UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Development and Evaluation of a Road Marking Recognition Algorithm implemented on Neuromorphic Hardware

Degree final work

Degree in Computer Engineering

Author: Bou Betran, Santiago
Tutor: Monserrat Aranda, Carlos
External Cotutor: Conradt, Jörg

Course 2021-2022

Resum

La conducció és una de les formes de transport més comunes i preferides en l'actualitat. Tanmateix, diferents estudis assenyalen també que hi és una de les més perilloses. Una solució per augmentar la seguretat a la carretera és aplicar la tecnologia per automatitzar i prevenir els errors evitables humans. No obstant, malgrat dels esforços per aconseguir sistemes fiables, encara no hi hem trobat una solució suficientment fiable i segura per resoldre aquest repte. Una de les raons és l'entorn de la conducció, amb situacions que disten molt de les ideals, per condicions d'il · luminació variables i entorns ràpids i imprevisibles. Aquest projecte desenvolupa i avalua un algorisme que pren l'entrada de sensors de visió dinàmica (DVS) i executa la seva computació amb xarxes neuronals neuromòrfiques (SNN) per obtenir un sistema robust de seguiment de carrils a la carretera.

Presentem mètriques quantitatives i qualitatives que avaluen el rendiment del reconeixement de carrils en condicions de poca llum, davant d'algorismes convencionals. Aquest projecte està motivat per la validació de les avantatges dels sensors de visió neuromòrfica: el reconeixement d'un alt rang dinàmic i la captura d'imatges d'alta velocitat. Un altra de les millores que s'espera d'aquest sistema és la velocitat de processament i l'eficiència energètica que caracteritza el hardware neuromòrfic basat en xarxes neuronals d'impulsos. Els resultats obtinguts mostren una precisió similar entre el nou algorisme en comparació amb implementacions anteriors en plataformes convencionales. I el que és més important, realitza la tarea proposada amb menor latència i requisits de potència de càlcul.

Paraules clau: Hardware Neuromòrfic,Sistemes de visió per computadora, Reconeixement de Carril, Conducció autònoma, Sensors de visió neuromòrfica, Xarxes neuronals d'Impulsos,Xarxes neuronals de Tercera Generació, Avaluació d'Algorismes

Resumen

La conducción es una de las formas de transporte más comunes y preferidas en la actualidad. Sin embargo, diferentes estudios muestran que también es una de las más peligrosas. Una solución para aumentar la seguridad en la carretera es aplicar la tecnología para automatizar y prevenir los evitables errores humanos. No obstante, a pesar de los esfuerzos por conseguir sistemas fiables, todavía no hemos encontrado una solución suficientemente fiable y segura para resolver este reto. Una de las razones es el entorno de la conducción, en situaciones que distan mucho de las ideales, con condiciones de iluminación variables y entornos rápidos e imprevisibles. Este proyecto desarrolla y evalúa un algoritmo que toma la entrada de sensores de visión dinámicos (DVS) y ejecuta su computación en redes neuronales neuromórficas (SNN) para obtener un sistema robusto de seguimiento de carriles en carretera.

Presentamos métricas cuantitativas y cualitativas que evalúan el rendimiento del reconocimiento de carriles en condiciones de poca luz, frente a algoritmos convencionales. Este proyecto está motivado por la validación de las ventajas de los sensores de visión neuromórficos: el reconocimiento de un alto rango dinámico y la captura de imágenes de alta velocidad. Otra de las mejoras que se espera de este sistema es la velocidad de procesamiento y la eficiencia energética que caracterizan al hardware neuromórfico basado en redes neuronales de impulsos. Los resultados obtenidos muestran una precisión similar entre el nuevo algoritmo en comparación con implementaciones anteriores en plataformas

convencionales. Y lo que es más importante, realiza la tarea propuesta con menor latencia y requisitos de potencia de cálculo.

Palabras clave: Hardware Neuromórfico,Sistemas de visión por computador, Reconocimiento de Carril, Conducción autónoma, Sensores de visión neuromórfica, Redes neuronales de Impulsos,Redes Neuronales de Tercera Generación, Evaluación de Algoritmos

Sammanfattning

Att köra bil är ett av de vanligaste och mest populära transportsätten i vårt samhälle. Enligt forskningen är det också ett av de farligaste. En lösning för att öka säkerheten på vägarna är att med teknikens hjälp automatisera bilkörningen och på så sätt förebygga misstag som beror på den mänskliga faktorn. Trots ansträngningarna för att få fram tillförlitliga system har man dock ännu inte hittat en tillräckligt tillförlitlig och säker lösning för självkörande bilar. En av orsakerna till det är att många körningar sker under förhållanden som är långt ifrån idealiska, med varierande ljusförhållanden och oförutsägbara miljöer i höga hastigheter. I det här projektet utvecklar och utvärderar vi en algoritm som tar emot indata från dynamiska synsensorer (Dynamic Vision Sensors, DVS) och kör datan på neuromorfiska pulserande neuronnät (Spiking Neural Networks, SNN) för att skapa ett robust system för att läsa av vägbanan.

Vi presenterar en kvantitativ och kvalitativ utvärdering av hur väl systemet läser av körbanans linjer i svagt ljus, och jämför därefter resultaten med dem för tidigare algoritmer. Detta projekt motiveras av de viktigaste fördelarna med neuromorfiska synsensorer: brett dynamiskt omfång och hög bildtagningshastighet. En annan fördel hos detta system är den korta beräkningstiden och den energieffektivitet som kännetecknar neuromorfisk hårdvara baserad på pulserande neuronnät. De resultat som erhållits visar att den nya algoritmen har en liknande noggrannhet som tidigare algoritmer på traditionella hårdvaruplattformar. I jämförelse med den traditionella tekniken, utför algoritmen i den föreliggande studien sin uppgift med kortare latenstid och lägre krav på processorkraft.

Nyckelord: SpiNNaker, Neuromorfiska hårdvara, neuromorfiska pulserande neuronnät, Datorseende system, läser av körbanans, självkörande bilar, läser av linjer, dynamiska synsensorer, Tredje generation neuronnät, utvärdering av algoritmen

Abstract

Driving is one of the most common and preferred forms of transport used in our actual society. However, according to studies, it is also one of the most dangerous. One solution to increase safety on the road is applying technology to automate and prevent avoidable human errors. Nevertheless, despite the efforts to obtain reliable systems, we have yet to find a reliable and safe enough solution for solving autonomous driving. One of the reasons is that many drives are done in conditions far from the ideal, with variable lighting conditions and fast-paced, unpredictable environments. This project develops and evaluates an algorithm that takes the input of dynamic vision sensors (DVS) and runs on neuromorphic spiking neural networks (SNN) to obtain a robust road lane tracking system.

We present quantitative and qualitative metrics that evaluate the performance of lane recognition in low light conditions against conventional algorithms. This project is motivated by the main advantages of neuromorphic vision sensors: recognizing a high dynamic range and allowing a high-speed image capture. Another improvement of this system is the computational speed and power efficiency that characterize neuromorphic hardware based

on spiking neural networks. The results obtained show a similar accuracy of this new algorithm compared to previous implementations on conventional hardware platforms. Most importantly, it accomplishes the proposed task with lower latency and computing power requirements than previous algorithms.

Key words: Neuromorphic Hardware; Spiking Neural Network; Computer vision system; Lane recognition; Autonomous driving; Line Recognition; Neuromorphic Vision Sensors; Third Generation Neural Networks; Algorithm Evaluation

Contents

Contents vii
List of Figures ix
List of Tables x

1 Introduction 3
1.1 Research Question . . . 4
1.2 Approach . . . 4
1.3 Scope . . . 5
2 Background 7
2.1 Neuromorphic Hardware . . . 7
2.1.1 The SpiNNaker Board . . . 7
2.1.2 Neuron architecture in the SpiNNaker Board . . . 8
2.2 Event Based Vision Sensors . . . 9
2.3 Lane and Line Detection . . . 10
2.3.1 Previous Studies on Lane Detection . . . 11
2.4 Spiking Neural Networks and Lane Detection . . . 11
2.5 Putting it all together . . . 11
3 Methods 13
3.1 Dataset . . . 13
3.1.1 Obtention of a Dataset of Simple Straight Lines . . . 14
3.1.2 Obtention of a Driving Dataset . . . 14
3.2 Line Detection . . . 15
3.2.1 Hough Transform . . . 15
3.2.2 Applying Hough transform to Spiking Neural Networks . . . 16
3.2.3 The implemented algorithm . . . 17
3.2.4 Accessing and interfacing with the Spiking Neural Hardware . . . 18
3.3 Evaluating the obtained data . . . 18
3.3.1 Visual evaluation . . . 19
3.3.2 Analytical Evaluation of Performance . . . 20
3.4 Power efficiency . . . 20
3.5 Time latency evaluation . . . 21
3.6 Optimizations . . . 21
3.6.1 Reduce input space . . . 22
3.6.2 Temporal subsampling of the input data . . . 22
3.6.3 Output Data processing for the Real Driving Dataset . . . 23
4 Results 25
4.1 Results from Test Cases . . . 25
4.1.1 Straight Line . . . 25
4.1.2 Two Angled Lines . . . 27
4.2 Results from Experimental analysis . . . 29
4.2.1 Visual Analysis . . . 30
4.2.2 Analytical Evaluation . . . 31

4.2.3 Power efficiency . . . 33
4.2.4 Latency Evaluation . . . 33
5 Discussion **35**
5.1 Driving Dataset . . . 35
5.2 Latency and Power . . . 36
5.3 The novelty of the Neuromorphic Systems . . . 37
6 Conclusions **39**
6.1 Archieved results . . . 39
6.2 Further Research . . . 39
Bibliography **41**

Appendices
A LIF Neuron Parameters **45**
B Results in Video **47**

List of Figures

1.1 Basic Model of a Neuron [1] 4

2.1 The SpiNNaker architecture [19] 8
2.2 Formula for the Voltage in the LIF neuron model [21] 9
2.3 Comparison Schema of a conventional camera and an Event vision camera. [24] 9

3.1 Process of capturing for DAVIS Sensor 14
3.2 MVSEC Dataset Recording setup 15
3.3 Sample of events reconstructed from MVSEC Nightdrive Dataset 15
3.4 Formula of the Hough Transform [32] 16
3.5 Original line setting 16
3.6 Result in Parameter Space 16
3.7 Schema for the implemented Neuronal model, based on the model by Wu et al.[11] 17
3.8 Equations defining the parameters of a straight line on cartesian coordinates given the polar coordinates (Hough Transform parameters) 19
3.9 Comparison of original and reduced input 22
3.10 Result of x10 downsampling of the input 23

4.1 Visual comparison between OpenCV's Hough Transform algorithm for straight line detection (red line on 4.1a) against the implementation using neuromorphic hardware on SpiNNaker (red line on 4.1b). Results from using a single straight line dataset recording(green background and blue marked lines). Animated video can be accesed at B 26
4.2 Visual comparison between OpenCV's Hough Transform algorithm for straight line detection (red line on 4.2a) against the implementation using neuromorphic hardware on SpiNNaker (red line on 4.2b). Results from using a dataset recording showing two angled straight line representing road markings (background in grayscale). Video of the output available in Annex B 28
4.3 Data processing carried out by the algorithm from the obtention of the events from the DVS Sensor to the output generated by the Neural Network according to the parameters of the Hough Transform equation 3.1 30
4.4 Visual comparison between OpenCV's Hough Transform algorithm for MVSEC line detection (red line on 4.4a) against the implementation using neuromorphic hardware on SpiNNaker (red line on 4.4b). Results from using a dataset of night drive recording (black an white frames). Video with side by side comparison available at the Annex B 31
4.5 32

List of Tables

4.1 Analytical results of the averaged parameter difference between the Spiking algorithm and the *OpenCV* implementation of the Hough Transform over 473 frames for each model. Animated video can be accesed at B 26
4.2 Analytical results of the **left line** using the averaged parameter difference between the Spiking algorithm and the *OpenCV* implementation of the Hough Transform over 219 frames for each model 28
4.3 Analytical results of the **right line** using the averaged parameter difference between the Spiking algorithm and the *OpenCV* implementation of the Hough Transform over 219 frames for each model 29
4.4 Analytical performance results for the detection **left lane** using the averaged parameter difference between the Spiking algorithm and the *OpenCV* implementation of the Hough Transform over 1400 frames for each model . 32

A.1 Parameters used for the obtention of the results in the neuron output layer of the SpiNNaker Board 45

Acknowledgements

I would like to thank Jörg Conradt as my supervisor on this report for his disposition and enthusiastic support from the start till the last day of this project's development, providing generous access to the necessary resources and hardware of the NeuroComputing Systems laboratory at KTH.

I want to thank Juan Pablo Romero Bermudez for his technical support and availability, for answering my requests in the least time possible, and for providing me with guidance and help on my journey through the many bumps on the road with the SpiNNaker platform. Thanks to my peer reviewer Alex for his helpful comments and suggestions on how to improve this report.

I would like to thank my friend Malin for her help and support with Swedish. And finally, thanks to my family and all the people who heard my explanations and provided motivation and suggestions to make this crazy idea a reality.

CHAPTER 1
Introduction

The development of autonomous driving vehicles represents one of the most significant challenges for current Artificial Intelligence models, as can be observed by the great number of studies developed in recent years [2, 3, 4, 5] For this reason, this field represents a baseline to evaluate the current capabilities and most advanced "state of the art" technologies. These technologies face the challenge of solving this car driving problem with accuracy and reliability. The solution motivates the advance of the new technology and allows using these tasks as an evaluation benchmark for comparing different technologies. [6]

The first approaches, such as the system created by Thorpe et al., show an initial strategy for solving this problem. One of the main obstacles was the low computational power of the available computers, which limited the system's speed to an average lower than 30km/h [7]. Nowadays, however, we have access to computing resources characterized by their powerful, flexible, and specialized topologies that allow better efficiency and performance while offering great flexibility to adapt to different use scenarios [8]. These new architectures provide researchers with a broad range of systems to implement new solutions to existing problems. For this reason, researchers work on looking for optimizations for the already available approaches. Another option is to research and evaluate the use of new emerging platforms and assess their adequacy compared to previous models. These are also key strategies to obtain more capable systems that improve performance over previous state-of-the-art research.

Therefore, in this project, we evaluate the development of a system based on a new computing paradigm for addressing the lane detection problem, in this instance, by using neurological hardware. This choice is motivated by the results of previous studies, where we can see how the application of this new hardware in well-known tasks can improve the accuracy and speed of its output. These systems, also called third-generation Neural networks, are based on Spiking Neural Networks (SNN), whose foundations are inspired by the biology of the brain's neurons, as shown in Figure 1.1. [3, 4]

The working nature of Spiking Neural Networks is based on replicating the network of connections formed between the brain's neurons down to the physical level. These neuronal networks represent a programming paradigm where systems can perform complex calculations based on many simple neurons that communicate between themselves with pulses of electricity, also called Spikes. One of the main advantages is achieving a high level of parallelization on high-scale networks of neurons that define different interconnections and weights between them. [9]

There exist different devices based on neuromorphic hardware. For this project, we work on an integral neuromorphic setup composed of an event-based sensor that provides the input for the system and a processing unit (Spiking Neural Network, SNN) that performs the computations based on the brain's architecture. This architecture allows us

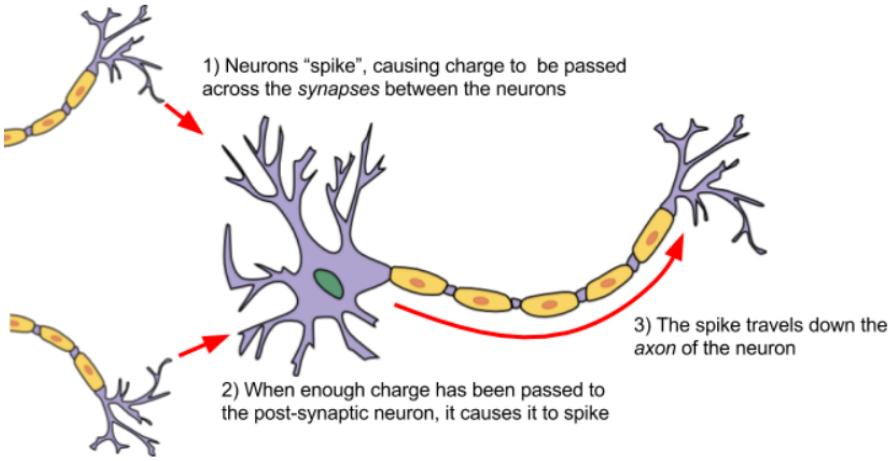


Figure 1.1: Basic Model of a Neuron [1]

to produce and receive all needed operations in a single computing device, saving power and space. The board used in this project is the *SpiNNaker*, developed by the University of Manchester [10].

In conclusion, we define the lane detection problem as one of the critical aspects of accomplishing the autonomous navigation of vehicles. The problem is based on the necessity for accurate perception and translation of the real world into digital data. Thanks to this system, future car driving algorithms can use the resulting data as input for decision-making. In this way, we center on solving one of the core tasks for autonomous driving: obtaining input data from lane detection. The road markings have to be precisely recognized to model the system's environment and analyze the car's current position with respect to its surroundings. This process also requires the shortest possible computing time, minimizing the latency to have a responsive system. We base our study on previous research done in this field ([4, 11, 12]) and expand it by combining the usage of a neuromorphic system for both obtention and computation of the lane markings.

1.1 Research Question

Development and Evaluation of a new Road Marking Recognition Algorithm implemented on Neuromorphic Hardware, based on the usage of Event-Based Cameras and Spiking Neural Networks and compared against previously studied conventional algorithms.
In this project, we implement and evaluate a new algorithm based on the findings of previous studies made on related topics to create a new system implementing the desired functionality of lane detection. For this reason, this system has no precedent, as it implements a lane tracking system using exclusively neuromorphic hardware, and its computation is based wholly on asynchronous events. Finally, we compare the obtained results with previous studies to evaluate the system's accuracy and correctness.

1.2 Approach

The approach of this project is divided into three steps. First, review and evaluate other techniques explored in previous systems [4, 11, 12] to set a comparison baseline from which we evaluate our development. Secondly, we create the new system, where the

new algorithm is defined and then implemented based on the constraints and desired functionalities to be achieved. Therefore, we carry out the implementation followed by careful study and optimization of the algorithm to ensure we obtain a correct output directly measured against previous research on the topic.

Finally, after obtaining our working system, we describe the results. We evaluate and compare the results by applying the same input dataset to our new model and to previous conventional solutions for the problem. Given this data, we present it visually in this project and extract different metrics, allowing us to establish an objective comparison and do a performance study based on the methods explained in the corresponding section. Results show whether the approach followed in this research leads to a more capable system than previous approaches used in earlier studies with different conventional methodologies aiming to solve the same task.

1.3 Scope

In this project, we aim to assess one of the key capabilities of neuromorphic hardware. To accomplish this goal, we evaluate the algorithm on the scope of lane detection under low light conditions [13]. We compare the model's output using a dataset captured by driving during the night. [5]

This scope selection allows us to show and prove one of the essential characteristics of neuromorphic sensors: their sensibility to illumination changes, which enables the obtention of meaningful data from their surroundings even in low light portions of the image. These characteristics also prove the suitability of neuromorphic hardware for autonomous driving. Use scenarios for autonomous cars often involve far from ideal conditions for illumination, making it very important to show flexibility to the usage in low luminance. According to the NHTSA (National Highway Traffic Safety Administration of the USA), at least 25 % of drives performed in the United States in 2008 were done at night [14]. These situations also involve the highest percentage of crashes, being 68% of the fatal injuries produced in night drives, as shown by the latest report from NHTSA in 2022 [15]. This data proves it as one of the most challenging use cases for autonomous driving and remarks on the benefits of applying neuromorphic hardware for this use case to reduce fatalities and achieve safer driving.

CHAPTER 2
Background

This project is based on two main fields of study: neuromorphic vision sensors and neuromorphic computing models. A new approach is explored in the project, aiming to take advantage of the most highlighted features of using both methods together.

2.1 Neuromorphic Hardware

Neuromorphic hardware represents the primary tool for the development of this project. Our main goal is to evaluate the performance of this hardware compared to implementations done on conventional computing platforms. The primary motivation for the development of this project is to assess the capabilities of neuromorphic hardware. In our case, the system will be using two neuromorphic devices, Event-Based cameras, which will act as input sensors for the model, and use a Spiking Neuronal Computer to perform the computations.

The architecture of the human brain is the main inspiration followed for the design of neuromorphic computers. We can define the brain as an extensive array of neurons connected by synapses. These synapses provide the neuron with specific ways to communicate by generating short pulses. These pulses, also called spikes, are then propagated so that the next neuron can produce another output based on the combination of different input pulses. This modification of the connections and their neuron behaviors in the system constitutes the main programming variables available, which we can use to simulate varying processes and successfully perform a specific task.

The study by Date et al. successfully proved Spiking Neural Networks as a Turing complete architecture. This study demonstrates how a set of μ-recursive functions can be computed using neurons on a mathematical model with an infinite number of neurons. With these results, we can prove the capabilities of neuromorphic computation to solve any general computing task, as the neurons can emulate any Boolean logic functions [16].

2.1.1. The SpiNNaker Board

In our project, we simulate a neurological network by using the development platform SpiNNaker[17]. The University of Manchester initially developed this board in 2005 with funding from the *Engineering and Physical Sciences Research Council*[18] and in posterior with the collaboration of other universities and companies, such as the processor designer *ARM*. Its main objective is to obtain a computer capable of simulating the neurons in the brain by following the fundamental structure and elements that make the functioning of the brain. In this way, it aims to accomplish a high-performance and efficient computing platform that can be used as a tool in different fields to research the impact of different

architectures instead of using conventional supercomputing methods for the simulation of neuronal models.

The main fundament on which SpiNNaker bases its working is the research based on the brain neurons and the synapses between them. According to each project's computing necessities and physical space requirements, there exist different variations of the board. In this project, because of the big parameter space used, we used the SpiNNaker 5 board. The SpiNNaker 5 board, as shown in Figure 2.1 is formed by 48 nodes, each of them containing 16 cores for a total of 768 available computing cores. The design of these cores is motivated to provide a high level of task parallelization. For this reason, all the elements needed for the computation, such as the 128MB of SDRAM memory per chip, Memory controllers, and routing elements, are enclosed in the same processor to improve latency and allow a higher replication by the distribution of these systems.[19, 17] Additionally, each of these nodes is interconnected using a high-speed bus. The board also provides a Gigabit Ethernet connection port for handling the communication with the managing computer for data inputting and programming of the board.[20, 21]

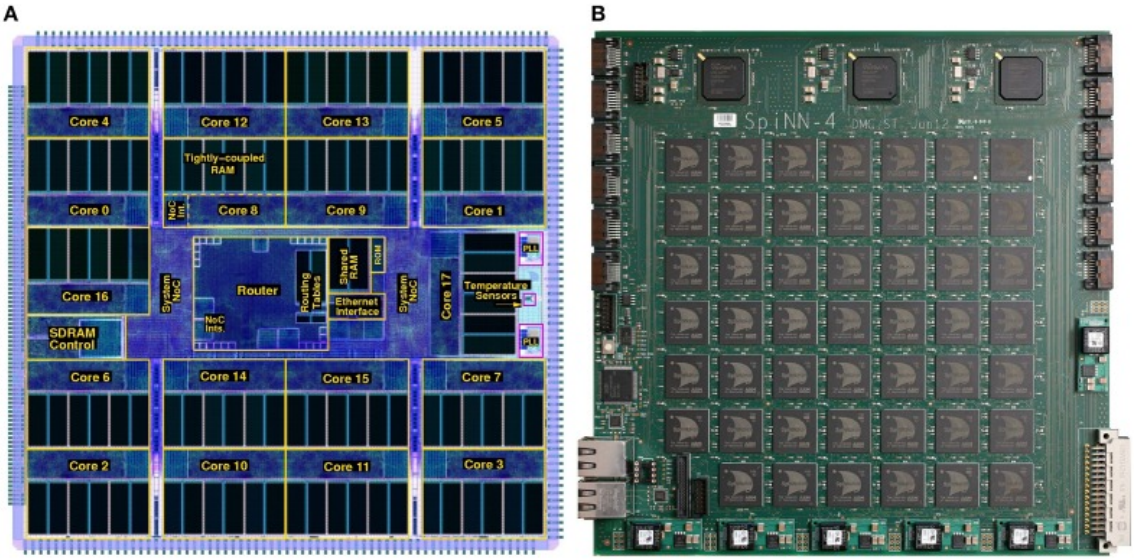


Figure 2.1: The SpiNNaker architecture [19]

2.1.2. Neuron architecture in the SpiNNaker Board

The SpiNNaker architecture is programmed using the standardized *PyNN* [22] python library, on top of which resides the lower-level layer in charge of communicating and sending the corresponding commands from the programming computer to the physical Spiking board. This library defines several Neuron models that encode different behaviors and parameters that we can modify to simulate real brain neurons and use them to obtain the most accurate results depending on the needed task. [21, 22]

The most widely used model is the Leaky Integrate and Fire (LIF), mainly because of its simplicity and performance. The LIF model is based on the electrical modeling of a neuron as a circuit with a capacitor and a resistor connected in parallel. This capacitor simulates the charging of a neuron, where the different electrical pulses received by the synapses are added until a specific threshold voltage. Once this limit value is reached, the neuron produces a spike, represented by the capacitor discharging and leaking its charge over the resistor, which models the time it takes for the capacitor to discharge fully. The model's formula that calculates the voltage at a given point in the neuron is presented

$$\frac{dV}{dt} = -\frac{V - (E_l + R_m I(t))}{\tau_m} \text{ If } V > V_0, V = V_{reset} \tag{2.1}$$

Figure 2.2: Formula for the Voltage in the LIF neuron model [21]

in Equation 2.1. In this mathematical equation, we show the resulting potential of the neuron V based on the input current I, and starting from a leak potential E_l. We can observe the relation between the possible parameters of the model, such as the resistance of the membrane R_m and the leak time constant τ_m. [1, 23]

We can also observe the threshold condition encoded in Equation 2.1; this models the neuron's behavior once it reaches the maximum voltage. In this case, the voltage V of the neuron is set to the value of parameter V_{reset}, which encodes the starting voltage of a neuron after producing a spike. [23]

The *PyNN* library implements other models apart from the presented LIF neuron. One of the most important is the Izhikevich Neuron. This model also implements the simulation of a biological neuron based on the characteristics and results observed in different studies. This model is based on a quadratic interpretation of the *integrate and fire*, mainly inspired by the fundamentals of the LIF neuron mentioned above. The main difference resides in the behavior and nature of the parameters, which are dimensionless and decoupled from a physical meaning as opposed to the previous model, which in some specific tasks may be more performant. [21]

2.2 Event Based Vision Sensors

Event-based vision sensors, also called Dynamic Vision Sensors (DVS) or neuromorphic cameras, allow us to record the world using an asynchronous, event-driven approach inspired by the eye's retina. This means the system only produces an output in a pixel once a change in its intensity is detected. The resulting data is presented as a tuple (x, y, t, p) where we find the coordinates of the pixel (x, y), the time where the event is generated t and the polarity of the change p, which is either positive (1) or negative (-1). This working principle is opposed to conventional cameras (CMOS). In event-based cameras, the output corresponds to an array of pixels generated regularly by following a constant frequency (e.g. 60fps), regardless of whether the pixels recorded are similar to their predecessors. [12]

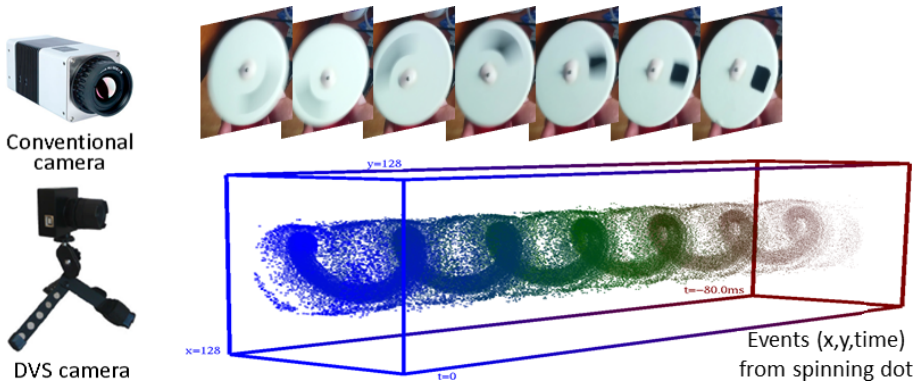


Figure 2.3: Comparison Schema of a conventional camera and an Event vision camera. [24]

The DVS architecture presents advantages that make it especially suitable for the line detection challenge. On the one side, they are more power-efficient than regular cameras. Their basis for output generation is based on the changes in the image. The amount of

power and data generated depends on the changes found in the picture and thus allows more flexibility to changing environments. This fact also leads to a better time resolution than conventional sensors, allowing event-based sensors to capture more detail on high-speed events, equivalent to a resolution of a million frames per second on conventional cameras. This feat is possible by limiting the tracking of the changes in the sensor where only the affected pixels generate a signal. [13, 25]

In addition, and related to the scope chosen for this project, DVS cameras have an extended dynamic range (120dB compared to 60dB on conventional cameras). This range results from using a logarithmic function on the sensors that determine when to trigger the events. Only those events exceeding a brightness difference higher than a predefined threshold will "fire" and produce an output event. This feature means these sensors can obtain details from images with a wide exposure range (from very brightly lit to dark sections), making them suitable for recording road lanes, where there are constant changes in illumination conditions. Additionally, cases such as night drives suppose an extra challenge that requires a high sensibility of the sensor to capture the lines in environments with low illumination. For this reason, using these sensors potentially allows us to obtain a higher amount of detail, resulting in a more capable and precise system. Results explored by Hu et al.[5] present a 17% better steering accuracy when using the dataset obtained at night when compared with the same environment at day, proving how DVS sensors can significantly obtain more details in poor illumination environments. [12, 5]

2.3 Lane and Line Detection

Lane detection is a well-known application for computer vision, precisely the task of line recognition to obtain the lanes' position in an environment. Computer vision research started in the late 1960s when the first studies explained different methods to translate geometric objects from the 3D world to a 2D perspective using computing machines. These methods set a starting point for further developments, soon switching to using images instead of figures as input. The obtained results led to a period of optimism, where most researchers considered the problem as easily solvable. However, soon it was proven how the challenge of computer vision required the analysis and computation of an extensive amount of data. Studies showed that the available power of computers at the time was not sufficient for showing any relevant results, and vision became one of the principal stepping stones that led to the AI winter. With the arrival of the internet and the development of computing power, studies centered on computer vision began gaining popularity in 1990.[26]

Current developments focus on training and using neural networks to apply artificial intelligence for line detection. The research was motivated by the improvement of computing hardware that led to new techniques based on higher amounts of data processing. Challenges such as the *ImageNet Large Scale Visual Recognition Challenge*[27] motivated new developments and allowed the direct competition to advance and obtain better methods for image recognition. The error rate obtained with each iteration of the models dropped significantly, opening a new era of optimism and motivating the application of computer recognition models to more situations and environments. Among them, we find the field of autonomous driving, where environment recognition has a crucial role in ensuring an accurate system that can obtain a significant amount of information in the shortest possible time.

2.3.1. Previous Studies on Lane Detection

To perform the task of lane recognition using DVS, we find previous studies such as the one made by Everding and Conradt[12]. This study develops an algorithm for multiple line recognition from scenes captured by event-based sensors. An algorithm is developed to accomplish the objective of clustering the events into differentiated groups. Then it is checked whether these clusters can form a plane in a specific time and position. The results from the study show how this system was capable of obtaining a high precision recognition of lines. It demonstrated a low latency and surpassed other algorithms based on CMOS cameras by getting a 10% better accuracy.

Another study carried out by Reverter Valeiras et al.[28] proposes another approach for line detection on DVS based on the idea of weight applied to the events triggered by the camera. In this algorithm, the events received are analyzed individually and assigned to a possible line in the plane. The algorithm checks if the line is closer than a threshold to the previous guess, then its weight is updated with a mean square error formula to fit the line. The main reason for this approach is to allow the algorithm to store within memory the previous events and be able to use them to classify future input based on it. This study showed high accuracy for the line recognition while presenting some difficulties when the camera moved in a parallel direction to the detected lines, which can be a problem for the task researched in this project.

2.4 Spiking Neural Networks and Lane Detection

As expressed before, we can find studies that develop algorithms to detect straight lines by using spiking neuronal networks [11, 12, 28]. We center our research on those studies showing the application of this technology for lane recognition on the road. The paper by Li et al.[4] shows how the use of the Hough Transform for detecting lanes from a captured set of images of road driving. This study presents an algorithm consisting of three stages. The first stage consists of RGB image processing to allow more detail and contrast to be extracted from the raw image. After it, edge detection is performed to obtain the lane markings boundaries from the pixels showing the lines. The spiking network performs this operation by setting a layer of four neurons in charge of recognizing the possible pixel positions representing an edge in an image, with one neuron used to detect each possible edge and the positions on a square. The information is passed to the output layer in charge of encoding the positions where an edge has been successfully detected. This image is then used in the final step of the algorithm that applies the Hough Transform, as mentioned above, on the detected edges to find a line that represents the lanes of the road (supposing the camera captures a straight lane with two parallel lines on each side of the camera stream). This research shows how the system can speed up the detection task compared to previous algorithms, in this case by being run on a parallel board resulting in a latency measurement within 100ms. The accuracy obtained resulted in a lane true positive rate of 92.6%.[4]

2.5 Putting it all together

We define a working guideline for our study using the provided background. This project aims to evaluate the performance of a lane recognition system implemented exclusively using neuromorphic hardware. As a complement to the previous studies, we are using both a capture system based on Event-based sensors and data processing using an algorithm based on a spiking neural network. This algorithm is then run on a spiking neural network

board named *"SpiNNaker"*. This fact allows us to inherit all the stated advantages that these systems provide, allowing a faster and more efficient computation motivated by an event-based workflow.[10] The number of operations required in this system aligns with the number of changes found in the environment. This characteristic allows for a reduction of the standby power consumption and lowers the latency for a real-time system.[29]

Analyzing previous studies, we observe the essential methods we use as the first approach to a solution based on the usage of weights to map each event produced by the DVS to a neuron in the model. The model developed by Seifozzakerini et al.[30] proves a system very similar to the one we aim to obtain in this project, using DVS and Spiking neural networks to detect straight lines. The basis of its work is to create a network of neurons with whom we represent the different parameters that define a straight line based on the Hough Transform. On the basis of this study, we will perform the development of a new algorithm for the recognition of lanes in a road instead of recognizing straight lines in synthetic images.

CHAPTER 3
Methods

A basic definition of the lane detection task involves the following basic steps: the system obtains its input from a sensor located on the front of a car, capturing the real world from the view of the driver. This input data is processed by the algorithm, capable of obtaining the most exact representation of the real world possible. The output obtained represents the positions of the lane markings by using straight lines in a mathematical description. In previous research for this topic in the field of neuromorphic hardware, methods such as edge detection algorithms and the usage of the Hough Transform have been used [30, 4]. We will center our methodology on the Hough Transform algorithm implemented on a SpiNNaker neuromorphic board.

3.1 Dataset

One of the main tasks to be carried out to evaluate and test a new computer vision algorithm is to obtain an accurate and representative dataset for testing the algorithm. It is crucial to ensure this data is obtained with calibrated sensors and has no noise affecting the information it represents. Hence, it is crucial to invest time and effort in revising and developing a suitable data capturing system capable of obtaining meaningful inputs that represent future use cases as tightly as possible.

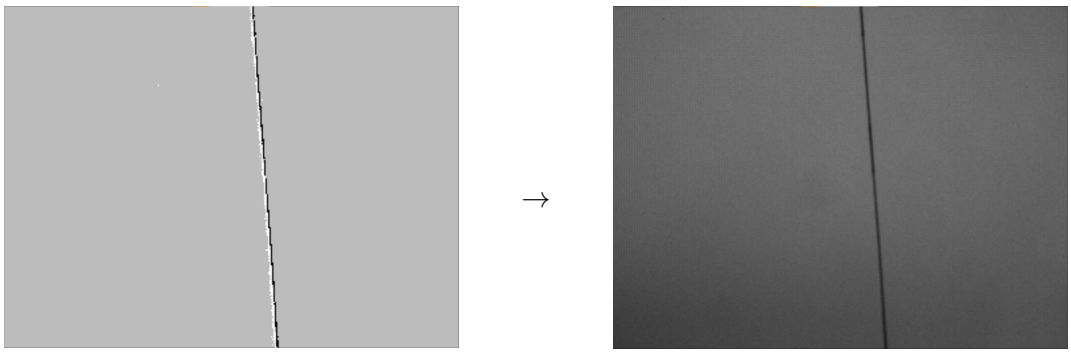
To carry out this task, we use two sets of recorded data to ensure the correct functioning of the algorithm. First, we obtain and use a simple dataset that records straight lines on blank paper with a Dynamic Vision Sensor. This dataset is used to perform a first evaluation of the algorithm's correctness with a simple test case. After this step, we use actual data recorded on real drives. For this, we use a freely available driving dataset.

Analyzing the literature, we find several datasets recorded with different vehicles and in varying road environments. We will use exclusively datasets that provide a recording using Dynamic Vision Sensors, which is a requirement for the algorithm. One example is the *DAVIS Driving Dataset (DDD20)* developed by the University of Zurich, which provides data aimed to help develop algorithms for driving assistance [5]. Another dataset, and the one we have chosen for this project, is the *Multi Vehicle Stereo Event Camera Dataset (MVSEC)* provided by Zhu et al.[31] from the GRASP Laboratory at the University of Pennsylvania. This choice is motivated by the scope of the project. We aim to evaluate the system's performance on drives performed at night in a low-light environment. The chosen dataset was captured on a regular car driving through the city of Philadelphia, Pennsylvania (US).

3.1.1. Obtention of a Dataset of Simple Straight Lines

The initial dataset for the project consists of recordings of a straight line performing a controlled translation movement in front of the sensor. The obtention of this same dataset is performed as part of the project's research scope. For this end, we use a DVS event camera *DAVIS 346 from iniVation*. This camera supports recording events from a resolution of 346x260 pixels and provides additional data such as a Greyscale frame image recording and gyroscopic sensor data.

The setup for the dataset is a drawing of a straight line on a blank piece of paper. We locate this figure in front of the camera to obtain events that correspond to the straight line and can easily be interpreted by the algorithm. For the recording, we use the *Dynamic Vision Viewer* software provided by the manufacturer of the sensor iniVation. The result is an *"aedat4"* data file, which contains the events, and can be imported to our algorithm. The visual output is observed in Figure 3.1. The obtained result allows us to define a basic input for a use case of the algorithm that reduces noise and distortion, ensuring only the most relevant data is processed.



(a) Greyscale Frame **(b)** Events captured by the DVS

Figure 3.1: Process of capturing for DAVIS Sensor

3.1.2. Obtention of a Driving Dataset

Additionally to the basic dataset, we use a more extensive dataset based on a real use case. In this case, we chose the one by Zhu et al.[31]. This dataset contains recordings performed by a sensor array as seen in Figure 3.2 consisting of two DAVIS 346B neuromorphic cameras, which offer a stereo capture with a resolution of 346x260 pixels. Along with these sensors, we find other devices used for data obtention, such as Visual-Inertial sensors, GPS, gyroscopes, or Motion capture cameras. These sensors can be helpful in other applications, such as autonomous driving, where the data obtained can provide ground truth information about the vehicle's position and orientation with respect to the actions taken by the system.[31]

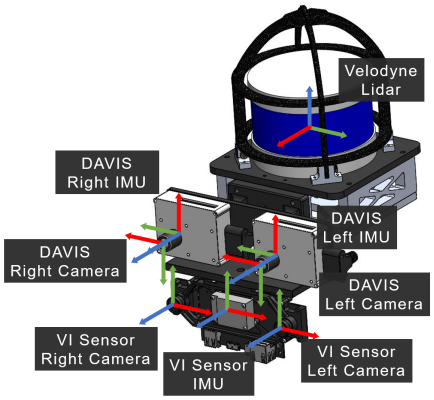


Figure 3.2: MVSEC Dataset Recording setup

This data is then encoded as a hierarchical file (*h5py*) accessible from Python and allows loading only the relevant sensor data on demand. Concerning the available data, this dataset includes over 10GB of pixel events captured from a drive at night, which aligns with our goal of evaluating the algorithm's performance while driving in low-light environments.

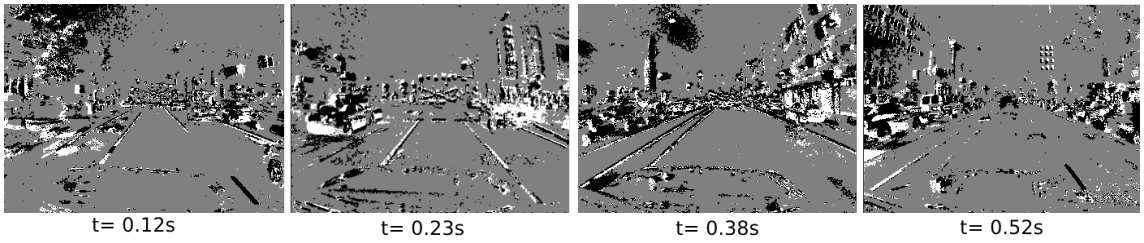


Figure 3.3: Sample of events reconstructed from MVSEC Nightdrive Dataset

3.2 Line Detection

Line detections correspond to one of the fundamental challenges for computer vision. The task of nature is to obtain a mathematical representation of an object with a known geometry, which forms part of an image where other different elements are also displayed. This image has to be analyzed to extract features belonging to a straight line and, therefore, obtain the parameters defining each of these lines, such as its position, length, and angle.

There are different options available throughout the literature of past studies to carry out the line detection task. One of the most effective and efficient algorithms, and for that reason, the one we have chosen as a principle for our project, is the use of the Hough Transform. [32]

3.2.1. Hough Transform

The usage of the Hough Transforms was first presented in the paper by Duda and Hart[32]. This method's main objective is to find and characterize mathematically concurrent lines visible in pictures. It is based on applying a mathematical operation that translates a stream of pixels belonging to a line into a single point in a polar parameter space that represents this same. This 2-dimensional parameter space, also called polar coordinates, encodes the straight line as the angle of the perpendicular line from the origin (θ in

radians), and the distance from the origin (ρ in pixels) with the following relation between them:

$$\rho = x \cos\theta + y \sin\theta \tag{3.1}$$

Figure 3.4: Formula of the Hough Transform [32]

If we apply this transformation to the image, we obtain a periodic curve representing each of the points of the line to the parameter space (Figure 3.5). Each of the points in the curve represent the possible lines that can be formed passing through each individual input point. Given these different input points, if they form a same straight line on the input, the different periodic curves on the output will intersect in a coincident point in the polar space. The parameters resulting of this intersection point represent the detected line's basic features, as can be seen in Figure 3.6. In this figure, we see a crossing point at (2.034 rad, 0 px), successfully defining the original line, which had an angle of $\theta = 26.6°$ (the perpendicular line is of $116.6° = 2.034rad$) and a distance from the origin of $\rho = 0$ px. [32]

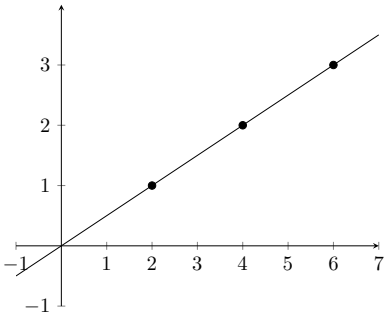


Figure 3.5: Original line setting

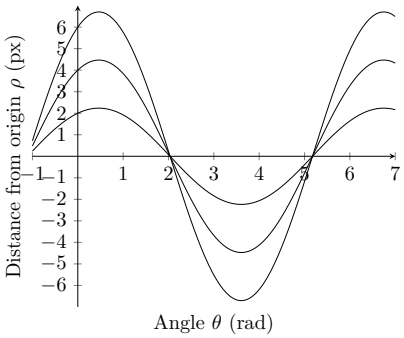


Figure 3.6: Result in Parameter Space

Given this equation, we can define the different straight lines shown in the image. The algorithm performs this task by analyzing the points of the parameter space (θ,ρ) where there have been more coincidences after applying the transform in Equation 3.1 to the active pixels of the image. These points will then represent the line in polar coordinates, which we can then translate back into cartesian coordinates that define the detected line.

3.2.2. Applying Hough transform to Spiking Neural Networks

This same idea presented in the previous section has also been applied to the case of Spiking Neurons, as shown by Wu et al.[11]. This paper shows how the mathematical expression of the Hugh Transform can be translated to a network of neurons. For this end, it is proposed to use a grid of neurons, representing in a 2D space the possible polar parameter values resulting from the transform. Then, a specific connection weight is used between each pixel of the picture and each neuron of the parameter space. In this way, when a certain number of points show a change, the weight for the neuron encoding the line increases and results in a spike being registered on the corresponding neuron. As a result, we observe a spike with the highest potential on the specific point representing the angle θ and the distance to the origin ρ.

We use the work by Wu et al. as a starting point for developing our algorithm. Our algorithm differs mainly from the one presented in the cited paper in obtaining the input

points from the stream of a Dynamic Vision Sensor. The proposed architecture forces the system to process the input sequentially and consider new parameters for the computation, such as the time of each event and its polarity. Additionally, our system needs to perform the desired task when applied to a real car driving scenario.

3.2.3. The implemented algorithm

Based on the previous references, we develop our model to implement the Hough Transform using Spiking Neural Networks to process the input of Event-Based Sensor cameras. The model's basic structure is based on two layers (populations) of neurons, as pictured in Figure 3.7. One layer acts as the input layer, representing the spikes obtained by the events generated by the DVS camera. When a pixel on the camera detects a change, it will produce a spike on the corresponding neuron for that pixel in the input layer.

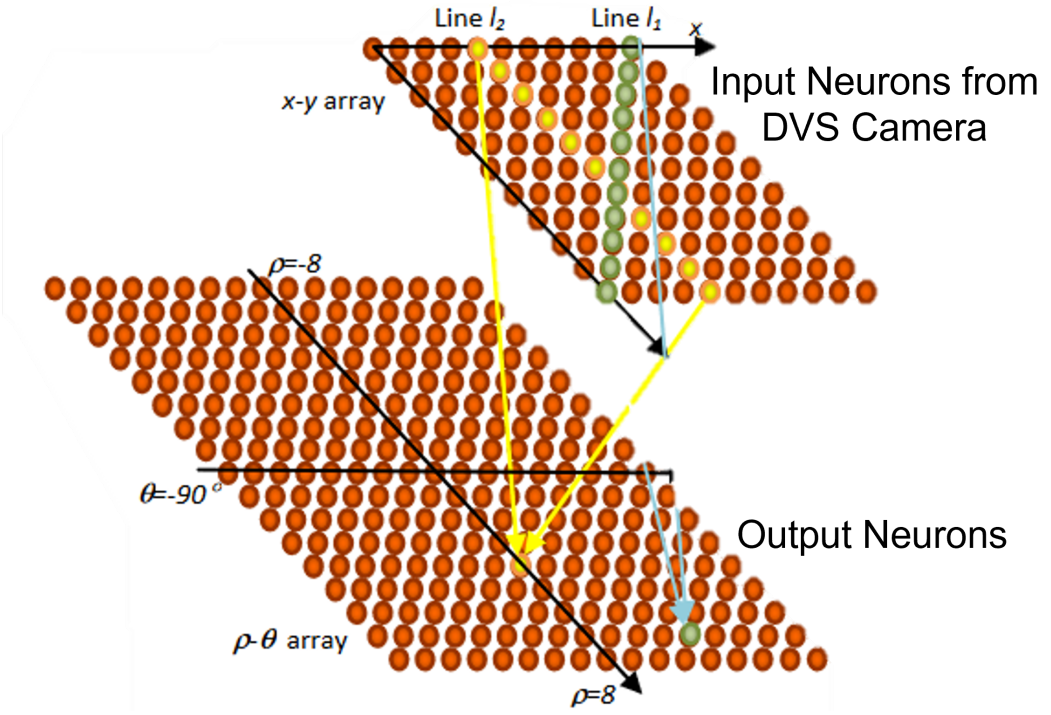


Figure 3.7: Schema for the implemented Neuronal model, based on the model by Wu et al.[11]

Each neuron in the input is connected to the output layer by using a series of connections defined in the connection array. The array's connections are determined based on the Hough Transform in Equation 3.1. For its creation, we use the algorithm presented in Algorithm 1. We traverse the pixels in the input layer and the parameters in the output. When the location of the pixel (x,y) matches the parameters of the output (θ, ρ) according to Equation 3.1, we add the connection from the input neuron encoding that pixel to the output neuron encoding the parameters.

Algorithm 1 Connection Creation Algorithm

```
connectionArray ← [] {Contains tuples (input, output, weight)}
for (x,y) in InputLayerPixels do
    for (θ, ρ) in OutputParameters do
        if ρ == x · cos(θ) + y · sin(θ) then
            connectionArray.append([(x,y), (θ, ρ)])
        end if
    end for
end for
```

Finally, we have the output neuron layer. This layer is formed by a grid representing all the possible parameter values for the lines in the input space. Each of these neurons representing a line will be connected to all the possible pixels in the input based on the connection array. When several points form a straight line on the input layer, they will have a coincident connection targeting a neuron in the output layer. If the sum of all the receiving connections is higher than the specified threshold, then the output neuron produces a spike. The neuromorphic board will then forward each of the output spikes and record them. Each of the resulting spikes defines the parameters of the detected lines in the image.

3.2.4. Accessing and interfacing with the Spiking Neural Hardware

Access to the Spiking Neural Network board is provided by the Neuro Computing Systems (NCS)laboratory of the Kungliga Tekniska Högskolan (KTH) of Stockholm. The hardware, as defined in section 2.1.1, consists of a SpiNNaker 5 board with 48 computing chips. The board is managed and programmed through an Ethernet connection by a gateway computer to which we connect through a remote Secure Shell (SSH). The input and output of data to the spiking board are performed through a Gigabit Ethernet connector, which is accessed by the same gateway computer after loading the program specification into the board's controller.

The programming of the script is done in Python 3.8, using the *spynnaker8*[21] library for communicating with the SpiNNaker board. For accessing the files encoded by the Event-Based Sensor, we use the *Dynamic Vision (DV)* library provided by the camera manufacturer iniVation.

3.3 Evaluating the obtained data

After developing the algorithm, we face one of the crucial challenges of the project, proving that the results obtained are meaningful and accurate for the objective of this report, which is evaluating a new algorithm for lane detection. To accomplish this evaluation task, we need to use a verified model that can serve as the ground truth for the comparison. With the results from both models, we can compare and establish an objective measurement of the accuracy and performance of our new model.

After evaluating different alternatives of lane detection models, we choose the algorithm proposed in the library *OpenCV 4.5.5*[33] for the Hough Transform. We motivate this choice because this algorithm represents a similar implementation to the one carried out in our project but uses a direct application of the mathematical formula. The OpenCV implementation is based on counting votes for each parameter in the polar coordinates of the Hough Transform. That differs from our algorithm that implements a similar process

using a specific set of connections between layers in a Neuronal Network. The main differences reside in how the data is inputted into the algorithm. Another difference is that the OpenCV algorithm is based on analyzing static frames obtained from conventional cameras. The algorithm presented in this report uses events in a neuromorphic environment, from whose it extracts the detected lines' parameters.

To compare both algorithms, we will analyze the results obtained after processing the input of two datasets of different nature. As presented in section 3.1 The two datasets used to assess the algorithm are:

First, we study the dataset consisting of simple straight lines, recorded as part of the scope of this report and used to evaluate the algorithm's performance in an easy and predictable scenario. The results from analyzing this dataset serve as a preliminary idea for the performance we expect to obtain when assessing the more complex input of the real-driving dataset. Additionally, it helps establish a baseline for an objective measure of the accuracy against the results obtained by the other algorithm that also implements the Hough Transform.

The second part of our study results examines the results of applying the same model used previously for detecting road markings in a driving dataset recorded in a real scenario (MVSEC Dataset [31]). The same techniques explored in the previous case are applied again in this section to compare our model's performance against other implementations already available.

Furthermore, we use two techniques to evaluate the two datasets described in this section: a visual evaluation based on the representation of the lines and an analytical evaluation of the parameters. We will introduce and detail these processes in the following two sections.

3.3.1. Visual evaluation

We perform an initial evaluation of our model's results by using a visual representation of the resulting data after analyzing the datasets to compare them against other models. To accomplish this goal, we implement a script that represents the parameters obtained as a result of applying the Hough Transform (θ,ρ) 3.1. We draw a straight line from the polar coordinate space into the cartesian grid that represents the predicted result with these parameters. This line is then overlapped on the extracted frame by the CMOS camera. The line representation operation is based on the formulas of Figure 3.8 for obtaining a straight line defined by an origin point (x, y) and its slope m.

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \rho * \cos(\theta) \\ \rho * \sin(\theta) \end{pmatrix}$$
$$m = tan\left(\theta + \frac{\pi}{2}\right) \tag{3.2}$$

Figure 3.8: Equations defining the parameters of a straight line on cartesian coordinates given the polar coordinates (Hough Transform parameters)

We created a script using the formula in Figure 3.8 so that we can represent for each image frame recorded by the dataset the lines representing the values resulting from each Hough Transform model. In the case of the OpenCV model, we plot the line with the most votes for each analyzed frame of the input.

In the case of the neuromorphic model, we note that the lines obtained by the model have no direct translation over a specific frame of the dataset, as the DVS generates the

events asynchronously as the changes are detected. For this reason, to represent the lines along with the captured images in a meaningful way, we grouped the events that occurred between the timestamps of two captured frames. We represented each range of resulting events in a single line by averaging the value of the parameters belonging to the same group.

3.3.2. Analytical Evaluation of Performance

One key result we aim to obtain from the study is a quantifiable and objective measurement of the accuracy of our model. This is needed to assess the suitability of the algorithm for solving the given task with the highest efficiency and precision. To that end, we use an analytical approach based on the numerical analysis of the statistical parameters resulting from the analysis of the dataset.

The evaluation of the results is done by comparing the parameters (θ,ρ) obtained by the model against the ground truth, from which we measure the error deviation of each result. The ground truth is established as in the previous section by using the result of applying the *OpenCV* Hough Transform algorithm to the frames captured in the dataset by the conventional grayscale camera.

The analytical parameters are obtained by comparing each frame's results of both algorithms with the ground truth. In order to obtain a single metric that represents the performance for the whole dataset, we **accumulate** the differences along the frames to result in an **averaged value** along the length of the dataset. For this reason, it is essential to ensure the length of the dataset is large enough to obtain a statistically meaningful sample when analyzing the accuracy.

The previously mentioned operations will be performed using the *Pandas* Python library for data manipulation. This tool allows us to access and statistically aggregate the results obtained from the data analysis to evaluate the results obtained by each algorithm by comparing the parameters obtained in each frame of the picture. This aggregation of the event data to frames is a limitation imposed by using the *OpenCV* Hough Formula based on the analysis of images, which has an output cadence lower than the outputs of the event algorithm. As a result, we effectively lose temporal resolution on the algorithm's output parameters.

3.4 Power efficiency

This study looks to motivate the advantages and prove de adequacy of the neuromorphic hardware for being applied to new algorithms. One of the main advantages of the development using this platform is the low power consumption of the computing boards as well as the high processing speed allowing real-time computation. As part of our results, we provide an estimation of the power consumption of our system.

Due to the limitations of our project, the implementation of a physical device for the experimental measurement of the power drawn by the system falls out of our scope. For this reason, we use the results from previous studies performed on the same SpiNNaker 5 board that we use in our research. The paper by Painkras et al.[34] shows experimental figures detailed to the lowest chip level that we will use as a base to approximate our consumption calculations scaled to the resources used in our implementation. As well, we base our figures on the efficiency analysis for high-performance computing of SpiNNaker studied in the paper by Sugiarto et al.[35]

The SpiNNaker 5 board is formed by 48 chip multiprocessors, each of those capable of simulating up to 1800 neurons (18 cores with 1000 neurons per core). Each chip is rated at a peak 950 mW power consumption when operating at 250MHz, and 250 mW when idle [35]. Additionally, we consider the figures for the SDRAM consumption, which is 170 mW per chip, and the off-chip connections, being 6.3mW for each of them.[34]

We will also consider the power specification for the event vision sensor used in our project, which consists of the *DAVIS 346 from iniVation*[36]. In the specification sheet, we can find the power consumption for the DVS to be in the range of 10 to 30mW without taking into account the pixel sensor (which is not necessary for obtaining the event data). To simplify our study, we consider the higher end of the consumption range as the reference value for our results.

With these figures, we use the resource usage calculated for our spiking neural network to scale an approximate measure of the power in Wh required to analyze the whole stream of events contained in the MVSEC dataset for a real driving environment. We will use the obtained result against the calculated power usage by other production platforms devoted to the task of lane recognition.

3.5 Time latency evaluation

Another metric we include in our results is the evaluation of the latency for the outputs of our system. In this case, we obtain an approximation for the measurements by employing an indirect analysis of the spike stream. This method is based on analyzing the output from a data synthetic data stream that we have generated based on the previously analyzed straight line dataset.

To obtain a precise measure of the latency, we create a new dataset based on a slice of events from the previous dataset where the line moves at a constant speed in a unique direction. This slice is duplicated but inverting its direction and appended after the original slice. The resulting dataset shows a symmetrical event stream where we can pinpoint precisely the exact time when the line changes direction.

Using this dataset, we can know precisely the timestamp when the line changes its parameters to the opposite angle. Then, by analyzing the output from the spiking neuronal network, we can look for the point where this change is located and compare its timestamp against the breakpoint we used when creating the dataset, from which we have its exact time stamp. The difference between these timestamps is an approximation for the time it takes to obtain the output of a line since it is first seen on the input stream of the Spiking Neuronal Network.

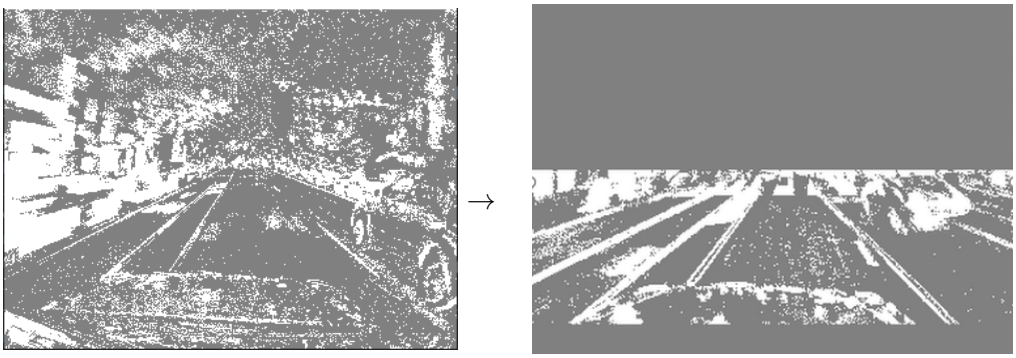
3.6 Optimizations

One of the key aspects we need to ensure for our algorithm to be competitive against other previous implementations is its ability to produce a correct and accurate output in the shortest time possible. Ensuring that we achieve the highest possible efficiency in every step of the computation is essential. Using the optimization techniques presented in this section, we allow the algorithm to result in its full potential and increase the efficiency concerning the number of calculations and connections in the model.

3.6.1. Reduce input space

To ensure a faster computation, we reduce the size of the data inputted to the algorithm. The process is based on cropping the image to reduce the number of pixels from the DVS sensor that can generate events. By removing the parts of the image where we don't expect to find road markings, we ensure the computer analyzes only the relevant data. This also leads to a lower probability of obtaining a false positive originating from another object in the image resembling a road marking while we reduce the time and clock cycles needed to analyze in a given period.

Furthermore, we minimize the overhead of establishing each required connection between layers of neurons. This is an important measure in our project as we are using two layers with an input of 89960 neurons. To recognize the road marking lines, we just need a slice of the whole image input, which contains the lanes of the road. By reducing our input space by cropping the upper and lower parts of the image, we reduce the number of connections down by a 90% while still being able to perform an accurate lane detection task. We can observe the resulting frames from the reduction in Figure 3.9.



(a) Original Input Space **(b)** Resulting input space after cropping

Figure 3.9: Comparison of original and reduced input

Thanks to this optimization, we ensure faster running times without sacrificing accuracy and avoiding noise originating from other image features that can difficult the obtention of a valid result.

3.6.2. Temporal subsampling of the input data

Another technique that also helps to process the extensive amount of events produced by the DVS camera is the use of temporal subsampling for the event stream. The temporal subsampling reduces the number of events given a specific time by just using a subset of the original stream of events. In our case, the datasets consist of more than a million events per ten seconds of recording. This fact makes it much more difficult for the computing board to keep up with the processing of each of these potential spikes in real time, causing the loss of data by the overflowing of input queues in the processors and time delays on the output.

To avoid this problem, we perform simple temporal downsampling of the input stream. This method is based on taking one sample for every group of 10 events. In this way, we reduce by 90% the number of packets sent to the computer while still keeping the main features in the image, as can be seen in Figure 3.10, where the road lines still represent a dense cloud of points that the neurons can recognize.

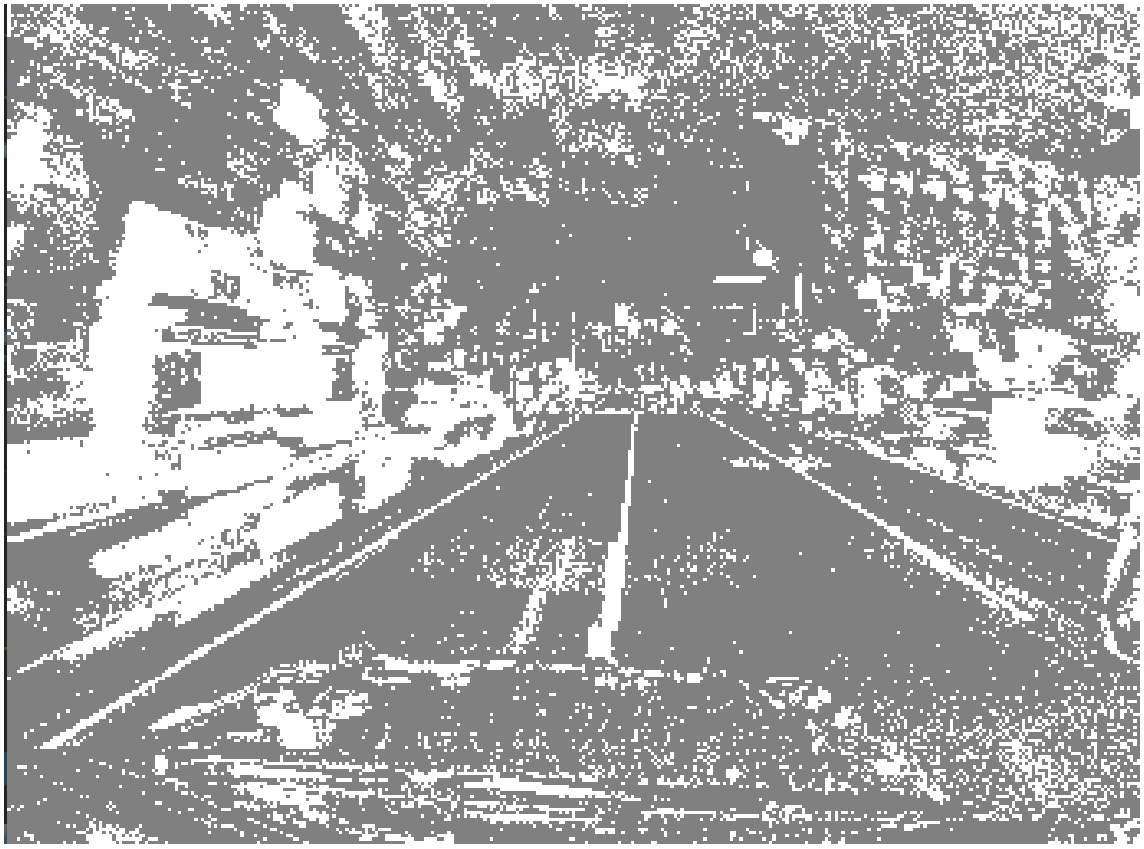


Figure 3.10: Result of x10 downsampling of the input

3.6.3. Output Data processing for the Real Driving Dataset

The analysis of the MVSEC requires additional post-processing of the results to obtain a meaningful result that reduces the noise and ensures we get a significant and comparable system when compared against the *OpenCV* implementation.

When using a real-life recording, we face an additional challenge in processing the output data of the dataset to classify and filter the most relevant information captured from the scene. This processing was not required in the previous cases as we had a reduced input with a smooth background (blank sheet of paper). For the case of the driving dataset, we find a more significant amount of elements on the input, which have to be tracked accordingly and selected to obtain the desired results.

For this reason, we will use a clustering algorithm on the output parameters of the model, as presented by Seifozzakerini et al.[30]. This procedure consists of grouping those events produced sequentially as belonging to the same line. Additionally, we establish a threshold for considering a line on the output, filtering noisy points generated by objects from the environment different from the lane markings. The clustering we used is illustrated in Algorithm 2. In this algorithm, we use a tuple (θ,ρ, expiryTime, numberOfSpikes) representing each of the lines of the clusters. Once we receive a spike, we classify it based on the shortest Euclidean distance between its parameters to those of each line. If the difference is smaller than a threshold, we consider the spike belongs to the line with the smallest distance. Then we update the parameters of the cluster by adding a weighted sum (based on the parameter α) of the spike's coordinates. If the distance is larger than the threshold, we consider the spike belongs to a new cluster. The parameter (*expiryTime*) keeps track of the last time a cluster received a spike so that the cluster can be removed after a specific time. Finally, the parameter (*numberOfSpikes*) stores the number of spikes

included in a cluster. This parameter is used as a threshold for choosing the best line as an output in the results.

Algorithm 2 Event clustering algorithm

```
clusterLine ← [ (lineClusterTuple) lines to detect]
for spikesFrame in frames do
  for spike in spikesFrame do
    • Delete all clusters in clusterLine such that the Expiry time is previous to the current
    rho, theta ← [spike.rho, spike.theta]
    distance ← EuclideanDistance[(rho, theta), clusters]
    minDistance, indexMinDistance ← MinimumArg[distance]
    if minDistance > threshold then
      # If the distance is greater than the threshold, create a new cluster
      clusters[EmptyIdx] = [rho, theta, currentTime, 1]
    else
      # If the distance is less than the threshold, update the cluster
      closestCluster ← clusters[indexMinDistance]
      closestCluster.rho ← closestCluster.rho*α+rho*α
      closestCluster.rho ← closestCluster.theta*α+theta*α,
      closestCluster.expiryTime ← currentTime
      closestCluster.numberOfSpikes += 1
    end if
  end for
end for
```

CHAPTER 4
Results

This chapter presents and evaluates the results from running the model on the proposed datasets. For this task, we implement the methodologies explained in **Chapter 3**. The results are divided into two sections, covering the performance observed by the algorithm when run on the two different datasets. The same model parameters are maintained through all analyses to establish an accurate and reliable comparison between both methods. The basic parameters that we can use to tweak our model are based on the biological properties of the neurons, in this case, a neuron following the Leaky Integrate and Fire (LIF) model. The parameters used for the neurons are presented in the table Table A.1 and model the basic behavior of the neurons after receiving an input from the vision sensor data stream.

With the resulting data from both dataset evaluations, we further analyze the power consumption and the latency shown by the model on the processing of the Driving Dataset to establish a complete comparison against conventional algorithms performing the same task.

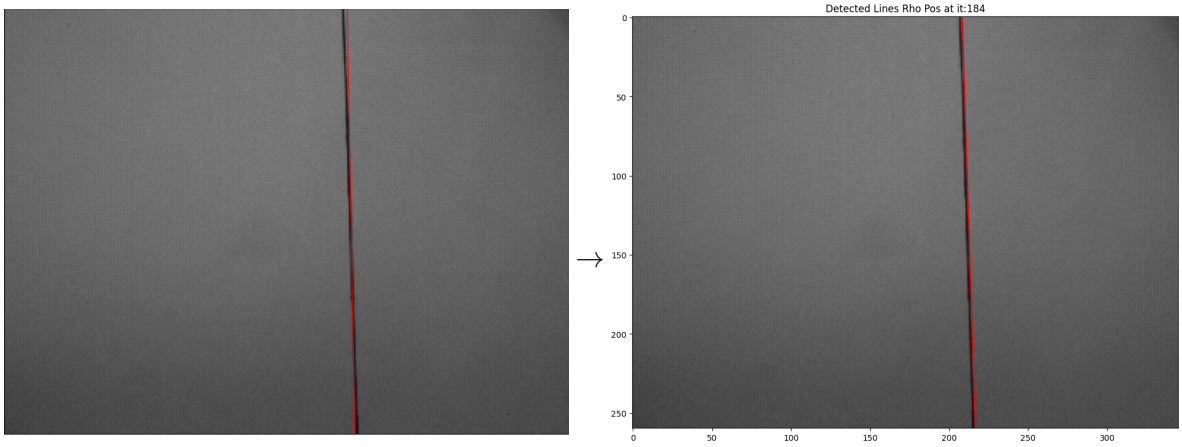
4.1 Results from Test Cases

4.1.1. Straight Line

The first dataset we analyze is formed by a straight line drawn on a white sheet of paper. This serves as calibration and proof of work for our algorithm and allows us to tweak the parameters to be used later to analyze the real MVSEC driving dataset.

Visual Evaluation

The results observed from the evaluation of the sample dataset can be evaluated with two main approaches, as explained in the methodology. In the first step, we plot the obtained data visually and overlay its results against the data obtained by the algorithms used in previous studies. We observe this comparison in Figure 4.1, consisting of two visualizations extracted from a specific frame of the dataset, where the predicted straight line is overlaid in red over the corresponding frame (with the line in dark blue). In Figure 4.1a, corresponding to the *OpenCV 4.5.5*[33] implementation, we can see how the line fits entirely over the line. In contrast, in Figure 4.1b, corresponding to the neuromorphic model results, the resulting line is slightly skewed to the right with respect to the input throughout its full extension. However, one remarkable finding we can extract from the analysis is that the detected angle for the line matches in both cases, as we show in the next section.



(a) Line obtained with *OpenCV* Hough Transform algorithm

(b) Line Obtained with our Hough Transform model using Spiking Neural Networks

Figure 4.1: Visual comparison between OpenCV's Hough Transform algorithm for straight line detection (red line on 4.1a) against the implementation using neuromorphic hardware on SpiNNaker (red line on 4.1b). Results from using a single straight line dataset recording(green background and blue marked lines). Animated video can be accesed at B

As can be observed in Figure 4.1, the accuracy obtained for our algorithm is similar to the one obtained by the Hough Transform applied to a visual static frame in the Python *OpenCV*. This preliminary visual observation of the results of the problem shows the correctness of the results obtained by our model. This result gives a baseline of what we can expect from the numerical analysis performed in the next section.

Analytical evaluation

After a first visual approach to the evaluation, we perform an analytical study of the results obtained by the model. To this end, we extract the numerical parameters that constitute the results for each algorithm to establish the accuracy of our model. With this study, we settle an objective statistical measure for the resulting values and evaluate how the metrics differ on average, as presented in the methodology (3.3). We obtain the averaged values by accumulating the parameter's results along the length of frames of the dataset, in this case, consisting of 473 images that correspond to 20 seconds of recording.

We obtain the main variables for our study from the model's output parameters, consisting of the angle (θ) and displacement (ρ) of the line. These values are compared to the ground truth, generated using the *OpenCV 4.5.5*[33] library. The results can be seen in Table 4.1:

	Mean difference	**Std Deviation**	**Quantile 90**
Theta (θ) º deg	0.5134	0.3703	1.1598
Rho (ρ) pixels	2.2957	1.7686	4.6441

Table 4.1: Analytical results of the averaged parameter difference between the Spiking algorithm and the *OpenCV* implementation of the Hough Transform over 473 frames for each model. Animated video can be accesed at B

We start our study by centering on the angle of the line detected by our algorithm. As we anticipated in the visual inspection Figure 4.1, we can confirm from the analytical data (Table 4.1) that the model was indeed successful in the task of recognizing the line angle on the input space. The neuromorphic-based model obtained values for the angle

of the lines in the image that differed by less than 1 degree from the baseline established by the *OpenCV* algorithm. This error represents a 0.28% over the parameter space for θ (180 degrees).

Observing other statistical parameters, such as the standard deviation of the angle difference, we see how the error between the models has not a significant dispersion. This figure motivates that the results are consistent, showing a predicted line similar to the one obtained by conventional methods. Analysis of the quantiles also shows that 90% of the difference values are lower than 1.15 degrees, proving the results' accuracy and correctness.

Furthermore, suppose we study the variation obtained on the displacement of the line, represented by the parameter Rho (ρ). In that case, we can observe how the variation is, in this case, higher than the one obtained for the angle. This result is also visible on the previous Figure 4.1, as the line is slightly displaced to one side of the image and does not fall directly over the detected frame. Analyzing the dispersion of the model, we can observe that the standard deviation shows a reliably contained range of results, which still can represent and successfully detect the position of the line in a given space. These results have to be taken into the context of the range of values for the parameter Rho, which expands from the origin (0) to the furthest pixel distance (433); in this way, the error obtained represents a 0.5 % error over the possible parameter space.

Analyzing these results together, we can see how the numerical results obtained by our model to track a straight line do not have a significant deviation when compared to the results established as a baseline by the *OpenCV* implementation of the Hough Transform. This result further encourages the usage of the Spiking Neural Networks, as in this case, the outputs obtained by the algorithm were given in real-time, with a negligible delay, as analyzed in section 4.2.4.

4.1.2. Two Angled Lines

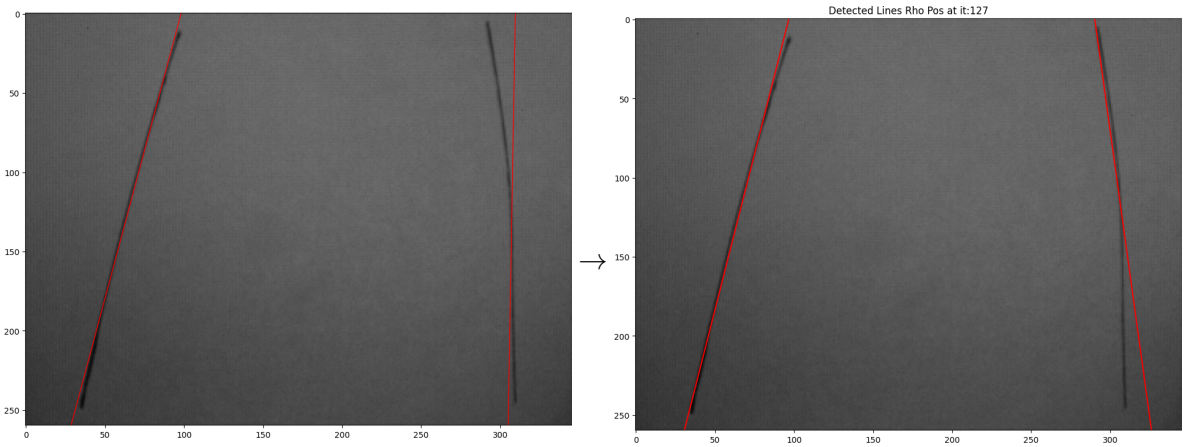
After proving in the previous section the ability of the algorithm to successfully track a single line in a sheet of paper, we move forward in complexity by using a sample dataset that represents more accurately the inputs expected in the real driving dataset. For this reason, we analyze a dataset consisting of two straight lines situated in a white paper sheet and with an inclined angle in between them. This setup tries to represent the view of the road lane markings in the same way they can be seen from the dash of a car.

This dataset represents a much higher complexity that results from the fact that the algorithm has to be capable of separating the two valid outputs representing the two lines on the paper. This task needs to be performed while avoiding other multiple points generated in the output space due to noise that comes, for example, from the fact that the thickness of the lines makes them occupy several pixels from the input. To deal with this challenge, we divided the output space into two clusters, based on the parameter rho (distance from the origin to the line), to effectively obtain a unique point in space representing each line without the other one interfering with the result.

Visual Evaluation

As we did in the previous case, we will start our study by analyzing the visual output of the algorithm and comparing it with the result obtained by the *OpenCV* model. An example frame extracted from the model can be seen in Figure 4.2. In this case, we can observe how the lines represented do not show a straight line, but because of the distortion of the lens, the lines are slightly arched inwards on the top of the image. This fact brings

the dataset closer to reality while still maintaining a controlled environment and allows us to match the expected results from the analysis of the real driving dataset's recordings.



(a) Lines obtained with *OpenCV* Hough Transform algorithm

(b) Lines Obtained with our Hough Transform model using Spiking Neural Networks

Figure 4.2: Visual comparison between OpenCV's Hough Transform algorithm for straight line detection (red line on 4.2a) against the implementation using neuromorphic hardware on SpiNNaker (red line on 4.2b). Results from using a dataset recording showing two angled straight line representing road markings (background in grayscale). Video of the output available in Annex B

The preliminary analysis of the results for this dataset extracted from the visual comparison shows us how the accuracy of the model's prediction is now significantly lower than in the previous case. Examining the line's features in Figure 4.2b, corresponding to the output of our neuromorphic algorithm, we can observe how the detected left line closely overlays the line in the frame. However, the right line shows a more significant difference between the resulting line and the actual position of the line in the frame.

We can further analyze these results by observing the performance given by our baseline algorithm on Figure 4.2a. In this frame, we can observe how the line tracking of the *OpenCV* algorithm slightly diverges from the actual result in the case of the left line. If we pay attention to the right line, we can see how each algorithm has taken different points as references for drawing the result. In the case of *OpenCV* (4.2a), the points taken for drawing the line belong to the lower half, resulting in an almost vertical line. Meanwhile, the result obtained by the spiking model (4.2b) has taken as reference the top points of the right line.

Analytical evaluation

As we did in the previous case, after the visual analysis of the results, we will study the analytical parameters obtained from the results for each model. For a more detailed study, we will analyze the resulting parameters for both lines in the pictures to establish an individualized comparison with each line in the dataset. The obtained results are presented in Table 4.2 for the left line and Table 4.3 for the right line of the image.

	Mean difference	**Std Deviation**	**Quantile 90**
Theta (θ) º deg	0.9653	0.6869	2.0962
Rho (ρ) pixels	5.0771	3.9066	12.6869

Table 4.2: Analytical results of the **left line** using the averaged parameter difference between the Spiking algorithm and the *OpenCV* implementation of the Hough Transform over 219 frames for each model

From the analysis of the figures in table Table 4.2, we can observe a very similar result to the difference one obtained for the single straight line in the previous section 4.1. Both the angle (θ) difference and the distance from the origin (ρ) show to be within the same specs as in the previous study, showing a slightly higher error in both parameters. We can further conclude that the results are robust and maintained along the analyzed dataset's length by analyzing the dispersion values for both parameters.

	Mean difference	**Std Deviation**	**Quantile 90**
Theta (θ) º deg	7.0305	1.0700	8.5783
Rho (ρ) pixels	19.7749	8.1947	36.9787

Table 4.3: Analytical results of the **right line** using the averaged parameter difference between the Spiking algorithm and the *OpenCV* implementation of the Hough Transform over 219 frames for each model

Analyzing the table from Table 4.3, we can observe how the difference for the results of the Spiking algorithm grows even further than in the previous case. This fact is also seen clearly in the visual study of the results Figure 4.2 where both algorithms find difficulties to precisely track the line belonging to the right side of the image. Furthermore, analyzing the animated video depicting the evolution of the tracking, we can observe that both algorithms lose track of the line at some point in the stream due to the lack of contrast with the background.

We can motivate this observation by the fact that the marking of the right line is noticeably lighter than the left one. This slight difference leads to difficulties for both algorithms to be capable of obtaining enough data points to trigger the parameters for the line according to the Hough Transform.

In conclusion, we obtain another example of the performance of the algorithm, in this case, when applied to two angled lines. We have shown how the performance of the implemented algorithm remains constant for the case of the left line. On the right line of the image, we have proven the tracking is still being successfully done but obtaining a higher margin of error.

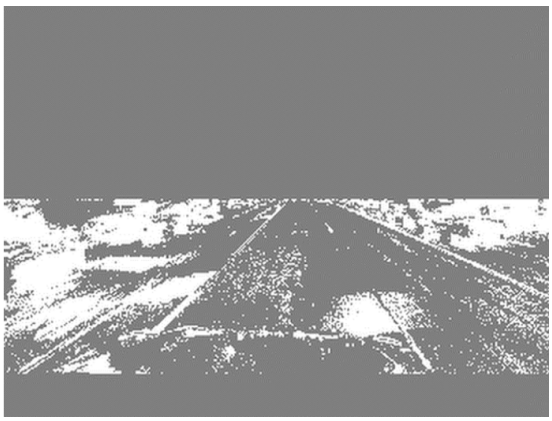
4.2 Results from Experimental analysis

After studying the results of the application of the algorithms to the basic datasets and concluding its ability to track the lines in a controlled and simple environment successfully, we test the capabilities of the model in a broader and realistic setup. For this reason, we are going to use the recordings of the MVSEC dataset [31]. From this dataset, we will select the drives regarding the scope of nighttime driving. The used drive recording consists of a file with more than 87 million recorded events showing a car driving down a street in a city and surrounded by other vehicles at night. We motivate this choice to prove one of the advantages of using dynamic vision sensors when recording low lit environments.

For this study, we will follow the same structure as the previous cases, first visually analyzing the obtained results and then centering on the analytical parameters. To reduce the computing time while getting meaningful results, we analyze the first 20 million events produced by the event camera, corresponding to the initial 2 minutes and 15 seconds of the recording.

4.2.1. Visual Analysis

As stated in the methodology 3, we will make an input space reduction to be able to process the vast amount of events generated by the dynamic vision sensor. We can observe an extract of the process taken by the algorithm in 4.3. Figure 4.3b presents the preliminary raw output obtained from the Spiking neural network when given as input the events corresponding to the recording of a real driving dataset. This output shows the spikes produced by the neural network according to the parameters of the Hough Transform. The horizontal axis shows the distance from the origin (ρ), and the vertical axis is the angle of the line (θ).



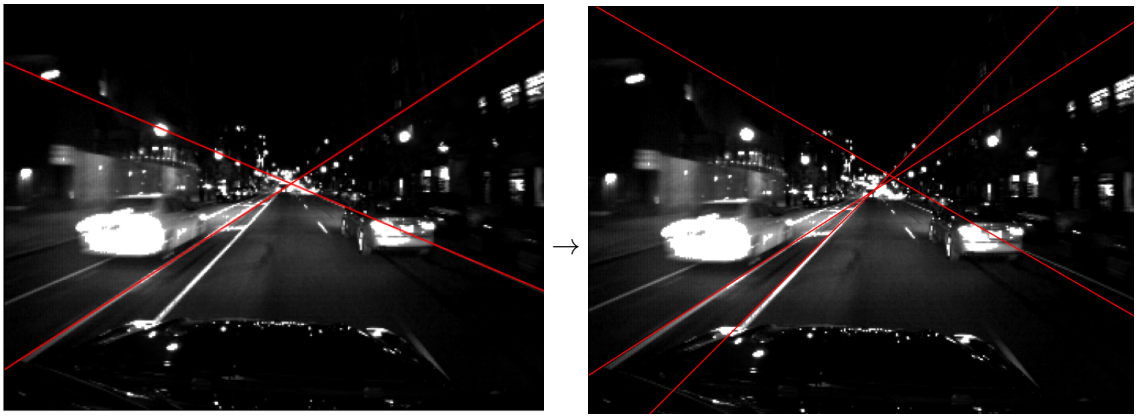
(a) Events recorded by DVS Sensor and cropped according to section 3.6.

(b) Results from the parameter space (ρ, θ) obtained by the Spiking Neural Network from the events

Figure 4.3: Data processing carried out by the algorithm from the obtention of the events from the DVS Sensor to the output generated by the Neural Network according to the parameters of the Hough Transform equation 3.1

From the Figure 4.3, we can observe how the algorithm is capable of producing a cleanly structured stream of spikes out of the stream of events recorded by the cameras. We can state that the algorithm can detect several features corresponding to straight lines in the images. The next step to obtaining a valid and comparable output is to divide the points into groups (clusters) so that we can successfully represent different individual lines present on the input.

After verifying the output resulting from the algorithm, we move on to represent the lines by translating the points in the polar coordinates of the parameter space into the cartesian coordinates that define the detected lines. Furthermore, we match the spikes obtained along the duration of a recorded frame to establish a visual result to evaluate the algorithm's accuracy. In the following Figure 4.4, after applying the clustering algorithm explained in the methodology 3.6.3, we can obtain the stream of frames with the detected lines overlaid above the frames (in grayscale). In this image, we can directly compare the results obtained by the analysis of the static images by *OpenCV* against our neuromorphic model.



(a) Resulting detected lines with *OpenCV* Hough Transform algorithm

(b) Resulting detected lines with the Hough Transform model using Spiking Neural Networks

Figure 4.4: Visual comparison between OpenCV's Hough Transform algorithm for MVSEC line detection (red line on 4.4a) against the implementation using neuromorphic hardware on SpiNNaker (red line on 4.4b). Results from using a dataset of night drive recording (black an white frames). Video with side by side comparison available at the Annex B

Observing the extract of a frame of the video presented in Figure 4.4, we observe the red lines that represent the detected lines for each analyzed algorithm. For the case of the *OpenCV* Figure 4.4a, we find a general trend in the recording where the number of detected lines tends to be lower than with the spiking neural implementation of the algorithm, shown in Figure 4.4b. This fact is explained by one of the main advantages we wanted to prove from the usage of event-based cameras. The higher dynamic range these sensors offer makes the sensibility to elements in low lit portions of the image much higher than the information obtained from a black and white recording.

Additionally, as expected, the spiking algorithm seen in the figure 4.4b shows a higher error than the previous "clean" datasets. The error can be seen in the frame succession for the tracking of the right-hand line's angle (seen in the video presented in Annex B). This result can be motivated by the nature of the situation where there are many objects presented simultaneously. The line is also occluded during a significant proportion of time by cars passing on the right side, which complicates the obtention of a large enough amount of events to define the line.

4.2.2. Analytical Evaluation

As done in the previous cases, we now analyze the differences between both algorithms based on the analytical parameters obtained by the computation of the Hough Algorithm. The following Table 4.4 shows the results extracted from the parameters detected for the left and right lanes, as shown visually in Figure 4.4.

Observing the left lane results (first section of Table 4.4), we can conclude that the difference obtained in the angle (θ) of the line is significantly higher than the one obtained in previously studied datasets. However, it still represents a 2.28% error accounting for the whole parameter space for the angles. The distance of the line to the origin (ρ) also shows a significant difference from the one experimentally obtained in the previous sections, with a lower increase than the one observed in the angles, leaving it in a 1.01% of the entire distance range of 433 pixels.

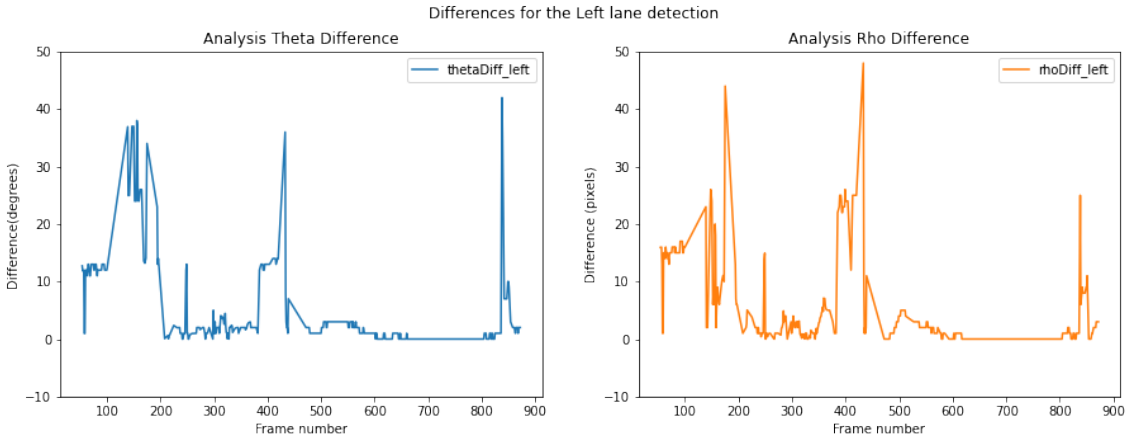
On the other side, analyzing the results from the right lane, we can observe an even greater difference. The obtained values still show a tracking being performed by our algorithm and dispersion figures that show a robust detection of the parameters. The error

\textbf{}	\textbf{Mean difference}	\textbf{Std Deviation}	\textbf{Quantile 90}
		Left Lane	
Theta (θ) º deg	3.8733	6.8962	14.0000
Rho (ρ) pixels	3.0753	6.8687	22.0000
		Right Lane	
Theta (θ) º deg	10.6240	3.2888	14.0000
Rho (ρ) pixels	33.9864	10.6515	45.000

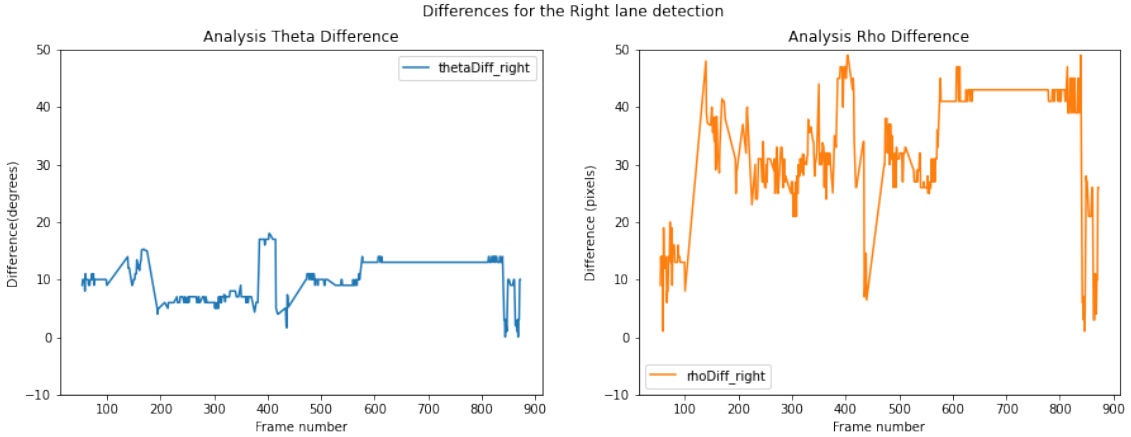
Table 4.4: Analytical performance results for the detection **left lane** using the averaged parameter difference between the Spiking algorithm and the *OpenCV* implementation of the Hough Transform over 1400 frames for each model

on the angle (θ), in this case, represents a 5.91 % of the space parameter, and the error of the distance (ρ) is 7.84 % with respect to the parameter's possible values. Additional analysis of the dispersion figures for the parameters corroborates how the obtained output in the results of this dataset is considerably less robust and concentrated than in the previous cases.

We can further study the progression of the errors for both lanes in a graphical plot, corresponding to Figure 4.5. This plot shows the differences for each lane and the parameters studied along the succession of frames that form the driving dataset.



(a) Differences between Spiking model and OpenCV along the captured frames for the *left lane*



(b) Differences between Spiking model and OpenCV along the captured frames for the *right lane*

Figure 4.5

Observing these plots, we confirm the results shown in the analytical figures. Figure 4.5a shows how the error in both theta and rho matches the averaged values while also showing some outlier values, especially at the start and middle section, where we can see how both parameters are affected by the higher error. On the other side, comparing this plot to the right line results in Figure 4.5b, we observe how the average and the maximum error values are significantly higher than the previous plot, especially for the error concerning the parameter ρ, where we find an overall higher mean value. We will further motivate the meaning of these outlying results in the Discussion section 5.

4.2.3. Power efficiency

One of the main advantages of neuromorphic computation shown in previous research was that the systems offered low latency and high power efficiency for the obtained results. In our studies, we have observed similar results in the computations obtained by our model. As stated before, we used a single SpiNNaker 5 board as the main computing platform. For the MVSEC dataset detection, our system used 99190 neurons (21250 input neurons and 77940 neurons on the output layer).

The SpiNNaker 5 board parameters are described in 3.4. Given the number of used neurons in our model's layer, we use 6 of the 48 chips available on the board. Each chip has a power draw of 950 mW, adding up to 5.7 W of power used by the active chips. The leftover 42 chips will be idle, using a total of 10.5 W. Aditionally, we consider the number of connections between the neurons, consisting of one connection per chip, as all the neurons from the input layer will have at least one synapse with the output so that the power usage will be 0.37 W for the six chips. Finally, we also take into account the 0.03 W used by the vision camera (*DAVIS 346 from iniVation*) to record the events.

The total approximation from the previous data is **16.23 W** of power, which along with the simulation time of 135.88 seconds, results in total power consumption of 0.61 Wh for the processing of 20 Million events from the MVSEC dataset with the Spiking Neural Network. To put the results in perspective, the specifications given by Tesla for their current self-driving computer in August 2019 show a power draw of 72 W [37]. This figure results in 2.712 Wh of power consumption considering the same time is taken (as the analysis is done in real-time as in our system). This value represents a tenfold increase in consumption compared to the system we researched in this report.

4.2.4. Latency Evaluation

Other interesting data that we can obtain from our study is the latency, in ms, that it takes for our algorithm to process an input and get the corresponding spike. For this end, we will use the method proposed in section 3.5, where we analyze a newly created synthetic dataset to evaluate the time when our algorithm detects the change of direction of a line.

For this case, we obtained a slice of 62.4 ms from the Straight Line dataset of the previous subsection 4.1.1. This slice of events is then duplicated and appended to the original to obtain a symmetrical stream of events with a breaking point at 120 miliseconds. We then analyzed the dataset with the spiking neuronal model and evaluated the stream of spikes resulting from its output. We observed the ρ parameter of the output and checked that at time position 125.1 ms it reaches its minimum point, after which the parameters have the opposite descending tendency, marking the break-up point at this position. From this result, we can conclude that the latency measured for the system is approximately 5.1 ms.

The value for the latency obtained from our system is 51 ms. This result can be compared to the measurement from running the OpenCV script on a conventional computer (i7-8550U processor with 8GB of RAM with Python 3.8 and *OpenCV 4.5.5*[33]). The average time taken by the conventional method is 38 ms, a slightly lower value than the results taken by the spiking neural network.

CHAPTER 5
Discussion

The previous examined results obtained different metrics to evaluate the algorithm's performance. This metrics analysis plays a crucial role in developing the Spiking Neural Network model, as it constitutes a new research line that has no precedence in the literature.

This project is centered on evaluating the efficiency and suitability of Neuromorphic Hardware applied for the task of road lane detection. For this reason, the results play a key role in assessing whether or not the newly developed model can be compared to previous algorithms dedicated to the same task. One of the key factors of this project resides in its novelty. Up to the date, there have not been any other implementations of a neuromorphic system for detecting lines using both Dynamic Vision sensors and Spiking Neural Networks for performing the computational task. For this reason, it is important to notice that the results obtained here are not trying to prove better accuracy and performance than the conventional models. However, we aim our work to demonstrate the capabilities and confirm the suitability of this new computing platform for performing complex tasks such as straight-line recognition.

For this reason, we can consider that the main objective of the work has been accomplished, as we successfully proved that a system implemented using Neuromorphic Hardware is capable of obtaining an accuracy comparable to the results of previous algorithms such as *OpenCV*. This fact has been observed in the three analyzed datasets. On the first dataset studied, we got an average error lower than 0.28% when comparing the angle of the detected line and 0.5 % of the parameter space when comparing the differences between the position of the line.

Looking at the two straight lines dataset, we can also conclude that the error difference rate was comparable to the single straight line. However, it is worth noticing that the error was slightly higher than in the previous case. This error is visible in the analysis of the right line of the image, where the worst results were obtained. This result can be motivated because of the characteristics of the line in the recording, where we can see how the line is more faint and distorted by the camera, preventing it from being shown as a completely straight figure. These characteristics represent a greater difficulty for the algorithm to track the parameters of the line, which increases the error.

5.1 Driving Dataset

Finally, we analyzed the model results in a practical use case. We inputted the events recorded from a drive performed by a car in a real environment and studied the obtained results. In this case, we could observe how a lower accuracy was obtained. Nevertheless, the model could still track the lines on the road and output a valid result.

One of the reasons that we motivate for the deterioration of the results with the MVSEC dataset analysis is the fact that the *OpenCV* algorithm used as a comparison was not able to reliably track the lines on the image. This can be based on different factors, such as the difficulty for the algorithm to obtain enough information due to the low light environment in which the recording is done. This further motivates the advantages of the spiking neural network; as we can see in the visual comparison between the algorithms in 4.4, the Spiking Neural network was able to track and follow the lines of the road more accurately and with more stability than the compared algorithm.

The errors found in the OpenCV dataset also motivate the outlying error spikes found when plotting the differences between both algorithms. In this case, the result given by the spiking neural network would have been correct. Still, as the OpenCV implementation showed less reliable tracking, the comparison resulted in a significant error between both algorithms, which does not correlate with our output being far from the optimal values.

For this reason and to avoid these outlying results, other measurements such as manual labeling of the input data could be used to establish a baseline for comparing different outputs. In this way, we could avoid adding up the error by the OpenCV algorithm to our results. However, in the averaged value of the comparison, we still can consider OpenCV as a valid ground truth that also represents a case where we can show how the output of the Neuromorphic Model is superior in some cases to the accuracy of the conventional methods.

5.2 Latency and Power

Additionally, it is interesting to comment on the results obtained from the evaluation of the consumption of the neuromorphic system. We have obtained a power consumption ten times lower than other state-of-the-art systems available in the industry. However, it is important to notice these compared systems lack the power scalability of Neuromorphic Computers, so the amount of power will account for other algorithms running on top of the line detection task. In any case, this result proves the suitability of this new platform for its application on autonomous cars, where the power consumption is relevant to developing more efficient vehicles, also allowing these systems to need less cooling, which is a critical constraint on embedded systems.

We can also comment on the latency obtained by our implemented algorithm, which resulted in a time delay of 51ms, being slightly longer than the result obtained by the conventional camera-based algorithm of the *OpenCV* library of 38ms. We can consider this result brings both methods to comparable time proportions, backing up the low latency claim expected by using neuromorphic hardware. We can motivate this result by noting some of the constraints presented in more detail in the next paragraph. Additionally, it is also remarkable to notice that the measurements for the latency, to avoid adding too much complexity, are based on an indirect computation that results in an estimated value with a higher margin of error. Better metrics could be developed by creating a set of synthetic events representing a line that appears at specific time intervals and comparing the resulting lines' time delays, but that implementation falls outside of the scope of this report.

Regarding the challenges for our algorithm that could affect the latency measured, we remark how data inputting into the spiking neural network board has proven to be a challenge. This issue limited the speed at which we could send the stream of events, becoming a bottleneck that affected the latency of the results. One possibility to obtain even lower latency would be to change the input method to a different interface, such as the FPGA (Field Programmable Gate Arrays) in development by the research group at

the *Neuro Computing Systems* (NCS) lab in KTH. This new communication hardware will allow the system to receive more significant amounts of data per unit of time and thus reduce the latency of the results significantly.

5.3 The novelty of the Neuromorphic Systems

One of the main points of this project is to analyze the current state-of-the-art technology and apply it to a new use case that has not been explored before. This fact comes with the inherent challenge that resides in the state of development of the Neuromorphic platform. The nascent state of many of the required components for neuromorphic computation became an additional challenge for the development of the algorithm. KTH university has a research group focused on addressing this issue and providing more tools for future developments on this very same platform. Unfortunately, this development is still underway, and we could not use it for this project. One of the main challenges that we faced during the development of this algorithm was motivated by the communication of the input data with the Spiking Neuronal Boards. The existing protocols, topping off at 100MB/s of Ethernet connection, became a bottleneck for the data connections when treating millions of events. This project becomes one example where this limit is not sufficient. The events resulting from the recordings performed by the event-based cameras in our algorithm consist of inputs of more than a million events per every 10 seconds of recording. These streams of events need to be communicated in real-time to the development board and afterward outputted with the resulting spikes representing the lines. Our experimental evaluation with the available communication interfaces showed losses in the resulting spikes and buffer overflows that prevented the computation in some of the cases.

For this reason, several workarounds were used that resulted in reduced efficiency and increased the latency of the obtained results, not because of the hardware's inability to process the data but because of the limitations of the input and output protocols currently available for the SpiNNaker platform.

A future improvement that could effectively improve the model's performance would be to use more capable sockets that can manage the required amounts of data without affecting the algorithm's performance or causing data losses. This new socket implementation would reduce the delay shown between the frame recording and the predicted line and allow more data to be inputted into the machine to increase the accuracy of the results.

CHAPTER 6
Conclusions

6.1 Archieved results

The usage of neuromorphic hardware is now experiencing an increase in the number of projects devoted to finding new applications where this system can replace and improve previous implementations using conventional hardware. And this fact is motivated by the need for better models that at the same time are portable and don't require a considerable amount of power. For this reason, the use of Spiking Neural Networks and hardware such as the SpiNNaker board prove that it is possible to create capable systems that fulfill these conditions. This is the reason why these Neuronal Networks are called Third Generation Neuronal Networks.

After developing the algorithm, we evaluated its performance by analyzing different datasets consisting of a representation of a controlled environment for calibration of the system. The errors obtained from the analysis of the initial datasets show a promising rate of error of 0.28 % of the detected line's angle parameter space and 0.5 % for the distance of the detected line to the origin when compared to the *OpenCV* detection algorithm. This analysis proves that this system can fulfill the objective of detecting the parameters that define a straight line in a plane.

Once we fulfilled this task, we moved into applying a new dataset belonging to a real scenario. The results, in this case, showed a higher error than the previous analysis. The errors for the driving dataset obtained by our model against conventional methods are 4% for the angle and 4.5% for the distance of the detected line. As stated before, these results are affected by the poor performance of the *OpenCV* algorithm used as a comparison. Observing the results visually, we notice how our model was able to track the lanes more reliably when compared with the conventional method. This result concludes the adequacy of the algorithm for its application to the recognition of lane markings.

With this study, we prove a new use application and model for which the use of neuromorphic hardware can be beneficial and represent a valid alternative when compared to previous implementations of similar algorithms. This goal is accomplished by the neuromorphic system while also using significantly lower resources, both in power consumption and computing time (lower latency), as shown in our analysis.

6.2 Further Research

Using this motivation as a starting point, we worked on developing an algorithm capable of tracking lanes present on the road and obtaining a valid numerical output that can be further used in subsequent systems. This system could be further applied to an autonomous

car system, where the whole data processing could be done by using neuromorphic hardware, then benefiting from its main advantages such as the low latency of the results and the minimal power consumption when compared to conventional computing systems.

Further work could be done on the system to reduce the number of events produced by the output layer so that the algorithm only returns those points that confidently represent a straight line. This work can be done by implementing a second neuronal layer to allow the model's output to be further processed and avoid false positives. The implementation could be based on considering the previous outputted spikes to encourage the detection of lines that follow a sequential path on the output. This improvement could allow a more accurate result which would be more resilient to noise in the image and false positives generated by objects that resemble the markings on the road.

Additionally, other techniques for processing the data, such as inhibitory neurons, could further improve the accuracy of the results. This method, developed by Li et al.[4] is based on deactivating the neurons that surround a recently fired one. In this way, if a line has a width of multiple pixels, we can ensure that the result will only represent this line on a single output point instead of obtaining very close points in the output parameters. The implementation of this technique is based on deactivating those neurons surrounding one that has recently spiked. As shown in the cited study, the output of the system was significantly reduced in size while maintaining accuracy by using this technique. However, current platform constraints for the Spiking Neural network disregard the direct manipulation of the neurons, so a different strategy would have to be followed.

To sum up, we have successfully developed and evaluated a new working use case for the incipient neuromorphic hardware platform. This feat pushes further the boundaries of the capabilities of this hardware while inspiring new paths of research to be taken in future studies.

Bibliography

[1] Andrew Rowley. *RunningPyNNSimulationsonSpiNNaker LabManual*. Sept. 2017. URL: https://spinnakermanchester.github.io/spynnaker/4.0.0/RunningPyNNSimulationsonSpiNNakerLabManual.pdf (visited on 05/29/2022).

[2] Jesse Levinson et al. "Towards fully autonomous driving: Systems and algorithms". In: *2011 IEEE Intelligent Vehicles Symposium (IV)*. 2011 IEEE Intelligent Vehicles Symposium (IV). Baden-Baden, Germany: IEEE, June 2011, pp. 163–168. ISBN: 978-1-4577-0890-9. DOI: 10.1109/IVS.2011.5940562. URL: http://ieeexplore.ieee.org/document/5940562/ (visited on 04/04/2022).

[3] Guang Chen et al. "NeuroIV: Neuromorphic Vision Meets Intelligent Vehicle Towards Safe Driving With a New Database and Baseline Evaluations". In: *IEEE Transactions on Intelligent Transportation Systems* 23.2 (Feb. 2022), pp. 1171–1183. ISSN: 1524-9050, 1558-0016. DOI: 10.1109/TITS.2020.3022921. URL: https://ieeexplore.ieee.org/document/9234108/ (visited on 02/15/2022).

[4] Xue Li et al. "Lane detection based on spiking neural network and hough transform". In: *2015 8th International Congress on Image and Signal Processing (CISP)*. 2015 8th International Congress on Image and Signal Processing (CISP). Shenyang, China: IEEE, Oct. 2015, pp. 626–630. ISBN: 978-1-4673-9098-9. DOI: 10.1109/CISP.2015.7407954. URL: http://ieeexplore.ieee.org/document/7407954/ (visited on 03/10/2022).

[5] Yuhuang Hu et al. "DDD20 End-to-End Event Camera Driving Dataset: Fusing Frames and Events with Deep Learning for Improved Steering Prediction". In: *arXiv:2005.08605 [cs]* (May 18, 2020). arXiv: 2005.08605. URL: http://arxiv.org/abs/2005.08605 (visited on 03/01/2022).

[6] A. Geiger, P. Lenz, and R. Urtasun. "Are we ready for autonomous driving? The KITTI vision benchmark suite". In: *2012 IEEE Conference on Computer Vision and Pattern Recognition*. 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). Providence, RI: IEEE, June 2012, pp. 3354–3361. ISBN: 978-1-4673-1228-8 978-1-4673-1226-4 978-1-4673-1227-1. DOI: 10.1109/CVPR.2012.6248074. URL: http://ieeexplore.ieee.org/document/6248074/ (visited on 04/04/2022).

[7] C. Thorpe et al. "Vision and navigation for the Carnegie-Mellon Navlab". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 10.3 (May 1988), pp. 362–373. ISSN: 01628828. DOI: 10.1109/34.3900. URL: http://ieeexplore.ieee.org/document/3900/ (visited on 03/12/2022).

[8] Keshav Bimbraw. "Autonomous Cars: Past, Present and Future - A Review of the Developments in the Last Century, the Present Scenario and the Expected Future of Autonomous Vehicle Technology:" in: *Proceedings of the 12th International Conference on Informatics in Control, Automation and Robotics*. 12th International Conference on Informatics in Control, Automation and Robotics. Colmar, Alsace, France: SCITEPRESS - Science, 2015, pp. 191–198. ISBN: 978-989-758-122-9 978-989-758-123-6. DOI: 10.5220/0005540501910198. URL: http://www.scitepress.

org/DigitalLibrary/Link.aspx?doi=10.5220/0005540501910198 (visited on 04/04/2022).

[9] Wolfgang Maass. "Networks of spiking neurons: The third generation of neural network models". In: *Neural Networks* 10.9 (1997), pp. 1659–1671. ISSN: 0893-6080. DOI: https://doi.org/10.1016/S0893-6080(97)00011-7. URL: https://www.sciencedirect.com/science/article/pii/S0893608097000117.

[10] Steve B. Furber et al. "The SpiNNaker Project". In: *Proceedings of the IEEE* 102.5 (May 2014), pp. 652–665. ISSN: 0018-9219, 1558-2256. DOI: 10.1109/JPROC.2014.2304638. URL: https://ieeexplore.ieee.org/document/6750072/ (visited on 03/28/2022).

[11] QingXiang Wu et al. "Detection of Straight Lines Using a Spiking Neural Network Model". In: *2009 Fifth International Conference on Natural Computation*. 2009 Fifth International Conference on Natural Computation. Tianjian, China: IEEE, 2009, pp. 385–389. ISBN: 978-0-7695-3736-8. DOI: 10.1109/ICNC.2009.484. URL: http://ieeexplore.ieee.org/document/5363996/ (visited on 02/21/2022).

[12] Lukas Everding and Jörg Conradt. "Low-Latency Line Tracking Using Event-Based Dynamic Vision Sensors". In: *Frontiers in Neurorobotics* 12 (2018). ISSN: 1662-5218. DOI: 10.3389/fnbot.2018.00004. URL: https://www.frontiersin.org/article/10.3389/fnbot.2018.00004.

[13] Henri Rebecq et al. "High Speed and High Dynamic Range Video with an Event Camera". In: *arXiv:1906.07165 [cs]* (June 15, 2019). arXiv: 1906.07165. URL: http://arxiv.org/abs/1906.07165 (visited on 02/15/2022).

[14] NHTSA. "Passenger Vehicle Occupant Fatalities by Day and Night—A Contrast". In: *Annals of Emergency Medicine* 51.4 (Apr. 2008), p. 443. ISSN: 01960644. DOI: 10.1016/j.annemergmed.2008.02.004. URL: https://linkinghub.elsevier.com/retrieve/pii/S0196064408004836 (visited on 04/04/2022).

[15] National Highway Traffic Safety Administration's. *Traffic Safety Facts 2019: A Compilation of Motor Vehicle Crash Data*. NHTSA, 2019, p87–88. URL: https://crashstats.nhtsa.dot.gov/Api/Public/ViewPublication/813141 (visited on 04/01/2022).

[16] Prasanna Date et al. "Neuromorphic Computing is Turing-Complete". In: *arXiv:2104.13983 [cs]* (Apr. 28, 2021). arXiv: 2104.13983. URL: http://arxiv.org/abs/2104.13983 (visited on 02/16/2022).

[17] The University of Manchester. *SpiNNaker Project Information Website*. SpiNNaker Project Information Website. May 30, 2022. URL: https://apt.cs.manchester.ac.uk/projects/SpiNNaker/ (visited on 05/30/2022).

[18] EPSRC. *EPSRC Project Website*. EPSRC Project Website. May 30, 2022. URL: https://www.ukri.org/ (visited on 05/30/2022).

[19] Xavier Lagorce et al. "Breaking the millisecond barrier on SpiNNaker: implementing asynchronous event-based plastic models with microsecond resolution". In: *Frontiers in Neuroscience* 9 (June 8, 2015). ISSN: 1662-453X. DOI: 10.3389/fnins.2015.00206. URL: http://journal.frontiersin.org/Article/10.3389/fnins.2015.00206/abstract (visited on 05/30/2022).

[20] Andrew G. D. Rowley et al. "SpiNNTools: The Execution Engine for the SpiNNaker Platform". In: *Frontiers in Neuroscience* 13 (Mar. 26, 2019), p. 231. ISSN: 1662-453X. DOI: 10.3389/fnins.2019.00231. URL: https://www.frontiersin.org/article/10.3389/fnins.2019.00231/full (visited on 05/30/2022).

[21] Oliver Rhodes et al. "sPyNNaker: A Software Package for Running PyNN Simulations on SpiNNaker". In: *Frontiers in Neuroscience* 12 (Nov. 20, 2018), p. 816. ISSN: 1662-453X. DOI: 10.3389/fnins.2018.00816. URL: https://www.frontiersin.org/article/10.3389/fnins.2018.00816/full (visited on 02/21/2022).

[22] Andrew P Davison. "PyNN: a common interface for neuronal network simulators". In: *Frontiers in Neuroinformatics* 2 (2008). ISSN: 16625196. DOI: 10.3389/neuro.11.011.2008. URL: http://journal.frontiersin.org/article/10.3389/neuro.11.011.2008/abstract (visited on 05/30/2022).

[23] Sangya Dutta et al. "Leaky Integrate and Fire Neuron by Charge-Discharge Dynamics in Floating-Body MOSFET". In: *Scientific Reports* 7.1 (Dec. 2017), p. 8257. ISSN: 2045-2322. DOI: 10.1038/s41598-017-07418-y. URL: http://www.nature.com/articles/s41598-017-07418-y (visited on 05/30/2022).

[24] Francisco Barranco et al. *Contour Detection and Characterization for Asynchronous Event Sensors*. URL: http://www.umiacs.umd.edu/research/POETICON/DVSContours/ (visited on 05/29/2022).

[25] Francisco Barranco et al. "Contour Detection and Characterization for Asynchronous Event Sensors". In: *2015 IEEE International Conference on Computer Vision (ICCV)*. 2015 IEEE International Conference on Computer Vision (ICCV). Santiago, Chile: IEEE, Dec. 2015, pp. 486–494. ISBN: 978-1-4673-8391-2. DOI: 10.1109/ICCV.2015.63. URL: http://ieeexplore.ieee.org/document/7410420/ (visited on 05/29/2022).

[26] Wikipedia contributors. *AI winter — Wikipedia, The Free Encyclopedia*. 2022. URL: https://en.wikipedia.org/w/index.php?title=AI_winter&oldid=1074228388.

[27] Olga Russakovsky et al. "ImageNet Large Scale Visual Recognition Challenge". In: *International Journal of Computer Vision* 115.3 (Dec. 2015), pp. 211–252. ISSN: 0920-5691, 1573-1405. DOI: 10.1007/s11263-015-0816-y. URL: http://link.springer.com/10.1007/s11263-015-0816-y (visited on 06/05/2022).

[28] David Reverter Valeiras et al. "Event-Based Line Fitting and Segment Detection Using a Neuromorphic Visual Sensor". In: *IEEE Transactions on Neural Networks and Learning Systems* 30.4 (Apr. 2019), pp. 1218–1230. ISSN: 2162-237X, 2162-2388. DOI: 10.1109/TNNLS.2018.2807983. URL: https://ieeexplore.ieee.org/document/8463622/ (visited on 02/15/2022).

[29] Basabdatta Sen-Bhattacharya et al. "Building a Spiking Neural Network Model of the Basal Ganglia on SpiNNaker". In: *IEEE Transactions on Cognitive and Developmental Systems* 10.3 (Sept. 2018), pp. 823–836. ISSN: 2379-8920, 2379-8939. DOI: 10.1109/TCDS.2018.2797426. URL: https://ieeexplore.ieee.org/document/8309360/ (visited on 02/24/2022).

[30] Sajjad Seifozzakerini et al. "Event-Based Hough Transform in a Spiking Neural Network for Multiple Line Detection and Tracking Using a Dynamic Vision Sensor". In: *Procedings of the British Machine Vision Conference 2016*. British Machine Vision Conference 2016. York, UK: British Machine Vision Association, 2016, pp. 94.1–94.12. ISBN: 978-1-901725-59-9. DOI: 10.5244/C.30.94. URL: http://www.bmva.org/bmvc/2016/papers/paper094/index.html (visited on 03/09/2022).

[31] Alex Zihao Zhu et al. "The Multivehicle Stereo Event Camera Dataset: An Event Camera Dataset for 3D Perception". In: *IEEE Robotics and Automation Letters* 3.3 (July 2018), pp. 2032–2039. ISSN: 2377-3766, 2377-3774. DOI: 10.1109/LRA.2018.2800793. URL: http://ieeexplore.ieee.org/document/8288670/ (visited on 02/15/2022).

[32] Richard O. Duda and Peter E. Hart. "Use of the Hough transformation to detect lines and curves in pictures". In: *Communications of the ACM* 15.1 (Jan. 1972), pp. 11–15. ISSN: 0001-0782, 1557-7317. DOI: 10.1145/361237.361242. URL: https://dl.acm.org/doi/10.1145/361237.361242 (visited on 03/16/2022).

[33] G. Bradski. "The OpenCV Library 4.5.5". In: *Dr. Dobb's Journal of Software Tools* (2022). URL: https://opencv.org/.

[34] Eustace Painkras et al. "SpiNNaker: A 1-W 18-Core System-on-Chip for Massively-Parallel Neural Network Simulation". In: *IEEE Journal of Solid-State Circuits* 48.8 (Aug. 2013), pp. 1943–1953. ISSN: 0018-9200, 1558-173X. DOI: 10.1109/JSSC.2013.2259038. URL: http://ieeexplore.ieee.org/document/6515159/ (visited on 06/01/2022).

[35] Indar Sugiarto et al. "High performance computing on SpiNNaker neuromorphic platform: A case study for energy efficient image processing". In: *2016 IEEE 35th International Performance Computing and Communications Conference (IPCCC)*. 2016 IEEE 35th International Performance Computing and Communications Conference (IPCCC). Las Vegas, NV, USA: IEEE, Dec. 2016, pp. 1–8. ISBN: 978-1-5090-5252-3. DOI: 10.1109/PCCC.2016.7820645. URL: http://ieeexplore.ieee.org/document/7820645/ (visited on 06/06/2022).

[36] Inivation Inc. *DAVIS 346 Specification Sheet*. Aug. 2019. URL: https://inivation.com/wp-content/uploads/2019/08/DAVIS346.pdf (visited on 06/01/2022).

[37] Inc. Tesla. *Tesla Autonomy Day (1:31:00)*. Tesla Autonomy Day. Apr. 22, 2019. URL: https://youtu.be/Ucp0TTmvqOE?t=5501.

APPENDIX A
LIF Neuron Parameters

Parameter	Value
cm	1
i_offset	-1
tau_m	20
tau refraction	0
tau excitatory synapse	1
tau inhibitory synapse	1
reset voltage	-55
rest voltage	-65
threshold voltage	-10

Table A.1: Parameters used for the obtention of the results in the neuron output layer of the SpiNNaker Board

APPENDIX B
Results in Video

Video Ouput results from Straight Line Dataset Available in: https://youtu.be/5tJUv-gndrE .

The annexed video shows a side-by-side comparison of the outputs obtained after analyzing the events recorded of a straight line moving on a white paper sheet. The sensor's black and white image frames are used as background to compare the line accuracy. Superimposed on the frames, we plot the red lines representing the detected lanes by each algorithm, on the left using the *OpenCV* implementation, and on the right the Neuromorphic system's output.

Video Ouput results from Two Angled Lines Dataset Available in: https://youtu.be/2E4TlXuS8qo .

Video Ouput results from MVSEC Dataset Available in: https://youtu.be/YzJfCZah2bk .

The annexed video shows a side-by-side comparison of the outputs obtained after analyzing the first 20 Million events on the MVSEC dataset, corresponding to the " *outdoor night data* " recording. Superimposed on the frames, we can find the red lines denoting the detected lanes by each algorithm, using clustering to obtain the most relevant lines corresponding to the lanes.