Trajectory Prediction with Event-Based Cameras for Robotics Applications

Author: Marco Monforte

Supervisors: Dr. Chiara Bartolozzi
Dr. Arren Glover
Dr. Ander Arriandiaga Laresgoiti

IIT - Istituto Italiano di Tecnologia

University of Genova

PhD Thesis for the curriculum of

Bioengineering and Robotics

May 16, 2021

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements. This dissertation contains fewer than 65,000 words including appendices, bibliography, footnotes, tables and equations and has fewer than 150 figures.

Marco Monforte

May 16, 2021

Abstract

This thesis presents the study, analysis, and implementation of a framework to perform trajectory prediction using an event-based camera for robotics applications.

Event-based perception represents a novel computation paradigm based on unconventional sensing technology that holds promise for data acquisition, transmission, and processing at very low latency and power consumption, crucial in the future of robotics. An event-based camera, in particular, is a sensor that responds to light changes in the scene, producing an asynchronous and sparse output over a wide illumination dynamic range. They only capture relevant spatio-temporal information - mostly driven by motion - at high rate, avoiding the inherent redundancy in static areas of the field of view. For such reasons, this device represents a potential key tool for robots that must function in highly dynamic and/or rapidly changing scenarios, or where the optimisation of the resources is fundamental, like robots with on-board systems.

Prediction skills are something humans rely on daily - even unconsciously - for instance when driving, playing sports, or collaborating with other people. In the same way, predicting the trajectory or the end-point of a moving target allows a robot to plan for appropriate actions and their timing in advance, interacting with it in many different manners. Moreover, prediction is also helpful for compensating

robot internal delays in the perception-action chain, due for instance to limited sensors and/or actuators.

The question I addressed in this work is whether event-based cameras are advantageous or not in trajectory prediction for robotics. In particular, if classical deep learning architecture used for this task can accommodate for event-based data, working asynchronously, and which benefit they can bring with respect to standard cameras. The a priori hypothesis is that being the sampling of the scene driven by motion, such a device would allow for more meaningful information acquisition, improving the prediction accuracy and processing data only when needed - without any information loss or redundant acquisition.

To test the hypothesis, experiments are mostly carried out using the neuromorphic iCub, a custom version of the iCub humanoid platform that mounts two event-based cameras in the eyeballs, along with standard RGB cameras. To further motivate the work on iCub, a preliminary step is the evaluation of the robot's internal delays, a value that should be compensated by the prediction to interact in real-time with the object perceived. The first part of this thesis sees the implementation of the event-based framework for prediction, to answer the question if Long Short-Term Memory neural networks, the architecture used in this work, can be combined with event-based cameras. The task considered is the handover Human-Robot Interaction, during which the trajectory of the object in the human's hand must be inferred. Results show that the proposed pipeline can predict both spatial and temporal coordinates of the incoming trajectory with higher accuracy than model-based regression methods. Moreover, fast

recovery from failure cases and adaptive prediction horizon behavior are exhibited. Successively, I questioned how much the event-based sampling approach can be convenient with respect to the classical fixed-rate approach. The test case used is the trajectory prediction of a bouncing ball, implemented with the pipeline previously introduced. A comparison between the two sampling methods is analysed in terms of error for different working rates, showing how the spatial sampling of the event-based approach allows to achieve lower error and also to adapt the computational load dynamically, depending on the motion in the scene. Results from both works prove that the merging of event-based data and Long Short-Term Memory networks looks promising for spatio-temporal features prediction in highly dynamic tasks, and paves the way to further studies about the temporal aspect and to a wide range of applications, not only robotics-related.

Ongoing work is now focusing on the robot control side, finding the best way to exploit the spatio-temporal information provided by the predictor and defining the optimal robot behavior. Future work will see the shift of the full pipeline - prediction and robot control - to a spiking implementation. First steps in this direction have been already made thanks to a collaboration with a group from the University of Zurich, with which I propose a closed-loop motor controller implemented on a mixed-signal analog/digital neuromorphic processor, emulating a classical PID controller by means of spiking neural networks.

Contents

List of Figures

Chapter 1

Introduction

Robots are increasing in complexity and skills day-by-day and, with them, the range of tasks they are demanded to solve is widening. In unconstrained environments, the necessity of smooth interactions and adaptation to dynamic scenarios is highlighting the importance of low-latency responses. In addition, faster computation capabilities, long-life batteries, and bigger memory banks needs are bringing to light the importance of resource optimisation. The research community is therefore being pushed not only towards the development of more and more sophisticated algorithms, but also to pay attention to aspects like latencies and power consumption.

With the exponential growth of machine learning - and in particular Deep Learning (DL) - applications, the attention of roboticists and computer vision scientists rapidly shifted from classical geometry-based approaches to learning methods, establishing new state-of-the-art levels [1] at the cost of large amounts of data and long and computationally expensive training. Along with the growing interest, new devices are being reinvented or designed anew for enhancing the training process, such as Graphics Processing Unit (GPU)s and Tensor Processing Units (TPUs), but none of them addresses the bottlenecks coming from the source

of the data itself. Traditional vision sensors, indeed, while having a very high spatial resolution, present some important drawbacks when dealing with fast dynamics. Mainly:

- The frame-rate is fixed and it is a property of the device, not depending on the actual scene. This, on one hand, limits the dynamics that can be captured, as fast movements might not be captured or lead to image artifacts like blur, and, on the other hand, the amount of data to store, transmit, and process grows enormously with the rate;
- Such amount of data becomes even highly redundant when the scene is changing little by little or not at all. The device pixels array, indeed, is thoroughly scanned anyway at the same rate, leading to a waste of power and computational resources;
- Finally, issues arise in environments with different illumination conditions, given the fact that the same exposure time is applied to every pixel. Under- or over-exposure can be therefore caused by such a limited dynamic range.

From this perspective, event-based cameras [2–4] hold great potential for filling these gaps. Their working principle is based on the amount of light falling on each pixel, measuring whether the intensity has increased or decreased more than a certain threshold and, in such case, outputting a local spike - named *event*. In this way, the information is compressed and sent out in an asynchronous fashion, with fine temporal resolution, latency in the order of microseconds, and low power consumption. On the other hand, neuromorphic devices still represent relatively new technology. The majority of the work developed for event-based cameras has taken in large part inspiration from their RGB counterpart, and only recently researchers started giving the same priority to purely event-based

algorithms. That is why a lot of effort and resources are being put in defining the range of action of this paradigm, trying to understand both the advantages and the limitations of this class of devices in several kinds of tasks. The combination of event-based cameras and Deep Learning techniques is also being studied by the neuromorphic community. However, given the deeply different nature of the data transmitted by this camera with respect to a standard frame-based camera, multiple approaches are actually being studied in parallel on how to best use their output: from collecting events to create a sort of *event-image*, to working with single, asynchronous, bits of information. These differences must be considered when working with such devices, as they also affect how the pipeline and the learning algorithm would work.

Figure 1.1: The ATIS Gen. 3 event-based camera.

1.1 Objective of the thesis

This thesis explores the use of event-based cameras for trajectory prediction of dynamic objects in robotics applications, assessing the advantages of event-based sensing in terms of precision and computational efficiency.

Prediction skills are innate in the human being. We daily rely on them when avoiding people or cars in our way, catching falling objects, shaking hands, or even in collaborative tasks like lifting a heavy box and playing sports. Similarly, they are crucial for autonomous systems that have to interact in unconstrained environments, safely navigating, and rapidly adapting to changes in their surroundings. Predicting the incoming trajectory or the end-point of a tracked object allows robots to plan and execute proper actions in advance, determining also their timing and the way to interact with the object. Moreover, prediction can be helpful for compensating delays in the perception-action chain of the robot. The task of motion prediction is widely studied in robotics using standard RGB cameras. The expansion of the application fields of robotics - like the growing interest in drones and self-driving cars - has also increased the importance of the safety and reliability of such systems. Up to the last decade, the answer to tasks requiring high performance was to use very powerful but costly devices, leaving issues such as the battery duration, memory occupation, and computational load previously mentioned to a second time. Since the rise of event-based cameras, however, another option became available, and researchers started combining these sensors with Deep Learning methods, probably the hottest topic in computer science and robotics of the past ten years, to address a wide variety of tasks.

In general, before predicting its trajectory, the object of interest must be located and segmented, in order to extract the relevant features. In standard frame-based applications, this process can be performed using classical methods

on individual frames, knowing that a new frame arrives after a well specified interval, constant in time. Given the different nature of the event-based camera output, however, designing an event-based pipeline requires some tweaks.

The main contribution of this work is the development of an event-based pipeline for trajectory prediction. The object moving in the scene generates events that are tracked with an event-based algorithm, and fed to an Encoder-Decoder Long Short-Term Memory (LSTM) network. The latter takes both spatial and temporal coordinates in input, with the goal of predicting the incoming points of the trajectory. Evaluation of the proposed method is done looking at the Root Mean Squared Error (RMSE) of the prediction, and comparing it with other model-based regression methods. Subsequently, the convenience of the spatial sampling performed by event-based cameras is further analysed with respect to standard fixed-rate sampling for different rate values. The robot considered for the experiments is the neuromorphic iCub [5], which presents two Asynchronous Time-based Image Sensor (ATIS) [4] event-based cameras in the eyeballs, along with a couple of regular frame-based cameras.

An overview about the event-based cameras working principle is now presented (Sec. 1.2), before discussing more in detail about LSTM networks (Sec. 1.3) and where trajectory prediction is nowadays applied and how it is addressed (Sec. 1.4). To conclude this section, an evaluation of the iCub internal delays (Sec. 1.5) is reported, to further motivate the application of the work presented in this document.

The following chapters explore the proposed pipeline and analyses the results of its test against regression methods (Chap. 2), followed by a deeper discussion about the advantages of a motion-driven sampling with respect to the classical

time-driven approach (Chap. 3). Ongoing work (Chap. 4) is discussed, highlighting changes and limitations when moving to the robot application. Conclusions and future work are finally exposed in the last chapter (Chap. 5).

1.2 Event-based cameras

Rather than with a fixed time-driven sampling (Fig. 1.2a), it appears from physiological studies [6] that biological systems sense in a substantially different way, encoding information in an asynchronous, data-driven fashion, under the form of sparse spike trains. Neural computation relies on large amounts of simple units - the *spikes*, indeed - which encode relative changes of a certain quantity, rather than absolute values (Fig. 1.2b).

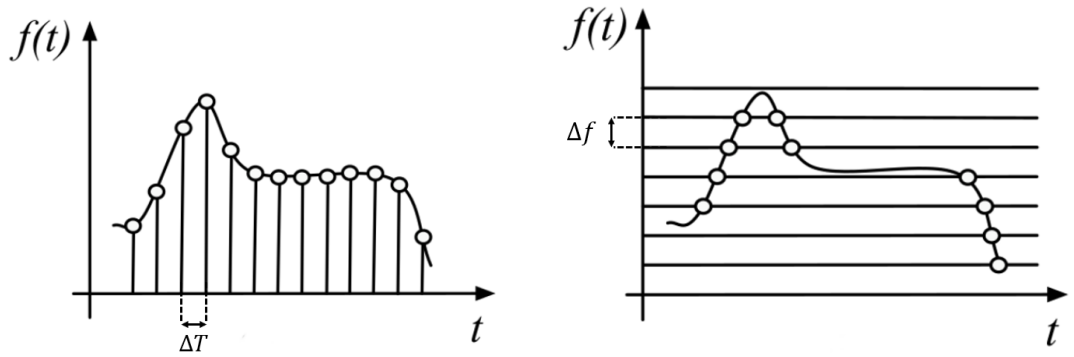


Figure 1.2: Two different ways of sampling and encoding information: a time-driven approach (a), sampling at fixed rate regardless of the trend in $f(t)$, and a data-driven one (b), that samples asynchronously only if and when there is an absolute change higher than a certain threshold in $f(t)$.

Such biological computational principles inspired new generations of researchers to start what is the nowadays called *neuromorphic engineering*, a research branch devoted at studying and replicating neural principles to create artificial bio-inspired sensors [7] and computational mechanisms [8].

Event-based cameras, of which one model can be seen in Fig. 1.1, probably represent the most mature technology of this field. Compared to traditional frame-based systems, they embody a significant revolution in how the scene is perceived and how it is represented at pixel level. Each pixel is sensitive to brightness changes at logarithmic scale, encoding temporal changes according to Eq. 1.1:

$$\frac{d}{dt}logI = \frac{dI/dt}{I} \tag{1.1}$$

Whenever a brightness change higher than a threshold - usually set to be the 15% of the contrast - is perceived, an *independent* event is emitted by the pixel, along with the associated time instant and a binary value, named *polarity*, indicating whether the change is positive or negative. According to the Address Event Representation (AER) protocol [9], an event is therefore mathematically defined by the following quadruplet:

$$\boldsymbol{e} = [x, y, p, t]^T \tag{1.2}$$

with the pair (x, y) defining the spatial coordinates in the pixel array, p representing the polarity with value 1 for a positive change and -1 for a negative change, and t the timestamp.

Being driven only by the illumination change, the event generation does not rely in any way from a device-dependent timing signal, but only from the visual scene. The unique limit to the temporal resolution is, hence, the hardware and its rate for reading incoming events, usually in the order of μs. On one hand, this allows for very fast sensing; on the other hand, if no change is perceived, no information is acquired, avoiding a waste of resources for subsequent transmission and processing. The computation of a self-normalised temporal derivative is enabled by the logarithmic photo-transduction. In such a way, the sensor behavior

is invariant to absolute values and can operate at a wide variety of illumination conditions. Moreover, thanks to the independence of the pixels, conventional image sensors's intrascene dynamic range is exceeded by several orders of magnitude.

Fig. 1.3 shows the different output of a standard camera against an event-based camera in response to a rotating dot. In the case of typical motion, frames capture different spatial positions of the dot with "jumps" in between, due to the time needed to fully scan the pixel array and send out the image, while the event-based camera produces a continuous stream of events in the (x,y,t) space. When the motion stops, the events stream stops as no change is detected anymore, while frame are still outputted at synchronous rate, leading to redundant acquisition. Finally, when the motion is very fast, frames acquisition suffers artifacts like image blur due to exposition time. This does not happen with event-based cameras as pixels are independent and very few computation is needed, allowing high rate acquisition and producing a dense stream in the (x,y,t) space.

To summarise, the main advantages of an event-based cameras are:

- The asynchronous emission of events at pixel level with a temporal resolution in the order of microseconds. External, device-dependent clocks are nonexistent. Single events can be gathered to create frame-like images, with higher rate than standard RGB devices (usually around 33 ms);

- Events are emitted only when brightness changes happen, no acquisition is made if there is no change. This makes the device free of redundant acquisitions;

- Being the exposure time adjusted independently from each pixel, a dynamic range in the order of 120-140 dB can be reached, in comparison to the standard 60-80 dB of RGB cameras;

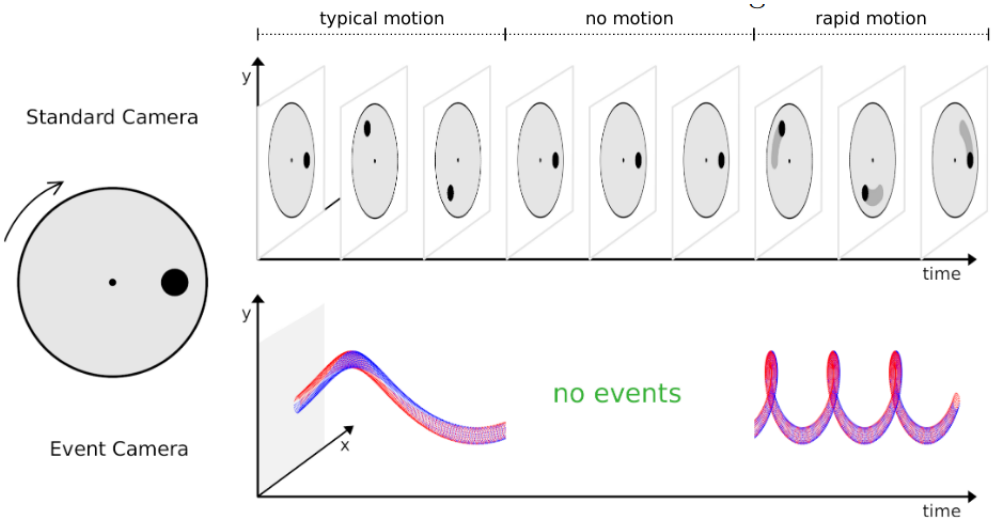


Figure 1.3: Considering a rotating dot, typical movements in front of standard cameras produce frames at a fixed rate depending from the device, while event-based cameras output a continuous stream of events with much higher temporal resolution. In case of no motion, the standard camera still outputs images, but without any meaningful information contained. The event-based camera, instead, does not output anything, since no motion - hence no light change - is happening in the scene. Finally, in case of rapid movements, standard cameras suffer image blur due to the acquisition time of the device. Event-based cameras, on the other hand, present independent pixels that are able to capture asynchronously the light changes and therefore output a denser stream of events, without loss of information. Image courtesy of [10]

- Low-power consumption, mainly because of the absence of A/D converters, typically used for pixel readout.

The first silicon retina was proposed in [11,12], exploiting the Very Large Scale Integration (VLSI) technology proposed in the late 1980s for the implementation of bio-mimetic and neuromorphic primitives (neurons, synapses, etc). Taking inspiration from the biological counterpart, it implemented the first three layers out of the five of the retina, including photoreceptors, horizontal and bipolar cells. Following this direction, later attempts are represented by [2–4, 13–16].

A remarkable leap in the technology was made by [2] with the implementation of the Dynamic Vision Sensor (DVS), a 128×128 pixel resolution silicon retina developed at the IniLabs of the Institute of Neuroinformatics in Zurich. Not only the device came with an easy-to-use library that made this technology available also for non-expert users, but it also drastically increased the variance in the pixel firing rates with respect to previous devices. Successive improvements led to the DAVIS240 camera, a combination of a DVS camera with a spatial resolution of 240×180 and a standard camera providing gray-scale intensity images overlapped to the events visual field [3, 17], and the most recent and CES Innovation Award for the 2020 Best Innovation, the DAVIS346, with a spatial resolution of 346×260. All these device are nowadays being manufactured and sold at the Inivation company [18], spin-off of IniLabs.

Produced and sold by Prophesee [19], the event-based camera model used in this work is the ATIS sensor [4]. It has a spatial resolution of 304×240 pixels and it allows to reconstruct gray-scale intensity images thanks to an asynchronous change detector - like in the DVS - and an exposure measurement circuit, both embedded in each pixel.

The main direction of improvement, anyway, seems to be the increase of the spatial resolution, like the Celex company [20] is trying to do with its new outstanding 1280×800 resolution event-based camera, unlocking new possibilities for algorithm deployment.

Noteworthy is also the commercial aspect of this new kind of technology. Given all the advantages previously mentioned - and in particular the high temporal resolution - comes natural to imagine such devices to be used in tasks where a fast response is crucial, such as surveillance, autonomous driving or any kind of high speed navigation. DVS-based products are being developed, for instance,

by Insightness [21] and Samsung [22]. The latter has released the first commercial product to the public, the SmartThings Vision [23]: an indoor event-based home monitoring system combined with Artificial Intelligence (AI) algorithms to detect from thieves to elder people falling. In general, also from a commercial and industrial point of view, this technology is attracting interest towards the development of new generation of vision tools.

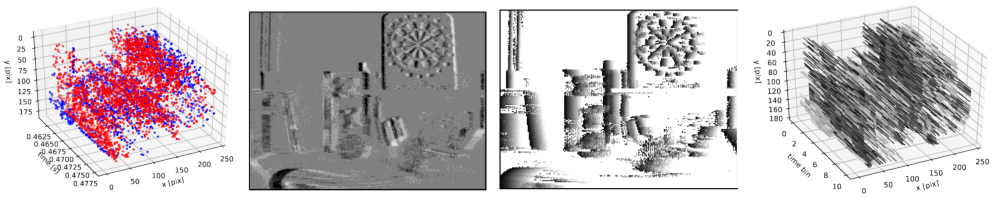
1.2.1 Events data representations

Given the strong difference in the output with respect to standard frame-based systems, the spatio-temporal events flow (Fig. 1.4a) must be adapted in a proper way to be used efficiently. Two main classes of event-based data representations can be identified, depending on how the spikes are treated: *event-by-event* methods and *groups of events* methods. Principal exponents of event-by-event applications are deterministic and probabilistic filters, thanks to their natural capability of handling asynchronous data and acting as local aggregators of information. This ensures the preservation of the sensor's properties like low latencies and low computational cost. Groups of events, instead, are processed together to yield a sufficient signal-to-noise ratio for the task at hand when the information carried by single events is little and subject to noise. In general, the choice of the representation mainly depends on the task at hand - which in general determines the type of algorithm to be used - and the hardware available. Given the novelty of these sensors and the fact that we are at the early stages of research for the field, it is still natural to consider principally dense representations for algorithms - like Deep Learning ones - running on Central Processing Units (CPUs) and GPUs; while event-by-event processing suits Spiking Neural Network (SNN) and dedicated hardware - such as the SpiNNaker [24], Loihi [25], or DYNAP [26]

platforms - in order to implement more efficient solutions.

The definition "group of events" is actually general. Different outputs can be obtained depending on how the spatial and temporal information are encoded, or possibly discarded. Common representations are:

- *Event images* [27, 28],where simple 2D images are created by collecting a certain number of events or grouping by temporal bins and projecting all of them on the 2D plane (Fig. 1.4b);
- *Time surfaces* (TSs) [29–31], a 2D map whose pixels's intensity depends on the time stamp of the corresponding most recent event. Pixels associated to the most recent events have higher intensity (Fig. 1.4c);
- Voxel grids [32, 33], a 3D quantisation of the original events flow. Each voxel represents the state of the corresponding pixel in a particular time bin (Fig. 1.4d);



(a) (b) (c) (d)

Figure 1.4: Different representations of groups of events: a classical visualisation of events in the 3D space (2D spatial + temporal dimensions) (a); an event-image obtained after collecting events for a certain time interval/in a certain number and then projecting all of them on the 2D image plane (b); a Time Surface image, whose pixels's intensity depends on when the latest associated events occurred (c); and a $(240 \times 180 \times 10)$ *voxel grid, a 3D representation of the 2D pixels matrix at different temporal instants. Each voxel contains the state of the relative pixel in a particular time interval. Images courtesy of [34].*

1.2.2 Event-based cameras applications in robotics

A field where event-based cameras hold great potential is computer vision for robotics, in particular on those tasks with fast dynamics. For this reason, they are being exploited on several classical computer vision problems.

An event-based tracking pipeline is implemented on FPGA, with filtering operations, in [35]. A cluster-based mechanisms initiates a tracker and adapts its behavior to the number of events in the scene and their speed. A maximum of 4 trackers can be run in parallel, thanks to the computational capacity of the FPGA. The work evolved then in [36], where it is included in a FPGA-based library for event-based post-processing, along with background activity filtering, pixel masking, object motion detection and object tracking. Feature detection and tracking in real time is addressed in [37], combining both spatial and temporal correlations of events in an asynchronous iterative framework. Gaussian, Gabor, combinations of Gabor functions, and arbitrary user-defined kernels are used to track features from incoming events, handling variations in position, scale, and orientation through the use of multiple pools of trackers. A particle filter-based tracker is introduced in [38] with the aim of making the tracker robust to temporal variation occurring when the camera and the target move with different relative velocities, like the case of a robot actively following the target with its gaze. Object detection and tracking is also performed in [39] using information about the temporal dynamic component of the event stream. A parametric model for motion-compensate approximates the 3D geometry of the events stream. Then, moving objects that do not conform to the model are detected in an iterative process, using these inconsistencies to locate multiple objects.

A unifying framework to estimate motion, depth and optical flow is proposed in [40] first and [41] then. The main idea is to find the point trajectories on the image plane that are best aligned with the event data by maximising the contrast

of an image of warped events. The method automatically handles events data association and, to accurately recover the motion parameters of the problem, motion-corrected edge-like images with high dynamic range are generated and available for possible further scene analysis. Optical flow, depth and ego-motion are also estimated through unsupervised learning with events in [32]. Considering as input representation a discretised volume that maintains the temporal distribution of the events, a neural network is used to predict the motion in order to attempt to remove any image blur. Then, two networks are trained: one to predict the optical flow, and one for the ego-motion and depth. [42] also proposes to estimate optical flow using Spiking Neural Networks with event-based data. In particular, a hierarchical spiking architecture in which motion selectivity emerges in an unsupervised fashion from the raw events is introduced. A novel adaptive neuron model and stable spike-timing-dependent plasticity formulation are proposed at the core of this neural network, governing its spike-based processing and learning.

3D reconstruction, the task of recovering 3D structures from multiple images of a scene, is considered in [43], formulating the stereo vision problem solely in the time domain as a problem of events coincidences detection. The stereo matching problem is also addressed in [44] using semi-global matching, working with single or local groups of events. Finally, in [45], the stereo vision problem is solved using a mirror-galvanometer driven laser. The laser beam is deflected by actuating two mirrors, thus creating a sequence of 'light spots' in the scene, where contrast changes quickly. Detection of the events generated by such a laser is possible thanks to event-based cameras, facilitating the matching of the pair.

Constantly knowing the current camera pose of the robot can be crucial. In [40] a contrast maximisation problem is set to constantly estimate the rotational motion of an event-based camera, without requiring optical flow or image

intensity estimation. The problem is further analysed in [46], starting from the hypothesis of having available a photometric depth map (i.e., intensity plus depth information), built via classic dense reconstruction pipelines. The method tracks the 6-DOF pose every time a new event arrives using a Bayesian filtering approach. A panoramic setting with three degrees of freedom is studied in [47] to track the camera position. The tracking problem is formulated as an optimisation one, presenting it as a mapping method that accurately links each position to the probability of events being generated there, independently from the direction of the current camera movement.

The Simultaneous Localisation And Mapping (SLAM) problem is addressed in [48], tightly coupling events, standard frames, and inertial measurements. Experiments show the first autonomous quadrotor flight using an event-based camera for state estimation, unlocking new possible scenarios for this kind of robot. In [49] a continuous-time representation is leveraged to perform visual-inertial odometry. Using splines, the event-based camera trajectory is approximated in the space of rigid-body motions, drastically reducing the number of variables in the problem.

Motion compensation is also another interesting application to event-based data. The authors of [50] address the problem proposing a per-event segmentation method for splitting the scene into independently moving objects and by jointly estimate the event-object associations (i.e., segmentation) and the motion parameters of the objects (or the background). This is done by maximising the events contrast for each object, providing also an image that can be used for further processing.

Pattern recognition is studied in [51] using time-surfaces. A hierarchical event-based architecture is proposed with focus on a time-oriented approach, extracting spatio-temporal features from the events stream. The central concept is to use

the rich temporal information provided by events to create contexts in the form of time-surfaces, which represent the recent temporal activity within a local spatial neighborhood.

Importantly, not only vision, but also neuromorphic control is being investigated. In [52] a computationally efficient method for feedback control that takes advantage of the asynchronous, event-based nature of event-based cameras is presented. Through experimental tests with a mobile robot, it is showed that neuromorphic vision sensors can provide good closed loop performance in terms of computation, data rate, frequency, latency, and tracking error. Also [53] considers the problem of designing a regulator for the heading of a vehicle based on the feedback from an on-board neuromorphic sensor. The authors show that a non-linear function of the events retinal positions, followed by retinal integration and a linear filter, is a simple design sufficient to guarantee stability. Computationally simple controllers are sufficient to control motion tasks - even with the feedback from noisy and ambiguous event data - and without having to compute explicit representations for the state. Finally, the problem of quadratically stabilising a continuous-time Linear Time Invariant (LTI) system using measurements from a neuromorphic sensor is presented in [54]. A H-infinity controller is developed and used to stabilise the LTI system.

1.2.3 Event-based vision meets Deep Learning

In the last decade, thanks also to the development of supporting hardware and computational resources, Deep Learning came back in vogue and attracted the attention of the community of many research fields. The neuromorphic computing community as well followed the trend and started exploring neural networks as a way to extract information from the sparse output of event-based cameras.

In [29] proposed EV-FlowNet as a self-supervised Deep Learning pipeline for

optical flow estimation with event based cameras. The input data of the neural network is an image-based representation of a given event stream, using the corresponding gray-scale image captured from the same camera at the same time as supervisory signal for training. Results show that the network is able to accurately predict optical flow in a variety of different scenes and with performance comparable to the frame-based architecture. The same authors, as mentioned in the previous section, in [32] move to an unsupervised learning setting to estimate optical flow, depth and ego-motion. Another lightweight, unsupervised pipeline for dense depth, optical flow and ego-motion estimation is introduced in [31]. At the core of this pipeline there is a novel Encoder-Decoder neural network architecture - named Evenly-Cascaded convolutional Network (ECN) - trained in self-supervised mode with very few parameters. Thanks to the lightweight design, the inference part of the network runs at 250 frames-per-second on a single GPU, making the pipeline ideal for real-time robotics applications.

Image/video reconstruction using event-based cameras are discussed in [33, 55]. A Recurrent Neural Network (RNN) is trained on artificial data to learn to reconstruct videos from a stream of events. Experiments show an improvement of the state-of-the-art of more than the 20%. Moreover, off-the-shelf computer vision algorithms are applied to the reconstructed videos for object classification and visual-inertial odometry, showing how this strategy consistently outperforms algorithms specifically designed for events. A similar work is done in [56], where a conditional Generative Adversarial Network (GAN) is used to create images/videos from an adjustable portion of the events stream. The usefulness of event-based cameras to generate high dynamic range, non-blurred images under rapid motion is discussed, along with the possibility of generating very high frame-rate videos (theoretically up to 1 million frames-per-second). The proposed method is compared to intensity images captured on the same pixel grid-line of

events using online available real datasets and synthetic datasets produced by an event-based camera simulator.

In [57] a predator/prey scenario is set up for the use of a Convolutional Neural Network (CNN), trained and run on both conventional images and event-images. The network rate is therefore proportional to the scene activity, varying from 15 Hz to 240 Hz depending on the robot speeds. Four possible outputs can be generated: steer right, steer left, center and non-visible. After offline training on labeled data, the network is imported on the robot, which can be controlled in real-time by the network. A similar use of an off-the-shelf deep-learning architecture is done in [58], where a CNN originally created for RGB images is used to detect objects in event images, analysing the impact of different temporal integration windows of the event data. In [59], instead, the authors implement a live demonstration of a CNN playing RoShamBo (rock-paper-scissors) against a human opponent. The pipeline presents real-time performance with an accuracy of 99.3% for the test set.

A regression task combining event-based cameras and CNNs is proposed in [27] for the steering angle prediction of a self-driving car. A state-of-the-art CNN is trained on event images obtained from a large scale dataset, the DDD17 [60], showing that event-based cameras allow robust steering prediction even in cases where traditional cameras fail (e.g. challenging illumination conditions and fast motion). Moreover, along with [58], are again demonstrated the advantages of combining classical Deep Learning for frame-based devices with event-based cameras.

1.2.3.1 Spiking Neural Networks

Given the nature of the event-based camera output, the the SNN framework [61] seems to be better suited than traditional Deep Learning algorithms. In these

networks, the communication neuron-to-neuron is carried out in a brain-like fashion under the form of spikes. Treating each pixel of the field-of-view (FOV) like a neuron that emits an event every time it spikes makes possible to easily combine these two technologies. With such a sparse data transmission, power consumption and latencies are contained [62]. The biggest reason for which SNNs did not achieve comparable success to classical neural networks is the lack of a well established way to train them. Spikes, indeed, are not differentiable by definition and, therefore, standard backpropagation cannot be applied. Nowadays, three main approaches allow to train SNNs: (i) mapping the weights of a pre-trained standard network to a spiking one, (ii) use of variant of backpropagation directly on the SNN, approximating the behavior of the standard differentiation process to spikes, and (iii) unsupervised learning techniques, which do not require the gradient computation at all.

The first approach is used, for instance, in [63]. Here the authors present a set of optimisation techniques to minimise performance loss in the conversion process for Convolutional Neural Networks and Fully Connected (FC) networks, outperforming all previous SNNs on the MNIST database. The method proposed for conversion enables low-latency classification with high accuracy already after the first output spikes. Further experiments with this training method are carried out in [64], where it is demonstrated that neuromorphic computing can implement DL architectures reaching state-of-the-art accuracy across eight standard datasets between vision and speech. These networks, however, did not include operations like max-pooling, softmax and batch-normalization. Spiking equivalents of them are proposed in [65]. Famous classical architectures like VGG-16 and Inception-v3 are in this work converted into SNNs, producing the best results for the time on famous available datasets.

Efforts are also going in the direction of direct supervised learning, trying to mimic the backpropagation behavior on spikes. In [66] the membrane potentials of spiking neurons are treated as differentiable signals, while discontinuities at spike times are considered as noise. In such a way, a backpropagation mechanism similar to the conventional one is permitted directly on spike signals and membrane potentials. With respect to previous methods transferring weights from a standard Artificial Neural Network (ANN), it is now possible to capture more precisely the statistics of spikes. Evaluating the method on MNIST,the error rate is reduced by a factor of more than three compared to the best previous SNN, achieving a higher accuracy also compared to conventional CNNs trained and tested on the same data. On the N-MNIST dataset, moreover, equivalent accuracy is achieved with about five times fewer computational operations. Authors of [67] introduce a new backpropagation mechanism using a temporal credit assignment policy for backpropagating error to preceding layers. The method allows to train both Fully Connected networks and CNN architectures.

To train SNNs in an unsupervised manner it is possible to use Spike-Timing Dependent Plasticity (STDP) [68], a learning rule that uses Long-Term Potentiation (LTP) to strengthen the connections between neurons that show a causal relationship in their firing, and Long-Term Depression (LTD) to weaken anti-causal relationships. Not needing labelled data, this approach is useful especially for practical applications or where annotated data are scarcely available. In [69] a method to train all layers of the network simultaneously in an unsupervised fashion is proposed. This allows approximate online inference already at learning stage, making the architecture suitable for online learning. Moreover, it is possible to train the network without providing implicit information about the

database, like the number of classes and the duration of input stimuli.

To the best of my knowledge, there is no work combining event-based cameras with Encoder-Decoder LSTM networks, especially evaluating also the different sampling approach. Even discussing their application to event-based data in [70], Phased-LSTM were evaluated in a preliminary test, but no benefit with respect to standard LSTM was found. Given the lack of a well established implementation of LSTM networks in a spiking fashion, in this thesis I focused on the use of the standard Deep Learning implementation.

1.3 Long Short-Term Memory (LSTM) Networks

One of the major drawbacks of standard neural networks like Fully Connected or Convolutional Neural Networks is that, in their basic implementation, there is no way to transmit information from one query to the other. If we want to classify an action or an object in the scene, there is no way to transmit the information outputted from the network in the previous instants to the current calculation.

This goes against what living beings actually do. When reading, watching a movie, following something with the gaze or just thinking, we do not start from scratch every time, looking all over the page or the screen to go back where our attention was, but we present a persistence of the information in time that allows us to keep memory of the short-term past.

This is also what a particular class of Deep Learning methods, named Recurrent Neural Networks, does [71]. As from their name, such networks present loops in their artificial neurons that allow to keep memory of previous network queries and create a temporal persistence similar to the one previously mentioned. A simple representation of such a neuron - but that can easily generalise also a full

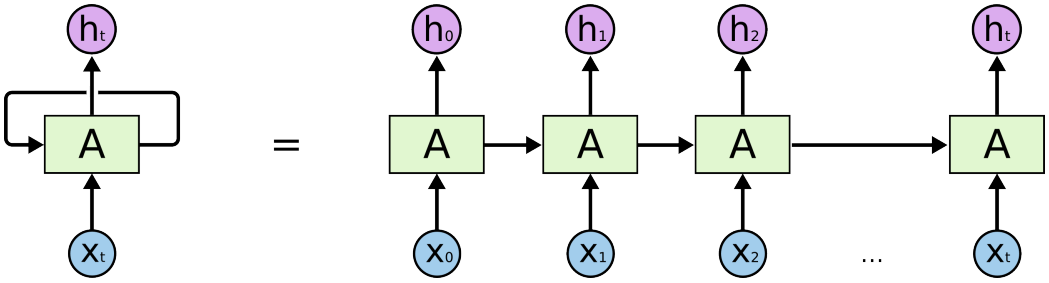
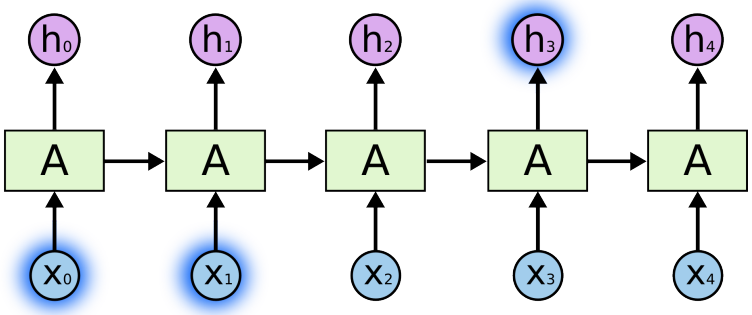


Figure 1.5: A recurrent neuron unrolled in time. Image courtesy of [72]

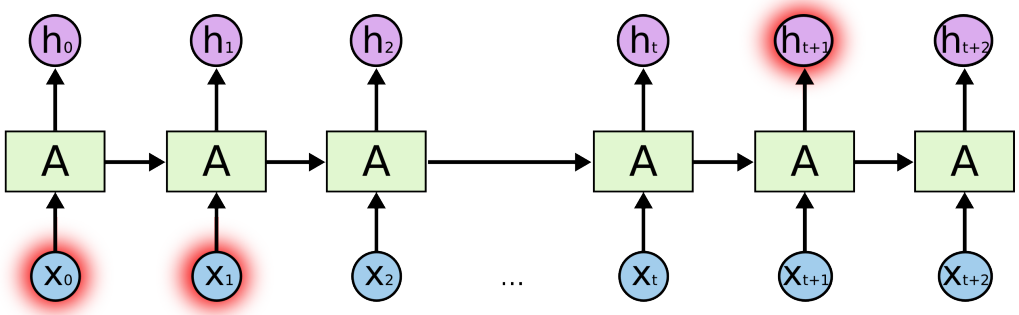
network, in a more abstract way - is the one in Fig. 1.5. The neuron A presents a *recurrent connection* that stores the information at time t to use it again at the next time step $t+1$. The same can be seen in its unrolled version, where the horizontal axis is the time and each vertical triplet (x_i, A, h_i) represents respectively the current input, the neuron A containing information about the network at time t_i, and the current state of the hidden layer (corresponding to the output if the layer is the last one). Each triplet receives and transmits the information through the horizontal black arrows.

Long Short-Term Memory networks, a particular class of Recurrent Neural Network, were introduced in [73] as a solution to the *vanishing gradient* problem affecting vanilla RNNs. This problem, discussed deeply in [74], indicates the difficulty in learning long-term dependencies, that is, the correlation between the current input/output and the information elaborated many time steps before (Fig. 1.6). Mathematically speaking, it shows up as a trend toward very small weights update values of the network layers in the early time steps, making impossible to adjust the network behavior with the Gradient Descent (GD) algorithm. More details about the LSTM training process, the Backpropagation Through Time (BPTT), will be provided later.

The power of the LSTM lies in its inner structure. Vanilla RNNs present a



(a) Short-term dependencies.



(b) Long-term dependencies

Figure 1.6: The vanishing gradient problem arises during the training of a network as the impossibility of linking the information being elaborated at the current time step with the one processed many steps before. Image courtesy of [72]

simple structure with only a single *tanh gate* to manage the information coming from previous time steps, represented by Eq. 1.3:

$$\mathbf{h}_t = tanh(\mathbf{W}\mathbf{h}_{t-1} + \mathbf{U}\mathbf{x}_t + \mathbf{b}) \tag{1.3}$$

with the vector $\mathbf{b}$ as the bias, and $\mathbf{W}$ and $\mathbf{U}$ as the weight matrices, respectively for the previous hidden state of the network and the current input.

The core idea behind LSTMs, instead, is the introduction of a *cell state* - a sort of memory of the network - carried through the different time steps by a connection named *Constant Error Carousel (CEC)*, and a *three-gated mechanism*, each of them a way to optionally let the information pass through the neuron

(Fig 1.7). Interacting in a very simple and linear way with the cell state, the gates manage what part can be forgotten or kept from the past, what part can be stored of the current input, and what is actually needed to be outputted.

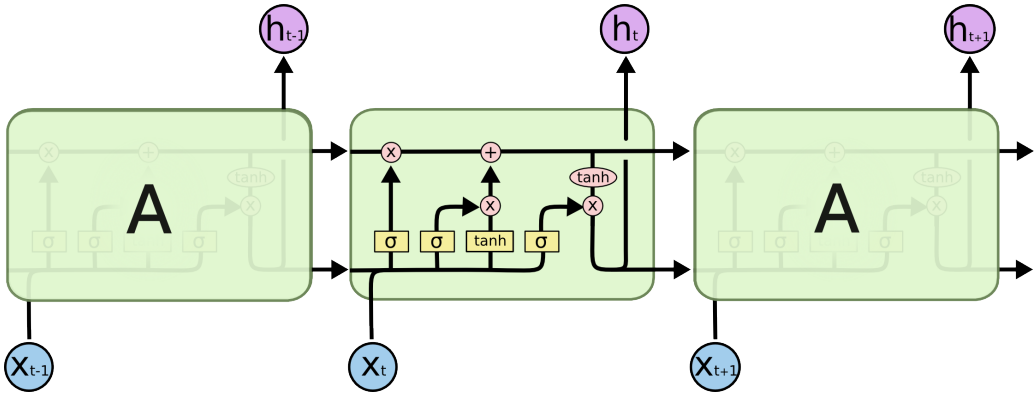


Figure 1.7: The LSTM node presents a complex inner structure, made of a memory vector - the cell state - and a three-gates mechanism that manages what to remember from the past iterations, what to pass from the current input, and what should be outputted at the current time instant. Image courtesy of [72]

More in detail, considering as input both $\mathbf{h}_{t-1}$ and $\mathbf{x}_t$:

- the *forget gate* $\mathbf{f}_t$ determines the information that can be discarded from the previous cell state. Through a *sigmoid* non-linearity $\sigma(\cdot)$, it looks at the input and decides which values to update. Being the output of a sigmoid, the values will be included between 0 and 1, with 0 equals to "totally forget this value" to 1 equals to "keep it entirely in memory". The equation describing this gate is Eq. 1.5;

$$\bar{\mathbf{f}}_t = \mathbf{W}_f \mathbf{h}_{t-1} + \mathbf{U}_f \mathbf{x}_t + \mathbf{b}_f \tag{1.4}$$

$$\mathbf{f}_t = \sigma(\bar{\mathbf{f}}_t) \tag{1.5}$$

- the *input gate* $\mathbf{i}_t$ chooses what part of the input to be saved in the cell state

through Eq. 1.7. Again, the reasoning about the sigmoid behavior holds for the input values;

$$\bar{\mathbf{i}}_t = \mathbf{W}_i \mathbf{h}_{t-1} + \mathbf{U}_i \mathbf{x}_t + \mathbf{b}_i \tag{1.6}$$

$$\mathbf{i}_t = \sigma(\bar{\mathbf{i}}_t) \tag{1.7}$$

- the *output gate* $\mathbf{o}_t$ filters the cell state following Eq. 1.9 to produce the new hidden state.

$$\bar{\mathbf{o}}_t = \mathbf{W}_o \mathbf{h}_{t-1} + \mathbf{U}_o \mathbf{x}_t + \mathbf{b}_o \tag{1.8}$$

$$\mathbf{o}_t = \sigma(\bar{\mathbf{o}}_t) \tag{1.9}$$

In parallel to this gating mechanism, a candidate cell state vector $\tilde{\mathbf{C}}_t$ to be added to $\mathbf{C}_{t-1}$ is created from the input with a *tanh* non-linearity, like in the RNN cell (Eq. 1.11):

$$\bar{\mathbf{C}}_t = \mathbf{W}_C \mathbf{h}_{t-1} + \mathbf{U}_C \mathbf{x}_t + \mathbf{b}_C \tag{1.10}$$

$$\tilde{\mathbf{C}}_t = tanh(\bar{\mathbf{C}}_t) \tag{1.11}$$

Such a candidate vector is first combined with the result of the input gate, to scale the values to how much we decided to update the previous cell state, and then combined with the result of the forget gate, as in Eq. 1.12:

$$\mathbf{C}_t = \mathbf{f}_t \odot \mathbf{C}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{C}}_t \tag{1.12}$$

Finally, we first scale the new cell state vector between -1 and 1 with a second *tanh* operation, before filtering it with the result of the output gate (Eq. 1.13):

$$\mathbf{h}_t = \mathbf{o}_t \odot tanh(\mathbf{C}_t) \tag{1.13}$$

The current idea of vanilla LSTMs is actually the result of a long process of experiments and improvements. The first implementation proposed in 1997 in [73], indeed, presented many fundamental differences:

- It did not have the forget gate and, for some experiments, the network was tested respectively in absence of the output gate, unit biases, and input activation functions;
- It did not train using only the *Backpropagation Through Time* algorithm [75, 76], but a mixture of BPTT and Real-Time Recurrent Learning (RTRL) [77], backpropagating the gradient of the cells and truncating the one for the recurrent connections;
- It was using a *full gate recurrence*, receiving not only the previous output as part of the current input, but also feeding recurrently the gates with their previous values. This feature actually disappeared from any later paper.

The forget gate was introduced in 1999 in [78] to learn continual tasks and allow the LSTM to reset its own state, while full BPTT was used for the first time in 2005 in [79] to train a LSTM network on the TIMIT benchmark [80]. This made possible to check the gradient using finite differences and the practical implementation more reliable.

Backpropagation Through Time represents an extension of the classical *Backpropagation algorithm* to the case of RNNs. Since the latter unfold through time, the weights update as well requires to be computed from the last current time step to the initial one before being applied to the weight matrices. As compactly summarised in [81], such operation can be separated in two main steps:

- the computation of the deltas inside the LSTM block, with the following

set of equations:

$$\delta \mathbf{h}_t = \Delta_t + \mathbf{W}_C^T \delta \tilde{\mathbf{C}}_{t+1} + \mathbf{W}_f^T \delta \mathbf{f}_{t+1} + \mathbf{W}_i^T \delta \mathbf{i}_{t+1} + \mathbf{W}_o^T \delta \mathbf{o}_{t+1} \tag{1.14}$$

$$\delta \bar{\mathbf{o}}_t = \delta \mathbf{h}_t \odot tanh(\mathbf{C}_t) \odot \sigma'(\bar{\mathbf{o}}_t) \tag{1.15}$$

$$\delta \mathbf{C}_t = \delta \mathbf{h}_t \odot \mathbf{o}_t \odot tanh'(\mathbf{C}_t) + \delta \mathbf{C}_{t+1} \odot \mathbf{f}_{t+1} \tag{1.16}$$

$$\delta \bar{\mathbf{f}}_t = \delta \mathbf{C}_t \odot \mathbf{C}_{t-1} \odot \sigma'(\bar{\mathbf{f}}_t) \tag{1.17}$$

$$\delta \bar{\mathbf{i}}_t = \delta \mathbf{C}_t \odot \tilde{\mathbf{C}}_t \odot \sigma'(\bar{\mathbf{i}}_t) \tag{1.18}$$

$$\delta \bar{\mathbf{C}}_t = \delta \mathbf{C}_t \odot \mathbf{i}_t \odot tanh'(\bar{\mathbf{C}}_t) \tag{1.19}$$

where Δ_t corresponds to $\frac{\partial L}{\partial \mathbf{h}_t}$, with L equal to the chosen loss function. If there is another layer below, also the deltas for the input need to be computed:

$$\delta \mathbf{x}_t = \mathbf{U}_C^T \delta \bar{\mathbf{C}}_t + \mathbf{U}_f^T \delta \bar{\mathbf{f}}_t + \mathbf{U}_i^T \delta \bar{\mathbf{i}}_t + \mathbf{U}_o^T \delta \bar{\mathbf{o}}_t \tag{1.20}$$

- and the weights gradients computation as follows:

$$\delta \mathbf{W}_\star = \sum_{t=0}^{T-1} < \delta \star_{t+1}, \mathbf{h}_t > \tag{1.21}$$

$$\delta \mathbf{U}_\star = \sum_{t=0}^{T} < \delta \star_t, \mathbf{x}_t > \tag{1.22}$$

$$\delta \mathbf{b}_\star = \sum_{t=0}^{T} \delta \star_t \tag{1.23}$$

with $\star$ representing any of $\{\bar{\mathbf{C}}, \bar{\mathbf{f}}, \bar{\mathbf{i}}, \bar{\mathbf{o}}\}$, and $< \star_1, \star_2 >$ denoting the outer product operation between two vectors.

With these δs, we can finally update the values themselves following the chosen algorithm. For instance, using Stochastic Gradient Descent (SGD):

$$W^{new} = W^{old} + \lambda \cdot \delta W^{old} \tag{1.24}$$

with λ equal to the learning rate.

1.3.1 LSTM common architectures and variants

Over the years, several LSTM-based architectures and variants have being proposed. For the sake of clarity and space, and not being directly involved in this thesis work, I am not going to list all of them. Yet it is worth mentioning the most famous ones.

Gers et. al [82], in 2000, introduced the idea of *peephole connections*, a set of connections from the CEC to the gates (Fig. 1.8) to control them, making precise timings easier to learn. In some variants of the peephole version, only a subset of gates might be connected to the CEC. Eqs. (1.5),(1.7),(1.9) in the original set

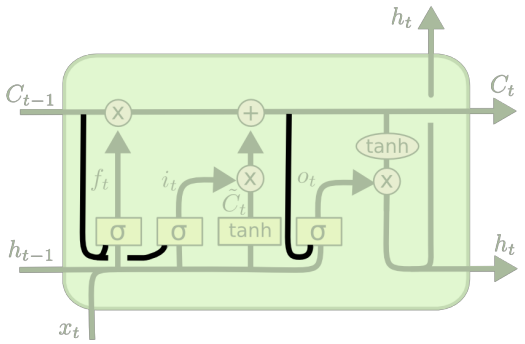


Figure 1.8: The peephole LSTM introduces a set of connections from the CEC to the gates. As reported in the paper introducing such variant [82], these connections make precise timings easier to learn. Image courtesy of [72]

change with the addition of the cell state to the three gates:

$$\mathbf{f}_t = \sigma(\mathbf{V}_f\mathbf{C}_{t-1} + \mathbf{W}_f\mathbf{h}_{t-1} + \mathbf{U}_f\mathbf{x}_t + \mathbf{b}_f) \tag{1.25}$$

$$\mathbf{i}_t = \sigma(\mathbf{V}_i\mathbf{C}_{t-1} + \mathbf{W}_i\mathbf{h}_{t-1} + \mathbf{U}_i\mathbf{x}_t + \mathbf{b}_i) \tag{1.26}$$

$$\mathbf{o}_t = \sigma(\mathbf{V}_o\mathbf{C}_{t-1} + \mathbf{W}_o\mathbf{h}_{t-1} + \mathbf{U}_o\mathbf{x}_t + \mathbf{b}_o) \tag{1.27}$$

Peephole LSTMs were recently used in [83] for *wind speed prediction*, in order to guarantee a higher level of safety for smart grids, in [84] for *ultra-short-term photovoltaic power forecasting*, and in [85] for *load prediction on distributed systems*, to better relocate the load and avoid limitations on such systems. In [86], instead, they were used in combination with convolutional layers for *abstractive text summarisation*.

Bidirectional RNNs, initially proposed in 1997 in [87] and extended to LSTMs in 2005 in [79], represent a way to increase the amount of information available to the network before producing the output. They present two layers working in opposite directions (Fig. 1.9): the first one follows the "time direction" inputting always newer information; the second one proceeds backward, inputting from the last point to the first one. In such a way, the output layer can get information simultaneously from a forward and a backward run, recreating a better context before the output.

Having two hidden layers now, the computation of $\mathbf{h}_t$ in Eq. (1.13) gets split

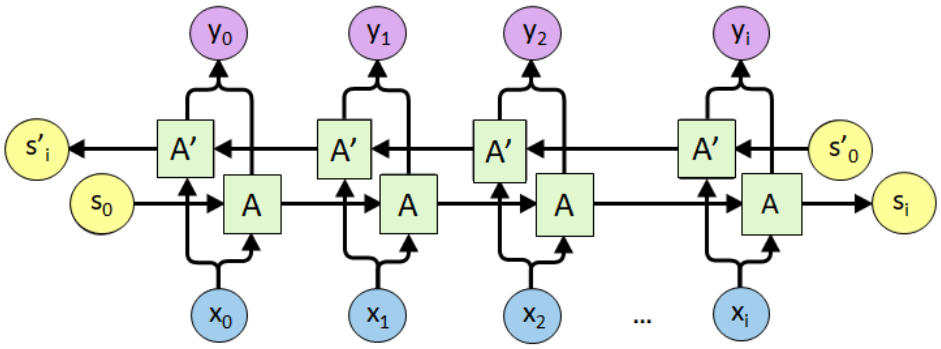


Figure 1.9: Bidirectional LSTMs are composed of two independent layers, one receiving inputs in their natural temporal order, and one in the inverse order. Together, they contribute to the output of the network, ensuring higher accuracy at the price of having the whole sequence already available at input time.

into two vectors, and the final output becomes a combination of the two:

$$\overrightarrow{\mathbf{h}}_t = \Phi(\mathbf{W}_{\overrightarrow{h}}\mathbf{h}_{t-1} + \mathbf{U}_{\overrightarrow{h}}\mathbf{x}_t + \mathbf{b}_{\overrightarrow{h}}) \tag{1.28}$$

$$\overleftarrow{\mathbf{h}}_t = \Phi(\mathbf{W}_{\overleftarrow{h}}\mathbf{h}_{t+1} + \mathbf{U}_{\overleftarrow{h}}\mathbf{x}_t + \mathbf{b}_{\overleftarrow{h}}) \tag{1.29}$$

$$\mathbf{y}_t = \mathbf{W}_{\overrightarrow{h}}\overrightarrow{\mathbf{h}} + \mathbf{W}_{\overleftarrow{h}}\overleftarrow{\mathbf{h}} + \mathbf{b}_y \tag{1.30}$$

with $\overrightarrow{\mathbf{h}}_t$ representing the hidden state of the *forward layer*, $\overrightarrow{\mathbf{h}}_t$ the hidden state of the *backward layer*, and Φ the activation function. Bidirectional LSTMs are useful for non-real-time applications, where the future input is already available. They proved to be very effective for *speech recognition* [88–92], *machine translation* [93, 94], *image/video captioning* [95–98] and *handwritten recognition* [99–102].

A more drastic variation are *Gated Recurrent Unit (GRU)* networks, introduced in 2014 in [103]. This type of network makes use of an *update gate*, obtained from the merging of the forget and input gates, and a *reset gate* to control the re-defined hidden state, obtained also from the union of the previous cell state and hidden state (Fig. 1.10). The set of equations describing the GRU model is:

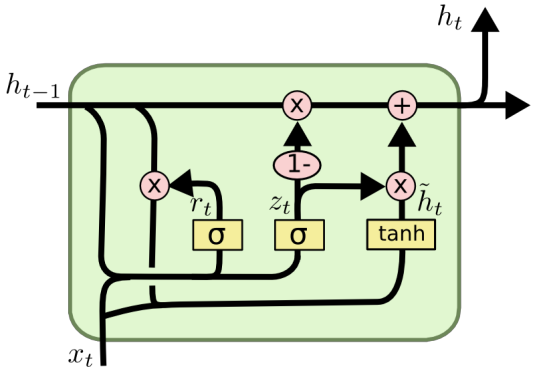


Figure 1.10: The GRU node variant presents a simplified inner structure with only two gates, a reset gate and an update gate, that manage current input, previous hidden state and current output. Image courtesy of [72]

$$\mathbf{z}_t = \sigma(\mathbf{W}_z \mathbf{h}_{t-1} + \mathbf{U}_z \mathbf{x}_t + \mathbf{b}_z) \tag{1.31}$$

$$\mathbf{r}_t = \sigma(\mathbf{W}_r \mathbf{h}_{t-1} + \mathbf{U}_r \mathbf{x}_t + \mathbf{b}_r) \tag{1.32}$$

$$\tilde{\mathbf{h}}_t = tanh(\mathbf{W}_h(\mathbf{r}_t \odot \mathbf{h}_{t-1}) + \mathbf{U}_h \mathbf{x}_t) \tag{1.33}$$

$$\mathbf{h}_t = (1 - \mathbf{z}_t) \odot \mathbf{h}_{t-1} + \mathbf{z}_t \odot \tilde{\mathbf{h}}_t \tag{1.34}$$

GRUs are being often used for *speech recognition* [104–106], but also for *intrusion detection in network traffic data* [107], *prediction of deterioration processes and remaining useful life of systems* [108], and *photovoltaic forecasting* [109].

LSTMs are often combined also with another famous class of Deep Learning methods, the Convolutional Neural Networks architectures. Initially referred to as *Long-term Recurrent Convolutional Networks* (LRCNs) [110] in 2015, they are now famous with the name of *CNN-LSTMs*. As from the name, the network involves convolution layers, to extract relevant features from the input data, and recurrent layers, to support temporal sequences, being in such a way deep both spatially and temporally (Fig. 1.11). It gained a lot of success in visual tasks involving sequential input/outputs and text generation from sequences of images, like *image/video description* and *activity recognition*, but also in Natural Language Processing (NLP) problems, where the CNN layers are used to extract features from audio/text sequences or even images like audio spectrograms. Given also the power of CNNs, CNN-LSTM were successfully applied to several tasks, like *sentiment analysis/recognition* [111, 112], *words recognition* [113, 114], *robot in-hand manipulation* [115], *veichle trajectory prediction* [114] and *image foreground segmentation* [116]

The *Phased LSTM* [70] introduces a new *time gate* $\mathbf{k}_t$, which opening and closing determines when to update the CEC and the hidden state $\mathbf{h}_t$ (Fig. 1.12). The state of this gate is controlled by a rhythmic oscillation, specified by three parameters: τ, controlling the oscillation period in real-time, r_{on} controls the

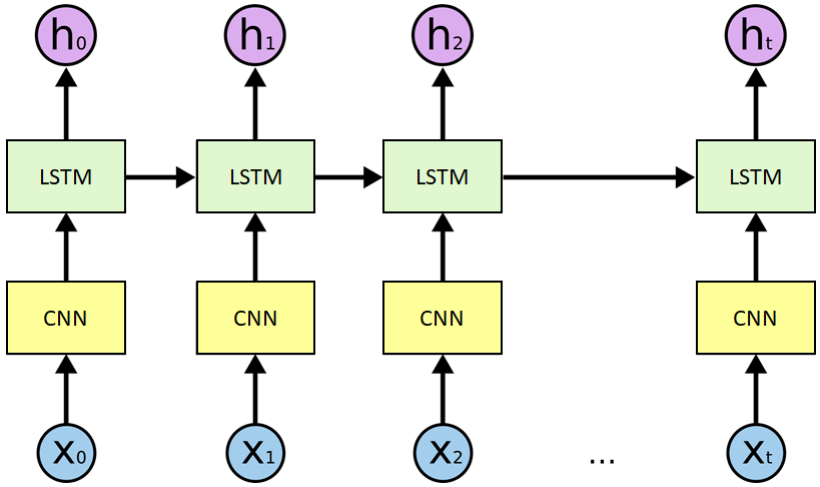


Figure 1.11: The CNN-LSTM architecture, as per its name, combines a Convolutional Neural Network for features extraction with a LSMT network for tracking their changes in time. This makes the network deep both in space and time. Image courtesy of [72]

ration between the "open" and the "closed" phase, and the shift s controls the phase shift for each Phased LSTM cell period.

Eqs. (1.12), (1.13) now become:

$$\tilde{\mathbf{C}}_t = \mathbf{f}_t \odot \mathbf{C}_{t-1} + \mathbf{i}_t \odot tanh(\mathbf{W}_c \mathbf{h}_{t-1} + \mathbf{U}_c \mathbf{x}_t + \mathbf{b}_c) \tag{1.35}$$

$$\mathbf{C}_t = k_t \odot \tilde{\mathbf{C}}_t + (1 - k_t) \odot \mathbf{C}_{t-1} \tag{1.36}$$

$$\tilde{\mathbf{h}}_t = \mathbf{o}_t \odot tanh(\tilde{\mathbf{C}}_t) \tag{1.37}$$

$$\mathbf{h}_t = k_t \odot \tilde{\mathbf{h}}_t + (1 - k_t) \odot \mathbf{h}_{t-1} \tag{1.38}$$

Even though Phased LSTM are not as famous as the previously introduced variants, they were applied successfully to gait prediction [117], water quality evaluation [118], and to predict longitudinal Electronic health records (EHRs) [119].

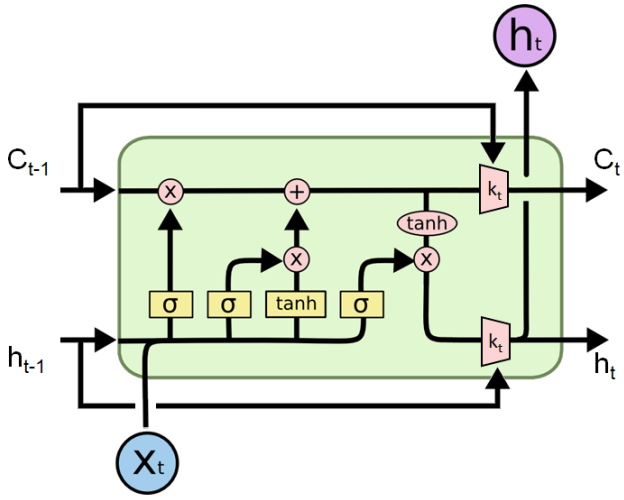


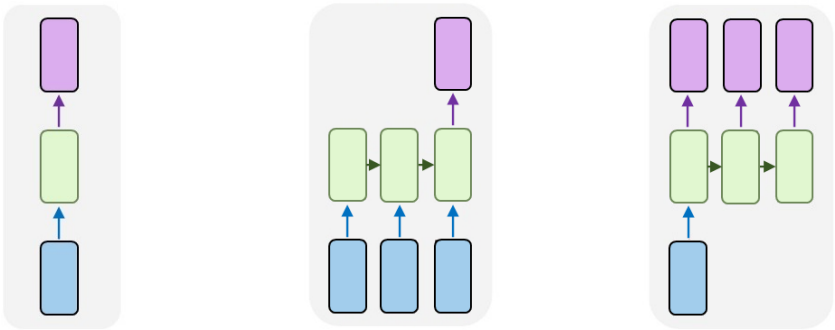
Figure 1.12: The authors of [70] propose a variant of the LSTM, specifically named Phased LSTM, that presents an additional time gate for dealing with inputs with different periods. The update of the CEC can happen only when this gate is open.

1.3.2 Sequence-to-Sequence problem settings

Main goal of a prediction problem is often the forecasting of the next value of a real valued sequence, or the labelling of the input choosing among a set of possible classes. Such goals are often framed in a *one-to-one problem* (Fig. 1.13a), where the input is the information of a single time step and so is the output. This is the typical structure of a standard Fully Connected network or Convolutional Neural Network, where any temporal linking between two network queries is ignored. A clear example is, indeed, the *image classification* task.

A different approach, instead, suggests to consider the temporal relationship between consecutive inputs, for it improves the network capability of discriminating among the different possible outputs. Such approach is named *many-to-one problem* (Fig. 1.13b) and it is the classical architecture of, for instance, a *sentiment analysis task*, where a sentence or a video is given as input and a 0 or a 1 is outputted whether the sentiment expressed is negative or positive. In the

opposite direction, the *one-to-many* problem (Fig. 1.13c) tries to build an output sequence from a single output. This is in general a tough task, given the limited information contained in a single input and the possible variants of a sequential output. A sample task is *image captioning*.

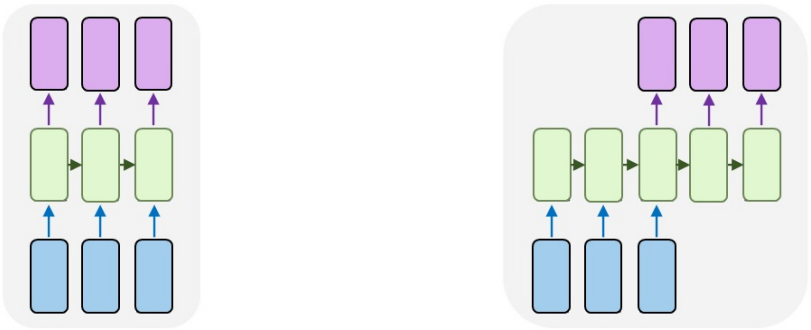


(a) *(b)* *(c)*

Figure 1.13: Typical problem settings are the one-to-one (e.g. image classification), many-to-one (e.g. sentiment analysis) and one-to-many (e.g. image captioning)

One more challenging type of prediction is, however, the problem that takes a sequence both as input and as output. We refer to it as a *many-to-many* or *sequence-to-sequence problem*. Input and output can be synchronised (Fig. 1.14a) or not (Fig. 1.14b). A classical example of the first case can be the *video classification task*, where the goal is to label each frame of a video exploiting information from past frames; while, for the asynchronous case, it can be considered the *machine translation task*, where a sentence must be entirely fed in input before start producing the output, in order to not have a literal word-for-word translation. Issues in the problem modelling arise when the input and output lengths can vary from one query to the other, or even between the input and output themselves.

The setting chosen for this work is the *many-to-many* in its asynchronous



(a) (b)

Figure 1.14: Sequence-to-Sequence problems involve a temporal sequence both in input and in output. Such architecture can be synchronous (a), with an output after each input, or asynchronous (b), with the prediction outputted only after the whole input sequence has ben fed to the network. Input and output sequences might also have different lengths.

version (Fig. 1.14b). This is because, once a new trajectory point is available, it is preferable to first feed the entire input sequence and then get the output prediction on the base of the latter. An approach like the synchronous one would generate output points from the beginning, when the network received only one or few informative points and cannot really discriminate the input or the temporal dynamics of the sequence.

1.3.3 The Encoder-Decoder architecture

Special mention and a detailed description is deserved by the *Encoder-Decoder* architecture for its use in this thesis work. I decided to rely specifically on this model because of its natural adaptation to the problem I address. Bidirectional LSTM cannot be used since, at real-time, data from the future are not available. I preferred this architecture to a CNN-LSTM - still a valid alternative - for two reasons: (i) I wanted to isolate the problem of prediction inputting trajectory points obtained by means of a tracker, without instead considering the whole scene, (ii) the absence of a well established event-based CNN model requires to

resort to event images and classical CNN, which I did not want to consider in this work, so as to remain in the pure event-based domain.

Designed especially for Sequence-to-Sequence models, the Encoder-Decoder architecture proved to be effective in several types of task. As from the name, it comprises of three elements:

- The *Encoder* network, a stack of recurrent units taking the temporal input sequence step-by-step and propagating the information forward, outputting the final hidden state of the network;
- The *intermediate vector*, result of the Encoder forward pass, encapsulating the input information in its dimensionality, and acting then as the initial hidden state of the decoder part of the model;
- The *Decoder* network, a second stack of recurrent units taking no input but initialised with the intermediate vector, and producing the output temporal sequence.

Given the use of an internal representation used both from input and output, the method can be considered also a case of *Sequence Embedding*. A full scheme of the model can be seen in Fig. 1.15.

In [120], where the Encoder-Decoder network is proposed the first time, such architecture is applied to the task of machine translation from English to French on the WMT'14 dataset, obtaining a BLEU score of 34.8 and achieving the best result for the time. It also showed how good is this model even for long sequences and how robust it is regarding sensible phrase and sentence representations sensitive to word order. The same idea is applied with convolution in [121], where it

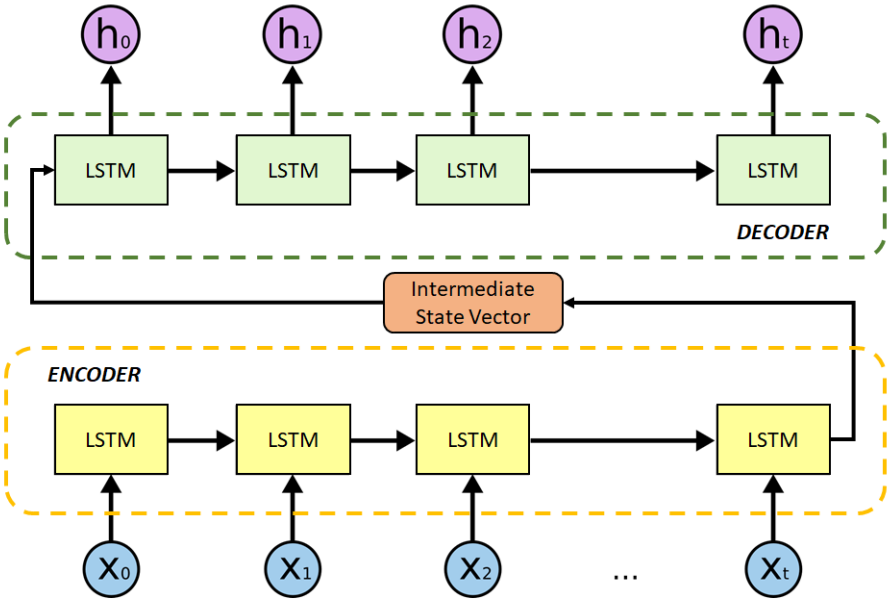


Figure 1.15: The Encoder-Decoder architecture gained a lot of attention thanks to its ability of mapping input sequences and output sequences of different length. This property resulted particularly useful for text translation tasks, where this model has been firstly applied [120]. This separation is possible thanks to the intermediate vector, a representation in an embedding space of the input after this has been fed entirely in the Encoder, and that triggers then the Decoder to output the prediction.

is proposed a network entirely convolutional that can be fully parallelised. State-of-the-art performance are defined both for the WMT'14 English-German and WMT'14 English-French translation datasets, with a speed higher of an order of magnitude with respect to previous best architectures. Many approaches for speech recognition make use of intermediate representations for converting from one language to another. Even slightly under-performing a baseline cascade of a direct speech-to-text translational model followed by a text-to-speech synthesis model, the network proposed in [122] can directly translate a language without needing intermediate text representations. Trained directly end-to-end, the network maps speech spectrograms into target spectrograms of the second language.

Again in the "words-domain", [123] proposes an interesting state-of-the-art review of Sequence-to-Sequence models for speech recognition, a task becoming every day more relevant with the introduction of many voice-commanded devices.

Another fundamental application is the one proposed in [124], where image and video captioning are automatically performed by means of a Sequence-to-Sequence architecture. This network is trained on video-sentence pairs and learns to describe with text a sequence of video frames.

A very interesting and particular application is the training of a network to evaluate short computer programs, a domain that has been traditionally seen as too complex for neural networks. In [125] a simple class of codes that can be evaluated with a single left-to-right pass is used as input data for an Encoder-Decoder in order to learn to map the character-level representations of such programs to their correct outputs. A variant of Curriculum Learning is implemented to train an LSTM to add two 9-digit numbers with 99% accuracy.

Both [126] and [127] address the problem of conversational speech modelling, crucial capability for chatbots, digital assistants and smart home devices, to mention few examples. The authors in [126] present a large scale dataset of conversational queries and various Sequence-to-Sequence models that learn from it to reformulate over half of all conversational queries. Formulating a conversational query understanding problem as a context-aware query reformulation, the input speech is reformulated into a search engine friendly question. Such context-aware query reformulation problem is modeled as a Sequence-to-Sequence problem.

Moving finally toward computer vision and robotics tasks, in [128] the authors use an Encoder-Decoder network architecture for solving a dense pixel-prediction problems. Specifically, the background subtraction in video sequences, showing good generalisation performance even with training video sequences of limited duration. An Encoder-Decoder model is also used for 3D pose estimation of

humans in [129]. The proposed method shows comparable results to the state-of-the-art in skeletal motion prediction.

1.4 Motion prediction

With the growing number of intelligent and autonomous systems in human environments, a lot of attention is being given to human motion trajectory prediction. Surveillance systems, self-driving vehicles and service robot need to anticipate human behavior for safety and smooth physical interaction. It is almost impossible to define a major direction for which set of methods to use to address prediction tasks. Both model-based and data-driven approaches resulted to be very effective in several tasks, but Encoder-Decoder LSTM resulted to be one of the most used, with almost every new paper proposing a variant or a combination of different tools.

An important work in the pedestrian trajectory prediction field is represented by the introduction of *Social LSTMs* in [130]. In this framework, a separate LSTM network is instantiated for each trajectory in the scene. The novelty is represented by a *social pooling layer* that connects networks belonging to nearby trajectories, allowing them to share information with each other through their hidden state values (Fig. 1.16). Fig. 1.17 qualitatively shows the results obtained in [130] on scenes of the ETH dataset [131], and the classical scenario of this task: a camera placed on top of the scene, framing groups of people whose trajectory has to be predicted. The work outperformed state-of-the-art level on two publicly available datasets, the ETH and the UCY [132], and gave origin to the trend of addressing human trajectory prediction in crowded environment with Recurrent Neural Networks [133–141]. As highlighted in [142], however, the only use of social pooling as it is lacks the modelling of human-human interaction, which is

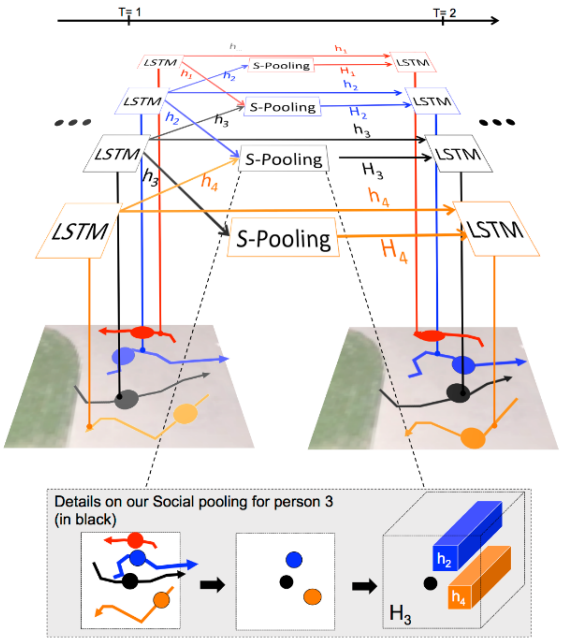


Figure 1.16: In [130] a social pooling layer is introduced between two layers of a LSTM network. Instantiating a network for each tracked pedestrian, this intermediate layer allows the exchange of information between networks associated to nearby trajectories.

Figure 1.17: The introduction of such a social pooling layer brought the work in [130] to define a new state-of-the-art for pedestrian trajectory prediction, encouraging the use of LSTM on the task.

proved to play a crucial role for more accurate and longer-term trajectory estimation. To this purpose, the authors of [142] propose the use of the pedestrians's head poses - named *vislets* - as additional network features, to strengthen the social aspect of the framework and improve overall performance.

Another field gaining exponential popularity year after year is the one of autonomous vehicles. Predicting what is going to be the behavior of surrounding vehicles and traffic participants is very hard, as it depends also on many factors often out of the system perception, like other driver's intention or the traffic condition. Nevertheless, in this task safety is crucial as in few others and the capability of anticipating others' movements is vital. A probabilistic approach to prediction is considered in [143]. From a dataset of trajectories, motion patterns are extracted and projected in a low dimensional manifold by computing the Chebyshev decomposition first, and using the Unsupervised Kernel Regression (UKR) then. A particle filter is used at run-time to track the history of the trajectory in such subspace and output the most probable position as future prediction. A model-based and maneuver recognition approach is proposed in [144], combining the so called Constant Yaw Rate and Acceleration (CYRA) tracking system and the Maneuver Recognition Module (MRM). The first is used for short term prediction, having resulted to be the best target tracker assuming a certain motion model from a comparison made in [145]. In the long term, however, the algorithm fails, and that is why the MRM is introduced. It uses a Hidden Markov Model (HMM) to select the most likely trajectory based on a set of possible trajectories and the current measurements sequence. A similar work is carried out in [146], where the Interactive Multiple Model Trajectory Prediction (IMMTP) method is introduced. Goal of the study is to build an integrated approach that combines physics- and maneuver-based prediction methods for ob-

taining an accurate long term prediction. The physics-based prediction makes use of the Unscented Transform (UT), but results indicate that such a physical model can achieve accurate predictions only in the short term, while the long term suffers a significantly large covariance. In the maneuver-based prediction, a Dynamic Bayesian Network (DBN) is used to model long term behaviors, considering sequential information. Result shows that the prediction variance for this method does not increase considerably in time, but in the short term the accuracy is lower than the physics-based method. Extending the IMM approach used in [147] and [148], the authors propose the IMMTP model, combining both the physical- and the maneuver-based terms, and where the probability of each of them recursively adjusts according to the predicting variance of the model. With the growing interest for Recurrent Neural Networks, LSTMs started to be applied also to this kind of prediction. In [149], a vanilla LSTM network is used to predict the future location of surrounding vehicles over an Occupancy Grid Map (OGM) imagined in front of the driven car. Here it is showed that such a method presents higher accuracy with respect to the existing Kalman filter-based ones. A similar work is carried out in [150]. The same authors of [149] brought the work to the next step by implementing a Encoder-Decoder architecture for vehicle trajectory prediction in [151]. The model takes the latest trajectory points from surrounding vehicles, along with the ego-motion of the car over the OGM, and recursively produces predicted paths. The latter are not directly used for avoidance maneuvers, but they are fed into a *Beam Search* algorithm, that keeps memory of the K locally best candidate sequences for generating the actual prediction at every time step (Fig. 1.18). With these, autonomous maneuvers can performed. A combination of techniques saw in [151] and [130] is presented in [152], where, in between of the Encoder and Decoder model, a Social Pooling layer is introduced for vehicles. Results show that using such a layer, instead of common fully con-

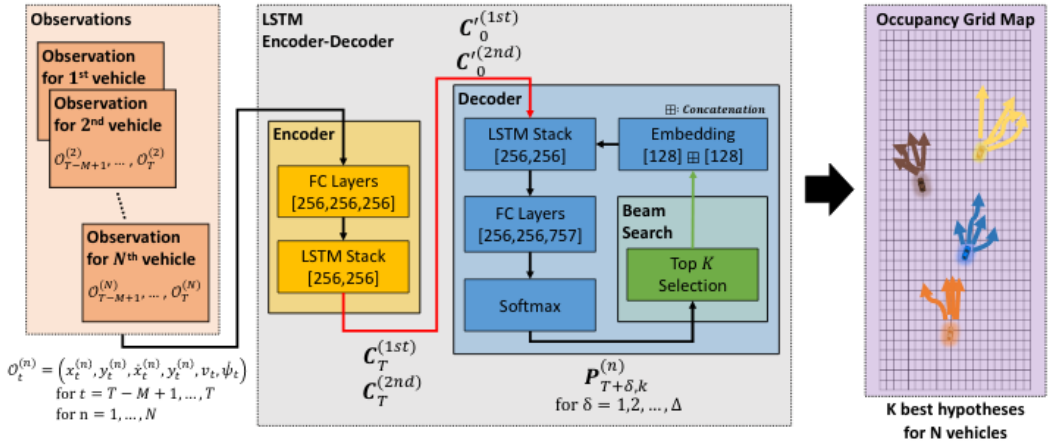


Figure 1.18: An Encoder-Decoder model is used in [151] in combination with a Beam Search algorithm for trajectory prediction of surrounding vehicles over an Occupancy Grid Map. The network takes the latest trajectory points of the surrounding cars, along with the ego-motion information, and keeps track of the K best possible candidate predictions, improving the network robustness and generalisation.

nected ones, improves the model robustness and generalisation capabilities over various spatial configurations with interacting agents.

The idea of using prediction to plan and move robots in advance has also been addressed in literature. The latency in the perception-action chain, indeed, can have a crucial impact on the robot performance, and discriminate between the success or the failure of the task. A deep analysis of the role of perception latency in high-speed robots is presented in [153]. The study considers the case of a robot moving from a point A to a point B in as little time as possible - assuming therefore a constant longitudinal velocity that cannot be changed - and in an environment with static obstacles. Goal of the paper is the derivation of the relationship between sensors and actuators for the maximum speed the robot can achieve to safely navigate. Moreover, it presents real-case results considering the test bed of a quadrotor. Interesting point is also the comparison of the

results between a single frame-based camera, a frame-based stereo system and a monocular event-based camera. As discussed in Sect. 1.2, working with frames requires the analysis of the full pixels matrix in order to detect an obstacle, independently of the texture. Conversely, event-based systems have the advantage of requiring very little processing and a very short time interval between two events, making the obstacle detection faster and allowing higher speed when the robot is particularly agile. Similar work to [153] is [154], addressing the problem of vision-based navigation for mobile robots. A prediction framework to compensate for the latency in the perception-action loop in a robotic platform is first proposed in [155]. Even though experimental results showed the benefits of this framework, they are not compared against the absence of the latter on the task at hand, ignoring the effects of such a latency on it.

A challenging problem in robotics where prediction can be really helpful is the task of catching objects thrown by a person. Knowing with anticipation where the object is going to be can help planning in advance the best way to catch it, if possible. A deep study of such problem is presented in [156]. The authors present a framework for teaching a robot to catch in-flight objects, composed of different modules. First, a modelling of the reachable space of the robot considered is performed, that will be used along with the prediction to compute the optimal grasping pose; second, it is the prediction algorithm for accurately estimate the trajectory of the fast-moving object; and third, it is the learning of the hand-finger coordination timing to avoid the bounce or misplacement of the object during the catching phase. The framework relies on the OptiTrack [157] for the marker tracking, both of the object and the human master during the learning phase. The trajectory prediction part, instead, is based on the work proposed in [158]. Here, through demonstrations of launches, the dynamics of complex objects with arbitrary shape and/or non-rigid bodies is modeled by means of a

dynamical system. No prior information about the object is required. Combined with a Extended Kalman Filter (EKF) to deal with noise and perturbations, this regression method refines the predicted trajectory time step after time step. The ball catching task is of high interest in robotics. It is helpful to understand the real capabilities of a tracker/prediction system and control strategy for a robot. In [159], for instance, a framework for high-speed motion generation and execution on mobile manipulators is proposed with this task as evaluation. A modified Kalman filter is proposed to estimate the ball's position and predict the ball's trajectory via numerical integration. Running at 5.1 ms, the estimator is combined with a learning-based inverse kinematics joint controller for planning the robot motion which achieve a success rate of 85.33%. The framework makes use of a powerful and costly vision system, the VICON [160], and the optimisation problem solved to control the robot is computationally heavy. In [161] a number of small high-speed cameras are arranged on the surface of a robotic hand for visual servoing control. In particular, while an external fixed high-speed camera system is used to track the ball and estimate the catching position, the "multi-vision hand" corrects the latter in real-time for catching the ball. Finally, a different approach to the trajectory estimation of a bouncing ball is proposed in [162]. In this work *online physical reasoning* is exploited for system identification, allowing to quickly estimate the parameters of the model by means of the combination of a dynamics model, a Variational Auto-Encoder (VAE) and the addition of a loss to enforce desired constraints. Experiments with a PR2 robot show that the latter is able to catch the ball the 77% of the time, mainly due to the fact that the prediction is based on single images and that the ball can move behind or in front of the gripper.

Most of these methods, however, present two fundamental characteristics: (i)

they rely on powerful, costly, power consuming external sensors, and (ii) the trajectories considered last typically for seconds, giving enough time to the prediction algorithm to gather sufficient information for adjusting the prediction step-by-step.

What I believed, and motivated me in addressing this task with event-based cameras - in addition to the previously mentioned advantages coming with this sensor - is that the latter applies a different type of sampling, a motion-driven spatial sampling, that can provide richer information with respect to standard fixed-rate samplings. Such cameras can give more, and more informative, points to the predictor, such as to enable it to converge to the right prediction in a short amount of time. This can be read also as the fact that the predictor can infer even trajectories with duration of less than a second - that, instead, would provide very few samples with a classical 30 Hz fixed-rate sampling for instance.

1.5 Preliminary study: iCub's internal delays

To further motivate the need of prediction capabilities, a preliminary analysis is conducted on the iCub robot [163] about the *vision delay* and the *control delay*.

Considering the classical structure of an application for the neuromorphic iCub in our lab (Fig. 1.19), I define:

- the *vision delay* as the time between the events acquisition from the zynqGrabber board in the iCub head, in charge of collecting events at FPGA level, and the availability of the information of interest in the application module in C++ (red arrows);
- the *control delay* as the time between the sending of the target position to the robot and its reaching by means of the low-level controller of the robot (blue arrows);

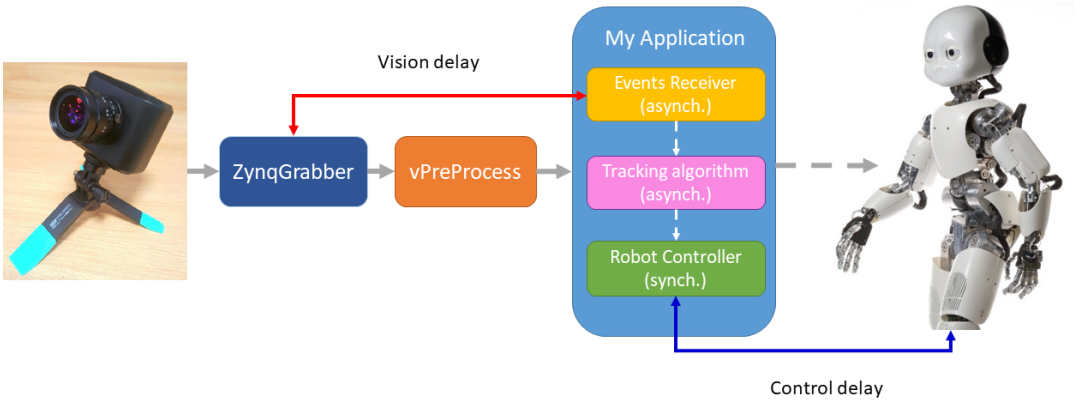


Figure 1.19: Classical scheme for an iCub application with event-based cameras. The vPreProcess module was developed in our lab to remove salt-and-pepper noise.

Fig. 1.20 shows single delays (in blue) every time a batch of events is received from the application module, and the average vision delay (in red) over a ∼65 s execution of the test. The latter results to be ∼2 ms. It must be taken into account, however, that in between of the events acquisition by means of the zynqGrabber and the availability of the information in the application, the pipeline includes a pre-processing operation, to remove salt-and-pepper noise, and a particle filter tracker [38], needed to generate the control reference signal. Both operations contribute, even slightly, to the delay

To compute the control delay, the target signal is generated tracking a circular object with the event-based tracker previously mentioned and developed in our lab, and mapping its vertical position in the field-of-view to a range of $15cm$ along the z axis of the Cartesian space. Also here, given the operations in the Cartesian space, the direct and inverse kinematics computation participate to the delay evaluation. Fig. 1.21 shows the target signal (in blue) and the robot hand position value read from the direct kinematics of the joints (in red). The signal presents both velocity and amplitude changes during its execution. Higher errors

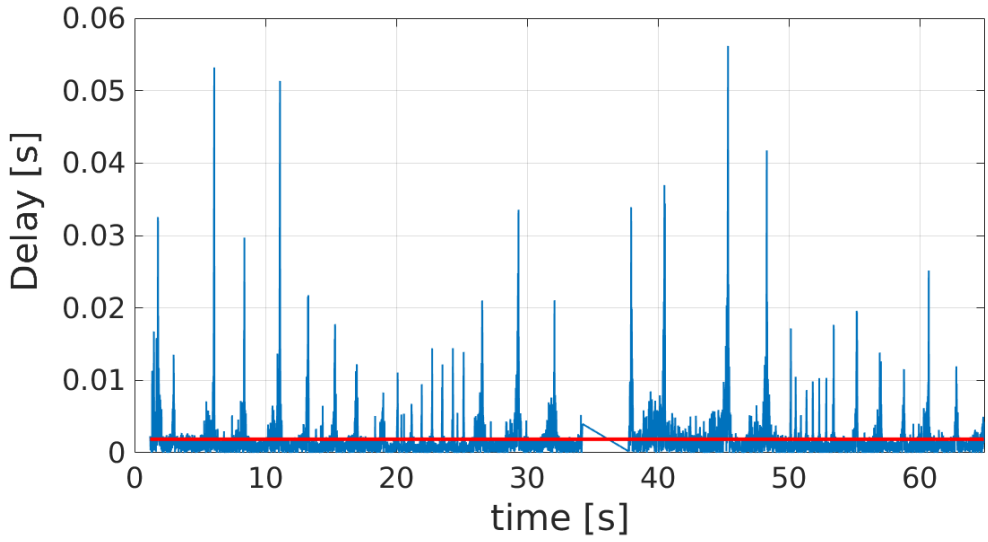


Figure 1.20: The vision delay is the time from when a batch of events is acquired by the zynqGrabber to the moment they are available in the YARP module. In blue are reported the vision delays for each batch, in red is the average vision delay.

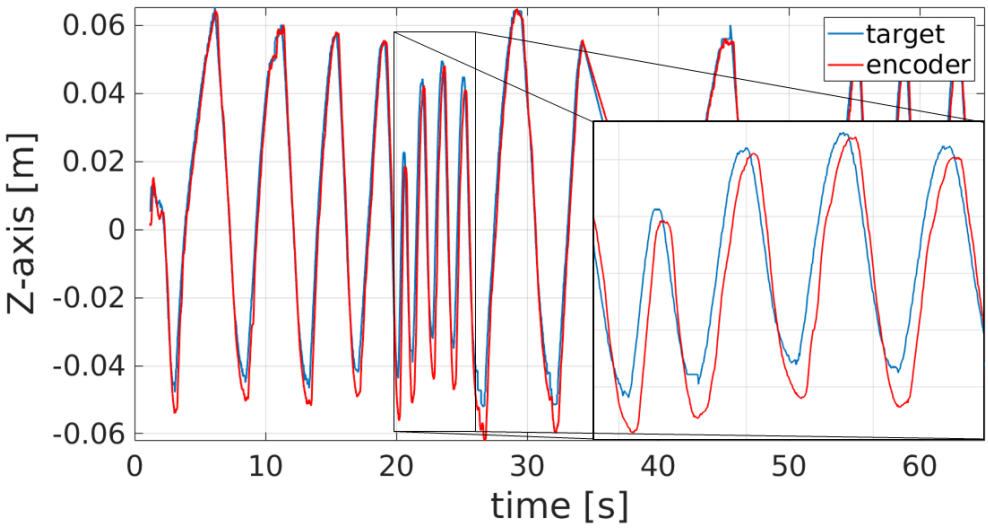


Figure 1.21: Target signal (in red) and actual Cartesian position of the iCub robot hand used for the experiment to compute the control delay.

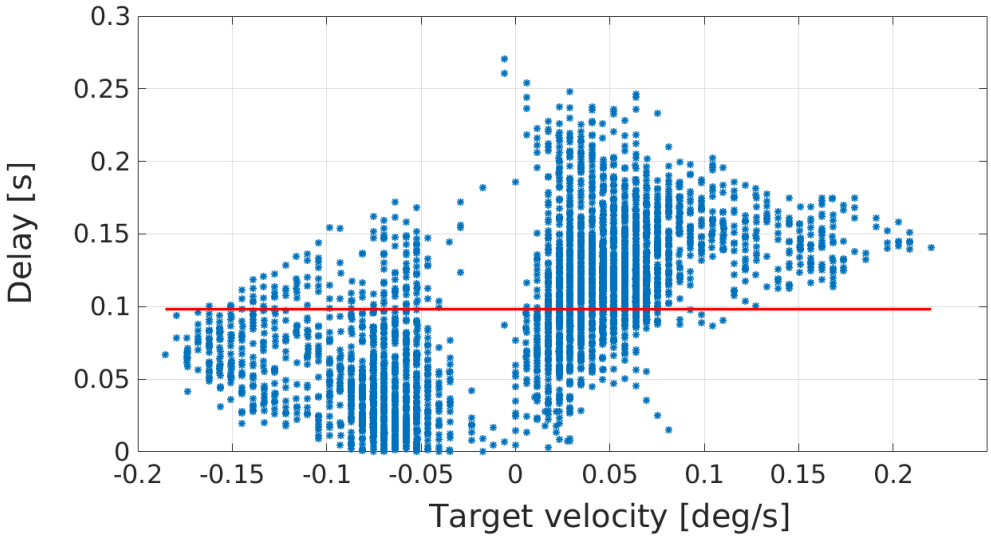


Figure 1.22: The control delay is the time between when a position in the Cartesian space is commanded and when it is finally reached. Individual control delays are reported as blue stars, while the red line is the average control delay.

are present when the arm is moving up; while, when moving down, the gravity "helps" the robot moving the arm faster.

A way to investigate this aspect and analyse the control delay, is to plot individual control delays with respect to the velocity commanded (Fig. 1.22). The robot, moving down the arm and applying negative velocities, presents much lower individual control delays. When moving up, instead, with positive velocities, the delays are higher. On average, the control delay can be estimated in the order of $\sim$100 ms. As expected, much higher than the vision delay. This means that, in order to compensate for such a delay, predictions should have a temporal horizon of at least the same amount of time.

Chapter 2

Where and When: Event-Based Spatio-temporal Trajectory Prediction from the iCub's Point-Of-View

The first part of this thesis addresses the problem of trajectory prediction using Long Short-Term Memory networks with event-based data and its comparison to different regression methods. In this chapter, the case of a person holding a ball and handing it to the robot is considered. Target of the pipeline is to acquire the latest N tracked points of the object and predict what should be the incoming part of the trajectory, with the future idea in mind of making the robot able to estimate a plausible exchange point and move in anticipation to take the object. In several fields indeed, like Human-Robot Interaction (HRI), Virtual Reality (VR) applications, gesture recognition and other types of real-time human-machine interaction, latencies between stimuli and responses play a crucial role. Systems are required to be highly responsive, with delays in the order of $100-200ms$ [164, 165] to be satisfactory to the human. The perception system should therefore be as fast as possible to give more time for input processing,

action planning and execution. To this purpose, the pipeline I imagined uses one of the two event-based ATIS cameras [4] mounted in the neuromorphic iCub [5, 163] humanoid robot for data acquisition. An event-based tracker is needed to take such raw events and compute the position of the center of the object, to be later fed to the neural network. The architecture implemented is an Encoder-Decoder model designed and tuned to possibly take input and output sequences of different lengths and to accomodate in input the (x, y) pixel spatial coordinates of the tracker and the relative temporal interval Δt between the current event and the previous one, useful to determine the dynamics of the motion. Experiments show that the Encoder-Decoder network is able to predict accurately the incoming trajectory and adapt the prediction horizon - meant as how far ahead in time the output prediction can "see" - to the motion speed. For high speed movements, indeed, long-term prediction are not accurate and we might want to estimate more intermediate points but in a shorter time window. Furthermore, failure cases and computational time are analysed. The full pipeline is summarised in Fig. 2.1

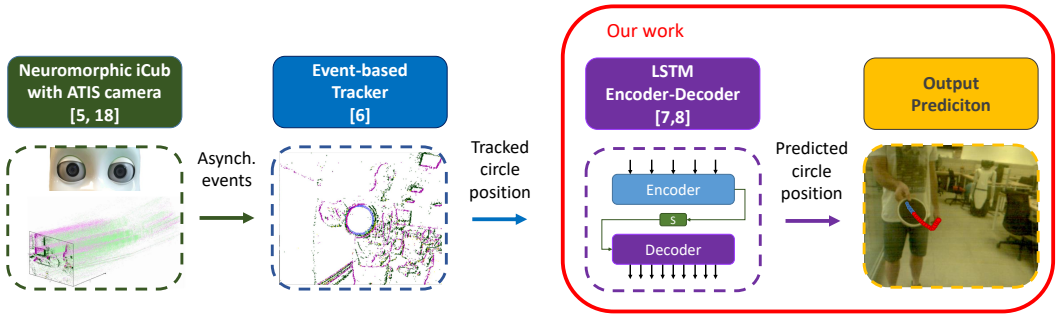


Figure 2.1: The full pipeline for predicting future positions of a handed object to the robot. The output of event-based cameras is fed into an event-based tracker, that asynchronously outputs the (x,y,t) position of the center of the object. Our contribution consists of the training of a LSTM Encoder-Decoder architecture that handles event-based data and predicts the future trajectory of the target, providing both spatial and temporal coordinates.

Differently from the temporal sampling performed by most of modern sensors, where a sample is produced at every time step, in spatial sampling a sample is produced every time the target moves of a given amount in space, triggering all the pixels along its trajectory. Key aspect of this framework is that the interval between two samples is not anymore fixed and depending on the device in use: time is now a variable from which information can be extracted, like the dynamics of the movement, and used as input in the LSTM network - or any other method. In such a way, it can be predicted not only *where* the tracked object is going to be, but also *when* it will pass at the predicted location.

As motion pattern, I considered a person handing an object to the iCub robot (Fig. 2.2). The robot must predict where the object is going - aim of this chapter - in order to intercept it smoothly with its own grasping strategy - task left to future work. The visual scene, however, presents noise due to the presence of the person in the field of view. Therefore, an event-based tracker robust to clutter [38], implemented in our lab, is integrated into the pipeline. Based on spatial sampling - since the output of the tracker is triggered by a change in position of the target - it perfectly suits the implementation of an asynchronous predictor.

The handover trajectory is also "unpredictable", as the person does not always have a consistent movement pattern (e.g., the trajectory can end closer to or farther from the robot). The hypothesis is that in this scenario learning is a crucial element to achieve good prediction results. A state-of-the-art sequence-based learning method, the LSTM, is therefore compared against model-based regression prediction, assuming linear, quadratic and sinusoidal motion patterns to model and predict from the noisy trajectory data.

The pipeline is also measured in terms of computational time, towards low-latency prediction systems for the iCub robot. Addressing the problem from the

Figure 2.2: The task considered in this work is a kind of handover task, where the person in front of iCub is handing an object to the robot, but can also retract the hand. This can be useful in future work to plan different behaviors for the robot.

iCub point-of-view is challenging as the pipeline does not rely on external, powerful but costly systems that can simplify the problem, but only on the on-board camera. This limits the field of view, and therefore the amount and quality of information obtainable, but is a problem that must be addressed to develop fully autonomous robots relying only on on-board sensors.

The LSTM Encoder-Decoder architecture, also known as Sequence-to-Sequence [120], allows to map fixed size input sequences to output sequences with different length. In [151] it is used for predicting the trajectory of surrounding vehicles of a self-driving car. The authors combined Encoder-Decoder LSTM and *Beam Search* to continuously predict a set of K most likely paths. About handover tasks, LSTM are used in [166] to directly generate the motion of the robot that has to grasp the object. Thanks to a motion capture system, the palm and the elbow position of the human handing the object are tracked and constantly fed into a LSTM network that generates, at each time step, the reference value for each joint. In this case, no prediction is involved for anticipating the human behavior, but the robot moves accordingly with the latter. If the human stops, the robot stops too.

With the growing interest for neuromorphic and spiking computation, event-based cameras have been used for several tasks characterised by high dynamics such as camera pose relocalisation [28], visual-inertial odometry [49], gesture recognition [167], tracking [38, 168] and trajectory estimation [49]. In this work, the authors aimed at estimating the position of the vision sensor in space, parameterising the trajectory by means of B-splines. Event-based vision is used also for predicting the steering angle of an autonomous vehicle in [27]. Here, classical Deep Learning methods are applied on "frames of events", analysing the performance of the networks under different light conditions and events integration time. In [153] low latency is exploited for high-speed quadrotors mounting

event-based cameras in order to sense potential obstacles and perform avoidance maneuvers.

2.1 Event-based prediction with LSTM networks

This work considers a handover-like task, in which a person is handing an object to the robot and the robot must grasp it at the optimal location and at the right time, in order to achieve smooth Human-Robot Interaction. Focus is on the trajectory prediction, exploiting existing work on tracking [38] and leaving the robot control for future work. To solve the prediction problem, given the current position and past positions of a target, the most likely future trajectory is estimated. The visual pipeline, as shown in Fig. 2.1, consists of the following components:

- The ATIS [4] event-based camera mounted on iCub;
- The particle filter circle tracker [38] that estimates the object trajectory from the raw events outputted by the camera;
- The LSTM-based prediction system that takes the current and past positions of the target and estimates the future points in space and in time.

2.1.1 The event-based camera and tracking system

An output event consists in the classical quadruplet (x_i, y_i, t_i, p_i), containing the spatial location, the timestamp and the polarity of the brightness change. The resolution of the camera here adopted is 304×240 pixels. The event-based camera is integrated into the neuromorphic iCub [5], using the event-driven library [169] integrated with YARP [170] - the middleware used to control iCub.

To track the position of the object (a round paddle) handed to the robot, I applied a known, robust approach [38] tuned to circles that rejects clutter generated by the robot's ego-motion and other moving agents. The position (x_c, y_c) of the target and the associated event timestamp t_c are updated by the tracker every time the target moves of a set amount of pixels, implementing a spatial sampling. A spatial sub-sampling of 3 pixels is chosen, in order to reduce the amount of information without any particular loss of information. The tracker is initialised with a Hough detection algorithm [171] with the assumption that there is only a single paddle in the scene at any given time.

2.1.2 The Encoder-Decoder LSTM architecture

LSTMs are adopted as state-of-the-art Recurrent Neural Networks to solve prediction problems. RNNs fully take advantage of the temporal information associated with the events by exploiting the sequential information coming from queries over time. In this work, it is introduced a LSTM compatible with the asynchronous event data by directly feeding the time as input feature to the network. As the event-based camera is not polled at a fixed-rate, and the tracker follows the same sampling strategy, the time between sample points is not constant, hence the LSTM is asynchronously queried and the input state is (x, y, dt), where (x, y) is the point on the trajectory and dt is the time interval between two consecutive sample points. An Encoder-Decoder architecture is used, consisting of two main components, as shown in Fig. 2.3. The Encoder is fed with the input data and attempts to represent the information with its own state vector in a sort of embedding space. The output of the encoder is then fed into the Decoder. The Decoder is queried and, from its initial state, produces the output sequence associated to the Encoder input. The separation in two sub-networks enables the input sequences of length w_{in} to be different from the output sequence length,

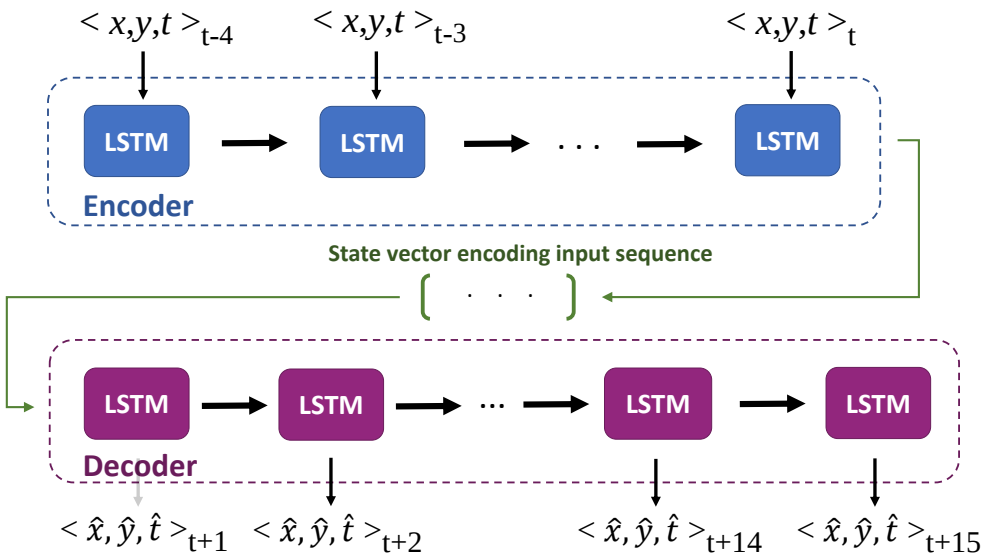


Figure 2.3: The architecture considered consists of an Encoder, that receives an input sequence of length w_{in} and maps the input sequence in a state vector at the end of the process, followed by a Decoder network, that outputs a sequence of length w_{out} after being fed with only the state vector and no external input. Such a decomposition of the model allows for the input and output of sequences with different length.

w_{out}, which becomes the predicted trajectory. Any time the network is queried, it is fed with the most recent w_{in} points, and the predicted trajectory of length w_{out} is obtained.

The spatial sampling of the tracker is 3 pixels, i.e. the tracker sends the location of the target to the input of the LSTM network every time the target position changes of at least 3 pixels. In the following, a single "point" corresponds to a movement of 3 pixels.

The architecture considered has an input layer with 3 neurons, 2 to input the (x, y) coordinates and one for the time interval Δt between the events, 25 neurons for both the Encoder and the Decoder networks, and 3 neurons in the output layer, in order to predict future (x, y) spatial coordinates and the relative estimated times of arrival. To train the model, the full 40 minute recording was

first windowed accordingly to the w_{in} and w_{out} values. Then, the total amount of sequences was shuffled and divided in 70% training and 30% validation. The final test was performed on a third dataset of 60 seconds. The network was trained using the Adam optimisation algorithm with learning rate $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. For regression, the Mean Squared Error (MSE) loss function was adopted. The training lasted for 200 epochs using batches of size 128. No dropout was required.

2.1.3 Prediction baselines

Prediction is, at a basic level, a regression task: from the current and past data available, plus some understanding (i.e. a model) of the system, the future states of the system can be extrapolated. However, regression methods need to be based on a good model of the process and might not scale to more complex scenarios. In this work, the LSTM-based predictor is compared to several hand-tuned regression baselines, with the aim to understand if LSTM are a good approach to the task and, in particular, with the event-based data. From an inspection of the data (Fig. 2.4) it is chosen to compare with a linear, a quadratic, and a mixed-sinusoidal model. Given w_{in} past points (including the current point), each model is fit and used to predict the position future instants and spatial points of the trajectory, always assuming that time increases linearly.

A *linear* model $x(t) = a_{1,x}\ t + a_{0,x}$ and $y(t) = a_{1,y}\ t + a_{0,y}$ is perhaps unsuited to the data, especially at "turning points", but is used as an absolute baseline of prediction. A *quadratic* function $x(t) = a_{2,x}\ t^2 + a_{1,x}\ t + a_{0,x}$ and $y(t) = a_{2,y}t^2 + a_{1,y}\ t + a_{0,y}$ should better model the "turning points" (assuming they are parabolic), and, in general, better model non-linear data. In both the linear and the quadratic regression, time is fit over a linear range, while x and y are time-dependent and fit over the input time. The parameters $a_{i,x}, a_{i,y}$ are

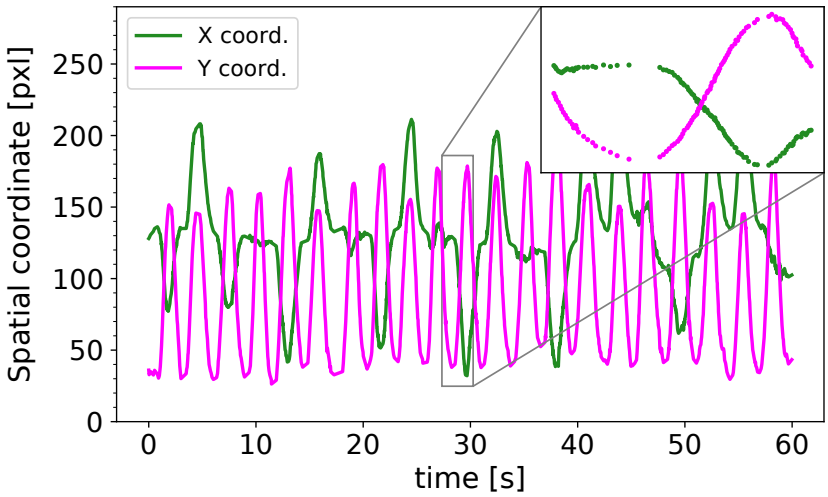


Figure 2.4: A sample of the dataset. One minute (over the total 40) of the tracking algorithm output, as the target object is handed to the robot and then retracted. The inset shows the spatial sampling strategy of event-based data with more samples for fast-motion, and less samples when the target is slowing down.

the coefficients optimised by the fitting procedure. Finally, as the y-axis shows a periodic motion, a *mixed-sinusoidal* model $y(t) = A\sin(2\pi ft + \phi) + B$ is used. In this case, I used a linear model for the temporal axis and the x-axis, as it more accurately predicts the motion along this direction. Again, to correctly fit the data, the value of w_{in} must be large enough to capture at least 1 period of motion: The amplitude A, the frequency f, the phase shift ϕ and the offset B are the parameters optimised with a least squares regression procedure.

Each of the baseline models is recalculated for each prediction point: when the predictor is queried, the past w_{in} points are taken, the model parameters are estimated (with either polynomial or least-squares regression) and the future points are predicted given the model as it is fit to the data. The number of input points, w_{in}, can be as low as 5 for the linear model, but has to be higher in the parabolic model (15 points considered) and in the mixed-sinusoidal model (100

points), in order to better capture the trajectory shape.

2.1.4 The dataset

A dataset of 40 minutes was created as a subject moved the paddle towards and away from the robot, as if periodically handing an object to the robot. The paddle trajectory was always in the field of view of the camera and the output of the tracker (x_c, y_c, t_c) was recorded. As the camera was not moving, the data mostly consists of the paddle and the person moving. However, as the tracker is robust to clutter, the data processed by the predictor is limited to the trajectory of the target itself, excluding the events generated by the person. A 1 minute section of the dataset is showed in Fig. 2.4.

Backward movements, where the human brings back the object away from the robot, were also included in the dataset, as it is important to understand human interactions beyond that of only giving the object. Understanding that the robot should not grasp under such circumstances is equally important.

2.2 LSTM Encoder-Decoder characterisation

System performance are characterised in terms of prediction accuracy for different w_{out} and w_{in}. The model is continuously queried inputting the latest w_{in} points received from the tracker and outputs a sequence of w_{out} points, representing the estimated future positions in the image plane and their temporal distance from the current time. A Sequence-to-Sequence error is defined for measuring how much the predicted sequence overlaps with the ground truth. Prediction error - obtained as average error over the entire dataset - is measured separating space and time, as described in Fig. 2.5.

The error increases linearly with w_{out}, i.e. further into the future, as shown in Fig. 2.6a. As such there is no optimal output value to use, and instead the

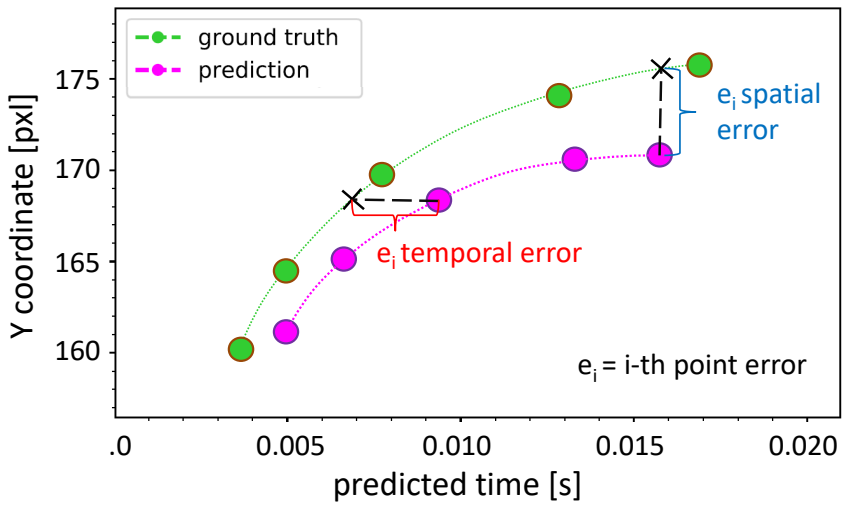
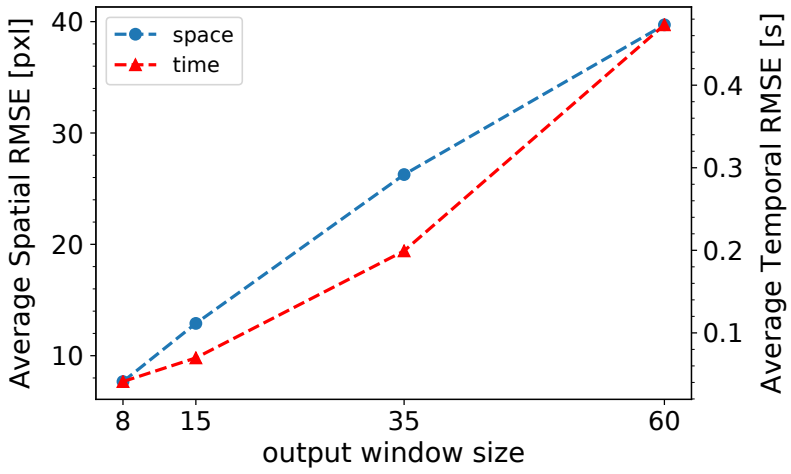


Figure 2.5: The error is measured between the predicted trajectory and the ground-truth as a spatial error (at a fixed time) and a temporal error (at a fixed location).

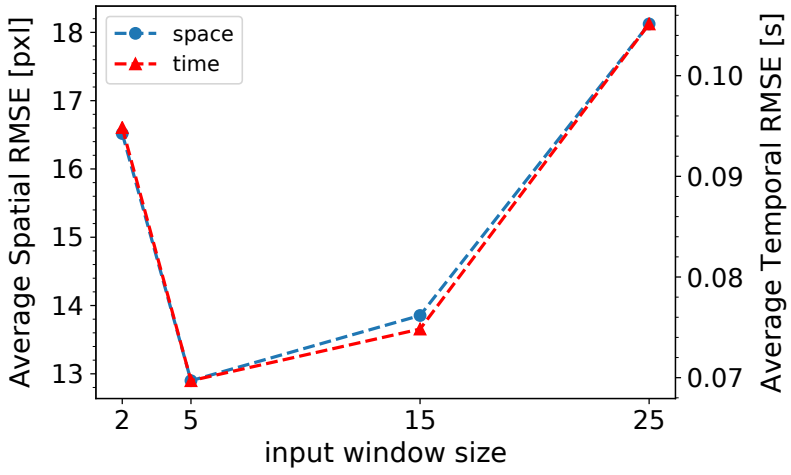
value can be chosen to trade-off the acceptable error and the required prediction look-ahead. Fig. 2.6b shows that the length of the input sequence, w_{in} has an optimal point at 5 points (a movement of 15 pixels), as too few points do not contain enough information and too many points lead to an increased error.

2.3 Network results and baseline comparison

The LSTM performance are compared against all regression models. The parameters for the Encoder-Decoder were set to be $w_{in} = 5$ (15 pixels) and $w_{out} = 15$ (45 pixels), as it is approximately one third of the maximum swing along the y-axis (as shown in Fig. 2.4), offering a good compromise between prediction window and prediction error. Fig. 2.7 shows example predictions for the LSTM and all regression baselines. The linear model performs well in linear sections of the dataset, but, as expected, cannot correctly fit the "turning-points" present in the data. The quadratic model can better fit the "turning-points" - where the



(a)



(b)

Figure 2.6: Prediction error for (a) different w_{out} values assuming $w_{in} = 5$ points and (b) different w_{in} values assuming $w_{out} = 35$ points. The error is separated in spatial component (red) and temporal component (blue).

slope changes - but the prediction diverges from the true trajectory in presence of small sources of noise in the linear section. Finally, the sinusoidal model can fit both the linear section and "turning-points", but it requires a long w_{in} to estimate the frequency. It is not sensitive to small, local, variations - for exam-

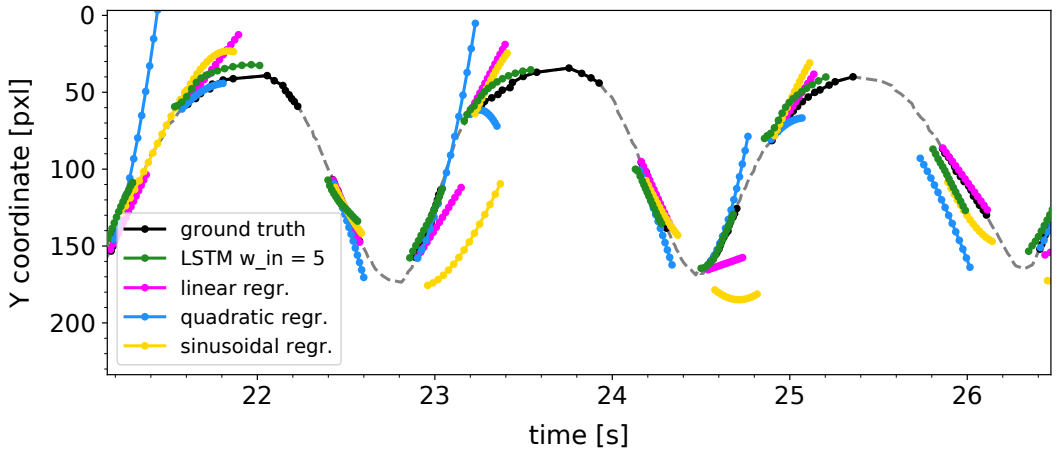


Figure 2.7: Example predictions in linear and non-linear segments of the trajectory for all methods.

ple, the starting point of the prediction is often not at the current position of the target because the best fit model to the long history does not intersect the current point. The LSTM is not a perfect predictor, but it is able to predict both linear and non-linear sections of the trajectory and might scale to more complex trajectories.

Fig. 2.8a summarises the results, showing the prediction error at different points of the w_{out} sequence, averaged over the whole dataset. The linear regression presents a low error for the first 10 points but degrades quickly, as it is not complex enough to represent the data. The quadratic regression achieves better results, but it is still unable to match the LSTM performance. The sinusoidal model is the second best fit for the last points, proving that the Y coordinate exhibits a periodic pattern. The large initial error is due to the fact that I am forcing the prediction to have a periodic behavior. The model therefore performs worse at local predictions, but better on the overall trajectory. Fig. 2.8b shows the analysis performed above, but only for the linear parts of the trajectories, where the linear regression and the LSTM perform best. As the sinusoidal model does

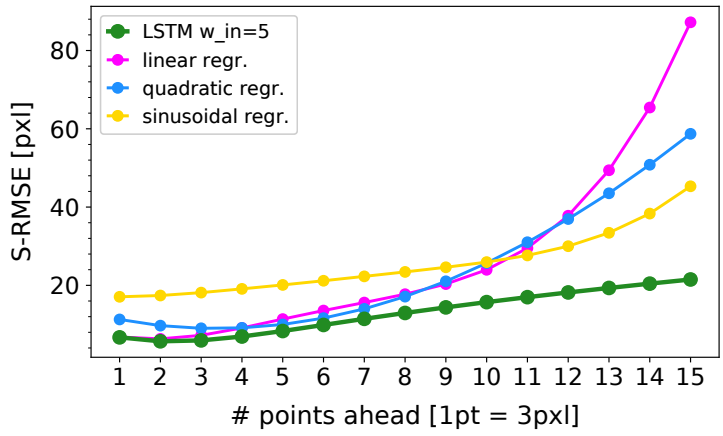
not always pass through the current point, it is often offset by a small amount, despite estimating a correct trajectory shape. Fig. 2.8c shows the analysis for the non-linear segments of the trajectories, confirming the observation that all the predictors, except for the linear one, can achieve good results, with the LSTM showing better accuracy for long-term predictions overall.

2.3.1 Prediction of direction change

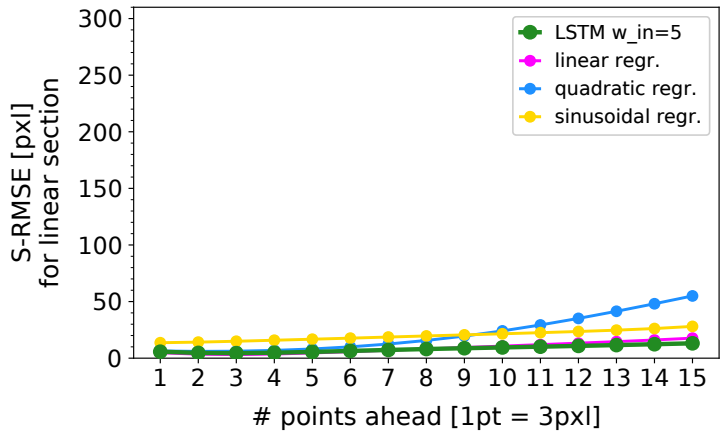
When attempting to predict the "turning-point" of a trajectory, it is not directly known when the subject will decide to change direction. It is not possible, therefore, to expect the LSTM to predict perfectly. While some cues can be used, such as decrease in velocity, the system can only predict the "turning-point" after some observations that it is occurring. What is important therefore is how quickly the model corrects given only a small amount of evidence. Fig. 2.9 shows the error statistics for some points before and some points after a turn, and indicates that within 2 points after a motion change, the LSTM trajectory error is within 20 pixels, - a typical value for the model in general.

2.3.2 Dynamics-driven prediction horizon

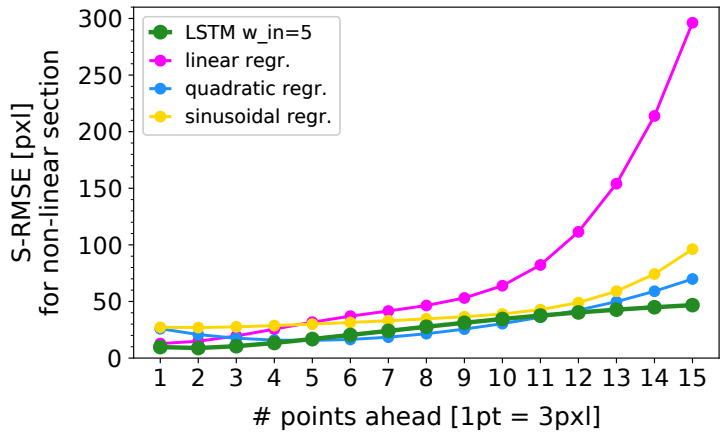
Given the asynchronous nature of the data, inputting and outputting a fixed number of points means that the prediction horizon is not fixed, but changes according to the dynamics of the source of the events. Fig. 2.10a shows how two different input sequences with different velocity produce two output sequences with different duration. This is particularly useful considering that for fast movements it is more interesting to have a dense, short-term prediction of the trajectory, more than a longer-term prediction which can change anytime. On the other hand, slower movements allow for longer-term prediction as their dynamics is less variable. In Fig. 2.10b, instead, the case of a turning point is showed. Predicting



(a)



(b)



(c)

Figure 2.8: Prediction error for all methods: (a) for the entire test sequence, and split into (b) linear-only, and (c) non-linear-only segments.

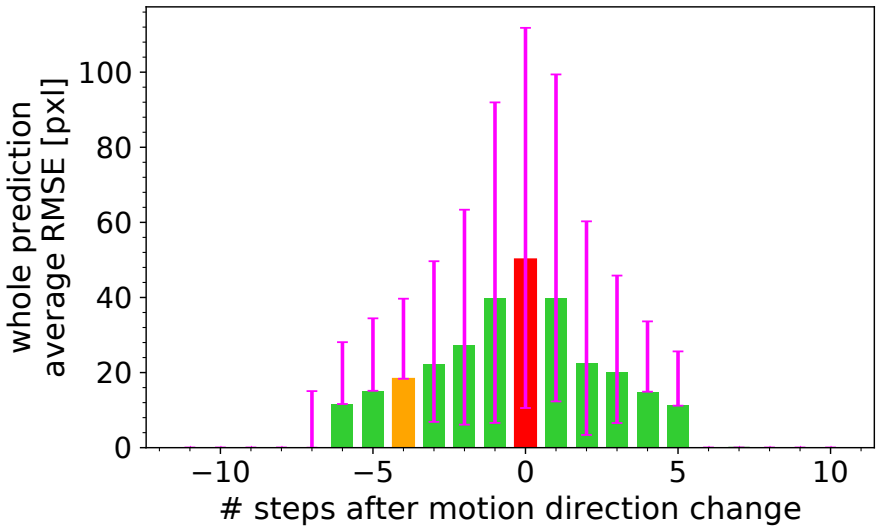
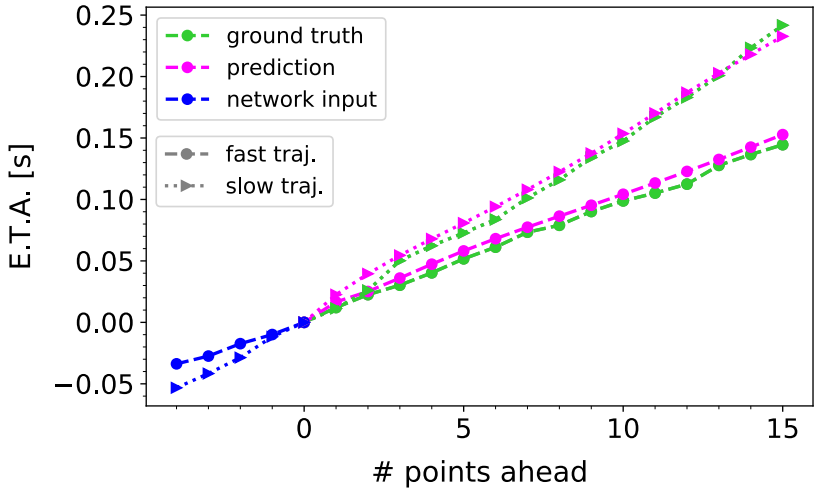


Figure 2.9: Error around a direction change point showing 25-50-75 error percentiles. As the point of direction change is "unpredictable", it is important to understand how fast the network can correct for the error after the change is detected. It can be seen as, in most of the cases, at maximum five points are needed to obtain a prediction with average RMSE lower than 10 pixels.

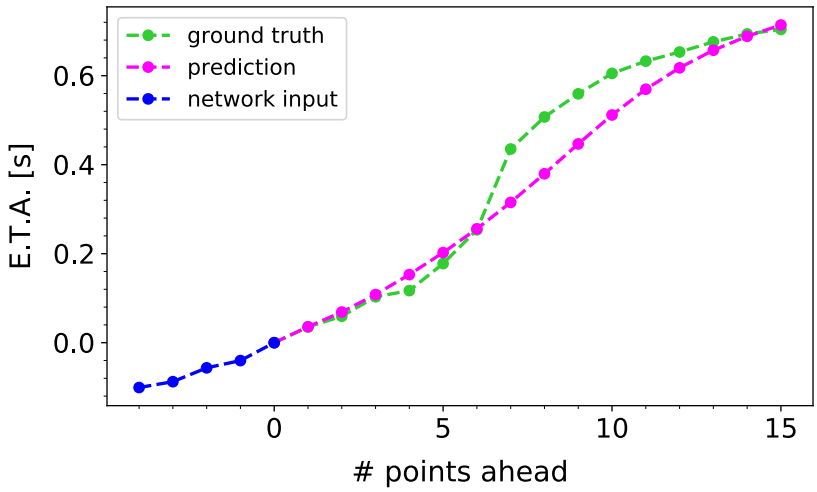
the exact time instants - and spatial coordinates - of the incoming trajectory is difficult. Nevertheless, after few input points showing that the motion dynamics is about to change, with bigger time intervals between consecutive points, the network can predict the correct temporal horizon of the prediction. As mentioned before, maximum 5 input points are sufficient in general to recover from the error.

2.3.3 Computational time

Table 2.1 reports the time needed to get a prediction from the Encoder-Decoder model for different pairs of input and output lengths. The inference time of the LSTM increases with the parameters of w_{in} and w_{out} and may affect design decisions depending on the dynamics of the task, e.g. how much time is available for prediction, and how fast objects are moving. The values tested above, $w_{in} = 5$



(a)



(b)

Figure 2.10: The motion dynamics influences the prediction horizon. In (a) it is visible how movements with different speeds produce predictions with different temporal horizons; in (b) it is showed how, at a turning point of the trajectory, the correct time interval is predicted as soon as few points are fed in input - even though the single points do not match exactly the ground-truth.

and $w_{out} = 15$, result in an update frequency of 500 Hz - when considering pre-processing and detection time, this drops to 250 Hz. Using an event-based camera enables a visual input at this rate, also enabling a trajectory prediction at 250 Hz. If using a frame-based camera, the algorithm would be limited by the sensor rate (about 30 Hz) or tracking algorithm rate and not by the prediction algorithm.

		w_{out}			
		8	**15**	**35**	**60**
w_{in}	**2**	1.7	2.0	3.5	5.0
	5	1.8	2.0	3.5	5.0
	15	2.3	2.5	3.8	5.7
	25	2.5	3.2	4.3	6.3

Table 2.1: Computational time for different w_{in} and w_{out}, measured in milliseconds [ms]. The average pre-processing is 2 ms which should be added to these values.

2.4 Conclusions

This work presents the first steps toward an event-based framework for Human-Robot Interaction on the neuromorphic iCub. Event-based and neuromorphic sensing and processing can lead to low-latency, low-computation processing for robots, but the problem of finding the best tools and techniques that fit the shift in sensing paradigm is still open. In this chapter I compared a LSTM learning architecture to parameterised models for regression, for asynchronous prediction of event-based trajectories. The only model that could achieve similar results as the Encoder-Decoder implemented is the mixed-sinusoidal model. It could be tuned to better fit the data by optimising w_{in} and introducing a method to more closely fit more recent data compared to past data. However, the method in general is

highly specific to the data and therefore does not generalise well, even to similar handover tasks, to motions that do not follow the periodic trend. In addition, the linear models for time and x-axis were selected post hoc, after visualising the data. While the LSTM still requires data to learn, it can adapt to a much wider variety of trajectory types and scales much better to more complex problems. It is showed that the Encoder-Decoder is a good choice to learn to predict object trajectories for HRI tasks, but also that it can run as fast as 250 Hz, thanks to the event-based technology. In particular, the neural network manages to predict linear and non-linear segments of the data with equal success, requiring many less points in history to make its prediction. While some parts of the data are *unpredictable*, the LSTM is able to quickly correct itself given new evidence of the true trajectory. However, other learning methods exist to which LSTM could be compared. I have avoided a CNN, which is the standard state-of-the-art Deep Learning technique, as the data does not lend itself to convolutions, but rather is a sequence, and thus more suited to the LSTM model. The focus is to take the Encoder-Decoder network as the state-of-the-art in learning methods for data sequences for initial investigation, and integrate improvements once the method is validated as a reasonable choice.

To this purpose, it has been already discussed that event-based camera and subsequent event-driven tracking techniques produce data which is spatially sampled, rather than temporally sampled, and therefore presents substantially different information in the sequence itself. For this reason, next chapter introduces a second test case for this pipeline and a deeper analysis of the motion-driven sampling performed by event-based cameras with respect to the standard time-driven one, to further legitimate the choice of this approach to the class of tasks considered.

Chapter 3

Exploiting Event-based Cameras for Spatio-Temporal Prediction of Fast-Changing Trajectories

Last chapter has demonstrated how events data coming from event-based cameras can be combined with Long Short-Term Memory networks for predicting trajectories in an asynchronous fashion, adapting the horizon of the prediction to the dynamics of the motion in the scene. Data-driven methods like neural networks, moreover, even requiring big amounts of data in their *supervised learning* setting, provide more robustness to noise and generalisation compared to model-based regression methods. The combination with event-based cameras allows also to receive new points at much higher rate - more than 250 Hz compared to the fixed device-dependent usual 30 Hz - making faster the detection of abrupt unpredictable changes and the consequent correction of the prediction.

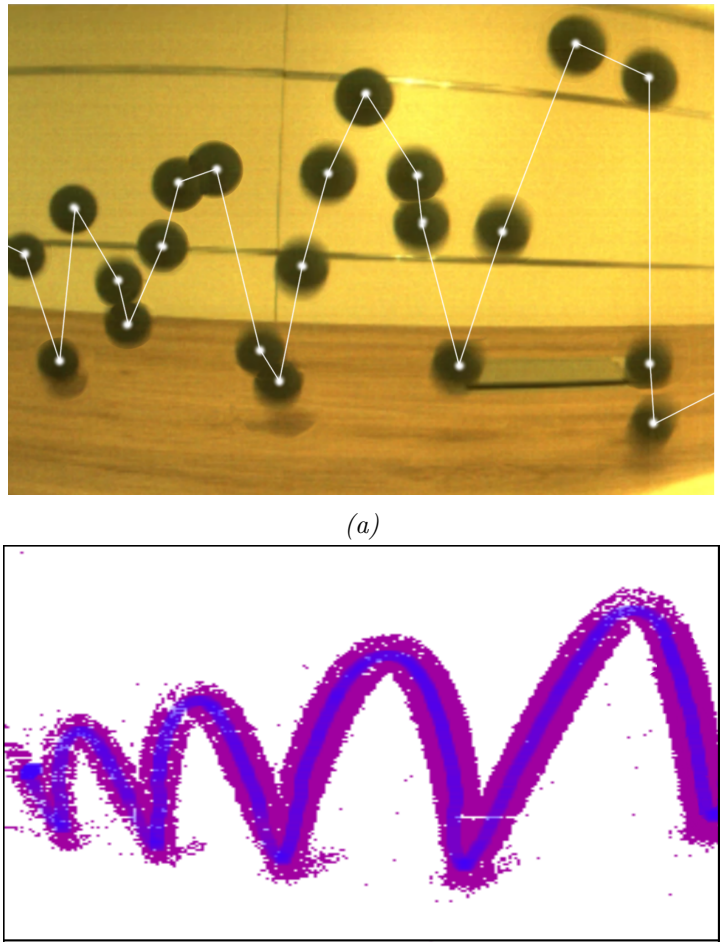
During the experiments and evaluation phase, however, I realised that the previous task, the handover-like task, does not perfectly highlights the advantages of such a event-based framework in a robotic application. The Human-Robot Interaction, even requiring a fast response and anticipation capabilities from the robot, it is not a complex movement with a fast dynamics. In addition, the lack

of a well established method for depth estimation in the event-based literature makes an eventual implementation of the previous framework to the 3D (or even 6D) Cartesian space of the robot highly prone to noise and errors.

In this chapter, for the above mentioned reasons, a step forward is carried out considering the trajectory prediction of a bouncing ball. This kind of motion presents dynamics changes like the slowdown of the ball in the air due to gravity and the sudden bounce when hitting the table. Aim of this work is the application of the framework tested in the previous chapter and the further evaluation of the motion-driven spatial sampling with respect to the standard time-driven sampling.

One of the biggest conceptual differences when compared to a traditional camera, indeed, is that an event-based cameras samples a target trajectory based on its change in spatial value, i.e. the sampling of the vision signal is driven by the change in position. It is therefore achieved a trajectory with samples at pixel - or sub-pixel - resolution, independently of the speed of the target. A comparison to a traditional camera, which samples at a fixed time period, is shown in Fig. 3.1.

In this chapter I investigate the integration of event-data with the LSTM architecture for learning to predict the trajectory of a bouncing ball, focusing on the effect of different sampling strategies when combined with the learning architecture: *fixed-rate* sampling, as performed in traditional cameras, compared with *spatial* sampling, as is associated with event-based cameras. Experiments are performed using the neuromorphic iCub robot [5] equipped with the ATIS [4] event-based camera. The ball bounces on a table in front of the robot and the robot, using the observed past ball positions, predicts the future trajectory for a given sequence. The ultimate goal of such a pipeline is for the robot to decide the best interception strategy - like position, orientation and velocity - with its



(a)

(b)

Figure 3.1: The trajectory of a bouncing ball comparing the sample points for (a) a traditional camera that samples at a fixed-rate in time, and (b) an event-based cameras that samples with a fixed spatial sampling. The event-based cameras output is shown with all the pixel-wise events (in purple) as well as the output of the trajectory tracker (in blue) used in this work.

hand in order to catch the ball. In this case, the temporal component, as well as the spatial positions, become important information, as the robot must select both the space and time at which the interception must occur. Predicting both allows the robot to know how much time is left before the end of the task, and plan and execute the most appropriate movement. In this study I focus on the prediction-only, leaving the control aspect to future development.

3.1 The pipeline

The system is comprised of the following components:

- The ATIS event-based cameras on the iCub robot;
- A target tracker that converts raw pixel events to the center of mass of the target along the trajectory;
- The trajectory sub-sampling strategy: either using a *fixed-rate* sample or *spatial* strategy;
- An LSTM network, trained to predict the future trajectory of the target.

3.1.1 The ATIS camera

Traditional cameras use a *fixed-rate* sampling strategy in which all the pixels are queried for the light intensity at fixed intervals (e.g. 30 Hz). Therefore the trajectory of a target viewed with a traditional camera will correspond to different spatial positions at known time intervals, which leads to a trajectory such as in Fig. 3.1a. For fast moving objects, fixed-rate sampling can lead to gaps in the trajectory as it is under-sampled, or motion blur. In event-based cameras, instead, each pixel is independent. Therefore the readout is not bound by the time it takes to read the entire array. Pixels can emit events with a very short interval between them. The effect is that a moving target will trigger each and every pixel, consecutively, along its trajectory. The positions of target observation are triggered by a *spatial* threshold (a change in pixel position), and the time at which a new observation occurs is *asynchronous*. The same cameras of the previous chapter are used, hence the resolution is again 304×240 pixels.

3.1.2 Target tracking

To simplify the input to the prediction system, the raw data from the ATIS is processed to extract the mean position of the target at any point in time. The target tracking is designed to reduced the effect of noise that could occur if the visual data is directly fed into the learning system, and focus on the effect of sampling strategies on the prediction. In more complex situations, in which the target is not the only part of the visual stimulus, a tracker would be needed as well to isolate the source of information from the background noise.

In this work I did not rely on the tracker proposed in [38] and used in the previous chapter. The initial deteIn this work I did not rely on the tracker proposed in [38] and used in the previous chapter. The initial detection, using the Hough transform for circles, is not fast enough for the trajectories considered and would affect the point of the prediction to infer the incoming trajectory as soon as possible.

ction, using the Hough transform for circles, is not fast enough for the trajectories considered and would affect the point of the prediction to infer the incoming trajectory as soon as possible. Instead, I implemented a tracker with the assumption of a single moving target, as shown in Fig. 3.1b. Similarly to [35], the tracker accumulates events within a region-of-interest (ROI) of size R, and then updates the position of the ROI based on the average position of events within it. As the events move along the trajectory, so it does their mean position and the centre of the ROI. The tracker position is taken as the new centre of the ROI and is outputted along with the timestamp: $(x_{b_i}, y_{b_i}, t_{b_i})$. Initialisation is performed by setting the ROI size R equal to the entire image plane on the first update, and outputting the first point only if the events collected show a standard deviation lower than a certain user-defined threshold. After this first step, a fixed R value is set.Working on the events received directly from the camera, the tracker is not

robust to distractors like would be the one previously used. If a second object moves across the ROI of the tracked one, indeed, the focus on its center-of-mass might be (even temporarily) lost.

3.1.3 Sub-sampling strategy

The high temporal resolution of the event-based cameras allows the trajectory to be sampled for every incoming event. The raw tracker data can therefore sample at over $1kHz$, but (as proposed in this paper) this raw signal can be sub-sampled to improve efficiency without compromising accuracy. Two sub-sampling strategies are proposed: in the *fixed-rate* sampling I sample the signal at fixed time intervals, taking each sample after F milliseconds, in a standard frame-based fashion; in the *spatial* sampling I sample the signal when the spatial coordinates change by a fixed distance, taking each sample every D pixels. The event-based cameras enables very high sampling rates for both *fixed-rate* and *spatial* strategies. Note that the *fixed-rate* strategy is more similar to a standard camera

3.1.4 The Encoder-Decoder LSTM architecture

The model used in this work is a Sequence-to-Sequence LSTM architecture [120] consisting of two separate sub-networks. The Encoder takes the input data and encodes the information in a summary vector, represented by the state of the network after the input has been fed. This output becomes the initial state vector of the Decoder, that is queried to extract the information and produce the output sequence. The decoupling between the two stages allows to input sequences with a length w_{in} different from the desired output sequence length w_{out}. This can be useful in tasks like machine translation, or where the input and output physical domains differ, possibly requiring different representations. Commonly, RNNs

are queried synchronously, limited by the working rate of devices like frame-based cameras. In our case, thanks to the event-based cameras nature, the network can be queried asynchronously every time a new event is produced by the tracker. In such a way, not only it is exploited the information coming from previous queries, but also the temporal information contained within the events. To do so, the network is fed with a sequence comprising the spatial coordinates of the trajectory in the camera space and the time interval between the events in the sequence. The final outcome of the network is a prediction of the next w_{out} points in space and time, which is continuously updated by querying the network for each new input. The prediction consists of future spatial coordinates and, for each point, its estimated time of arrival from the current instant. This spatio-temporal trajectory output could be helpful in robotics, allowing for accurate action planning, considering both spatial and temporal information.

The Sequence-to-Sequence model is firstly fed with the whole input sequence of w_{in} points and, only then, the output sequence of w_{out} point is generated at the same time.

The Encoder architecture considered has an input layer with 3 neurons, in order to accomodate the (X, Y) spatial coordinates and the temporal interval ΔT occurred from the previous input, and an output layer of 25 neurons. The Decoder architecture, on the contrary, consists of an input layer of 25 neurons to be initialised with the final Encoder state vector, and an output layer of 3 neurons outputting the future spatial position and its time of arrival. The Adam optimisation algorithm [172] was used with a learning rate of 0.01. Dealing with a regression problem, the Root Mean Squared Error was adopted as loss function. No dropout was used and the networks were trained for 200 epochs on batches of size 128.

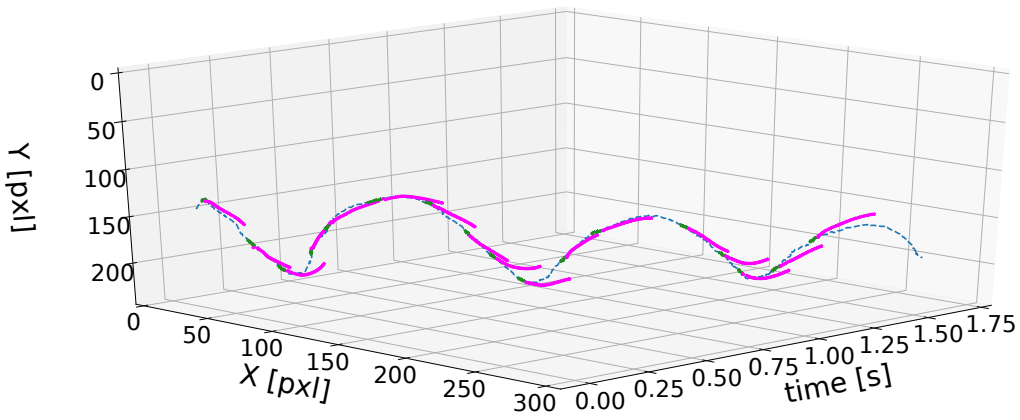


Figure 3.2: Complete example of a ball trajectory (in blue) with pairs of input (in green) and prediction (in purple) at different time instants.

3.1.5 The dataset

A dataset was collected with 250 individual trajectories of the ball thrown in front of the camera and bouncing from one to three times. The $(x_{b_i}, y_{b_i}, t_{b_i})$ output of the tracker module was recorded. To generalise with respect to the trajectory direction, all datasets were flipped along the x-axis resulting in a total of 500 trajectories. The dataset was split into 470 training trajectories, 20 validation trajectories and 10 test trajectories. A trajectory example is shown in space-time in Fig. 3.2.

3.2 LSTM Encoder-Decoder characterisation

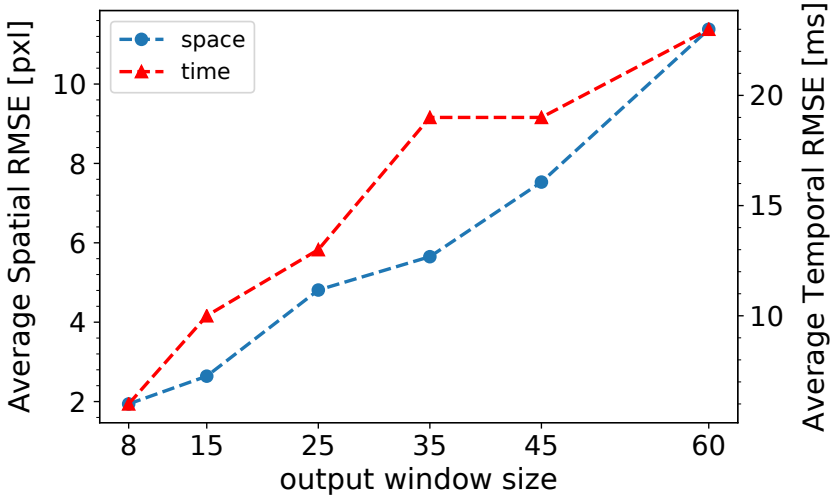
To choose the right Encoder-Decoder architecture, a cross-validation procedure is needed. Starting with a spatial sub-sampling of $D = 2$ pixels, several models were trained assuming different values for w_{in} and w_{out}.

The prediction RMSE is calculated for all the trajectories, according to the trajectory *spatial error* and *temporal error* defined in Sec. 2.2 and showed in Fig. 2.5. The *spatial error* for a single trajectory point is defined as the difference

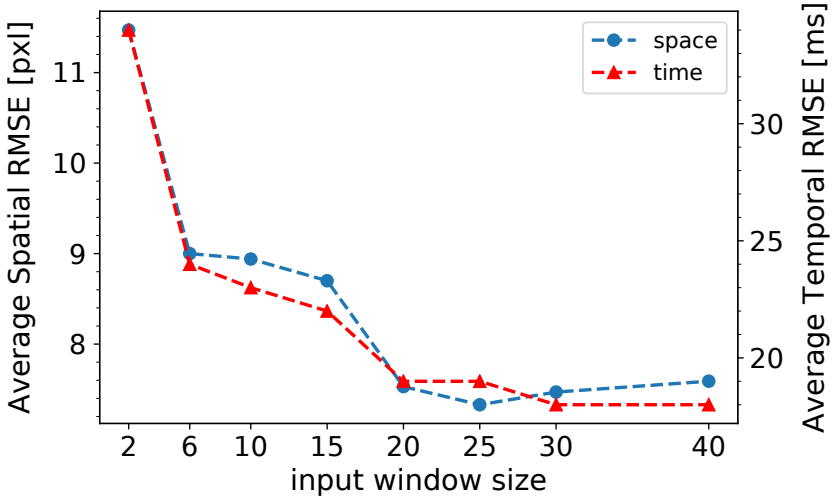
between the estimated pixel coordinate $(\hat{x_{b_i}}, \hat{y_{b_i}})$ and the ground truth coordinates (x_{b_i}, y_{b_i}) at a fixed point in time. In order to perform a comparison, the ground-truth trajectory is interpolated to calculate the position at exact time of a predicted point t_{b_i}. On the other hand, the *temporal error* for a point $(\hat{x_{b_i}}, \hat{y_{b_i}})$ represents the difference between its predicted time of occurrence t_{b_i} and the time when the ball actually reached the same position t_{b_i}. Again the ground-truth trajectory is interpolated and the closest point on the trajectory is considered. The RMSE is the mean error along the entire predicted length, and not only the final position.

The error increases linearly with the output window length w_{out}, meant as number of points predicted in a single network interrogation, for both the spatial error and the temporal error, as shown in Fig. 3.3a. As the network tries to predict further ahead, the accuracy at which it can do it decreases. Since the relationship is linear, there is no optimal point at which the network operates. The choice is therefore application specific, trading off the amount of time ahead that is needed, compared to the highest acceptable error. The spatial error decreases the more data in the past is used, w_{in}. However, after a certain value, there is no real benefit in feeding more points, as shown in Fig. 3.3b. In this case, a systematic operating point could be chosen to achieve the lowest error. In the following, the output window length w_{out} was set to be 45 and the input window length w_{in} was set to 20.

Figure 3.2 shows a qualitative evaluation of the predictor capabilities over the entire trajectory of a single ball bounce. On the complete trajectory (in blue) recorded from the ATIS camera, are shown the input points (in green) and the respective prediction (in purple) obtained from the Encoder-Decoder model at different time instants. The network is continuously queried with new inputs,



(a)



(b)

Figure 3.3: Prediction error for (a) different w_{out} values assuming $w_{in} = 5$ points and (b) different w_{in} values assuming $w_{out} = 35$ points. The error is separated in spatial component (red) and temporal component (blue).

providing an updated estimate every time. This, along with the low latency guaranteed by the event-based cameras, allows for a rapidly adaptive predictive system.

3.3 Sampling strategy comparison

A comparison between sampling strategies is then made with respect to the prediction accuracy for the test set. Considering an input window of about ∼90 ms and an output window of about ∼200 ms - defined by the previous windows length choice - I compare the performance of the two sub-sampling strategies on the same time intervals.

Suitable F values for the *fixed-rate* sampling and D values for the *spatial* sampling are chosen to cover the same time intervals. Given the asynchronous nature of event-based data, it is not possible to precisely say what is the corresponding temporal interval spanned by, to say, 5 points with a spatial delta $D = 4$ pixels. This value can be computed only on average. For this experiments, the mapping between the applied D value and the average sampling rate (+ standard deviation) is reported in Tab. 3.1. This means that the output window of $200ms$, for instance, can be represented by 6 points in the case of time-driven sampling at 33 Hz, and by 36 points in the case of spatial sampling with $D = 2$ pixels.

Fig. 3.4a shows the error for both sampling strategies. In order to perform the comparison, the mean rate of the spatial sampling is calculated and used as a comparison point to the fixed-rate sampling. Fixed-rate sampling showed an inverse relationship to error. A low rate results in a high error and, as the rate increases, the error decreases too. However, the benefit of increased sample rate beyond 67 Hz is minimal.

Spatial sampling, instead, shows a more linear trend in the error with respect to the sampling rate. The error reaches a minimum value of 7.52 pixels for a

Spatial delta D (pxl)	Mean+STD rate [Hz]
-	357±88
2	182±44
4	92±22
6	63±14
8	48±11
10	39±9
12	33±7

Table 3.1: Spatial delta sampling values adopted in this work with the corresponding computational rate (mean and standard deviation).

spatial of 2 pixels, compared to a value of 10.08 pixels when using a fixed 200 Hz sampling rate. Importantly, the lower error is also achieved with less processing requirements as the average rate - and hence the total number of calculations - is lower for the 2 pixel spatial sampling.

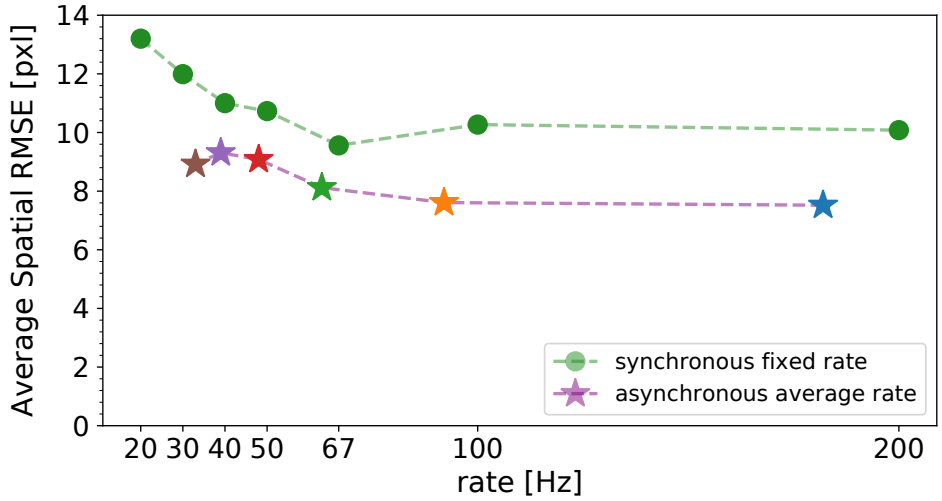
The reason that the spatial sampling can give a lower error for an identical mean sampling rate is that it gives a varying sample rate in time, over the length of the dataset. In periods of fast and non-linear motion, where fine details are required to correctly trace the motion, the sample rate increases also enabling a more accurate prediction. On the other hand, when the target is moving slowly, the sample rate decreases and processing savings can be achieved. Fig. 3.4b shows an example sample rate (in time) for the spatial sampling, and it can be seen that for larger spatial deltas the sample rate becomes more constant across the dataset (a flatter profile). For sample rates without much variation, the error is much closer to that of fixed-rate sampling, as shown in Fig. 3.4a.

3.4 Conclusions

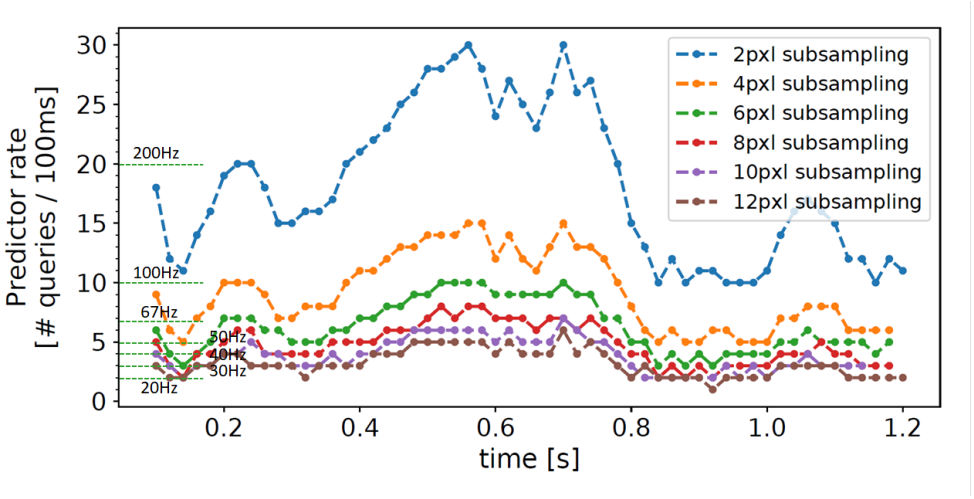
In these experiments I showed that the Encoder-Decoder LSTM network introduced in the previous chapter can be adapted for a bouncing ball task, with only a small input window required to achieve a low error. On the other hand, the output window can be extended for as long as necessary, with the trade-off that the error increases linearly. Asynchronous motion-driven *spatial* sampling outperforms *fixed-rate* sampling both in terms of accuracy and computational load, showing the advantages of the first approach in input data using event-based cameras and in the following visual processing, using asynchronous output events also in computational models such as the tracking.

I believe that such a strategy, combined with event-based cameras, is a promising approach for learning to predict trajectories, motivating a further quest in the use of event-based perception coupled with LSTM architectures for fast vision tasks in artificial intelligence and robotics.

Depending on the exact constraints of the task, however, other solutions might be to have the network learn different goals, such as only the final position and time of a single trajectory rather than the full path of the trajectory, allowing the robot to intercept the ball with the correct motion timing. This is the aim of the ongoing work, exposed in the next chapter.



(a)



(b)

Figure 3.4: The (a) spatial RMSE for fixed-rate and spatial sampling, in which the mean sample rate of time is used to place the spatial sampling, and (b) the sample rate for spatial sampling over time for an example trajectory.

Chapter 4

Ongoing Work: End-Point Prediction of a Bouncing Ball

The prediction of the trajectory of the bouncing ball obtained in the previous chapter is instrumental to plan the control of the robot to catch the ball. When considering this task, we realised that the dense and continuous prediction of a segment of the trajectory right after the observed points is less relevant than the prediction of the position (in time and space) of the ball at a later point. This future point can have an operative description, for example it can be defined as the position and time of the target exiting the robot's field-of-view, or of the target entering into the operational space of the robot. I therefore implemented a different LSTM architecture that, given a trajectory of the center of mass of the target, can update continuously the predicted trajectory end-point, defined as the position at which the ball exits the robot's FOV.

The aim of the prediction is now to estimate at which pixel coordinates the ball will be out of the camera and when, meant as the time remaining from the current position update before it disappears. This will be useful in the future for planning the appropriate direction to approach the ball and catch it, but also the action speed.

The predictor considered is not anymore an Encoder-Decoder architecture, but still a many-to-many framework, a synchronous one, as described in 1.3.2. Every time a new input point is received, an updated estimate of the end-point is outputted. However, tracking and inputting now the whole trajectory - and not only the latest N points received - raises the question of how many time steps should perform the recursion the LSTM at run-time, being this parameter usually fixed at training time. If I choose a value too short, the network could lose the information from the last tracked point; if the value is too long, the pipeline might start to receive points not related to the actual trajectory but to noise, outputting misleading prediction. Or, it could wait an indefinite time before resetting its memory. For these reasons, I resorted to a property of the LSTM which is called *statefulness*. Setting this property on when the model is created, gives manual control to the user about when to reset the memory of the network, which is otherwise kept forever. Not resetting the memory can be an issue at training time for very long sequences, as the BPTT is not able to match the long-term dependencies for such samples. Ongoing experiments I am carrying out, however, show that this is not a problem in the specific case, and results seem promising.

With the prediction of the end-point working fine, the focus moves on the robot control side. At the moment I am dealing with timing and precision issues in the iCub arm control, given also the limited operative space guaranteed by the arm's length of the robot. Due to time constraints in the YARP middleware functions, continuous control using the predicted position results in a chattering motion, with the hand also wobbling due to the tendon-based control and single-screw joints.

For all these reasons, I am currently implementing the same pipeline on a

Franka Panda robot [173] (Fig. 4.1) present in our lab. To endow the robot with an event-based camera, I am using a custom setup of our group, visible in Fig. 4.2. The setup comprises two ATIS cameras and two frame-based sensors, for ground truth and comparison purposes. The camera is external and does not move with the robot, simplifying the tracking problem.

Quantitative results must be yet carried out but, thanks to the wider operative space and the enhanced performance guaranteed by the manipulator, qualitative tests show promising results - as it can be seen from the following frames taken from a trial (Fig. 4.3).

Each picture shows an instant of the live demonstration, with the ball thrown from right to left. In the bottom-left corner I show a view of the camera and prediction output:

- The green-yellow-purple trail is a collection of all the events generated by the bouncing ball, that show its full (dense) trajectory. In green negative polarity events (leading edge of a dark ball moving on a light background), in purple the positive polarity events (trailing edge) and in yellow the su-

Figure 4.1: The Franka Emika Panda robot. Image from [173]

Figure 4.2: The custom setup endowed, among other sensors, with event-based cameras and the Panda robot.

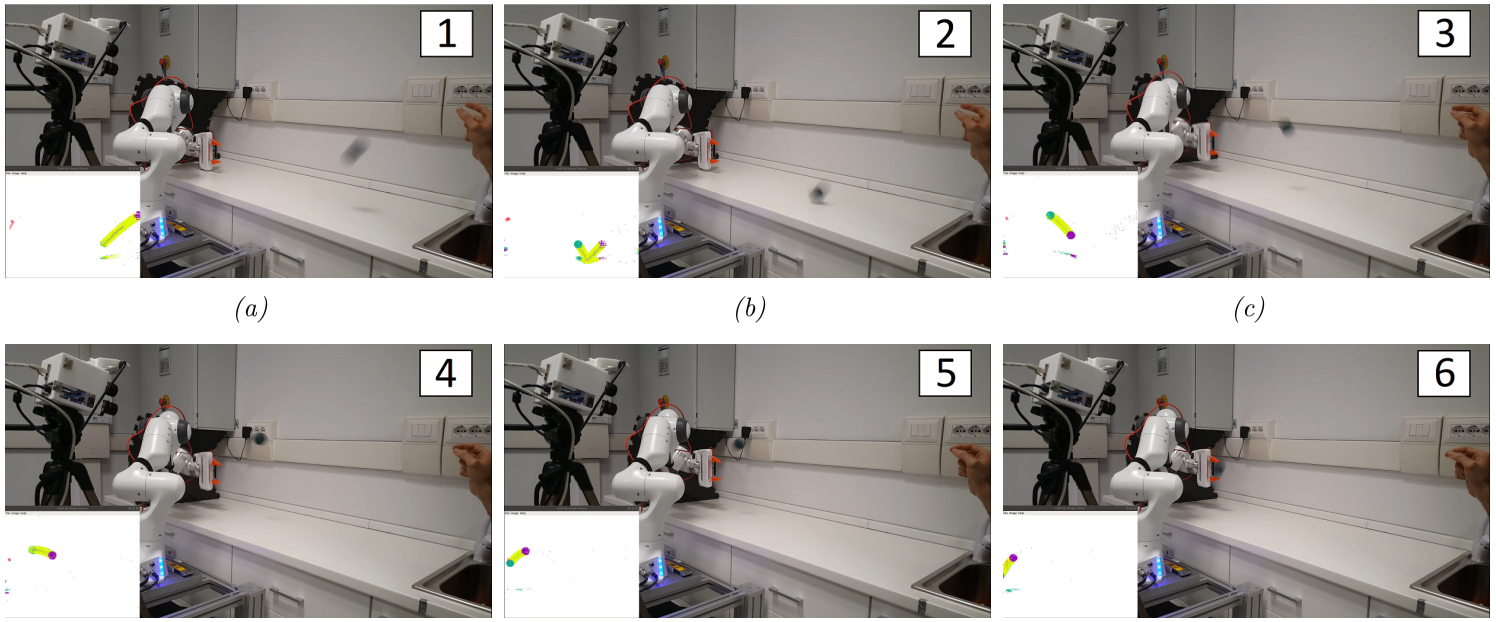


Figure 4.3: Frames of a live demonstration. In the bottom-left corner of each picture it is visible the view from the event-based camera. The green-yellow-purple trail is a collection of all the events generated by the bouncing ball, that show its full trajectory: in green negative polarity events, in purple the positive polarity events and, in yellow, the superposition of the positive and negative events. The blue trail are tracker events, i.e. the position of the center of mass of the ball computed by the tracking. The red trail are prediction events, converging toward the estimated end-point of the trajectory

perposition of the positive and negative events;

- The blue trail overlapped to the yellow trail are tracker events, i.e. the position of the center of mass of the ball computed by the tracking module and used as input to the LSTM network;

- The red trail are prediction events, converging toward the estimated end-point of the trajectory. The output of the LSTM is updated for each incoming event of the center of mass trajectory.

Figs. 4.3a-4.3d show in an extremely narrow temporal window (∼40 ms), (i) that a large amount of events is produced, allowing a very fine spatio-temporal resolution for the tracked, and hence predicted, trajectory, (ii) the strength of the blur effect on the standard frame for such fast movements (present also in the other pictures), and (iii) the spatial dispersion of the predictor events is much narrow than the tracker ones, showing a fast convergence of the prediction and its advantage in giving more time to reach the estimated end-point. Imagining to catch the ball relying only on the tracker, this would mean that the robot either moves a lot without purpose before catching the ball, or it waits the last moments before moving. Fig. 4.3e shows the ball going out of the FOV almost exactly where it was predicted from the very beginning. Very small adjustments, indeed, can be seen in the prediction red points in each picture. Fig. 4.3f, finally, shows the ball hitting the end-effector of the robot.

However, this is not always the case. The robot is not always able to intercept the ball. Beside a prediction error, this might be due also to the fact that fast trajectories last up to ∼0.5 s. Subtracting from this value the time needed for the prediction to converge and the ∼100 ms control delay to compensate, leaves little time for the controller to move the robot fast and precisely - even for a

manipulator like the Panda robot. Further tests are aiming at understanding the roles of the control and the neural network in failed executions, while gathering more live examples and trying to train also on synthetic data, generated with an event-based camera simulator.

Chapter 5

Conclusions

This thesis work presented the implementation of a combined framework of event-based cameras and Long Short-Term Memory networks for trajectory prediction in robotic tasks. I believe, indeed, that event-based cameras, thanks to their sparse, asynchronous output, hold great potential for such a class of tasks. The challenge was to prove that event-based data can fit classical Recurrent Neural Network architectures, outperforming other methods on the task, and being convenient from the sampling and computational point of view.

This challenge was addressed by implementing a pipeline composed of an event-based tracker and an Encoder-Decoder network, state-of-the-art for Sequence-to-Sequence problems. In the first part of this thesis (Chap. 2), a handover-like task is considered to study the feasibility of such a pipeline. Here, exploiting the tracker presented in [38], the object in a person's hand in front of the iCub robot is tracked and its events' position and timestamp are fed to the network, to predict the incoming part of the trajectory. The Encoder-Decoder allows decoupling the input and output sequence lengths of the network, embedding the input sequence in a state vector at the end of the Encoder network, and feeding the latter as unique input to the Decoder for triggering the prediction. After

a cross-validation process to select the optimal input and output lengths, the network output is compared to model-based regression methods for prediction, considering a linear, a quadratic, and a sinusoidal movement. Results showed how a data-driven learning method is preferable for trajectory prediction, given the sensitivity to noise and the need to pick the right model post hoc for regression approaches. A deeper analysis of the network failure cases showed how fast the event-based approach allows recovery from the latter in case of "unpredictable" motion changes. Moreover, again thanks to the event-based nature of the pipeline, the prediction horizon can be adapted to the dynamics of the motion in the scene, providing dense, short-term predictions for fast movements and more reliable longer-term predictions for slower ones. Computational time also is negligible, allowing for fast updates.

The handover-like task, however, does not completely highlight the advantages of such an approach to the robot. This task, indeed, does not present a fast dynamics and further studies on the comparison with a fixed-rate sampling - common to frame-based systems - would not be clear enough.

For this reason, the second part of this thesis (Chap. 3) focused on a more dynamic task like the bouncing ball trajectory prediction. In this work, the main goal is to evaluate the difference between a time-driven fixed-rate sampling (like the one performed by the majority of current sensors) and a motion-driven spatial sampling, both applied on data coming from event-based cameras. To do so, the same pipeline as before is used, substituting the tracker of [38] with a responsive, shape-independent, ROI-based tracker I have implemented. Several models are trained firstly to select the right input and output sequence lengths, and then for comparing the two sampling approaches on the same time intervals. Results showed that spatial sampling is always a better solution, presenting a lower error

for any spatial threshold D adopted in the work. Moreover, the dynamic query rate of the network allows to output predictions only when it is needed, without any computation waste if the scene is changing slowly or not at all. Depending on the task demands, then, one can choose the right spatial threshold to achieve both a certain average error and a certain computational load. The latter can be particularly useful for robots with on-board systems.

5.1 Open questions and future work

Still, a lot of work remains to be carried out. Unanswered questions must be addressed, like the introduction of a depth estimate, the compensation of the ego-motion, or how to provide the prediction capabilities in a faster way, without requiring the acquisition of hundreds - or even thousands - sample trajectories to train the neural network.

As mentioned before in this work, there is no well-assessed method for depth estimate. Researchers are pushing for unsupervised learning methods exploiting Spiking Neural Networks, like it was presented in [32, 40, 41] in Sec. 1.2.2 and Sec. 1.2.3. For the tasks considered in this work, however, the problem is relatively easier. Given the center-of-mass computed by the tracker, the problem could be framed as a stereo vision problem - using a pair of event-based cameras - and triangulate the position of the tracker from both visual fields. Nevertheless, to have a faithful estimate, the tracker must be very robust to noise. This cannot be said at the time and experiments must be carried out.

Ego-motion suppression is a hot topic for the event-based setting. Imagining, indeed, to move the robot to follow the ball with the gaze, a huge amount of events

would be generated from the background, disturbing the tracker implemented in Chap. 3. The tracker in [38] does not suffer this problem but, for the reasons explained in 3.1, cannot be used for the task. A solution might be to use an event-based motion segmentation by motion compensation technique like the one presented in [50], in order to separate the ball from the background thanks to the different relative velocities. Another possibility might be to use the motion predicted direction to dynamically shape the ROI of the tracker so as to capture a higher amount of events coming from the ball with respect to the background generated ones.

The motion of the eyes/head, moreover, would alter the spatial coordinates of the tracked object in the image plane, creating irregular trajectories that would be then fed to the Encoder-Decoder network. Even though this has not been tested yet, I believe the irregularity of the trajectory could represent an issue for the prediction. A possible solution would be to introduce the depth to estimate the 3D ball position in the robot coordinates frame, and feed these to the network.

About the speed-up of the training procedure, acquiring large scale datasets can be a tedious experience, and it is not always feasible. Interesting options that can be evaluated with the robots at hand are Online Learning procedures or Reinforcement Learning (RL). Both, however, suffer many problems when moving to the real world and might need a huge amount of trials before learning for real. An interesting approach that could be tested is trying to bootstrap these methods with a supervised training procedure, like the Encoder-Decoder LSTM showed in this work. A valid alternative could be introducing the physics of the task in the network, like recently showed in [162] and mentioned in 1.4.

After this, two main interesting paths can be pursued for the bigger picture:

the implementation of a spiking LSTM network, still an open problem in the neuromorphic community, and the implementation of a spiking controller for the iCub robot. Both could also possibly run on our group's SpiNNaker [24] board. Toward the implementation of a spiking controller, first steps are already made in a work I contributed to and explained in [174]. Here we present a closed-loop motor controller implemented on a mixed-signal analog/digital neuromorphic processor which emulates a spiking neural network. The latter computes the error between the desired target and the encoder reading of a joint of the robot and, after some preprocessing operations, feeds it into three populations of spiking neurons reproducing a sort of spiking Proportional-Integral-Derivative (PID) controller exploiting the temporal dynamics of the network synapses and neurons. To validate the approach, the neuromorphic motor controller is interfaced with the iCub simulator provided with YARP. Experiments on a step response and a target pursuit task demonstrated the validity of the approach and promising performance for further development.

Publications

M. Monforte, A. Arriandiaga, A. Glover and C. Bartolozzi, "Where and When: Event-Based Spatiotemporal Trajectory Prediction from the iCub's Point-Of-View," 2020 IEEE International Conference on Robotics and Automation (ICRA), Paris, France, 2020, pp. 9521-9527, doi: 10.1109/ICRA40945.2020.9197373.

M. Monforte, A. Arriandiaga, A. Glover and C. Bartolozzi, "Exploiting Event Cameras for Spatio-Temporal Prediction of Fast-Changing Trajectories," 2020 2nd IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS), Genova, Italy, 2020, pp. 108-112, doi: 10.1109/AICAS48895.2020.9073855.

J. Zhao, N. Risi, M. Monforte, C. Bartolozzi, G. Indiveri and E. Donati, "Closed-Loop Spiking Control on a Neuromorphic Processor Implemented on the iCub," in IEEE Journal on Emerging and Selected Topics in Circuits and Systems, vol. 10, no. 4, pp. 546-556, Dec. 2020, doi: 10.1109/JETCAS.2020.3040390.

References

[1] Y. Lecun, Y. Bengio, and G. Hinton, "Deep learning," 2015. 1

[2] 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. 2, 9, 10

[3] C. Brandli, R. Berner, M. Yang, S. C. Liu, and T. Delbruck, "A 240 × 180 130 dB 3 μs latency global shutter spatiotemporal vision sensor," *IEEE J. Solid-State Circuits*, vol. 49, no. 10, pp. 2333–2341, 2014. 2, 9, 10

[4] C. Posch, D. Matolin, and R. Wohlgenannt, "A QVGA 143 dB dynamic range frame-free PWM image sensor with lossless pixel-level video compression and time-domain CDS," *IEEE J. Solid-State Circuits*, vol. 46, no. 1, pp. 259–275, 2011. 2, 5, 9, 10, 51, 55, 71

[5] C. Bartolozzi, F. Rea, C. Clercq, D. B. Fasnacht, G. Indiveri, M. Hofstätter, and G. Metta, "Embedded neuromorphic vision for humanoid robots," *IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. Work.*, pp. 129–135, 2011. 5, 51, 55, 71

[6] T. Gollisch and M. Meister, "Rapid neural coding in the retina with relative spike latencies," *Science (80-.).*, vol. 319, no. 5866, pp. 1108–1111, 2008. [Online]. Available: https://science.sciencemag.org/content/319/5866/1108 6

[7] S. C. Liu and T. Delbruck, "Neuromorphic sensory systems," *Curr. Opin. Neurobiol.*, vol. 20, no. 3, pp. 288–295, 2010. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0959438810000450 6

[8] H. Ramachandran, S. Weber, S. A. Aamir, and E. Chicca, "Neuromorphic circuits for Short-Term Plasticity with recovery control," *Proc. - IEEE Int. Symp. Circuits Syst.*, pp. 858–861, 2014. 6

[9] A. Mortara, "A Pulsed Communication/Computation Framework for Analog VLSI Perceptive Systems," in *Neuromorphic Syst. Eng.*, T. S. Lande, Ed. Boston, MA: Springer US, 2007, pp. 201–215. [Online]. Available: https://doi.org/10.1007/978-0-585-28001-1{_}9 7

[10] H. Kim, S. Leutenegger, and A. J. Davison, "Real-time 3D reconstruction and 6-DoF tracking with an event camera," in *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 9910 LNCS, 2016, pp. 349–364. 9

[11] M. A. Mahowald and C. Mead, "The silicon retina," pp. 76–82, 1991. 9

[12] M. A. Mahowald, "{VLSI} analogs of neuronal visual processing: a synthesis of form and function," vol. 1992, no. May, 1992. 9

[13] P. F. Rüedi, P. Heim, F. Kaess, E. Grenet, F. Heitger, P. Y. Burgi, S. Gyger, and P. Nussbaum, "A 128 × 128 Pixel 120-dB Dynamic-Range Vision-Sensor Chip for Image Contrast and Orientation Extraction," in *IEEE J. Solid-State Circuits*, vol. 38, no. 12, 2003, pp. 2325–2333. 9

[14] K. A. Zaghloul and K. Boahen, "Optic Nerve Signals in a Neuromorphic Chip I: Outer and Inner Retina Models," *IEEE Trans. Biomed. Eng.*, vol. 51, no. 4, pp. 657–666, 2004. 9

[15] P. Lichtsteiner and T. Delbruck, "A 64×64 AER logarithmic temporal derivative silicon retina," in *2005 PhD Res. Microelectron. Electron. - Proceedingsof Conf.*, vol. II, 2005, pp. 406–409. 9

[16] P. Lichtsteiner, C. Posch, and T. Delbruck, "A 128??128 120dB 30mW asynchronous vision sensor that responds to relative intensity change," in *Dig. Tech. Pap. - IEEE Int. Solid-State Circuits Conf.*, 2006. 9

[17] R. Berner, C. Brandli, M. Yang, S. C. Liu, and T. Delbruck, "A 240×180 10mW 12us latency sparse-output vision sensor for mobile applications," in *IEEE Symp. VLSI Circuits, Dig. Tech. Pap.*, 2013. 10

[18] "iniVation — Neuromorphic vision systems." [Online]. Available: https://inivation.com/ 10

[19] "Prophesee - Metavision for Machines," p. 14, 2006. [Online]. Available: https://www.prophesee.ai/https://www.prophesee.ai/resources/PROPHESEE{_}Neuromorphic-event-based-vision{_}2006.pdf 10

[20] "CelePixel — World-leading Neuromorphic Vision System." [Online]. Available: https://www.celepixel.com/#/Home 10

[21] Insightness AG, "Insightness – Sight for your device," 2018. [Online]. Available: https://www.insightness.com/http://www.insightness.com/ 11

[22] Y. Yaffe, N. Levy, E. Soloveichik, S. Derhy, A. Keisar, E. Rozin, L. Artsi Jun-Seok Kim, K. Park, B. Son, Y. Suh, H. Jung, C. Shin, J. Woo, Y. Roh, H. Lee, and H. Ryu, "ICRA'17 Workshop on Event-based Vision; Dynamic Vision Sensor Technical report," Tech. Rep., 2017. [Online]. Available: http://rpg.ifi.uzh.ch/docs/ICRA17workshop/Samsung.pdf 11

[23] "SmartThings Vision — GP-U999GTEEAAC — Samsung AU." [Online]. Available: https://www.samsung.com/au/mobile-iot/smartthings-vision-u999/ 11

[24] S. B. Furber, F. Galluppi, S. Temple, and L. A. Plana, "The SpiNNaker project," *Proc. IEEE*, 2014. 11, 95

[25] M. Davies, N. Srinivasa, T. H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, Y. Liao, C. K. Lin, A. Lines, R. Liu, D. Mathaikutty, S. McCoy, A. Paul, J. Tse, G. Venkataramanan, Y. H. Weng, A. Wild, Y. Yang, and H. Wang, "Loihi: A Neuromorphic Manycore Processor with On-Chip Learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, 2018. 11

[26] S. Moradi, N. Qiao, F. Stefanini, and G. Indiveri, "A Scalable Multicore Architecture with Heterogeneous Memory Structures for Dynamic Neuromorphic Asynchronous Processors (DYNAPs)," *IEEE Trans. Biomed. Circuits Syst.*, 2018. 11

[27] A. I. Maqueda, A. Loquercio, G. Gallego, N. Garcia, and D. Scaramuzza, "Event-Based Vision Meets Deep Learning on Steering Prediction for Self-Driving Cars," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 5419–5427. 12, 18, 54

[28] A. Nguyen, T. T. Do, D. G. Caldwell, and N. G. Tsagarakis, "Real-time 6DOF pose relocalization for event cameras with stacked spatial LSTM networks," in *IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. Work.*, vol. 2019-June, 2019, pp. 1638–1645. 12, 54

[29] A. Z. Zhu, L. Yuan, K. Chaney, and K. Daniilidis, "EV-FlowNet: Self-supervised optical flow estimation for event-based cameras," 2018. 12, 16

[30] A. Mitrokhin, C. Ye, C. Fermuller, Y. Aloimonos, and T. Delbruck, "EV-IMO: Motion Segmentation Dataset and Learning Pipeline for Event Cameras," in *IEEE Int. Conf. Intell. Robot. Syst.*, 2019, pp. 6105–6112. 12

[31] C. Ye, A. Mitrokhin, C. Fermüller, J. A. Yorke, and Y. Aloimonos, "Unsupervised learning of dense optical flow, depth and egomotion from sparse event data," 2018. 12, 17

[32] A. Z. Zhu, L. Yuan, K. Chaney, and K. Daniilidis, "Unsupervised event-based learning of optical flow, depth, and egomotion," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 2019-June, 2019, pp. 989–997. 12, 14, 17, 93

[33] H. Rebecq, R. Ranftl, V. Koltun, and D. Scaramuzza, "Events-to-video: Bringing modern computer vision to event cameras," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 2019-June, 2019, pp. 3852–3861. 12, 17

[34] G. Gallego, T. Delbrück, G. Orchard, C. Bartolozzi, B. Taba, A. Censi, S. Leutenegger, A. Davison, J. Conradt, K. Daniilidis, and D. Scaramuzza, "Event-based vision: A survey," 2019. 12

[35] A. Linares-Barranco, F. Gómez-Rodríguez, V. Villanueva, L. Longinotti, and T. Delbrück, "A usb3.0 fpga event-based filtering and tracking framework for dynamic vision sensors," in *2015 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2015, pp. 2417–2420. 13, 74

[36] A. Linares-Barranco, F. Perez-Peña, D. P. Moeys, F. Gomez-Rodriguez, G. Jimenez-Moreno, S. Liu, and T. Delbruck, "Low latency event-based filtering and feature extraction for dynamic vision sensors in real-time fpga applications," *IEEE Access*, vol. 7, pp. 134 926–134 942, 2019. 13

[37] X. Lagorce, C. Meyer, S. H. Ieng, D. Filliat, and R. Benosman, "Asynchronous Event-Based Multikernel Algorithm for High-Speed Visual Features Tracking," *IEEE Trans. Neural Networks Learn. Syst.*, vol. 26, no. 8, pp. 1710–1720, 2015. 13

[38] A. Glover and C. Bartolozzi, "Robust visual tracking with a freely-moving event camera," in *IEEE Int. Conf. Intell. Robot. Syst.*, vol. 2017-Septe, 2017, pp. 3769–3776. 13, 47, 52, 54, 55, 56, 74, 91, 92, 94

[39] A. Mitrokhin, C. Fermuller, C. Parameshwara, and Y. Aloimonos, "Event-Based Moving Object Detection and Tracking," in *IEEE Int. Conf. Intell. Robot. Syst.*, 2018, pp. 6895–6902. 13

[40] G. Gallego and D. Scaramuzza, "Accurate angular velocity estimation with an event camera," *IEEE Robot. Autom. Lett.*, vol. 2, no. 2, pp. 632–639, 2017. 13, 14, 93

[41] G. Gallego, H. Rebecq, and D. Scaramuzza, "A Unifying Contrast Maximization Framework for Event Cameras, with Applications to Motion, Depth, and Optical Flow Estimation," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 3867–3876. 13, 93

[42] F. Paredes-Valles, K. Y. W. Scheper, and G. C. De Croon, "Unsupervised Learning of a Hierarchical Spiking Neural Network for Optical Flow Estimation: From Events to Global Motion Perception," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 42, no. 8, pp. 2051–2064, 2020. 14

[43] S. H. Ieng, J. Carneiro, M. Osswald, and R. Benosman, "Neuromorphic event-based generalized time-based stereovision," *Front. Neurosci.*, vol. 12, no. JUL, 2018. 14

[44] Z. Xie, J. Zhang, and P. Wang, "Event-based stereo matching using semiglobal matching," *Int. J. Adv. Robot. Syst.*, vol. 15, no. 1, 2018. 14

[45] J. N. Martel, J. Müller, J. Conradt, and Y. Sandamirskaya, "An Active Approach to Solving the Stereo Matching Problem using Event-Based Sensors," in *Proc. - IEEE Int. Symp. Circuits Syst.*, vol. 2018-May, 2018. 14

[46] G. Gallego, J. E. Lund, E. Mueggler, H. Rebecq, T. Delbruck, and D. Scaramuzza, "Event-Based, 6-DOF Camera Tracking from Photometric Depth Maps," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 40, no. 10, pp. 2402–2412, 2018. 15

[47] C. Reinbacher, G. Munda, and T. Pock, "Real-time panoramic tracking for event cameras," in *2017 IEEE Int. Conf. Comput. Photogr. ICCP 2017 - Proc.*, 2017. 15

[48] A. R. Vidal, H. Rebecq, T. Horstschaefer, and D. Scaramuzza, "Ultimate SLAM? Combining Events, Images, and IMU for Robust Visual SLAM in HDR and High-Speed Scenarios," *IEEE Robot. Autom. Lett.*, vol. 3, no. 2, pp. 994–1001, 2018. 15

[49] E. Mueggler, G. Gallego, H. Rebecq, and D. Scaramuzza, "Continuous-Time Visual-Inertial Odometry for Event Cameras," *IEEE Trans. Robot.*, vol. 34, no. 6, pp. 1425–1440, 2018. 15, 54

[50] T. Stoffregen, G. Gallego, T. Drummond, L. Kleeman, and D. Scaramuzza, "Event-based motion segmentation by motion compensation," in *Proc. IEEE Int. Conf. Comput. Vis.*, vol. 2019-Octob, 2019, pp. 7243–7252. 15, 94

[51] X. Lagorce, G. Orchard, F. Galluppi, B. E. Shi, and R. B. Benosman, "HOTS: A Hierarchy of Event-Based Time-Surfaces for Pattern Recogni-

tion," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 7, pp. 1346–1359, 2017. 15

[52] E. Mueller, A. Censi, and E. Frazzoli, "Low-latency heading feedback control with neuromorphic vision sensors using efficient approximated incremental inference," in *Proc. IEEE Conf. Decis. Control*, vol. 54rd IEEE, 2015, pp. 992–999. 16

[53] A. Censi, "Efficient neuromorphic optomotor heading regulation," in *Proc. Am. Control Conf.*, vol. 2015-July, 2015, pp. 3854–3861. 16

[54] P. Singh, S. Z. Yong, J. Gregoire, A. Censi, and E. Frazzoli, "Stabilization of linear continuous-time systems using neuromorphic vision sensors," in *2016 IEEE 55th Conf. Decis. Control. CDC 2016*, 2016, pp. 3030–3036. 16

[55] H. Rebecq, R. Ranftl, V. Koltun, and D. Scaramuzza, "High speed and high dynamic range video with an event camera," *arXiv*, pp. 1–1, 2019. 17

[56] L. Wang, I. S. Mostafavi, Y. S. Ho, and K. J. Yoon, "Event-based high dynamic range image and very high frame rate video generation using conditional generative adversarial networks," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 2019-June, 2019, pp. 10 073–10 082. 17

[57] D. P. Moeys, F. Corradi, E. Kerr, P. Vance, G. Das, D. Neil, D. Kerr, and T. Delbruck, "Steering a predator robot using a mixed frame/event-driven convolutional neural network," in *2016 2nd Int. Conf. Event-Based Control. Commun. Signal Process. EBCCSP 2016 - Proc.*, 2016. 18

[58] M. Iacono, S. Weber, A. Glover, and C. Bartolozzi, "Towards Event-Driven Object Detection with Off-the-Shelf Deep Learning," in *IEEE Int. Conf. Intell. Robot. Syst.*, 2018, pp. 6277–6283. 18

[59] I. A. Lungu, F. Corradi, and T. Delbruck, "Live demonstration: Convolutional neural network driven by dynamic vision sensor playing RoShamBo," in *Proc. - IEEE Int. Symp. Circuits Syst.*, 2017. 18

[60] J. Binas, D. Neil, S. C. Liu, and T. Delbruck, "DDD17: End-to-end DAVIS driving dataset," 2017. 18

[61] W. Maass, "Networks of spiking neurons: The third generation of neural network models," *Neural Networks*, 1997. 18

[62] W. Maass and H. Markram, "On the computational power of circuits of spiking neurons," *J. Comput. Syst. Sci.*, 2004. 19

[63] P. U. Diehl, D. Neil, J. Binas, M. Cook, S. C. Liu, and M. Pfeiffer, "Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing," in *Proc. Int. Jt. Conf. Neural Networks*, vol. 2015-Septe, 2015. 19

[64] S. K. Esser, P. A. Merolla, J. V. Arthur, A. S. Cassidy, R. Appuswamy, A. Andreopoulos, D. J. Berg, J. L. McKinstry, T. Melano, D. R. Barch, C. Di Nolfo, P. Datta, A. Amir, B. Taba, M. D. Flickner, and D. S. Modha, "Convolutional networks for fast, energy-efficient neuromorphic computing," *Proc. Natl. Acad. Sci. U. S. A.*, vol. 113, no. 41, pp. 11 441–11 446, 2016. 19

[65] B. Rueckauer, I. A. Lungu, Y. Hu, M. Pfeiffer, and S. C. Liu, "Conversion of continuous-valued deep networks to efficient event-driven networks for image classification," *Front. Neurosci.*, vol. 11, no. DEC, 2017. 19

[66] J. H. Lee, T. Delbruck, and M. Pfeiffer, "Training deep spiking neural networks using backpropagation," *Front. Neurosci.*, vol. 10, no. NOV, 2016. 20

[67] S. B. Shrestha and G. Orchard, "Slayer: Spike layer error reassignment in time," in *arXiv*, 2018. 20

[68] Y. Dan and M. M. Poo, "Spike timing-dependent plasticity of neural circuits," 2004. 20

[69] J. C. Thiele, O. Bichler, and A. Dupret, "Event-based, timescale invariant unsupervised online deep learning with STDP," *Front. Comput. Neurosci.*, 2018. 20

[70] D. Neil, M. Pfeiffer, and S. C. Liu, "Phased LSTM: Accelerating recurrent network training for long or event-based sequences," in *Adv. Neural Inf. Process. Syst.*, 2016, pp. 3889–3897. 21, 31, 33

[71] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986. 21

[72] C. Olah, "Understanding lstm networks," 2015. [Online]. Available: http://colah.github.io/posts/2015-08-Understanding-LSTMs/ 22, 23, 24, 28, 30, 32

[73] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997. 22, 26

[74] Y. Bengio, P. Simard, and P. Frasconi, "Learning Long-Term Dependencies with Gradient Descent is Difficult," *IEEE Trans. Neural Networks*, vol. 5, no. 2, pp. 157–166, 1994. 22

[75] P. J. Werbos, "Generalization of backpropagation with application to a recurrent gas market model," *Neural Networks*, vol. 1, no. 4, pp. 339–356, 1988. 26

[76] D. Zipser and R. J. Williams, "Gradient-Based Learning Algorithms for Recurrent Networks and Thei r Computational Complexity," *Back-propagation Theory, Archit. Appl.*, pp. 433–486, 1995. 26

[77] A. Robinson and F. Fallside, "The utility driven dynamic error propagation network," 1987. 26

[78] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with LSTM," in *IEE Conf. Publ.*, vol. 2, no. 470, 1999, pp. 850–855. 26

[79] A. Graves and J. Schmidhuber, "Framewise phoneme classification with bidirectional LSTM and other neural network architectures," in *Neural Networks*, vol. 18, no. 5-6, 2005, pp. 602–610. 26, 29

[80] J. S. Garofolo, L. F. Lamel, W. M. Fisher, J. G. Fiscus, and D. S. Pallett, "DARPA TIMIT acoustic-phonetic continous speech corpus CD-ROM. NIST speech disc 1-1.1," *NASA STI/Recon Tech. Rep. N*, vol. 93, p. 27403, 1993. [Online]. Available: papers://e7d065ae-9998-4287-8af0-c9fa85af8e96/Paper/p44370 26

[81] K. Greff, R. K. Srivastava, J. Koutnik, B. R. Steunebrink, and J. Schmidhuber, "LSTM: A Search Space Odyssey," *IEEE Trans. Neural Networks Learn. Syst.*, vol. 28, no. 10, pp. 2222–2232, 2017. 26

[82] F. A. Gers and J. Schmidhuber, "Recurrent nets that time and count," in *Proc. Int. Jt. Conf. Neural Networks*, vol. 3, 2000, pp. 189–194. 28

[83] T. Yang, H. Wang, S. Aziz, H. Jiang, and J. Peng, "A Novel Method of Wind Speed Prediction by Peephole LSTM," in *2018 Int. Conf. Power Syst. Technol. POWERCON 2018 - Proc.*, 2019, pp. 364–369. 29

[84] Y. Bo, J. Tang, M. Yu, and W. Wei, "Ultra-short-term PV power forecasting based on LSTM with PeepHoles connections," in *iSPEC 2019 - 2019 IEEE Sustain. Power Energy Conf. Grid Mod. Energy Revolution, Proc.*, 2019, pp. 1222–1226. 29

[85] L. Fu, "Time series-oriented load prediction using deep peephole LSTM," in *12th Int. Conf. Adv. Comput. Intell. ICACI 2020*, 2020, pp. 86–91. 29

[86] M. M. Rahman and F. H. Siddiqui, "An optimized abstractive text summarization model using peephole convolutional LSTM," *Symmetry (Basel).*, vol. 11, no. 10, 2019. 29

[87] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," *IEEE Trans. Signal Process.*, vol. 45, no. 11, pp. 2673–2681, 1997. 29

[88] A. Zeyer, P. Doetsch, P. Voigtlaender, R. Schluter, and H. Ney, "A comprehensive study of deep bidirectional LSTM RNNS for acoustic modeling in speech recognition," in *ICASSP, IEEE Int. Conf. Acoust. Speech Signal Process. - Proc.*, 2017, pp. 2462–2466. 30

[89] X. Chen, A. Ragni, X. Liu, and M. J. Gales, "Investigating bidirectional recurrent neural network language models for speech recognition," in *Proc. Annu. Conf. Int. Speech Commun. Assoc. INTERSPEECH*, vol. 2017-Augus, 2017, pp. 269–273. 30

[90] S. Xue and Z. Yan, "Improving latency-controlled BLSTM acoustic models for online speech recognition," in *ICASSP, IEEE Int. Conf. Acoust. Speech Signal Process. - Proc.*, 2017, pp. 5340–5344. 30

[91] S. Lokesh, P. Malarvizhi Kumar, M. Ramya Devi, P. Parthasarathy, and C. Gokulnath, "An Automatic Tamil Speech Recognition system by using

Bidirectional Recurrent Neural Network with Self-Organizing Map," *Neural Comput. Appl.*, vol. 31, no. 5, pp. 1521–1531, 2019. 30

[92] W. Wang, Q. Tang, and K. Livescu, "Unsupervised Pre-Training of Bidirectional Speech Encoders via Masked Reconstruction," in *ICASSP, IEEE Int. Conf. Acoust. Speech Signal Process. - Proc.*, vol. 2020-May, 2020, pp. 6889–6893. 30

[93] M. Sundermeyer, T. Alkhouli, J. Wuebker, and H. Ney, "Translation modeling with bidirectional recurrent neural networks," in *EMNLP 2014 - 2014 Conf. Empir. Methods Nat. Lang. Process. Proc. Conf.*, 2014, pp. 14–25. 30

[94] X. Huang, H. Tan, G. Lin, and Y. Tian, "A LSTM-based bidirectional translation model for optimizing rare words and terminologies," in *2018 Int. Conf. Artif. Intell. Big Data, ICAIBD 2018*, 2018, pp. 185–189. 30

[95] C. Wang, H. Yang, C. Bartz, and C. Meinel, "Image captioning with deep bidirectional LSTMs," in *MM 2016 - Proc. 2016 ACM Multimed. Conf.*, 2016, pp. 988–997. 30

[96] J. Wang, W. Jiang, L. Ma, W. Liu, and Y. Xu, "Bidirectional Attentive Fusion with Context Gating for Dense Video Captioning," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 7190–7198. 30

[97] C. Wang, H. Yang, and C. Meinel, "Image Captioning with Deep Bidirectional LSTMs and Multi-Task Learning," *ACM Trans. Multimed. Comput. Commun. Appl.*, vol. 14, no. 2s, 2018. 30

[98] Y. Bin, Y. Yang, F. Shen, N. Xie, H. T. Shen, and X. Li, "Describing video with attention-based bidirectional LSTM," *IEEE Trans. Cybern.*, vol. 49, no. 7, pp. 2631–2641, 2019. 30

[99] J. Yang, H. Zhang, M. Dencler, and C. Lu, "Comparison of shape-based and stroke-based methods for segmenting handwritten chinese characters," in *Proc. - Fourth Annu. ACIS Int. Conf. Comput. Inf. Sci. ICIS 2005*, vol. 2005, 2005, pp. 114–119. 30

[100] Y. Chherawala, P. P. Roy, and M. Cheriet, "Combination of context-dependent bidirectional long short-term memory classifiers for robust offline handwriting recognition," *Pattern Recognit. Lett.*, vol. 90, pp. 58–64, 2017. 30

[101] O. Zubarieva, I. Deriuga, V. Holosko, and T. Ignatova, "Space balancing in online handwriting recognition postprocessing using deep bidirectional LSTM," in *Proc. Int. Conf. Front. Handwrit. Recognition, ICFHR*, vol. 2018-Augus, 2018, pp. 576–581. 30

[102] M. Mhiri, C. Desrosiers, and M. Cheriet, "Convolutional pyramid of bidirectional character sequences for the recognition of handwritten words," *Pattern Recognit. Lett.*, vol. 111, pp. 87–93, 2018. 30

[103] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," in *EMNLP 2014 - 2014 Conf. Empir. Methods Nat. Lang. Process. Proc. Conf.*, 2014, pp. 1724–1734. 30

[104] M. Ravanelli, P. Brake, M. Omologo, and Y. Bengio, "Improving speech recognition by revising gated recurrent units," in *Proc. Annu. Conf. Int. Speech Commun. Assoc. INTERSPEECH*, vol. 2017-Augus, 2017, pp. 1308–1312. 31

[105] Z. Tang, Y. Shi, D. Wang, Y. Feng, and S. Zhang, "Memory visualization for gated recurrent neural networks in speech recognition," in *ICASSP, IEEE Int. Conf. Acoust. Speech Signal Process. - Proc.*, 2017, pp. 2736–2740. 31

[106] G. Cheng, D. Povey, L. Huang, J. Xu, S. Khudanpur, and Y. Yan, "Output-gate projected gated recurrent unit for speech recognition," in *Proc. Annu. Conf. Int. Speech Commun. Assoc. INTERSPEECH*, vol. 2018-Septe, 2018, pp. 1793–1797. 31

[107] A. F. M. Agarap, "A neural network architecture combining gated recurrent unit (GRU) and support vector machine (SVM) for intrusion detection in network traffic data," in *ACM Int. Conf. Proceeding Ser.*, 2018, pp. 26–30. 31

[108] J. Chen, H. Jing, Y. Chang, and Q. Liu, "Gated recurrent unit based recurrent neural network for remaining useful life prediction of nonlinear deterioration process," *Reliab. Eng. Syst. Saf.*, vol. 185, pp. 372–382, 2019. 31

[109] Y. Wang, W. Liao, and Y. Chang, "Gated recurrent unit network-based short-term photovoltaic forecasting," *Energies*, vol. 11, no. 8, 2018. 31

[110] J. Donahue, L. A. Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, T. Darrell, and K. Saenko, "Long-term recurrent convolutional networks for visual recognition and description," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 07-12-June, 2015, pp. 2625–2634. 31

[111] Q. Huang, R. Chen, X. Zheng, and Z. Dong, "Deep sentiment representation based on CNN and LSTM," in *Proc. - 2017 Int. Conf. Green Informatics, ICGI 2017*, 2017, pp. 30–33. 31

[112] J. Zhao, X. Mao, and L. Chen, "Speech emotion recognition using deep 1D & 2D CNN LSTM networks," *Biomed. Signal Process. Control*, vol. 47, pp. 312–323, 2019. 31

[113] Y. Kim, Y. Jernite, D. Sontag, and A. M. Rush, "Character-Aware neural language models," in *30th AAAI Conf. Artif. Intell. AAAI 2016*, 2016, pp. 2741–2749. 31

[114] T. Bogaerts, A. D. Masegosa, J. S. Angarita-Zapata, E. Onieva, and P. Hellinckx, "A graph CNN-LSTM neural network for short and long-term traffic forecasting based on trajectory data," *Transp. Res. Part C Emerg. Technol.*, vol. 112, pp. 62–77, 2020. 31

[115] H. Van Hoof, T. Hermans, G. Neumann, and J. Peters, "Learning robot in-hand manipulation with tactile features," *IEEE-RAS Int. Conf. Humanoid Robot.*, vol. 2015-Decem, pp. 121–127, 2015. 31

[116] T. Akilan, Q. J. Wu, A. Safaei, J. Huo, and Y. Yang, "A 3D CNN-LSTM-Based image-to-image foreground segmentation," *IEEE Trans. Intell. Transp. Syst.*, vol. 21, no. 3, pp. 959–971, 2020. 31

[117] Q. Chen, B. Liang, and J. Wang, "A Comparative Study of LSTM and Phased LSTM for Gait Prediction," *Int. J. Artif. Intell. Appl.*, vol. 10, no. 4, pp. 57–66, 2019. 32

[118] L. Michieletto, B. Ouyang, and P. Wills, "Investigation of water quality using transfer learning, phased LSTM and correntropy loss," 2020, p. 25. 32

[119] S.-J. Bang, Y. Wang, and Y. Yang, "Phased-LSTM Based Predictive Model for longitudinal EHR Data with Missing Values," 2016.

[Online]. Available: https://www.cs.cmu.edu/{~}epxing/Class/10708-17/project-reports/project8.pdf 32

[120] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Adv. Neural Inf. Process. Syst.*, vol. 4, no. January, 2014, pp. 3104–3112. 36, 37, 54, 75

[121] J. Gehring, M. Auli, D. Grangier, D. Yarats, and Y. N. Dauphin, "Convolutional sequence to sequence learning," in *34th Int. Conf. Mach. Learn. ICML 2017*, vol. 3, 2017, pp. 2029–2042. 36

[122] Y. Jia, R. J. Weiss, F. Biadsy, W. Macherey, M. Johnson, Z. Chen, and Y. Wu, "Direct speech-to-speech translation with a sequence-to-sequence model," in *Proc. Annu. Conf. Int. Speech Commun. Assoc. INTERSPEECH*, vol. 2019-Septe, 2019, pp. 1123–1127. 37

[123] C. C. Chiu, T. N. Sainath, Y. Wu, R. Prabhavalkar, P. Nguyen, Z. Chen, A. Kannan, R. J. Weiss, K. Rao, E. Gonina, N. Jaitly, B. Li, J. Chorowski, and M. Bacchiani, "State-of-the-Art Speech Recognition with Sequence-to-Sequence Models," in *ICASSP, IEEE Int. Conf. Acoust. Speech Signal Process. - Proc.*, vol. 2018-April, 2018, pp. 4774–4778. 38

[124] S. Venugopalan, M. Rohrbach, J. Donahue, R. Mooney, T. Darrell, and K. Saenko, "Sequence to sequence - Video to text," in *Proc. IEEE Int. Conf. Comput. Vis.*, vol. 2015 Inter, 2015, pp. 4534–4542. 38

[125] W. Zaremba and I. Sutskever, "Learning to Execute," pp. 1–25, 2014. [Online]. Available: http://arxiv.org/abs/1410.4615 38

[126] G. Ren, X. Ni, M. Malik, and Q. Ke, "Conversational query understanding using sequence to sequence modeling," in *Web Conf. 2018 - Proc. World Wide Web Conf. WWW 2018*, 2018, pp. 1715–1724. 38

[127] C. Weng, J. Cui, G. Wang, J. Wang, C. Yu, D. Su, and D. Yu, "Improving attention based sequence-to-sequence models for end-to-end English conversational speech recognition," in *Proc. Annu. Conf. Int. Speech Commun. Assoc. INTERSPEECH*, vol. 2018-Septe, 2018, pp. 761–765. 38

[128] S. Choo, W. Seo, D. J. Jeong, and N. I. Cho, "Multi-scale Recurrent Encoder-Decoder Network for Dense Temporal Classification," in *Proc. - Int. Conf. Pattern Recognit.*, vol. 2018-Augus, 2018, pp. 103–108. 38

[129] J. Bütepage, M. J. Black, D. Kragic, and H. Kjellström, "Deep representation learning for human motion prediction and classification," in *Proc. - 30th IEEE Conf. Comput. Vis. Pattern Recognition, CVPR 2017*, vol. 2017-Janua, 2017, pp. 1591–1599. 39

[130] A. Alahi, K. Goel, V. Ramanathan, A. Robicquet, L. Fei-Fei, and S. Savarese, "Social LSTM: Human trajectory prediction in crowded spaces," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 2016-Decem, 2016, pp. 961–971. 39, 40, 42

[131] S. Pellegrini, A. Ess, K. Schindler, and L. Van Gool, "You'll never walk alone: Modeling social behavior for multi-target tracking," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2009, pp. 261–268. 39

[132] A. Lerner, Y. Chrysanthou, and D. Lischinski, "Crowds by example," *Comput. Graph. Forum*, vol. 26, no. 3, pp. 655–664, 2007. 39

[133] T. Fernando, S. Denman, S. Sridharan, and C. Fookes, "Soft + Hardwired attention: An LSTM framework for human trajectory prediction and abnormal event detection," *Neural Networks*, vol. 108, pp. 466–478, 2018. 39

[134] H. Xue, D. Q. Huynh, and M. Reynolds, "SS-LSTM: A Hierarchical LSTM Model for Pedestrian Trajectory Prediction," in *Proc. - 2018 IEEE Winter*

Conf. Appl. Comput. Vision, WACV 2018, vol. 2018-Janua, 2018, pp. 1186–1194. 39

[135] H. Manh and G. Alaghband, "Scene-lstm: A model for human trajectory prediction," 2018. 39

[136] B. Cheng, X. Xu, Y. Zeng, J. Ren, and S. Jung, "Pedestrian trajectory prediction via the Social-Grid LSTM model," *J. Eng.*, vol. 2018, no. 16, pp. 1468–1474, 2018. 39

[137] I. Hasan, F. Setti, T. Tsesmelis, A. Del Bue, F. Galasso, and M. Cristani, "MX-LSTM: Mixing Tracklets and Vislets to Jointly Forecast Trajectories and Head Poses," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 6067–6076. 39

[138] N. Bisagno, B. Zhang, and N. Conci, "Group LSTM: group trajectory prediction in crowded scenarios," in *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 11131 LNCS, 2019, pp. 213–225. 39

[139] Z. Pei, X. Qi, Y. Zhang, M. Ma, and Y. H. Yang, "Human trajectory prediction in crowded scene using social-affinity Long Short-Term Memory," *Pattern Recognit.*, vol. 93, pp. 273–282, 2019. 39

[140] M. Huynh and G. Alaghband, "Trajectory Prediction by Coupling Scene-LSTM with Human Movement LSTM," in *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 11844 LNCS, 2019, pp. 244–259. 39

[141] N. Bisagno, C. Saltori, B. Zhang, F. G. De Natale, and N. Conci, "Embedding group and obstacle information in LSTM networks for human tra-

jectory prediction in crowded scenes," *Comput. Vis. Image Underst.*, p. 103126, 2020. 39

[142] I. Hasan, F. Setti, T. Tsesmelis, V. Belagiannis, S. Amin, A. Del Bue, M. Cristani, and F. Galasso, "Forecasting people trajectories and head poses by jointly reasoning on tracklets and vislets," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–1, 2019. 39, 41

[143] J. Wiest, M. Höffken, U. Kreßel, and K. Dietmayer, "Probabilistic trajectory prediction with Gaussian mixture models," in *IEEE Intell. Veh. Symp. Proc.*, 2012, pp. 141–146. 41

[144] A. Houenou, P. Bonnifait, V. Cherfaoui, and W. Yao, "Vehicle trajectory prediction based on motion model and maneuver recognition," in *IEEE Int. Conf. Intell. Robot. Syst.*, 2013, pp. 4363–4369. 41

[145] R. Schubert, E. Richter, and G. Wanielik, "Comparison and evaluation of advanced motion models for vehicle tracking," in *Proc. 11th Int. Conf. Inf. Fusion, FUSION 2008*, 2008. 41

[146] G. Xie, H. Gao, L. Qian, B. Huang, K. Li, and J. Wang, "Vehicle Trajectory Prediction by Integrating Physics- and Maneuver-Based Approaches Using Interactive Multiple Models," *IEEE Trans. Ind. Electron.*, vol. 65, no. 7, pp. 5999–6008, 2018. 41

[147] W. Zhu, W. Wang, and G. Yuan, "An improved interacting multiple model filtering algorithm based on the cubature Kalman filter for maneuvering target tracking," *Sensors (Switzerland)*, vol. 16, no. 6, 2016. 42

[148] W. Li and Y. Jia, "Location of mobile station with maneuvers using an IMM-based cubature Kalman filter," *IEEE Trans. Ind. Electron.*, vol. 59, no. 11, pp. 4338–4348, 2012. 42

[149] B. D. Kim, C. M. Kang, J. Kim, S. H. Lee, C. C. Chung, and J. W. Choi, "Probabilistic vehicle trajectory prediction over occupancy grid map via recurrent neural network," in *IEEE Conf. Intell. Transp. Syst. Proceedings, ITSC*, vol. 2018-March, 2018, pp. 399–404. 42

[150] F. Altche and A. De La Fortelle, "An LSTM network for highway trajectory prediction," in *IEEE Conf. Intell. Transp. Syst. Proceedings, ITSC*, vol. 2018-March, 2018, pp. 353–359. 42

[151] S. H. Park, B. Kim, C. M. Kang, C. C. Chung, and J. W. Choi, "Sequence-to-Sequence Prediction of Vehicle Trajectory via LSTM Encoder-Decoder Architecture," in *IEEE Intell. Veh. Symp. Proc.*, vol. 2018-June, 2018, pp. 1672–1678. 42, 43, 54

[152] N. Deo and M. M. Trivedi, "Convolutional social pooling for vehicle trajectory prediction," in *IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. Work.*, vol. 2018-June, 2018, pp. 1549–1557. 42

[153] D. Falanga, S. Kim, and D. Scaramuzza, "How Fast Is Too Fast? the Role of Perception Latency in High-Speed Sense and Avoid," *IEEE Robot. Autom. Lett.*, vol. 4, no. 2, pp. 1884–1891, 2019. 43, 44, 54

[154] P. Sermanet, R. Hadsell, J. Ben, A. N. Erkan, B. Flepp, U. Muller, and Y. LeCun, "Speed-range dilemmas for vision-based navigation in unstructured terrain," in *IFAC Proc. Vol.*, vol. 6, no. PART 1, 2007, pp. 300–305. 44

[155] S. Behnke, A. Egorova, A. Gloye, R. Rojas, and M. Simon, "Predicting away robot control latency," in *Lect. Notes Artif. Intell. (Subseries Lect. Notes Comput. Sci.*, vol. 3020, 2004, pp. 712–719. 44

[156] S. Kim, A. Shukla, and A. Billard, "Catching objects in flight," *IEEE Trans. Robot.*, 2014. 44

[157] "OptiTrack — Motion Capture Systems." [Online]. Available: https://optitrack.com/ 44

[158] S. Kim and A. Billard, "Estimating the non-linear dynamics of free-flying objects," *Rob. Auton. Syst.*, 2012. 44

[159] K. Dong, K. Pereida, F. Shkurti, and A. P. Schoellig, "Catch the ball: Accurate high-speed motions for mobile manipulators via inverse dynamics learning," *arXiv*, no. ii, 2020. 45

[160] O. Clarisse and S.-K. Chang, "Vicon," in *Vis. Lang.*, 1986. 45

[161] M. Sato, A. Takahashi, and A. Namiki, "High-Speed Catching by Multi-Vision Robot Hand," pp. 9131–9136, 2020. 45

[162] M. Asenov, M. Burke, D. Angelov, T. Davchev, K. Subr, and S. Ramamoorthy, "Vid2Param: Modeling of Dynamics Parameters from Video," *IEEE Robot. Autom. Lett.*, 2020. 45, 94

[163] G. Metta, L. Natale, F. Nori, and G. Sandini, "The iCub project: An open source platform for research in embodied cognition," *Proc. IEEE Work. Adv. Robot. its Soc. Impacts, ARSO*, pp. 24–26, 2011. 46, 51

[164] R. B. Miller, "Response time in man-computer conversational transactions. Introductions and major concepts," *Fall Jt. Comput. Conf.*, 1968. [Online]. Available: www.computerhistory.org 50

[165] S. K. Card, G. G. Robertson, and J. D. Mackinlay, "The information visualizer, an information workspace," in *Conf. Hum. Factors Comput. Syst. - Proc.*, 1991. 50

[166] X. Zhao, S. Chumkamon, S. Duan, J. Rojas, and J. Pan, "Collaborative Human-Robot Motion Generation Using LSTM-RNN," in *IEEE-RAS Int. Conf. Humanoid Robot.*, 2019. 54

[167] A. Amir, B. Taba, D. Berg, T. Melano, J. Mckinstry, C. Di Nolfo, T. Nayak, A. Andreopoulos, G. Garreau, M. Mendoza, J. Kusnitz, M. Debole, S. Esser, T. Delbruck, M. Flickner, and D. Modha, "A low power, fully event-based gesture recognition system," in *Proc. - 30th IEEE Conf. Comput. Vis. Pattern Recognition, CVPR 2017*, vol. 2017-Janua, 2017, pp. 7388–7397. 54

[168] T. Delbruck and M. Lang, "Robotic goalie with 3 ms reaction time at 4% CPU load using event-based dynamic vision sensor," *Front. Neurosci.*, 2013. 54

[169] A. Glover, V. Vasco, M. Iacono, and C. Bartolozzi, "The Event-Driven Software Library for YARP—With Algorithms and iCub Applications," *Front. Robot. AI*, 2018. 55

[170] G. Metta, P. Fitzpatrick, and L. Natale, "YARP: Yet another robot platform," 2006. 55

[171] A. Glover and C. Bartolozzi, "Event-driven ball detection and gaze fixation in clutter," in *IEEE Int. Conf. Intell. Robot. Syst.*, 2016. 56

[172] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," in *3rd Int. Conf. Learn. Represent. ICLR 2015 - Conf. Track Proc.*, 2015. 76

[173] "Franka Emika — Introducing the Franka Emika Robot." [Online]. Available: https://www.franka.de/ 86

[174] J. Zhao, N. Risi, M. Monforte, C. Bartolozzi, G. Indiveri, and E. Donati, "Closed-loop spiking control on a neuromorphic processor implemented on the iCub," *IEEE J. Emerg. Sel. Top. Circuits Syst.*, 2020. 95

Acronyms

ATIS Asynchronous Time-based Image Sensor. 5, 10, 51, 55, 71, 73, 74, 78, 86

BPTT Backpropagation Through Time. 22, 26, 85

CEC Constant Error Carousel. 23, 28, 31, 33

CNN Convolutional Neural Network. 18–21, 31–33, 35, 36, 69

DL Deep Learning. 1, 3, 4, 11, 16, 18, 19, 21, 31, 54, 69

DVS Dynamic Vision Sensor. 10

FC Fully Connected. 19–21, 33

FOV field-of-view. 19, 47, 84, 89

FPGA Field Programmable Gate Array. 13, 46

GPU Graphics Processing Unit. 1, 11, 17

GRU Gated Recurrent Unit. 30, 31

HRI Human-Robot Interaction. 50, 55, 68–70

LSTM Long Short-Term Memory. 5, 21–23, 26, 28–32, 35, 38, 39, 42, 50, 52, 54–58, 61, 63, 64, 66, 68–71, 73, 75, 82, 84, 85, 91, 94, 95

NLP Natural Language Processing. 31

RMSE Root Mean Squared Error. 5, 76–78

RNN Recurrent Neural Network. 17, 21, 22, 25, 26, 29, 39, 42, 56, 75, 91

ROI region-of-interest. 74, 75, 92, 94

Seq2Seq Sequence-to-Sequence. 36, 38, 54, 60, 75, 76, 91

SNN Spiking Neural Network. 11, 14, 18–20, 93