1 2 9 0

UNIVERSIDADE Ð
COIMBRA

Laura Sofia Ferreira Duarte

HUMAN BEHAVIOUR SPOTTING FROM EVENT CAMERA DATA: AN APPROACH FOR COLLABORATIVE MANUFACTURING

Dissertação no âmbito do Mestrado Integrado em Engenharia Mecânica, no ramo de produção e projeto orientada pelo Professor Doutor Pedro Mariano Simões Neto e apresentada ao Departamento de Engenharia Mecânica da Universidade de Coimbra.

October of 2020

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE D COIMBRA

Human behaviour spotting from event camera data: an approach for collaborative manufacturing

Submitted in Partial Fulfilment of the Requirements for the Degree of Master in Mechanical Engineering in the speciality of Production and Project

Reconhecimento de comportamento humano a partir de dados de camara de eventos: uma abordagem de manufatura colaborativa

Author

Laura Sofia Ferreira Duarte

Advisor

Pedro Mariano Simões Neto

Jury

President	**Professor Doutor Samuel de Oliveira Moniz**
	Professor Auxiliar da Universidade de Coimbra
Vowels	**Professor Doutor Mohammad Safeea**
	Professor Convidado da Universidade de Coimbra
	Professor Doutor Nuno Alberto Marques Mendes
	Professor Auxiliar da Universidade Nova de Lisboa
Advisor	**Professor Doutor Pedro Mariano Simões Neto**
	Professor Auxiliar da Universidade de Coimbra

Coimbra, October of 2020

Any sufficiently advanced technology is indistinguishable from magic.

Arthur C. Clarke, in *Profiles of the Future*, 1973

ACKNOWLEDGEMENTS

I want to express my gratitude towards all people who, either directly or indirectly, helped make this study come to fruition.

I want to thank my advisor, Dr. Pedro Neto, for accepting me, keeping me motivated, always being reachable and offering great suggestions. Also, I want to acknowledge Dr. Nuno Mendes for his practical knowledge and recommendations.

I want to express gratitude towards my parents and my brother for always being present whenever I needed emotional or work-related support. At last but not least, I also like to thank all my friends who reminded me to unwind and have fun.

Thank you,

Laura Duarte

ABSTRACT

Continuous and real-time action/gesture recognition is one of the main issues of collaborative robots and their successful application in different domains, namely in manufacturing. Nevertheless, the reliable and real-time recognition of actions/gestures in unstructured environments is still difficult to achieve. When it comes to manufacturing, the recognition of human actions related to basic manufacturing tasks (grab, lift, screw, etc.) is still a challenge. This study proposes to model and analyse events data aiming to recognize human actions related to basic manufacturing tasks. Noise reduction techniques, namely cropping, will be applied to data captured from an event camera (EC), eliminating most of the noise originated by shadows and body motion. The EC data grid is scanned to identify the outlines that define the region of interest (ROI). The number of captured events per frame is optimized to properly capture both slow and fast human motion. Linear and non-linear Data Dimensionality Reduction (DDR) techniques such as Principal Component Analysis (PCA) and kernel PCA are applied. The classification of tasks is achieved recurring to Long-term Recurrent Convolutional Networks (LRCNs) which combine the classification of spatial features using Convolutional Neural Networks (CNNs) and the temporal features using Long Short-Term Memory networks (LSTMs). Experimental tests were conducted using the new ECmanufacturing20 dataset composed by EC data representing 10 different classes of basic assembly manufacturing tasks. The classification models show an accuracy of 46,7% when using the original data, 63,3% using noise reduction, 26,7% when using PCA and 36,7% applying kernel PCA to data. The application of noise reduction techniques showed a positive effect on classification accuracy, increasing it by about 17 percentage points.

Keywords Data Dimensionality Reduction, Deep Neural Networks, Collaborative Manufacturing, Noise Reduction, Event Camera.

RESUMO

O reconhecimento de ações/gestos em contínuo e em tempo-real é um dos principais problemas enfrentados pela robótica colaborativa, afetando o sucesso da sua aplicação em vários domínios, nomeadamente na indústria de manufatura. O reconhecimento fiável de ações/gestos num meio não estruturado e em tempo-real continua a ser difícil de alcançar. Quanto à manufatura, o reconhecimento de ações humanas relacionadas com tarefas básicas de manufatura (como agarrar, levantar, aparafusar, etc.) continua a ser um grande desafio. Este estudo propõe modelar e analisar dados de eventos com o objetivo de reconhecer ações humanas relacionadas com tarefas básicas de manufatura. Técnicas de redução de ruído, nomeadamente *cropping*, são aplicadas a dados capturados por uma câmara de eventos (EC), eliminando a maioria do ruído originado por sombras e movimento do corpo. A grelha de dados da câmara de eventos é examinada para identificar os contornos que definem a região de interesse (ROI). O número de eventos capturados por *frame* é otimizado para capturar devidamente tanto os movimentos lentos como os movimentos rápidos dos humanos. Técnicas de redução de dimensionalidade de dados (DDR) lineares e não lineares como *Principal Component Analysis* (PCA) e *kernel* PCA são utilizadas. A classificação de tarefas é concretizada aplicando *Long-term Recurrent Convolutional Networks* (LRCNs) que combinam a classificação de características espaciais usando *Convolutional Neural Networks* (CNNs) e características temporais usando *Long Short-Term Memory networks* (LSTMs). Testes experimentais foram realizados usando o novo ECmanufacturing20 *dataset* constituído por dados da câmara de eventos que representam 10 classes diferentes de tarefas de montagem básicas utilizadas em manufatura. Os modelos de classificação demonstram uma fiabilidade de 46,7% quando se usou os dados originais, 63,3% quando a redução de ruído foi usada, 26,7% quando se usou PCA e 36,7% quando se aplicou *kernel* PCA aos dados. A aplicação de técnicas de redução de ruído teve um efeito positivo na fiabilidade da classificação, aumentando-a cerca de 17 pontos percentuais.

Palavras-chave Redução de Dimensionalidade de Dados, Redes Neurais Profundas, Manufatura Colaborativa, Redução de Ruído, Câmara de Eventos.

Contents

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS/ABBREVIATIONS

CNN – Convolutional Neural Network

DDR – Data Dimensionality Reduction

DNN – Deep Neural Network

EC – Event Camera

HRI – Human-Robot Interaction

LRCN – Long-term Recurrent Convolutional Network

LSTM – Long-Short Term Memory

NLPCA – Non-Linear Principal Component Analysis

NR – Noise Reduction

PCA – Principal Component Analysis

RNN – Recurrent Neural Network

ROI – Region of Interest

1. INTRODUCTION

Over the years, robotics has been inspired by biological systems, both at the level of mechanisms, sensory systems, and reasoning. Visual sense allows humans to capture information about objects, the space that surrounds them, distances, among other information. With this in mind, the robotics community has been putting a lot of effort into developing reliable vision systems through which robots can perceive the real world. There has been given special attention to improving computer vision sensing, mainly through the development of different sensors and better object/action recognition algorithms. Reliable vision sensing is essential as it allows the robot to recognize humans and objects, providing it with the ability to interact with them.

In robotics, vision sensing has been used in the recognition of objects, human actions, and behaviours, which are then applied in autonomous robots, for example, autonomous driving. It is also used for Human-Robot Interaction (HRI) which is becoming an increasingly relevant application, as seen in collaborative environments. This will be the focus of this work.

Until recently, robots were only used for predictable and repeatable processes, working automatically or semi-automatically. Now, the focus shifted to a more flexible approach, which enhances the interaction between robots and humans through a co-working partnership. This has the advantage of joining the coordination and adaptability of humans with the robot’s accuracy and ease of execution of repeatable tasks. Thus, there is a need to establish a system that allows for natural and intuitive HRIs. Human actions and gestures are a common type of interface, as they are intuitive to use and allows for a wide range of commands to interact with humans. To be fully operable, it needs a way to obtain real-world gesture data and recognize the gestures being made.

The data needed are usually acquired by using one of the most prevalent vision sensors, frame-based cameras. The two main types are the charge-coupled device (CCD) and the complementary metal-oxide semiconductor sensor (CMOS sensor), being the CMOS more commonly used. Although this type of sensor has been proven viable, more suitable types of sensors are available, like wearables and Event Cameras (ECs).

One of the main difficulties associated with action/gesture-based HRI is the continuous recognition, requiring real-time recognition of human behaviour. Thus, the system must be able to recognize the actions/gestures quickly and reliably. Another challenge is the environment. The gesture may have a busy background space which leads to capturing mostly unnecessary information and making its class difficult to identify. Lighting conditions and fast-moving objects also provide additional difficulty in recognition.

Lastly, there is, also, interest in reducing the amount of data that needs to be processed to recognize a certain action/gesture, to reduce overall system latency. Even when there is a low data rate, there is still a need to reduce the data as much as possible to allow for a quick feedback loop between the human and the robot.

ECs are an emerging technology that presents great potential for HRI purposes. The main advantages of the EC are its great adaptability to lighting conditions and high-speed movements, its disregard for static backgrounds, and its relative low latency and high temporal resolution.

A Noise Reduction (NR) algorithm is used to reduce unnecessary noise existent within the EC data. The use of Data Dimensionality Reduction (DDR) is aimed at providing a low-dimensional representation of the data generated by an EC, to reduce the time and resources needed to process a gesture. Both a linear DDR, PCA, and a non-linear DDR, Kernel PCA, are tested.

For image classification, a Deep Neural Network (DNN) is used. This DNN is comprised of a CNN, for feature extraction, and an LSTM, to learn temporal sequences. Both the original, NR, and DDR classification results are compared against each other to verify which of the methods performs better in classification. A simplified schematic of the proposed approach is shown in Figure 1.1.

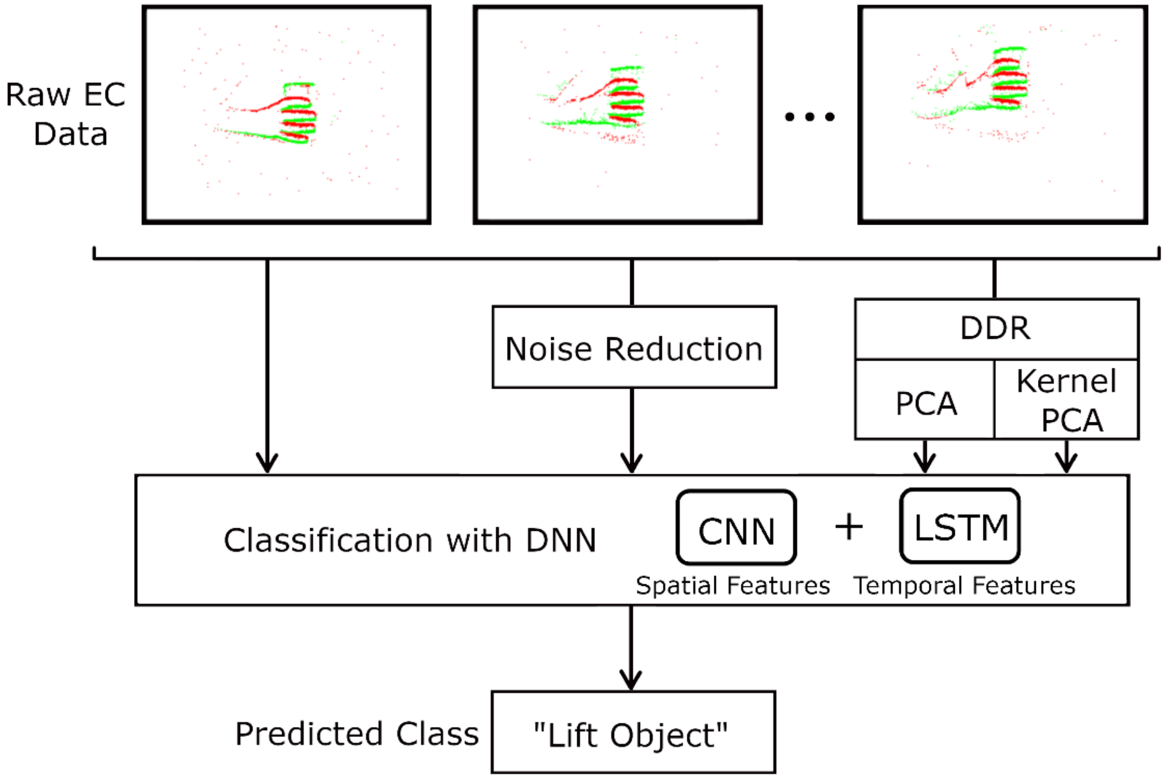


Figure 1.1: Schematic of the proposed system for gesture classification.

2. STATE OF THE ART

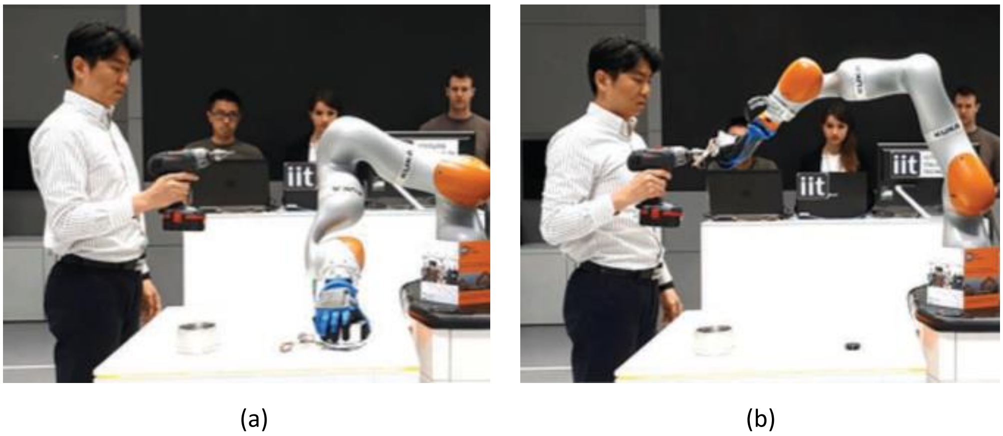
This section presents a brief analysis of the current state of knowledge about the topics relevant to this work.

2.1. Gesture-based HRI

2.1.1. Main Concepts

A gesture is a form of non-verbal communication in which specific messages are communicated through the movement of hands, face, or other parts of the body. Gestures are an integral part of all collaborative assembly processes. While most frequently used to enhance speech-based communication, they can also be used to completely replace some spoken commands [1]. A gesture-based HRI system should operate solely on gestures, without the need for complementary speech recognition.

A gesture-based HRI system grants the user a way to communicate to the robot how to operate, while also providing additional information on where the action should take place in the physical world, e.g. by pointing. Additional context can be given by using a specific tool, as it is the case in Figure 2.1. The robot first detects the tool in the user's hand and picks up the associated part, Figure 2.1(a). The robot then identifies the location of the tool, adapting its position for the user to be in an ergonomic position, Figure 2.1(b).



(a) (b)

Figure 2.1. Gesture-based HRI with context conveyed through the usage of a drill [2].

Communication through gestures is innate, as it is used in daily human-human interactions. The user, by knowing intuitively how to interact with the robot, has little to learn about the HRI interface. This makes the robot accessible to use, which in turn allows the user to better focus on the task itself [3].

2.1.2. Hand Gestures

Hand gestures are at the core of gesture-based HRI. They can condense a lot of information into a single hand motion or position. Action and location are conveyed through the hand's shape, movement and position. There are two main types of gestures, Static Gestures (SGs) and Dynamic Gestures (DGs). The SGs convey messages with a specific static hand position while DGs express their messages through the hand's movement, as shown in Figure 2.2.

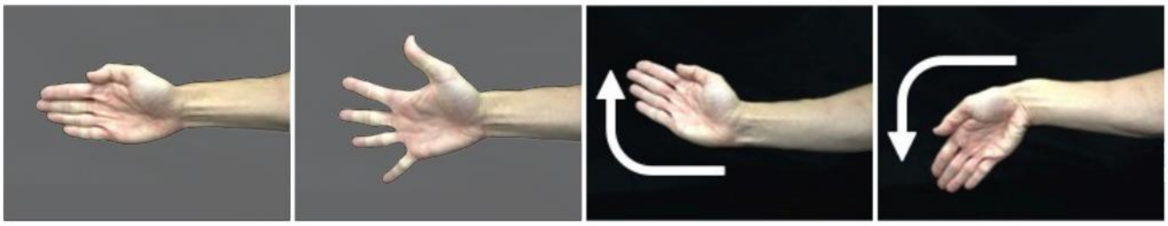


Figure 2.2. Examples of SGs and DGs. Adapted from [4].

Dynamic gestures usually consist of 3 phases, as shown in Figure 2.3: preparation, stroke, and retraction [5]. Starting at a resting position, the user moves their hand into position, executes the intended gesture, and finally returns it to resting position. In the case of a sequence of gestures, the preparation for the next gesture will be made instead of retraction. It is essential to pause before and after the gesture for it to be more easily recognized. This is usually done by assuming a resting position.

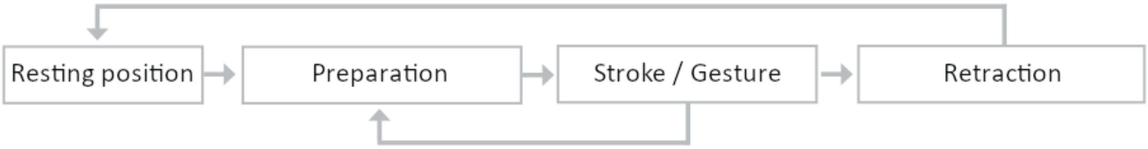


Figure 2.3. Structure of a Dynamic Gesture.

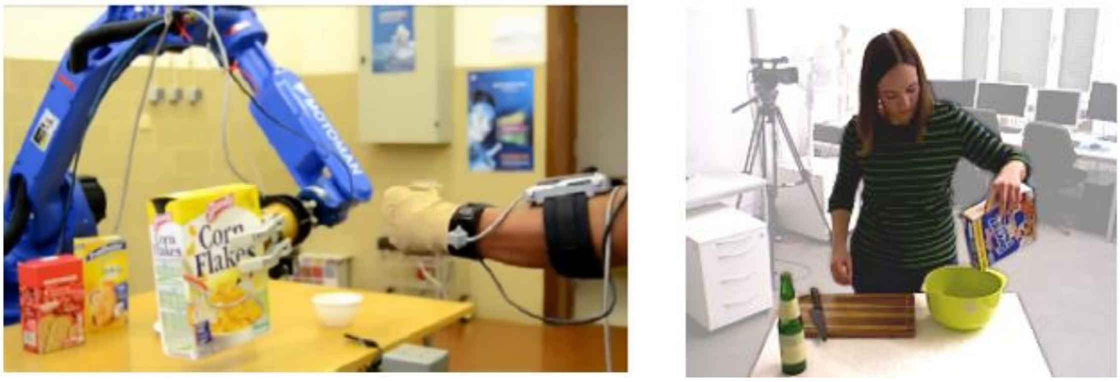
In the gesture phase, the actual gesture is executed with the intent of conveying a specific meaning. In the preparation and retraction phases, while hand movement occurs, no communication is implied. This is called Movement Epenthesis (ME), which involves a change in hand shape and movement without meaning. There are many ways of dealing with ME [6] but, in this work, only the gesture phase of the DGs will be studied.

A library of gestures is a collection of predefined gestures, either SGs, DGs, or both. The gesture library must be selected with caution, as the gestures need to be distinguishable from one another. Although a lot of different gestures can be made, only some can be consistently recognized by robots and humans alike [1]. All hand gestures that constitute a gesture library should be easy to understand and replicate for users without knowledge of sign language. The most relevant gestures for HRI are of the referencing type, such as "This one!" and "From here to here!" or terminating gestures like "Stop" and "No" [1].

2.1.3. Applications

Gesture-based HRI systems can generally be one of two types: Wearable or Vision-based. Wearable systems are characterized by relying on wearable equipment, usually hand-worn, which tracks the hand's movement. Vision-based systems perceive gestures using a vision-based sensor.

Wearables can be used for the recognition of both SGs and DGs. They have exhibited the ability to command, through gestures, diverse types of robots such as a manipulator arm, a ground robot, and a robotic hand [4][7][8], Figure 2.4(a). Wearable technology has also been used to perform demonstrations of gestures to teach a humanoid robot to reproduce such gestures with a smooth and human-like feel to it [9].



(a) (b)

Figure 2.4. Real-life applications of gestures: (a) Command a robot to grab a cereal box [8]; (b) Recognize the pouring of cereal [10].

Typical Vision-based systems use frame-based cameras to identify both SGs and DGs. They have the advantage of, in addition to the gesture itself, being able to discern the objects which the user is interacting with. It has been used, for example, on continuous gesture recognition for aircraft handling [3] and recognition of bimanual action sequences [10] such as cooking, disassembling a hard drive, and preparing breakfast cereals, Figure 2.4(b).

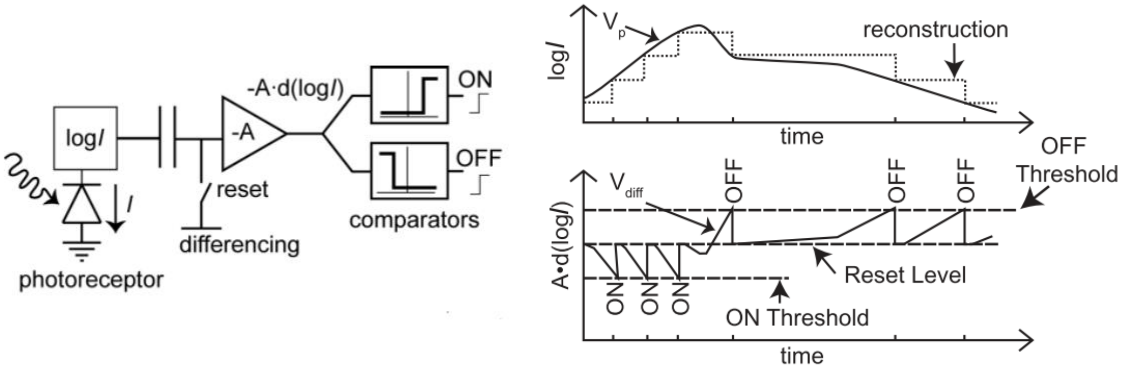
Although both gesture recognition approaches share characteristics [11], wearables are susceptible to slippage issues [4] and need worn equipment, which is an inconvenience to the operator. Typical frame-based Vision-based systems, meanwhile, have issues in recognizing gestures from all directions [11], due to possible occlusions, and in dealing with background noise. Also, the vision-based solutions require large amounts of data for the recognition of a simple pattern, demanding high-performance processors. As such, there is a need for a different kind of sensing technology to empower gesture-based HRI that does not suffer from these drawbacks, like the Event Camera.

2.2. Event Camera

2.2.1. Working Principles

Event-based computation is characterized by its sparse asynchronous events. It is used by biological systems, like our brain and eyes, to guarantee an efficient real-time performance [12]. Research into new types of artificial circuits, inspired by these biological systems, resulted in an event-based sensor named Event Camera (EC).

An EC, such as the Dynamic Vision Sensor (DVS), responds to changes in brightness (log intensity) of the captured scene through its distinctive pixel architecture, Figure 2.5(a). The log intensity of each pixel is continuously compared to an “ON” and “OFF” threshold which, when exceeded, generates an asynchronous event, Figure 2.5(b).



(a) (b)

Figure 2.5. DVS Characteristics: (a) Simplified pixel schematic. Adapted from [13]; (b) Principle of operation [14].

This event contains information about the pixel coordinates, x and y, the timestamp of the event, ts, and the sign of the occurred brightness change, also known as polarity, pol. Each event is usually expressed as a 1x4 array such as in (1), where the index i represents the unique number of the event.

$$Ev_i = (x, y, pol, ts) \tag{1}$$

To transmit this information, while preserving the low latency of the EC’s asynchronous output, the Address-Event Representation (AER) protocol is used. It enables the asynchronous communication of events by relaying the address of the pixel that sent the event [15].

2.2.2. Advantages

The EC has many features that make them better sensors, in comparison to frame-based cameras, to empower numerous robotics and artificial intelligence applications, Figure 2.6. The most relevant are discussed in the following section.

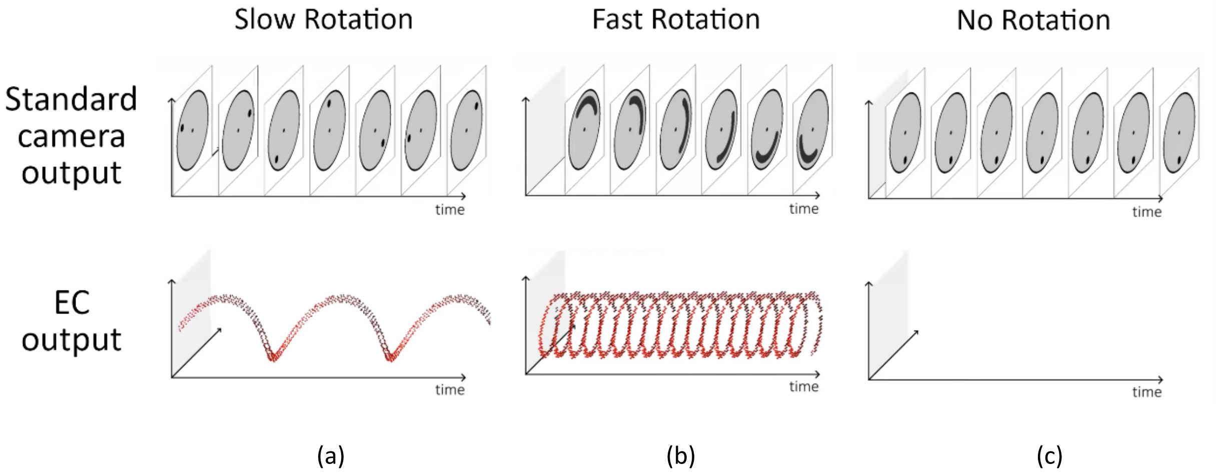


Figure 2.6. Differences between CMOS sensor and EC outputs of a rotating disk [16]:
(a) Slow Rotation; (b) Fast Rotation; (c) No Rotation.

2.2.2.1. Low data rate

Considering the camera is in a fixed position, a frame-based camera captures all the information contained within a given frame. An EC, only captures the movement present in the scene, ignoring the background information, as shown in Figure 2.6(a). This is useful for gesture detection, as it reduces data size and facilitates motion tracking.

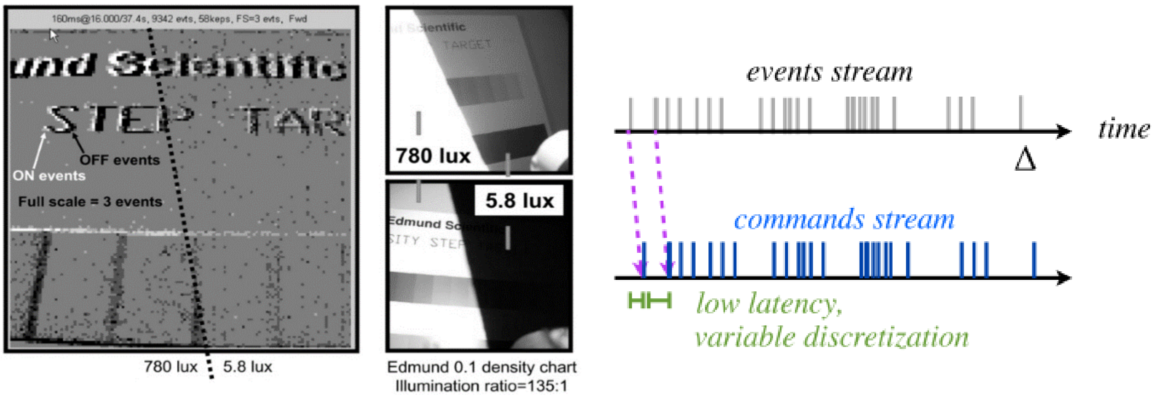
The data rate of a sensor, usually expressed in kB/s or MB/s, is the rate at which the sensor's data are transmitted per second. Even though the EC has a high acquisition rate, Figure 2.6(b), each event only accounts for about 2 bytes on average, making the data rate of ECs typically orders of magnitude lower than that of a frame-based camera [14],[15]. In a static scene, as there is no change in the environment, ECs will not capture any data at all, as seen in Figure 2.6(c). This shows that the data rate of ECs scales dynamically with actual demand [12].

The low data rate of the EC leads to the typical low power consumption of about $30\ mW$ [14],[17], due to subsequent low data processing. While the amount of power consumption is not usually a concern in a fixed setup, it is crucial when employed on mobile devices such as drones. By being battery-powered, these have a limited amount of energy available, benefiting from low power expenses.

2.2.2.2. High Dynamic Range

Dynamic range, in the context of vision sensors, describes the range of brightness values the sensor can detect. A high dynamic range allows for better performance in unsupervised environments with inconsistent and/or high contrast lighting conditions, as in Figure 2.7(a). It also enables indoor and outdoor operation, without the need to change any parameters [16].

The high dynamic range of an EC is, on average, of about 120 dB. Aside from certain frame-based cameras that are designed for high dynamic range purposes, like [18], they usually have a lower dynamic range of about 60 dB. These specialized cameras do not, however, have the other key advantages associated with ECs.



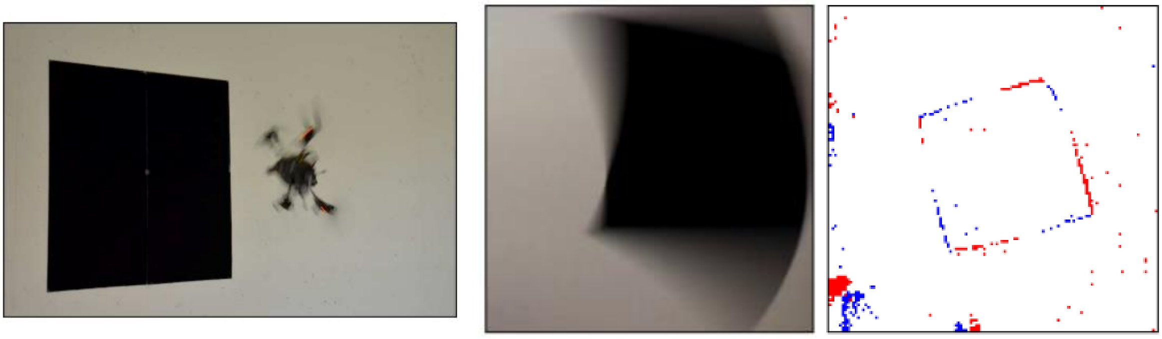
(a) (b)

Figure 2.7. EC features: (a) High Dynamic Range. Adapted from [17]; (b) Low Latency [19].

2.2.2.3. Low Latency and High temporal resolution

Latency, also known as delay, is the time interval occurring between the sensor's detection of scene change and the ensuing response to that change. In ECs this delay is very small, as seen in Figure 2.7(b), generally lasting less than 100 μs [14], [17]. This makes them ideal for real-time applications like the detection of gestures in gesture-based HRI systems. These systems, to be considered responsive, should manage to give a reply within 100–200 ms [20]. Low latencies promote high temporal resolutions. By lowering the delay between the stimuli and the reply, the sensor can record data at a much faster pace. This enables high data acquisition rates of about 1 million events/second [14].

In frame-based architectures, frames are outputted at a consistent time interval, measured in frames/second. ECs, as data rate scales dynamically with its demand, have small but variable time intervals between retrieved events, as seen in Figure 2.7(b). When fast movements occur, frame-based cameras tend to present high motion blur in its output frames which limits the camera's ability to portray the motion. A DVS, however, manages to detect fast movements accurately due to its high temporal resolution. This difference is made evident in Figure 2.8.



(a) (b)

Figure 2.8. A quadrotor, equipped with a CMOS sensor and a DVS, performing a flip [16]: (a) CMOS output; (b) DVS Output.

2.2.3. Processing EC data

The EC is a vision-based movement sensor mostly used for motion tracking purposes. As the events are asynchronously generated by the EC, practical ways of processing this kind of data must be developed.

The simplest method is that of accumulating events over a fixed interval of time into a frame, called an integrated image [16]. Owing to this, already developed frame-based tracking and recognition algorithms can be used. It does, however, limit some of the EC's capabilities, like its low latency and high temporal resolution. Accumulating events is a solution fit for tracking movements that contain one or multiple complex shapes to be tracked. It has been successfully applied to the acquisition of traffic data [21].

Another method relies on an integrated image followed by continuously updating the image as soon as new events arrive. While more complex in its implementation, needing dedicated techniques like the Hough Transform [13],[16], it enables extremely low latencies needed to track high-speed movements. This approach is generally used to track simple shapes. This solution is mostly used in high-speed robotics, like quadrotors, to be able to track its rotation during a flip [19],[16]. This approach has also been used to successfully control an actuated table to balance a pencil [13].

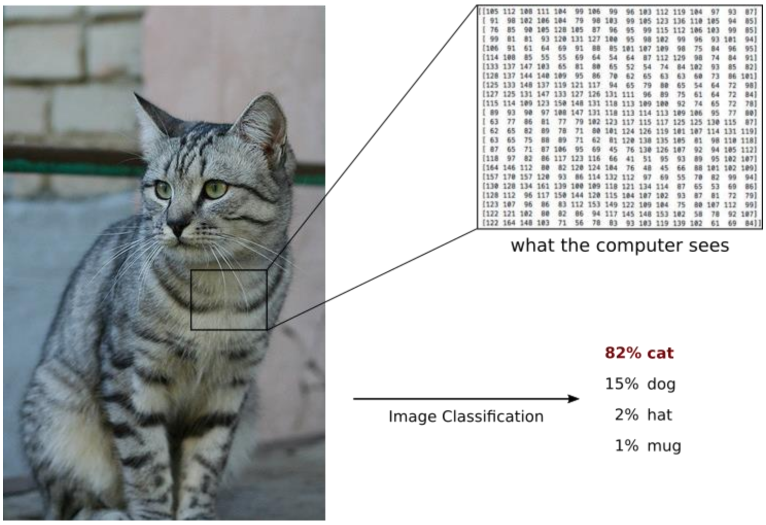
Although continuously updating the image would provide low latency, it would be difficult to use it for gesture tracking. This is in part due to the inconsistency of hand shape, derived from different users or the angle of the hand while facing the EC. It also has the big disadvantage of not being suitable for object recognition.

The discretization of the continuous output of ECs into integrated images, although limiting temporal resolution, makes it easier to comprehend and process the obtained data. This approach not only allows better tracking of complex movements, but it is also the most favourable for enabling gesture recognition.

2.3. Classification

2.3.1. Image Classification

Classification is the process of categorizing a given set of input data into classes, also called labels or categories. This is done through algorithms known as classifiers, which classify the data into the most appropriate category. The classifiers must be trained with pre-labelled training data to determine which input variables are related to which output classes. Image classification is the process that classifies an image with a label according to its visual content. While for humans it is easy to classify Figure 2.9 as a cat, with computer vision this is more challenging, as this classification has to be made by processing arrays of numeric values.



```
[[105 112 108 111 104  99 106  99  96 103 112 119 104  97  93  87]
 [ 91  98 102 106 104  79  98 103  99 105 123 136 110 105  94  85]
 [ 76  85  90 105 128 105  87  96  95  99 115 112 106 103  99  85]
 [ 99  81  81  93 120 131 127 100  95  98 102  99  96  93 101  94]
 [106  91  61  64  69  91  88  85 101 107 109  98  75  84  96  95]
 [114 108  85  55  55  69  64  54  64  87 112 129  98  74  84  91]
 [133 137 147 103  65  81  80  65  52  54  74  84 102  93  85  82]
 [128 137 144 140 109  95  86  70  62  65  63  63  60  73  86 101]
 [125 133 148 137 119 121 117  94  65  79  80  65  54  64  72  98]
 [127 125 131 147 133 127 126 131 111  96  89  75  61  64  72  84]
 [115 114 109 123 150 148 131 118 113 109 100  92  74  65  72  78]
 [ 89  93  90  97 108 147 131 118 113 114 113 109 106  95  77  80]
 [ 63  77  86  81  77  79 102 123 117 115 117 125 125 130 115  87]
 [ 62  65  82  89  78  71  80 101 124 126 119 101 107 114 131 119]
 [ 63  65  75  88  89  71  62  81 120 138 135 105  81  98 110 118]
 [ 87  65  71  87 106  95  69  45  76 130 126 107  92  94 105 112]
 [118  97  82  86 117 123 116  66  41  51  95  93  89  95 102 107]
 [164 146 112  80  82 120 124 104  76  48  45  66  88 101 102 109]
 [157 170 157 120  93  86 114 132 112  97  69  55  70  82  99  94]
 [130 128 134 161 139 100 109 118 121 134 114  87  65  53  69  86]
 [128 112  96 117 150 144 120 115 104 107 102  93  87  81  72  79]
 [123 107  96  86  83 112 153 149 122 109 104  75  80 107 112  99]
 [122 121 102  80  82  86  94 117 145 148 153 102  58  78  92 107]
 [122 164 148 103  71  56  78  83  93 103 119 139 102  61  69  84]]
```

Figure 2.9. Classification of an image of a cat [22].

As objects are often in different orientations then those in the training data, no image will have a 100% resemblance. Image classification predicts the label that most closely correlates to the input data, Figure 2.9. This way, with enough training data, the algorithm can reliably label objects in new positions and orientations. A confusion matrix is often used to visualize the accuracy of predicted classes by comparing them to the correct class. It is an important verification tool to see if any specific types of classes are systematically misclassified.

A popular approach to achieving image classification is Deep learning, a machine learning method based on Deep Neural Networks (DNNs). It allows for a computer model to learn how to classify images into a specific set of classes. It does so by training on data from a big database of images already labelled into those categories.

2.3.2. Deep Neural Network Architecture

A Deep Neural Network (DNN) is an artificial neural network comprised of an input and output layer with several hidden layers in between, as seen in Figure 2.10. The presence of many hidden layers improves the ability of the network to act upon its input. This allows DNNs to outperform single-layer neural networks, especially when assigned to complex problems [23], like gesture detection and localization [24].

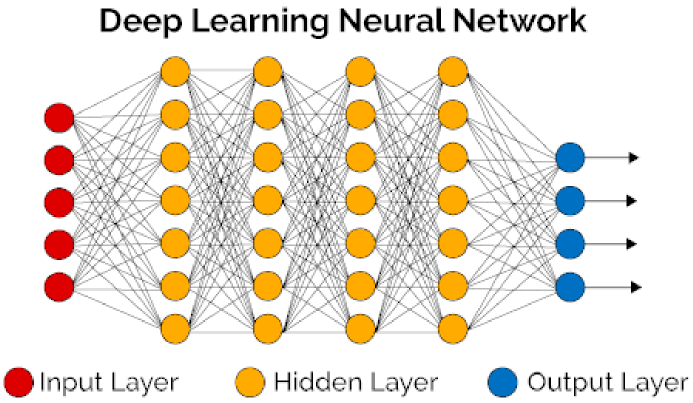


Figure 2.10. Constitution of a Deep Neural Network [25].

Each of these hidden layers contributes as a computational stage which transforms, often non-linearly, the information received from the previous stage and outputs it into the next. Depending on the desired DNN's application, different sequences of layer types are used, Figure 2.11. In fully connected layers, each node of the current layer is connected, through weights, to all the nodes of the next layer, Figure 2.11(a). In Convolutional layers, features are extracted from the current layer through a filter, Figure 2.11(b). Both of these types of layers are used to create feed-forward networks, in which no feedback exists to previous layers. In Recurrent layers, however, output data are reused as an input for the next computation in the same node, Figure 2.11(c). These layers are the main element used to form recurrent neural networks.

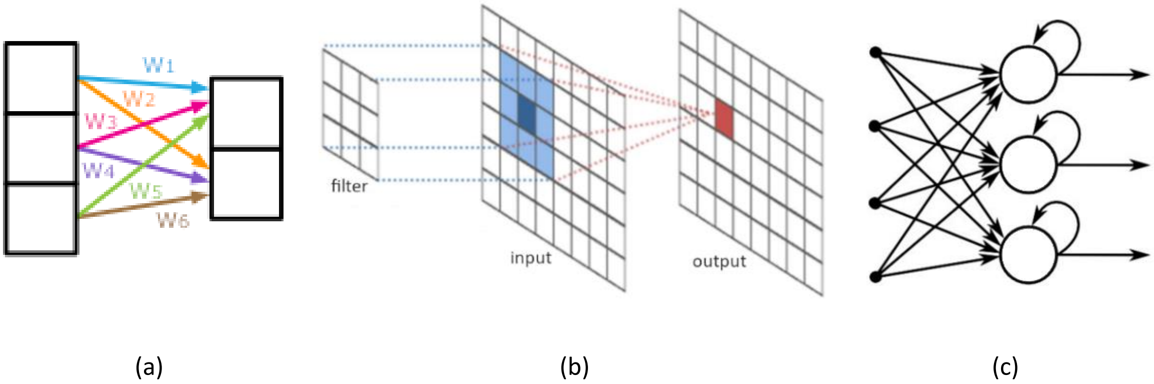


Figure 2.11. Common types of neural network layers: (a) Fully connected layer. Adapted from [26]; (b) Convolutional layer; (c) Recurrent layer [27].

2.3.2.1. Convolutional Neural Networks

A Convolutional Neural Network (CNN) is a feed-forward DNN comprised of a series of convolutional layers, followed by fully connected layers. The convolutional layers extract features of the input image, and the fully connected layers then make a prediction based on those features. CNNs are usually applied to image classification applications, e.g. multi-class geospatial object detection [28].

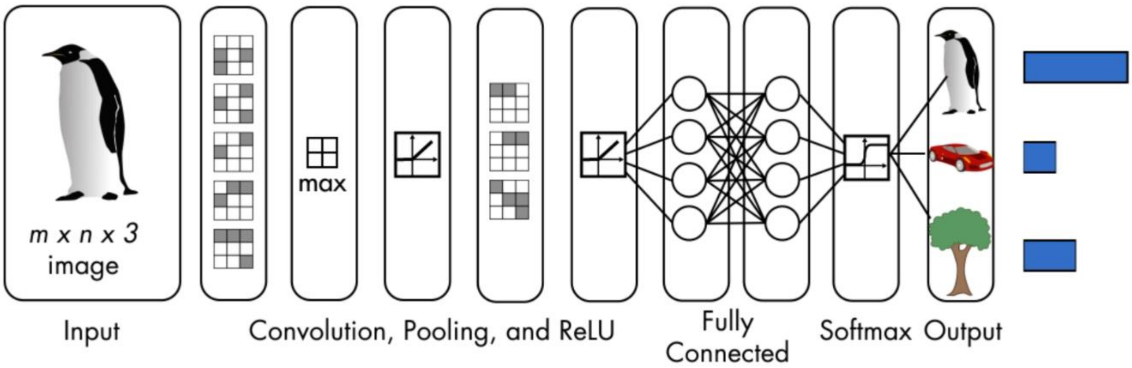


Figure 2.12. Typical structure of a Convolutional Neural Network [29].

CNNs usually have an architectural structure like the one in Figure 2.12. Alongside fully connected and convolutional layers, additional types of layers are used. While Pooling layers are used for down-sampling, Rectified Linear Unit (ReLU) layers are used to turn negative values into zero and the Softmax layer turns numeric input values into probabilities.

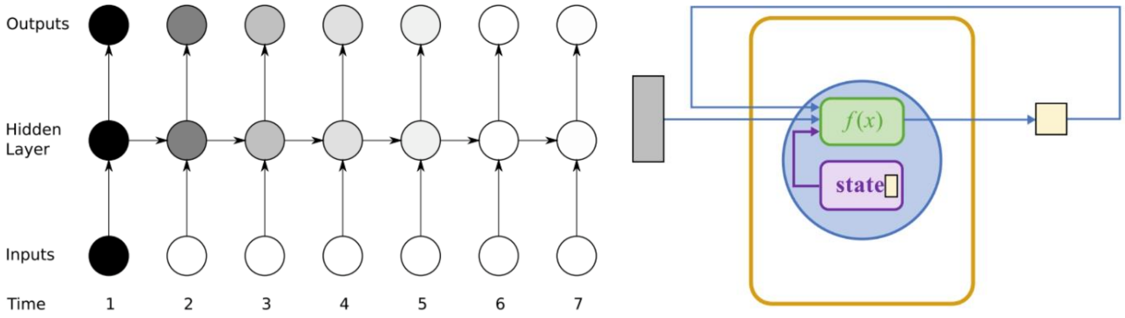
Training a CNN from scratch has the potential for the best classification results but requires a huge amount of training data and high computational time. Otherwise, pre-trained networks, like ResNet-50 [30], while easy to use, are trained towards a large set of generic classes. Transfer Learning is the middle ground between these two options. By modifying some layers of a pre-trained network, Transfer Learning preserves the network's aptitude to classify images while allowing the network to classify a more specific set of classes.

Depending on its architecture, CNNs can be applied to the classification of images, with a 2D CNN, or videos, with a 3D CNN. As 3D CNNs are computationally expensive and quite difficult to train, an alternative way of interpreting temporal context is needed.

2.3.2.2. Recurrent Neural Networks

A Recurrent Neural Network (RNN) is a type of DNN in which all its hidden layers are recurrent. As introduced before, this type of layer uses the output of previous computations to do the next. Owing to this, RNNs are used in applications that require a sequence predictor, such as speech recognition [31].

A common issue of RNNs is that the influence of older inputs decreases over time, called "the vanishing gradient problem". New inputs replace the activations of the hidden layer, making the network 'forget' the old inputs, making it difficult to learn long-term dependencies. Figure 2.13(a) displays this by representing the loss of information from the initial input with the increasing fade to white at each timestep.



(a) (b)

Figure 2.13. (a) The vanishing gradient problem for RNNs [32]; (b) LSTM node architecture [29].

A Long Short-Term Memory (LSTM) [33] is a type of RNN architecture created as a response to this problem. The nodes of an LSTM are very similar to the ones found in recurrent layers, except they also contain a cell state, Figure 2.13(b). This cell state is regulated by structures called gates. Through training, these gates learn when to alter the information of the cell state. This allows the LSTM to preserve information of long sequences of data. LSTMs are, thus, ideal for learning long-term dependencies, necessary for applications like the composition of music [34], speech recognition [35], gesture classification [36], anomaly prediction [37], among others.

2.3.2.3. Long-term Recurrent Convolutional Networks

The structure resulting from the junction of a CNN with an LSTM is called a Long-term Recurrent Convolutional Network (LRCN). It is used for example to classify videos by first extracting the visual features of its frames through a CNN and then classifying them, considering their temporal sequence, with an LSTM.

The CNN used in an LRCN is usually a pre-trained network or one modified with transfer learning. This way only the LSTM needs to be created from scratch. Generally, a 2D CNN is used, but in some cases, a 3D CNN is used instead [38]. LRCNs have been proven successful in recognition of gestures from a video stream in real-time [39] and gesture interaction with a Smart TV [38], which demonstrates their capability to handle the classification of gestures.

2.4. Data Dimensionality Reduction

2.4.1. Core concepts

Data Dimensionality Reduction (DDR) methods provide a way to represent a high-dimensional dataset by transforming it into a lower-dimensional state, Figure 2.14. This downward dimensional shift aims to preserve, as best as possible, the general structure of the high-dimensional original data.

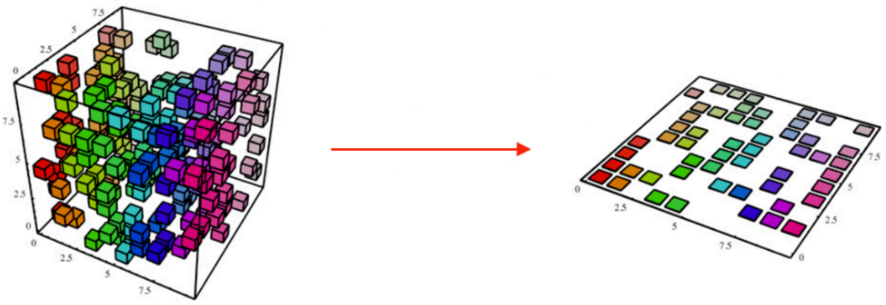


Figure 2.14. Dimensionality reduction from high-dimensional data to low-dimensional data. Adapted from [40].

The employment of DDR is necessary as the depiction of high-dimensional data, especially when it exceeds the 2D space. A DDR method usually reduces the data to a lower space, many times a 2D space, which depicts variable correlations in a more comprehensible way.

As ECs output events in a 1x4 array (1), it means that, at most, it is possible to use 4D data. However, to classify images using a pre-trained network, data typically must be organized into frames, which display data within a 2D space. Not using a DDR means only being able to represent, at most, each event's x coordinate, y coordinate, and its polarity. The polarity can be represented in a 2D space along with the coordinates by using two different colours, e.g. green and red, due to it being a binary variable.

Using a DDR would allow for a way to include the information related to the timestamp of each event into image classification. This additional information, while previously discarded, could provide more insight into the correlation of data, possibly improving classification accuracy. Another possible benefit of converting to a lower dimension is the significant decrease in the amount of data. This is particularly crucial for high-dimensional datasets that are quickly generated, like those created by dynamic gestures [8]. This advantage, though, may later be negated, as using a pre-trained CNN requires the data to be in the specific format requested by the input layer.

2.4.2. Types of DDR

There are two kinds of DDR methods: linear and non-linear methods. Linear DDR methods are useful to obtain the global structure of the data, more easily finding the zones of interest. Non-linear DDR methods tend to preserve, with reasonably good accuracy, the local neighbourhood structure of each event.

For the scope of this work, 2 distinct DDR methods have been selected: linear PCA and the non-linear Kernel PCA.

2.4.2.1. Principal Component Analysis

Principal component analysis (PCA) [41] is a mathematical tool that performs an orthogonal linear transformation of a set of n p-dimensional observations, $\mathrm{X} \in \mathbb{R}^{\mathrm{n}\times\mathrm{p}}$, into a space defined by Principal Components[1] (PCs). As these PCs must have a size less than or equal to the number of original dimensions, p, PCA is typically used to represent the original data within a lower-dimensional space.

The first component, PC 1, correlates to the highest variance in the data. Each of the following PCs is orthogonal to the preceding one and has the highest variance possible under its orthogonality constraint. Commonly, only the first and second components are plotted against each other to obtain a two-dimensional space that shows most of the variance of the original data [42], allowing for easier visualization of the data. The remaining PCs are correlated to the lowest variance in the data, so they are not used to build the component space. The process of applying PCA is exemplified in Figure 2.15.

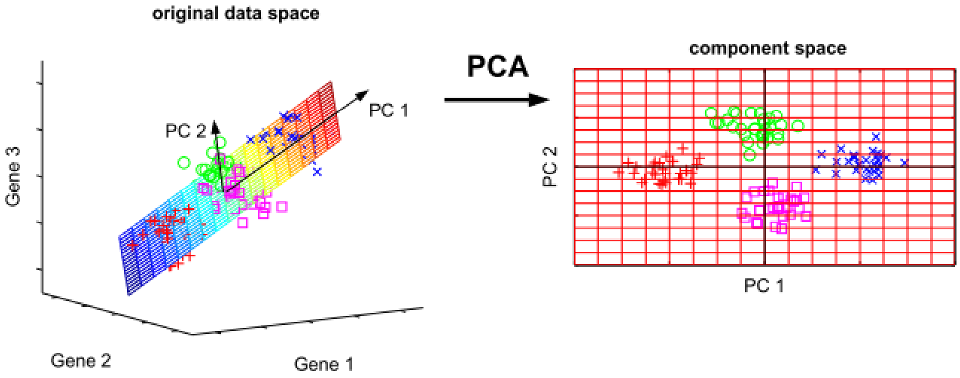


Figure 2.15: PCA transformation: Projection of 3D data onto a 2D component space [42].

PCA can also enable the detection of features through its PCs, as has been shown in [43] where the eigenvectors resultant from PCA are used to extract features for face recognition. It greatly reduced the dimensionality of the samples while also improving classification accuracy.

[1] Principal Components are the eigenvectors of the covariance matrix computed from the dataset. They are sorted according to their corresponding eigenvalue in decreasing order.

Using PCA as the DDR method has, though, a major drawback which is related to the unwanted environmental factors that may have an impact on the variance of the data. PCA cannot distinguish, only from variance, relevant from nonrelevant contributions [42]. To solve this, there is a need to use normalization techniques to reduce non-relevant variance.

2.4.2.2. Non-linear Principal Component Analysis

Time-dependent actions, like DGs, are generally located within a non-linear subspace. This means they are most likely better explained by a low number of non-linear components rather than by an equal number of linear components. This prompts the use of non-linear principal analysis (NLPCA), which is a non-linear generalization of standard linear PCA. The main difference between PCA and NLPCA is that the latter involves non-linear mappings between the original and reduced dimension spaces. This enables NLPCA to identify both linear and non-linear correlations. There are 2 well-known approaches to NLPCA, the use of Principal Curves and Kernel PCA.

2.4.2.2.1. Principal Curves

In the case of PCA, the principal components are estimated by finding the line that minimizes the orthogonal deviations. Data tends to follow in a non-linear distribution, thus the sum of the distances from the data points to that line tends to be very high. To try to solve this issue, the PCs are generalized to curves, called Principal Curves [44], instead of straight lines. The curve minimizes the sum of orthogonal deviations just like a straight line, but the distances of the data points to the now curved line are significantly lower than its linear counterparts, as shown in Figure 2.16. The NLPCA is specifically called Circular PCA when the Principal Curve is a closed curve [45].

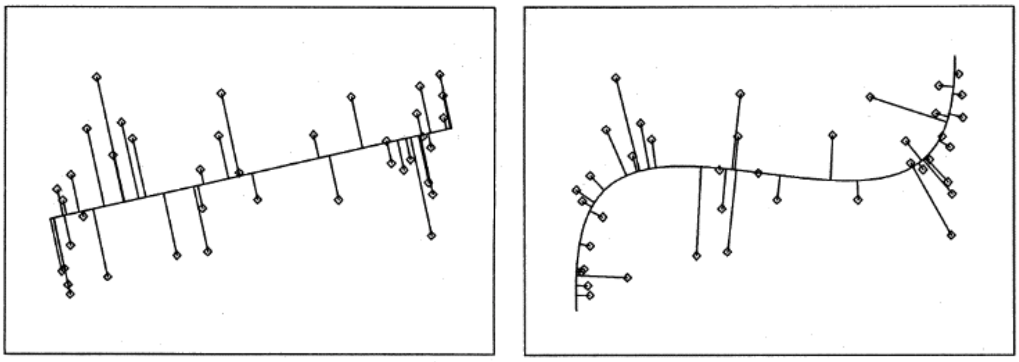


Figure 2.16: A Linear principal component (left) and Principal Curve (right). Both minimize the sum of squared deviations for all data points [44].

Though Principal Curves seem conceptually simple, they are demanding for large data sets, as a lot of computational steps are involved [46]. This method is mainly used to detect non-linear curves and two-dimensional non-linear planes, due to being limited to the use of two principal components because of its complexity [42].

2.4.2.2.2. Kernel Principal Component Analysis (Kernel PCA)

Kernel PCA [47] maps the original set of datapoints $z_i \in \mathbb{R}^N, i = \{1, 2, \dots, K\}$ from input space onto a high-dimensional feature space called *F*, through $z \rightarrow \Phi(z)$, and then performs PCA on F [46]. This can be visualized in Figure 2.17. As linear PCA is applied to the feature space, since F is non-linearly related to input space through Φ, the contour lines of the projections of the data points onto the principal eigenvector in *F*, represented as dotted lines in Figure 2.17, become non-linear in the 2D input space.

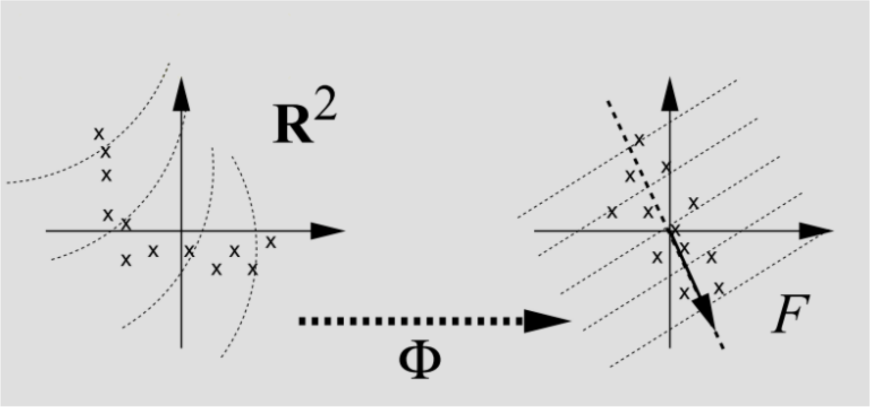


Figure 2.17: Kernel PCA technique, applied to a 2D input space, using a Polynomial kernel function [47].

The transformation is made by a non-linear function Φ, called the kernel function. The most common kernel functions used are either "Gaussian" or "Polynomial". This method has the benefit of not actually "mapping" the information into the feature space, but instead carrying out all computations through the kernel function in input space. This allows it to use high-dimensional feature spaces with reasonable computational cost [47].

2.4.3. Comparison and applicability of DDR

PCA has already been shown to be successful applied in gesture recognition [48]. As it is a linear method and gesture data have a non-linear behaviour, an attempt for better results will also be made with a non-linear method.

Both the NLPCA methods presented previously provide different solutions for dealing with non-linear problems. Kernel PCA is the most versatile, efficient, and reliable of the two. The usage of Principal Curves is too computationally demanding, especially considering the significant amount of data that needs to be processed. Kernel PCA will, thus, be selected to compete against the results obtained with PCA.

Other DDR methods that could have been applied were outside the scope of this study. Some noteworthy DDR methods that were considered are TriMap [49] and t-SNE [50]. However, after deliberation, the first was deemed too complex and the latter unreliable due to random computations.

3. METHODOLOGY

3.1. Dataset for Primitive Manufacturing Tasks

In this section, the setup for capturing data for the newly created dataset for primitive manufacturing tasks (ECmanufacturing20) is detailed. It ensures the quality of the dataset and covers the most relevant manufacturing tasks, namely the ones related to assembly tasks.

3.1.1. Setup for Capturing EC Data

The EC used to capture the dataset footage is a DAVIS 240C. It is a 240x180 pixel DVS camera with simultaneous active pixel frame output. Only its event data will be captured. The DVS's characteristics are presented in Table 3.1. The camera lens used has a manually adjustable Aperture (Iris), Zoom, and Focus.

Table 3.1: DAVIS 240C: DVS's Specifications

Resolution	240 x 180 pixels
Dynamic Range	120 dB
Minimum latency	~ 12 us @ 1klux with optimized biases
Bandwidth	12 MEvents/ second
Power Consumption	5-14 mW (activity dependent)

To record the data for the dataset, the setup conditions in Figure 3.1 should be respected.

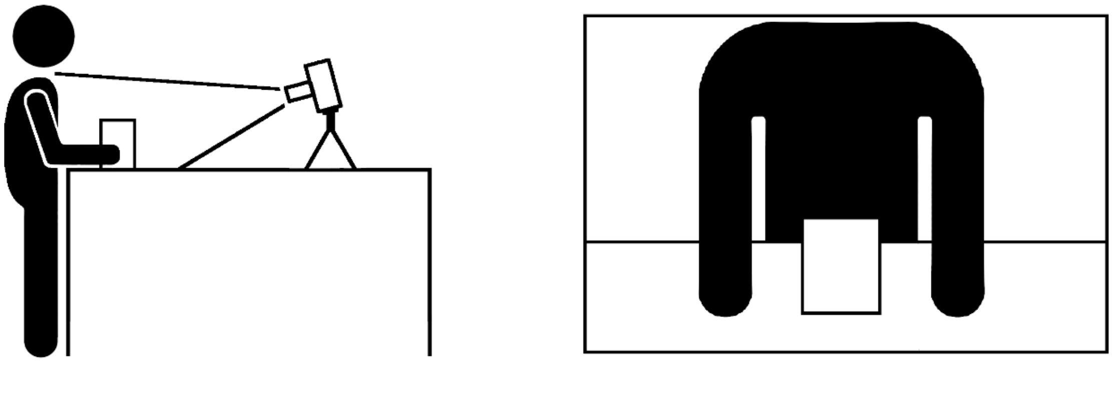

(a) (b)

Figure 3.1. Recording gesture data with an EC: (a) Physical Setup; (b) Sensor View.

The EC should be positioned in a way to capture as much useful information as possible. As is common for datasets for gesture recognition, this means the sensor view will be the front view, capturing the upper body motion, Figure 3.1(a). In this dataset, the main objective is to be able to detect which primitive tasks the user performs while interacting with an object. Thus, the object and hands should be within the camera range of view. To reduce the amount of noise caused by involuntary movements by the user, like blinking, the user's head should not be within the range of view.

The more space the hands and object occupy within the sensor's view, the less pointless information the camera captures. However, the camera's view should accommodate for the user's range of motion to be able to capture all the performed actions. The chosen sensor view is depicted in Figure 3.1(b). To achieve this sensor view, a combination of the camera's lens "zoom" setting and the distance between the human and the camera must be defined. To do this, first, the desired camera distance from the user is chosen and then the "zoom" setting is changed accordingly to fit the user within camera range of view as illustrated in Figure 3.1(b). For this dataset, a distance of 60 cm between the camera and user has been selected.

A working stance can either be standing or sitting. Assuming an ergonomic position, for both cases, the camera views the user's upper body. Thus, there are no significant differences between the data of a standing or sitting user. The surrounding environment and background are not controlled. The only thing to consider is to not have any moving parts behind the user while recording the dataset.

3.1.2. Primitive tasks

The dataset is comprised of primitive tasks the user makes while performing manufacturing assembly tasks. When these primitive tasks are combined, they cover the most common manufacturing tasks. To exemplify these tasks, some interactable objects had to be selected, Figure 3.2. The chosen objects are three simple 3D shapes, a cube, a cuboid (with a single screw), and a cylinder, which will be used to demonstrate all tasks. A screwdriver will also be used as a tool for the "Screw" task.

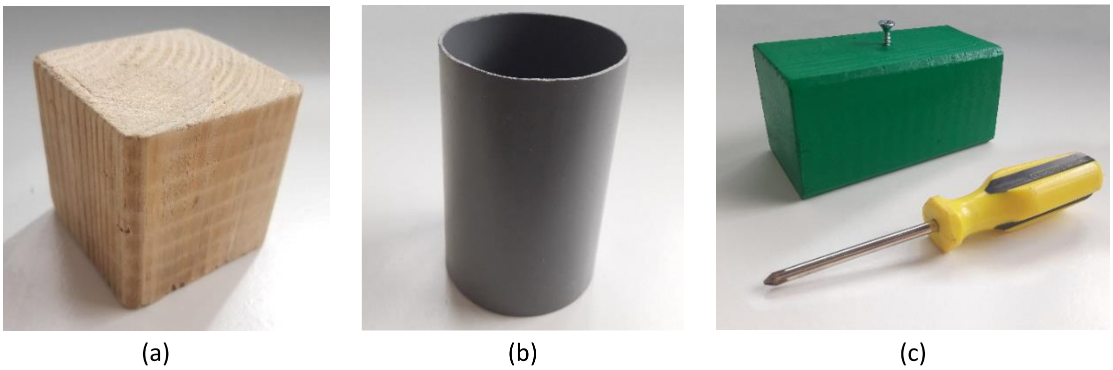

(a) (b) (c)

Figure 3.2. Objects used in the creation of the dataset: (a) A cube; (b) A cylinder; (c) A cuboid with an inserted screw and the designated screwdriver.

The primitive tasks chosen for this dataset are as shown in Figure 3.3.

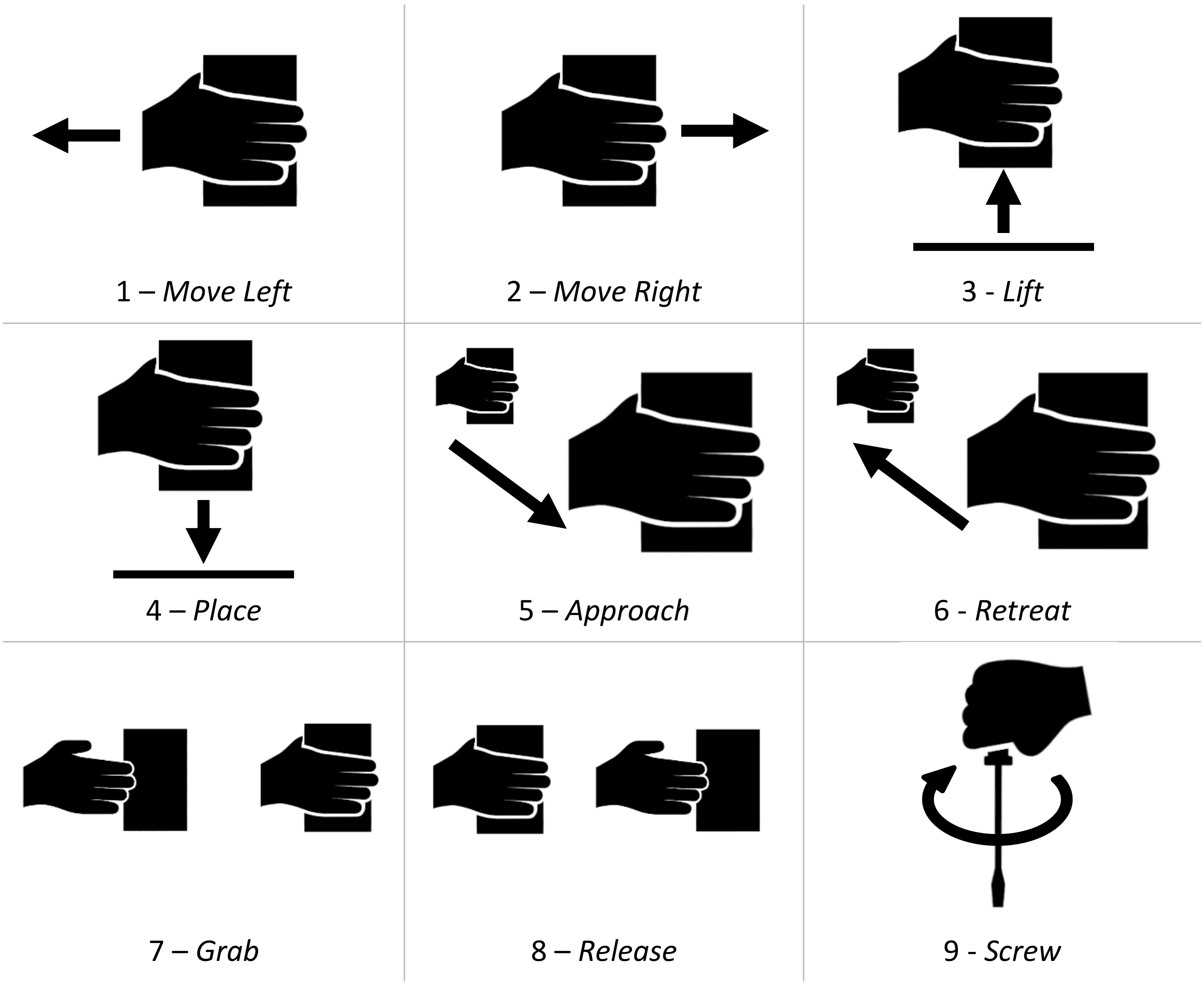


Figure 3.3: Representation of all tasks and their respective number used in the ECmanufacturing20 dataset.

These include simple translational movements in the 3D space while holding the object, Tasks 1 through 6, a grab and release gesture, Tasks 7 and 8, and a screwing gesture, Task 9. In addition to these dynamic primitive tasks, the dataset will also be trained to recognize the user as being idle, Task 10.

The main dataset with 23 sequences for each task was recorded containing, in total, 230 sequences. About half of the sequences for each task were recorded with the right hand and the rest with the left hand. When recording the sequences, diversity is introduced by recording the tasks in different locations on the table, closer or further away from the EC. More variety is endorsed by alternate use of the objects. An additional smaller testing dataset with 6 sequences for each task was recorded for DNN testing, adding up to 60 sequences. For each task, 3 sequences are recorded for each hand. Each one of these 3 sequences uses a different object. This way, at a later stage, the LRCN's performance can be thoroughly tested.

3.2. Pre-processing of data

As discussed previously, to process EC data in a visually perceptive way, integrated images are created. Usually, integrated images construct a frame with all the events generated in a predefined interval of time, having the advantage of easily identifying the movement in time. However, for very slow movements a lot of redundant frames will be captured and in fast movements the framerate is limited.

An alternative approach is to capture a frame at each predetermined number of events. This is much more versatile as it adapts the framerate to the speed of the movement, accounting for both fast and slow movements. For a high number of events used per frame, the image generated at each frame is more easily recognizable, due to the increase of data, but has higher motion-related noise and lower framerate. Meanwhile, for a low number of events per frame, the lower overall noise and higher framerate come at a cost of less recognizable images.

To decide on which the number of events per frame should be, both a slow and a fast version of a task sequence (grab-lift-left-right-place-release) is recorded with the previously defined setup. Then, according to the obtained results, a score from 1 to 5 is given to each category, 1 being the worst score and 5 the best. The rounded down values of the obtained Frames/second for each attempt are shown in Table 3.2.

Table 3.2: Average frames/second values obtained for different events/frame values.

Number of Events per Frame							
1000		2000		3000		4000	
Slow	Fast	Slow	Fast	Slow	Fast	Slow	Fast
44 fps	85 fps	21 fps	42 fps	14 fps	28 fps	10 fps	21 fps

A weighted decision matrix is then constructed with the designated scores, Table 3.3. The number of events per frame that displays the total highest score is 3000 Events/frame, and thus this will be the chosen value. Some of the frames obtained for 3000 Events/frame are shown in Figure 3.4.

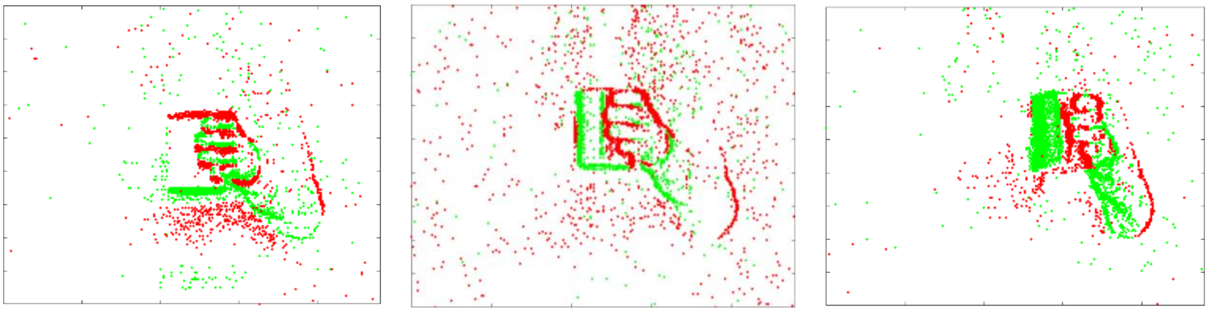


Figure 3.4. The transition between “Up” and “Left” task in the sequence with 3000 Events/frame.

Table 3.3: Weighted decision matrix to compare performance between Events/Frame values.

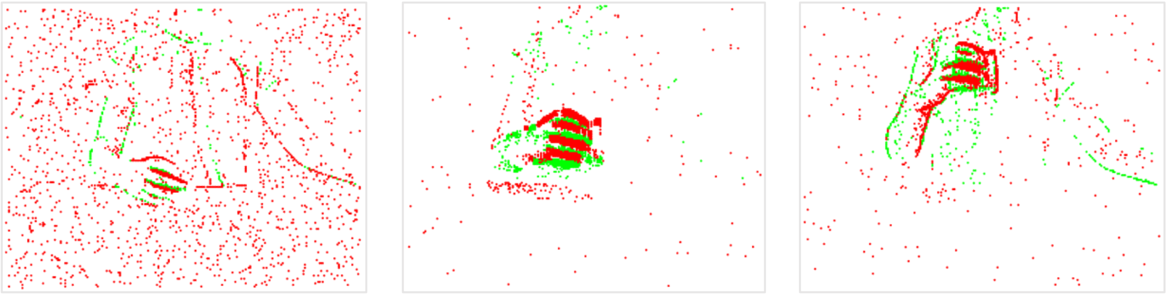
		Number of Events per Frame							
Criteria	Weighting / Importance	1000		2000		3000		4000	
		Slow	Fast	Slow	Fast	Slow	Fast	Slow	Fast
Perceptibility of hands	5	2	3	3	4	4	4	4	4
Absence of random noise	4	2	2	3	3	4	4	4	4
Absence of movement related noise	3	4	4	3	4	3	3	2	2
Number of Frames per Second	2	5	5	4	4	4	4	3	3
	Total	85		96		106		96	

3.3. Noise Reduction

Noise is output data which is irrelevant. It occurs along with the relevant data, making it difficult to differentiate. Noise reduction removes as much irrelevant data as possible. The frames obtained by creating integrated images with 3000 Events, Figure 3.4, while being recognizable, still have a lot of noise. To avoid poor classification performance resulting from too much irrelevant information, a noise reduction algorithm should be used. After identifying the main noise sources, a noise reduction algorithm is discussed.

3.3.1. Identification of Noise Sources

To discover which are the existing main noise sources, all output frames from a certain task are analyzed. From the frames obtained in Task 3, the most relevant sources of noise can be identified, Figure 3.5, as being idle, shadows, and body movement.



(a) (b) (c)

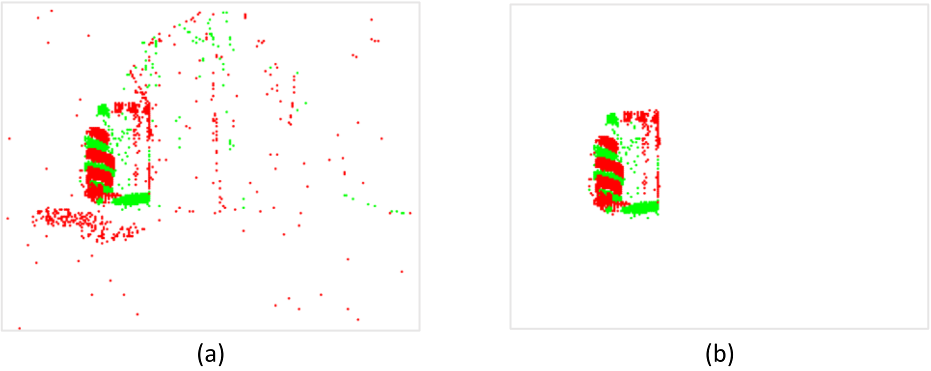
Figure 3.5. Noise source types identified within a Lift task: (a) Idle; (b) Shadows; (c) Body Movement.

Idle noise is created due to almost no movement occurring in the camera's range of view. This causes smaller changes in brightness to be captured, which means a lot of irrelevant events are generated, as seen in Figure 3.5(a). This specific behavior is very common when the user is in an idle position.

Shadows are an inevitable issue when using a sensor that captures movement through changes in brightness, Figure 3.5(b). Differences in lighting conditions tend to produce different kinds of shadows. This inconsistency may lead to inferior classification performance, and as such, the shadows should preferably be removed. Even when attempting to only move an arm to lift an object, the whole body tends to move accordingly, Figure 3.5(c). This is the most difficult type of noise to manage, as it is the one with the most variable outcome and multiple clusters of events may be formed, whereas in the shadow case usually only one is generated and in the idle case none.

3.3.2. Preparation and Interpretation of Data

The main objective of the noise reduction algorithm is for all images to accurately represent the hand performing a task and the object used, with as little noise as possible. One approach is to ignore the events outside the region of the image containing the hands and the object. This type of image manipulation process is called *Cropping*. It eliminates the vast majority of noise originated by shadows and body movement, ideally performing as shown in Figure 3.6.


(a)

(b)

Figure 3.6. Frame obtained from a Lift task: (a) Original frame; (b) Desired cropping result.

This action must be accomplished for each frame, without any knowledge of what kind of task is being performed and of what the proximity of the hand to the EC is. It should allow for the preservation of all the information of the events contained within the preserved region: its x coordinate, y coordinate, polarity, and timestamp.

Assuming a single frame with 3000 events, such as Figure 3.6(a), the location of the hand within the camera's view is the zone with the highest density of events. It is difficult to locate this zone directly from the raw EC data, in which each event is represented by (1). To better display the EC data, each frame is transposed to a grid with dimensions 240 x 180 pixels (width x length), in which each cell corresponds to the coordinates of a pixel of the EC. The function of each cell is to store the number of times an event occurs at that specific coordinate, as specified in Algorithm 1. Using Figure 3.6(a) as the original frame, the produced grid is as shown in Figure 3.7, with darker cells representing many events.

Algorithm 1 Creation of the Event occurrences Grid

```
Prepare the grid as a 240x180 empty matrix

for each Event
        Find its x and y coordinate
        Sum 1 to corresponding coordinates in the grid
End
```

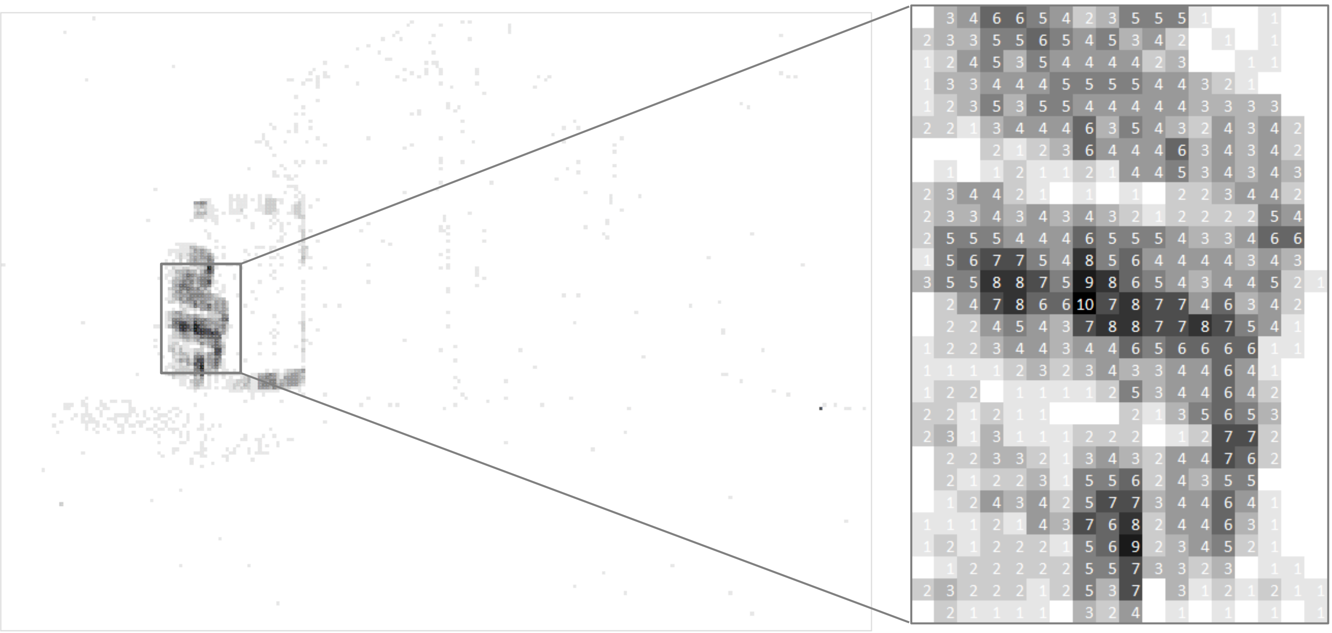


Figure 3.7. Grid with the number of event occurrences at each cell coordinate with magnified section. Obtained from applying Algorithm 1 to the data of Figure 3.6(a).

By observing the grids obtained for different tasks, some important recurring behaviors are discovered:

- The cells with the highest count of events are located within the hand and the outer edge of the object. In tasks that require significant arm movement, the arm's outline also exhibits a high event count. Identifying these edges is the most effective way of finding the hand within the image and, thus, defining the area of the image that should be preserved;
- The closer the hand holding the object is to the camera, the more difficult it is to detect its outline, as it is defined by more sporadic events;
- Most of the cells corresponding to previously identified noise sources, mainly shadows and body movement, have a low count of events.

Occasional singular cells with an unusually high event count without many nearby event occurrences can be attributed to a "Stuck Pixel"[2]. To solve this issue, each cell of the grid that has an event count greater than 10 should be changed to 2. This must be done as the pixel may or may not be used depending on the task and, as such, should not be completely erased. In rare cases, this does target some cells used to represent either the hand or the object. However, as they have a lot of surrounding high event count cells, it won't impact the proficiency of the algorithm.

3.3.3. Creation of the Region of Interest

The main part of the noise reduction algorithm is to identify the hand and the object, which can be distinguished by their outlines. There is a need to scan the grid to identify these outlines to define the Region of Interest[3] (ROI). Most objects chosen for the dataset will have some of their outlines aligned vertically. Thus, a grid column corresponding to the location of said vertical outlines will contain a lot of cells with high event occurrence values. This means that the sum of all events in each column is a good indicator to find these object outlines. The same is true for finding outlines aligned horizontally by using the grid's rows. To see this in practice, three separate frames from the Approach task, Tasks 5, were selected, with the hand and object at different distances from the camera, Figure 3.8.

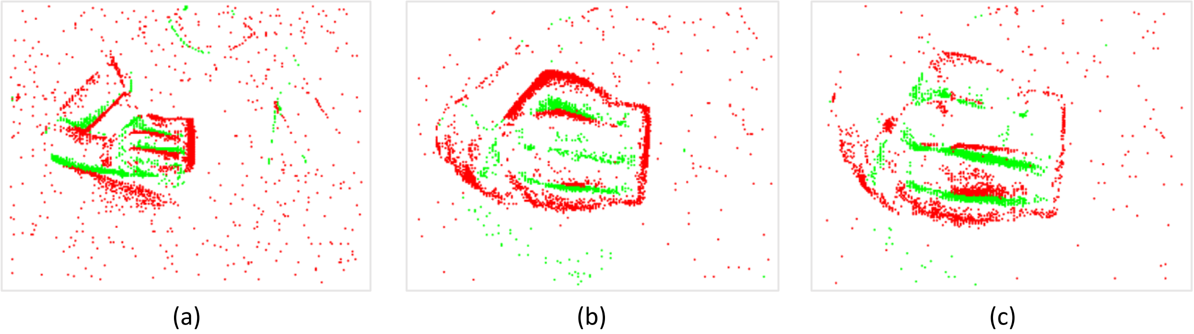


Figure 3.8: Selected original frames from the Approach task.

Figure 3.8(a) will be used to test the hypothesis about finding hand and object outlines through columns and rows with high sums. The result of summing the values of the cells of each row of the grid obtained for this image are plotted in a line graph shown side-by-side with the grid, while the same is done for each column, showing the resulting sums underneath the grid, Figure 3.9. This confirms that the zone with the hand and object corresponds to the zone with the highest sums.

[2] Also known as a "Hot Pixel", it is a pixel that is always "on" (can be caused by any of its sub-pixels).

[3] The portion from the original image where most relevant information is located.

There is a need for a certain limit that, when intersected with the plot of the grid sums, creates the region of interest. This limit should not be constant, as it needs to be able to adapt to the different kinds of tasks. Thus, two hypothetical limits are tested against the data, the median and the average of the grid sums. The ROI will be defined as the zone starting when at least 3 consecutive sums are higher than the limit and ending when 3 consecutive sums are lower than it, as described in Algorithm 2. Multiple zones can be obtained this way, but for now, only the ROI which contains most events is selected. Figure 3.10, Figure 3.11, and Figure 3.12 show the results obtained for the frames shown in Figure 3.8(a), Figure 3.8(b), and Figure 3.8(c), respectively. The blue line represents the median and the orange line portrays the average.

Algorithm 2 Creation of Region of Interest with single zone. Exemplified for the vertical limits of the ROI. The horizontal limits are obtained similarly.

```
Sum the values of each column of the grid
Find the average of the sums $\mu$

for each column of the grid
    Build an array with the sum value of the current and previous two columns
    if all values in the array are greater than $\mu$ then
        Save starting column of the cluster
    elseif all values in the array are less than $\mu$ then
        Save ending column of the cluster
    end
end

Use the first column and the last column of the biggest cluster to define the ROI
```

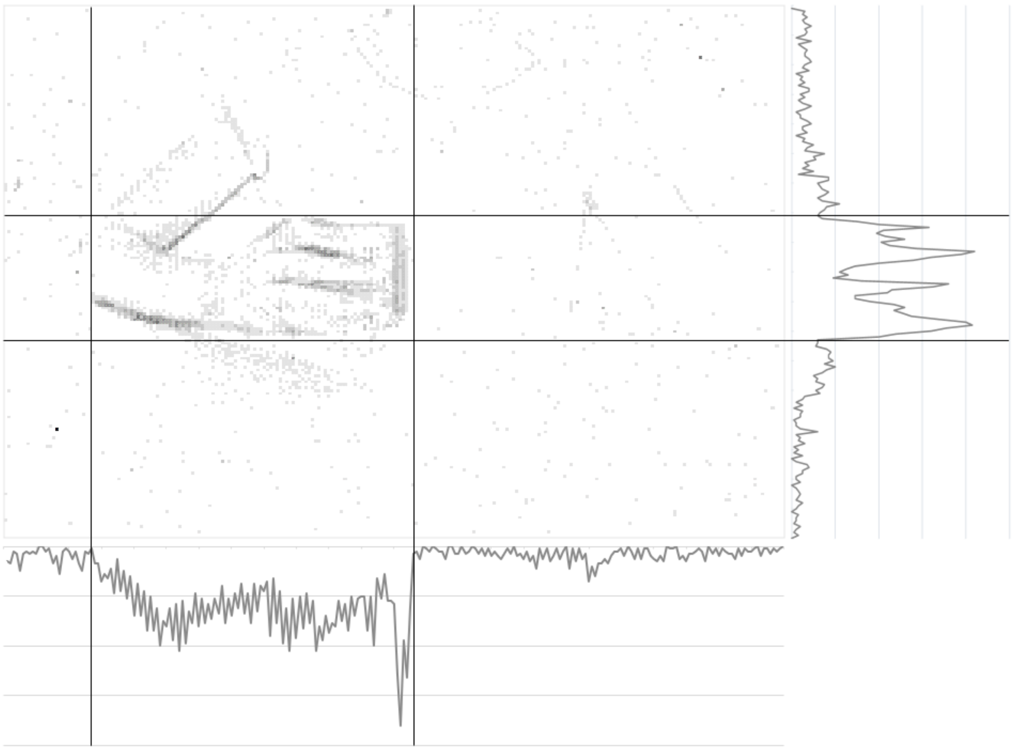


Figure 3.9: Desired ROI, contained within black outlines, compared to the grid sums. Obtained from data of Figure 3.8(a).

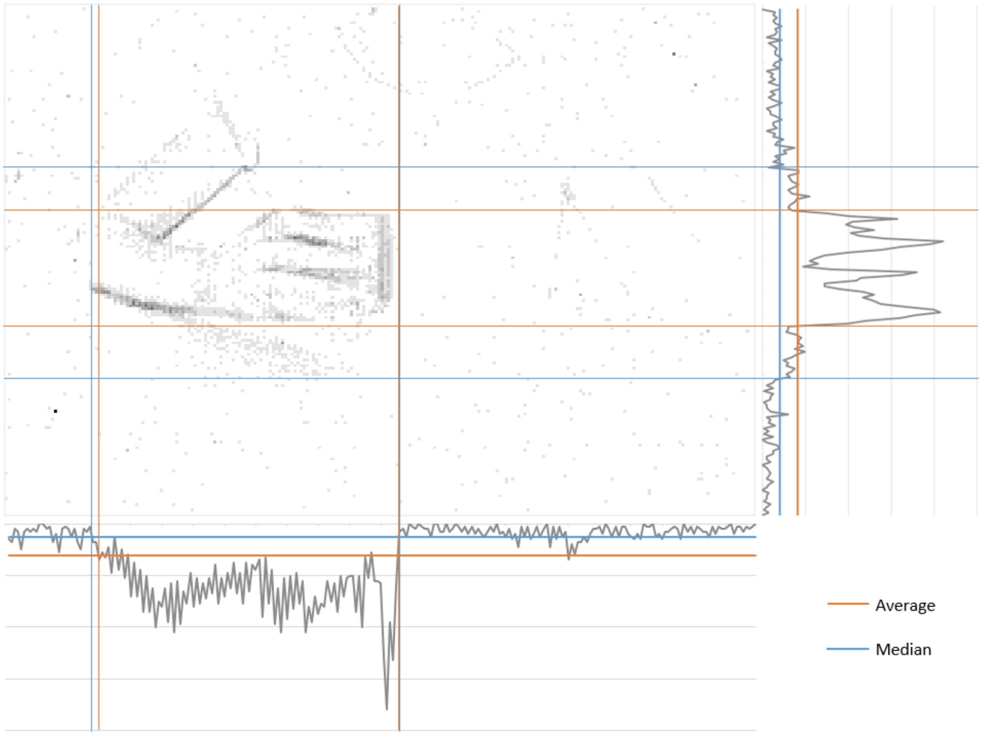


Figure 3.10: ROI created by the intersection of the grid sums and their average/median. Obtained from data of Figure 3.8(a).

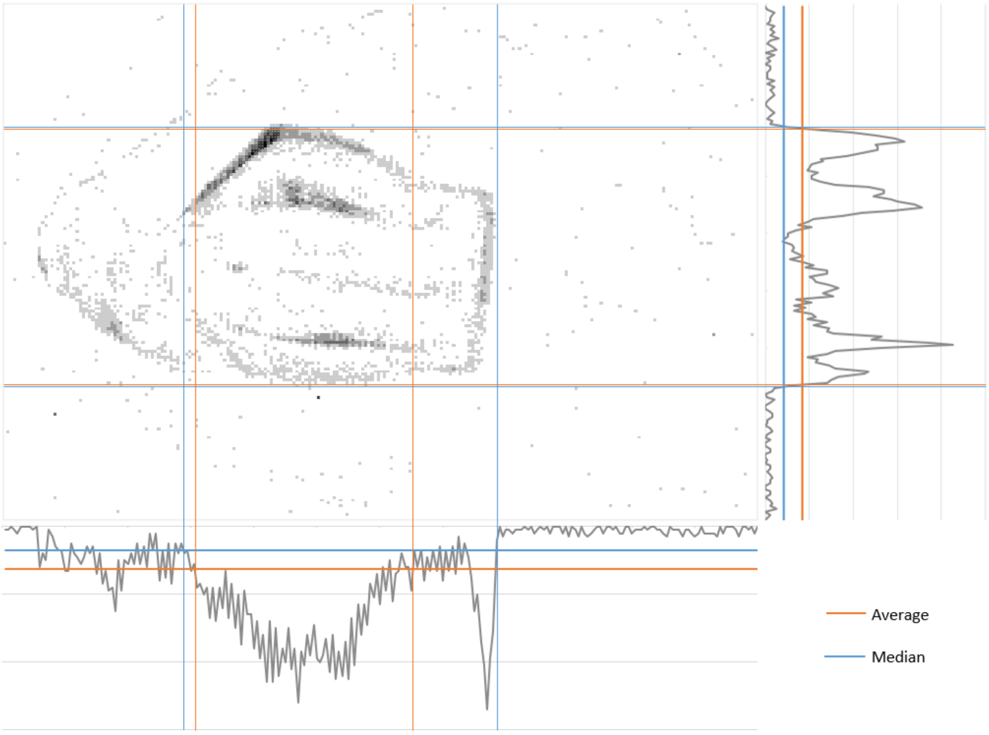


Figure 3.11: ROI created by the intersection of the grid sums and their average/median. Obtained from data of Figure 3.8(b).

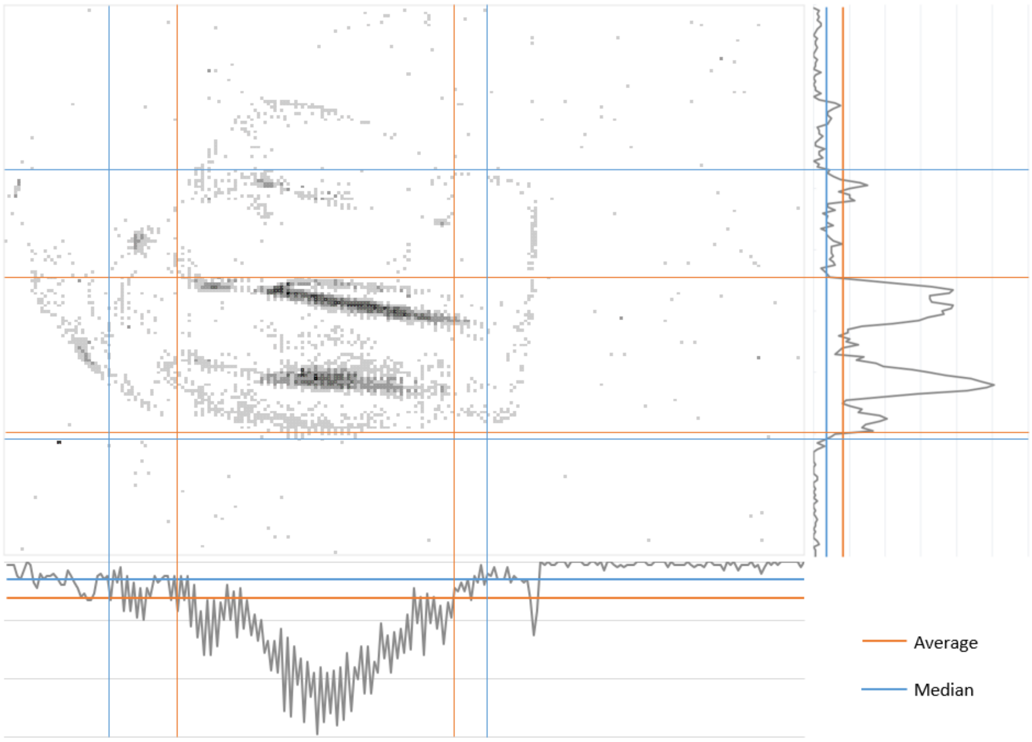


Figure 3.12: ROI created by the intersection of the grid sums and their average/median. Obtained from data of Figure 3.8(c).

From Figure 3.10, both hypothesis of using the median and the average of the sums, seem effective, having the average the advantage of being able to remove the shadows. In Figure 3.11 both lines have a very similar resulting zone. However, the object's edge could not be detected by the average line by following the rules set previously. Figure 3.12 seems to have the same issue, now for both the median and the average. Thus, a rule of joining multiple zones should be added to correct this.

To avoid having a lot of limits to deal with, due to joining multiple regions of interest, only the ones that contribute with more than a certain number of events, α, should be joined. This also has the purpose of preventing noise to accidentally create a zone. At the end of the algorithm, the remaining zones are joined into a single continuous zone, as detailed in Algorithm 3.

Algorithm 3 Creation of Region of Interest with multiple zones, using a threshold. Exemplified for the vertical limits of the ROI. The horizontal limits are obtained similarly.

```
Sum the values of each column of the grid
Find the average of the sums $\mu$
Define threshold $\alpha$

for each column of the grid
    Build an array with the sum value of the current and previous two columns
    if all values in the array are greater than $\mu$ then
        Save starting column of the cluster temporarily
    elseif all values in the array are less than $\mu$ then
        Save ending column of the cluster temporarily
        if size of cluster is bigger than threshold $\alpha$ then
            Save both start and end of cluster permanently
        end
    end
end

Use the first column of the earliest cluster and the last column of the last cluster to
define the ROI
```

The two hypotheses, average and median, are compared again to decide which is the better limiter, now considering the rule of joining multiple zones, Figure 3.13. The values of $\alpha=50$ and $\alpha=100$ have been tested as the minimum number of events to entitle as a zone.

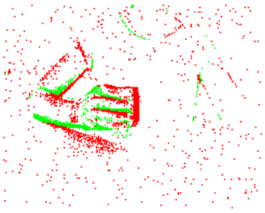
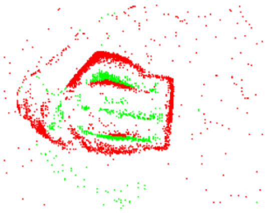
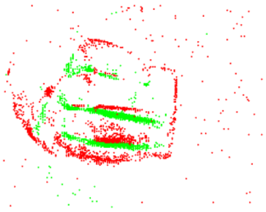
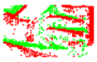
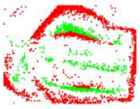
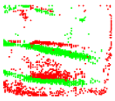
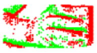
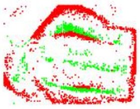
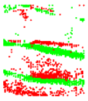
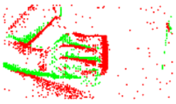
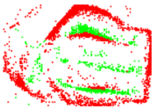
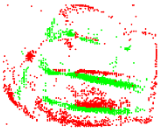
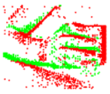
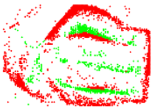
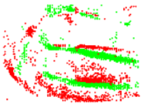
Original			
Average $\alpha=50$			
Average $\alpha=100$			
Median $\alpha=50$			
Median $\alpha=100$			
	(a)	(b)	(c)

Figure 3.13: Results of comparing average and median NR with $\alpha=50$ and $\alpha=100$. Obtained from data of Figure 3.8(a), Figure 3.8(b), and Figure 3.8(c), respectively.

From Figure 3.13 it is possible to conclude that the best combination of parameters is that of a minimum number of events per zone of *α=50* using as limit the average. This is made clear by the fact that, in both cases of using the median, the shadow of the hand is still clearly visible, Figure 3.13(a). Even though using the median with *α=50* has the best result in Figure 3.13(c), its noise output at the other frames is not acceptable. While using the average with *α=100* has great performance in its initial frames, like Figure 3.13(b), it loses almost all useful information in Figure 3.13(c). Thus, using the average as the limit with *α=50* is the best at preserving useful information, while managing to remove almost all noise.

All the other tasks have been tested with the limit being average and α=50. The results are shown in Appendix A. The results imply that the noise reduction algorithm does a good job of removing body movement noise and manages to frequently remove the shadows. The algorithm seems to have the most trouble with the Screw task, because the body movement is much more prominent. When idle, two different behaviours for the NR algorithm have been observed. When the image has no distinct zone of many events, the algorithm, as it depends on an average, only slightly crops the image or doesn't crop the image at all. But when little movement occurs, the algorithm locates it and crops the image accordingly.

To summarize, the first operation within the noise reduction algorithm is creating a grid representative of the event count at each coordinate. Within this grid, the algorithm sums the value of each cell belonging to the same column (corresponding to the x coordinate). The algorithm then checks for high consecutive sums, compares it against its average, and outputs the limits in which they are contained. The same procedure is done to the rows of the grid (correlated to the y coordinates). The last operation is that of preserving the original data within the limits defined by the algorithm. The core operations of algorithm are described in Algorithm 3.

3.4. Data Dimensionality Reduction

In another attempt to improve classification results of the original frames, obtained by creating integrated images with 3000 Events, the previously selected DDR techniques, PCA and Kernel PCA, are applied to the original frames. This is done, as explained before, to make full use of the EC's event 4D data and, as such theoretically provide better insight into the correlation of data, improving classification accuracy.

The frames that will be altered with the DDR techniques are the frames obtained by creating integrated images with 3000 Events. In both cases, the high dimensional space will be 4D, to use the full extent of the data available, and the lower-dimensional space will be 2D, as the intended result of both DDR techniques is to obtain 2D frames for classification through the DNN. The PCA and Kernel PCA methodologies will be separately discussed.

3.4.1. DDR with PCA

As a quick recap, PCA takes a high dimensional space and defines a corresponding lower-dimensional space by using principal components, which correlate to the highest variance in the data. As the lower-dimensional space should be 2D, at most 2 Principal Components will be generated.

When using PCA it is important for all the data to be normalized, for all data parameters to be considered equally relevant. If the data have different orders of magnitude, the results of PCA will be biased towards favouring the data with the highest order of magnitude, due to PCA's linear nature. The range of 3 of the parameters is constant and, as such, known. The x parameter is ranged between 1-240 and the y parameter has values between 1-180, both representative of the EC's resolution. The polarity is a binary variable, only including values of either 0 or 1. The timestamp parameter, ts, although being always limited between a minimum and maximum time and being ordered in a roughly continuous manner, does not have a fixed range when comparing different frames.

The most straightforward way of normalizing the EC's 4D data is to turn all the parameters' ranges into the common range of 0 to 1. This way all the data parameters are considered equally significant when applying PCA. The polarity doesn't have to be normalized as it is already within this range. However, as the polarity is a binary variable, the data will have a high variance in this direction, resulting in the first Principal Component to be heavily skewed in the polarity direction. As seen in Figure 3.14, this makes the frames from different tasks very similar. Thus, to represent polarity, the most appropriate course of action is to use colours as was done previously with both original and noise reduced frames.

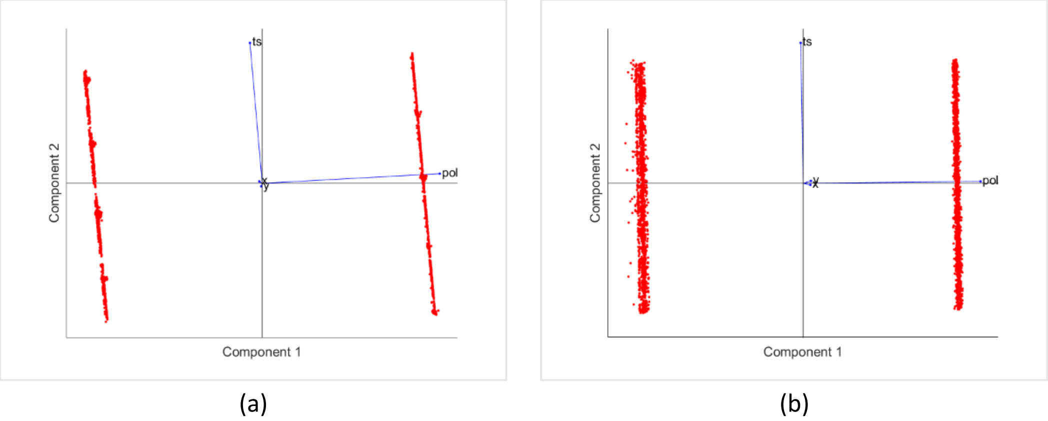


Figure 3.14: Results of PCA with all parameters normalized within the range of 0-1: (a) Move left task; (b) Lift task.

Using colour to represent the polarity means that PCA will only be applied to 3 of 4 parameters of the 4D data, the x and y coordinates, and the timestamp. The results obtained using this approach, Figure 3.15, are much more visually distinct than in their counterparts in Figure 3.14. Even though the first principal component is almost coincident with the timestamp direction, now the tasks are distinguishable between each other and, thus, appropriate for classification purposes. The images in the top row of Figure 3.14 show the contribution of each parameter to each principal component, while the images in the bottom row show the images as will be used for classification.

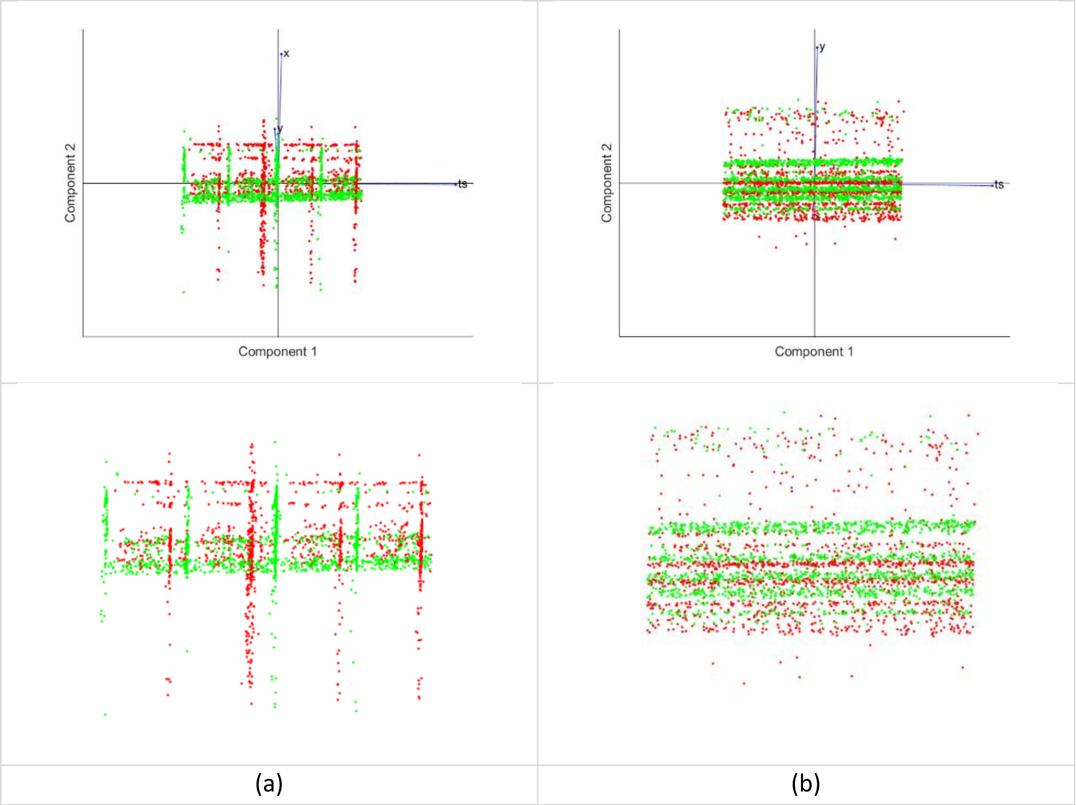


Figure 3.15: Results of PCA with the x, y and timestamp parameters normalized within the range of 0-1: (a) Move left task; (b) Lift task.

3.4.2. DDR with Kernel PCA

Kernel PCA maps the original data points into a high-dimensional space using a kernel function, typically a Polynomial or Gaussian function, and then performs PCA on the newly created feature space. All computations are done through the kernel function in input space. Unlike PCA, a non-linear DDR method such as Kernel PCA works well with linearly inseparable data, and as such the polarity can be used as a parameter in the algorithm. It is still convenient to do normalization to work with parameters of similar orders of magnitude. For Kernel PCA's case, the EC's output data will be normalized by scaling each parameter using its standard deviation, followed by centering its data. This kind of normalization, when obtaining Kernel PCA frames, reduced the processing time from 20 seconds/frame, for normalizing to a 0-1 range, to 14 seconds/frame.

The Kernel PCA algorithm follows 4 main steps. First, the kernel function is chosen, next, the centered kernel matrix is constructed, then the eigenvalue problem is solved and finally, the data points are represented in the desired dimension. The computational part will not be elaborated on. Instead, the selection of kernel function type and its hyperparameter will be discussed, followed by the frames used for classification.

3.4.2.1. Gaussian Kernel Function

Gaussian kernels are some of the most well-known kernel functions. Also known as Radial Basis Function (RBF) kernel, they have a generalized structure (2) which is used to calculate the kernel matrix $k(x, x')$ for each pair of samples, x and x',

$$k(x, x') = \exp(-\gamma \| x - x' \|^2) \tag{2}$$

Where γ represents how "wide" the Gaussian function is. This means γ controls how the decision boundary[4] is affected by the data, represented in Figure 3.16 by a solid line. The higher the value of γ, the more "wiggling" the boundary will be [51].

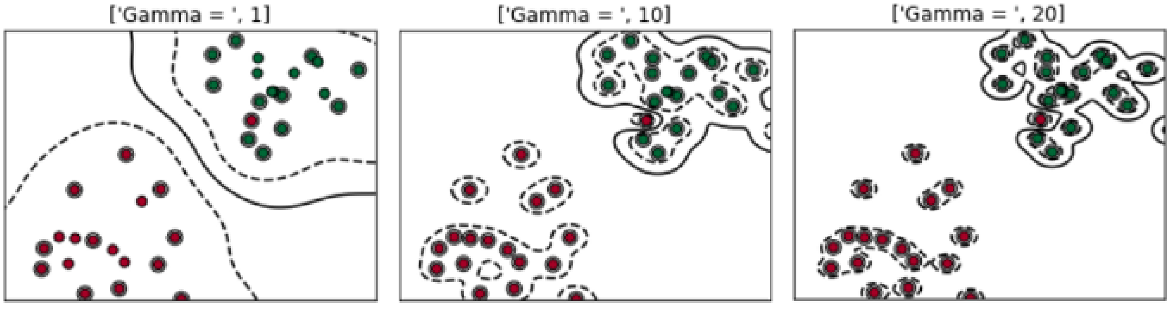


Figure 3.16: Decision boundaries for different values of γ.

[4] The hypersurface that partitions the vector space within the two different sets, one for each class. Linear decision surfaces within a high-dimensional feature space correspond to non/linear decision surfaces in the original feature space [52].

Gaussian functions have a single hyperparameter, being easier to readjust for a specific task in comparison to a polynomial kernel which needs, at least, two hyperparameters.

3.4.2.2. Hyperparameter Value

The smaller the chosen value of γ, the further away the points can be from the decision boundary to influence it. This leads to a relatively smooth decision boundary but may overlook some class outliers. For higher values of γ only the data points close to the decision boundary influence it. This promotes an irregular decision boundary but deals much better with class outliers.

In the case of EC output data, the two different values of polarity are the two classes considered. From this, it is possible to deduce that a relatively high value of γ will be needed to effectively separate these classes, as each class of polarity is not easily contained within a single zone. To confirm this, the first frames resultant from Kernel PCA from 3 sequences of different tasks are compared against each other with three different values of γ. The results are shown in Figure 3.17, where blue dots represent polarity = 1 and red dots represent polarity = 0.

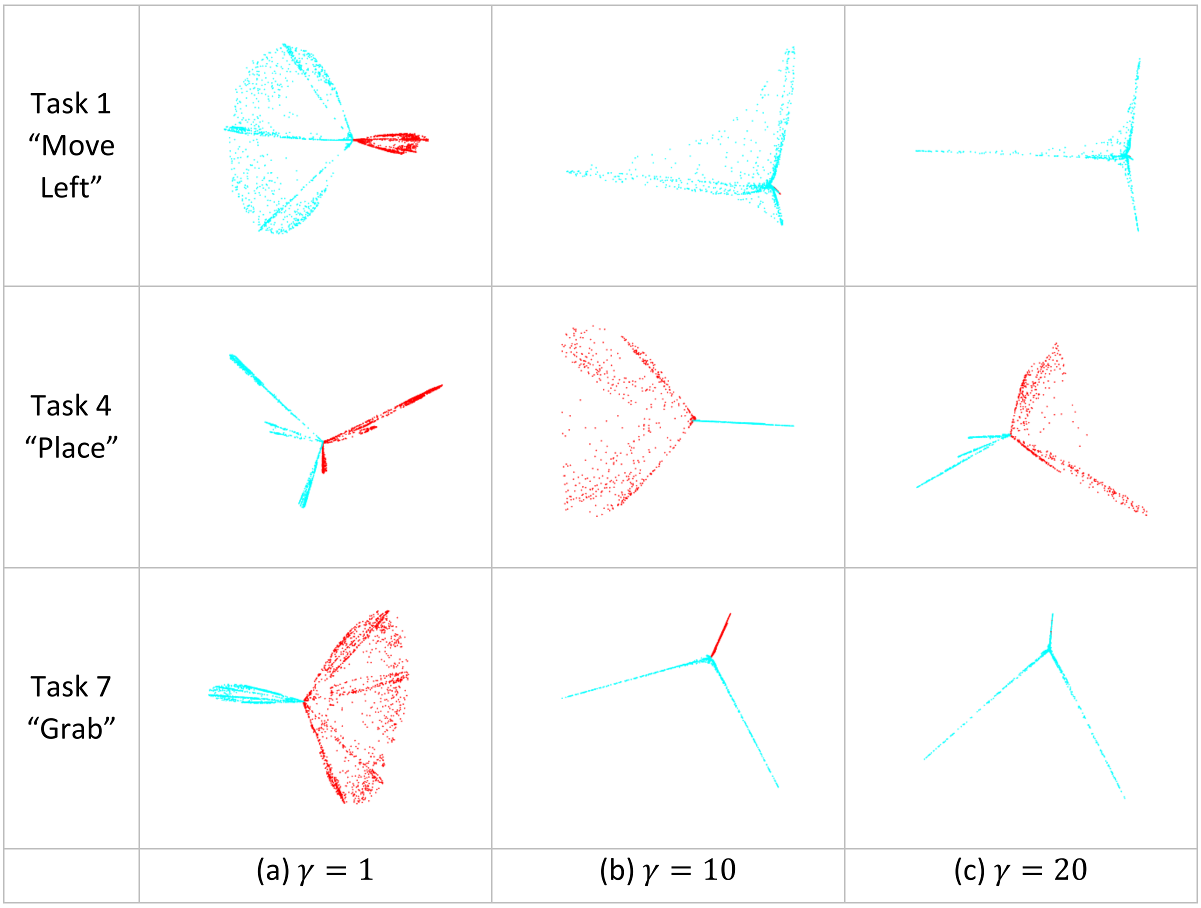


Figure 3.17: Comparison of Kernel PCA results for different values of γ.

As can be seen in Figure 3.17 higher values of γ do, in fact, perform better in creating distinctive frames in comparison to lower values. The results obtained for $\gamma = 10$ and $\gamma = 20$ are very similar. In this case, the value of $\gamma = 10$ will be selected as the preferred option because, by creating a smoother decision boundary, it is expected to behave more consistently when applying to different types of tasks.

It is noteworthy that the results obtained within similar conditions for PCA, Figure 3.14, were not conclusive, because of the naturally high variance of the polarity, due to its binary nature, and because PCA is a linear method. Kernel PCA, being a non-linear method, was capable of separating the two classes of polarity relatively well.

3.5. Construction of the LRCN

A popular approach for image classification is the usage of DNNs. While CNNs are applicable for image feature extraction, RNNs are used to learn temporal sequences, Figure 3.18. Joining these two types of DNN creates an LRCN, which is used for video classification. The LRCN created in this section is optimized for the dataset without any data modifications, i.e. without NR nor DDR.

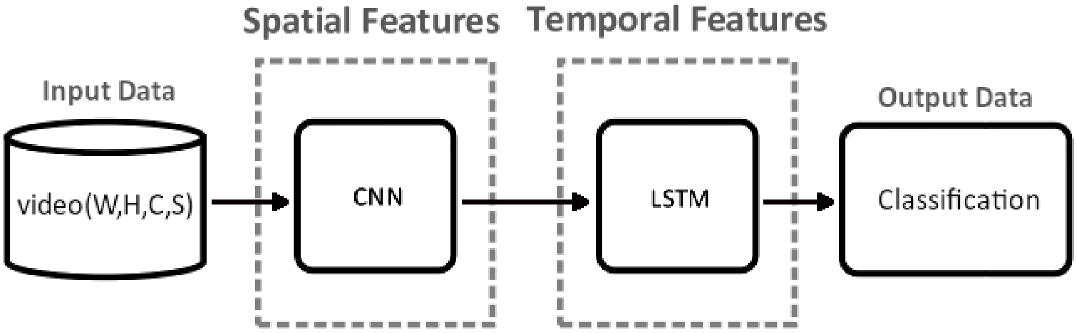


Figure 3.18: Structure of the LRCN. Adapted from [39].

As such, to create the LRCN that will be used to classify the tasks included in the dataset, both a CNN and an RNN are needed. A CNN trained from scratch would need a significant amount of training data. Instead, a pre-existing CNN, fine-tuned through Transfer Learning, will be used. The RNN will be of the LSTM type and will be trained on this work’s dataset. Both Transfer Learning and the combination of the CNN with the LSTM to create the LRCN are accomplished using the example shown in [53] as a guide and will therefore not be further elaborated on.

3.5.1. Preparation of the Input Data

For a DNN to classify anything, all input data must be within a specific format. In this case, it should be the same type of input as requested by the chosen CNN. They typically require a square RGB image input with sizes of about 230x230 pixels. CNNs can also use a video as input, classifying all the frames that constitute it.

The ideal case, for the dataset presented in this study, would be to take the raw data created by the EC and, through the sole use of algorithms, directly create the correctly formatted video file without the need to create the images associated with the many frames. However, by attempting this, the CNN did not show differences in activations, being unable to classify. Thus, from the original EC files, all individual frames are converted into image files, with their respective frame changes. These image files are then recombined into video files with the format needed for CNN classification. This was done for both datasets and for all types of frames previously discussed in this work. This includes original frames without alterations, frames with noise reduction, frames obtained through PCA, and frames obtained through Kernel PCA.

3.5.2. Partition of Training, Validation, and Testing Data

The training data are used to train the networks, the validation data is used to continuously evaluate the network while it is training, and the testing data is used to verify the final LRCNs' performance. The data from the main dataset will be divided into two partitions, training and validation data. This is a simple cross-validation technique, known as the holdout method. As the main dataset has a limited amount of data, the training portion should be large compared to the validation, to preserve as much data for training as possible. As such, an 85% training – 15% testing division seems suitable, Figure 3.19.

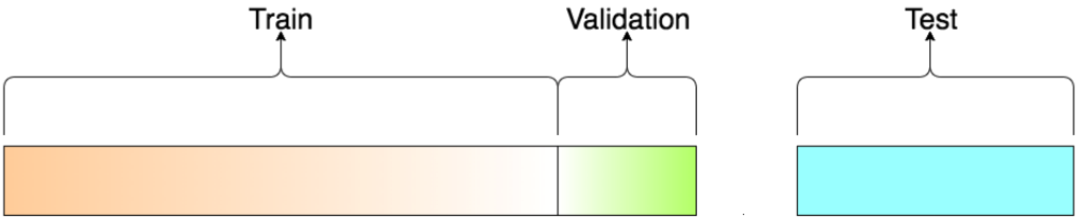


Figure 3.19: Division of the datasets into training, validation and testing data [54].

When separating the data into portions, if the selection of data is done randomly, there is a high risk of imbalance of the class representation within both training and validation sets. To counter this problem, 15% of the data from each task is selected, rounded down to 3 sequences per task. This method also ensures that no tasks are, by chance, excluded from the validation set. The partition is then reused for each network training attempt. With this approach, instead of using the more effective approach of using multiple partitions and taking the average, there is a significant save in computational time.

3.5.3. Selection of the CNN

For the selection of the CNN, two well-known pre-trained networks will be considered, GoogLeNet [55] and ResNet50 [30], with basic properties shown in Table 3.4.

Table 3.4: Main network properties of the selected CNNs[5] [56].

	Network Depth[6]	Validation Accuracy	Prediction Time relative to SqueezeNet
GoogLeNet	22	67 %	2
ResNet50	50	75%	3,5

The objective is to strike the balance between accuracy and computational time. Simpler architecture is also essential as it assists in reducing overfitting, which happens when the model recognizes specific images instead of general patterns. Overfitting is most prevalent when using small datasets, such as the one used in this work. By having a 50 layers depth, ResNet50 is quite competent in classifying objects with high-level features [57]. GoogLeNet, by having less network depth and simpler architecture, is faster and tends to obtain good scores for a small training sample size [57].

The most important factor is, naturally, how well the network classifies the data of the primitive manufacturing dataset. To test this, a default LSTM was trained using the activations from either CNN. The LSTM used is identical to the one in [53], only changing the number of hidden units to 250, as it provided good results. The LSTM is also tested for different quantities of BiLSTM layers. The average validation score and elapsed time of the LSTM, for three training runs, is shown in Figure 3.20.

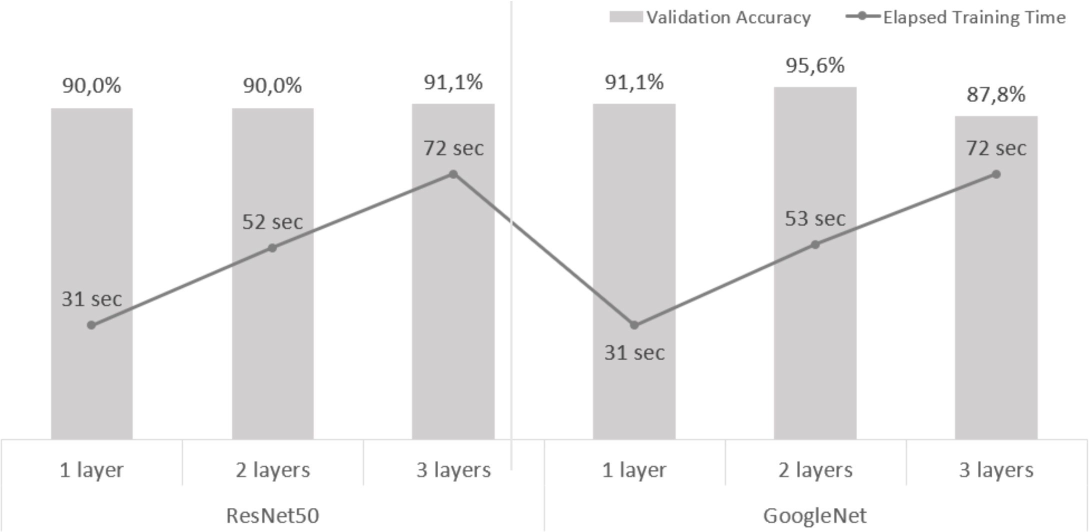


Figure 3.20: LSTM validation accuracy and elapsed time for GoogLeNet and ResNet50.

Although the validation scores can all be considered acceptable, the highest scoring combination between CNN and LSTM layer depth will be chosen, which is GoogLeNet + LSTM with 2 BiLSTM layers, achieving 96% validation accuracy.

[5] Results obtained for ImageNet database.

[6] “The largest number of sequential convolutional or fully connected [56].

3.5.4. Creation of the LSTM network

After the selection of the CNN, to finish constructing the DNN, an LSTM has to be built. To accomplish this, both necessary LSTM layers and the most applicable hyperparameters must be selected. The general structure of the LSTM is shown in Figure 3.21, taking into account the 2 BiLSTM layers detailed in the previous sub-section 3.5.3.

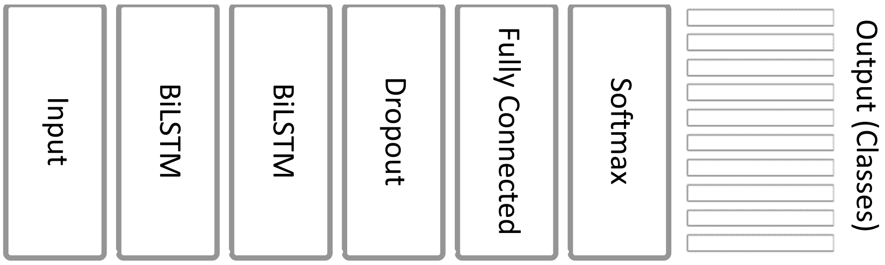


Figure 3.21: Architecture of the LSTM network.

3.5.4.1. LSTM network layers

First off, to build an LSTM network, a layer is needed to input data to the network and another to output data from the network. The Input layer should match the number of features generated by the CNN's output layer and the Output layer is the layer that will reveal the classification results.

The most essential part of an LSTM type of RNN network is, naturally, its LSTM-related layers. Two different kinds of LSTM layers can be used, Figure 3.22, either simple LSTM layers or their bidirectional variants, BiLSTM layers. Differentiating themselves from the original simple version, BiLSTM layers, instead of only learning forward-in-time dependencies between time steps, learn in both temporal directions. This way BiLSTM layers enable additional training by traversing the input data twice [58].

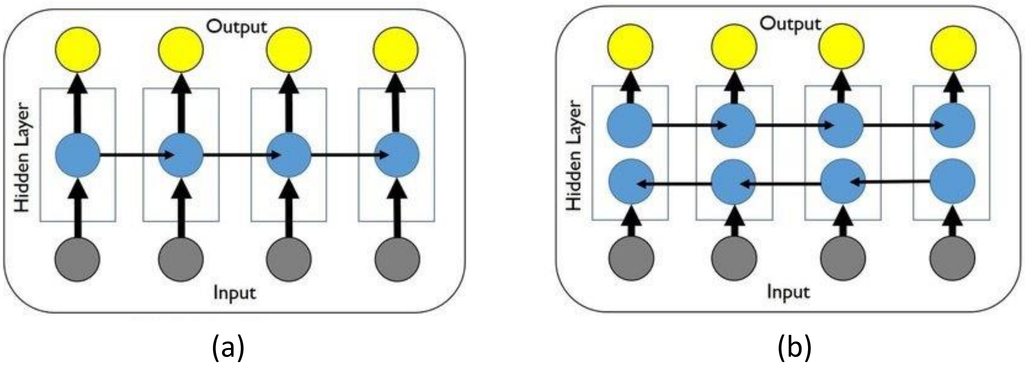


Figure 3.22: The two types of LSTM layers [59]: (a) LSTM layer; (b) BiLSTM layer.

The Dropout layer is used to reduce overfitting by dropping out a percentage of units from its preceding layer. For hidden layers, as is the case for the BiLSTM layer, a usual value is 50% dropout. The fully connected layer then, through the use of weights and biases, is used to predict the classes, using the Softmax layer to turn the numeric results from the fully connected into probabilities to determine the classification results.

3.5.4.2. Learning Rate and Number of Epochs

The two hyperparameters that, together, most influence the LSTM's training behaviour are the learning rate and the number of epochs. The learning rate controls how fast the network "learns", representing how much the model is allowed to change at each iteration. For a high learning rate, the LSTM is more sensitive towards new information, which makes it unstable. For low learning rates, the network may take much longer to converge, needing more iterations.

The number of epochs is the number of times the network runs through the entire training dataset. The more epochs used while training, the more likely the network will attain convergence, especially when using lower learning rates. Using too many epochs may, however, be adding unnecessary computational time if convergence is already obtained far earlier than the stipulated number of epochs.

To study the parameters' simultaneous impact on the LSTM training, the averaged validation accuracy and elapsed time results of three training runs are evaluated for a selection of learning rates with different number of epochs, Figure 3.23.

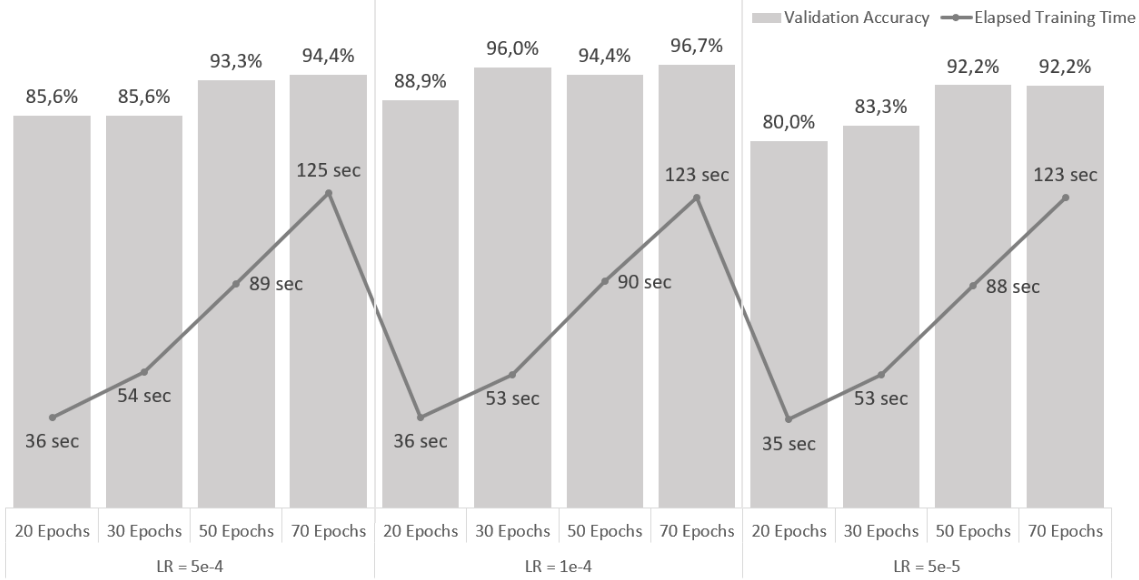


Figure 3.23: LSTM validation accuracy and elapsed time for different combinations of learning rates (LR) and number of epochs.

From analyzing the outcomes, shown in Figure 3.23, it stands out immediately that the learning rate that achieves the best results is that of LR = 1e-4. In general, the best classification results are obtained with 30 and 70 Epochs. However, for the same learning rate, an unexpected decline in classification accuracy occurs for 50 Epochs. For this reason, a more detailed analysis of the variance for each epoch's training results is made for LR = 1e-4, now using ten classification models. The average classification accuracy achieved by the models are presented in Figure 3.24. The computational times obtained were approximately the same as the ones obtained for Figure 3.23.

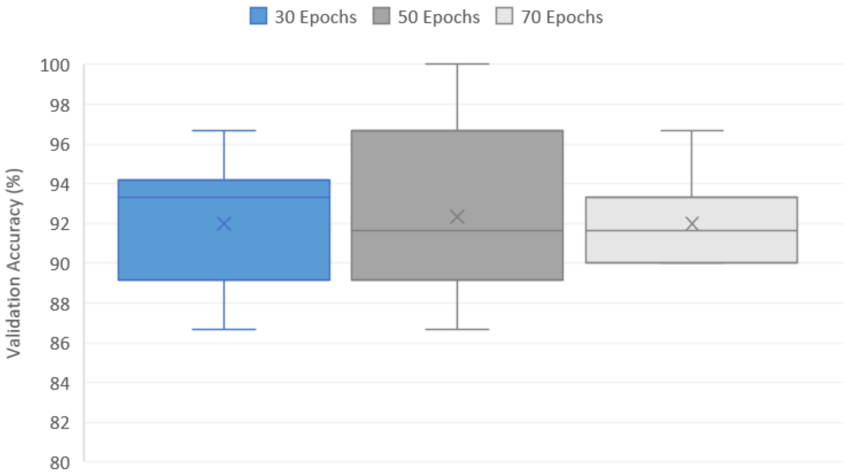


Figure 3.24: Validation accuracy variance for different values of the number of epochs with LR = 1e-4.

This graph gives a lot of insight into the effect of the epochs on the variance of training results of an LSTM network. The cross represents the average value of classification accuracy, which is approximately the same for all epoch values, $\approx 92\%$. A low variance option is more beneficial because it means the network has a more consistent behaviour. As a result, 50 epochs should not be chosen. Its variance, with a maximum value equal to 100% and minimum value of 86,7%, leads to the lower validation accuracy in Figure 3.23. Similarly, 30 Epochs models should also not be chosen because they present a higher variance than the 70 Epoch models.

It is worth noting that there is some difference between the average results shown in Figure 3.23 and Figure 3.24, showing a reduction from $\approx 96\%$ to $\approx 92\%$ classification accuracy. This means there is some variability regardless of the choice made. Also, due to few validation data, each failed validation reduces the validation score by about 3,3 percentage points, which causes a lot of discrepancy in the results. As a conclusion, next optimization attempts will be made averaging ten runs instead of three, to avoid irregular behaviour such as the one seen in Figure 3.23.

3.5.4.3. Number of Hidden Units

The BiLSTM layers have an important hyperparameter that must be defined, which is the number of hidden units, also called nodes, present within each one of the layers. With too few nodes, the layer will struggle to capture complex information. The presence of too many nodes, however, leads to an increase in computational time and a tendency to overfit the training data. To simplify this problem, the same quantity of hidden units will be selected for both BiLSTM layers. The results for different values of hidden units are compared against each other, Figure 3.25. The objective is to find the minimum number of hidden units needed to obtain consistently good classification results, without causing overfitting of the network and high computational times.

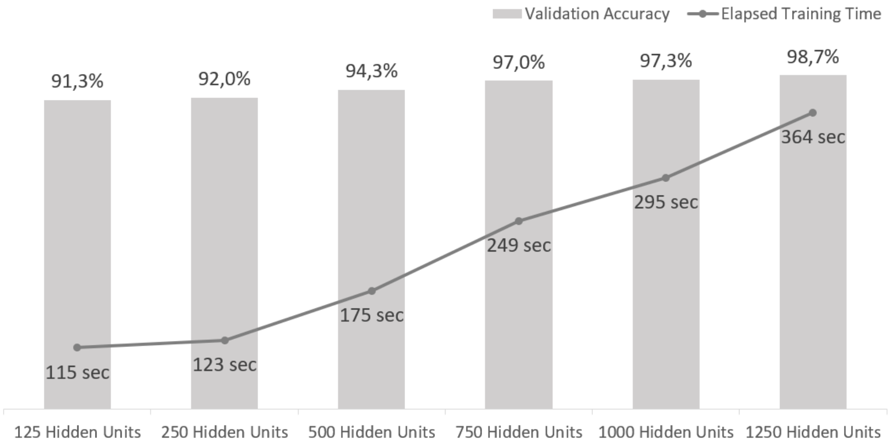


Figure 3.25: LSTM validation accuracy and elapsed time for different quantities of hidden units.

From Figure 3.25 it is possible to confirm that a higher number of hidden units tends to improve validation accuracy at a cost of computational time. Values of 750 or more hidden units attained good validation accuracies, each having multiple training runs with 100% validation accuracy. The best results were obtained for 1250 hidden units, although at some computational time cost. No further increase of hidden units is needed as classification accuracy is unlikely to improve, while run time and the risk of overfitting the LSTM will inevitably increase.

3.5.4.4. Mini-Batch Size

The last hyperparameter deemed essential to optimize is the mini-batch size. The mini-batch size is the number of training samples used at each iteration to update the network. Dividing the training data into multiple subsets has the advantage of being more computationally efficient than using the entire dataset each iteration, attaining faster convergence and using fewer data at each iteration. The mini-batch size should not be too small to avoid inaccurate estimations. Usually, the value of the mini-batch size is a number that is a power of two, like 16 or 32 [60]. The results of training with three different mini-batch sizes are shown in Figure 3.26.

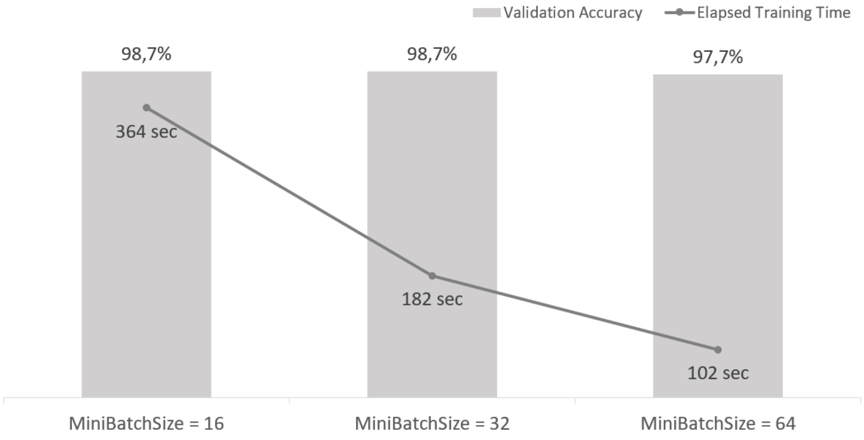


Figure 3.26: LSTM validation accuracy and elapsed time for different mini-batch size values.

It is quite evident in Figure 3.26 that an increase in mini-batch size significantly reduces computational time, without reducing the validation accuracy by much. The more the values approach the training dataset size, which in this case is 200, validation accuracy drops. For this reason, the value of 32 samples for each mini-batch is chosen.

3.5.5. LRCN model

The following schematic, Figure 3.27, summarizes the final composition of the LRCN that will be used to obtain the results. The hyperparameters chosen for the LSTM are compiled in Table 3.5.

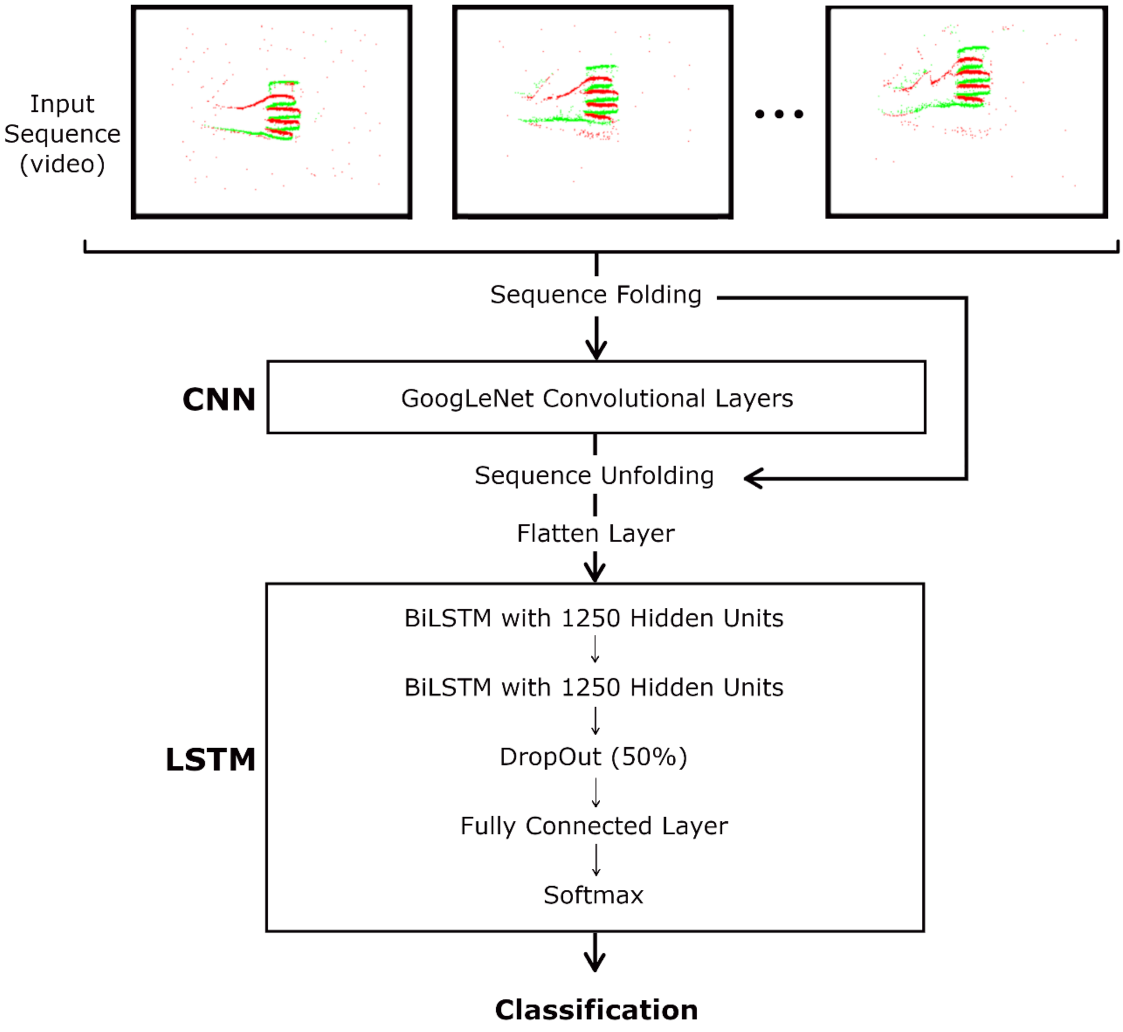


Figure 3.27: Complete LRCN schematic.

Table 3.5: LSTM training hyperparameters.

Learning Rate	Number of Epochs	Mini-batch Size
0,0001	70	32

4. RESULTS AND DISCUSSION

The EC data were retrieved with a Java software package made for Address-Event Representation called jAER [61]. All data processing was achieved using MATLAB software. The networks were trained and tested by using resources from the MATLAB Deep Learning Toolbox [62]. This toolbox also contained the models for the GoogLeNet network [63] and the ResNet-50 network [64]. The hardware used was a computer with an Nvidia GeForce RTX 2080 Ti GPU, an AMD Ryzen 9 3900X CPU and 64GB of RAM. Its GPU was used both for network training and testing.

While, ideally, each type of data alteration should have its optimized LSTM, the same LSTM that was fitted to the original data will be reused for all LRCNs. The DNN is trained on its specific data type for all 10 categories and then tested with the testing dataset. The results are shown in Figure 4.1. Both LSTM training times and LRCN run times were very similar between model types, averaging around 200 seconds and 16 seconds respectively.

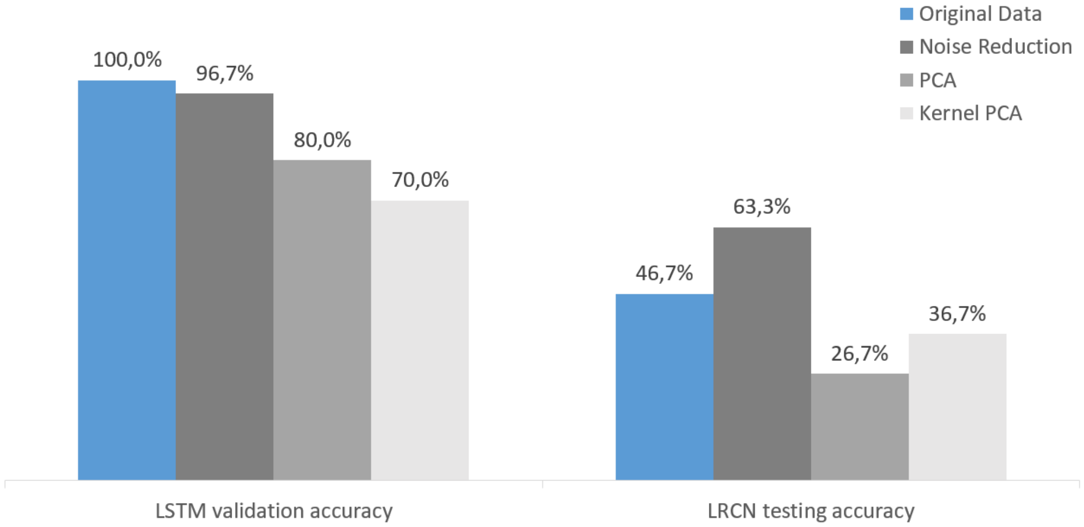
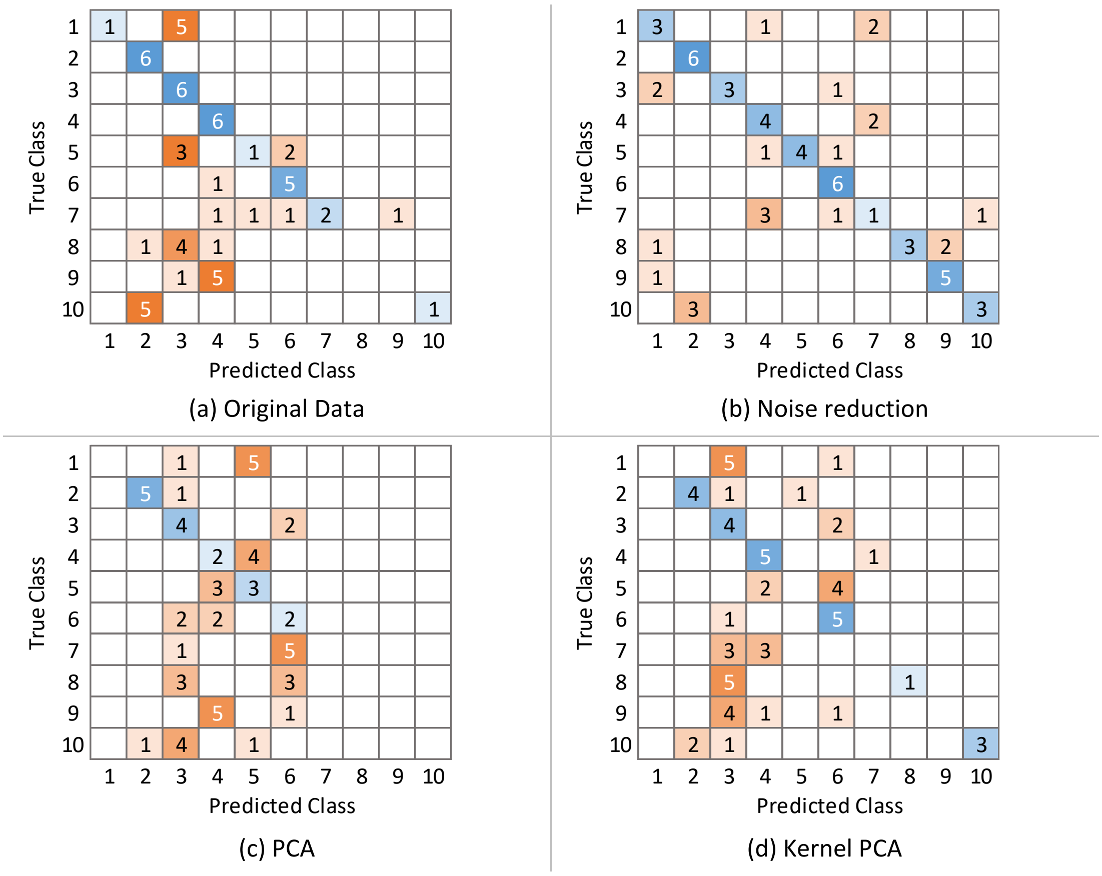


Figure 4.1: Results obtained for the different frame types using the LRCN built in section 3.5.

The LSTM networks do not provide acceptable performance for both PCA and Kernel PCA data, achieving 80% and 70% accuracy respectively, as their data are different in contrast to the original data. Better results should be expected with an LSTM optimized for each DDR techniques and/or a different CNN. The noise reduced frames have a good LSTM accuracy, as their general data structure is similar to the original data.

The testing accuracy obtained for all the model types is significantly lower than their LSTM counterparts. The best performing model type is the model that implemented noise reduction. Removing non-essential sections of each frame from the original data improves the network's accuracy by a significant 17 percentage points.

Both the PCA and Kernel PCA networks performed worse than the original network, losing accuracy of 20 and 10 percentage points respectively. Kernel PCA managed to perform better than PCA, presumably due to its capability of dealing with non-linear correlations. For a better understanding of the results of the LRCN testing accuracy, the resulting confusion matrices of each model type are examined, Figure 4.2.



True Class	1	2	3	4	5	6	7	8	9	10
1	1		5							
2		6								
3			6							
4				6						
5			3		1	2				
6				1		5				
7				1	1	1	2		1	
8		1	4	1						
9			1	5						
10		5								1

Predicted Class

(a) Original Data

True Class	1	2	3	4	5	6	7	8	9	10
1	3			1			2			
2		6								
3	2		3			1				
4				4			2			
5				1	4	1				
6						6				
7				3		1	1			1
8	1							3	2	
9	1								5	
10		3								3

Predicted Class

(b) Noise reduction

True Class	1	2	3	4	5	6	7	8	9	10
1			1		5					
2		5	1							
3			4			2				
4				2	4					
5				3	3					
6			2	2		2				
7			1			5				
8			3			3				
9				5		1				
10		1	4		1					

Predicted Class

(c) PCA

True Class	1	2	3	4	5	6	7	8	9	10
1			5			1				
2		4	1		1					
3			4			2				
4				5			1			
5				2		4				
6			1			5				
7			3	3						
8			5					1		
9			4	1		1				
10		2	1							3

Predicted Class

(d) Kernel PCA

Figure 4.2: Confusion matrices obtained for each LRCN model.

Immediately the noise reduction model, Figure 4.2(b), stands out as being the most consistent by managing to correctly classify all classes multiple times, with Task 7 being the exception. Wrong guesses of classification are mostly unbiased and distributed between the classes. The original data model, considering its LSTM validation accuracy of 100%, should have the capability to recognize all classes. However, as seen in Figure 4.2(a), not a single correct prediction is made for Task 8 and Task 9. Also, a high tendency to predict with either Tasks 2, 3 or 4 can be observed.

The worst performing model is the PCA model, Figure 4.2(c), which only predicts within classes 2 to 6, disregarding 50% of the classes. This poor prediction behaviour may mean that the frames are too similar, which makes the model's predictions default to the classes with most events, Tasks 2 to 6, as they have more data. A slightly better but similar behaviour can be seen in Figure 4.2(d), for Kernel PCA. The predictions still occur mostly on Tasks 2 to 6, but some predictions start to occur outside this range.

5. CONCLUSIONS AND FUTURE WORK

The purpose of this study was to investigate the effects of data manipulation on the speed and accuracy of EC data classification. The best performing LRCN model has a testing accuracy of 63,3%, achieved through the use of noise reduction. The noise reduction algorithm was proven to be effective in removing most noise, as is proved by the results presented in Appendix A. This improved the network's accuracy by a significant 17 percentage points in comparison to the model created for the original data. This method shows the most potential for future studies. The main suggestion is to use the trajectory of the centre of each frame's ROI to predict the gesture class by purely analytical means. This way a DNN would not be needed and classification could be faster and more accurate.

Both PCA and Kernel PCA had an adverse effect on classification accuracy, lowering it 20 and 10 percentage points respectively. Most likely, the discrepancy is caused by the loss of information when transforming the EC data through a DDR method. This change also alters the data's nature, which might make it incompatible with an object-recognition CNN such as GoogLeNet. As such, the results from PCA and from Kernel PCA should improve by using an LSTM without a CNN. However, Kernel PCA is not recommended for continuous gesture recognition, as its eigenvalue function takes too long to process. Another problem might have stemmed from the small number of events that compose each frame. As the timestamp is relevant for the DDR, having a very limited time range limits the DDRs applicability.

The most likely cause of the general subpar results is the small training sample size. Considering the very high number of 10 task classes, the training dataset should have been much more substantial to provide enough data for each class. For future work the main suggestion is, therefore, to use a larger and more diverse dataset that includes more users and many more sequence recordings. To the tasks proposed in this study should also be added a "non-task" or ME class, which is essential for sequences of tasks to be tested. These two aspects are fundamental to validate if the proposed system could be used for continuous real-life gesture recognition.

To test the issues with the LRCN classification accuracies, a simple single-layer network should be used to evaluate if any of the chosen parameters were skewing the results. Also, it is recommended to use other approaches for measuring computational complexity. Rather than measuring computational time through elapsed time, it is advised to, for example, count floating-point operations per second (FLOPS).

6. REFERENCES

[1] T. Ende, S. Haddadin, S. Parusel, T. Wusthoff, M. Hassenzahl, and A. Albu-Schaffer, “A human-centered approach to robot gesture based communication within collaborative working processes,” in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2011, pp. 3367–3374, doi: 10.1109/IROS.2011.6094592.

[2] W. Kim *et al.*, “Adaptable Workstations for Human-Robot Collaboration: A Reconfigurable Framework for Improving Worker Ergonomics and Productivity,” *IEEE Robot. Autom. Mag.*, vol. 26, no. 3, pp. 14–26, 2019, doi: 10.1109/MRA.2018.2890460.

[3] Y. Song, D. Demirdjian, and R. Davis, “Continuous body and hand gesture recognition for natural human-computer interaction,” *ACM Trans. Interact. Intell. Syst.*, vol. 2, no. 1, 2012, doi: 10.1145/2133366.2133371.

[4] M. T. Wolf, C. Assad, M. T. Vernacchia, J. Fromm, and H. L. Jethani, “Gesture-based robot control with variable autonomy from the JPL BioSleeve,” in *2013 IEEE International Conference on Robotics and Automation*, 2013, pp. 1160–1165, doi: 10.1109/ICRA.2013.6630718.

[5] L. D. Riek, T. C. Rabinowitch, P. Bremner, A. G. Pipe, M. Fraser, and P. Robinson, “Cooperative gestures: Effective signaling for humanoid robots,” *5th ACM/IEEE Int. Conf. Human-Robot Interact. HRI 2010*, pp. 61–68, 2010, doi: 10.1145/1734454.1734474.

[6] R. Yang, S. Sarkar, and B. Loeding, “Enhanced level building algorithm for the movement epenthesis problem in sign language recognition,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, 2007, doi: 10.1109/CVPR.2007.383347.

[7] P. Neto, D. Pereira, J. N. Pires, and A. P. Moreira, “Real-time and continuous hand gesture spotting: An approach based on artificial neural networks,” *Proc. - IEEE Int. Conf. Robot. Autom.*, pp. 178–183, 2013, doi: 10.1109/ICRA.2013.6630573.

[8] M. Simão, P. Neto, and O. Gibaru, “Using data dimensionality reduction for recognition of incomplete dynamic gestures,” *Pattern Recognit. Lett.*, vol. 99, pp. 32–38, Nov. 2017, doi: 10.1016/j.patrec.2017.01.003.

[9] S. Calinon and A. Billard, “Incremental learning of gestures by imitation in a humanoid robot,” *HRI 2007 - Proc. 2007 ACM/IEEE Conf. Human-Robot Interact. - Robot as Team Memb.*, pp. 255–262, 2007, doi: 10.1145/1228716.1228751.

[10] C. R. G. Dreher, M. Wächter, and T. Asfour, “Learning Object-Action Relations from Bimanual Human Demonstration Using Graph Networks,” *IEEE Robot. Autom. Lett.*, vol. 5, no. 1, pp. 187–194, 2020, doi: 10.1109/LRA.2019.2949221.

[11] P. Gustavsson, M. Holm, A. Syberfeldt, and L. Wang, “Human-robot collaboration - Towards new metrics for selection of communication technologies,” in *Procedia CIRP*, 2018, vol. 72, pp. 123–128, doi: 10.1016/j.procir.2018.03.156.

[12] A. Amir *et al.*, “A low power, fully event-based gesture recognition system,” in *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, 2017, vol. 2017-Janua, pp. 7388–7397, doi: 10.1109/CVPR.2017.781.

[13] J. Conradt, M. Cook, R. Berner, P. Lichtsteiner, R. J. Douglas, and T. Delbruck, “A pencil balancing robot using a pair of AER dynamic vision sensors,” *Proc. - IEEE Int. Symp. Circuits Syst.*, pp. 781–784, 2009, doi: 10.1109/ISCAS.2009.5117867.

[14] P. Lichtsteiner, C. Posch, and T. Delbruck, “A 128 × 128 120 dB 15 μs latency asynchronous temporal contrast vision sensor,” *IEEE J. Solid-State Circuits*, vol. 43, no. 2, pp. 566–576, 2008, doi: 10.1109/JSSC.2007.914337.

[15] C. Brändli, “Event-Based Machine Vision,” ETH Zürich, 2015.

[16] E. Mueggler, B. Huber, and D. Scaramuzza, “Event-based, 6-DOF pose tracking for high-speed maneuvers,” *IEEE Int. Conf. Intell. Robot. Syst.*, pp. 2761–2768, 2014, doi: 10.1109/IROS.2014.6942940.

[17] P. Lichtsteiner, C. Posch, and T. Delbruck, “A 128 x 128 120dB 30mW asynchronous vision sensor that responds to relative intensity change,” *Dig. Tech. Pap. - IEEE Int. Solid-State Circuits Conf.*, vol. 51, no. 7, pp. 2060–2069, 2006, doi: 10.1109/isscc.2006.1696265.

[18] P. F. Rüedi *et al.*, “A 128 × 128 Pixel 120-dB Dynamic-Range Vision-Sensor Chip for Image Contrast and Orientation Extraction,” *IEEE J. Solid-State Circuits*, vol. 38, no. 12, pp. 2325–2333, 2003, doi: 10.1109/JSSC.2003.819169.

[19] A. Censi, J. Strubel, C. Brandli, T. Delbruck, and D. Scaramuzza, “Low-latency localization by active LED markers tracking using a dynamic vision sensor,” *IEEE Int. Conf. Intell. Robot. Syst.*, pp. 891–898, 2013, doi: 10.1109/IROS.2013.6696456.

[20] R. B. Miller, “Response time in man-computer conversational transactions. Introductions and major concepts,” *Fall Jt. Comput. Conf.*, 1968.

[21] M. Litzenberger *et al.*, “Vehicle counting with an embedded traffic data system using an optical transient sensor,” *IEEE Conf. Intell. Transp. Syst. Proceedings, ITSC*, pp. 36–40, 2007, doi: 10.1109/ITSC.2007.4357700.

[22] F.-F. Li, J. Johnson, and S. Yeung, “Image Classification Pipeline.” pp. 1–65, 2018.

[23] J. Schmidhuber, “Deep Learning in neural networks: An overview,” *Neural Networks*, vol. 61, pp. 85–117, 2015, doi: 10.1016/j.neunet.2014.09.003.

[24] N. Neverova, C. Wolf, G. W. Taylor, and F. Nebout, “Multi-scale deep learning for gesture detection and localization,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 8925, pp. 474–490, 2015, doi: 10.1007/978-3-319-16178-5_33.

[25] F. Vázquez, “Deep Learning Made Easy With Deep Cognition,” 2017.

[26] P. W. Battaglia *et al.*, “Relational inductive biases, deep learning, and graph networks,” pp. 1–40, 2018.

[27] B. K. Gnanamoorthy, “RNNs to write like Shakespeare,” 2019. [Online]. Available: https://medium.com/@gnabr/rnns-to-write-like-shakespeare-226609863cd1.

[28] N. Mo, L. Yan, R. Zhu, and H. Xie, “Class-specific anchor based and context-guided multi-class object detection in High Resolution Remote Sensing Imagery with a convolutional neural network,” *Remote Sens.*, vol. 11, no. 3, 2019, doi: 10.3390/rs11030272.

[29] “MATLAB Deep Learning Onramp,” *MATLAB Deep Learning*. [Online]. Available: https://www.mathworks.com/learn/tutorials/deep-learning-onramp.html.

[30] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 2016-Decem, pp. 770–778, 2015, doi: 10.1109/CVPR.2016.90.

[31] A. Graves, A. R. Mohamed, and G. Hinton, “Speech recognition with deep recurrent neural networks,” *ICASSP, IEEE Int. Conf. Acoust. Speech Signal Process. - Proc.*, no. 3, pp. 6645–6649, 2013, doi: 10.1109/ICASSP.2013.6638947.

[32] A. Graves, “Supervised Sequence Labelling with Recurrent Neural Networks,” 2012.

[33] S. Hochreiter and J. Schmidhuber, “Long Short-Term Memory,” *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997, doi: 10.1162/neco.1997.9.8.1735.

[34] D. Eck and J. Schmidhuber, “A First Look at Music Composition using LSTM Recurrent Neural Networks,” *Idsia*, pp. 1–11, 2002.

[35] M. Liu, Y. Wang, J. Wang, J. Wang, and X. Xie, “Speech Enhancement Method Based On LSTM Neural Network for Speech Recognition,” *2018 14th IEEE Int. Conf. Signal Process.*, no. October, pp. 245–249, 2019, doi: 10.1109/ICSP.2018.8652331.

[36] J. Cifuentes, P. Boulanger, M. T. Pham, F. Prieto, and R. Moreau, “Gesture Classification Using LSTM Recurrent Neural Networks,” *Proc. Annu. Int. Conf. IEEE Eng. Med. Biol. Soc. EMBS*, no. 2, pp. 6864–6867, 2019, doi: 10.1109/EMBC.2019.8857592.

[37] P. Malhotra, L. Vig, G. Shroff, and P. Agarwal, “Long Short Term Memory Networks for Anomaly Detection in Time Series,” *ESANN 2012 proceedings, 20th Eur. Symp. Artif. Neural Networks, Comput. Intell. Mach. Learn.*, no. April, pp. 1–650, 2012.

[38] N. L. Hakim, T. K. Shih, S. P. K. Arachchi, W. Aditya, Y. C. Chen, and C. Y. Lin, “Dynamic hand gesture recognition using 3DCNN and LSTM with FSM context-aware model,” *Sensors (Switzerland)*, vol. 19, no. 24, 2019, doi: 10.3390/s19245429.

[39] R. Siriak, I. Skarga-Bandurova, and Y. Boltov, “Deep convolutional network with long short-term memory layers for dynamic gesture recognition,” *Proc. 2019 10th IEEE Int. Conf. Intell. Data Acquis. Adv. Comput. Syst. Technol. Appl. IDAACS 2019*, vol. 1, pp. 158–162, 2019, doi: 10.1109/IDAACS.2019.8924381.

[40] H. Tran, “A Survey of Machine Learning and Data Mining Techniques used in Multimedia System,” no. 113, pp. 13–21, 2019, doi: 10.13140/RG.2.2.20395.49446/1.

[41] I. T. Jolliffe, “Principal components analysis,” *Int. Encycl. Educ.*, p. 518, 2002, doi: 10.1016/B978-0-08-044894-7.01358-0.

[42] M. Scholz, “Approaches to analyse and interpret biological profile data,” 2006.

[43] F. Song, Z. Guo, and D. Mei, “Feature selection using principal component analysis,” *Proc. - 2010 Int. Conf. Syst. Sci. Eng. Des. Manuf. Informatiz. ICSEM 2010*, vol. 1, pp. 27–30, 2010, doi: 10.1109/ICSEM.2010.14.

[44] W. Hastie, Trevor; Stuetzle, “Principal Curves,” *J. Am. Stat. Assoc.*, vol. 84, no. 406, pp. 502–516, 1989.

[45] M. Scholz, “Analysing periodic phenomena by circular PCA,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 4414 LNBI, no. 2005, pp. 38–47, 2007, doi: 10.1007/978-3-540-71233-6_4.

[46] U. Kruger, J. Zhang, and L. Xie, “Developments and Applications of Nonlinear Principal Component Analysis – a Review,” in *Principal Manifolds for Data Visualization and Dimension Reduction*, no. August 2015, 2007, pp. 1–43.

[47] B. Schölkopf, A. Smola, and K. R. Müller, “Nonlinear Component Analysis as a Kernel Eigenvalue Problem,” *Neural Comput.*, vol. 10, no. 5, pp. 1299–1319, Jul. 1998, doi: 10.1162/089976698300017467.

[48] S. Kumar, T. Srivastava, and R. S. Singh, "Hand Gesture Recognition Using Principal Component Analysis," no. January, 2017.

[49] E. Amid and M. K. Warmuth, "TriMap: Large-scale Dimensionality Reduction Using Triplets," *arXiv Prepr. arXiv1910.00204*, Oct. 2019.

[50] L. Van Der Maaten and G. Hinton, "Visualizing data using t-SNE," *J. Mach. Learn. Res.*, vol. 9, no. November 2008, pp. 2579–2625, 2008.

[51] L. Chen, "Support Vector Machine — Simply Explained," *Towards Data Science*, 2019. [Online]. Available: https://towardsdatascience.com/support-vector-machine-simply-explained-fee28eba5496.

[52] T. Briggs and T. Oates, "Discovering domain-specific composite kernels," *Proc. Natl. Conf. Artif. Intell.*, vol. 2, no. January, pp. 732–738, 2005.

[53] MathWorks, "Classify Videos Using Deep Learning," 2020. [Online]. Available: https://www.mathworks.com/help/deeplearning/ug/classify-videos-using-deep-learning.html.

[54] Shah Tarang, "About Train, Validation and Test Sets in Machine Learning," *Towards Data Science*, 2017. [Online]. Available: https://towardsdatascience.com/train-validation-and-test-sets-72cb40cba9e7.

[55] C. Szegedy *et al.*, "Going deeper with convolutions," *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 07-12-June, pp. 1–9, 2015, doi: 10.1109/CVPR.2015.7298594.

[56] MathWorks, "Pretrained Deep Neural Networks," 2020. [Online]. Available: https://www.mathworks.com/help/deeplearning/ug/pretrained-convolutional-neural-networks.html.

[57] F. Özgenel and A. Gönenç Sorguç, "Performance comparison of pretrained convolutional neural networks on crack detection in buildings," *ISARC 2018 - 35th Int. Symp. Autom. Robot. Constr. Int. AEC/FM Hackathon Futur. Build. Things*, no. Isarc, 2018, doi: 10.22260/isarc2018/0094.

[58] S. Siami-Namini, N. Tavakoli, and A. S. Namin, "The Performance of LSTM and BiLSTM in Forecasting Time Series," *Proc. - 2019 IEEE Int. Conf. Big Data, Big Data 2019*, pp. 3285–3292, 2019, doi: 10.1109/BigData47090.2019.9005997.

[59] A. T. Mohan and D. V. Gaitonde, "A Deep Learning based Approach to Reduced Order Modeling for Turbulent Flow Control using LSTM Neural Networks," no. April, 2018.

[60] J. Brownlee, "A Gentle Introduction to Mini-Batch Gradient Descent and How to Configure Batch Size," *Mach. Learn. Mastery*, pp. 1–17, 2017.

[61] "Java tools for Address-Event Representation (AER) neuromorphic processing." [Online]. Available: http://jaerproject.org.

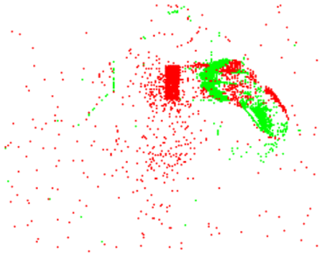
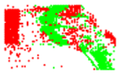
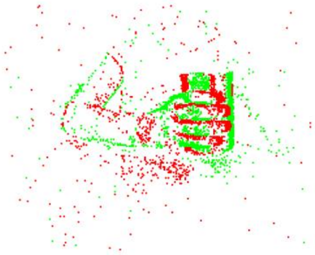
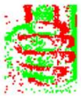
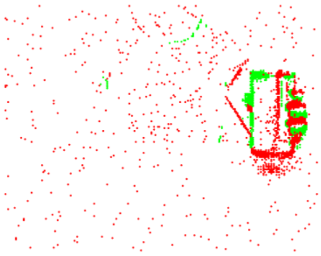
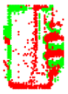
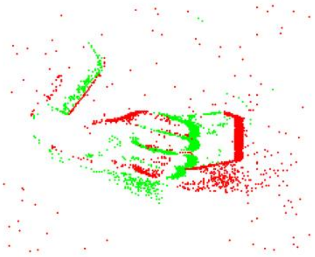
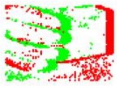
[62] "MATLAB Deep Learning Toolbox."

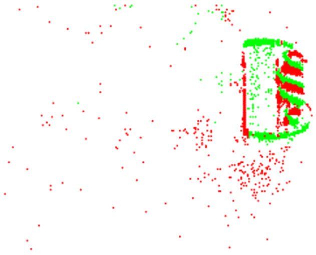
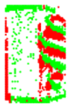
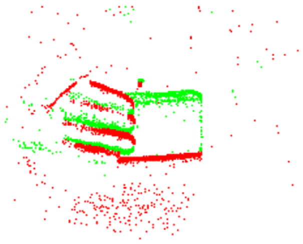
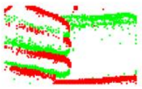
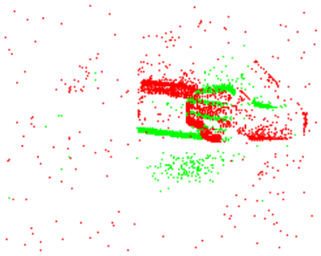
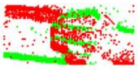
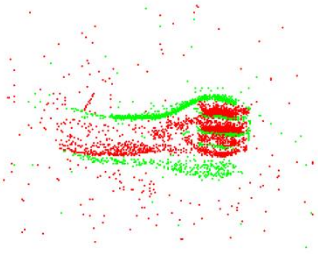
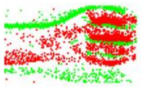
[63] "Deep Learning Toolbox Model for GoogLeNet Network." [Online]. Available: https://www.mathworks.com/matlabcentral/fileexchange/64456-deep-learning-toolbox-model-for-googlenet-network?s_tid=srchtitle.

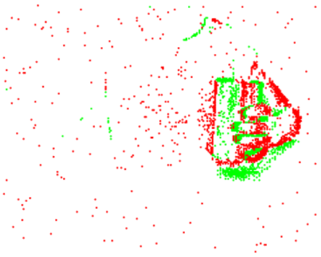
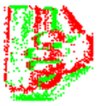
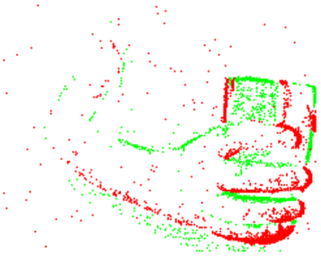
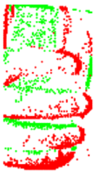
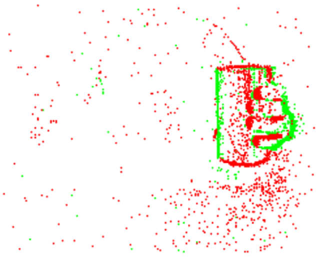
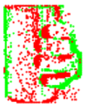
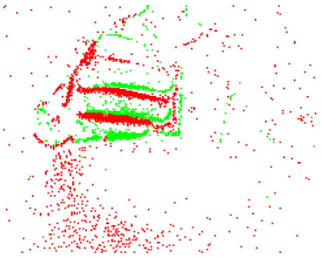
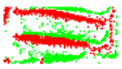
[64] "Deep Learning Toolbox Model for ResNet-50 Network." [Online]. Available: https://www.mathworks.com/matlabcentral/fileexchange/64626-deep-learning-toolbox-model-for-resnet-50-network?s_tid=FX_rc2_behav.

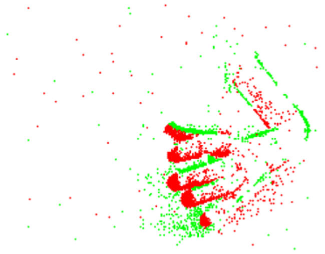
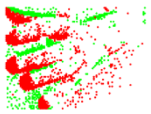
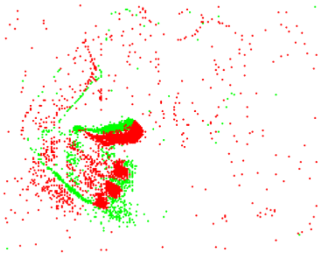
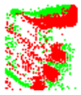
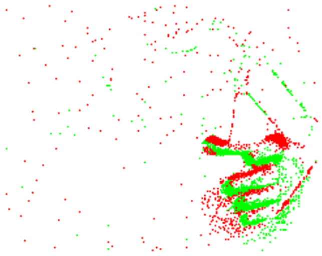
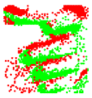
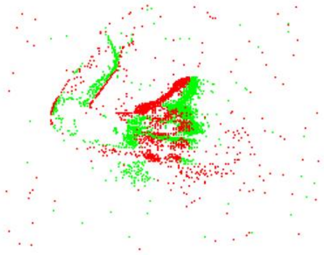
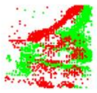
APPENDIX A – NR ALGORITHM RESULTS

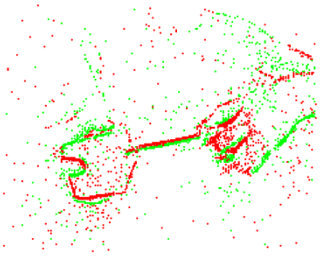
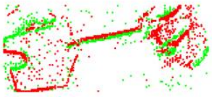
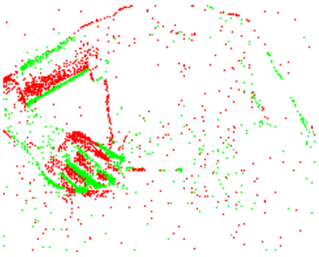
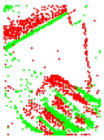
The results from the NR algorithm for each task of the dataset and the idle position are presented as following. The frames were obtained for 3000 Events/frame, using the average as the limit and a minimum number of events per zone of α=*50*. The top image of each task corresponds to the left hand and the bottom image to the right hand.

	Original Frame	Noise Reduced Frame
Task 1 *Move Left*		
		
Task 2 *Move Right*		
		

	Original Frame	Noise Reduced Frame
Task 3 *Lift*		
		
Task 4 *Place*		
		

	Original Frame	Noise Reduced Frame
Task 5 *Approach*		
		
Task 6 *Retreat*		
		

	Original Frame	Noise Reduced Frame
Task 7 *Grab*		
		
Task 8 *Release*		
		

	Original Frame	Noise Reduced Frame
Task 9 *Screw*		
		
Task 10 *Idle*