POLITECNICO DI TORINO

Master degree course in Computer Engineering

Master Degree Thesis

Exploring Brain-Inspired Multi-Sensor Data Fusion Models for Improving Performances in Navigation and Tracking Applications

Advisors
Gianvito Urgese
Vittorio Fra

Candidate
Salvatore Tilocca

October 2024

Abstract

The computational resources necessary for training and utilizing traditional deep learning (DL) models are becoming increasingly costly as time progresses. The amount of computing power required has increased tenfold from 2012 to 2019, leading to a significant rise in costs. Neuromorphic technologies and spiking neural networks (SNNs) are inspired by how human brains work and offer an innovative approach, significantly reducing the required resources.
The objective of this thesis is to evaluate the potential for the adoption of brain-inspired solutions as a substitute for established techniques used to solve engineering tasks. In particular, this thesis aims to assess the feasibility of employing SNN-based models to enhance the performance of navigation and tracking tasks.
In the first part of the work, we investigate the use of a SNN for implementing a dead reckoning task. We use the Neural Kalman Model [1], a deep learning-based solution that combines synthetically generated GPS measurements and IMU data, as a baseline. This model uses a Temporal Convolutional Network (TCN) to estimate the parameters of an Extended Kalman Filter state. Our contribution is the replacement of the TCN with a fully spiking variant of the Legendre Memory Unit (LMU), a specialized recurrent cell that captures long-range dependencies using orthogonal Legendre polynomials. The spiking-based solution achieves a significant reduction in memory usage, improving efficiency by nearly 96%, and enhances accuracy, reducing error by 53%.
In the second part of the thesis, we shift focus to a fully spiking model, without using the Extended Kalman Filter, and work with real sensor data from the University of Michigan's NCLT dataset [2]. This dataset contains data from a Segway robot on campus, recorded using various sensors of differing quality. A comprehensive analysis is conducted to evaluate how each type of sensor impacts the accuracy of the model. Specifically, we assess the contribution of wheel encoders, IMUs, and gyroscopes to the model's performance. The objective is to estimate the next position from the initial position using multiple sensors, correcting GPS errors. Commercial GPS systems are often prone to errors due to environmental interference, multipath effects, or satellite synchronization issues. To address these limitations, our fully SNN-based model integrates inputs from these various sensors, aiming to minimize GPS-induced errors and provide accurate position prediction.

The proposed model determines the location of a moving object at each time step, relying on GPS when signals are strong and considering the previous prediction of the neuromorphic model when GPS is degraded. This allows continuous tracking in challenging environments. Results are compared with ground truth data from the NCLT dataset, obtained via a SLAM algorithm using LiDAR and high-quality GPS. Our spiking model shows up to a 13% improvement in accuracy over GPS alone, underscoring its effectiveness in difficult conditions.
In conclusion, the work presented in this thesis demonstrates that SNN models can serve as a viable alternative to traditional deep learning models. It shows that SNN can achieve results comparable to state-of-the-art methods while offering significant advantages in terms of computational efficiency. This is because they can be deployed on neuromorphic HW, which has shown order-of-magnitude efficiency improvements when compared to standard architectures in terms of energy per inference.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Moore's law is a descriptive model that has been employed to forecast the trajectory of integrated circuit development for several decades. The model posits that the number of transistors on a chip will double every two years, resulting in enhanced performance. Recent research has indicated that a physical limit has been reached, for this reason the size of transistor cannot be reduced anymore. Additionally Dennard scaling hypothesis, proposed that the miniaturization of transistor reduction in terms of power consumption, nevertheless, this hypothesis has been refuted by the observation that the contemporary generation of transistors, which are approximately 6 nm in size, are no longer capable of dissipating energy efficiently, resulting in an increase in energy consumption. This suggests that a physical limit may have been reached in this process.
Furthermore, the extensive utilization of neural networks in contemporary devices represents an additional source of complications. Indeed, the rising complexity of deep neural networks is intensifying the challenge of energy constraints in existing devices. The aforementioned energy constraints are becoming increasingly significant not only in the context of edge computing, but also in other contexts such as autonomous systems and mobile devices.
The substantial transfer of data between memory and processor during training and the utilization of deep neural networks presents a challenge to the efficacy of the Von Neumann architecture, which separates the memory from the processor (fig: 1.1). The continuous data exchange between memory and processor can be considered a bottleneck, as it introduces limitations. One potential solution to the aforementioned physical limits is the emerging field of **neuromorphic computing**. This approach represents a significant departure from the classic von Neumann architecture.

The term "neuromorphic" was first used by Carver Mead in the 1980s to describe brain-inspired computing systems. The components of neuromorphic hardware are in fact named like the basic units of the nervous system: neurons and synapses.

This naming convention reflects the intention of these systems to mimic the

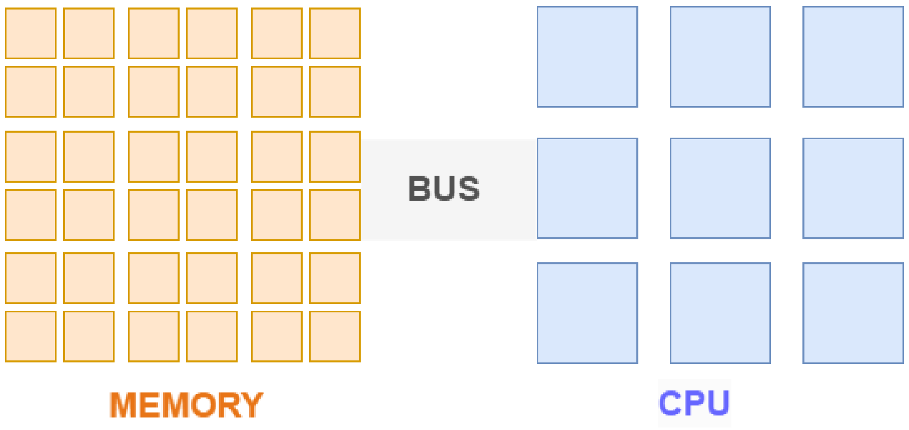


Figure 1.1: Von Neuman architecture: memory and CPU are separated and connected by a BUS.

functionality of biological neural networks.

In a biological context, neurons are the part of the nervous system, responsible for processing and transmitting information through electrical and chemical signals. Similarly, in neuromorphic systems, artificial neurons serve as computational units that process input signals and produce output spikes, thereby mimicking the activity of biological neurons. The way these artificial neurons integrate and respond to inputs is designed to replicate the behavior of their biological counterparts. Neuromorphic computing is considered an **unconventional technology**, since, as can be seen from the fig: 1.2, it is a novelty in many aspects. It departs from traditional computing architectures by emulating the parallel and distributed processing characteristics of the brain, opening new possibilities for efficient, adaptive, and energy-efficient computing systems, and low power operation that can often operate on orders of magnitude less power than traditional computing systems [3].

Neuromorphic hardware is still under development and available for research with projects such as SpiNNaker [4], BrainScaleS [5], ODIN [6] and the Tianjic chip [7] offering different capabilities from large-scale neuroscientific simulations to optimization of learning processes. Industry and academia are also involved, with examples such as IBM's TrueNorth [8] and Intel's Loihi [9] , demonstrating the importance of these innovative technologies today.
Intel's Loihi architecture exemplifies the advancements in neuromorphic computing, specifically designed for Spiking Neural Networks (SNNs). Loihi features 128 neuromorphic cores that enable real-time processing of information with significantly reduced energy consumption. By integrating memory closely with processing units, Loihi minimizes latency and maximizes energy efficiency, effectively addressing the

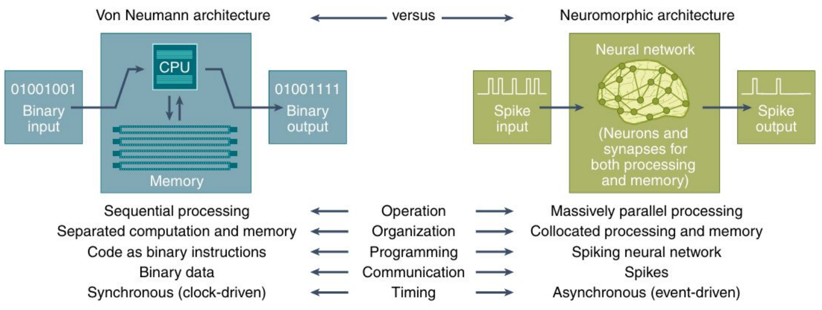

Von Neumann architecture
versus
Neuromorphic architecture
CPU
01001001 Binary input
01001111 Binary output
Memory
Neural network
Spike input
Spike output
(Neurons and synapses for both processing and memory)
Sequential processing
Separated computation and memory
Code as binary instructions
Binary data
Synchronous (clock-driven)
Operation
Organization
Programming
Communication
Timing
Massively parallel processing
Collocated processing and memory
Spiking neural network
Spikes
Asynchronous (event-driven)

Figure 1.2: Comparison between Neuromorphic architecture and von Neumann architecture [3].

limitations posed by traditional architectures.

Neuromorphic hardware can be considered a form of non-Von Neumann hardware. Indeed, in neuromorphic computers, the processor and memory are closely integrated (fig: 1.3) and managed by neurons and synapses. Information is encoded in pulses (spikes) with a value of either 0 or 1, whose frequency or spike timing represents numerical values.

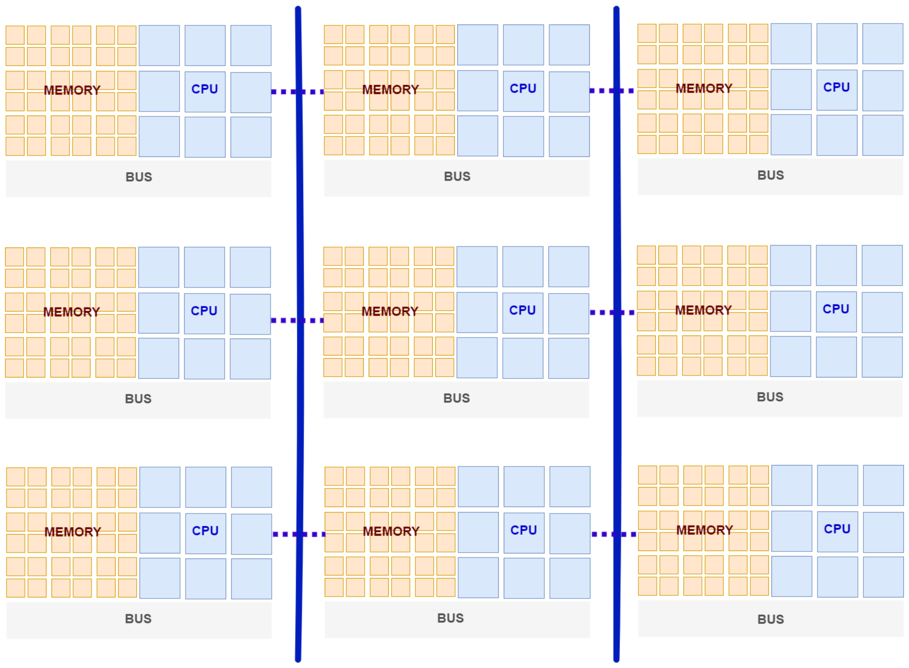


Figure 1.3: In Neuromorphic hardware memory and CPU are not physically separated

SNNs are a fundamental part to work with neuromorphic computing, since they leverage the event-based nature of these systems, indeed SNN are able to process asynchronous information only when data is available significantly improving energy efficiency. A key feature of neuromorphic hardware is its capacity to perform operations in a parallel manner, as all neurons and synapses are capable of functioning in this mode. Another crucial aspect of neuromorphic computing is event-based processing, whereby these computers process information only when data is available, thus enhancing efficiency. Computations are executed by neurons and synapses only when they receive spikes, which are generally infrequent in the overall operation of the network.

In essence, neuromorphic computing represents a radical breakthrough in the field of computation, offering a potentially more efficient and scalable solution than traditional systems. It could prove crucial to the future of computation, given that existing technologies are reaching their limits. The research presented in this thesis demonstrates that Spiking Neural Network offer a viable alternative for the execution of complex Deep Learning tasks and are capable of achieving results comparable to the state of the art, offering at the same time an order of magnitude greater energy efficiency and processing speed. In particular, our research has focused on tasks such as dead reckoning and the enhancement of GPS estimation. This has resulted in an improvement of approximately 50% in accuracy and a significant reduction in the required memory compared to the state of the art. This has enabled us to overcome issues such as those pertaining to the interruption of the Global Positioning System (GPS) signal or the presence of low-quality GPS signals. Two case studies were conducted to address these tasks. The first employed a spiking neural network in conjunction with an extended Kalman filter on a synthetic dataset, while the second employed an SNN without the extended Kalman filter, working with real-world measurements.
In conclusion, this research demonstrates the potential of SNNs and neuromorphic systems to not only match the performance of traditional architectures but to exceed them in terms of efficiency and speed. The results position neuromorphic networks as a promising technology for addressing future challenges in computation and artificial intelligence.

Chapter 2

Background

2.1 Sensors: Functional Analysis and Potential Applications

Nowadays, the usage of advanced sensors together with deep neural network has led to important innovations in several fields, such as autonomous driving, localization and medical task. Among the sensors most used especially in the context of autonomous driving, but not only, is important to mention IMU (Inertial Measurement Unit),Wheel Encoder, GPS (Global Position System) and LiDar (Light Detection and Ranging). Moreover, to specific task such as Road Surface Classification cameras are employed in the majority of state of the art methods. Such as [10] that propose a deep analyses of road surface classification using multi sensors data, composed of: accelerometer, gyroscope, magnetometer, temperature sensor and camera. The authors of this paper propose an analysis of different methods for road surface classification, which include an analysis starting from more classical methods like SVM, KMC, KNN up to deep learning methods like CNN, LSTM and CNN-LSTM. An interesting article is also [11] which propose a solution for the road surface classification using camera-based image recognition, the peculiarity of this paper is the use of a continual learning solution. **Continual Learning** is a concept to learn a model for a large number of tasks, domain, class sequentially without forgetting knowledge obtained from the preceding tasks, domain, class where the data in the old tasks are not available anymore during training new ones. It has been acknowledged that reliance on cameras can present challenges, such as higher costs due to the necessity for high-quality image processing hardware and the associated computational power. Consequently, in this thesis, the decision was taken not to utilise cameras for this reason. The objective of this chapter is to analyses the functionalities and possible use cases of the sensors mentioned above, and how they can be employed collaboratively to overcome each other's limitations.

2.1.1 Global Positioning System (GPS)

The Global Positioning System (GPS) is a space-based global navigation satellite system (GNSS) that is able to provide provides reliable positioning. If the signals of the GPS are provided by at least 4 satellites, it can be stated with reasonable certainty that the quality of position results are reliable. Besides the accuracy in providing the position of moving objects, the fact that it is light and cheap (about 5€) has made sure that in recent years there was an increase in its spread, making it available in everyday devices, and offering the possibility of various use cases: In navigation contexts such as Google Maps, it can be used to guide the user to their destination using the best route [12], can be used in fitness contexts such as running or cycling to provide useful data on the distance traveled [13], GPS is also utilized by emergency services to facilitate rapid responses to incidents [14], ensuring the precise localization of the event in question.

However, GPS positioning has many limitation such as signal outage. In urban and mountainous areas, or in situation of indoor or underground, the GPS signal can have interruptions [15]. Moreover in certain situation such as navigation for cars sport or airplane, the low update frequency of GPS can be not acceptable, a good solution for mitigate this issue can be the integration of multiple sensors like Inertial sensors.

One of the objectives of this thesis is to address the limitations of GPS by integrating its data with other sensors, such as IMUs and wheel encoders, as demonstrated in recent work, where sensor fusion techniques have been used to improve state estimation [16].

2.1.2 Inertial Measurement Unit (IMU)

Among the most used sensors, there are Inertial Measurement Unit (IMU) sensors, composed by accelerometer that measure the global linear acceleration, gyroscope able to measure angular velocities and sometime is possible to find magnetometer that measure the magnetic field. This sensors can be used in a wide range of use case, such as Human Activity Recognition like [17] that explore solution for this using neuromorphic approach, Augmented Reality (AR)/ Virtual Reality (VR) [18] used in headsets to track head orientation, medical equipment IMUs are used in surgical robotic systems to provide precise feedback on the position and movement of surgical instruments [19], IMU can be used to provide time information about the position, orientation, and motion of autonomous vehicles, this can be helpful in absence of signal GPS, we focus our research in this topic.
The cost of IMU sensors depends on several factors, such as accuracy, application, or technical specification. For low cost IMU, typical used in smartphone or game application, price ranges from 10€ to 50€. Mid range IMU, typical used for application such as autonomous guide and robotics context their price can vary from

50€ to 500€. For high-end IMUs, designed for precision-critical applications like aerospace, defense, and medical devices, prices range from €500 to over €5000. For specific case such as missile guidance or advanced research the cost can be reach €20000.

2.1.3 Wheel Encoder

Wheel Encoder sensors counts the number of rotation of robot/car wheels, therefore allow to estimate the speed or distance traveled with reasonable accuracy. Sometimes wheel encoders are combined together with IMUs sensors, since in scenarios where the axle is wet or slippery, the wheel encoder is not able to detect wheel, authors of paper [16] proposes a system that integrates multiple sensors to overcome these limitations. There are three main types of Wheel Encoders:

- **Mechanical Type**: Mechanical type wheel encoder has a variable resistor that change its electrical resistance in proportion to the rotation of the angle.
- **Optical Type**: This method employs a light sensor to verify whether light is traversing a slot in the radial direction of a rotating disc, designated a code wheel, affixed to the motor shaft. When light pulse signal traverses the slit there is a transformation, enabling the detection of the drive shaft's rotational velocity through pulse counting (see fig: 2.1).
- **Magnetic type**: This method employs the use of a magnetic sensor to quantify alterations in the magnetic field distribution generated by a permanent magnet affixed to the motor shaft. As the motor rotates, the magnetic field distribution of the above-mentioned permanent magnet changes accordingly. Therefore, by detecting the magnetic field distribution using a magnetic sensor, the rotational position of the motor shaft can be accurately determined.

Their cost can range from €50 to over €2000.

2.1.4 Light Detection and Ranging (LiDAR)

Light Detection and Ranging (LiDAR) are special sensors that uses pulsed laser to measure distance. At first, LiDAR was used to help make maps of small rivers and streams. Subsequently in the 1980s, with the emergence of GPS, LiDAR became an integral tool in collecting large-scale geospatial data and in creating topographical maps. Nowadays LiDAR sensors are cheaper and available in more device, recent iPhone have LiDAR scanner that can reach 4.5m of depth. LiDAR senors are used in different use case, such as Self-Driving Cars that can be used to ensure that follow the correct path and detect pedestrian or other object in the route, they can be used also for geospatial mapping to create high resolution topographic map, can

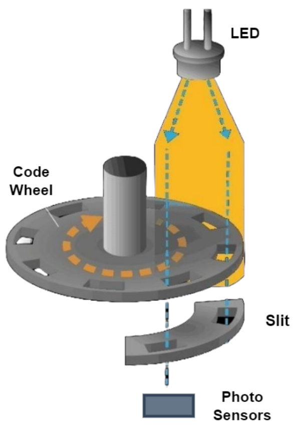


Figure 2.1: Optical Wheel Encoder taking inspiration from [20].

be used also for building construction infact it allow to build an extremely precise 3D model of object. The price of these sensors varies according to the context of use, starting from €150 for phone use cases and can reach €100000 in self-driving car contexts where extreme precision is required.

2.2 Dead Reckoning

Accurate positioning of vehicle is becoming an increasingly important task especially in the context of autonomous driving, Global Navigation Satellite System such as GPS are a good way to obtain vehicle position. GPS with technique Real Time Kinematic (RTK) uses a station located in a fixed and known point, which calculates and transmits the corrections to the mobile point, are extremely precise device and are used in contexts such as agriculture and autonomous driving, it can position with sub-decimetre accuracy in open area environments, but in urban areas environment this accuracy can be reduced due to the blockage or reflection of the satellite signal by the tall buildings. In the area where is not possible to obtain an accuracy in position, dead reckoning plays an important role. Dead Reckoning is a technique that compute the position change using the vehicle motion, such as acceleration, direction etc. This method does not rely on external signals, like those from satellites but instead uses internal data provided by onboard sensors such as accelerometers, gyroscopes, and magnetometer. The key concept is that, once the initial position of the vehicle is known, dead reckoning can track the new position by continuously calculating subsequent movements (fig: 2.2), as demonstrated in the work of Brossard et al. (2020), a variant of the Kalman filter is used in combination with a Convolutional Neural Network (CNN) to enhance the accuracy of dead reckoning by correcting sensor drift and improving robustness in challenging environments [21].

In recent autonomous driving systems, dead reckoning is performed using very expensive sensors such as fiber optic gyroscopes or LiDAR, these last provide excellent precision but their cost is too high for use them in mass-market vehicles. Fiber optic gyroscopes are extremely accurate in detecting the vehicle's rotation and changes in orientation, while LiDAR sensors use lasers to measure distances with great accuracy, creating detailed maps of the surrounding environment. However, the cost of these sensors can be prohibitive for mass production. One of the goals of our research is to develop a dead reckoning system using low-cost IMU sensors that can provide sufficiently accurate data for position calculations, and to demonstrate the impact of different sensor types across various price ranges on result quality

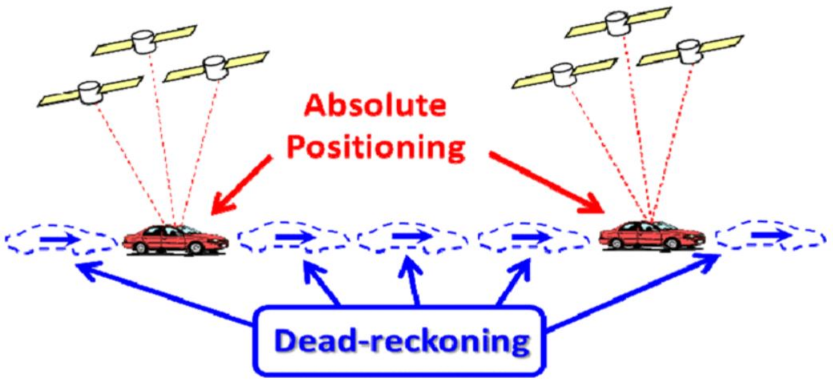


Figure 2.2: Image of absolute positioning and dead reckoning[22]

2.2.1 Why use Dead Reckoning?

As I said before despite the continuous progress in this field, sometimes use only the GPS still present limitations, here some use cases where dead reckoning could be critical:

- Fill gaps due to interruptions GPS signals: A record that occurs in everyday life is interruptions of GPS signals, this occur when signals of GPS are obstructed or can't reach receiver, like for instance when we are drive in dense urban area, or in a tunnel.
- Improve GPS signals quality: The same factors of interruption of GPS signals can degrade the quality of the sensors, and sometimes even other sources of RF interference (such as nearby cell towers or modems) can play a role.

- Collecting Data: While this data is primarily collected to support positioning calculations when GNSS signals fail, it also offers the added benefit of providing movement history that cannot be derived solely from GNSS signal data. These insights can be utilized to monitor driving behavior, assess the performance of autonomous vehicles (AV), and analyze similar trends.

- Improving Frequency of data: A further advantage of inertial sensors is their ability to continuously acquire vehicle-specific data or the hardware to which they are connected. While positioning history can be viewed from GNSS and RTK data, dead computing solutions collect data about the unique movement of the object they are connected to, including speed, direction, altitude, and more. While this data is mainly captured for fuel positioning calculations when GNSS fails, it provides the added benefit of showing the motion history that cannot be derived from GNSS signal data alone. You can use this information to monitor driving behavior, autonomous vehicle performance (AV) and similar trends.

Despite ancient origins Dead Reckoning remains relevant even nowadays thanks to its simplicity and efficiency. In spits of its limitations, like accumulation of errors over time, dead reckoning continues to be a crucial task, especially when combined with other navigation techniques to enhance overall accuracy.

Table 2.1 below presents comparison with state of the art methods, majority of this approach include Deep Learning, Kalman Filter or its variants, or a combination of both. IONet [23] use a Deep Learning model, based on LSTM, the input is provided by IMU sensors with accelerometer and gyroscope, the ground-truth labels are provided by visual motion tracking system (Vicon). IONet was developed for pedestrian dead reckoning. L-IONet [24] is a variant of IONet that replace LSTM with a Convolutional layer, to increase speed and reduce memory required. AbolDeepIO [25] employ an LSTM model that takes in input from accelerometer and gyroscope, the model was trained by the authors on Micro Aerial Vehicle (MAV) dataset [26], and retrained on Agrobot dataset by authors of [1]. VeTorch [27] utilizes a Temporal Convolutional Network (TCN) taking input IMU sensors from smartphone in a vehicle when GPS signal is not available. While UKF-MINS+GPS [28] and EKF INS+GPS [29] propose variants of classic Kalman Filter without employ deep learning model. Based on the results detailed in Table 2.1, we have made the decision to adopt the Neural Kalman Filter (Neural-KF) as the baseline for our model. This choice reflects our assessment of its performance and suitability in comparison to other potential approaches.

Method	RMSE (m)	Model Size (MB)	GPS
IONet	16.5	1.71	No
L-IONet	16.4	0.55	No
AbolDeepIO	15.7	12.5	No
VeTorch	14.8	29.6	No
Neural-KF	7.85	1.10	No
UKF-MINS+GPS	4.06	8.09	Yes
EKF INS+GPS	2.22	5.48	Yes
Neural-KF	1.07	2.17	Yes

Table 2.1: Comparison of state of the art methods for dead reckoning in context, trained and tested on Agrobot Dataset by the authors of paper [1].

2.3 GPS Refinement

In the lasts years, precise positioning is becoming a crucial task in numerous applications, in particular in mobile devices such as smartphone and tablet that use Global Navigation Satellite Systems (GNSS) systems to provide position estimation. Although the progress technologies in this field, the precision of GPS measurement available on commercial device is often limited, with error that can reach 10-15 meters in the majority of case [30] . It's possible to obtain sub-meters position precision using advanced technologies such as Differential Global Positioning System (DGPS) and GPS RTK that uses a station in a fixed position to correct measurement provided by signal of GPS, or Assisted GPS (AGPS) that use data coming from mobile devices, Wi-Fi to improve the fix of GPS when the signal is weak. Since the high cost their efforts are often limited. A possible solution to these issues is proposed in the paper [31], where the authors explore the integration of IMU data with GPS to obtain reliable positioning even in contexts where GPS signals are not precise, using a variant of Kalman Filter with IMU sensors, moreover using accelerometer contained in IMU is revealed static position, when static position is revealed mean of lasts GPS measurement is computed to reduce drift error. One of the use cases of this thesis's research regards GPS refinement. As we will discuss in Chapter 4.2, a use case of GPS tracking is presented in which, starting from the initial position provided by the GPS, the task is using a Spiking Neural Network estimate the next position while reducing its error.

2.3.1 Why use GPS Refinement?

The usage of GPS refinement is principally related to reduce the GPS error and improve the precision of GPS systems, here some use cases where GPS refinement could be critical:

- Autonomous Navigation: Vehicle, Drone and Robot need to have a precise position to navigate safely, GPS refinement reduce the error of GPS, improving the capabilities to follow path.
- Mapping and Topography Reliefs: Obtain precise position information in the ground it's a fundamental task to design buildings, roads and infrastructure.
- Urban Application: GPS refinement can be useful to track traffic. Have a precise position allow to have estimate traffic in real time and predict congestion in the roads.

2.4 Kalman Filter

Nowadays Kalman Filter is one of the most used algorithms of sensor fusion, it is used to reduce the bias of the measurement and to make predictions of the next state. Although it was discovered between 1958-1961 it is still present in many states of the art models, one of the first practical uses dates to 1969 when Stanley Schimdt researcher of Nasa use it to estimate trajectories of Apollo 11 [32].
The key idea behind Kalman Filter involves predicting the values of interest based on current data, comparing this prediction with sensor measurements at the next time step, and the performing a weighted average. The determination of this weight is a crucial task and his known as Kalman Gain. Kalman Filter is capable to make optimal predictions under the hypothesis of linear system and zero-mean Gaussian error distribution.

2.4.1 Kalman Filter for Linear System

Kalman Filter use differential equations and initial conditions to make prediction of the next state, for instance if we take in consideration a generic discrete linear system described by the eq: 2.1:

$$x_{k+1} = A_k x_k + B_k u_k + w_k \tag{2.1}$$

Where:

- x_k : states vector containing measure of our interest at the time k, for instance position and speed
- A_k :the matrix that connect the state of k-instant to the state of time k+1,
- B_k : matrix that connect the input at time k to the state of time k+1,
- u_k : input of the system like for instance measurement of accelerometer, gyroscope and magnetometer
- w_k : is the process noise, which is assumed to be drawn from a zero mean multivariate normal distribution,with covariance Q_k: .

- Q_k: Q is the process noise covariance matrix, this matrix represents the uncertainty of the process; if Q is large, it implies there is significant uncertainty in the process model, the filter will give more weight to the new measurements than to the prediction based on the model. If Q is small, the model is considered very accurate so the filter will rely more on the model predictions

The measurements predicted and the state variables are related by the eq: 2.2:

$$y_{k+1} = H_k x_k + v_k \tag{2.2}$$

Considering:

- H_k :is the observation model, it maps the true state space into the metric space.
- v_k : is the observation noise, which is assumed to be zero mean Gaussian white noise with covariance R_k.
- R_k: R is the measures noise covariance matrix, this matrix reflects the uncertainty in the sensor measurements, for a large R we have noise measurement, for a small R the measurement are accurate

In the most of case estimate the values of covariance matrix Q and R it's hard, and usually this values are chosen with a fine tuning phase.
At this point, we introduce the eq: 2.3:

$$\hat{x}_k^- = A_k \hat{x}_k + B_k u_k \tag{2.3}$$

Where the minus sign indicates the a priori estimate obtained through the knowledge of the model assuming zero error. With $\hat{x}$ we instead mean the a posteriori estimate, which is given by the linear combination between the a priori estimate and the weighted difference between the current measurement and the predicted measurement, namely:

$$\hat{x}_k = \hat{x}_k^- + K_k(z_k - H_k \hat{x}^-) \tag{2.4}$$

Where:

- z_k : Observation (or measurement) a time k of the true state x_k
- $\hat{x}_k$: Is the a posteriori estimate of the state at time step k, it represent the final estimate.
- $z_k - H_k \hat{x}^-$: Innovation or measurement pre-fit residual
- K_k : Kalman Gain, is a crucial component of the Kalman filter, the objective of the Kalman Gain is to minimize covariance matrix P, that represents the covariance matrix of the estimation error.

The error and covariance matrix a posterior of the process can be defined as with eq: 2.5, 2.6:

$$e_k = (x_k - \hat{x}) \tag{2.5}$$

$$P_k = E[(x_k - \hat{x})(x_k - \hat{x})^T] \tag{2.6}$$

Combining eq: 2.2 with eq: 2.4, we obtain eq 2.7:

$$\hat{x}_k = (I - K_k H)\hat{x}_k^- + K_k H x_k + K_k v_k \tag{2.7}$$

At this stage, we can now refer to the posterior error eq: 2.5 and the covariance matrix eq: 2.6 as:

$$e_k = ((I - K_k H)(x_k - \hat{x}_k^-) - K_k v_k) \tag{2.8}$$

$$P_k = (I - K_k H)P_k^-(I - K_k H)^T + K_k R_k K_k^T \tag{2.9}$$

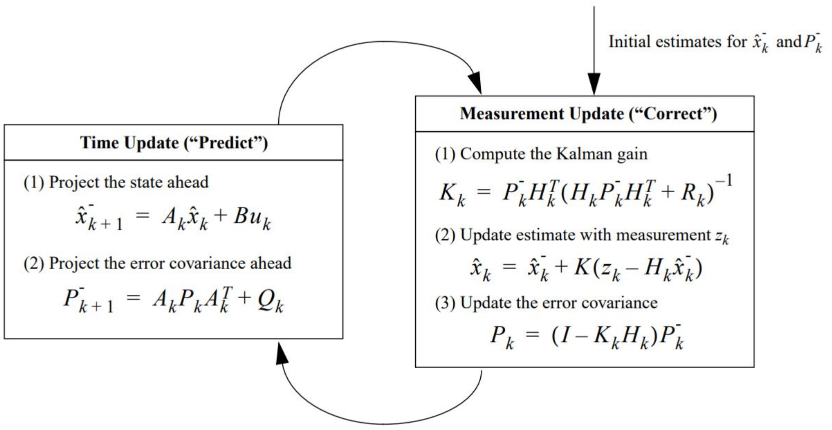
As mentioned earlier, the goal of the Kalman Gain is to minimize the error covariance matrix P of our prediction. To achieve this, the derivative of the trace of P with respect to K is computed, yielding the solution:

$$K_k = P_k^- H_k^T (H_k P_k^- H_k^T + R_k)^{-1} \tag{2.10}$$

Now, it is possible to update the posterior covariance matrix P using the calculated gain:

$$P_k = (I - K_k H)P_k^- \tag{2.11}$$

What has been written so far is summarized in the fig: 2.4, where x_k^- and P_k^- are respectively the state vector containing the initial conditions and the error covariance matrix for the initial conditions



Initial estimates for $\hat{x}_k^-$ and P_k^-
Time Update ("Predict")
(1) Project the state ahead
$\hat{x}_{k+1}^- = A_k\hat{x}_k + Bu_k$
(2) Project the error covariance ahead
$P_{k+1}^- = A_kP_kA_k^T + Q_k$
Measurement Update ("Correct")
(1) Compute the Kalman gain
$K_k = P_k^-H_k^T(H_kP_k^-H_k^T + R_k)^{-1}$
(2) Update estimate with measurement z_k
$\hat{x}_k = \hat{x}_k^- + K(z_k - H_k\hat{x}_k^-)$
(3) Update the error covariance
$P_k = (I - K_kH_k)P_k^-$

Figure 2.3: Comprehensive diagram illustrating the operation of the Kalman filter [33].

2.4.2 Kalman filter for non-Linear systems

As mentioned above Kalman Filter work under assumption of linear system, this can be limitation since the most of the phenomena are highly non linear therefore, for this reason it is imperative to offer a feasible alternative to address the issue of

estimation within this class of systems, which is why it is important to know how to use the **Extended Kalman Filter**.
Consider a non linear system described by the eq: 2.12

$$x_{k+1} = f(x_k, u_k, w_k) \tag{2.12}$$

$$z_k = h(x_k, v_k) \tag{2.13}$$

Where f and h are respectively the non linear function that connect the system from step k to the system of the step $k+1$, and the non linear function that relates the state at step k to the actual measurements at step k.
Extended Kalman Filter is based on the Kalman Filter algorithm, but incorporates certain modifications, discussed below, that make it suitable for nonlinear systems. As in the linear case, the objective of the filter is to provide an estimate of the state that is as close as possible to the true state of the system. The first step to lead us back to the theory already discussed for the KF is to define a linearization of the system state with eq: 2.14, 2.15

$$\hat{x}_{k+1} \approx \hat{x}^{-}_{k+1} + A_k(x_k - \hat{x}_k) + W_k w_k \tag{2.14}$$

$$z_k \approx \hat{z}^{-}_{k+} + A_k(x_k - \hat{x}_k) + V_k w_k \tag{2.15}$$

Where:

- A_k is the Jacobian Matrix of function f w.r.t x:
$$A_k = \left.\frac{\partial f}{\partial x}\right|_k \tag{2.16}$$
- W_k is the Jacobian Matrix of function f w.r.t w:
$$W_k = \left.\frac{\partial f}{\partial w}\right|_k \tag{2.17}$$
- H_k is the Jacobian Matrix of function h w.r.t x:
$$H_k = \left.\frac{\partial h}{\partial x}\right|_k \tag{2.18}$$
- V_k is the Jacobian Matrix of function h w.r.t v:
$$V_k = \left.\frac{\partial h}{\partial v}\right|_k \tag{2.19}$$

At this stage, the process operates in a manner similar to the Kalman filter. For clarity and reference, a summary diagram is provided below, which delineates the functioning of the Extended Kalman filter. This diagram is intended to offer a comprehensive overview, highlighting the key steps and mechanisms involved in the extended Kalman filter's operation.

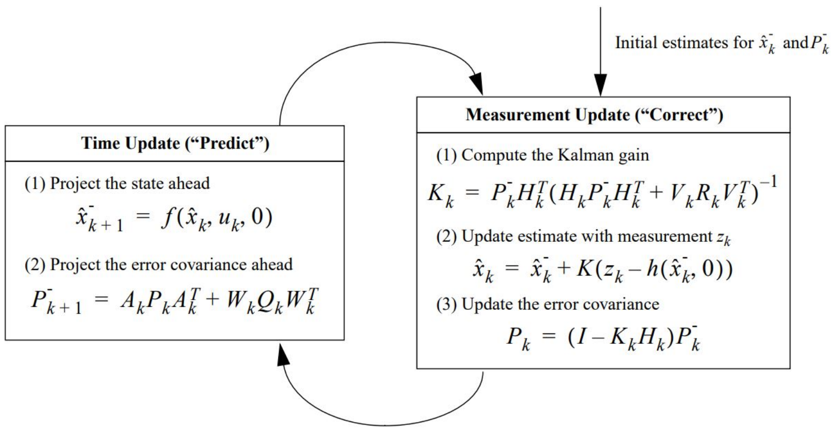

Initial estimates for $\hat{x}_k^-$ and P_k^-
Time Update ("Predict")
(1) Project the state ahead
$\hat{x}_{k+1}^- = f(\hat{x}_k, u_k, 0)$
(2) Project the error covariance ahead
$P_{k+1}^- = A_k P_k A_k^T + W_k Q_k W_k^T$
Measurement Update ("Correct")
(1) Compute the Kalman gain
$K_k = P_k^- H_k^T (H_k P_k^- H_k^T + V_k R_k V_k^T)^{-1}$
(2) Update estimate with measurement z_k
$\hat{x}_k = \hat{x}_k^- + K(z_k - h(\hat{x}_k^-, 0))$
(3) Update the error covariance
$P_k = (I - K_k H_k) P_k^-$

Figure 2.4: Comprehensive diagram illustrating the operation of the Extended Kalman Filter [33].

2.5 Artificial Neural Networks

Artificial Intelligence term was introduced for the first time in the 1950s, with the aim of enabling computers to think in a human-like manner [34]. At a later time in the 1980s was introduced **machine learning** concept, because it was understood that the issue lay in the necessity of a machine learning mechanism for handling the data, while before of machine learning era artificial intelligence was based on a set of pre-fixed rules, this was a source of problems in the presence of complex data. Around 2010, there was a significant surge in the adoption of **Deep Learning**. It differs from the preceding approach, known as **shallow learning**, in that it necessitates the transformation of input data into a mathematical representation, such as vectors. All these concepts are subfields of artificial intelligence.
To define the concept of learning, three elements are required: experience, task, and performance measures. Essentially, it is stated that an agent is learning when its performance in a given task improves with experience. We talk about **supervised learning** when for every data we use for the training of our model we have a label. Instead we talk about **unsupervised learning** when the training data used for thetraining process has not label, an example of task in this family is **clustering**, it allows us to create cluster based on similarities on data. Finally **reinforcement learning** (fig: 2.7), revolves around the concept of an agent interacting with an environment by executing actions and receiving rewards based on the impact of these actions.

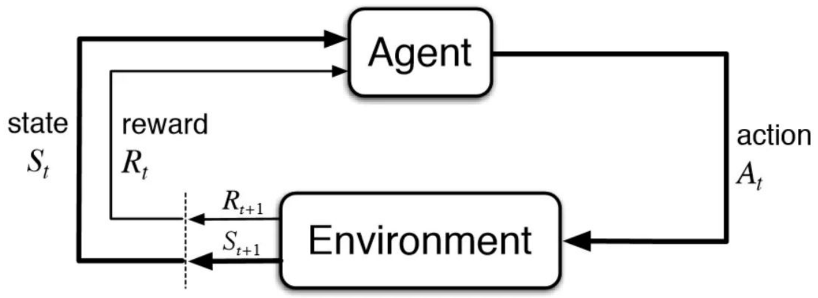


Figure 2.5: Summary scheme of Reinforcment Learning [35].

2.5.1 Neural Network and Neuron Perceptron

The initial objective of Artificial Intelligence is to find a model able to simulate the behaviour of the human neuron. The output of perceptron is +1 or -1 accordingly to his weighted input, it's behavious is like a linear classifier defined in eq: 2.20:

$$h(x) = sign(w^T x) \tag{2.20}$$

The goal of perceptron is to find the hyperplane in the input space that maximize the margin between data point of different classes, ensuring that points from different classes lie on opposite sides of this hyperplane (fig: 2.6). This involves in find the weight vector w that separates the data points of different classes.

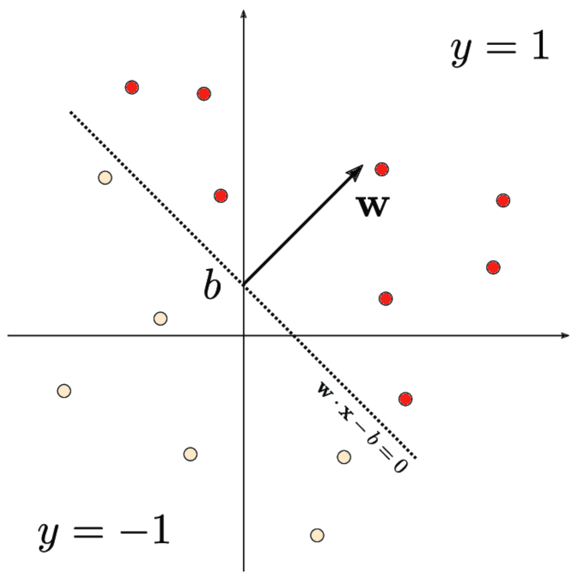


Figure 2.6: The interpretation of a perceptron as a oriented hyperplane [36].

Perceptron works under the assumption of linearly separable data, in presence of non-linear separable data the algorithm not reach convergence. One solution to address this limitation is to combine multiple perceptrons to construct a more complex function. An idea to improve performance of perceptron is to substitute step function with sigmoid function, in this way the behavior it's like logistic regression algorithm: **logistic regression** is a classification algorithm that use sigmoid function to model the probability of an input belonging to a certain class. As previously mentioned, the key idea to overcome problem of neuron perceptron, is to build a chain of neuron perceptron, the final structure is called **Neural Network** fig: 2.7.

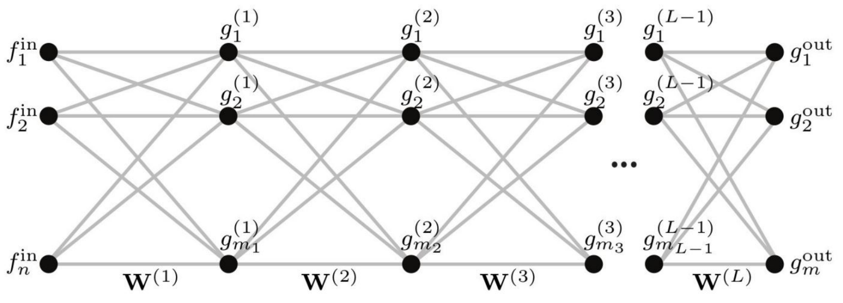


Figure 2.7: Deep neural network consisting of L layers, where g_m represent the neuron perceptron within each layer.

The objective is to have a model that is as universally applicable as possible while still being easy to estimate its parameters to avoid overfitting: **overfitting** occurs when a model learns to fit the training data too closely, capturing noise or random fluctuations rather than generalizing well to unseen data. Since most phenomena are nonlinear, the approach combines a linear function with a nonlinear one (activation function); the output of this process serves as the input to the next layer. This composition is called **multi layer perceptron** or **feed-forward neural network**.

2.5.2 Training Neural Network

Training a neural network consists in finding the value of the weights θ that minimize the loss function in eq: 2.21

$$\ell_\theta = \frac{1}{n}\sum_{i=1}^{n}(y_i - g_\theta(x_i))^2 \tag{2.21}$$

Loss function measures the discrepancy between the predicted values and the expected values. Since generally is not convex, a possible solution is to minimize it using **Gradient Descent Alghorithm**, the fundamental concept of this strategy is to start from a given point in the loss landscape (fig: 2.8) and move towards the direction that leads us to a local minimum, iteratively calculating the next point as follows until a minimum is reached:

$$\theta^{(t+1)} = \theta^{(t)} - \alpha\nabla\ell_{\theta^{(t)}} \tag{2.22}$$

The parameter $\alpha > 0$ also called **learning rate** represent the length of the step done in each iteration, it is an hyperparameter, and its choice is a crucial task, indeed if it is too large there is a risk to overshooting, and if too small there is a slow convergence. Such approach can encounter problems when dealing with large

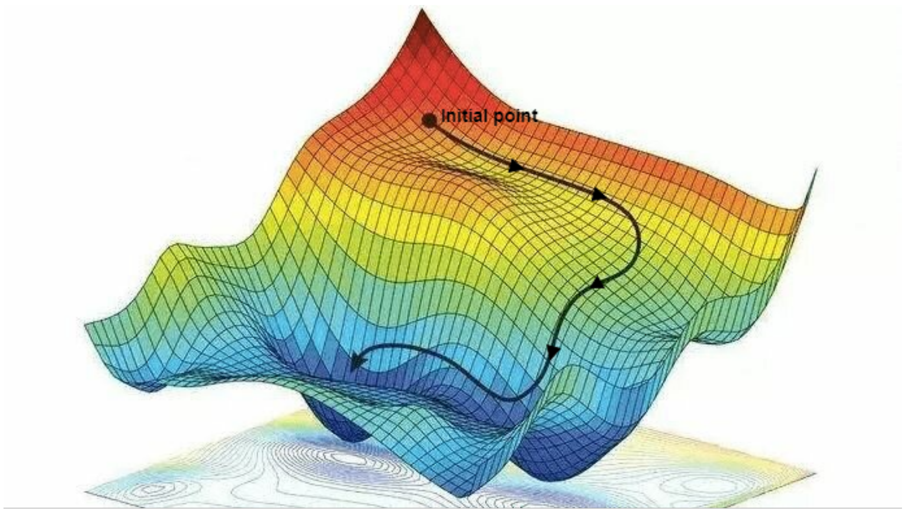


Figure 2.8: Landscape of loss function, taking inspiration from [37].

datasets. To address this, the idea is to extract a small subset from the training set and compute the gradient descent on this subset, called **batch**. Once all batches are processed, an **epoch** is completed, this method is called **Stochastic Gradient Descent**.

When we use feedforward neural networks, every neuron in the network introduces a combination of different functions. In this situation, calculating the gradient could constitute a bottleneck. Finding a method to automate this computation is therefore essential. To achieve this, the idea of a computational graph [38] is presented as fig: 2.9. An acyclic graph that depicts the function $f(x)$ that our model computes is called a computational graph. The graph enables us to create a structure that explains the decomposition of a complex function. In this way, we can calculate the derivative of the complex function by combining the derivatives

$$f(x) = \frac{\log(x + \sqrt{x^2+1})}{x^2} - \frac{\log^3(x + \sqrt{x^2+1})}{\sqrt{x^2+1}}$$

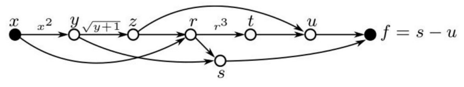


Figure 2.9: Example of computational graph

of the parts into which it has been decomposed. Therefore, the cost of computing the derivative will be equivalent to that of the decomposition. It is possible to calculate the derivative not by starting from the input, but from the output (**backpropagation**). This approach leads to the same results; however, since the number of necessary operations is proportional to the dimensionality of the data, starting from the output allows us to have a more efficient algorithm, indeed inputs are usually multi-dimensional, whereas outputs are typically scalars (e.g the loss value).

2.5.3 Convolutional Neural Network

To introduce certain constraints and enable the construction of a network capable of handling specific data, it is necessary to incorporate priors. Generally, training a neural network is highly complex due to the large number of parameters involved; hence, the use of priors is essential. Each type of data has its own structure. For instance, in images, there are repetitions of elements and patterns that can be utilized as priors to simplify the complexity of the network. The goal is to reduce the number of connections between input and output, which leads to what is known as a convolutional layer. In this layer, each output is connected to a group of inputs that share the same weights across the group. As can be seen from the fig: 2.11,

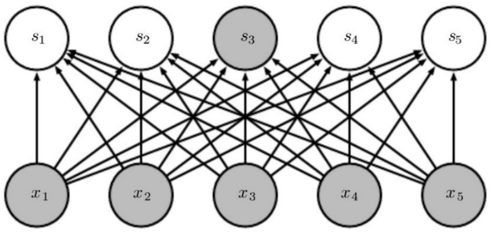


Figure 2.10: Connection of Fully Connected Layers [39].

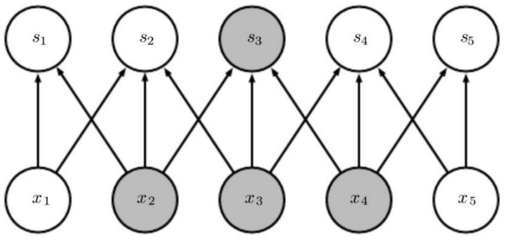


Figure 2.11: Connection of Convolutional Layers [39].

we have groups of three inputs, and there are three weights that are shared among all the groups. This set of weights is referred to as a kernel. In contrast, in a fully connected layer, each input is connected to every output. Given two functions, their convolution is a function whose output is the value of the product of the two functions, shifting one of the functions at each step. Let $f(t)$ and $g(t)$ be two functions. The convolution of f and g, denoted by $(f * g)(t)$, is defined as eq: 2.23

$$(f * g)(t) = \int_{-\pi}^{+\pi} f(\tau)\, g(t-\tau)\, d\tau \tag{2.23}$$

Where:

- $(f * g)(t)$ is the feature map .

- $g(t-\tau)$ is the kernel.

In the case of images, the kernel is a matrix of weights that is convolved over the image tensor, calculating the product with the image tensor's data in that region (see fig: 2.12).

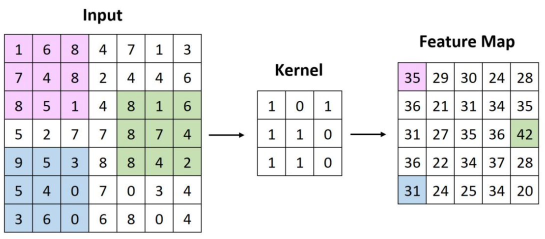


Figure 2.12: Example of convolutional operation [40].

The same kernel is used throughout the entire image, as seen in fig: 2.12. By using this method, fewer parameters are needed and it is guaranteed that the characteristics picked up by the kernel will be identified regardless of where they are in the picture. To add non-linearity after the convolution operation, an activation function is usually used; the ReLu is most frequently utilized. To lower the dimensionality of the output, a pooling layer is placed after the activation function. There are several different kinds of pooling algorithms; one of them is called max pooling, which summarizes the most important characteristics of the tensors by choosing the maximum value from a predetermined region of pixels.

2.5.4 Recurrent Neural Network

In some cases, we may want to model where the input of the previous step affects the output. A particular kind of feed-forward neural network called a **Recurrent Neural Network** (RNN) uses information from the past state to produce the current state. This this translates into a special model that work with two inputs: the input data and the previous state (fig: 2.13).
These networks are specifically used for learning language models (after reading, predicting the next sentence) and event prediction in the context of textual data. The basic RNN cell, also known **vanilla RNN cell**, processes inputs sequentially while preserving a hidden state h that stores the data processed up to that point.t. The hidden state is calculated using eq: 2.24 by applying a non-linear activation function, which is typically *tanh*.

$$h_t = \tanh(W_x x_t + W_h h_{t-1}) \tag{2.24}$$

Where:

- h_t: hidden state at time t

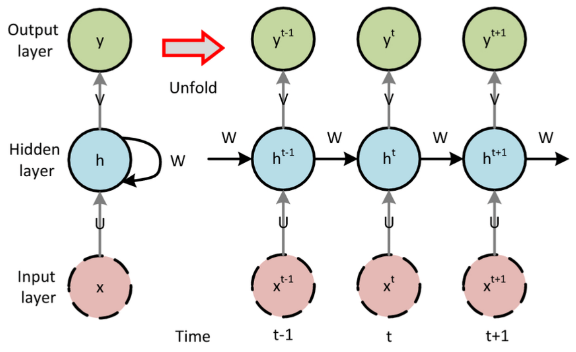


Figure 2.13: RNN computational graph [41]

- x_t: input at time t
- W_x: weight matrix associated to the input x_t
- W_h: weight matrix associated to the hidden state h_t

The vanilla cell is not commonly used in practice, primarily due to issues that arise in the flow of backpropagation [42] (see fig: 2.14):

- The **exploding gradient** problem occurs when gradients of the loss increase untile it become very large during the backpropagation process. Indeed, if the weight matrices W have values greater than 1, the product of these matrices will cause the gradients to increase exponentially with the number of time steps, leading to very large values. This cause loss divergence, with weights of the network that varying in an unstable manner, causing instability during training. The exploding gradient problem can be mitigated by using **gradient clipping**, which insert a limit on the value of the gradient norm during backpropagation.

- The **vanishing gradient** occurs when the gradient of the loss function becomes very small. This is a consequence of the fact that at each step of backpropagation, the gradient is multiplied by the weight matrix W. Indeed if this matrix has values less than 1, the repeated products will cause the exponentially decrease of the gradient, accordingly with the number of time steps. In this situation the model will not able to learn and retain information over long temporal sequences, the only way to solve this issue is to change the architecture to utilize cells that support this type of problem, like LSTM and GRU.

Long-Short Term Memory(LSTM) cell is an improvement of vanilla cell. It prevents the vanish gradients introducing a gating mechanism, it obtain great success in remembering information for long periods of time. The LSTM's gating system consists of four gates:

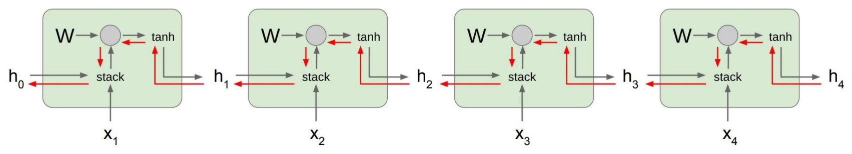


Figure 2.14: Vanilla RNN gradient flow [43].

- **Forget Gate** f: Whether to erase the cell
- **Input Gate** i: Whether to write the cell
- **Output Gate** o: How much to reveal cell
- **Gate Gate** g: How much to write to cell

Unlike the vanilla RNN, the cell is composed of two states: the hidden state h and the cell state c. The different gates are described by the eq: 2.25-2.28

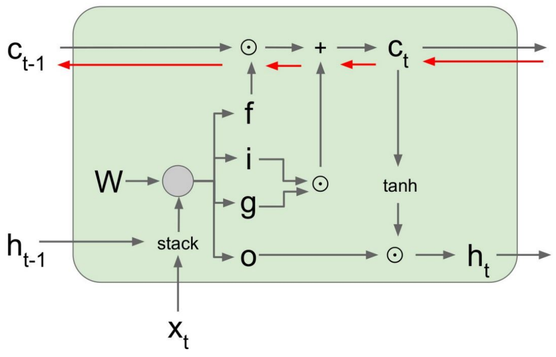


Figure 2.15: LSTM cell [44].

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t]) \tag{2.25}$$

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t]) \tag{2.26}$$

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t]) \tag{2.27}$$

$$g_t = \tanh(W_g \cdot [h_{t-1}, x_t]) \tag{2.28}$$

While the state of LSTM cell is described by the eq: 2.29, 2.30

$$c_t = f_t * c_{t-1} + i_t * g_t \tag{2.29}$$

$$h_t = o_t * \tanh(c_t) \tag{2.30}$$

To compute the cell state c_{t-1} (eq: 2.29) at the timestep t , the process involves the previous cell state c_{t-1} passing through the forget gate f_t, which determines how much of the previous state c_{t-1} we want to forget. Simultaneously, the input gate i_t regulates how much of the new input information x_t we want to add to the cell state.
For the computation of the hidden state h_t (eq: 2.30), the process use the cell state that pass through the output gate o_t, the output gate is delegated to determine how much information to reveal externally.
The reason why the vanishing gradient does not occur in the LSTM cell is due to the fact that during backpropagation from c_t to c_{t-1} (red arrows fig: 2.15) is computed only element wise multiplication by f, in this way the multiplication by the weight matrix W that until now caused the gradient to vanish not happen anymore. This change in architectural enables the LSTM to obtain good results also with larger timesteps, allowing it to maintain and learn dependencies over much longer sequences compared to traditional RNNs.

2.6 Neuromorphic Engineering

Neuromorphic computing, is a computational approach designed to emulate behavior and architecture of biological nervous systems found in our brain. In recent years there has been an increase in the usage of ANN. Neuromorphic computing is gaining more and more traction in the scientific community, as it offers a valid and more efficient alternative to classical ANN models. Neuromorphic computing adopts an innovative approach based on asynchronous events. It differs from traditional models which continuously process information sequentially, in that it is inspired by biological sensors such as the retina or cochlea which only process information when there is a variation in the signal, instead of having a continuous sampling which would be more costly. Signals are only generated when a change occurs, and the signal is referred to as a "spike" . The fundamental components of this model are spiking neurons, which are interconnected to form a Spiking Neural Network (SNN). Each neuron processes spikes independently and remains inactive when no input is present. This alternative method of processing signals is an innovation compared to the classic Artificial Neural Networks (ANNs) models which instead process input signals continuously. Furthermore, this method emulating the functioning of the human brain, allow the user to enhancing efficiency and potentially increasing computational power for specific tasks. However, it is important to highlight some of the limitations associated with these models, particularly during the implementation phase. These challenges introduce many difficulties caused mainly by hardware constraints related to the use of neuromorphic hardware. These limitations should always be given due importance so that the full benefits of neuromorphic technologies can be exploited.

2.7 Spiking Neural Network

Over the past decade, deep neural networks have achieved significant success across various domains. However, these networks are highly resource-intensive, demanding substantial energy consumption, extensive data, and considerable computational costs. The amount of energy consumption is increased during years, at rate of 10 x from 2012 to 2019, example of this can be GPT-3, with 175 billion learnable parameters, with estimated energy consumption around 190 000kwH [45].
Our brain with lot of sensors in input utilizes 10/20 W to process information, if our brains dissipated as much heat as state-of-the-art deep learning models, then natural selection would have wiped humanity.

As mentioned earlier, neuromorphic engineering imitates the behavior of our brain, to works with this branch some requirements are necessary:

- Neuromorphic sensors: Neuromorphic sensors capture only changes in signals rather than recording data at consistent intervals, enabling more efficient and targeted data collection.
- Neuromorphic Algorithms: Algorithms that learn to make sense of spikes, the are known as Spiking Neural Networks.
- Neuromorphic Hardware: Hardware that take inspiration by the human brain, it's designed to make task in brain-like manner. Neuromorphic model simulated in this hardware require less energy and works faster compared to model elaborated on conventional hardware

The main goal is to combine the optimal Artificial Neural Network (ANN), that has already demonstrated good performance in lot of task, with Spiking Neural Network (SNN), that allow us to achieve better performance in terms of energy efficiency and speed.
In fig: 2.16 are reported some examples where energy efficiency and low inference time is a crucial requirement.

2.7.1 Biologically Neuron

Biologically Neurons (or Spiking Neuron) are the essentials cells that are responsible for the proper functioning of the nervous system. These component when are electrically excited generate action potential called spikes. The human and animal nervous system is composed of a set of connected neurons, which communicate through spikes.

As can be seen from the figure 2.17, the neuron is composed mainly of 4 components whose functioning can be summarized as follows:

- **Dendrites**: Are a fundamental part of the neuron, enabling it to receive signals from other neurons, these structures are characterized by an intricate branching, similar to the branches of a tree,they are collocated in the extremities part of the neuron. Each neuron is composed by a lot of dendrites.

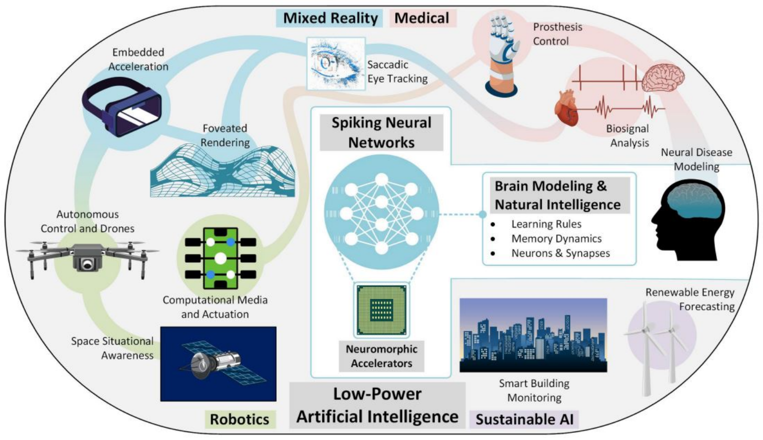

Mixed Reality
Medical
Prosthesis Control
Embedded Acceleration
Saccadic Eye Tracking
Foveated Rendering
Spiking Neural Networks
Biosignal Analysis
Neural Disease Modeling
Brain Modeling & Natural Intelligence
Learning Rules
Memory Dynamics
Neurons & Synapses
Autonomous Control and Drones
Computational Media and Actuation
Renewable Energy Forecasting
Space Situational Awareness
Neuromorphic Accelerators
Smart Building Monitoring
Robotics
Low-Power Artificial Intelligence
Sustainable AI

Figure 2.16: Main fields of use of Spiking Neural Networks [45].

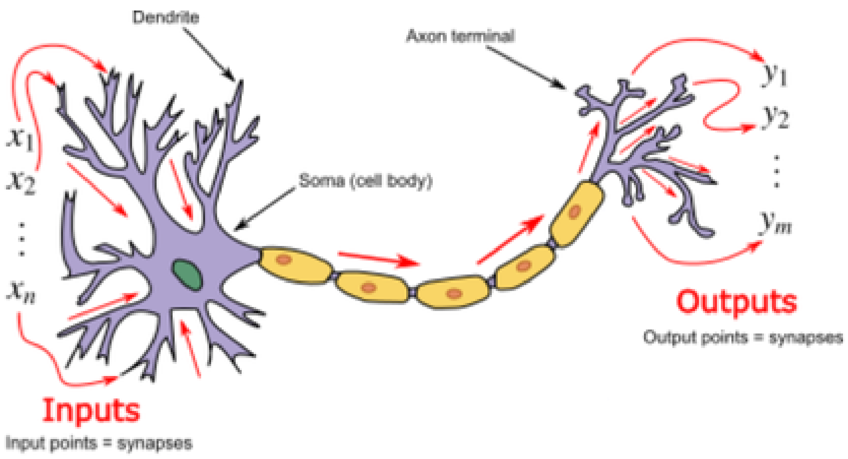


Figure 2.17: Structure of Biological Neuron taking inspiration from [46].

- **Soma**: Is a key component whose task is to receive signals from dendrites, process them and generate a response. Soma (or body) is characterized by a membrane potential, that can be heightened by excitatory inputs or diminished by inhibitory inputs originating from dendrites. Since the membrane potential is not perfectly isolated there may be losses, which cause a decrease in the membrane potential in the absence of input spikes up to a resting value. When membrane potential reach a certain threshold it emit a spike, and the membrane potential turn into the rest value. Choose the right value for this threshold is a crucial task, indeed it ensures that the neuron fires an action potential only when it receives a sufficiently strong signal.

- **Axon**: The primary role of the axon is to conduct electrical signals away from the cell body to other neurons.

- **Axon Terminal**: Terminal part of the axon, it connect it other neuron.

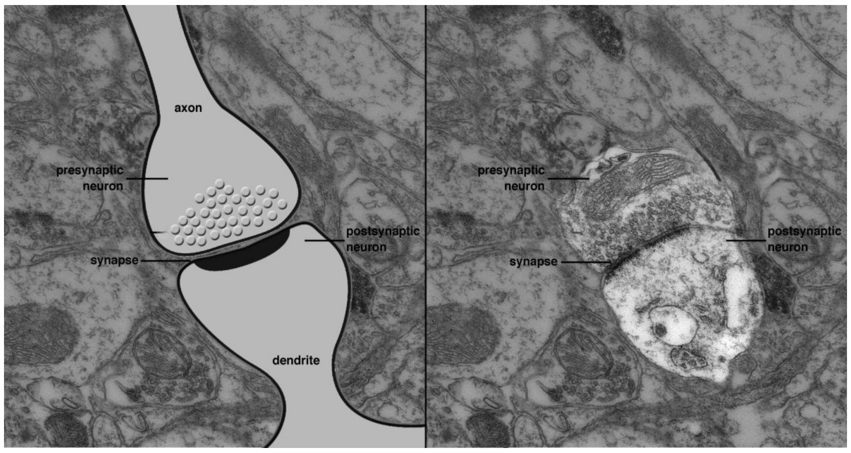


Figure 2.18: Synapse at the electronic microscope, taking inspiration from [47].

The action potential ranging from axons to dendrites manifests as current spikes, in particular these spikes are connected by axons to dendrites through **synapses**, synapses (fig: 2.18) are regions where an axon terminal communicates with a target cell. Within these regions, the presynaptic cell transmits the signal, while the postsynaptic cell receives it. When spike input passes through a synapse (fig: 2.19), the membrane potential of the post-synaptic neuron increases or decreases (depending on whether it is excitatory or inhibitory). The learning process in the brain involves the adaptation of synaptic weights so that neurons can respond more effectively to received inputs, giving more weight to synaptic connections that are more active and giving less weight to those less used. This process of adaptation and change of synapses, called **synaptic plasticity**. For instance: say you want to learn to play a musical instrument, at the beginning the synaptic connections that concern tasks such as finger movement and the reading of staff notes have weights that can vary. Other synapses may have low weights because they are not often used, as we train, the synapse weights that allow us to have a regards (like playing the right note at the right time) will increase, those that are not relevant to our goal will be decreased.

2.7.2 Neural Code

By Neural Code we mean how the human brain represent the information. Despite being the subject of many studies, understand fully Neural Code is even in our days an open challenge. Common point among these theories are "the three S's", where means:

- Spikes: In the brain, the communication between neuron is done by broadcasting trains of action potential (around 100 mV) called spikes. Neurons

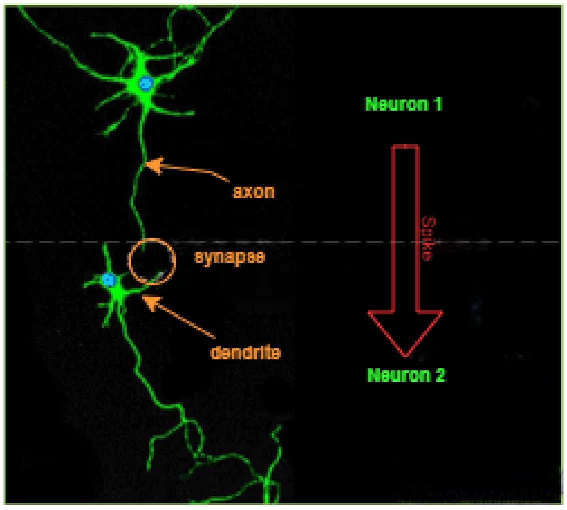


Figure 2.19: The figure shows real neurons stained with a fluorescent substance communicating with each other [48].

communicate through the presence or absence of spikes rather than variations in their intensity. In common artificial neuron, information is represented by high precision numbers, so when performing operation like multiplication is necessary converting them to integer, this introduce delay, in contrast Spiking Neural Network simplifies this process, indeed information is represent by spike, which is a binary information, in this way multiply a weight by a spike reduce the amount of computation.

- Sparsity: Refers to the concept that neurons in the human brain spent major of their time at rest, storing only non-zero elements significantly reduces memory consumption, especially in large datasets or complex models. Various methods are used to represent sparse data, storing only the indices of non-zero positions and their corresponding values.
- Static Suppression: In sensory systems like the visual and auditory systems, neurons exhibit mechanisms that enhance their responsiveness to dynamic and changing stimuli while suppressing responses to static stimuli

2.7.3 Neuron Model

ANN and SNN can model the same type of problem, but their neuron model is different:

- ANN: In the ANN the neuron model take in input the weighted sum of inputs, pass it to an activation function, that is a non linear function like ReLu, this compute the output of the neuron.

- SNN:In Spiking Neural Network, instead of passing the result through a sigmoid or rectified linear unit (ReLU), the weighted sum contributes to the membrane potential $U(t)$ of the neuron. If the neuron is sufficiently excited by this weighted sum and the membrane potential reaches a certain threshold θ, the neuron will emit a spike to its subsequent connections, most of the input are spikes that arrives in different time instant, fig: 2.20 illustrate the behavior.

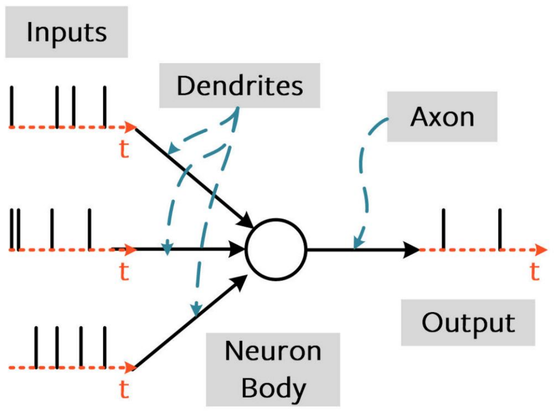


Figure 2.20: Spike Input in Spiking Neuron Model [45].

2.7.4 Encoding/Decoding Spikes

Inspired by the functioning of the brain, spikes were chosen for the input and output of neurons in the spiking neural network. The fundamental question is: “If all spikes are considered equal, how can they carry information?” To address this, two cases must be distinguished (fig: 2.23) :

- Input Encoding: How can we convert input information into spikes?
- Output Decoding: How we can interpret the information of spikes?

Input Encoding

Input encoding is used not only for continuous and sequential data but it is possible to use it in data such as images and treat them as DC signals, there are mainly 3 encoding mechanisms:

- **Rate Coding**:This method converts input intensity into a firing rate or spike count. For instance, if we have an intense input we expect that the encoding generate a higher number of spikes within a given time frame fig: 2.21.

- **Latency Coding**:This method converts input intensity into a spike time. A more intense input would cause the neuron to spike earlier compared to a less intense input fig: 2.22.
- **Delta Modulation**: This method converts temporal changes in input intensity into spikes and remains otherwise silent. In other words, spikes are generated only when there is a change in input intensity, rather than in response to the absolute intensity of the input itself

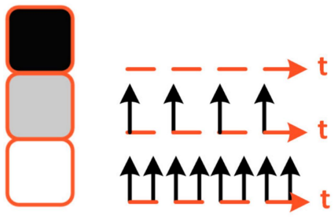


Figure 2.21: Rate-coded input pixel. An input pixel of greater intensity corresponds to a higher firing rate [45].

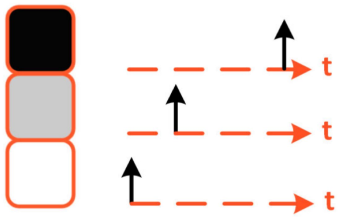


Figure 2.22: Latency-coded input pixel. An input pixel of greater intensity corresponds to an earlier spike time [45].

Previous techniques generally involve "converting" data into spikes. However, it is more efficient to capture data directly in a "pre-encoded" spiking form. This is possible with devices such as Dynamic Vision Sensor (DVS) [49] cameras and Silicon Cochleas [50] that uses delta modulation to record changes in visual or auditory scenes. These devices are able detect variations in their environment and convert them into spikes without the need of additional conversion steps. Dynamic Vision Sensor is an imaging sensor that use the local changes in brightness, in this camera each pixel functionally independently and asynchronously, they report changes in brightness as the occur while remaining inactive otherwise, same operation in Silicon Cochleas that mimic human ear, and use delta modulation when it record changes in auditory sensor.
However, it's important to note that encoding technique are still under development and research. Even though methods such as Rate Coding, Latency Coding and Delta Modulation has demonstrated promising results, there are many challenges to be addressed and improvements to be implemented. The scientific community is working on this topic, for instance the paper [51] presents a comprehensive comparison of various spike encoding techniques for converting time-varying signals into spike trains, using a Spiking Convolutional Neural Network (sCNN) for classification tasks. Furthermore, it proposes a benchmarking pipeline that includes preprocessing steps, feature extraction, and model compression, thereby providing valuable insights for optimizing encoding methods in embedded machine learning applications.

Output Decoding

Regarding the output, we require techniques that enable us to interpret the output spikes:

- **Rate Coding**: The predicted class is represent by the neuron with the highest firing rate.
- **Latency Coding**: Chose the neuron that fires first.
- **Population Coding**: The limitation of processing only two or three spikes within a reaction time of 250 ms is due to the average firing rate of neurons, which is approximately 10 Hz. This means a neuron fires an impulse every 100 ms. Consequently, in 250 ms, a neuron can emit about two or three impulses. To overcome this limitation and achieve more effective and rapid processing, population coding aggregate information from a larger number of neurons.

Rate Codes is more tolerance to error, indeed if a neuron fails to fire there are more spike to reduce this error, using more spikes we have a stronger signal for the backpropagation; on the other hand, Latency Codes produce less spikes in output this implies less dynamic power dissipation in tailored hardware.

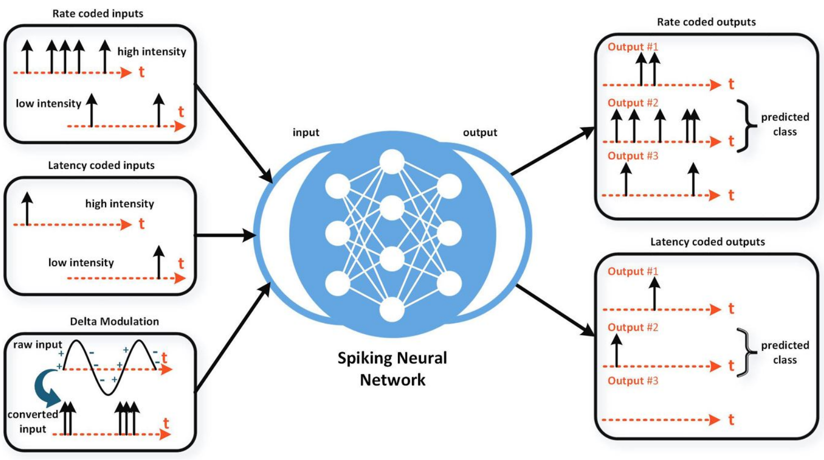


Figure 2.23: Example Strategies for Interpreting Input Spikes and Output Spikes [45].

2.7.5 How to train a Spiking Neural Network

As we saw earlier the output of a neuron has several interpretations, this implies a variety of training methods, each of them must take into account the dead neuron problem: being the Spike out function an Heaviside function centered in the threshold, we have that for point that are different from θ $\frac{\partial S}{\partial U} = 0$, but for $S(t) = \theta$ we have $\frac{\partial S}{\partial U} = \infty$, this introduce a problem in the gradient step, to overcome this problem over the years were introduce different methods which are:

- **Shadow Training**: Instead of training directly in a Spiking Neural Network, what state of the art method propose is to training on a shadow ANN and convert it into a Spiking Neural Network, in this way as well as overcome dead neuron problem, is that conventional deep learning methods can be applied to SNN. Shadow Training is used in task of computer vision, but the usage of ANN during training implies the low training efficiency. Where training efficiency is not important and input data are not time-varying shadow training could be considered the optimal decision, example of Shadow Training can be founded in paper [52].

- **Backpropagation of Spikes in time**: Another possible solution, to train a Spiking Neural Network and avoid Dead Neuron Problem is to take the derivative at spike times, unlike the spikes themselves, which are discontinuous events, spike timings exhibit continuity. This approach present some limitations like for instance the fact that once the neurons become inactive their weights become frozen, and applying solution to fix this can introduce less precision, more example can be founded in paper [53].

- **Surrogate Gradient**: An alternative method to overcome dead neuron problem is the surrogate gradients, this method propose to replace the non differentiable function during backward pass with a differentiable function, surrogate gradient is used in lot of state of the art method such as [54]. As default snnTorch use arcTan function.

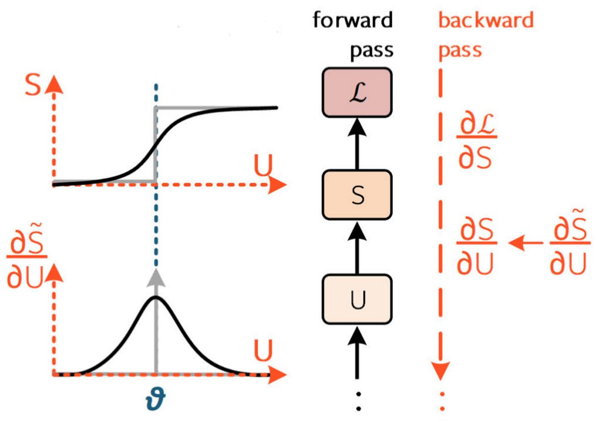

forward pass
backward pass
S
U
$\frac{\partial \tilde{S}}{\partial U}$
U
ϑ
$\mathcal{L}$
S
U
$\frac{\partial \mathcal{L}}{\partial S}$
$\frac{\partial S}{\partial U} \leftarrow \frac{\partial \tilde{S}}{\partial U}$

Figure 2.24: In the context of surrogate gradients, the spike generation function is approximated as a continuous function during the backward pass [55].

2.8 Biological Neuron Models

This chapter explores the evolution of neural network models, following their path from the early binary models that were modeled after McCulloch-Pitts neurons to the more contemporary spiking neuron models that more closely resemble biological neuron dynamics. To further explore how these ideas have influenced the development of more complex computational units, such as the Integrate-and-Fire (IF) and Leaky Integrate-and-Fire (LIF) neurons, we also look at the seminal work of Louis Lapicque, who presented the first mathematical model for neuronal excitability. Lastly, the Izhikevich model renowned for its adaptability in recreating a broad spectrum of neuronal firing patterns is examined.

2.8.1 The Three Generations of Neural Network Models

Artificial neuron can be classified into three categories, according to their computational units. The **first generation** of artificial neurons is based on McCulloch-Pitts neurons [56], which laid the foundation for the development of Artificial Neural Networks. These neurons use binary threshold activation functions, where the output is either 0 or 1, depending on whether the input surpasses a certain threshold. This model contributed significantly to the development of Multilayer Perceptrons, Boltzmann Machines, and Hopfield Networks, marking a key step in the early exploration of neural computation.
Second Generation of artificial neurons introduce activation functions, which are a key element, without it the output of the neuron would be just a linear combination of input, using activation function we can describe also non-linear relationship. An example of non linear activation function is sigmoid function defined as: $\sigma(x) = \frac{1}{1+e^{-x}}$. Typical network of this generation are Feedforward Neural Network, and Recurrent Neural Network. A notable characteristic of second-generation neural networks is their support for learning algorithms based on gradient descent, such as backpropagation. This allows the networks to adjust their weights iteratively to minimize error, leading to more accurate models over time. The second generation of neural networks represents a significant advancement over the first, introducing computational units that employ activation functions. Recent studies have demonstrated that visual pattern analysis and classification can occur in just 100 milliseconds in humans and macaques, while the time required to encode information in artificial neuron of second generation is much greater [57].
This point of criticism prompted researcher to the discovery of the **third generation** [57]. Third generation neurons use **spiking neuron** as computation unit, this model emulate how biologically neuron system encode information, through the timing of spikes. This new way to model artificial neurons model promises not only to increase our knowledge of the functioning of the human brain, but also to develop more efficient and faster neural technologies.

2.8.2 Louis Lapique's Intuition

An important contributions to understanding the neural excitability was given by Lous Lapicque. Lous Lapique, was a French physiologist active in the late 19th and early 20th centuries. Before him, researcher like Luigi Galvani had demonstrated that nerves could be excited electrically, but the stimuli were challenging to control and quantify. A fundamental study was published by Lapique in 1907 [58], in which he introduced a model able to explain the neuronal excitability. This model was based on a simple electrical circuit with a capacitor. Through experiments on frog nerves, Lapicque compared the responses of these nerves to electrical stimulation with the predictions of his model. Lapique stimulated a nerve fiber in a frog's leg using an improvised current source, he observed how long it took for the leg to contract according to the amplitude and duration of the applied current, Lapicque concluded that a neuron generating pulses roughly resembles a low-pass filter circuit in fig: 2.25 composed of a resistor R_M and a capacitor C_M, later referred to as **integrated-and-fire neuron** (IF)

2.8.3 Integrate-and-Fire

Integrate and Fire neuron starts from the assumption that the capacitor representing the membrane potential behaves like an ideal capacitor, therefore is a perfectly isolated capacitor. The Integrate and Fire neuron can be described by the circuit in fig: 2.26, and by the following equation:

$$C_M \cdot \frac{dV}{dt} = I(t) \tag{2.31}$$

By rearranging this expression to determine the temporal evolution of the voltage, we obtain equation:

$$V(t) = \frac{1}{C_M} \int_{t_0}^{t} I(t)\, dt \tag{2.32}$$

The temporal evolution of the membrane potential and the current output in response to both constant current input and spike current input can be observed fig: 2.27 Analyzing the fig: 2.27, in present of constant current in input the membrane voltage increase linearly, until it reach the threshold. Upon reaching it, the voltage turn to rest value and spike in output is emitted. When the neuron is subjected to spike current inputs, each time a spike arrives, the membrane voltage increases in proportion to the synaptic weight associated with that particular spike Each time the voltage of the capacitor exceed a threshold it emit a spike in output hence the name "Integrate and Fire".

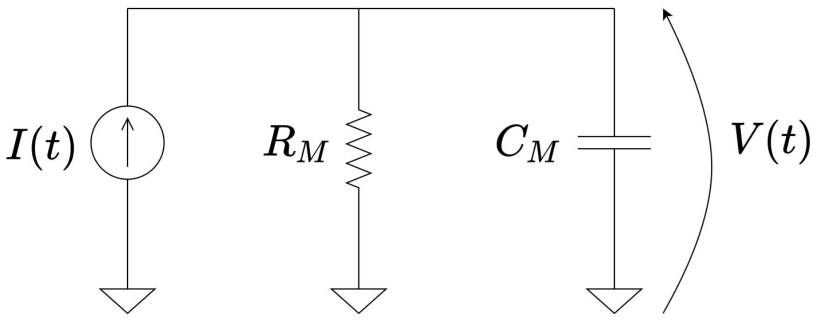


Figure 2.25: Capacitive membrane and resistive form an RC circuit.When the membrane potential exceeds a threshold θ, a spike is generated [45].

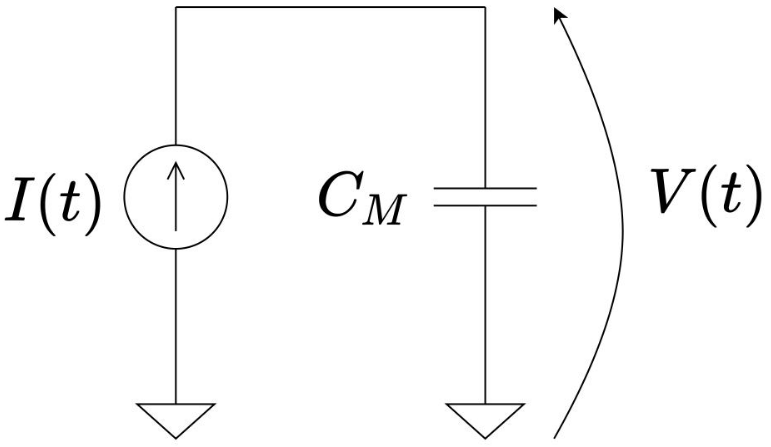


Figure 2.26: Integrate and Fire equivalent circuit.

2.8.4 Leaky Integrate-and-Fire

In the Integrate and Fire model, a very important assumption is made: that the capacitor representing the membrane potential is ideal and lossless. However, in reality, this is not possible. In reality when the capacitor does not receive input stimuli for a defined time will end up being discharged. To model this behavior, the Leaky Integrate-and-Fire (LIF) model incorporates a resistance in parallel with the capacitor of the original Integrate-and-Fire (IF) circuit, as illustrated in the fig: 2.25.
LIF Model can be described by the following equation:

$$C_m \frac{dV_m(t)}{dt} = I(t) - \frac{V_m(t)}{R_m} \tag{2.33}$$

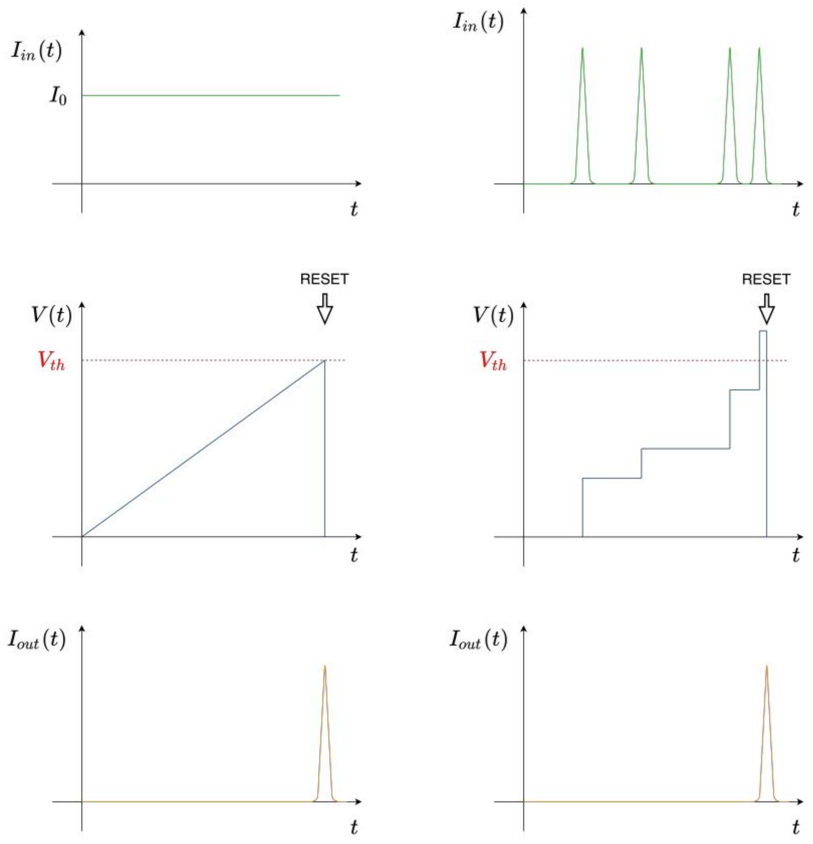


Figure 2.27: Behavior of a IF Neuron with constant current input and current spikes input

The fig: 2.28 illustrates the behavior of the membrane potential and the output based on the incoming inputs. As shown, in the absence of input, the membrane potential decreases over time. The LIF neuron differs from the **Leaky** neuron beacause, in the LIF model, once the membrane potential exceeds a certain threshold, the neuron emits a spike, and the membrane potential is reset. While, in Leaky neuron model, there is no reset of the membrane potential or emission of a spike.

2.8.5 Izhikevich Model

Izhikevich model [59] employs differential equations to describe the behavior of the membrane potential and one recovery variable. Recovery variable is employed to model the behavior of the current that allows the membrane potential to return to rest after a spike. Izhikevich model is described by the eq: 2.34, 2.35

$$\frac{dv}{dt} = 0.04v^2 + 5v + 140 - u + I \tag{2.34}$$

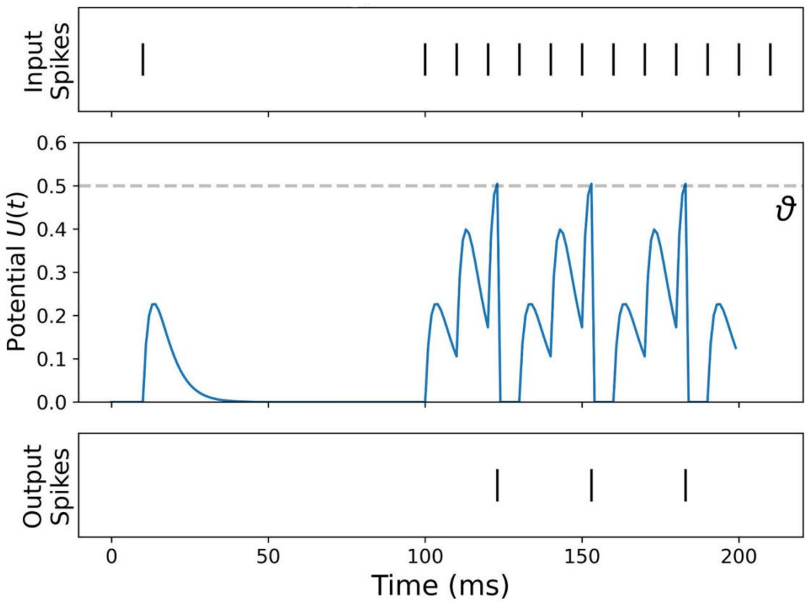


Figure 2.28: Simulation depicting the membrane potential U(t) reaching the threshold, arbitrarily set to $\theta = 0.5V$, which generates output spikes [45].

$$\frac{du}{dt} = a(bv - u) \tag{2.35}$$

Where:

- v potential membrane of Neuron.
- u recovery variable.
- I current in input.
- a, b, c e d parameter that describe neuron behavior.

When spike occurs, variable are upload following the behavior described in eq: 2.36

$$v \geq \theta \ , \quad v \leftarrow c, \quad u \leftarrow u + d \tag{2.36}$$

Where:

- c rest value of membrane potential.
- d variable used to update the recovery variable u after spike.

This type of model is becoming increasingly prevalent in the scientific community, largely due to its capacity to accurately describe any firing pattern of neurons present in the state of the art, provided that the aforementioned parameters are correctly set.

Chapter 3

Materials and methods

3.1 Agrobot Benchmark

In the first phase of this thesis a pre-test step has been done, taking inspiration by the agrobot model on [1]. Authors of this paper propose neural-inertial Extended Kalman Filter able to track position of a robot using IMU and GNSS sensors in a agriculture context, in addition they propose a neural inertial navigation dataset, called agrobot dataset for precision agricultural tracking position, it contains 6.5 hour of recording data for a total distance of 4.5 Km outdoor and indoor.

3.1.1 Architecture

Authors of papers [1] proposed a Neural-Kalman formulation to integrate GNSS with IMU sensors. The approach is based on a neural networks to estimate Kalman filter's state, incorporating accelerator gyroscope and magnetometer data. GNSS measurements are employed for Extended Kalman filter updates. This method effectively combines the high-resolution odometry provided by neural networks with the long-term accuracy of GPS.

Neural Network Model

The reference neural network utilizes inertial data from sensors such as accelerometers, gyroscopes, and magnetometers to estimate position. This data is particularly used for position estimation in the absence of GPS or when the signal quality is bad. At each time step k the network use the predicted speed a along the two axis x and y to estimate the 2D position. The network optimizes the parameters θ by minimizing the Mean Squared Error (MSE) loss based on the velocity. The authors of this model have chosen a specific type of convolutional neural network designed for handling temporal data: Temporal Convolutional Networks (TCNs), In particular, they opted for a network composed by 8 TCN layers with 32 filters, and kernel size 5, follow by a fully connected layer, and finally 2 fully connected one for speed along axis x and the other for speed along y axis. The contribution in this part of the thesis involves replacing the neural network described above, with the neuromorphic model outlined in chapter 3.4, thereby introducing benefits in terms of both efficiency and results.

Neural Kalman Model

The final model is implemented through a Kalman Filter introduced in section 2.4, where the state prediction is influenced by a neural network. Specifically, the **propagate** and **update** steps are governed by the following equations:

$$\hat{x}_{k+1|k} = A\hat{x}_k + g(u_{k+1}) \tag{3.1}$$

$$P_{k+1|k} = AP_kA^T + B_{k+1}U_kB_{k+1}^T \tag{3.2}$$

$$B_{k+1} = \frac{\partial g}{\partial u}\bigg|_{\hat{x}_k, u_{k+1}} \tag{3.3}$$

$$K_{k+1} = P_{k+1|k}H_{k+1}^T \left(H_{k+1}P_{k+1|k}H_{k+1}^T + R_{k+1}\right)^{-1} \tag{3.4}$$

$$\hat{x}_{k+1|k+1} = \hat{x}_{k+1|k} + K_{k+1}\left(z_{k+1} - h(\hat{x}_{k+1|k}, v_k)\right) \tag{3.5}$$

$$P_{k+1|k+1} = \left(I - K_{k+1}H_{k+1}\right)P_{k+1|k} \tag{3.6}$$

$$H_{k+1} = \frac{\partial h}{\partial x}\bigg|_{\hat{x}_{k+1|k}} \tag{3.7}$$

Where x represents the state described by:

$$x = \begin{bmatrix} \hat{L}_x \\ \hat{L}_y \\ v_x \\ v_y \end{bmatrix}$$

u denotes the system input ,L_x and L_y are the predicted displacements along the x and y axes, respectively, and v_x and v_y are the corresponding velocities, A is the state transition matrix:

$$A = \begin{pmatrix} I_{2\times2} & 0_{2\times2} \\ 0_{2\times2} & 0_{2\times2} \end{pmatrix}$$

g is the nonlinear function that captures the complex non-linear relationships between the system's input and its effect on the state, y_θ is the function that represent the predicted speed along the two axis of the neural network:

$$g(\cdot) = \begin{pmatrix} \Delta t \cdot I_{2\times 2} & I_{2\times 2} \end{pmatrix} \cdot y_\theta(\cdot)$$

The covariance matrix P (eq: 3.2) is computed using B and U, where B represents the Jacobian of g with respect to the input, and U is the matrix encapsulating the uncertainties associated with the input.

Regarding the update phase, the Kalman gain is calculated using several key components:

- The predicted covariance matrix P.
- The matrix H, which represents the Jacobian computed from the observation function h with respect to the state x.
- The matrix R, which represents the measurement noise covariance.

The Kalman gain determines the extent to which the measurement influences the update of the state estimate; it quantifies the weight that the measurement should have in adjusting the predicted state estimate. The final state estimate (eq: 3.5) is obtained by correcting the predicted state with the measurement residual, which is defined as the difference between the actual measurement and the predicted measurement. Here, h represents the linear function that map the predicted measurement using the current state estimate, and z denotes the measurements from the GPS. Following this correction, the process covariance matrix P reflects the uncertainty associated with the newly updated state estimate.

3.1.2 Agrobot Dataset

The proposed dataset contains a total of 6.5 hours of IMU sensor recordings. Additionally, there are 2 hours of GPS data. The remaining hours of GPS are synthetically generated. The ground truth is provided with an accuracy of ± 5 cm by a video toolbox capable of identifying a bounding box of the agricultural robot, once this is done, the ground truth position is given by the center of the bounding box. The dataset is divided into three phases fig: 3.1, in each of which the IMU sensors operate at 100 Hz and the GPS at 1 Hz:

- **Phase 1**: Authors mounted Sparkfun Razor IMU sensor on board of the robot, which has been bound to travel in an indoor space of dimension 3m x 2m, it covers around 2.5Km with 3h of dataset, the GPS is synthetically generated.

- **Phase 2**: In this phase they use Bosch BNO055 IMU, and GNSS module for GPS data, this phase was recorded on a rooftop farm of dimension 3.7m x 2.5m. The distance traveled was 1.4 km 2 hours of data were acquired.
- **Phase 3**: In this phase the robot was used on a real farm of strawberry in California, the data were logged for 1.5 hours and 0.6 Km.

As I've mentioned only phase 2 contains GPS data, other phase generates GPS using the ground truth, for this reason we've used this dataset only for preliminary test.

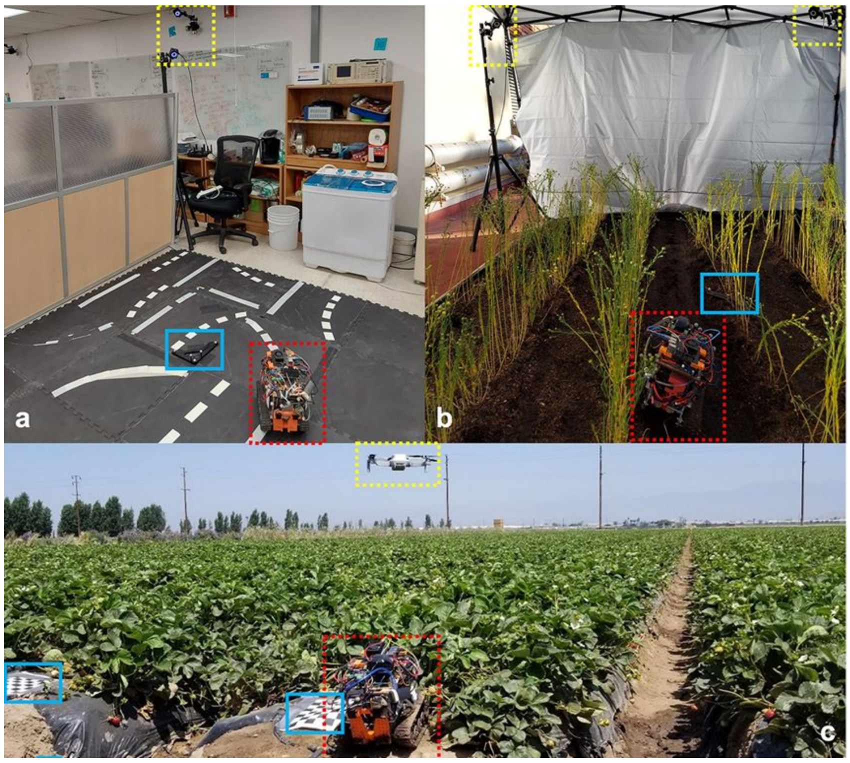


Figure 3.1: Data collection setup for the Agrobot dataset. Dotted red insets show the robot, dotted yellow insets show the ground truth setup, and the solid blue insets show the reference landmarks. (a) Phase 1 (b) Phase 2 (c) Phase 3 [60].

3.2 NCLT Dataset

In most cases, synthetically generated data do not accurately reflect the data obtained from sensors in a real-world context. For this reason, we shifted the next step of the thesis using a dataset that offered a wide range of sensors and data not generated synthetically. University of Michigan North Campus Long-Term Vision and Lidar Dataset [2] (NCLT dataset) is used, the dataset is recorded across various seasons and in differnet weather conditions on a campus in Michigan fig:3.2. The recordings were made using a Segway robot as can be see in fig: 3.3 equipped with multiple sensors, including an IMU, camera, LiDAR, and GPS.

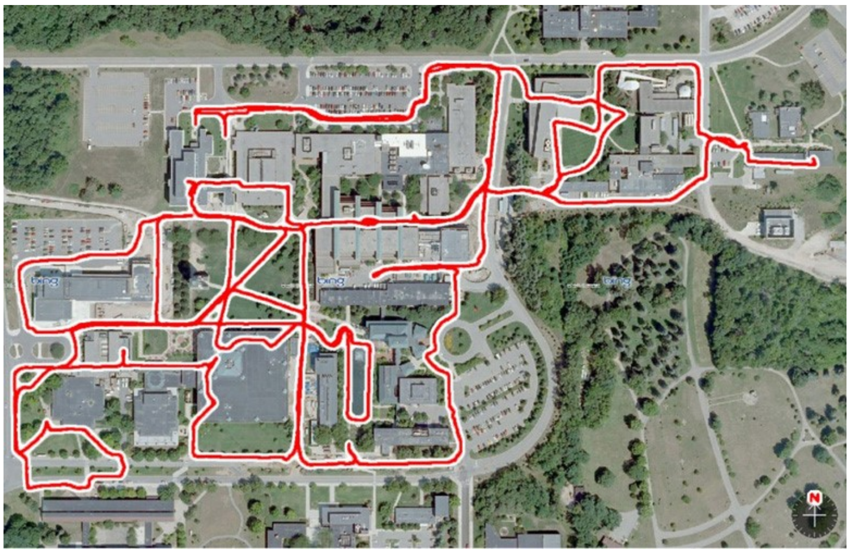


Figure 3.2: Trajectories recorded on Michigan's campus [2].

NCLT Dataset was recorded with different sensors, it support different task such as such as SLAM (Simultaneous Localization and Mapping), dead reckoning, place recognition, and others. The sensors in this dataset are of different price ranges and qualities, thereby providing comprehensive data for different research and application needs. In particular, these latter devices can be summarized as follows:

- Velodyne HDL-32E lidar: LiDAR sensor consisting of 32 lasers with a range of up to 100 meters, operating at a frequency of 10 Hz, it is assembled to operate along the vertical axis.
- Pointgrey Ladybug3 camera: Camera system comprised of six 2-Megapixel cameras, operating at 5 Hz.
- Hokuyo UTM-30LX lidar: Lidar sensor that operates within a 270-degree field

of view at 40Hz. It is horizontally mounted to detect objects and environments parallel to the ground

- Microstrain 3DM-GX3-45 IMU: IMU sensors containing accelerometers, gyroscopes, and magnetometers across all three axes at a frequency of 50Hz.
- KVH DSP-3000 single-axis FOG: Fiber optic single axis gyroscope that provides highly accurate measurements of the vehicle's rotations, operating at a frequency of 100 Hz.
- Garmin 18x 5Hz: Low-quality commercial GPS operating at a frequency of 5 Hz.
- NovAtel DL-4 plus RTK GPS: GPS operating at high quality thanks to the RTK (Real-Time Kinematic) correction system. An RTK base station has been installed on campus to provide the necessary corrections. Outdoors, it is used as ground truth. The system operates at 1 Hz.

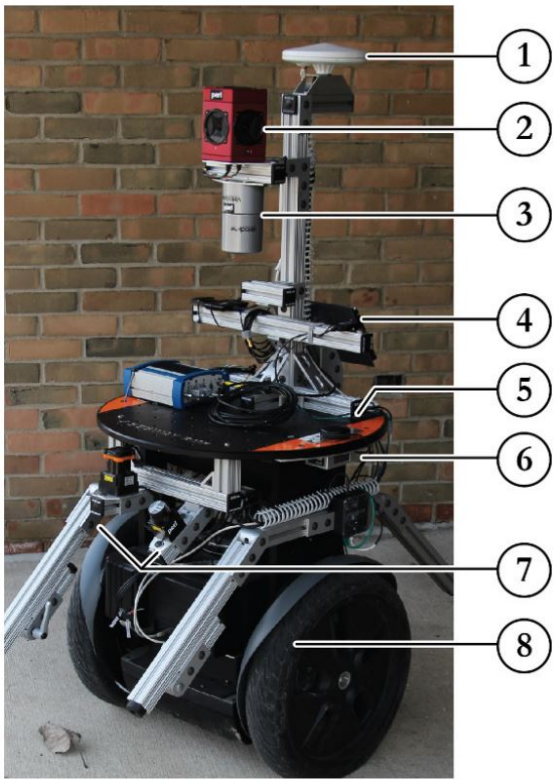


Figure 3.3: The Segway robotic platform used for experimental data collection. Outfitted with an RTK GPS (1), omni-directional camera (2), 3D lidar (3), IMU (4), consumer-grade GPS (5), 1-axis FOG (6), 2D lidars (7), and CPU (8) [2].

In the dataset are presents elements of every day life. Such as bicycles, traffic lights, pedestrians, and automobiles, each of them contributes to have a rich and diverse scenarios in the dataset. Have elements of this type is fundamental to have a robust representation of typical urban scenes and activities. Pre-processing step

Name	Length (Km)	Duration (min:sec)	Foliage	Snow
2012-01-08	6.4	92:16	No	No
2012-01-15	7.5	110:46	No	Yes
2012-01-22	6.1	86:11	No	Yes
2012-02-02	6.2	96:39	No	No
2012-02-04	5.5	78:31	No	No
2012-02-05	6.5	94:17	No	No
2012-02-12	5.8	85:38	No	Yes
2012-02-18	6.2	88:19	No	No
2012-02-19	6.2	88:19	No	No
2012-03-17	5.8	81:51	No	No
2012-03-25	5.8	81:51	No	No
2012-03-31	6.0	81:51	No	No
2012-04-29	3.1	81:51	Yes	No
2012-05-11	6.0	83:36	Yes	No
2012-05-26	6.3	97:00	Yes	No
2012-06-15	4.1	55:10	Yes	No
2012-08-04	5.5	79:27	Yes	No
2012-08-20	6.0	88:44	Yes	No
2012-09-28	5.6	78:01	Yes	No
2012-10-28	5.6	86:06	No	No
2012-11-04	4.8	80:34	No	No
2012-11-16	4.8	81:57	No	No
2012-11-17	4.8	89:41	No	No
2012-12-01	5.0	76:47	No	No
2013-01-10	1.1	17:01	No	Yes
2013-02-23	5.2	80:10	No	Yes
2013-04-05	4.5	69:30	No	Yes

Table 3.1: Data collection sessions [2].

has been done in raw data before the usage of them. This step is crucial to ensure the quality and reliability of the data to avoids anomaly behavior during training of the model. In this phase, all data sequences identified as outliers will be discarded to avoid inconsistencies in the analysis, a data is considered outlier on the basis of parameter such as synchronization between sensors, pose fix of GPS, the distance between the GPS measurement at time t and the ground truth at time $t + 1$. The outcome of this rigorous pre-processing will be a refined dataset, where each sample is structured as follows:

- Initial position at instant t_n .
- Window of 50 measurements obtained from various sensors, ranging from time t to time $t+1$.
- Final position given by the ground truth at instant t_{n+1}, where the ground truth is computed with a SLAM algorithm.

To have a more truthful error measurement we decided to convert the geodetic coordinates in radians (given by the measurements contained in the GPS of the NCLT dataset), into ECEF coordinates, using python library pymap3d [61]. ECEF systems uses uses the Cartesian coordinates (X,Y,Z) to represent position relative to the center of the reference ellipsoid (we assumed the ellipsoid standard WGS84, reference ellipsoid for most GPS on the market), while geodetic coordinates uses the coordinates (lat,lon,h) to represent position relative to a reference ellipsoid. Given N the radius of curvature of the prime vertical section,r the ratio between the semi-minor axis semi-major axis in WGS84 ellipsoid, and the geodetic coordinates in radians the ECEF coordinates, (x,y,z) are determined by the following equations:

$$x = (N + alt) * \cos(lat) * \cos(lon) \tag{3.8}$$

$$y = (N + alt) * \cos(lat) * \sin(lon) \tag{3.9}$$

$$z = (N * r^2 + alt) * \sin(lat) \tag{3.10}$$

3.3 Legendre Memory Unit

The LMU (Legendre Memory Unit) is a specialized type of recurrent cell designed to address the challenges associated with LSTM, particularly the vanishing gradient problem that occurs with long temporal sequences.

3.3.1 Memory Cell Dynamics

The input signal $u(t)$ is intended to be represented orthogonally in the memory of the LMU inside a window of length θ. A form based on Legendre polynomials allows for this orthogonalization. Orthogonality, or the idea that each component of the signal is independent of the others, is a characteristic of legendre polynomials. Ordinary differential equations (ODEs) are mathematical equations that describe how a system evolves over time. The Legendre Memory Unit (LMU) utilizes these ODEs to model the memory cell, the cell is described by the eq: 3.11

$$\theta \dot{m}(t) = Am(t) + Bu(t) \tag{3.11}$$

Where $m(t)$ is a state-vector with d dimensions, A and B represents the ideal state-space matrices determined using the Padé approximants:

$$A = [a_{ij}] \in \mathbb{R}^{d\times d}, \quad a_{ij} = (2i+1)\begin{cases} -1 & \text{if } i < j \\ (-1)^{i-j+1} & \text{if } i \geq j \end{cases} \tag{3.12}$$

$$B = [b_i] \in \mathbb{R}^{d\times 1}, \quad b_i = (2i+1)(-1)^i, \quad i,j \in [0, d-1] \tag{3.13}$$

The memory cell $m(t)$ of the Legendre Memory Unit (LMU) must represent the input signal $u(t)$ over a window of length θ using Legendre (from here derive the name of the cell) polynomials of degree $d-1$. The signal $u(t-\theta_0)$ is approximated as a linear combination of the Legendre polynomials and the memory coefficients m_i, as we can see by the following equation:

$$u(t-\theta_0) \approx \sum_{i=0}^{d-1} P_i\left(\frac{\theta_0}{\theta}\right) m_i(t), \quad 0 \leq \theta_0 \leq \theta \tag{3.14}$$

$$P_i(r) = (-1)^i \sum_{j=0}^{i} \binom{i}{j}\binom{i+j}{j}(-r)^j \tag{3.15}$$

Having the various coefficients m_i in eq: 3.14 independent from each other give us the possibility to have to an improved representation. Indeed in this way each coefficient captures distinct aspects of the signal, thereby avoiding the overlap of information.

3.3.2 Layer Design

The LMU cell takes in input x_t at a given time t and computes the hidden state h_t at that moment. The hidden state is derived from the nonlinear combination of the current input, the memory, and the hidden state from the previous time step;

$$h_t = f\left(W_x x_t + W_h h_{t-1} + W_m m_t\right) \tag{3.16}$$

Where:

- f is a non linear function like tanh.
- W_x, W_h, W_m are the learned kernels.

The input signal u_t in Equation 3.11 is computed using the following equation:

$$u_t = \mathbf{e}_x^T x_t + \mathbf{e}_h^T h_{t-1} + \mathbf{e}_m^T m_{t-1} \tag{3.17}$$

where $\mathbf{e}_x$, $\mathbf{e}_h$, and $\mathbf{e}_m$ are learned encoding vectors, responsible for projecting relevant information from x_t, h_{t-1}, and m_{t-1}, respectively.
The cell architecture can be summarized by the unrolled scheme in fig: 3.4

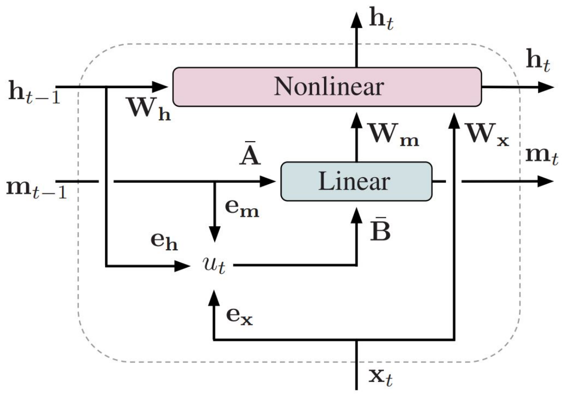


Figure 3.4: Time-Unrolled Legendre Memory Unit [62]

3.3.3 Comparison Legendre Memory Unit with SOTA methods

MNIST dataset consists of images of handwritten digits, each image has a size of 28 $\times$ 28 pixels, representing digits from 0 to 9, so there are 10 labels. The comparision with state of the art model is made with a variant of MNIST, permuted sequential MNIST (psMNIST). This variant is designed for Recurrent Neural Networks (RNNs). In psMNIST, the images has a size of 784x1. The pixels are then provided to the network one at a time the network must recognize the digits and reconstruct the input, to find the correct digits.

The table 3.2 presents the validation and test set results on the psMNIST dataset. The data is reported from the table in [62]. The number of parameter for all models are approximately 165,000, except for LMU that has 102.000 parameters. The LMU has demonstrated itself as the superior model in terms of both accuracy and efficiency, with 102,000 parameters compared to 165,000 in other models. Additionally, it is capable of learning rapidly, requiring only 10 epochs as opposed to the 100 epochs needed by other models.

Model	**Validation**	**Test**
RNN-orth	88.70	89.26
RNN-id	85.98	86.13
LSTM	90.01	89.86
LSTM-chrono	88.10	88.43
GRU	92.16	92.39
JANET	92.50	91.94
SRU	92.79	92.49
GORU	86.90	87.00
NRU	95.46	95.38
Phased LSTM	88.76	89.61
LMU	**96.97**	**97.15**

Table 3.2: Validation and test set results on the psMNIST dataset (data provided by [63]).

3.4 Neuromorphic Legendre Memory Unit (L^2MU)

The final model employed is inspired by L^2MU (LIF-based Legendre Memory Unit) [64], which was initially designed for classification tasks such as Human Activity Recognition. The model has been adapted for a regression task, specifically distance traveled estimation taking in input multi sensors data.

3.4.1 Data structure

To handle the sequential data from the various sensors in the NCLT dataset, a windowing phase was implemented with a window size of 1 second, corresponding to 50 raw data of IMU and a stride of 100 milliseconds. The data used in the model were extracted from the dataset taking in consideration two different contexts: one in which the initial position is provided by the ground truth (GT), this require the model to calculate the distance traveled, and another where the initial position is provided by GPS, necessitating the model to estimate the traveled distance and correct the GPS error. The dataset consists of 27 logs, of which 4 were used for testing, 4 for validation, and the remaining logs for training.

3.4.2 Encoding

The first part of the model consists of an encoding phase. This encoding phase is implemented using a fully connected structure composed by three layers (i.e the output of each neuron is the input of the of all neurons in the next layer).

In the first layer, each input channel (i.e., the input from a specific sensor along a particular axis) is assigned to a population of LIF neurons. Here there is a an increasing of dimension of data and it generates a series of spikes. The spikes from the first layer are then passed as input to the second layer, which also contains LIF neurons. These neurons take the input data from the previous layer's output and construct a more unified representation. Finally the last layer refines the information, ensuring that it is more coherent and give him as a input of the next step to the model. The decay factors and the threshold value for each LIF neuron were determined through a process of hyper parameter optimization through the framework NNI.

3.4.3 Architecture Design

The L^2MU cell in fig:3.5 is developed to redesign the LMU architecture with a neuromorphic approach, the authors of this paper has converted each element into a population of neurons. In the L^2MU, the various states of the LMU architecture (hidden state, memory state, and input representation u) are substituted into neuron populations named m, h, and u respectively. These neuron populations are employed to translate the equations used among the LMU components into neural activity, aiming to mimic human behavior through spike-based communication. The conversion of these interactions was accomplished using the leaky neuron model. In the neuromorphic model of the LMU, equation 3.17 is converted using a population of neurons resulting in eq: 3.18

$$u_{t,\text{curr}} = e_x^T \cdot x_{t,\text{spk}} + e_y^T \cdot h_{t-1,\text{spk}} + e_m^T \cdot m_{t-1,\text{spk}} \tag{3.18}$$

The output will be the input current of the neurons, $u_{t,\text{curr}}$.
Similarly, equation 3.11 is rewritten in eq: 3.19

$$m_{t,\text{curr}} = A \cdot m_{t-1,\text{spk}} + B \cdot u_{t,\text{spk}} \tag{3.19}$$

Where m_{spk} and u_{spk} represent the spiking activities of the memory and input, respectively, determined by the following conditions: $m_{t,\text{mem}} > \Theta^m$ and $u_{t,\text{mem}} > \Theta^u$ Subsequently, the hidden state is calculated by substituting the equation with the following:

$$h_{t,\text{curr}} = W_x \cdot x_{t,\text{spk}} + W_h \cdot h_{t-1,\text{spk}} + W_m \cdot m_{t,\text{spk}} \tag{3.20}$$

with the corresponding spiking activity $h_{t,\text{spk}}$ determined by following spiking condition $h_{t,\text{mem}} > \Theta_h$. The spikes of the hidden state are fed into a fully connected layer, whose output is provided to a population of Leaky neurons. The final output will be determined by the membrane potential of these neurons.

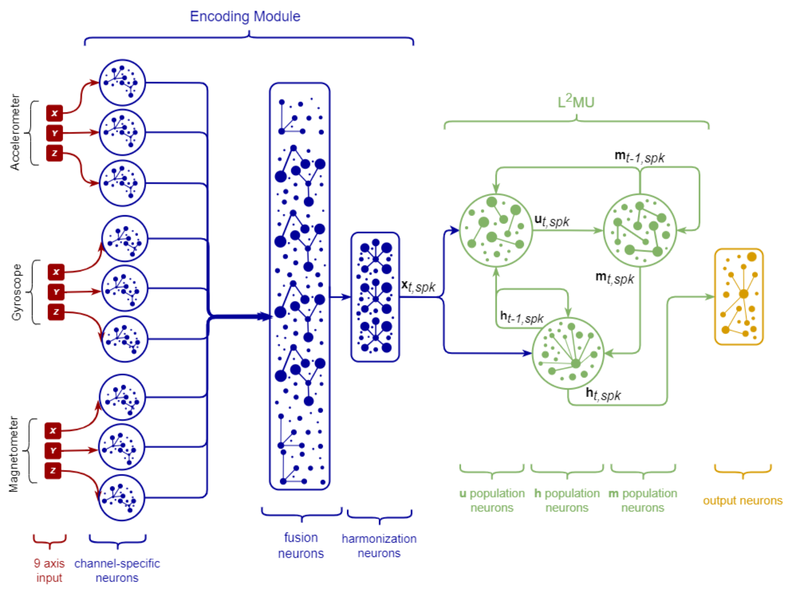


Figure 3.5: Illustration of the neuromorphic model. Each axis channel of the respective sensor has its own input channel. Connected to this input channel is an encoding module composed of a fully connected layer, which communicates with the L^2MU cell through spikes

3.5 snnTorch

The python snnTorch [45] library is used to build Spiking Neural Network. The library aims to integrate neuromorphic models into modern deep learning techniques, offering pre-defined pattern that can be easily used within PyTorch [65] a popular framework for deep learning. snnTorch offers a wide choice of neuron models, ranging from the most used as Leaky and Synaptic (2nd order leaky integrate and fire neuron model accounting for synaptic conductance)but also less traditional as alpha(variant of the leaky integrate and fire neuron where membrane potential follows an alpha function), Lapique and others. Additionally, snnTorch provides surrogate gradient functions to train spiking neural networks, which helps to overcome the spike non-differentiability issue. The package also has utilities for handling datasets and models, tools for arithmetic operations on spikes, data translation to SNN-compatible formats, and tools for spike visualization (fig: 3.6). With these characteristics, snnTorch becomes a comprehensive and integrated PyTorch solution for spiking neural network building and training.

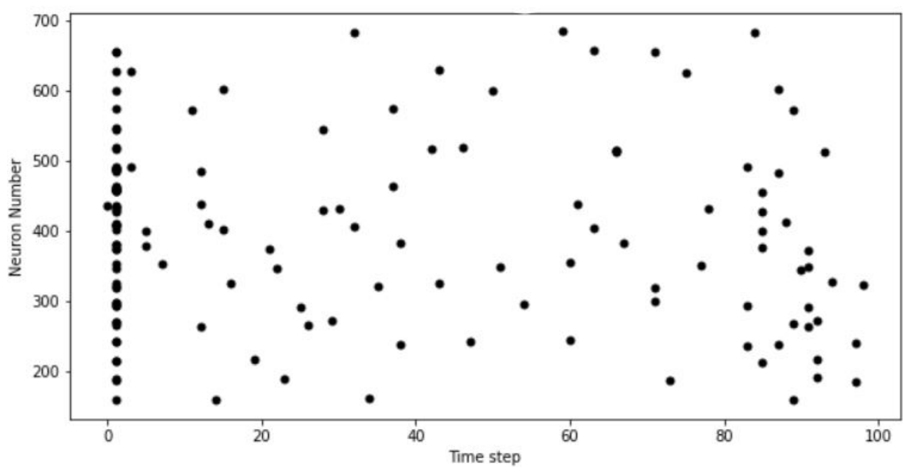


Figure 3.6: Raster plot showing the firing times of neurons at the input of a layer, taking inspiration from plot of [45]

3.6 Hyperparameter Tuning

An hyperparameter is a parameter used during the training process, but it is not determined by it; rather, it is the responsibility of the user to set it. The selection of these hyperparameters is crucial, as they have a significant impact on the final model's performance. However, due to time constraints, it is not possible to perform a complete tuning. The hyperparameter tuning has been done minimizing the validation RMSE loss. When we work with neuromorphic models, there are numerous hyperparameters to consider, such as decay rate and threshold for the various neuron populations. Additionally, the model also includes traditional hyperparameters, such as learning rate and batch size. Since we have a very large searchspace we employed the **Neural Network Intelligence (NNI)** toolkit [66]. NNI is a toolkit developed by Microsoft for automating machine learning (AutoML), the goal is to find the best hyperparameters for machine learning models in efficient way. The way in which the search for hyperparameters is carried out is driven by an algorithm called a "**tuner**", it decides the order in which the various sets of hyperparameters are searched. Within NNI there are several tuners, we opted for a tuner algorithm called annealing, at the beginning, the annealing algorithm starts by sampling randomly the sets of hyperparameters from the search space, as time goes on, the algorithm does not keep sampling randomly, it tends to sample hyperparameters that are closer and closer to the best sets already observed.

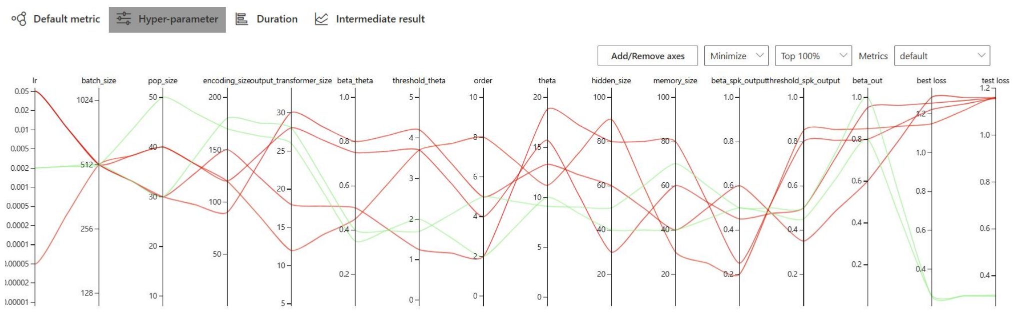


Figure 3.7: The image illustrates the process of hyperparameter optimization. In particular, the diagram demonstrates how NNI effectively explores various combinations of hyperparameters to enhance the model's performance.

Chapter 4

Results and discussion

The results can be considered as divided into two different but closely related use cases.

In the first use case, the Agrobot model [1] is used as a benchmark. This model contains a Temporal Convolutional Network (TCN) that takes as input IMU sensor data for the estimation of the state of an Extended Kalman Filter, whose measurements come from synthetically generated GPS. IMU and GPS data are provided by the Agrobot Dataset [1]. The goal is to obtain a Spiking Neural Network configuration by replacing the TCN model with the L^2MU model. This not only provides gains in terms of the number of parameters and, consequently, the memory used, but also improves the overall results.

In the second use case, a fully Spiking Neural Network model without the use of the Extended Kalman Filter was used, with the objective of reducing the computational cost introduced by the Jacobians of the Extended Kalman Filter. For this use case, we used the NCLT Dataset, in which there are other sensors, as well as IMU, like Wheel Encoders and Optical Gyroscopes.

In the training phase of the second use case, several experiments are carried out involving **GPS refinement** and **displacement prediction**. In the first case, an SNN is trained to predict the displacement from the initial position provided by the GPS while simultaneously correcting the GPS-induced error. An analysis is performed considering only synchronized data, in which it is pointed out how each sensor affects the model's accuracy.

The experiment of displacement prediction is carried out, in which the model is responsible for predicting the displacement from a clean position, i.e., the ground truth. Also here, an analysis is made on data from several synchronized sensors.

Both these models are used in the final test, whose objective is to estimate the position of a moving object from different sensors under real conditions, i.e., with bias, GPS interruptions, and data from different sensors not always being synchronized. In particular, the L^2MU model trained for displacement prediction is used when the initial position from a GPS is not available; otherwise, we use the model trained for GPS refinement.

The training of both models was performed on the CINECA supercomputing infrastructure [67], which provided the necessary computational power to handle the dataset and optimize the Spiking Neural Network models.

4.1 Use Case 1: Agrobot Dataset

The first use case is focused on the replacement of TCN responsible for calculating the state of an Extended Kalman Filter, Agrobot [1] is used as a baseline benchmark. Agrobot is a model able to predict position and speed along 2 axis, in agricolture context, in particular this model employs a Temporal Convolutional Network (TCN), which takes as input accelerometer, magnetometer, and gyroscope data along the three axes. The network's prediction is used as the propagation model for an Extended Kalman Filter, where the state includes velocity and position along the x and y axes, and the update measurement are provided by a GPS measurements generated synthetically. Our contribution allow to have advantages both in terms of accuracy, particularly,in terms of efficiency, reducing drastically the memory required. This was made possible by replacing the Temporal Convolutional Network in the initial Agrobot model, responsible for performing the state propagation of the Extended Kalman Filter, with L^2MU model as can bee see in fig: 4.2, in particular, for this use case, the L^{2}MU model without the encoding module was specifically utilized, as can be seen in the fig: 4.1, where the model receives the IMU data directly as input.

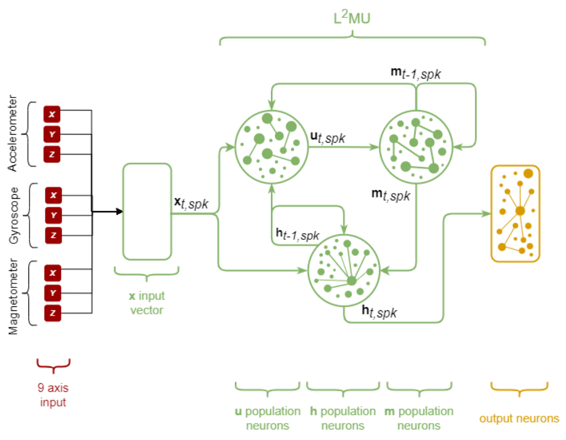


Figure 4.1: In the figure, the architecture of the L^{2}MU model for this use case is illustrated. Notably. It differentiates from fig: 3.5 by the absence of the encoding module. In this case, the model directly receives raw data as input.

The following metrics are used to measure the error between prediction and ground-truth, the ground-truth is computed by a video tool able to identify the bounding box of the interested object, the center point of this bounding box is taken to identify the coordinates:

- **Absolute Trajectory Error (ATE):** RMSE between ground truth and the corresponding point, the error is defined as the distance between the predicted point and the groundtruth point at the same time step i, eq: 4.1

$$E_i = \sqrt{(\hat{x}_i - x_i)^2 + (\hat{y}_i - y_i)^2} \tag{4.1}$$

- **Relative Trajectory Error (RTE):** RMSE between displacement over 60 seconds between model and groundtruth, the error is defined in the eq: 4.2

$$E_i = \sqrt{(\hat{x}_i - \hat{x}_{i+60})^2 + (\hat{y}_i - \hat{y}_{i+60})^2} - \sqrt{(x_i - x_{i+60})^2 + (y_i - y_{i+60})^2} \tag{4.2}$$

- **Distance Relative Trajectory Error (DRTE):** RMSE between displacement over *td* seconds between model and groundtruth, where *td* is defined as the average time to travel 1 meter, the error is defined in the eq: 4.3

$$E_i = \sqrt{(\hat{x}_i - \hat{x}_{i+td})^2 + (\hat{y}_i - \hat{y}_{i+td})^2} - \sqrt{(x_i - x_{i+td})^2 + (y_i - y_{i+td})^2} \tag{4.3}$$

While the Root Mean Square Error (RMSE) is defined in the eq: 4.4

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n} E_i^2} \tag{4.4}$$

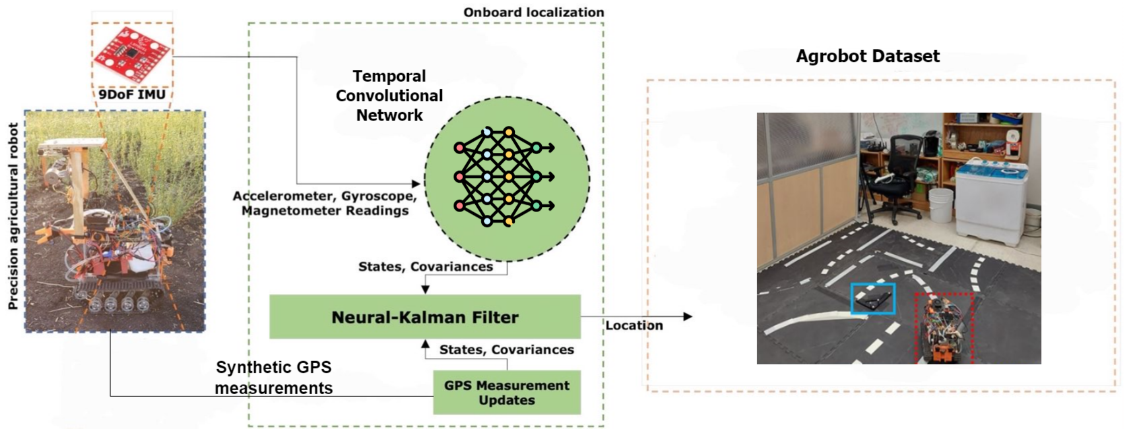
The results of the comparison between Agrobot, which utilizes the Temporal Convolutional Neural Network, and Agrobot, which employs the proposed neuromorphic model, are summarized in Table 4.1.
The results show that the Agrobot (L^2MU) performs better than the TCN model on every metric that was assessed. In contrast to the 2.61 m recorded for the TCN model, the neuromorphic model's ATE is recorded at 1.14 m. This significant improvement is a reflection of the improved trajectory following accuracy of the neuromorphic model.
In addition, the neuromorphic approach's benefits are further highlighted by the RTE, which comes in at 0.71 m as opposed to 5.28 m for the TCN. This significant distinction implies that, even after 60 seconds, the neuromorphic model continues to follow a more accurate trajectory. The neuromorphic model performs better than the TCN model, also DRTE metric, which records 1.94 m compared to 1.07 m for the neuromorphic model.
Not only does the neuromorphic model exhibit improved performance measures, but it also utilizes resources far more efficiently. Compared to the 1.12 MB of the TCN model, its code size is far less, at just 0.048 MB. Additionally, compared

Metric	**Agrobot (TCN)**	**Agrobot (L^2MU)**
ATE (m)	2.61	1.14
RTE (m)	5.28	0.71
DRTE (m)	1.94	1.07
Code Size (MB)	1.12	0.048
Number of Parameter	79.8 k	12.1 k

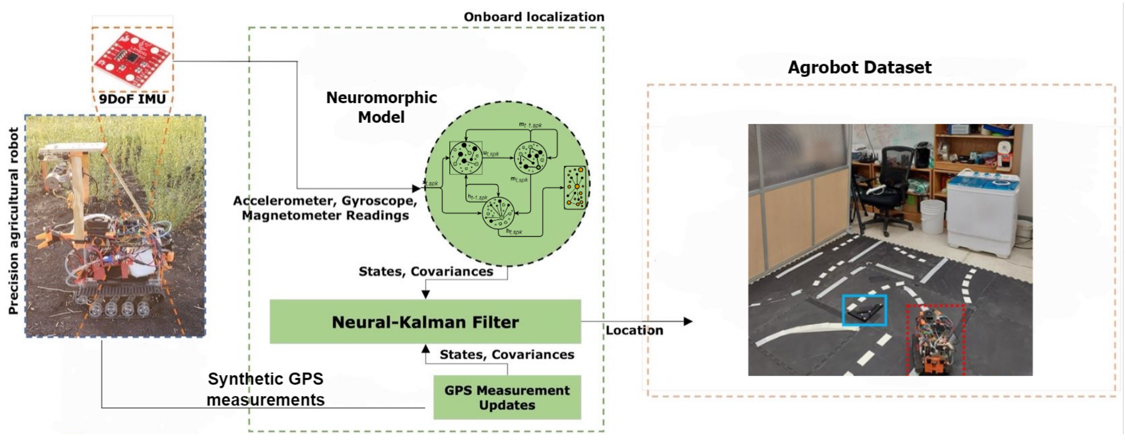
Table 4.1: Comparison between implementation of Agrobot with Temporal Convolutional Network and Neuromorphic Model. Results of Agrobot (TCN) are obtained from the model weights uploaded to the authors' repository.

to the TCN model's 79.8K parameters, the neuromorphic model has 12.1K less parameters.

Since the measures of GPS proposed by the authors of this benchmark [1] are generated in synthetically way, we move our focus on another type of dataset which GPS measurement are provided by real sensors, in a way that model is able to reflect situation of our everyday life.



Agrobot model with Temporal Convolutional Network.



Agrobot model with L^{2}MU model.

Figure 4.2: In the figure above, is presents the initial architecture, while the one below shows our implementation. As can be seen, the architecture remains largely unchanged except for the model responsible for predicting the state of the Extended Kalman Filter, indeed the Temporal Convolutional Network is substituted by the L^{2}MU model.

4.2 Use Case 2: NCLT Dataset

For the second use case of the experiments for this thesis, the NCLT Dataset [2] was used, in view of the fact that it offers a wide range of sensors across various price ranges in everyday life contexts, specifically, GPS data, data provided by IMU sensors, wheel encoders, and an optical gyroscope were utilized to train and test the model.
Since Extended Kalman Filter need to compute Jacobean matrix, it introduce computational additional cost, for this reason to improve energy efficiency we opted

to remove it from the model, and assign the role of predicting the displacement entirely to the neuromorphic model.

4.2.1 Train Model

The model was trained using three configurations: IMU, IMU + Wheel Encoder, and IMU + Wheel Encoder + Optical Gyroscope, as can bee seen in fig: 4.3 - 4.5. For the training phase, the data is structured as follows:

- GPS data at time $t - 1s$.
- Sensor measurements between $t - 1s$ and t, since the frequency of sensor is around 50Hz the window has 50 measurement.
- Ground truth at time t.

The developed spiking neural network is trained with the objective of predicting the final position, starting from the initial position provided by the GPS and using sensor measurements from the different configurations over a 1-second window. The RMSE is used as the loss function. The ground truth is provided by the authors of the referenced paper [2], and it is calculated using a SLAM algorithm that combines

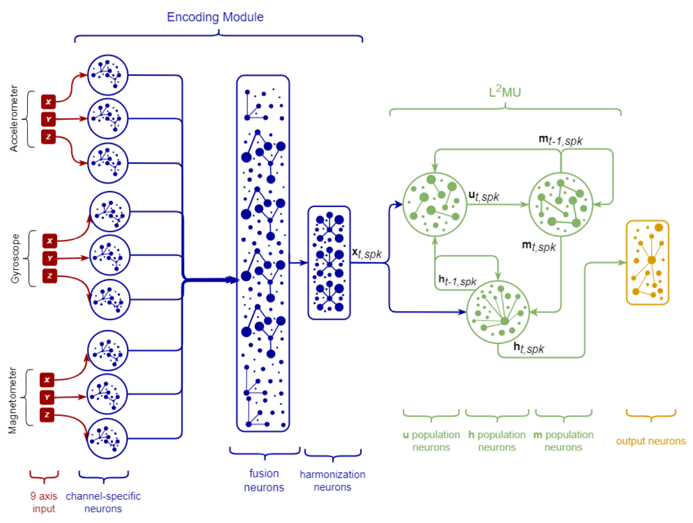


Figure 4.3: Illustration of L^2MU model with **IMU** configuration. Each channel of the respective sensor has its own input channel. Connected to this input channel is an encoding module composed of a fully connected layer, which communicates with the L^2MU cell through spikes

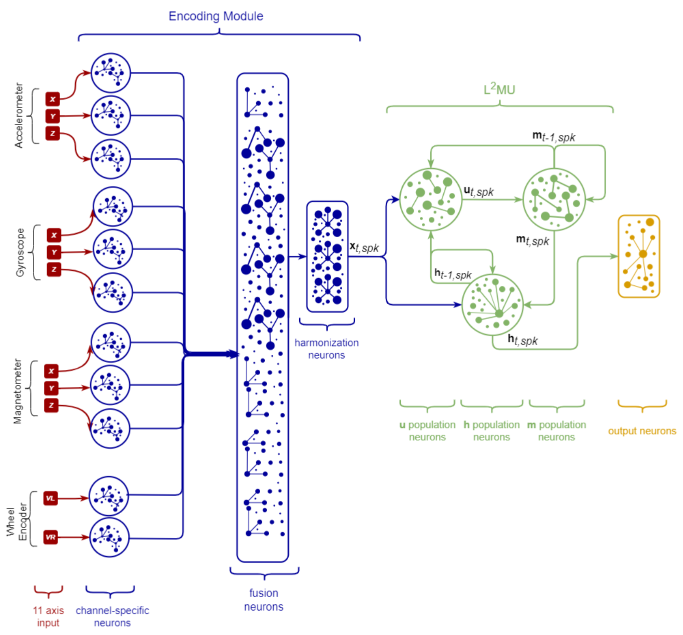


Figure 4.4: Illustration of L^2MU model with **IMU, Wheel Encoder** configuration. Each channel of the respective sensor has its own input channel. Connected to this input channel is an encoding module composed of a fully connected layer, which communicates with the L^2MU cell through spikes

high-quality GPS and LiDAR sensors to determine the position at each moment. Prior to training, a bias removal phase is conducted on the dataset.
A data is considered biased if it meet one of the following requirements:

- The distance in meters between the ground truth at time t and the GPS at time t-1 is greater than 20 meters, as 20 m/s is the maximum speed recorded by the wheel encoder.
- The difference between the time step of the GPS and the first sensor measurement must not exceed 10 ms.
- The difference between the time step of the GPS and the last sensor measurement must not exceed 1.05 s.
- The difference between the time step of the first and last sensor measurements must not exceed 10 ms.

The testing phase is conducted on three logs over three different days. As shown in Table 4.2, the quality of the GPS influence significantly the model accuracy. Since the spiking neural network, which takes inputs from IMU sensors, wheel encoders, and optical gyroscopes, is not able to predict GPS errors arising from factors such as:

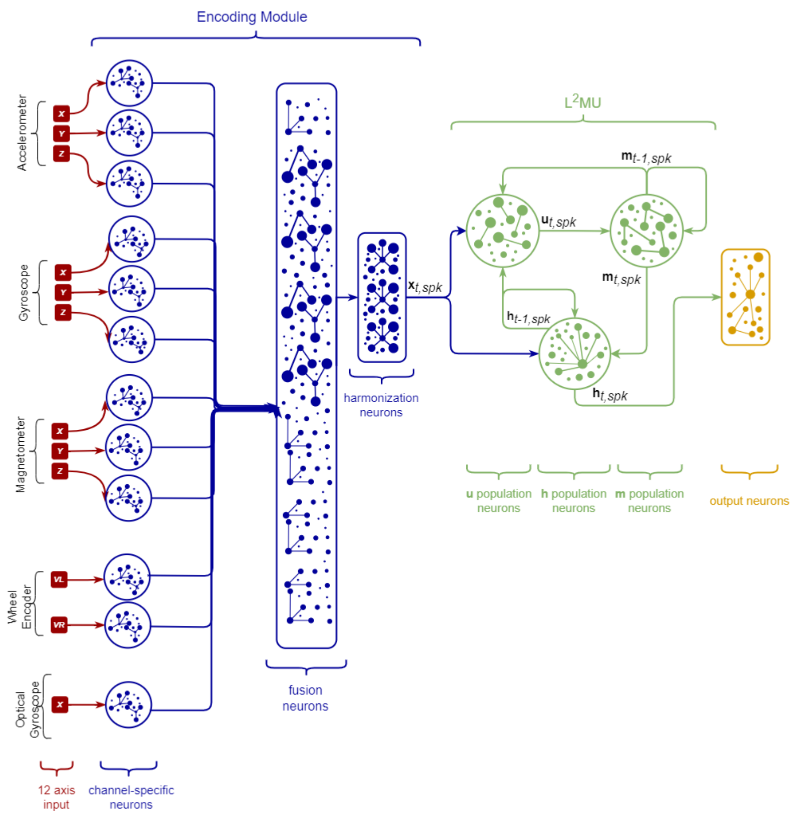


Figure 4.5: Illustration of L^2MU model with **IMU, Wheel Encoder and Optical Gyroscope** configuration. Each axis channel of the respective sensor has its own input channel. Connected to this input channel is an encoding module composed of a fully connected layer, which communicates with the L^2MU cell through spikes

- Signal Arrival C/A: The C/A signal is transmitted by GPS satellites and is subject to delays that can introduce errors.
- Atmospheric Errors: Atmospheric conditions affect the speed of GPS signals as they pass through the atmosphere, especially the ionosphere. Atmospheric errors are minimal when the satellite is directly overhead and increase as the satellite approaches the horizon, due to the longer path the signals must travel.
- Multipath: This issue arises from the reflection of GPS signals off nearby surfaces.

Configuration	**2013-01-10**	**2013-02-23**	**2013-04-05**
GPS Error	5.2 m	5.7 m	6.2 m
RMSE (IMU)	5.10 m	5.7 m	6.15 m
RMSE (IMU+WE+OG)	5.00 m	5.5 m	6.10 m
RMSE (IMU+WE)	5.00 m	5.5 m	6.10 m

Table 4.2: Comparison of RMSE across different sensor configurations and GPS, calculated with respect to the ground truth provided by the NCLT dataset, using the commercial GPS as the initial position.

As previously stated, using a GPS affected by bias as the initial position introduces error into the model's prediction. Therefore, to assess the model's accuracy in predicting movement, an additional training and testing phase is conducted (Table 4.3), replacing the initial position provided by the GPS with the initial position from the ground truth. This approach eliminates GPS-induced errors, providing a clearer understanding of the model's true performance in predicting movement.

Configuration	**2013-01-10**	**2013-02-23**	**2013-04-05**
RMSE (IMU)	0.40 m	0.45 m	0.51 m
RMSE (IMU+WE)	0.11 m	0.15 m	0.18 m
RMSE (IMU+WE+OG)	0.14 m	0.18 m	0.16 m
Mean (IMU)	0.29 m	0.30 m	0.33 m
Mean (IMU+WE)	0.08 m	0.10 m	0.11 m
Mean (IMU+WE+OG)	0.12 m	0.12 m	0.10 m

Table 4.3: Comparison of RMSE across different sensor configurations and GPS, calculated with respect to the ground truth provided by the NCLT dataset, using the ground truth as initial position.

As can be seen from the fig: 4.6 the accuracy of the model in predicting position is significantly affected by the speed of the Segway Robot, indeed, as the mean velocity within test sets rises, the model's accuracy declines. When the speed increases, the ability of the model to make accurate predictions may decrease. This is because at high speeds, even small errors in measurements or accumulated estimates can be amplified, leading to larger deviations from the actual position.

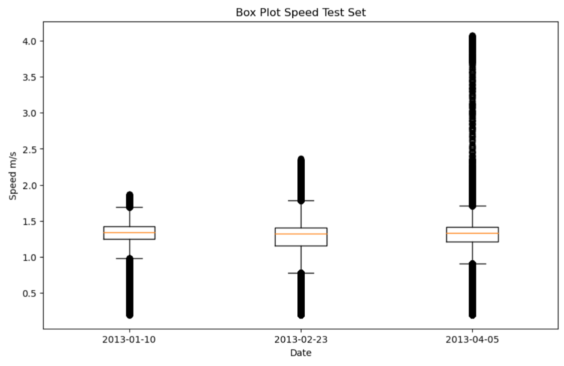


Figure 4.6: Box Plot speed of Segway Robot in test set

4.2.2 Model Performance on Full Dataset: Including Outliers

For the testing phase, data that were previously considered outliers are now also taken into account. In particular, sensor data with synchronization errors, as well as GPS data affected by interruptions and positioning errors, are considered. During the testing phase, while all GPS data are taken into account, only those not originating from interruptions or classified as potential non-outliers are used to determine the initial position. If, due to these issues, the GPS cannot provide the initial position, it is instead derived from the most recent predictions of the spiking neural network (fig:4.7). In more detail, if the initial position is provided by GPS, the L^{2}MU model trained with GPS as the initial position is employed. Otherwise, the L^{2}MU model trained using the ground truth (GT) as the initial position is utilized.
A GPS data is considered outlier if it deviates by more than a certain threshold set at 12m by last prediction of the model, additionally a multivariate time series classification model was trained to detect GPS outlier, the model called MLSTM-FCN [68] integrates a convolutional block with an LSTM block. The architecture includes three convolutional layers designed to extract features from temporal data, each followed by batch normalization and ReLU activation. A squeeze-and-excite block is incorporated into the first two convolutional layers, adaptively recalibrating feature maps based on their relevance, thereby enhancing the model's accuracy. Following the convolutional block, an LSTM layer captures sequential dependencies in the multivariate data. MLSTM-FCN model was trained with a window size of

1000 GPS data, with the aim of detecting if the last GPS data of this window is an outlier, and then tested with the same test set of position predictions, achieving an accuracy of 88%.

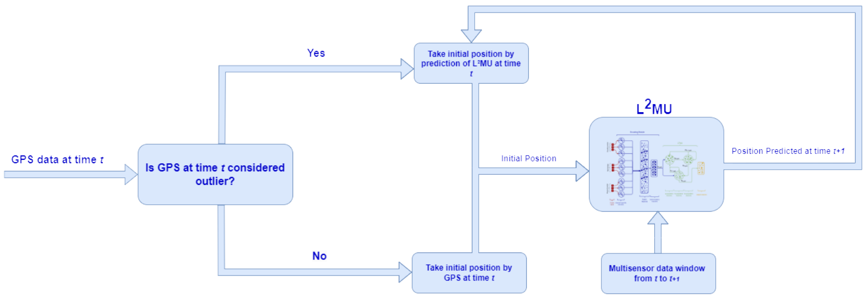

GPS data at time *t*
Is GPS at time *t* considered outlier?
Yes
No
Take initial position by prediction of L²MU at time *t*
Take initial position by GPS at time *t*
Initial Position
L²MU
Position Predicted at time *t+1*
Multisensor data window from *t* to *t+1*

Figure 4.7: The figure illustrates the model architecture used for the testing phase. The initial position for the model's prediction can either be sourced from the latest prediction of the L^{2}MU model in the presence of GPS outliers, in this case L^{2}MU trained with initial position provided by ground truth is used, while if the initial position is provided by GPS L^{2}MU trained with GPS is used.

In this section are presents experiments about how different sensors can estimate the position of a moving object, and the importance of synchronization in multi sensors context. The model was evaluated under three different configuration:

- **IMU**.
- **IMU** combined with **Wheel Encoder (WE)**.
- **IMU** with both **Wheel Encoder** and **Optical Gyroscope (OG)** .

The primary object was to asses the model's ability to predict position both in contexts where initial position can be provided by GPS as well as in contexts where GPS are not available or is affected by bias. Three test set are defined to explore different context, starting from a context in which GPS interruptions or biases are minor, leading up to a context where GPS conditions become critical.
The Table 4.4 shows the results obtained using only IMU sensors operating at 8 Hz. The L^2MU model's RMSE is slightly higher than GPS in the 2013-01-10 test (13.71 m compared to 13.04 m). However, in subsequent tests, the L^2MU model significantly outperforms GPS, particularly in the 2013-04-05 test, where the RMSE is 18.90 m compared to 636.03 m for GPS. This indicates that the L^2MU model delivers greater accuracy than GPS under more challenging conditions. Initial position contribution represent the percentage of the time in which initial position is provided my L^2MU model, and not by GPS.

Metrics and Percentage	**2013-01-10**	**2013-02-23**	**2013-04-05**
RMSE GPS	13.04 m	435.10 m	636.03 m
RMSE L^2MU	13.71 m	25.65 m	18.90 m
Mean GPS	8.15 m	49.52 m	192.22 m
Mean L^2MU	8.91 m	12.34 m	10.06 m
Initial Position Contribution (%)	17.67%	25.7%	23.51%

Table 4.4: Performance metrics, including RMSE and Mean values for GPS and L^2MU, using **IMU** sensors, operating at a frequency of **8 Hz**.

The Table 4.5 presents the results obtained using both IMU and Wheel Encoder sensors at 8Hz. Although the L^2MU model initially shows improved RMSE compared to the IMU-only setup, particularly in the 2013-01-10 log (with an RMSE of 11.91 m), the addition of the Wheel Encoder appears to negatively impact the model's performance in more challenging environments, such as the 2013-02-23 and 2013-04-05 logs. In these cases, the RMSE increases to 57.93 m and 72.45 m, respectively, suggesting that the inclusion of the Wheel Encoder can lead to reduced accuracy when the model is used in harsher conditions

Metrics and Percentage	**2013-01-10**	**2013-02-23**	**2013-04-05**
RMSE GPS	13.04 m	435.10 m	636.03 m
RMSE L^2MU	11.91 m	57.93 m	72.45 m
Mean GPS	8.15 m	49.52 m	192.22 m
Mean L^2MU	8.01 m	21.39 m	27.11 m
Initial Position Contribution (%)	15.3%	16%	27%

Table 4.5: Performance metrics, including RMSE and Mean values for GPS and L^2MU, using **IMU and Wheel Encoder** sensors, operating at a frequency of **8 Hz**.

The Table 4.6 includes results from using IMU, Wheel Encoder, and Optical Gyroscope sensors combined. This configuration produces the best results across all sensor setups. The L^2MU model's RMSE is significantly reduced, with 11.24 m in the 2013-01-10 log and 47.80 m in the 2013-04-05 log, well below the GPS error. Mean values also show improvement, highlighting the system's high precision.

Metrics and Percentage	2013-01-10	2013-02-23	2013-04-05
RMSE GPS	13.04 m	435.10 m	636.03 m
RMSE L^2MU	11.24 m	31.23 m	47.80 m
Mean GPS	8.15 m	49.52 m	192.22 m
Mean L^2MU	7.13 m	15.74 m	20.78 m
Initial Position Contribution (%)	21.84%	25.46%	24.72%

Table 4.6: Performance metrics, including RMSE and Mean values for GPS and L^2MU, using **IMU, Wheel Encoder and Optical Gyroscope** sensors, operating at a frequency of **8 Hz**.

It was decided that additional tests should be conducted at 1 Hz due to the common limitations of commercial GPS units, such as those found in smartphones, which often cannot operate at higher frequencies. These devices are typically constrained to lower frequencies in order to reduce the computational load and conserve energy. The results of the current testing phase, which were collected at 1 Hz, are presented in order to reflect these real-world conditions.
The Table 4.7 reveal a constant performance in the IMU-only configuration, with RMSE values of 12.03 m, 25.65 m, and 22.80 m across the logs from 2013-01-10, 2013-02-23, and 2013-04-05. These figures are similar to the findings at 8 Hz, where all logs have somewhat lower RMSE. For example, the findings for the more complicated situations (2013-02-23 and 2013-04-05) indicate improved accuracy at 8 Hz, although the RMSE for the 2013-01-10 log at 8 Hz is 13.71 m, a slight increase compared to 12.03 m at 1 Hz. This implies that more frequent data points are provided by higher sensor frequencies, which improve accuracy in critical conditions.

Metrics and Percentage	2013-01-10	2013-02-23	2013-04-05
RMSE GPS	13.04 m	435.10 m	636.03 m
RMSE L^2MU	12.03 m	25.65 m	22.80 m
Mean GPS	8.15 m	49.52 m	192.22 m
Mean L^2MU	8.02 m	12.3 m	11.79 m
Initial Position Contribution (%)	32.4%	25.7%	17.42%

Table 4.7: Performance metrics, including RMSE and Mean values for GPS and L^2MU, using **IMU** sensors, operating at a frequency of **1 Hz**.

The L^2MU model's performance with both IMU and Wheel Encoder sensors running at a frequency of 1 Hz is shown in the Table: 4.8. The RMSE figures demonstrate that the L^2MU model's performance degrades at lower frequencies, especially in more difficult situations like the logs for 2013-02-23 and 2013-04-05, which have RMSE values of 57.10 m and 72.81 m, respectively. The RMSE figures for 2013-01-10 (11.91 m), which indicate a less complicated environment, are much lower than these results. Interestingly, there are not much changes in the performance at 1 Hz when comparing the findings with the similar 8 Hz arrangement. For example, the 2013-01-10 RMSE for both frequencies stays at 11.91 m, however the 2013-02-23 and 2013-04-05 logs barely slightly increase the frequency at 1 Hz compared to 8Hz. This suggests that increasing the sampling frequency has little impact on improving the results for this sensor configuration.

Metrics and Percentage	**2013-01-10**	**2013-02-23**	**2013-04-05**
RMSE GPS	13.04 m	435.10 m	636.03 m
RMSE L^2MU	11.91 m	57.10 m	72.81 m
Mean GPS	8.15 m	49.52 m	192.22 m
Mean L^2MU	8.01 m	21.39 m	27.62 m
Initial Position Contribution (%)	15.3%	16.44%	27.23%

Table 4.8: Performance metrics, including RMSE and Mean values for GPS and L^2MU, using **IMU and Wheel Encoder** sensors, operating at a frequency of **1 Hz**.

Performance metrics for L^2MU employing IMU, Wheel Encoder, and Optical Gyroscope sensors at a frequency of 1 Hz are shown in Table 4.9. The RMSE for L^2MU for 2013-01-10 drops from 11.24 m at 8 Hz to 9.18 m at 1 Hz, suggesting a marginal improvement in lower frequency performance. However, the RMSE for L^2MU indicates a considerable rise at 1 Hz (41.52 m and 51.90 m) compared to 8 Hz (31.23 m and 47.80 m) when the conditions grow more difficult on 2013-02-23 and 2013-04-05. This suggest that when operating at a lower frequency, the accuracy of the model decreases more noticeably under unfavorable circumstances.

Metrics and Percentage	**2013-01-10**	**2013-02-23**	**2013-04-05**
RMSE GPS	13.04 m	435.10 m	636.03 m
RMSE L^2MU	9.18 m	41.52 m	51.90 m
Mean GPS	8.15 m	49.52 m	192.22 m
Mean L^2MU	6.48 m	18.41 m	22.43 m
Initial Position Contribution (%)	24.41%	21.10%	20.43%

Table 4.9: Performance metrics, including RMSE and Mean values for GPS and L^2MU, using **IMU, Wheel Encoder and Optical Gyroscope** sensors, operating at a frequency of **1 Hz**.

In the preceding tests, GPS data were considered both indoors and outdoors. Additionally, in Table 4.10 - 4.12 are reported experiments conducted to entirely replace GPS measurements in indoor contexts with the neuromorphic model L^2MU. The objective of this approach is to eliminate the disturbances and biases commonly associated with indoor (see fig 4.8-4.14) environments while evaluating the performance of L^2MU as a reliable alternative to GPS for obtaining accurate information and localizing objects. Based on the results, the model utilizing only IMU sensors (Table: 4.10) demonstrates superior performance compared to those incorporating Wheel Encoders and Optical Gyroscopes (Table: 4.11 and Table: 4.12), exhibiting lower RMSE and mean errors. This underscores the effectiveness of the L^2MU model, particularly when used in conjunction with the IMU sensor, as it consistently outperforms other sensor combinations, reinforcing its potential as a reliable solution for indoor localization challenges.

Metrics and Percentage	**2013-01-10**	**2013-02-23**	**2013-04-05**
RMSE GPS	13.04 m	435.10 m	636.03 m
RMSE L^2MU	6.13 m	10.56 m	21.09 m
Mean GPS	8.15 m	49.52 m	192.22 m
Mean L^2MU	5.33 m	7.02 m	11.14 m
Initial Position Contribution (%)	18.36%	21.93%	24.29%

Table 4.10: Performance metrics, including RMSE and Mean values for GPS and L^2MU, using **IMU** sensors, operating at a frequency of **1 Hz**, where L^2MU **replace GPS in indoor context**.

Metrics and Percentage	**2013-01-10**	**2013-02-23**	**2013-04-05**
RMSE GPS	13.04 m	435.10 m	636.03 m
RMSE L^2MU	22.23 m	53.83 m	77.96m
Mean GPS	8.15 m	49.52 m	192.22 m
Mean L^2MU	10.82 m	21.75 m	29.58 m
Initial Position Contribution (%)	18.36%	21.93%	24.29%

Table 4.11: Performance metrics, including RMSE and Mean values for GPS and L^2MU, using **IMU and Wheel Encoder** sensors, operating at a frequency of **1 Hz**, where L^2MU **replace GPS in indoor context**.

Metrics and Percentage	**2013-01-10**	**2013-02-23**	**2013-04-05**
RMSE GPS	13.04 m	435.10 m	636.03 m
RMSE L^2MU	13.30 m	38.50 m	43.24 m
Mean GPS	8.15 m	49.52 m	192.22 m
Mean L^2MU	7.92 m	16.72 m	18.55 m
Initial Position Contribution (%)	18.36%	21.93%	24.29%

Table 4.12: Performance metrics, including RMSE and Mean values for GPS and L^2MU, using **IMU, Wheel Encoder and Optical Gyroscope** sensors, operating at a frequency of **1 Hz**, where L^2MU **replace GPS in indoor context**.

The figures in 4.8-4.14 illustrate various trajectories of the L^{2}MU model with IMU and GPS. the ground truth trajectory is depicted in blue, the GPS trajectory is represented in red, it reflect good performance when the object is outside. However, as soon as the object enters an area with weak or disrupted GPS signals, such as inside a building, the GPS loses accuracy and may fail entirely. This is typical in situations where GPS signals are obstructed, not only indoors but also in scenarios like tunnels, dense urban areas, or under overhangs.

In contrast, the L^{2}MU model is depicted in yellow and consistently estimates the position accurately. This enables the L^{2}MU model to maintain a reliable estimate of the object's position even without GPS. In particular in fig: 4.10 it's possible to approximately after 600 seconds, the GPS error increases in the x and y axes, which corresponds roughly to the moment when the Segway robot enters the building. At approximately 1500 seconds, as illustrated in fig: 4.13 in particular in y-axes, it can be observed that the model utilizing only the IMU experiences a gradual increment in error. This occurs because, at this moment, the system is located inside the building and cannot obtain an initial position from the GPS. Instead, it relies on the last predicted position provided by the model, which leads to a gradual accumulation of error over time. It's possible to see a similar behavior in fig: 4.14 for x-axes at approximately instant 1500.

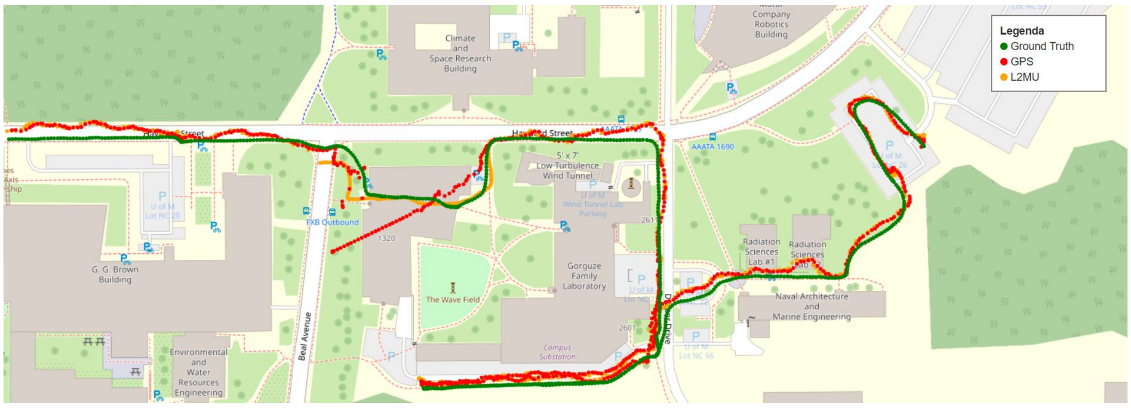


Figure 4.8: Comparison of trajectories on OpenStreetMap using Folium [69] library based on GPS data, Ground Truth (GT), and L^2MU model estimation. Test set from NCLT Dataset [2] log 2013-01-10.

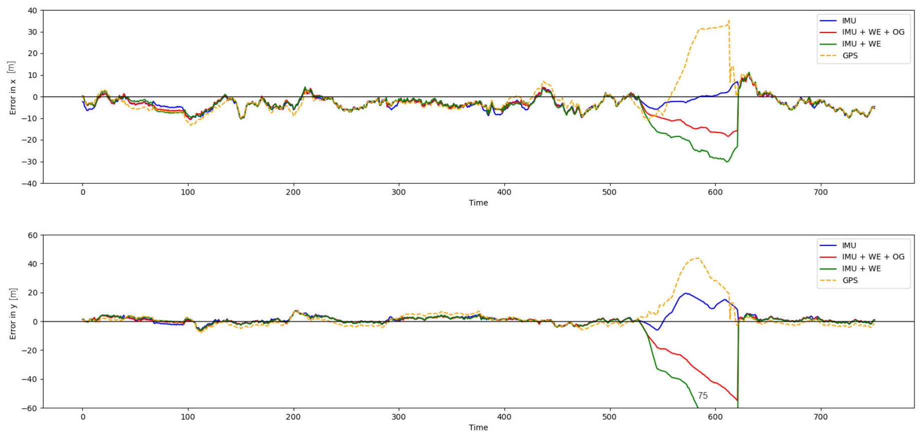


Figure 4.9: Comparison of the error along the x-axis in the first photo, and y-axis in the second, referred to the log 2013-01-10 of NCLT Dataset, between L^2MU using IMU, L^2MU using IMU + WE + OG, L^2MU using IMU + WE, and GPS.

Figure 4.10: Overall comparison of trajectory and position error analysis for the log of 2013-01-10 in the NCLT Dataset.

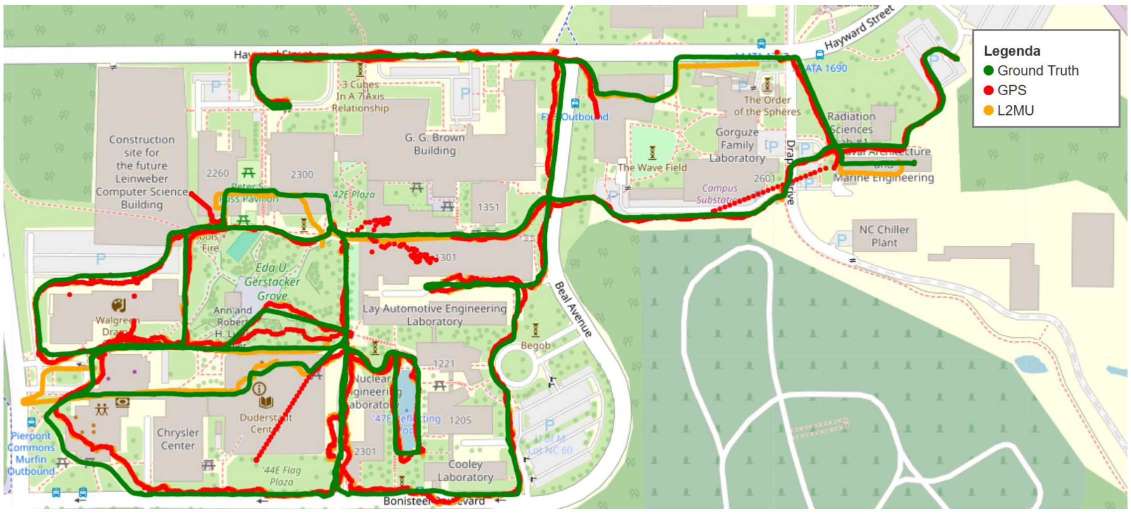


Figure 4.11: Comparison of trajectories on OpenStreetMap using Folium [69] library based on GPS data, Ground Truth (GT), and L^2MU model estimation. Test set from NCLT Dataset [2] log 2013-02-23.

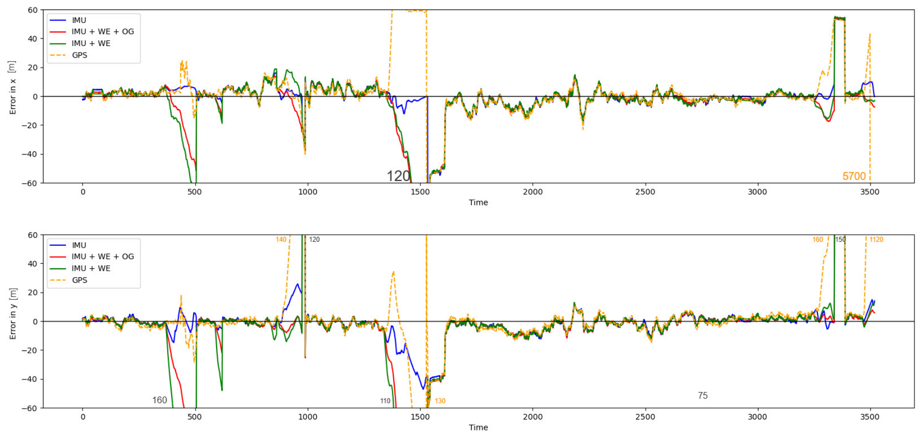


Figure 4.12: Comparison of the error along the x-axis in the first photo, and y-axis in the second, referred to the log 2013-02-23 of NCLT Dataset, between L^{2}MU using IMU, L^{2}MU using IMU + WE + OG, L^{2}MU using IMU + WE, and GPS.

Figure 4.13: Overall comparison of trajectory and position error analysis for the log of 2013-02-23 in the NCLT Dataset.

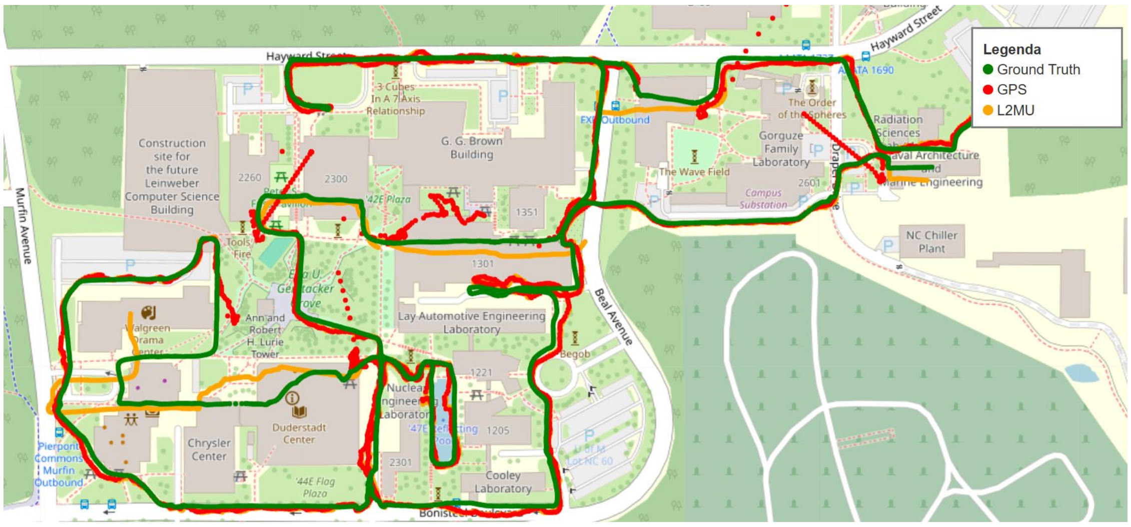


Figure 4.14: Comparison of trajectories on OpenStreetMap using Folium [69] library based on GPS data, Ground Truth (GT), and L^2MU model estimation. Test set from NCLT Dataset [2] log 2013-04-05.

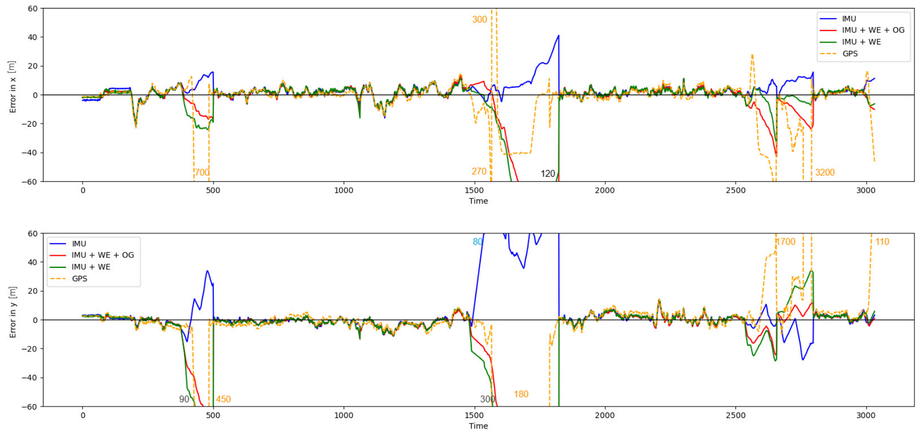


Figure 4.15: Comparison of the error along the x-axis in the first photo, and y-axis in the second, referred to the log 2013-04-05 of NCLT Dataset, between L^{2}MU using IMU, L^{2}MU using IMU + WE + OG, L^{2}MU using IMU + WE , and GPS.

Figure 4.16: Overall comparison of trajectory and position error analysis for the log of 2013-04-05 in the NCLT Dataset.

In Table 4.13, 4.14 below are number of parameters, and size in MB of the respective configurations multi sensors of L^2MU. In Table 4.13, the IMU-only configuration has the highest number of learnable parameters (102000) and a size of 0.41 MB. Adding Wheel Encoder (IMU+WE) reduces parameters to 69600 and size to 0.279 MB, while the full configuration with Optical Gyroscope (IMU+WE+OG) further decreases them to 46700 and size to 0.187 MB. Table 4.14 shows that the IMU configuration has the fewest learnable parameters (37200) and the smallest size (0.183 MB). In contrast, the IMU+WE configuration increases parameters to 56600 and size to 0.227 MB, with the IMU with Wheel Encoder and Optical Gyroscope configuration having the highest parameters (124000) and size (0.497 MB).

Configuration	**Learnable Parameters**	**Model Size (MB)**
IMU	102 k	0.41
IMU+WE	69.6 k	0.279
IMU+WE+OG	46.7 k	0.187

Table 4.13: Comparison of models configuration for displacement prediction with learnable parameters and model sizes.

Configuration	**Learnable Parameters**	**Model Size (MB)**
IMU	37.2 k	0.183
IMU+WE	56.6 k	0.227
IMU+WE+OG	124 k	0.497

Table 4.14: Comparison of models configuration for GPS refinement with learnable parameters and model sizes.

The outcomes yielded during the training and testing phases of the model underscore a pivotal aspect pertaining to the efficacy of a system based on multi-sensor data: the significance of data synchronization.
During the training phase, the model was trained with a filtered dataset, whereby data that was not aligned were discarded. This approach enabled us to ensure the quality of the data during the training phase, where all sensors provided coherent information and were synchronized. In this scenario, the addition of more sensors improved the model's performance, since each synchronized sensor contributed to provide precise and reliable data, allowing the model to learn coherent patterns between the different input sources.
In a real-world context, it is a fundamental to note that not all data from all sensors are always synchronized. As highlighted in paper [70], time synchronization plays a crucial role in multi-sensor systems, as exemplified by the case of a camera and a LiDAR sensor. If the measurements from these sensors are not aligned, it can lead to inaccurate and ambiguous perceptions of the environment, potentially causing catastrophic outcomes. Similarly, in circumstances where disparate sensors are simultaneously acquiring data, temporal inconsistency may arise due to the latency

of the sensor, the variability of the sampling interval, or the presence of interference. The lack of synchronization can have a significant impact on the performance of the model. Ultimately, these experiments emphasized a crucial aspect of multi-sensor models: the synchronization of data is essential to ensure the model's functionality and the accuracy of its forecasts, even in more complex contexts. In the absence of proper synchronization, the performance of the model can deteriorate significantly, irrespective of the quality of the training data.

Chapter 5

Conclusion

With the work done in this thesis, we aimed to demonstrate how Spiking Neural Networks (SNNs) offer a valid alternative to conventional Deep Learning architectures for classic engineering tasks. In particular, we focused our research on dead reckoning and GPS refinement.

One of the main contributions of this thesis was the adaptation of an SNN used in previous studies for classification problems to a regression problem. The model in question was subsequently used for two use cases. In the first use case, it involved the replacement of a Temporal Convolutional Network (TCN) responsible for estimating the state of an Extended Kalman Filter with the SNN adapted to regression problems. This change led to a performance increase of 50% in accuracy and approximately 90% in memory usage compared to the state of the art.

In a second use case, I demonstrated the feasibility of improving GPS measurements using an SNN with various configurations for multi-sensor integration. In this scenario, one network was trained for displacement prediction, while the other was trained for GPS refinement. In this case, the neuromorphic network introduced advantages over the use of GPS alone, achieving improvements of up to 98% in critical conditions and up to 50% in less critical contexts.

One interesting element that emerged from these experiments is the importance of data synchronization. It was observed that in situations where the integration of multiple sensors concerned only synchronized data, this led to an improvement in position estimation of up to 80%, while the use of multiple sensors with misaligned data tended to degrade performance.

Future work should focus on implementing these algorithms on dedicated architectures, using NIR systems [71] to deploying on multiple hardware architecture. These systems would fully leverage the parallel computing capabilities and energy efficiency offered by neuromorphic systems, making them even more practical for real-world applications.

In conclusion, while we continue to push the limits of traditional computing technologies, neuromorphic computing is strategically positioned to play a crucial role in the future of AI and IoT. Indeed, it has the potential to offer solutions to the physical limits that are being reached with traditional architectures.

Acknowledgements

Concludendo questo importante capitolo della mia vita, desidero ringraziare tutte le persone che hanno contribuito alla realizzazione di questa tesi.
Desidero esprimere la mia sincera gratitudine ai miei relatori, Gianvito Urgese e Vittorio Fra, per il loro supporto e i preziosi consigli forniti durante questo percorso.
Grazie a mia madre, sempre pronta a fare tutto per me, rinunciando anche a qualcosa di personale pur di assicurarmi il meglio. Non potrò mai ringraziarla abbastanza per tutto ciò che ha fatto.
Grazie di cuore a mio padre, che mi ha insegnato il valore del sacrificio e l'importanza della dedizione per raggiungere i propri obiettivi. I suoi insegnamenti mi hanno guidato durante tutto il mio percorso e mi hanno mostrato cosa significa lavorare duramente per qualcosa di importante nella vita. La sua saggezza mi ha sempre spronato a dare il massimo.
Un ringraziamento speciale a mia sorella, che, anche se non ci sentiamo ogni giorno, è sempre presente nei momenti più importanti. So di poter contare su di lei in qualsiasi situazione.
Un ringraziamento ai miei zii, che hanno sempre avuto fiducia in me e mi hanno sostenuto in ogni momento del mio percorso. Il loro incoraggiamento incondizionato è stato fondamentale per il mio percorso.
Un ringraziamento speciale ai miei cugini, che sono sempre stati una spalla su cui poter contare.
Voglio ringraziare i miei coinquilini e amici, che hanno reso questo percorso molto più leggero. Hanno mostrato un supporto e un calore tali da far sentire la casa come una seconda famiglia. Grazie a loro, le sfide sono state più facili da affrontare e i momenti di gioia sono stati condivisi insieme.
Un ringraziamento speciale va alla mia ragazza Marica, che da quando è al mio fianco è diventata un sostegno costante. Ha sempre avuto cura di ogni mio stato d'animo, trovando le parole giuste per spronarmi e aiutarmi a dare il meglio di me. La sua fiducia in me ha avuto un impatto profondo sulla mia vita.

We acknowledge a contribution from the Italian National Recovery and Resilience Plan (NRRP), M4C2, funded by the European Union – NextGenerationEU (Project IR0000011, CUP B51E22000150006, “EBRAINS-Italy”).
We acknowledge the EuroHPC Joint Undertaking for awarding this project access to the EuroHPC supercomputer LEONARDO, hosted by CINECA (Italy) and the LEONARDO consortium through an EuroHPC.

Bibliography

[1] Yayun Du et al. «Neural-kalman gnss/ins navigation for precision agriculture». In: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2023, pp. 9622–9629.

[2] Nicholas Carlevaris-Bianco, Arash K Ushani, and Ryan M Eustice. «University of Michigan North Campus long-term vision and lidar dataset». In: *The International Journal of Robotics Research* 35.9 (2016), pp. 1023–1035.

[3] Catherine D Schuman et al. «Opportunities for neuromorphic computing algorithms and applications». In: *Nature Computational Science* 2.1 (2022), pp. 10–19.

[4] Steve B Furber et al. «The spinnaker project». In: *Proceedings of the IEEE* 102.5 (2014), pp. 652–665.

[5] Johannes Schemmel et al. «A wafer-scale neuromorphic hardware system for large-scale neural modeling». In: *2010 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE. 2010, pp. 1947–1950.

[6] Charlotte Frenkel et al. «A 0.086-mm^2 12.7-pJ/SOP 64k-synapse 256-neuron online-learning digital spiking neuromorphic processor in 28-nm CMOS». In: *IEEE transactions on biomedical circuits and systems* 13.1 (2018), pp. 145–158.

[7] Jing Pei et al. «Towards artificial general intelligence with hybrid Tianjic chip architecture». In: *Nature* 572.7767 (2019), pp. 106–111.

[8] Saber Moradi et al. «A scalable multicore architecture with heterogeneous memory structures for dynamic neuromorphic asynchronous processors (DYNAPs)». In: *IEEE transactions on biomedical circuits and systems* 12.1 (2017), pp. 106–122.

[9] Mike Davies et al. «Loihi: A neuromorphic manycore processor with on-chip learning». In: *Ieee Micro* 38.1 (2018), pp. 82–99.

[10] Jeferson Menegazzo and Aldo Von Wangenheim. «Road surface type classification based on inertial sensors and machine learning: A comparison between classical and deep machine learning approaches for multi-contextual real-world scenarios». In: *Computing* 103.10 (2021), pp. 2143–2170.

[11] Paolo Cudrano et al. *Continual Cross-Dataset Adaptation in Road Surface Classification.* 2023. arXiv: `2309.02210 [cs.CV]`. URL: `https://arxiv.org/abs/2309.02210`.

[12] Prashant Beldar et al. «Traveler Guide using GPS». In: *International Journal of Computer Science and Mobile Computing* 3.2 (2014), pp. 406–409.

[13] Maged N Kamel Boulos and Stephen P Yang. «Exergames for health and fitness: the roles of GPS and geosocial apps». In: *International journal of health geographics* 12 (2013), pp. 1–7.

[14] Stephen C Brown, Shannon Crum, and V Stuart Foote. «GIS and GPS emergency response lessons learned from the space shuttle Columbia disaster». In: *The Journal of Extension* 41.4 (2003), p. 17.

[15] Alex Souza Bastos and Hisashi Hasegawa. «Behavior of GPS signal interruption probability under tree canopies in different forest conditions». In: *European Journal of Remote Sensing* 46.1 (2013), pp. 613–622.

[16] Wenhong Wu et al. «LIO-fusion: Reinforced LiDAR inertial odometry by effective fusion with GNSS/relocalization and wheel odometry». In: *IEEE Robotics and Automation Letters* 8.3 (2023), pp. 1571–1578.

[17] Vittorio Fra et al. «Human activity recognition: suitability of a neuromorphic approach for on-edge AIoT applications». In: *Neuromorphic Computing and Engineering* 2.1 (2022), p. 014006.

[18] Matthias Neges et al. «Combining visual natural markers and IMU for improved AR based indoor navigation». In: *Advanced Engineering Informatics* 31 (2017), pp. 18–31.

[19] Hongliang Ren and Peter Kazanzides. «Investigation of attitude tracking using an integrated inertial and magnetic navigation system for hand-held surgical instruments». In: *IEEE/ASME Transactions on Mechatronics* 17.2 (2010), pp. 210–217.

[20] *rwr.ethz.ch.* `https://rwr.ethz.ch/slides/7_Slides_Control.pdf`. [Accessed 23-09-2024].

[21] Martin Brossard, Axel Barrau, and Silvère Bonnabel. «AI-IMU dead-reckoning». In: *IEEE Transactions on Intelligent Vehicles* 5.4 (2020), pp. 585–595.

[22] Kojiro Takeyama et al. «Improvement of Dead Reckoning in Urban Areas Through Integration of Low-Cost Multisensors». In: *IEEE Transactions on Intelligent Vehicles* 2.4 (2017), pp. 278–287. DOI: `10.1109/TIV.2017.2767825`.

[23] Changhao Chen et al. «Ionet: Learning to cure the curse of drift in inertial odometry». In: *Proceedings of the AAAI Conference on Artificial Intelligence.* Vol. 32. 1. 2018.

[24] Changhao Chen et al. «Deep-learning-based pedestrian inertial navigation: Methods, data set, and on-device inference». In: *IEEE Internet of Things Journal* 7.5 (2020), pp. 4431–4441.

[25] Mahdi Abolfazli Esfahani et al. «AbolDeepIO: A novel deep inertial odometry network for autonomous vehicles». In: *IEEE Transactions on Intelligent Transportation Systems* 21.5 (2019), pp. 1941–1950.

[26] Michael Burri et al. «The EuRoC micro aerial vehicle datasets». In: *The International Journal of Robotics Research* 35.10 (2016), pp. 1157–1163.

[27] Ruipeng Gao et al. «Glow in the dark: Smartphone inertial odometry for vehicle tracking in GPS blocked environments». In: *IEEE Internet of Things Journal* 8.16 (2021), pp. 12955–12967.

[28] Martin Brossard, Silvere Bonnabel, and Jean-Philippe Condomines. «Unscented Kalman filtering on Lie groups». In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2017, pp. 2485–2491.

[29] Honghui Qi and John B Moore. «Direct Kalman filtering approach for GPS/INS integration». In: *IEEE Transactions on Aerospace and Electronic Systems* 38.2 (2002), pp. 687–693.

[30] Nelson Acosta and Juan Toloza. «Techniques to improve the GPS precision». In: *International Journal of Advanced Computer Science and Applications* 3.8 (2012).

[31] Evelina Forno et al. «Techniques for improving localization applications running on low-cost IoT devices». In: *2020 AEIT International Conference of Electrical and Electronic Technologies for Automotive (AEIT AUTOMOTIVE)*. 2020, pp. 1–6. DOI: 10.23919/AEITAUTOMOTIVE50086.2020.9307411.

[32] Leonard A McGee and Stanley F Schmidt. *Discovery of the Kalman filter as a practical tool for aerospace and industry*. Tech. rep. 1985.

[33] Greg Welch, Gary Bishop, et al. «An introduction to the Kalman filter». In: (1995).

[34] A. M. Turing. «Computing Machinery and Intelligence». English. In: *Mind*. New Series 59.236 (1950), pp. 433–460. ISSN: 00264423. URL: http://www.jstor.org/stable/2251299.

[35] Martina De Castro, Umberto Zona, and Fabio Bocci. «"L'apprendimento macchinico tra Skinner box e Deep Reinforcement Learning. Rischi e opportunità. Machine Learning between Skinner box and Deep Reinforcement Learning. Risks and opportunities", in "Dalle Teaching Machines al Machine Learning" a cura di Graziano Cecchinato, Valentina Grion - PREPRINT». In: July 2020, pp. 27–33. ISBN: 978-88-6938-199-7.

[36] Yani Ioannou. «Structural Priors in Deep Neural Networks». PhD thesis. Sept. 2017. DOI: 10.17863/CAM.26357.

[37] Carlos R. Argüelles and Santiago Collazo. «Galaxy Rotation Curve Fitting Using Machine Learning Tools». In: *Universe* 9.8 (2023). ISSN: 2218-1997. DOI: 10.3390/universe9080372. URL: https://www.mdpi.com/2218-1997/9/8/372.

[38] John Schulman et al. «Gradient estimation using stochastic computation graphs». In: *Advances in neural information processing systems* 28 (2015).

[39] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[40] Marina MM Rocha, Gabriel Landini, and Joao B Florindo. «Medical image classification using a combination of features from convolutional neural networks». In: *Multimedia Tools and Applications* 82.13 (2023), pp. 19299–19322.

[41] Morteza Zakeri Nasrabadi, Saeed Parsa, and Akram Kalaee. «Format-aware learn&fuzz: deep test data generation for efficient fuzzing». en. In: *Neural Comput. Appl.* 33.5 (Mar. 2021), pp. 1497–1513.

[42] Yong Yu et al. «A review of recurrent neural networks: LSTM cells and network architectures». In: *Neural computation* 31.7 (2019), pp. 1235–1270.

[43] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. «Learning long-term dependencies with gradient descent is difficult». In: *IEEE transactions on neural networks* 5.2 (1994), pp. 157–166.

[44] Praveen Venkatesh, Rwik Rana, and Varun Jain. *Memory Guided Road Detection*. June 2021. DOI: 10.48550/arXiv.2106.14184.

[45] Jason K. Eshraghian et al. «Training Spiking Neural Networks Using Lessons From Deep Learning». In: *Proceedings of the IEEE* 111.9 (2023), pp. 1016–1054. DOI: 10.1109/JPROC.2023.3308088.

[46] Wikipedia contributors. *Biological neuron model — Wikipedia, The Free Encyclopedia*. [Online; accessed 6-July-2024]. 2024. URL: https://en.wikipedia.org/w/index.php?title=Biological_neuron_model&oldid=1232408002.

[47] *Biosegnali del sistema visivo — hdl.handle.net*. https://hdl.handle.net/20.500.12608/32244. [Accessed 07-07-2024].

[48] Antoine Bourrier. «Graphene bioelectronics for long-term neuronal interfacing in-vivo». In: 2017. URL: https://api.semanticscholar.org/CorpusID:139472009.

[49] Elias Mueggler et al. «Lifetime estimation of events from dynamic vision sensors». In: *2015 IEEE international conference on Robotics and Automation (ICRA)*. IEEE. 2015, pp. 4874–4881.

[50] Lloyd Watts et al. «Improved implementation of the silicon cochlea». In: *IEEE Journal of Solid-state circuits* 27.5 (1992), pp. 692–700.

[51] Evelina Forno et al. «Spike encoding techniques for IoT time-varying signals benchmarked on a neuromorphic classification task». In: *Frontiers in Neuroscience* 16 (2022), p. 999029.

[52] Michael Pfeiffer and Thomas Pfeil. «Deep learning with spiking neurons: opportunities and challenges». In: *Frontiers in neuroscience* 12 (2018), p. 409662.

[53] Sander M Bohte, Joost N Kok, and Johannes A La Poutré. «SpikeProp: backpropagation for networks of spiking neurons.» In: *ESANN*. Vol. 48. Bruges. 2000, pp. 419–424.

[54] Sumit B Shrestha and Garrick Orchard. «Slayer: Spike layer error reassignment in time». In: *Advances in neural information processing systems* 31 (2018).

[55] Friedemann Zenke and Tim P. Vogels. «The Remarkable Robustness of Surrogate Gradient Learning for Instilling Complex Function in Spiking Neural Networks». In: *Neural Computation* 33.4 (Mar. 2021), pp. 899–925. ISSN: 0899-7667. DOI: `10.1162/neco_a_01367`. eprint: `https://direct.mit.edu/neco/article-pdf/33/4/899/1902294/neco\_a\_01367.pdf`. URL: `https://doi.org/10.1162/neco%5C_a%5C_01367`.

[56] Michael Marsalli. «Mcculloch-pitts neurons». In: *The 2008 Annual Meeting of the consortium on cognitive science instruction (ccsi)*. Vol. 1161. 2006, p. 1162.

[57] Wolfgang Maass. «Networks of spiking neurons: the third generation of neural network models». In: *Neural networks* 10.9 (1997), pp. 1659–1671.

[58] Louis Édouard Lapicque. «Louis lapicque». In: *J. physiol* 9 (1907), pp. 620–635.

[59] Roberto Vazquez. «Izhikevich neuron model and its application in pattern recognition». In: *Australian Journal of Intelligent Information Processing Systems* 11.1 (2010), pp. 35–40.

[60] Verena Brehm et al. «A proposal for leaky integrate-and-fire neurons by domain walls in antiferromagnetic insulators». In: *Scientific Reports* 13.1 (2023), p. 13404.

[61] Michael Hirsch. «PyMap3D: 3-D coordinate conversions for terrestrial and geospace environments». In: *Journal of Open Source Software* 3.23 (2018), p. 580.

[62] Aaron Voelker, Ivana Kajić, and Chris Eliasmith. «Legendre memory units: Continuous-time representation in recurrent neural networks». In: *Advances in neural information processing systems* 32 (2019).

[63] Sarath Chandar et al. *Towards Non-saturating Recurrent Units for Modelling Long-term Dependencies*. 2019. arXiv: `1902.06704 [cs.NE]`. URL: `https://arxiv.org/abs/1902.06704`.

[64] Vittorio Fra et al. «Natively neuromorphic LMU architecture for encoding-free SNN-based HAR on commercial edge devices». In: *International Conference on Artificial Neural Networks*. Springer. 2024, pp. 377–391.

[65] Adam Paszke et al. «Automatic differentiation in PyTorch». In: (2017).

[66] Microsoft. *Neural Network Intelligence*. Version 2.0. Jan. 2021. URL: `https://github.com/microsoft/nni`.

[67] Matteo Turisini, Giorgio Amati, and Mirko Cestari. «Leonardo: A pan-European pre-exascale supercomputer for HPC and AI applications». In: *arXiv preprint arXiv:2307.16885* (2023).

[68] Fazle Karim et al. «Multivariate LSTM-FCNs for Time Series Classification». In: *CoRR* abs/1801.04503 (2018). arXiv: `1801.04503`. URL: `http://arxiv.org/abs/1801.04503`.

[69] python-visualization. *Folium*. Version 0.11.0. Dec. 28, 2020. URL: `https://python-visualization.github.io/folium/`.

[70] Shaoshan Liu et al. «Brief Industry Paper: The Matter of Time — A General and Efficient System for Precise Sensor Synchronization in Robotic Computing». In: *2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. 2021, pp. 413–416. DOI: `10.1109/RTAS52030.2021.00040`.

[71] Jens E Pedersen et al. «Neuromorphic intermediate representation: a unified instruction set for interoperable brain-inspired computing». In: *Nature Communications* 15.1 (2024), p. 8122.