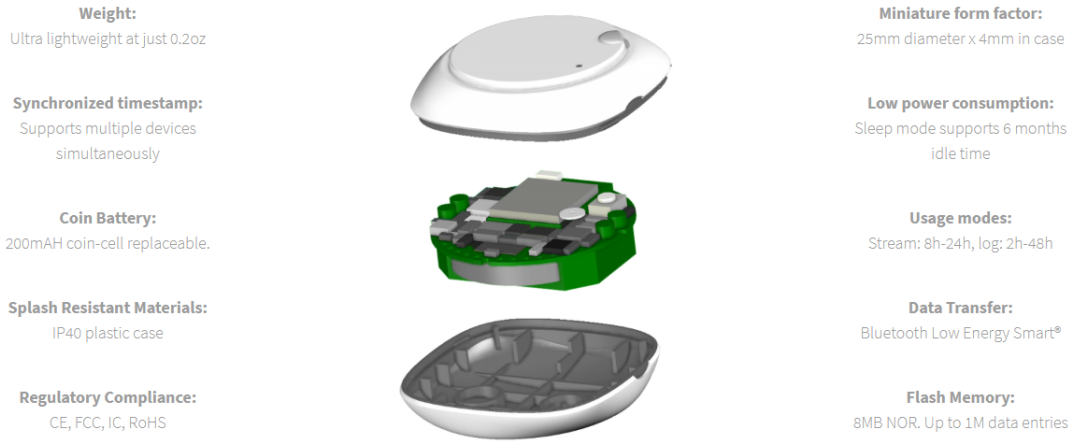

Figure 3.6: MetaMotionC

In a perspective of industrial use, the wearable platform that will be selected will be the $CC2650SensorTag$, which has lower costs (since it does not integrate the Sensor Fusion functionality), while guaranteeing the same measurement accuracy as the MEMS sensors (Figure 3.7)

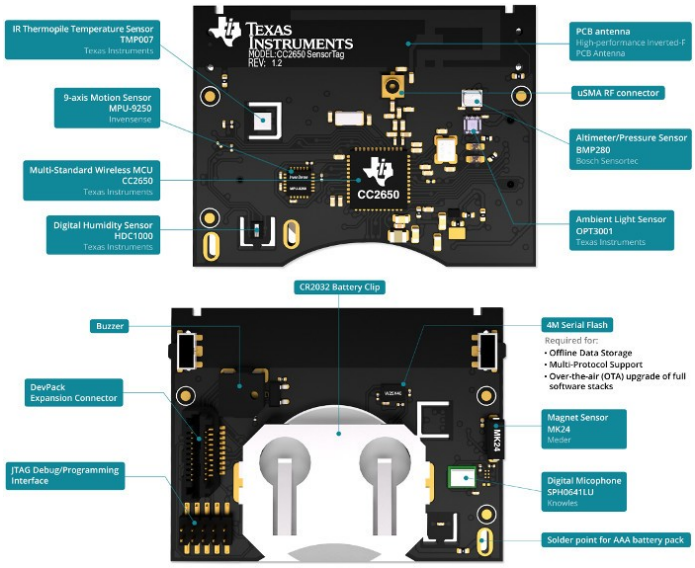

Figure 3.7: CC2650 SensorTag

Chapter 4

DVS For Man Down Detection

The main goal of the WP9 inside the Disloman project is to design a device able to detect the fall of the operator inside to an isolate working environment.
As shown in Figure 4.1, the operator is equipped with a bracelet that allows the acquisition of the movements and communicate them in real time to a control unit which integrates the Artificial Intelligence software for the estimation of the fall condition. Moreover it integrates a touch pad that interfaces with the operator and performs the acoustic and luminous signal in case of emergency situation.

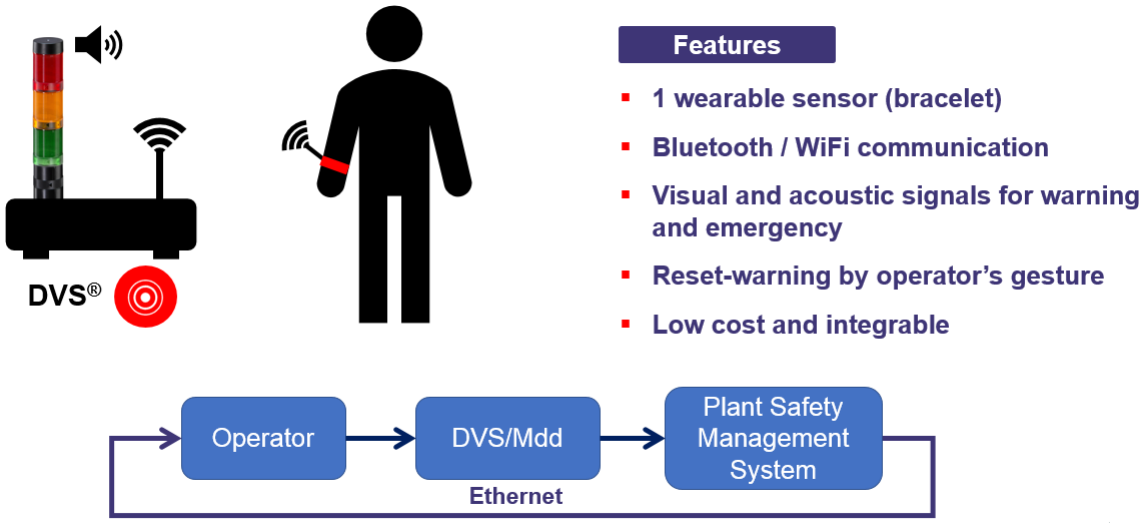

Figure 4.1: MDD Wearable Sensors and Control Unit

For the design of the Virtual Sensor, the Artificial Intelligence methodology developed by Modelway and called DVS (Direct Virtual Sensing) has been used. The designed estimation algorithm is indicated in this document with the DVS/MDD acronym (DVS for the Man Down Detection).

4.1 Design Settings and Planning (MDD)

During the design and validation definition phase of the device, the need to reset the DVS/MDD warning estimate through a precise gesture by the operator was born. This is due to the fact that a maintenance operator can be on the ground voluntarily in order to carry out his work activities.
For this purpose, a second virtual sensor called DVS/R (DVS for Reset) has been therefore designed to recognize the reset gesture. The gesture consists on beating the fist twice on a horizontal surface.

The Figure 4.2 summarizes the main design choices related to the operation of the two virtual sensors:

- The two virtual sensors estimations are based on inertial variables acquired in real-time from the wearable bracelet;
- The data from the inertial station are acquired with a sampling time of $50ms$;
- Both Virtual Sensors estimate a binary variable for the fall of the operator and for the reset;
- Estimated variables are updated once every 1.5 seconds.

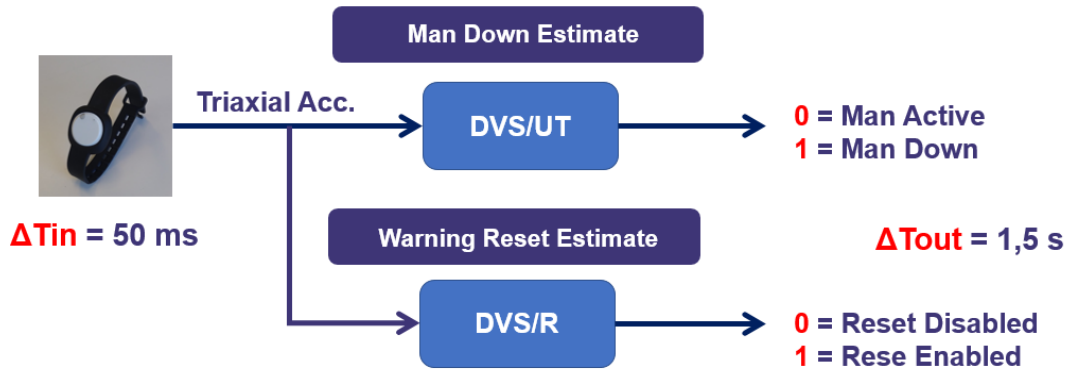

Figure 4.2: DVS/MDD and DVS/R-Working Principle

Considering that the fall of the operator and the reset gesture last more than a sampling time ($50ms$), the prediction of the two DVS takes place after the collection of 30 acceleration samples in $30x3$ "batches" (30 data for three axes) . This implies that the state machine is updated every 1.5 seconds (30 samples per $50ms$ of sampling time). This choice has been implemented in order to obtain a time window wide enough to capture the whole movements, but at the same time able to provide a response as fast as possible.

4.1.1 Acquisition and Elaboration Data (MDD)

Data have been acquired wearing the bracelet on the left hand and simulating the operations of falling, reset and any movements that can generate false positives. After this, the resulting *csv* file was filtered with MatLab in order to obtain normalized and ordered accelerations along x, y and z in three different columns.
By plotting them, it is very simple to identify the window containing the gesture of interest (Figure 4.3); Once this is done, again through MatLab, the output in correspondence with this last window is tagged to 1 and the remaining samples to 0.
In the following figure an example is represented, in which the accelerations of the generic work activities and the fall movement are shown.

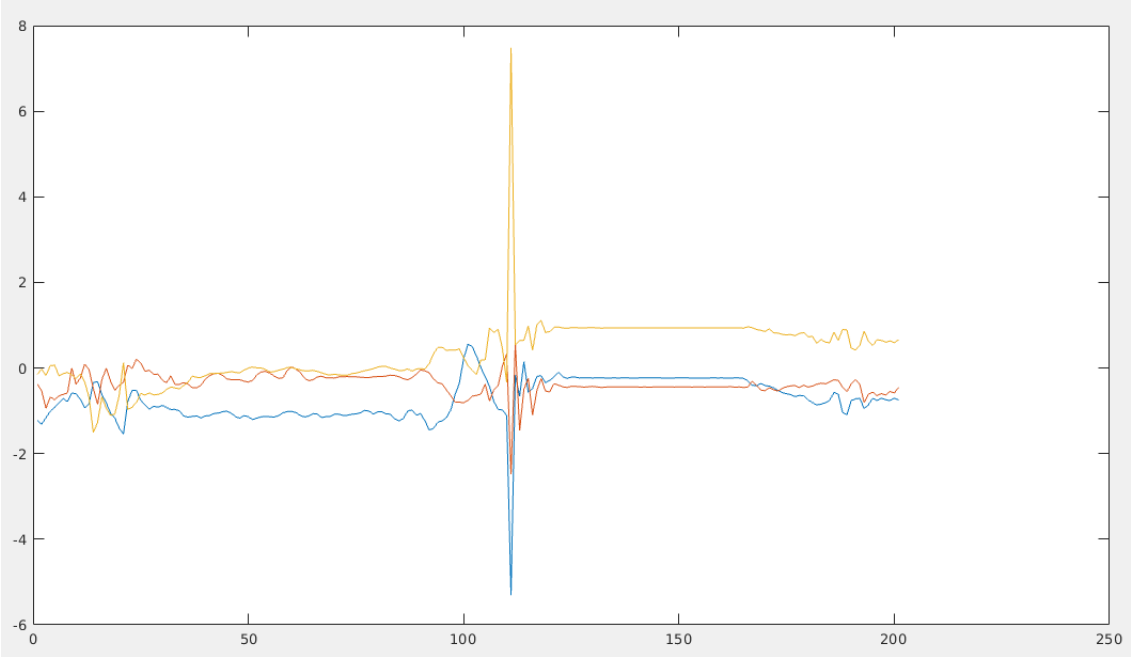

Figure 4.3: Accelerations of the fall

Considering that the window in which the fall movement is included must have a size of 30 samples the sector from 100 to 130 has been selected.
It is important to specify that the algorithm analyzes batches separately, and so under these conditions it is able to recognize the movement only if it is entirely covered by the window. If for example the operator falls between two different windows the virtual sensor estimation will be wrong. The training dataset has been therefore expanded taking into account also when a partial movement is acquired.

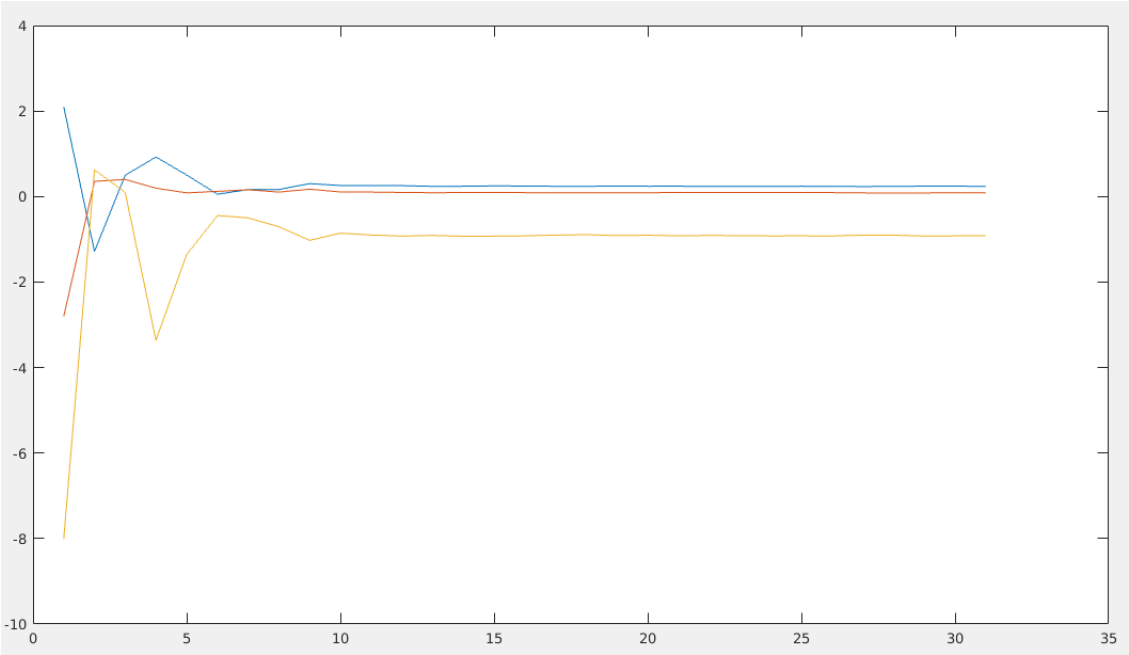

Figure 4.4: Cut off Accelerations of fall

The Figure above shows what just mentioned, in particular the output corresponding to this window is imposed to 1, in order to train the algorithm to recognize the fall movement even if it is not completely acquired.

The Figure 4.5 shows an example of the Reset gesture. According to previous procedure, a window composed by 30 samples and in which the gesture is included has been selected.
In this case it starts from sample 120 and ends at 150. The output vector has length equal to the input (170 in this figure) and its values are 0 until the $120th$ value, 1 from 120 to 150 and again zero until the end.

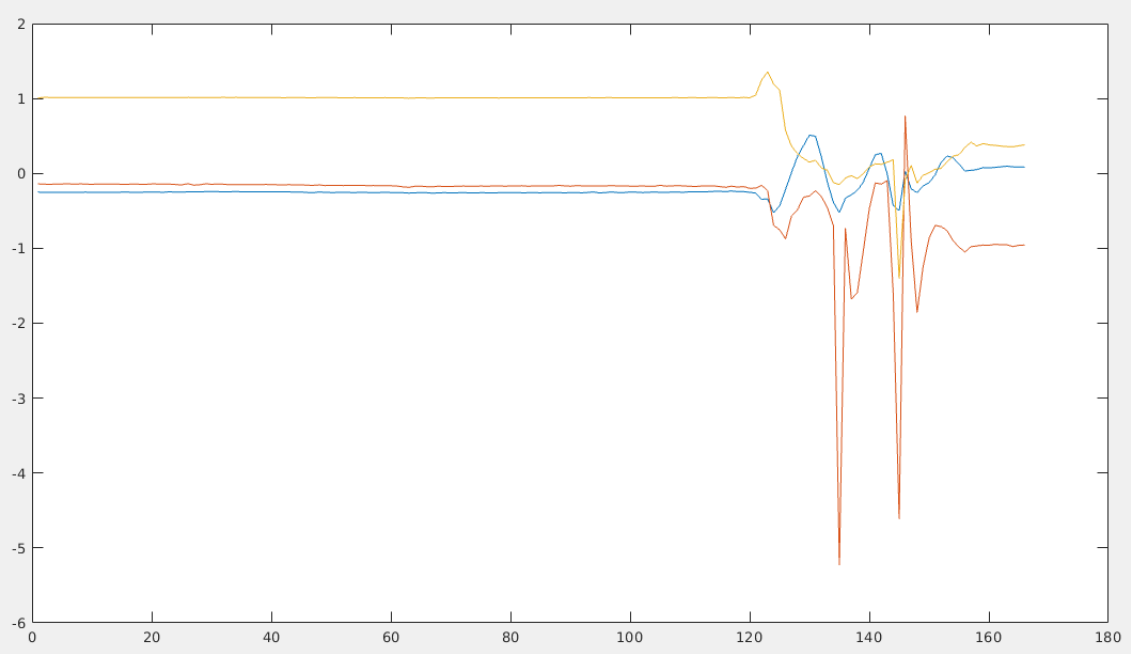

Figure 4.5: Accelerations of the Reset Gesture

Also in this case the window has been shifted in order to train the algorithm to recognize the reset gesture even if it happens during the updating of the state machine.

In the example shows in the figure below the second part of the complete movement has been cut.

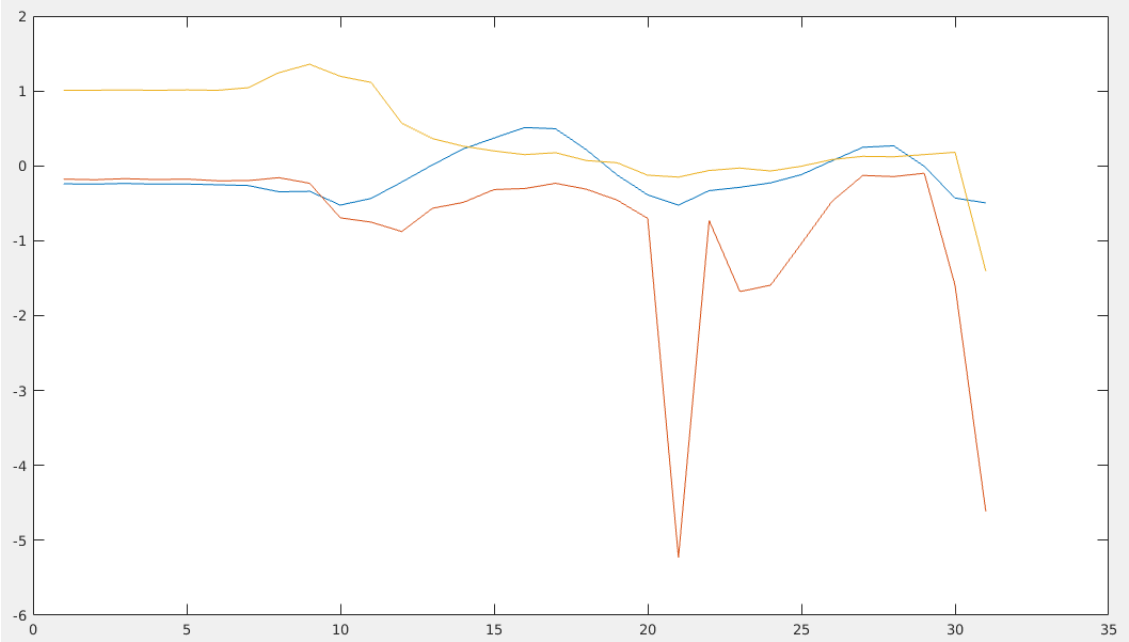

Figure 4.6: Cut off Reset Accelerations

At the end of this procedure a dataset is obtained in which the data samples are divided into batches characterized by the three accelerations as input and one binary value (1 or 0) for each row as output.
The last step consists in dividing this matrix into training data and validation ones, paying attention in selecting a homogeneous proportion; To do this, the cross-validation method has been used, which the total dataset is divided into k parts of equal size, of which $(k-1)/k$ samples are destined for training and $1/k$ ones for validation.

4.1.2 DVS/MDD

The DVS/MDD training starts from the acceleration measurements of the MetaMotionC platform. To do this, various experimental tests were carried out, at the Modelway office, simulating the following possible situations:

- Generic work activities of an operator;
- Falls of an operator caused by possible illnesses;
- Operator stopped on the ground in different possible positions.

The first of the three occurrences corresponds to a safe situation, while the two remaining to a warning condition. 68 tests were carried out, so as to have an exhaustive design and validation dataset of the different possible operating conditions.

4.1.3 DVS/R

The gesture that identifies the reset has been defined with the project partners, following these specifications:

- The movement must not be confused with those that normally the operator must perform during maintenance (i.e. wrist rotation, when using a screwdriver);
- It must be a gesture that clearly identifies that the operator is conscious and intends to reset the condition of potential danger;

On the basis of these hypothesis the movement chosen has been represented by "2 or 3 fist shots in a vertical direction on a rigid plane, obviously with reference to the limb on which the bracelet is worn.

4.2 Methodological Approach (MDD)

As far as the methodology is concerned, the Virtual Sensor for Man Down Detection is based on a "Deep" Neural Network called LSTM (Long-short term memory). The reason lies in the need to collect, as discussed in the previous paragraph, the acquired accelerations in batch composed by 30 samples before analyzing them. This requirement implies the use of an LSTM network, thanks to its characteristic of recognizing patterns inside temporal data strings.
The working principle of this Neural Network is explained in the paragraph 2.2.2

4.3 State Machine MDD

Figure 4.7: State Machine MDD

State Machine has been implemented in order to establish the worker status. It must be able to provide the transitions between status and to return a feedback of the current one. The Transitions are managed by the DVS output that can be 1 or 0. Four different situations are possible:

- **Safe State**: Standard and initial condition;
- **Warning State**: The State Machine enters in this status when the output of the DVS/MDD sensor is 1 and so accelerations that can be dangerous for the worker have been intercepted. In this scenario a signal of the current status is sent to the operator and a timer starts to count until a certain value that has been decided in the design phase. If the worker resets the device in this range the output of the Reset sensor is 1 and so the State machine moves to the "Maintenance" state, otherwise the worker is not able to perform the reset gesture and the State Machine moves to the Dangerous state;
- **Maintenance State**: Reset gesture has been detected during the Warning state and the operator is in the Maintenance one. It is important to highlight that this state is necessary to avoid an infinite loop in the State Machine. Considering for example a situation in which the State Machine returns to the Safe State as a consequence of the reset gesture, in this case if the DVS/MDD output is 1 again the current state becomes "Warning State" and another reset gesture is requested by the device. Summarizing it is possible to assert that this state allows us to describe more precisely the behaviour of the operator. At the begin of this state another timer is activated waiting the next gesture by the operator: if another reset is performed in a default range the worker returns to the Safe State, else it moves to the Warning situation. Also this choice is designed to avoid dangerous situations.
- **Dangerous State**: The operator is not able to perform a reset gesture during the warning condition the new state becomes Dangerous and signal is sent to the main alarm system of the Factory. Only through a manual reset the monitoring can be restarted.

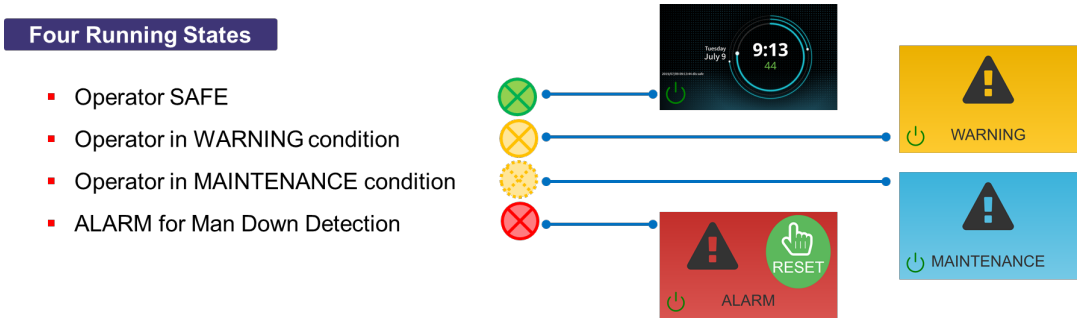

Figure 4.8: Possible States

4.4 Results (MDD)

4.4.1 DVS/MDD

The DVS/MDD has been designed and validated using the "2 fold cross validation" method: the acquired data were divided into 2 ($D1$ and $D2$) dataset, and the virtual sensor was first designed on the $D1$ data and validated on the $D2$ data, and then vice versa (designed on $D2$ and validated on $D1$). The results obtained can be summarized through the confusion matrix (Figure 4.9) , in which each column represents the predicted values, while each row represents the real ones.
It therefore describes the DVS accuracy percentage in the validation phase which is obtained by dividing the sum of the diagonal values by the sum of all values (Table 4.1).
As shown in the confusion matrix, the DVS/MDD identifies the fall 270 times out of 270, and commits only two errors, identifying two non-existent falls.

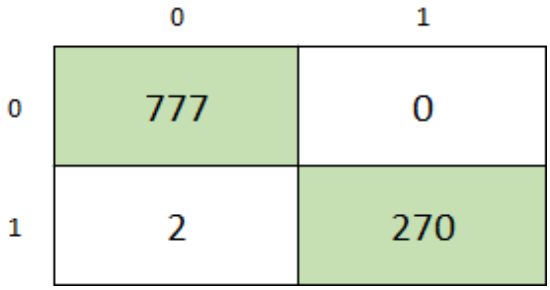

Figure 4.9: Confusion Matrix MDD

Training Accuracy Estimation	100 %
Validation Accuracy Estimation	99.8 %

Table 4.1: Training/Validation Accuracy DVS/MDD

The following table shows the accuracy of the whole device during a Real Test carried out in Modelway wearing the bracelet and simulating all the possible activities of the operator.
Twelve tests were carried out for 12 minutes each one and by falling once every 2 minutes for a total of 120 falls. 114 were identified, with 5 false positives.

Accuracy DVS/MDD	92 %

Table 4.2: Test Accuracy DVS/MDD

4.4.2 DVS/R

Similarly, the DVS/R has been designed using the "2 fold cross validation" method. The following confusion matrix (Figure 4.10) shows the estimation results obtained in validation for the DVS/R; In this case, there are no false positives, this means that a reset signal is never reported when the respective gesture has not been carried out. Furthermore only one reset on 121 is missed.

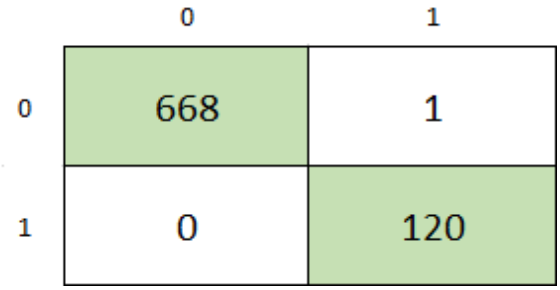

Figure 4.10: Confusion Matrix Reset

The following table shows the DVS accuracy in training and validation.
It is important to highlight that the confusion matrix shows the DVS performance during the validation phase. However the second percentage of the table 4.3 can be easily computed by dividing the sum of the diagonal values by the sum of all values of the matrix.

Training Accuracy Estimation	100 %
Validation Accuracy Estimation	99.9 %

Table 4.3: Training/Validation Accuracy DVS/R

The table below shows the DVS/R accuracy during a test carried out in Modelway office, wearing the bracelet and simulating all the possible activities of the operator. Also in this case 12 tests were carried out for 20 minutes each one and performing a reset every 2 minutes for a total of 120 gestures. 116 were identified, with 0 false positives.

Accuracy DVS/R	97 %

Table 4.4: Test Accuracy DVS/R

Chapter 5

DVS For Warning Area Detection

The main goal of this section inside the Disloman project is to design a device able to track the arm position of the operator that works near a mechanical press (at Honestamp Factory). In particular, it stops the action of the machinery each time the hand of the worker is inside the dangerous area.
In the design phase, an Artificial Intelligence methodology developed by Modelway and called DVS (Direct Virtual Sensing) has been used. In this section it is mentioned with DVS/WAD acronym (DVS for Warning Area Detection).
The total working area is divided into safe area(green), warning area(yellow) and dangerous one(red) as shown in figure below.

Figure 5.1: Mechanical Press

An initial choice was based on the use of three wearable sensors, reduced later at two of them in order to find a good trade-off between reduction of the problem complexity and DVS performance.

Sensors have been placed as follow:

- Sensor 1: Placed in a bracelet in order to detect the wrist movement;
- Sensor 2: Placed on top of the arm in order to detect its movement;

Sensor Signals (accelerometer, gyroscope and magnetometer) have been received and analyzed by a control unit which is able to perform the Artificial Intelligence algorithm (DVS) for the Arm Tracking estimation (Figure 5.2).

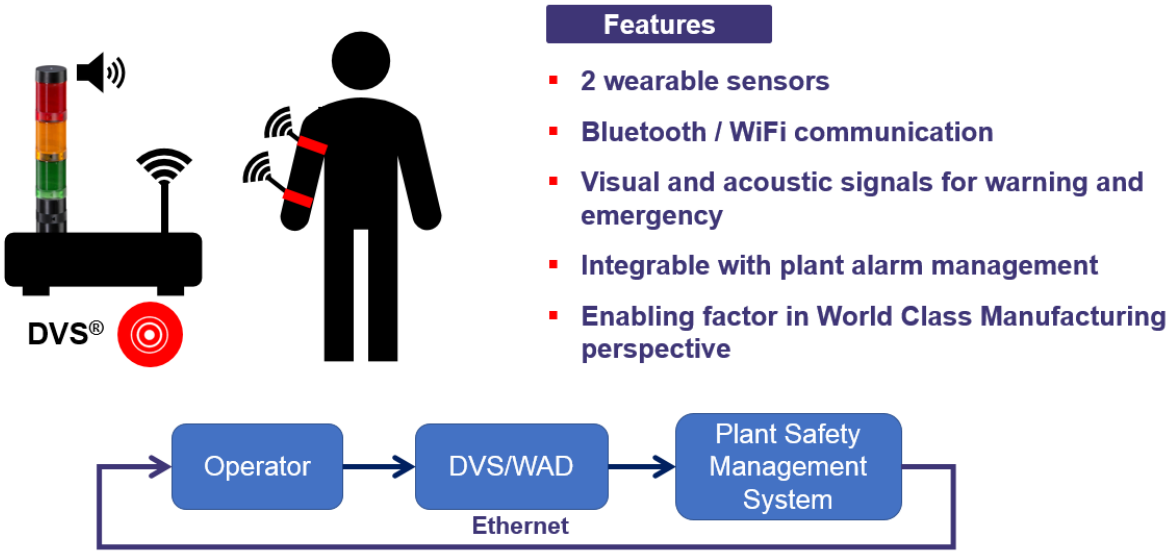

Figure 5.2: WAD Wearable Sensors and Control Unit

The Designing approach of this Virtual Sensor has been divided in two parts in order to validate the methodology starting from a simplified model ($2D$ implementation) and use it in the complete one ($3D$ Implementation).

5.1 Design settings and planning (WAD)

5.1.1 2D Implementation

In this first phase, the DVS/WAD1 has been trained using the quaternions estimation and considering the working area as a plane splitted into Safe zone and Warning one (Figure 5.3).

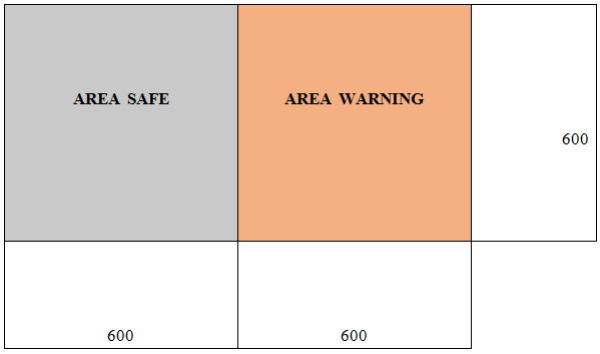

Figure 5.3: 2D Safe and Warning Area

Gestures of the operator have been acquired in both zones to obtain a balanced dataset for the training and validation session.
Considering that the estimation errors are concentrated around the boundary zone, an additional division has been performed in order to help the algorithm to recognize the differences between the two areas (Figure 5.4).

Figure 5.4: Safe and Warning Area Additional Division

The following table shows the results obtained.

Accuracy Safe Area Estimation	100 %
Accuracy Warning Area Estimation	95 %

Table 5.1: Accuracy DVS/WAD1

5.1.2 3D Implementation

During the training phase, drawing and replicating a copy of the same size of the mechanical press is mandatory for a good designing of the DVS/WAD. This is because it learns gestures inside this space and replicates them only if the orientation and dimensions are the same.
With AutoCad, an illustration of the interested areas has been drawn (Figure 5.5) and after this a duplicate of the machinery has been built.
It represents Safe area (green), warning area(yellow), and dangerous one (red).

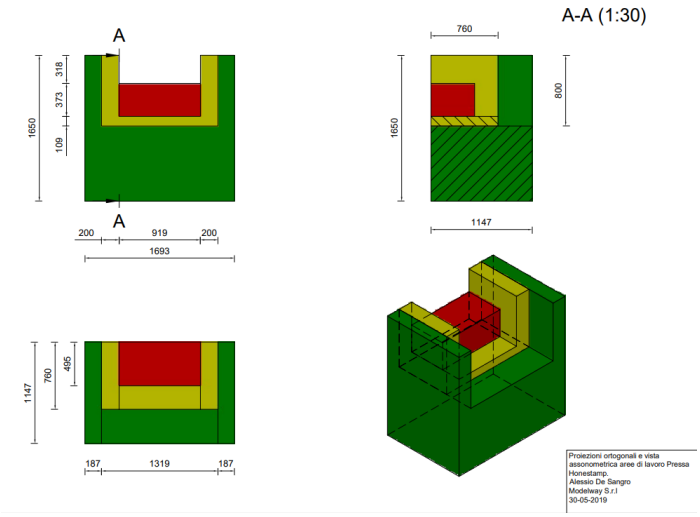

Figure 5.5: Drawing of the working areas

At the starting phase of the project, two different solutions were identified:

1. DVS/WAD1 for the warning area detection with the use of the absolute orientation estimation (quaternions);

2. DVS/WAD2 for the warning area detection with the use of MEMS sensors (accelerometer, gyroscope and magnetometer)

The first solution is more simple to implement than the second one, but it provides only the orientation of the arm with respect to the starting reference point. For this reason the initial position of the operator must be always the same, otherwise the virtual sensor gives a tenuous estimation. Moreover, the quaternions are computed by the control unit through sensor fusion algorithm (Extended Kalman Filter), that guarantees a consistent evaluation of the arm position only for few seconds, after this time the drift errors become too big and consequently the estimations are wrong.

Therefore, the second method is selected. It is important to highlight that the magnetometer computes the absolute position (with respect to Magnetic Field of the Earth) by integrating this measure with the accelerometer one through a sensor fusion algorithm. Obviously, the problem of the rise of the drift error described before is relevant also in this approach, with the difference that the time period in which the estimation is reliable is more greater.

5.1.3 Acquisition and Elaboration Data Procedure

The Acquisition Data procedure requires that:

- The starting point of the operator is always the same;
- The dimension of the working areas and their absolute orientation are equal to the mechanical press of Honestamp Factory.

The sampling frequency has been fixed to a $20Hz$ in order to have fast update of the device and to avoid the loss of a possible important sample for the correct estimation. Experimental data have been acquired in Modelway Office, during which operator's gestures have been simulated near the cardboard copy of the mechanical press. During the acquisition, the time spent by the hand-operator in each area has been timed in order to label, as unambiguous as possible, movements with their respective area (Safe, Warning, Dangerous). In fact, in the pre-processing phase it was possible to obtain the number of acquisitions corresponding to each area by dividing the timed time by the sampling one (50ms).
The resulting file is a table in which at each sample (composed by acceleration, angular velocity and position along x, y, z), correspond an output that can be 0 (Safe),1 (Warning) or 2 (Alarm). Particular attention has been paid to the boundary areas in order to strengthen the DVS in the estimation of transitions.

5.2 Methodological Approach (WAD)

For the DVS/WAD a simple Neural Network algorithm composed by two sub-layers with 25 Neurons each one and three regressors is implemented. This means that the Virtual Sensor estimation happens after the collection of the sensors data until the third previous time.
Imaging for example to turn on the device and starting the monitoring. At a first time instant (after $50ms$) the control unit acquires data (of accelerometer, gyroscope and magnetometer) from the two wearable sensors and save them into a vector. Hence at a time t its dimension is $1x18$ (3 sensors,3 axis, 2 wearables). In the same way, at a

time $t+1$ (after $100ms$) sensors data are captured and added at the previous ones; now the dimension is $1x36$.
The procedure is equal for the time $t+2$ and $t+3$ until the overall dimension of the input vector becomes $1x72$. Now the DVS/WAD estimation happens and, as a consequence, the state machine establish the current state.
In other words, the collection of data until three previous instants time is necessary for the DVS prediction. Initially, when the device is turned on, the estimation happens after $200ms$, after this time period every sampling time the control unit substitutes new data with the older ones (i.e. $t-3$) with the FIFO approach. In this way the dimension of the input vector is always $1x72$.
This choice hes been implemented in order to take into account the dynamic of the arm. Indeed, this Virtual Sensor must be able to track the arm of the operator, for this purpose performing the estimation considering also the previous position of the arm help it to improve its accuracy.

5.3 State Machine WAD

The DVS/WAD reads the variables acquired by the wearables and generates in output the estimation of the status relative to the operator's hand, coherently with its current position in the Safe, Warning or Alarm area.
State Machine has been implemented in order to provide the transitions between status and to return a feedback of the current one (Figure 5.6).

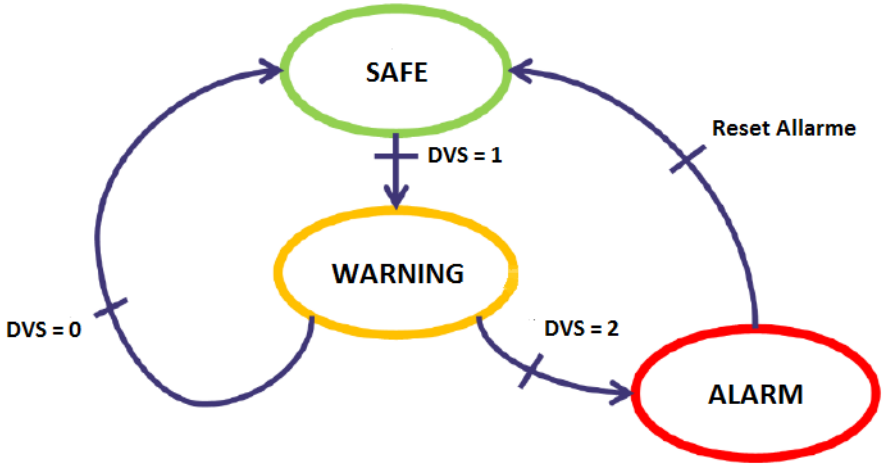

Figure 5.6: State Machine WAD

As shown in the figure below, the transitions are managed by the DVS output. Initially, the operator is in a safe state, and remains so until the sensor detects the

position of the hand in correspondence with the warning area; at this point the DVS output is 1 and the state machine performs the transition. With the same approach, when output 2 is detected, it passes to the alarm state with consequent acoustic and luminous signaling. A relevant exception concerns this one, which can change state only through an intervention by the operator, who manually disables the alarm signal once the safety condition is restored.
The software performs the functions described above for a limited period of time, after which the operator is forced to place his hand on the starting point (tablet) and to restart the monitoring. This solution has been established to overcome the drift problem described above, according to which the DVS estimation are reliable only for a certain time interval.

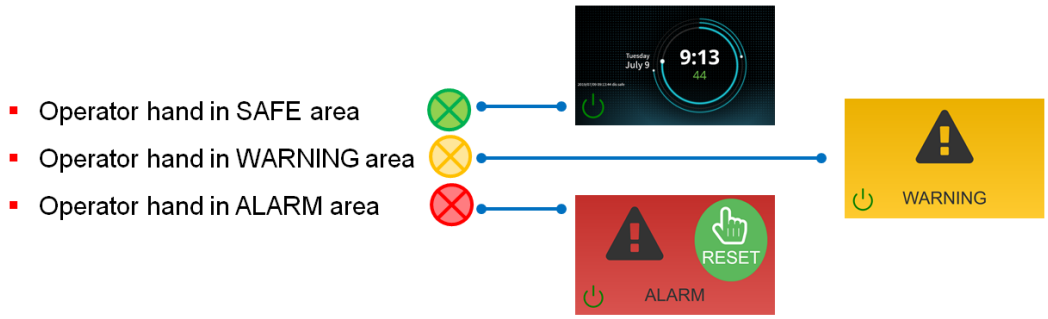

Figure 5.7: Possible States

5.4 Results (WAD)

The obtained results can be summarized through the confusion matrix, in which each column represents the predicted values, while each row represents the real ones. It therefore describes the DVS accuracy percentage in the validation phase which is obtained by dividing the sum of the values on the diagonal by the sum of all the values (Table 5.2).
As an example the DVS/WAD predicts the alarm zone 2672 times correctly, 247 times predicts 2 when the arm of the operator is in warning zone, and 2 times predicts 2 instead 0.

	0	1	2
0	3489	202	17
1	154	4417	213
2	2	247	2672

Figure 5.8: Confusion Matrix WAD

Higher indexes along the diagonal indicate good DVS/WAD accuracy. Obviously the higher these values are, the more precision increases. Moreover, as expected, the major errors are collected in adjacent areas, a problem that could be limited by acquiring further experimental data on border areas with consecutive re-calibration of the algorithm.
Table 5.2 shows the percentages of accuracy in training and validation. In particular, the algorithm has been trained on 4/5 of the total data and validated on the remaining ones (1/5). The percentages are obtained by taking into account all three states, making the sum of the exact predictions divided by the total number of data. The validation estimation is homogeneous with the above confusion matrix, as explained above.

Training Accuracy Estimation	96.3 %
Validation Accuracy Estimation	88.2 %

Table 5.2: Training/Validation Accuracy DVS/WAD

Chapter 6

Conclusions

The main scope of this work was to demonstrate that Artificial Intelligence algorithms can help the development of new devices for the Human Safety Monitoring. The results obtained with this Thesis show that it is possible to Design Virtual sensors for the fall detection and for the arm tracking of the operator.
However, they are two demonstrators (before prototype), so many aspects must be modified in order to improve their performance. The main technical problems encountered during both research projects are analyzed below. Finally, potential ideas are suggested for future project developments.

DVS/MDD

- False Positives: The first problem concerns the detection by the device of false positives, i.e. a condition in which a fall is identified even if the operator is in that moment safe and conscious. These cases are strictly dependent on the heterogeneity of the data that has been considered in the training phase. Particular attention must be paid to the identification of critical movements in order to further strengthen the DVS robustness, minimizing these types of errors. In the future industrial applications, the device will require an initial "calibration" period in which the virtual sensor is trained to correctly recognize the most critical movements;

- Real-Time Implementation: The second issue is linked to the intrinsic characteristics of the hardware. As the latter is "Linux-based" and therefore not Real-Time, it is not able to provide immediate action in response to an event. In fact, whenever the state machine is updated, it loses approximately $100/120ms$ to provide the current state; in this period of time fundamental samples for the identification of the fall condition could be lost. As an example, the comparison between the software simulation performance of Virtual Sensors (on PC) and the

real one (obtained during the test on Raspberry Pi) shows how the difference in time can lead to the failure to recognize a gesture that occurs straddling two time windows, that is, during the state machine update. Also in this case, during the design phase, DVS has been trained to recognize a certain movement even if it is only partially sampled. Despite that with this improvement excellent results have been obtained, it remains a factor to be taken into consideration.

The future developments include the use of a Real-Time system able to provide an immediate response without any lost of sample, further perfecting the accuracy of the device and ensuring complete reliability during operation.

DVS/WAD

The most important problems mainly concern the lack of a direct measurement reference of the position and the non-real-time system of the device. In particular, as seen previously, the magnetometer does not guarantee sufficient accuracy for a position reference of the hand, resulting in a difficulty to improve the performance of the virtual sensor. Furthermore, a non-immediate upgrade of the state machine implies a loss of samples which can compromise the accuracy of the estimation. For these reasons, future developments are automatically outlined:

- Use, during the training phase, of a sensor (like camera) for position measurements of the arm, in order to provide a precise and stable reference for the DVS and to make the design dataset more dense;
- Replacement of the Linux-based hardware device with a Real-Time platform, able to update the state machine instantly and, consequently, provide immediate feedback on the operator's current position.

This involves in lower installation costs and the improvement of performance with respect to the actual sensors.

Appendix A

Python Code

A.1 DVS/MDD Design

```
import keras
import pandas as pd
import numpy as np
import os
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import LSTM, Dense
import random
from tensorflow import set_random_seed
from sklearn.metrics import confusion_matrix

seed = 10

random.seed(seed)
np.random.seed(seed)
set_random_seed(seed)

X_train_container = pd.read_csv(os.path.join("Source","DD_MDD_X.csv"))
T_train_container = pd.read_csv(os.path.join("Source","DD_MDD_T.csv"))
X_test_container  = pd.read_csv(os.path.join("Source","VD_MDD_X.csv"))
T_test_container  = pd.read_csv(os.path.join("Source","VD_MDD_T.csv"))

# Example. Create 3d Tensor
def prepare_data(X_train_container, T_train_container,
```

```
num_of_classes = 2):

# one hot encode
T_train_container = np.asarray(T_train_container)
encoded = to_categorical(T_train_container)

idxs = np.unique(X_train_container['batch'])
X = []
Y = []
for i in idxs:
    x = np.asarray(X_train_container[X_train_container['batch'] == i]
    [['inp1','inp2','inp3']])
    y = encoded[i-1,:]
    X.append(x)
    Y.append(y)
    return np.asarray(X), np.asarray(Y)

X_train, T_train = prepare_data(X_train_container, T_train_container)
X_test,  T_test  = prepare_data(X_test_container, T_test_container)

data_dim = 3
timesteps = None
num_classes = 2

# expected input data shape: (batch_size, timesteps, data_dim)
model = Sequential()
model.add
(LSTM(6, return_sequences= True, input_shape=(timesteps, data_dim)))
# returns last of seq
model.add
(LSTM(6, return_sequences= False, input_shape=(timesteps, data_dim)))
model.add(Dense(num_classes, activation='softmax'))

model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
model.summary()
```

```
model.fit(X_train, T_train, epochs = 160)

T_train_est = model.predict(X_train).argmax(axis = 1)
T_train_ref = T_train.argmax(axis = 1)
T_test_est = model.predict(X_test).argmax(axis = 1)
T_test_ref = T_test.argmax(axis = 1)

acc_train = np.mean(T_train_est == T_train_ref)
acc_test  = np.mean(T_test_est  == T_test_ref)
print("Training Accuracy =
{:.2f}\nTest Accuracy = {:.2f}".format(acc_train, acc_test))

cm = confusion_matrix(T_test_est , T_test_ref)
print(cm)

model.save('DVS_MDD.h5')
```

A.2 DVS/R Design

```
import keras
import pandas as pd
import numpy as np
import os
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import LSTM, Dense
import random
from tensorflow import set_random_seed
from sklearn.metrics import confusion_matrix

seed = 10

random.seed(seed)
np.random.seed(seed)
set_random_seed(seed)

# Load Training Data
X_train_container =
```

```
pd.read_csv(os.path.join("Source","DD_Reset_X.csv"))
T_train_container =
pd.read_csv(os.path.join("Source","DD_Reset_T.csv"))
X_test_container  =
pd.read_csv(os.path.join("Source","VD_Reset_X.csv"))
T_test_container  =
pd.read_csv(os.path.join("Source","VD_Reset_T.csv"))

# Create 3d Tensor
def prepare_data(X_train_container, T_train_container,
num_of_classes = 2):

# one hot encode
T_train_container = np.asarray(T_train_container)
encoded = to_categorical(T_train_container)

idxs = np.unique(X_train_container['batch'])
X = []
Y = []
for i in idxs:
    x = np.asarray(X_train_container[X_train_container['batch'] == i]
    [['inp1','inp2','inp3']])
    y = encoded[i-1,:]
    X.append(x)
    Y.append(y)
    return np.asarray(X), np.asarray(Y)

X_train, T_train = prepare_data(X_train_container, T_train_container)
X_test,  T_test  = prepare_data(X_test_container, T_test_container)

data_dim = 3
timesteps = None
num_classes = 2

# expected input data shape: (batch_size, timesteps, data_dim)
model = Sequential()
model.add
(LSTM(6, return_sequences= True, input_shape=(timesteps, data_dim)))
# returns last of seq
```

```
model.add
# add another layer
(LSTM(6, return_sequences= False, input_shape=(timesteps, data_dim)))
model.add(Dense(num_classes, activation='softmax'))

model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
model.summary()


model.fit(X_train, T_train, epochs = 160)

T_train_est = model.predict(X_train).argmax(axis = 1)
T_train_ref = T_train.argmax(axis = 1)
T_test_est = model.predict(X_test).argmax(axis = 1)
T_test_ref = T_test.argmax(axis = 1)

acc_train = np.mean(T_train_est == T_train_ref)
acc_test  = np.mean(T_test_est  == T_test_ref)
print
("Training Accuracy =
{}\nTest Accuracy = {}".format(acc_train, acc_test))

cm = confusion_matrix(T_test_est, T_test_ref)
print(cm)

model.save('DVS_R.h5')
```

A.3 State Machine DVS/MDD

```
from keras.models import load_model
#from keras.utils import to_categorical
import os
import pandas as pd
import numpy as np
import datetime
import time # TODO: REMOVE IT
```

```
class UAT_SM:

    def __init__(self, val_twar, val_tman):
        """
        Init UAT State Machine with its default values (SAFE)
        Possible values = {'SAFE', 'WARNING', 'ALARM', 'MAINTENANCE'}
        Then, Load pre-learned DVS-UAT and DVS-R
        val_twar is the maximum window to wait for reset
        in warning (seconds)
        val_tman is the maximum window to wait for reset
        in maintenance (seconds)
        """
        self.status = 'SAFE'

        dvs_path = os.path.join("DVS", "DVS_UAT.h5")
        print("Loading DVS from " + dvs_path)
        self.dvs_uat = load_model(dvs_path)

        dvs_path = os.path.join("DVS", "DVS_R.h5")
        print("Loading DVS from " + dvs_path)
        self.dvs_r = load_model(dvs_path)

        # Set Normalization parameters
        self.xmean_MDD  =
        [-0.234176602927583, -0.451919566602261, 0.352203552565829];
        self.xstd_MDD  =
        [0.628305948458802, 0.472688551239224, 0.612469411302056];
        self.ymean_MDD = 0;
        self.ystd_MDD  = 1;

        self.xmean_R =
        [-0.150430213329108, -0.519511740228644, 0.401992645711142];
        self.xstd_R  =
        [0.542904989123885, 0.384523788870667, 0.512241237085408];
        self.ymean_R = 0;
        self.ystd_R  = 1;

        # Init warning and maintenance time window
```

```
        self.val_twar  = val_twar
        self.val_tman = val_tman

        # Init warning and maintenance time counters
        self.twar = None
        self.tman = None

        # Init flag for MDD and R filtering

        self.U = 0
        self.Res = 0


    def __str__(self):
        return "MDD SM STATE = {}".format(self.status)

    def dvs_uat_prediction(self, X):
        """
        DVS-MDD prediction
        X must be a 3d (1,N,3) numpy ndarray representing
        a sequence to classify
        """

        T = self.dvs_uat.predict(X).argmax(axis = 1)
        return T

    def dvs_r_prediction(self, X):
        """
        DVS-R prediction
        X must be a 3d (1,N,3) numpy ndarray representing
        a sequence to classify
        """

        T = self.dvs_r.predict(X).argmax(axis = 1)
        return T

    def time_elapsed(self, t2, t1):
        df = t2 - t1
        return df.total_seconds()
```

```
def map_norm_MDD(self,x, i):
    """
    Data Normalization
    """
    x = [ (xx - self.xmean_MDD[i]) *
    (self.ystd_MDD/self.xstd_MDD[i]) +
    self.ymean_MDD for xx in x ]
    return x

def map_norm_R(self,x, i):
    """
    Data Normalization
    """
    x = [ (xx - self.xmean_R[i]) *
    (self.ystd_R/self.xstd_R[i]) + self.ymean_R for xx in x ]
    return x

def update(self, inp1, inp2, inp3, val_twar = None, val_tman
    = None, resetAlarm = False):
    """
    Update UAT SM state and return a feedback
    Feedback: Updated SM state or Error Msg
    """
    if len(inp1) == len(inp2) == len(inp3):

        if val_twar is not None:
            self.val_twar = val_twar

        if val_tman is not None:
            self.val_tman = val_tman

        if max(inp3) > 1.5:
        # Set flag for R to 1 in order to impose R=0
           self.Res = 1

        else:
            self.Res = 0
```

```
n, m = len(inp1), 3
# Concat inp1, inp2 and inp3 into a 3d (1,N,3) ndarray
X_MDD = np.zeros((1,n,m))
X_MDD[:,:,0] = self.map_norm_MDD(inp1, 0)
X_MDD[:,:,1] = self.map_norm_MDD(inp2, 1)
X_MDD[:,:,2] = self.map_norm_MDD(inp3, 2)

X_R = np.zeros((1,n,m))
X_R[:,:,0] = self.map_norm_R(inp1, 0)
X_R[:,:,1] = self.map_norm_R(inp2, 1)
X_R[:,:,2] = self.map_norm_R(inp3, 2)
UAT = self.dvs_uat_prediction(X_UAT)

if (self.Res) == 1:
    R = [0]
else:
    #if np.amax(X_R[:,:,1])
    R = self.dvs_r_prediction(X_R)

print(self.Res)
print("Prediction MDD = {} R = {}".format(MDD[0],R[0]))

####### Logic Implementation ##########################
# Transitions:
# Safe:
# - 1 Safe
# - 2 Reset the Maintenance State
# - 3 Manual Reset of Alarm State
# Warning:
# - 1 MDD == 1 from Safe State
# - 2 tm = 0 after Maintenance State
# Alarm:
# - 1 tw = 0 after Warning State
# - 2 No Reset Alarm
# Maintenance:
# - 1 Reset from Warning State
# - 2 Mantainance and tm is not 0
```

```
if self.status == "SAFE":
    # MDD and not Reset status becames WARNING
    if MDD[0] == 1:
        if R[0] == 0:
            # If MDD and not Reset,
            # Status becames WARNING
            self.twar = datetime.datetime.now()
            # Set initial war timer
            self.status = "WARNING"
        elif R[0] == 1:
            # If R Then Status becames MANTAINANCE
            self.tman   = datetime.datetime.now()
            # Set initial man timer
            self.status = "MAINTENANCE"

elif self.status == "WARNING":
    t = datetime.datetime.now()
    print("I've been in WARNING for
    {:.2f}".format(self.time_elapsed(t, self.twar)))
    if self.time_elapsed(t, self.twar) >=
    self.val_twar:
        # If still not R until val_twar status
        becames ALARM
        self.status = "ALARM"
    elif R[0] == 1:
        # Else, If R Then Status becames MANTAINANCE
        self.tman   = datetime.datetime.now()
        # Set initial man timer
        self.status = "MAINTENANCE"
        self.twar   = None # Clear initial war timer

elif self.status == "MAINTENANCE":
    t = datetime.datetime.now()
    print("I've been in MAINTENANCE since {:.2f}
    seconds".format(self.time_elapsed(t, self.tman)))
    if self.time_elapsed(t, self.tman) >=
    self.val_tman:
        # If still not R until val_tman status
        becames WARNING again
```

```
                    self.twar   = datetime.datetime.now()
                    # Set initial war timer
                    self.status = "WARNING"
                    self.tman   = None
                elif R[0] == 1:
                    # Else, If Reset Then Status becames SAFE
                    self.status = "SAFE"
                    self.tman = None # Clear initial man timer

            elif self.status == "ALARM":
                if resetAlarm:
                    self.status = "SAFE"
                    self.twar = None # Clear initial war timer
                    self.tman = None # Clear initial man timer

            ################################################

            # Status update
            feedback = self.status

        else:
            feedback = 'Exception. Input lengths does not match'

        return feedback

if __name__ == '__main__':

    print("UAT State Machine Simulation...")
    uat_sm = UAT_SM(5,5)
    print(uat_sm)

    # Real-time SM simulation using pre-stored data in a csv
    print("Loading Examples...")
    data_path = os.path.join("Source","test5posizioni5.csv")
    data_df   = pd.read_csv(data_path)
    unq_btcs  = sorted(list(set(data_df['batch'])))
    print("Loaded {} different sequences.".format(len(unq_btcs)))

    for batch in unq_btcs:
```

```
        if batch == 2 or batch == 3:
            time.sleep(6)

        print("Loading from batch {}".format(batch))
        inp1 = list(data_df[data_df['batch'] == batch]["inp1"])
        inp2 = list(data_df[data_df['batch'] == batch]["inp2"])
        inp3 = list(data_df[data_df['batch'] == batch]["inp3"])

        if batch == 11:
            feedback = uat_sm.update(inp1, inp2, inp3,
            resetAlarm = True)
        else:
            feedback = uat_sm.update(inp1, inp2, inp3)
        print("Status = {}".format(feedback))
```

A.4 DVS/WAD

```
from matplotlib import pyplot as plt
import numpy as np

from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from sklearn.pipeline import Pipeline
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import RFE
from sklearn.feature_selection import VarianceThreshold
from sklearn.preprocessing import PolynomialFeatures
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix
from sklearn.externals import joblib
import pickle
import utils
from utils import load_data

seed = 1
np.random.seed(seed) # Fix seed
```

```
# Design a Neural Network using scikit-learn

if __name__ == "__main__":

    Xtrain, Ttrain, Xtest, Ttest = load_data() ’

    classes = 3 # Equal to output (0,1,2)
    m = np.shape(Xtrain)[1] #Number of Columns

    pipeline = Pipeline(
        steps=[
            (’Normalization’, StandardScaler(copy=True,
            with_mean=True, with_std=True)),
     #       (’FeatSel’, PolynomialFeatures(2)), #Feature selection
#           (’Classification’, GradientBoostingClassifier(
#               learning_rate = 0.75,
#           # Increasing learning_rate, more fast and less
            # precise to find the minimum of the loss function
#               n_estimators = 20, #number of models in ensamble
#               random_state=seed
#           ))
            (’Classification’, MLPClassifier(
                activation = ’relu’,
                solver = ’lbfgs’,
                alpha = 1e-5,     # L2-Regul.
                max_iter = 150, #iteration
                hidden_layer_sizes = (25,25),
                #number of layers and neurons
                early_stopping = True,
                # Stopping when validation’s score does not
                improve anymore
                #verbose = True,
                random_state=seed
            ),

            )
        ]
    )
```

```
#     '''
#     With RandomizedSearchCV I can select a range of parameters
      that i want to tune.
#     The algorithm help to find the best combination of the
#     selected parameters.
#     I can use also GridSearchCv that explores all the possible
#     combinations.
#     Randomized only the possible best ones. More Fast but less
#     precise
#     It perform Cross Validation on the training Data

#    parameters = {
##         "Classification__alpha" : [1e-5, 1e-4, 1e-3],
#         "Classification__hidden_layer_sizes" : [
#             (60,50,50),
#             (80,70),
#             (10, 7, 2),
#             (10,6,6),
#             (20,20,20),
#             (50,47),
#             (35,27,20),
#             (14,6),
#             (15,10,6),
#         ]
#    }
##
#    grid = RandomizedSearchCV(pipeline, parameters)
#    Network and features which i waant to tune

    history = pipeline.fit(Xtrain, Ttrain.ravel())

    # Training Evaluation
    train_est = pipeline.predict(Xtrain)
    train_acc = np.mean( train_est == Ttrain.ravel() )
    print("Train Accuracy = {:.2f}".format(train_acc))

    # Test Evaluation
    test_est = pipeline.predict(Xtest)
    test_acc = np.mean( test_est == Ttest.flatten() )
```

```
    print("Test Accuracy = {:.2f}".format(test_acc))

    cm = confusion_matrix(Ttest.flatten(), test_est)
    print(cm)

    filename = 'DVS_WAD.h5'
    pickle.dump(pipeline, open(filename, 'wb'))
```

A.5 State Machine WAD

```
from keras.models import load_model
#from keras.utils import to_categorical
import os
import pandas as pd
import numpy as np
import datetime
import time
import pickle

class ARMTR_SM:

    def __init__(self, val_treset):
        """
        Init UAT State Machine with its default values (SAFE)
        Possible values = {'SAFE', 'WARNING', 'ALARM', 'RESET'}
        Then, Load pre-learned DVS-ARMTR
        val_treset is the maximum window without Reset gesture
        """
        self.status = 'SAFE'

        #dvs_path = os.path.join("DVS", "DVS_ARMTR.sav")
        #print("Loading DVS from " + dvs_path)
        self.dvs_armtr = pickle.load(open('DVS_ARMTR.h5', 'rb'))

        # Init warning and maintenance time window
        self.val_treset  = val_treset
```

```
    # Init warning and maintenance time counters
    self.treset = None
    self.tstart= datetime.datetime.now()

def __str__(self):
    return "SM STATE = {}".format(self.status)

def time_elapsed(self, t2, t1):
    df = t2 - t1
    return df.total_seconds()

def dvs_armtr_prediction(self, X):
    """
    DVS-ARMTR prediction
    """
    T = self.dvs_armtr.predict(X)
    return T

def update(self, Inp, val_treset = None, ManualReset = True):
    """
    Update UAT SM state and return a feedback
    Feedback: Updated SM state or Error Msg
    """
    # Concat inp1, inp2 and inp3 into a 3d (1,N,3) numpy ndarray
    X_ARMTR = np.zeros((1,m))
    X_ARMTR = Inp

    if len(X_ARMTR) == 1:

        if val_treset is not None:
            self.val_treset = val_treset

        #print(X_ARMTR)
        # Get Predictions
        ARMTR = self.dvs_armtr_prediction(X_ARMTR)
        #print("Prediction ARMTR = {}".format(ARMTR[0]))

        ####### Logic Implementation ####################
        # Transitions:
```

```
# Safe:
# - 1 Safe Area
# - 2 From Warning to Safe
# - 3 From Alarm to Safe
# Warning:
# - 1 From Safe to Warning
# - 2 From Alarm to Warning
# Alarm:
# - 1 From Warning to Alarm
# - 2 From Safe to Alarm
# Reset:
# - Timer ends, for zero positioning

self.treset = datetime.datetime.now()
d = self.time_elapsed(self.treset, self.tstart)

if d <= self.val_treset:
    print("Prediction ARMTR = {}".format(ARMTR[0]))
    print(d)
    if self.status == "SAFE":
        if ARMTR[0] == 1:
            self.status = "WARNING"
        elif ARMTR[0] == 2:
            self.status = "ALARM"

    elif self.status == "WARNING":
        if ARMTR[0] == 0:
            self.status = "SAFE"
        elif ARMTR[0] == 2:
            self.status = "ALARM"

    elif self.status == "ALARM":
      # If the arm position is in red zone
        if ARMTR[0] == 0:
            self.status = "SAFE"
        elif ARMTR[0] == 1:
            self.status = "WARNING"

elif d > self.val_treset:
```

```
            self.status = "RESET"
            if self.status == "RESET":
                if ManualReset:
                    self.status = "SAFE"
                    self.tstart = datetime.datetime.now()
                    self.treset = None # Clear initial timer
                    print(d)
                else:
                    print("Please Press the Reset Button")

        # Status update
        feedback = self.status

    else:
        feedback = 'Exception. Input lengths does not match'

    return feedback

    if __name__ == '__main__':

print("UAT State Machine Simulation...")
armtr_sm = ARMTR_SM(100)
print(armtr_sm)

# Real-time SM simulation using pre-stored data in a csv
print("Loading Examples...")
data_path = os.path.join("Source","sensordata017_proc_Test.csv")
data_df   = np.asarray(pd.read_csv(data_path))
print("Loaded {} different Sequences.".format(len(data_df)))
n=np.shape(data_df)[0]
m=np.shape(data_df)[1]
i=0

for i in range(n):
    Inp = data_df[i,:].reshape((1,-1))

    feedback = armtr_sm.update(Inp, ManualReset = True)
    print("Status = {}".format(feedback))
```

Bibliography

[1] Borsa Italiana, 14 Giugno 2016.
https://www.borsaitaliana.it/notizie/sotto-la-lente/rivoluzione-252.htm

[2] Disloman Project
http://www.disloman.it/abstract/

[3] Nilsson N.J., Sept 26, 1996
Introduction to machine learning (1996)

[4] Ethem Alpaydin, 2017.
Machine Learning. The New AI-The MIT Press (2017)

[5] Andriy Burkov.
The Hundred-Page Machine Learning Book, 2019

[6] Andrea Missinato, Feb 20, 2018.
https://www.spindox.it/it/blog/ml1-reti-neurali-demistificate/

[7] Sagar Sharma, Sept 6, 2017.
https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

[8] Chi-Feng Wang, Jan 8, 2019.
https://towardsdatascience.com/the-vanishing-gradient-problem-69bf08b15484

[9] Nimesh Sinha, Feb 19, 2018.
https://towardsdatascience.com/understanding-lstm-and-its-quick-implementation-in-keras-for-sentiment-analysis-af410fd85b47

[10] Understanding LSTM.
https://colah.github.io/posts/2015-08-Understanding-LSTMs/

[11] Jayesh Bapu Ahire, Aug 24, 2018.
https://medium.com/coinmonks/the-artificial-neural-networks-handbook-part-1-f9ceb0e376b4