Proyecto Fin de Máster
Máster en Ingeniería Electrónica, Robótica y Automática (MIERA)

Estudio de mecanismos de foveación para un sensor de visión neuromórfico

Autor: Isabel Ortiz Ramírez

Tutor: Sergio Luis Toral Marin

Tutor Externo: Luis Alejandro Camuñas Mesa

Dpto. Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2022

Proyecto Fin de Máster
Máster en Ingeniería Electrónica, Robótica y Automática (MIERA)

Estudio de mecanismos de foveación para un sensor de visión neuromórfico

Autor:

Isabel Ortiz Ramírez

Tutor:

Sergio Luis Toral Marin

Catedrático de Universidad

Dpto. de Ingeniería Electrónica

Escuela Técnica Superior de Ingeniería

Universidad de Sevilla

Sevilla, 2022

Proyecto Fin de Máster: Estudio de mecanismos de foveación para un sensor de visión neuromórfico

Autor: Isabel Ortiz Ramírez
Tutor: Sergio Luis Toral Marin

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2022

El Secretario del Tribunal

A mi familia
A los profesores que me han ayudado

Agradecimientos

Quiero agradecer a mi tutor de la beca, Luis Camuñas, por ayudarme con este trabajo. También quiero agradecer al resto de compañeros del Instituto de Microelectrónica que me han proporcionado ayuda y material. Por otra parte, agradecer a Sergio Toral por ser mi tutor académico y ayudarme con la redacción del documento.

Doy gracias por haber tenido la oportunidad de disfrutar de la beca JAE INTRO del CSIC. Además, tengo que agradecer a todos los profesores que me han ayudado durante mi realización del master.

Finalmente, agradezco a mi familia y especialmente a mi padre y a mi madre por contribuir a mi formación académica e intentar ayudarme cuando lo necesito.

Resumen

Este trabajo se realizó en el Grupo de Sistemas Neuromórficos del Instituto de Microelectrónica de Sevilla. En el cual se trabaja en la implementación en hardware de sistemas de procesamiento de visión bio-inspirados.

A pesar del impresionante desarrollo de los sistemas de computación durante las últimas décadas, el cerebro humano sigue demostrando una capacidad inalcanzable de procesar información sensorial a muy alta velocidad, con muy bajo consumo de potencia y minimizándose cantidad de datos necesarios. Esto se debe a la eficiencia del procesamiento neuronal, masivamente paralelo. En el grupo se diseñan sensores de visión basados en eventos que emulan la retina humana.

No se trabaja con fotogramas, sino con flujos continuos de impulsos eléctricos (que llamamos eventos o 'spikes') producidos de forma asíncrona por cada foto-sensor (o píxel) de la retina de forma autónoma cuando detecta un cambio suficiente de luz. Cada píxel almacena un nivel de brillo de referencia y lo compara continuamente con el nivel de brillo actual. Si la diferencia de brillo supera un umbral, ese píxel restablece su nivel de referencia y genera un evento, en el que se codifica el tiempo, la ubicación y el signo del cambio de brillo. Las cámaras de eventos permiten capturar el movimiento con mucha más resolución temporal y menos consumo de energía. Además de aumentar el rango dinámico y disminuir el desenfoque de movimiento.

La visión humana usa un mecanismo de foveación para maximizar la resolución espacial en la zona donde se enfoca la vista mientras que mantiene una baja resolución en las zonas de visión periférica. Así se reduce la cantidad de información generada por la retina manteniendo la capacidad de reconocimiento visual.

Este proyecto se enmarca en el estudio de la aplicación de un mecanismo de foveación sobre los sistemas de visión artificial basada en eventos. En concreto, se ha modelado a nivel de software distintas estrategias para controlar de forma dinámica la resolución de un sensor de visión, usando la máxima resolución posible en una o varias regiones de interés (identificadas a partir de distintos métodos de detección y seguimiento de objetos) y reduciendo la resolución en la periferia, con el fin de estudiar su comportamiento; hacerse una idea de la resolución que se debe aplicar y de otros parámetros cuando se implemente en hardware.

Se parte de una grabación de eventos, se convierte a fotogramas, luego se interpolan las intensidades de esos fotogramas para aumentar la resolución temporal, se seleccionan las zonas de interés, se le aplica la foveación combinando pixeles en la periferia de los objetos, y finalmente se generan eventos con las imágenes modificadas.

Abstract

This work was carried out in the Neuromorphic Systems Group of the Microelectronics Institute of Seville. It is working on the hardware implementation of bio-inspired vision processing systems.

Despite the impressive development of computing systems over the last decades, the human brain still demonstrates an unattainable ability to process sensory information at very high speed, with very low power consumption and minimising the amount of data required. This is due to the efficiency of massively parallel neural processing. The group designs event-driven vision sensors that emulate the human retina.

We do not work with frames, but with continuous streams of electrical impulses (which we call events or 'spikes') produced asynchronously by each photo-sensor (or pixel) in the retina autonomously when it detects a sufficient change in light. Each pixel stores a reference brightness level and continuously compares it to the current brightness level. If the difference in brightness exceeds a threshold, that pixel resets its reference level and generates an event, encoding the time, location and sign of the brightness change. Event cameras allow motion to be captured with much higher temporal resolution and lower power consumption. They also increase dynamic range and decrease motion blur.

Human vision uses a foveation mechanism to maximise spatial resolution in the area where the view is focused while maintaining low resolution in the peripheral vision areas. This reduces the amount of information generated by the retina while maintaining visual recognition capability.

This project is part of the study of the application of a foveation mechanism on event-based artificial vision systems. Specifically, different strategies have been modelled at software level to dynamically control the resolution of a vision sensor, using the maximum possible resolution in one or several regions of interest (identified from different object detection and tracking methods) and reducing the resolution in the periphery, in order to study its behaviour; to get an idea of the resolution to be applied and other parameters when implemented in hardware.

We start from a recording of events, convert it to frames, then interpolate the intensities of those frames to increase the temporal resolution, select the areas of interest, apply foveation by combining pixels at the periphery of the objects, and finally generate events with the modified images.

Índice corto

Índice

Notación

V_m	Potencial de la membrana
C_m	Capacitancia de la membrana
I_{mA}	Corriente eléctrica de analogía eléctrica de la membrana en SNN
R_m	Resistencia eléctrica de membrana
I	Valor o valores de intensidad de la imagen
Iph	Corriente eléctrica producida en píxel de cámara de eventos
Io_indice	Matriz que guarda intensidad de cada píxel; la intensidad de la última vez que se produzco evento en ese píxel.
θ_{ev}	Mínimo contraste temporal detectable
umbral	Valor con el que se compara la diferencia de intensidades normalizada
ln	Logaritmo neperiano
Δ, Delta	Cambio o diferencia
i	Coordenada fila del píxel
j	Coordenada columna del píxel
GMedia	Matriz con intensidades después de foveación
C	Sensibilidad al contraste (umbral)
w	Función que define el kernel de convolución en la SNN

1 INTRODUCCIÓN

"Sin duda, no hay progreso."

Charles Robert Darwin

Este proyecto se centra en la visión neuromórfica. Se realizó este documento basándose en lo realizado durante una beca de introducción a la investigación ofertada por el CSIC. Se solicitó concretamente esta beca porque se consideró interesante saber sobre sistemas que imitan el cerebro humano.

El tutor de la beca fue Luis Camuñas, que forma parte del Grupo de Sistemas Neuromórficos del Instituto de Microelectrónica de Sevilla, donde se trabaja en la implementación en hardware de sistemas de procesamiento de visión bio-inspirados. En el grupo se diseñan sensores de visión basados en eventos que emulan la retina humana.

1.1 Motivación

A pesar del impresionante desarrollo de los sistemas de computación durante las últimas décadas, el cerebro humano sigue demostrando una capacidad inalcanzable de procesar información sensorial a muy alta velocidad, con muy bajo consumo de potencia y minimizándose cantidad de datos necesarios. Esto se debe a la eficiencia del modelo de procesamiento neuronal, basado en un procesamiento masivamente paralelo y distribuido mediante miles de millones de neuronas.

Por este motivo, se persigue la realización de retinas artificiales que intenten acercarse, lo máximo posible, al eficiente mecanismo biológico, pudiendo ser de gran utilidad en diferentes ámbitos relacionados con la ingeniería robótica (drones) y la ingeniería biomédica (prótesis de ojo). En la Figura 1 se muestra un esquema del funcionamiento del ojo, sirve para entender mejor el proceso que se va a imitar.

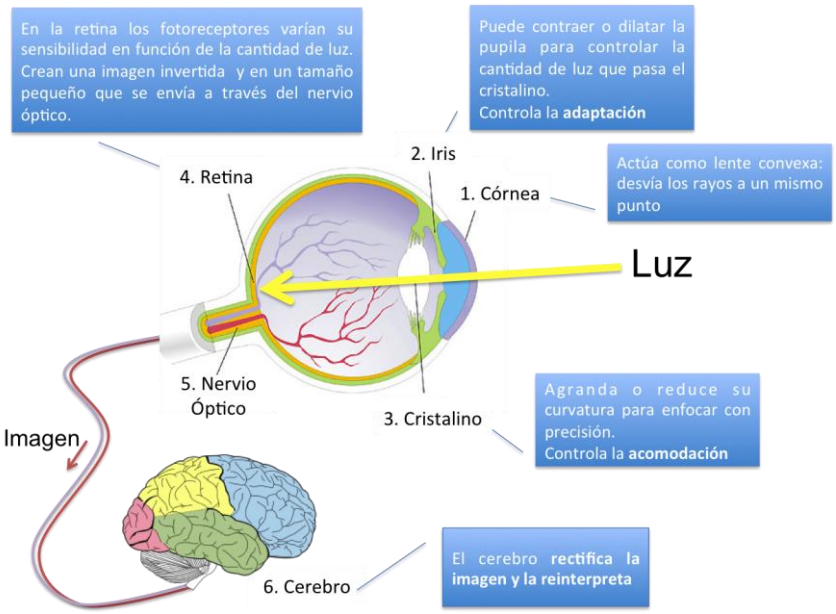


Figura 1. Funcionamiento del ojo. Fuente: [1].

1.2 Objetivos

El objetivo principal del trabajo es realizar grabaciones de visión por eventos que incluyan un mecanismo de foveación, es decir, que permitan definir zonas de interés donde se represente la imagen con alta resolución espacial, mientras el resto del campo visual se represente a baja resolución, reduciendo así la cantidad de información generada sin perder prestaciones. En otras palabras, se va a emular cómo una retina genera eventos usando el mecanismo de foveación. Se realizará un programa que devuelva como resultado lo que debería dar la retina; permitiendo hacerse una idea del resultado que se conseguiría.

1.3 Organización de la memoria

La memoria de este trabajo se estructura de la siguiente forma:

En el capítulo 1 se establece la motivación, objetivos y organización de la memoria.

En el capítulo 2 se resume lo necesario para entender este trabajo. Se introducen los conceptos de cámara de eventos, spiking neural network y la foveación.

El capítulo 3 se explican los pasos realizados en el proyecto, primero se hace un esquema general y luego en cada subcapítulo se explica uno de los pasos.

En el capítulo 4 se muestran los resultados experimentales de cada etapa del proceso.

En el capítulo 5 se exponen los diferentes códigos realizados o utilizados en Matlab, explicando de manera esquemática los programas principales.

En el capítulo 6 se expresan unas conclusiones sobre este trabajo y posibles líneas futuras a realizar. Algunas de ellas se empezaron a realizar, pero no se llegaron a terminar, por ello se han puesto en ese apartado: uso de tarjeta PYNQ-Z2 y seguimiento de objetos en el ámbito de los eventos.

1.4 Conclusiones

En este capítulo se ha hecho una breve introducción de porqué interesa conseguir sistemas que imiten los procedimientos del cerebro humano. También se ha planteado los objetivos principales del trabajo y se ha explicado cómo se estructura la memoria.

2 MARCO TEÓRICO

"El cerebro humano tiene 100 mil millones de neuronas, cada neurona conectada a otras 10 mil neuronas. Sentado sobre sus hombros está el objeto más complicado del universo."

Michio Kaku

Se presentarán los conceptos necesarios para comprender de lo que trata este proyecto. Se familiarizará con los conceptos principales de visión por eventos y de las Spiking Neural Network (SNN). Aunque no se llegaron a implementar las SNN en este proyecto, pertenecería a una de las líneas futuras, ya que están pensadas para poder trabajar con los eventos que produciría el sensor de visión. Las SNN es una variante de las redes neuronales bastante novedosa, "*presentada por los investigadores de la Universidad de Heidelberg y la Universidad de Berna y se desarrolló como una técnica rápida y energéticamente eficiente para la computación*" [2].

2.1. Cámara de eventos (sensor de visión por eventos)

Las cámaras de eventos, sensor de visión dinámica[1] (DVS) o cámaras neuromórficas, son sensores bioinspirados. Son distintas de las cámaras de fotogramas convencionales [3] [4] [5] [6]. Están disponibles comercialmente solo desde 2008 y existe un gran interés comercial en explotar estos nuevos sensores de visión [7] (se puede ver un ejemplo de cámara comercial en la Figura 8). En la Figura 2 se puede ver un diagrama de los elementos que conforman un píxel de un DVS. Este esquema simplifica lo que se muestra en la Figura 3; el circuito del píxel del DVS. En esa figura también se muestran algunas gráficas que explican el funcionamiento del circuito. En este documento no se entrará en muchos detalles sobre el funcionamiento del circuito.

[1] Sensor Visión Dinámica: sus píxeles emiten de forma asíncrona un potencial de acción (es decir, un spike, o evento) siempre que el cambio en la log-luminancia en esta ubicación desde el último evento, alcanza un umbral.

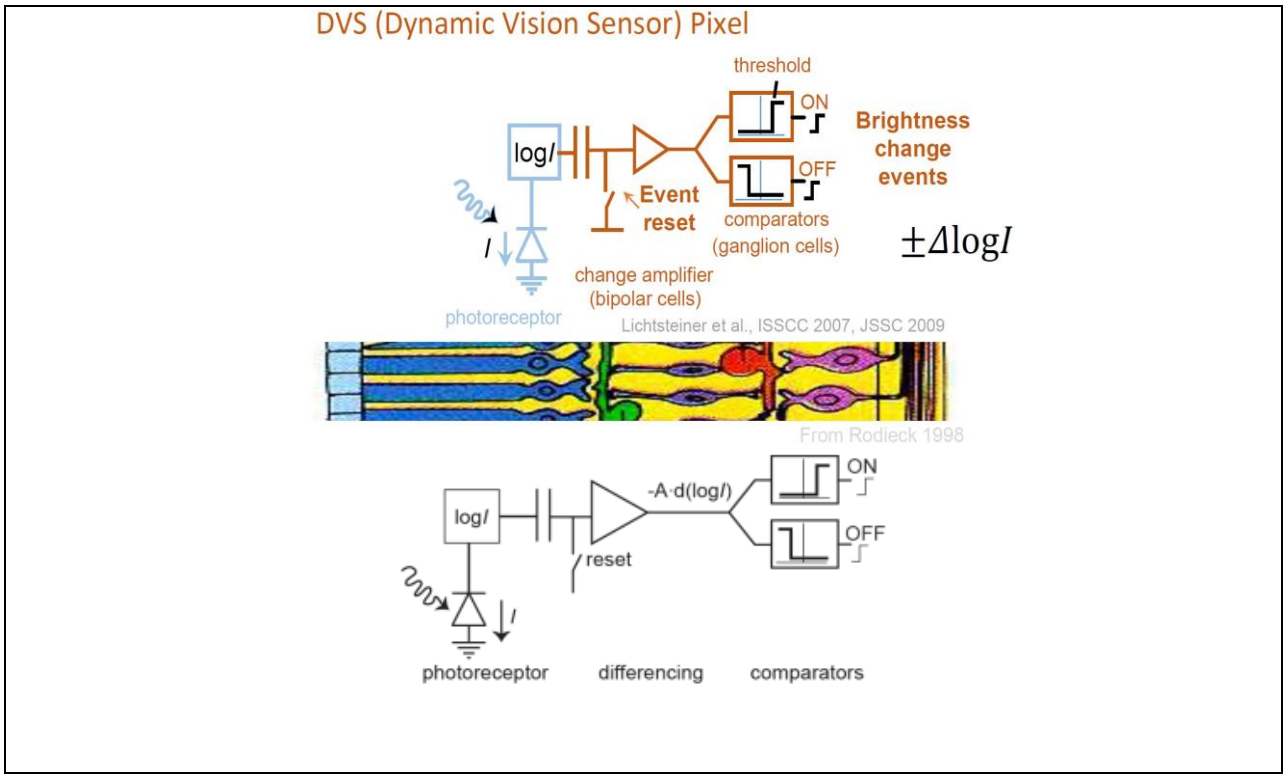


Figura 2. Diagrama DVS (Dynamic Vision Sensor) Píxel. Fuente: [8] [9].

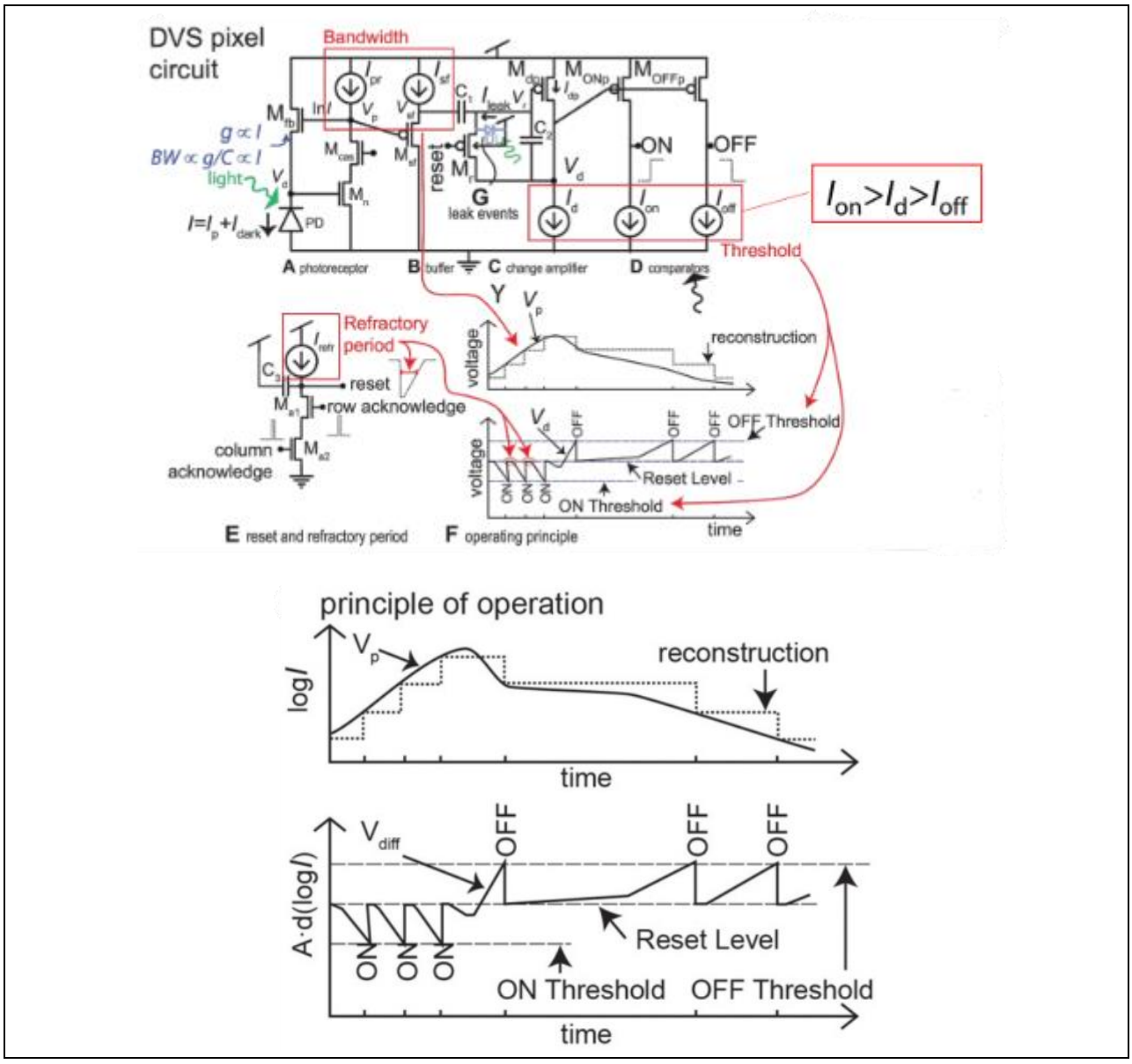


Figura 3. Circuito DVS píxel (arriba) y gráficas de principio de operación (abajo). Los ON y OFF son eventos. Fuente: [10].

Para entender por qué es bioinspirado, se muestra la Figura 4 y la Figura 5. En la Figura 4 se compara cada parte del píxel del sensor con lo que sería un 'píxel' de la retina humana. "*La primera capa es similar a las células cónicas de la retina para la conversión fotoeléctrica; la segunda capa, similar a las células bipolares de la retina, se utiliza para obtener los cambios en la intensidad de la luz; la tercera capa es similar a las células ganglionares de la retina para la salida del signo de cambio de intensidad de la luz*" [11]. En la Figura 5 se muestra dónde se sitúa la retina en el ojo humano, teniendo una gran multitud de 'píxeles'.

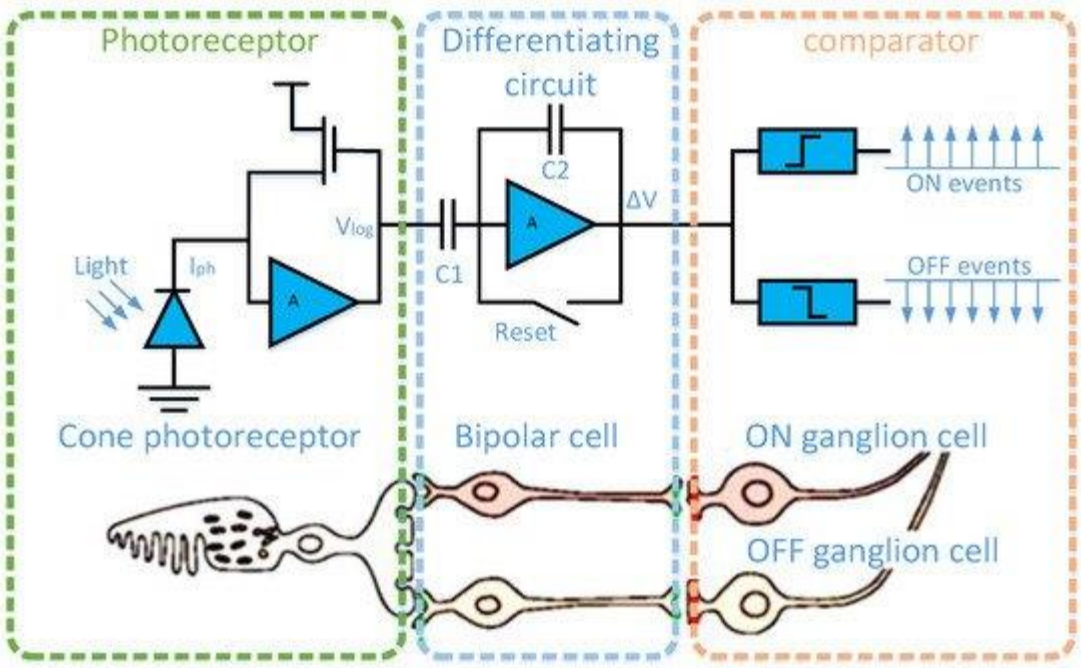


Figura 4. Diagrama DVS comparando con retina del ojo humano. Fuente: [11].

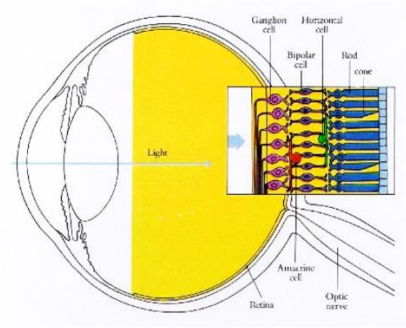


Figura 5. Ojo humano. La cámara de eventos imita al ojo humano. Fuente: [10].

También existe el DAVIS (sensor de visión dinámica y de píxeles activos (Figura 6)) [12], el cual posee un sensor de píxeles activos (APS) de obturación global, además del sensor del DVS, que comparte la misma matriz de fotosensores. En otras palabras, "*tiene la capacidad de producir cuadros de imagen junto con los eventos*" [3].

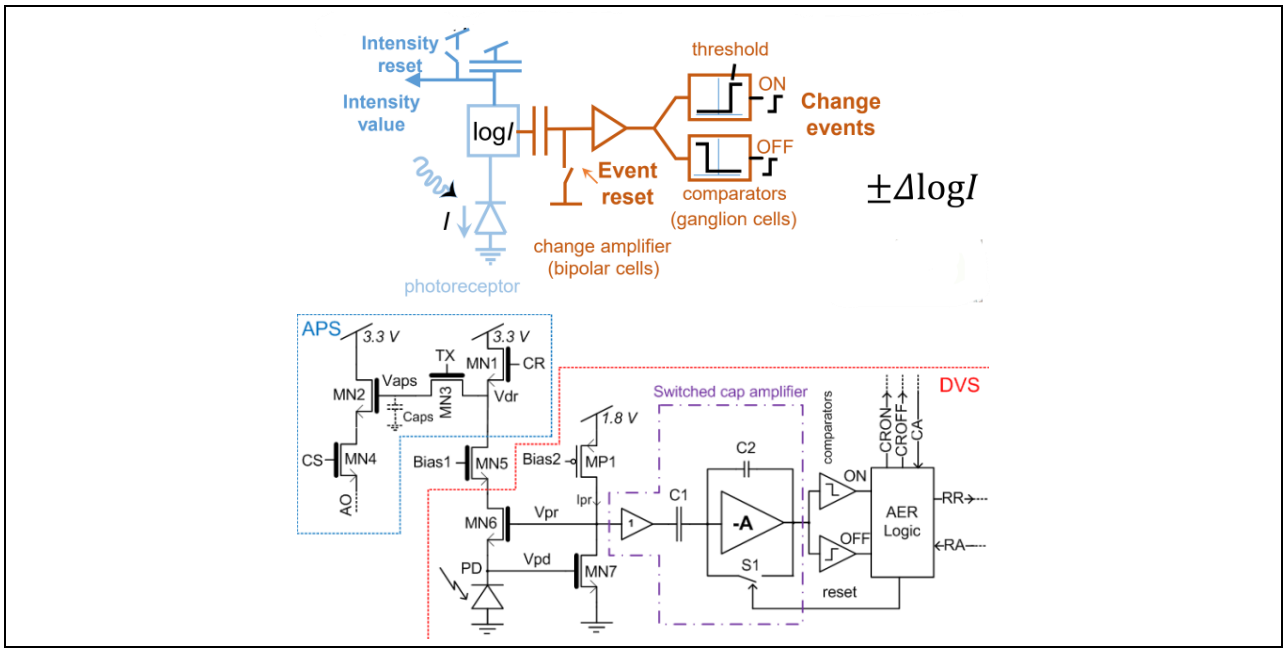


Figura 6. Esquema y circuito de DAVIS (Dynamic and Active Pixel Vision Sensor) Pixel. Fuente: [10] [9].

En lugar de capturar imágenes a una velocidad fija, miden de forma asíncrona los cambios de brillo por píxel y emiten un flujo de eventos que codifican el tiempo, la ubicación y el signo de los cambios de brillo, como se ilustra en la Figura 7. Como se ve en la Figura 3, cada fotosensor integra luz. El fotosensor compara umbrales y cuando detecta un cambio suficiente de la luz genera un evento. Los ON (positivo) y OFF (negativo) que se muestran son los eventos. Los ON son cambios de menos a más luminosidad y los OFF lo contrario.

Si se almacenan los eventos en una matriz, se incluye en la columna 1 una marca temporal (indica cuándo se generó ese evento), en la columna 2 la coordenada 'x' del píxel que generó ese evento, en la columna 3 la coordenada 'y', y en la columna 4 la polaridad de ese evento (indica si el evento representa un incremento de luminosidad o un decremento). El número de filas de la matriz es el número de eventos.

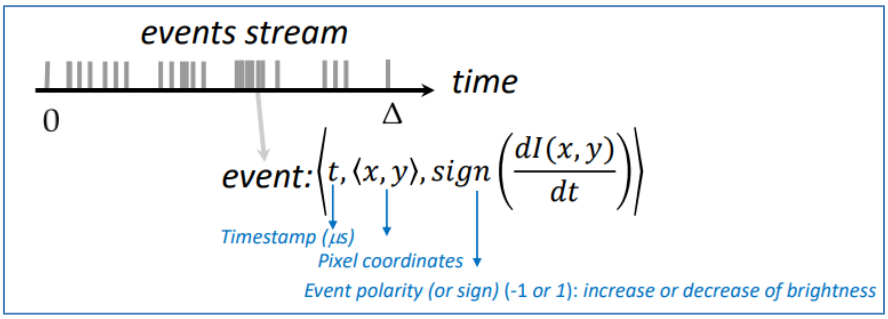


Figura 7. Los eventos ocurren de forma asíncrona y se codifican de una determinada manera. Fuente: [8].

Una cámara de eventos captura cambios de luminosidad, es decir, puede capturar sólo movimiento. Como se explica en la Figura 8, con la cámara de eventos no se pierde tanta información del movimiento como en una cámara convencional. Cuando no hay cambios de movimientos, es decir, cambio de luminosidad significativos; no se producen eventos. "*Los píxeles de la cámara de eventos responden de forma independiente a los cambios de brillo a medida que se producen. Cada píxel almacena un nivel de brillo de referencia y lo compara continuamente con el nivel de brillo actual. Si la diferencia de brillo supera un umbral, ese píxel restablece su nivel de referencia y genera un evento*" [3]. El nivel de referencia se va actualizando cada vez que se produce un evento, tomándose justamente el valor que ha producido ese evento. Para entender mejor este concepto véase en qué consiste el principio de operación, el cual se aprecia en la Figura 3. "*Cómo se representan internamente los eventos ON y OFF y se emiten en respuesta a una señal de entrada*" [13].

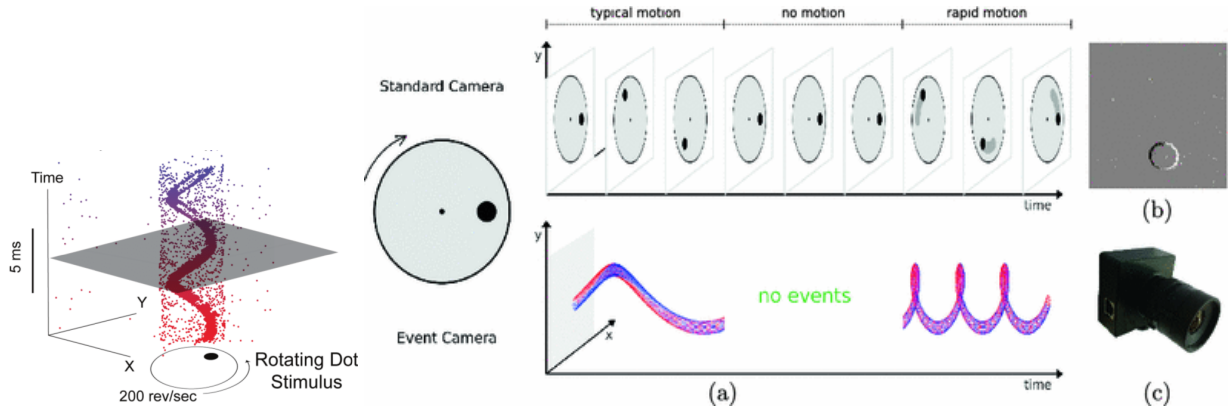


Figura 8. a) Cuando no hay cambio de luminosidad significativos; no se producen eventos. b) Ejemplo de representación gráfica de los cambios de luminosidad en determinados pixeles, mediante la acumulación de eventos en un intervalo de tiempo. Se está teniendo en cuenta la polaridad: los píxeles blancos y negros representan eventos positivos y negativos, respectivamente. c) La cámara comercial para eventos, DVS128, de iniLabs Ltd. Fuente: [14] [15].

Las cámaras de eventos ofrecen propiedades atractivas en comparación con las cámaras tradicionales: una alta resolución temporal (del orden de 1 µs), un rango dinámico[2] muy alto (140 dB frente a 60 dB), un bajo consumo de energía (media: 1mW frente a 1W) y un gran ancho de banda de píxeles (del orden de kHz) que permite reducir el desenfoque de movimiento (Figura 9) [15]. *"Al igual que con la visión humana, este concepto conduce a un tráfico de datos considerablemente menor y, por lo tanto, también a un menor volumen de transferencia y un tiempo de procesamiento más corto"* [16].

Figura 9. Problemas que resuelven las cámaras de eventos. Fuente: [8].

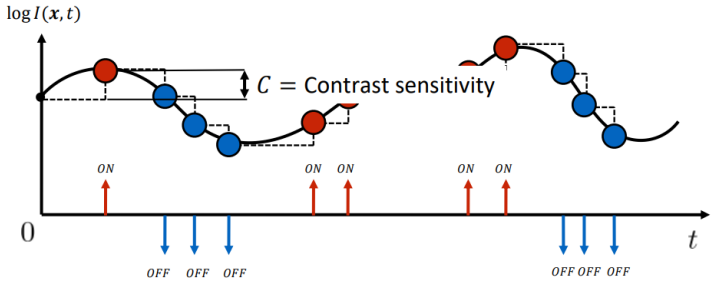


Figura 10. Ejemplo considerando la intensidad de un solo píxel. Los eventos se lanzan de forma asíncrona. Fuente: [8].

No obstante, se necesitan métodos novedosos para procesar la salida no convencional de estos sensores. Los algoritmos tradicionales no se pueden usar, porque la salida se compone de una secuencia de eventos asíncronos

[2] "*La cantidad de señales que es capaz captar, distinguir o representar*". Con respecto a las cámaras fotográficas, "*el rango dinámico mide el conjunto de tonos desde los más oscuros a los más claros que una cámara es capaz de tomar en una fotografía*" [124].

en lugar de imágenes de intensidad real (se guardan cambios de intensidad binario) [8]. Se producen cambios de brillo a nivel de píxel en lugar de cuadros de intensidad estándar. Emite eventos asíncronos con una resolución de microsegundos; generándose un evento cada vez que un solo píxel detecta un valor de cambio de intensidad. Como se aprecia en Figura 10, se considera que sucede un evento cuando se cumple que $\pm C = \log I(\mathbf{x}, t) - \log I(\mathbf{x}, t - \Delta t)$, siendo C la sensibilidad al contraste. Para comprender mejor el concepto es recomendable ver el siguiente video: [17]. Uno de los algoritmos que se suelen usar para trabajar en conjunto con la cámara de eventos, son las Spiking Neural Network, las cuales se explicarán próximamente.

En resumen, la visión neuromórfica es un sistema de visión inspirado en la biología; el cual está basado en eventos y no en fotogramas. Estos sistemas artificiales se basan en una retina artificial que genera flujos de eventos y esos se procesarán por una red neuronal. La visión humana no utiliza sucesiones de fotogramas como en las cámaras convencionales, nuestro cerebro trabaja con eventos, que son flujos continuos de información puntual generada por cada píxel, cada fotosensor de la retina (la matriz de fotosensores) de forma autónoma. En lugar de capturar imágenes a una velocidad fija, miden de forma asíncrona los cambios de brillo por píxel y emiten un flujo de eventos que codifican el tiempo, la ubicación y el signo de los cambios de brillo. Captura cambios de luminosidad, es decir, puede capturar sólo movimiento. Se basa en la resolución espacio-temporal entre la información que viaja por el cerebro, eso se procesa por todas las capas neuronales conforme se generan (en tiempo real), sin esperar tiempo de fotograma.

2.1.1 Aplicaciones

El hecho de tener un alto rango dinámico, no sufrir de desenfoque de movimiento y consumo de energía muy bajo, hacen que las cámaras para eventos sean un buen complemento para las cámaras estándar. En la Figura 11 se muestra una tabla comparando las propiedades de la cámara de eventos, la cámara estándar y la cámara de alta velocidad.

Pueden ser de gran utilidad en distintas aplicaciones. Tienen un gran potencial para la robótica (la visión por ordenador en escenarios difíciles para las cámaras tradicionales), vehículos autónomos, automatización industrial, Internet of Things (IoT), videojuegos y VR/AR, entre otras [7] [18] [8]. Sin embargo, "*actualmente la calidad del mapa 3D creado con cámaras de eventos no alcanza el mismo nivel de detalle y precisión que el de las cámaras estándar*" [7]. Es de gran utilidad en los vehículos autónomos o los drones. Se puede usar para control de tráfico, control de trayectoria y control cinemático [16]. En situaciones donde las escenas cambian mucho, "*el sensor tiene un consumo de energía equivalente al de un sensor de imagen convencional, aunque la precisión de tiempo es mucho mayor*" [19]. En otras palabras, *"permite rastrear el movimiento de la cámara y el objeto (flujo óptico) con mayor precisión*" [3]. Además, "*la independencia de píxeles permite que estas cámaras se adapten a escenas con regiones con mucha o poca luz*" [3].

La principal razón por la que estas cámaras de eventos aún no tienen mucho éxito, es que actualmente son costosas y "*las decisiones de diseño de hardware que toman las empresas de robótica tienden a ser conservadoras*". "*Las cámaras de eventos son simplemente demasiado nuevas*" [20]. En nuestra opinión, como ha pasado con muchas tecnologías, a medida que pase el tiempo es posible que baje su precio al aumentar su desarrollo. No obstante, las cámaras CMOS ofrecen obturadores cada vez más rápidos, mayor densidad de píxeles y rangos dinámicos más altos que las hacen bastante buenas para algunas aplicaciones en las que destacan las cámaras de eventos [20].

Es recomendable ver el siguiente video para conocer algunos vendedores de cámara de eventos, ver sus aplicaciones y conocer algunas de las redes neuronales que se pueden aplicar: [21]. También puede resultar interesante el video [22] sobre la cámara de eventos de la marca Sony. Además, si se quiere saber más sobre las cámaras de eventos puede consultar [23]. En [24] se resume la investigación realizada por el Grupo de Robótica y Percepción de la Universidad de Zúrich sobre la visión basada en eventos entre 2013 y 2017.

	High speed camera	Standard camera	Event Camera
Max fps or measurement rate	**Up to 1MHz**	100-1,000 fps	**1MHz**
Resolution at max fps	64x16 pixels	**>1Mpxl**	**>1Mpxl**
Bits per pixels (event)	12 bits	8-10 per pixel	**~40 bits/event** {t,(x,y),p)}
Weight	6.2 Kg	**30 g**	**30 g**
Active cooling	yes	**No cooling**	**No cooling**
Data rate	1.5 GB/s	32MB/s	**~1MB/s on average (depends on dynamics)**
Mean power consumption	150 W + external light	1 W	**1 mW**
Dynamic range	n.a.	60 dB	**140 dB**

Figura 11. Comparación de cámara estándar y cámara de eventos. Fuente: [8].

2.1.2 Resumen

No se trabaja con fotogramas, sino con flujos continuos de impulsos eléctricos (que se llaman eventos o 'spikes'). Son producidos de forma asíncrona por cada foto-sensor (o píxel) de la retina (la matriz de fotosensores) de forma autónoma cuando detecta un cambio suficiente de luz.

Cada píxel almacena un nivel de brillo de referencia y lo compara continuamente con el actual. Si la diferencia de brillo supera un umbral, genera un evento, en el que se codifica el tiempo, la ubicación y el signo del cambio de brillo. Los eventos positivos (ON) son cambios de menos a más luminosidad y los negativos (OFF) de más a menos luminosidad. Cada vez que se produce un evento, ese píxel restablece su nivel de referencia.

Las cámaras de eventos permiten capturar el movimiento con mucha más resolución temporal y menos consumo de energía. Además de aumentar el rango dinámico y disminuir el desenfoque de movimiento. En otras palabras, "*permite rastrear el movimiento de la cámara y el objeto con mayor precisión"*.

2.2 Spiking Neural Network (SNN)

Spiking Neural Network (Redes neuronales de impulsos) son redes neuronales artificiales[3] (ANN) que imitan más de cerca las redes neuronales biológicas. La SNN se diferencia de la ANN en la manera de propagar la información. La SNN, en lugar de trabajar con valores temporales que cambian continuamente, trabaja con eventos discretos que ocurren en momentos definidos. Toma un conjunto de spikes (picos, pulsos o impulsos) como entrada y produce un conjunto de spikes como salida, como se muestra en Figura 12.

[3] Para saber qué es una red neuronal, consulte [138].

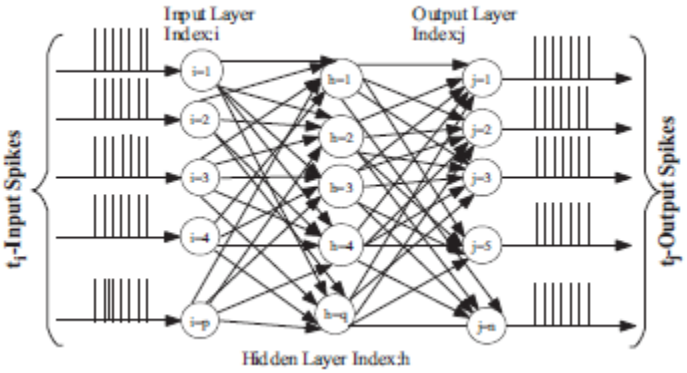


Figura 12. Arquitectura de una multicapa SNN. Fuente: [2].

Aparte del estado neuronal y sináptico[4], las SNN incorporan el concepto de tiempo en su modelo operativo. Como se ha mencionado antes, las entradas a las neuronas consisten en una sucesión de spikes, se presentan distribuidas en el tiempo. La idea es que las neuronas[5] en la SNN no transmiten información en cada ciclo de propagación (como sucede con las típicas redes de perceptrones multicapa), sino que transmiten información solo cuando un potencial de membrana[6] alcanza un valor específico, llamado umbral de disparo [2]. Cuando el potencial de membrana alcanza el umbral, la neurona se dispara y genera una señal (impulso o evento de salida) que viaja a otras neuronas vecinas, las cuales aumentan o disminuyen sus potenciales como consecuencia de la acción de esta señal. Funciona como las neuronas del cerebro, llegan varios spikes a una neurona y cuando pasa un límite, pasa un spike a la siguiente capa de neuronas que siguen a ésta. Sin embargo, las neuronas artificiales y biológicas no se comportan exactamente igual [2]. En la Figura 13 se ve la representación de la neurona biológica, comparándose con la neurona de una SNN. La sinapsis es "*un preprocesador de señales muy complejo que es fundamental para el aprendizaje y la adaptación*" [25]. En la Figura 14 se representan 'n' neuronas conectadas a una sola por sinapsis, siendo 'n' un número entero positivo.

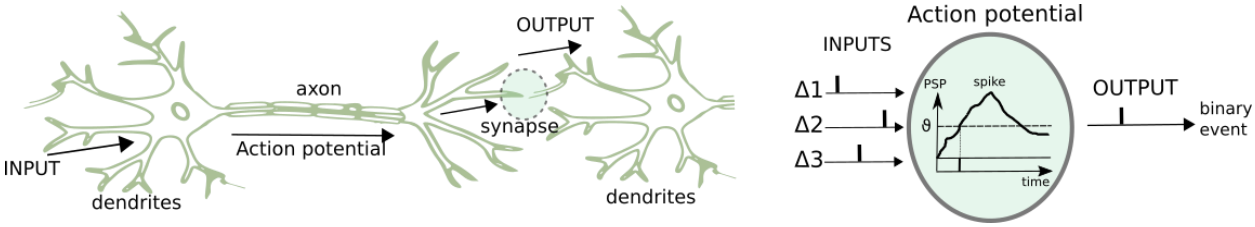


Figura 13. A la izquierda se ve la representación de dos neuronas biológicas. La imagen de la derecha es su funcionamiento, asociándolo con una SNN. Fuente: [26].

[4] Sinapsis: conexiones neuronales

[5] Neurona es el componente básico de un SNN y varias neuronas interconectadas forman las capas de entrada, ocultas y de salida. Esta neurona imita el modelo de integración y disparo general. *"El modelo de neurona de este tipo de redes cuenta con al menos una variable de estado, la cual representa el potencial de membrana de la neurona. Algunos modelos de neurona más complejos pueden incluir otras variables; como son el estado que cada tipo de sinapsis"* [33].

[6] Potencial de membrana: una cualidad intrínseca de la neurona relacionada con su carga eléctrica de membrana.

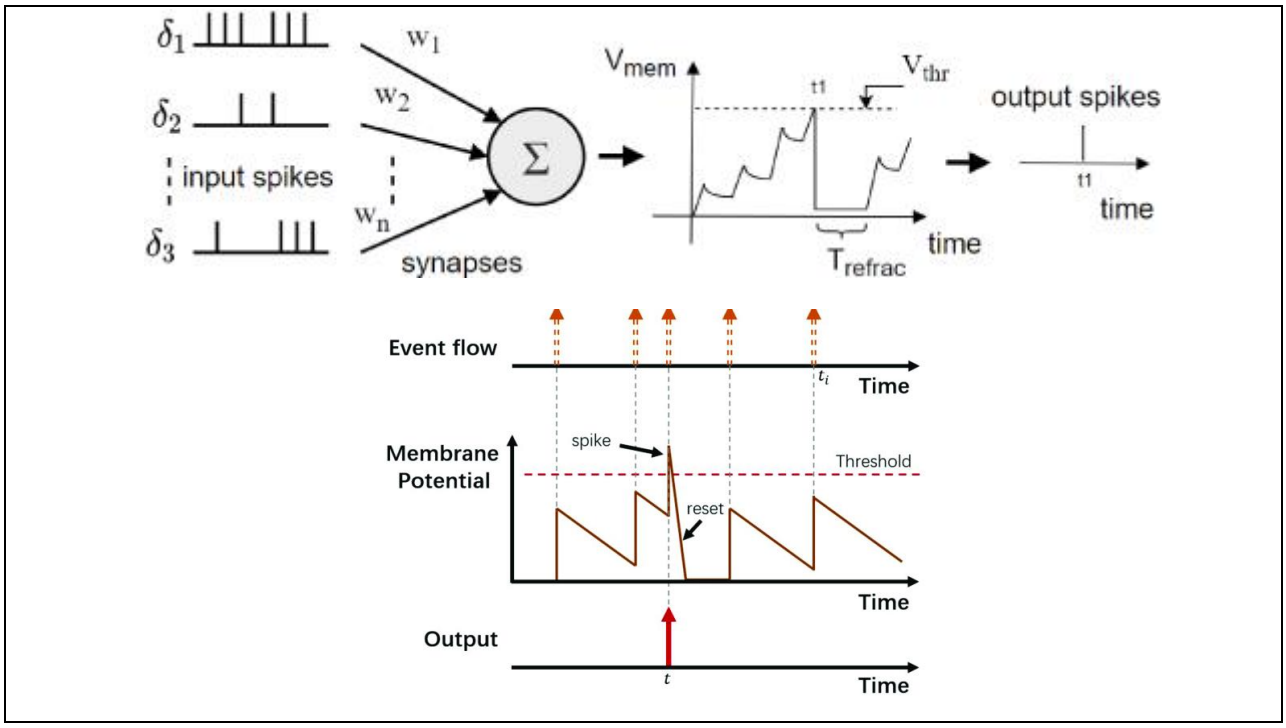


Figura 14. Potencial de membrana de la neurona de fuga-integración-disparo (LIF). La imagen de abajo es la gráfica de potencial de membrana en más detalle. Fuente: [27] [28].

En otras palabras, el estado actual de una neurona se define como su potencial de membrana (modelado como una ecuación diferencial). En la Figura 14 se ilustra en qué consiste el potencial de membrana de la neurona de fuga-integración-disparo (Leaky-Integrate-and-Fire (LIF)) [29]. La V_{mem} es el potencial de membrana, V_{thr} es la tensión umbral y la T_{refrac} es el período refractario en una neurona. Los spikes de entrada y salida se representan como eventos instantáneos. Los pulsos de entrada provocan que el potencial de membrana se incremente durante un período de tiempo, después disminuirá gradualmente (se produce el período refractario en la neurona). En el modelo LIF se considera que es el estado de la neurona, con spikes entrantes que empujan este valor hacia arriba o hacia abajo, hasta que el estado finalmente decae o, si se alcanza el umbral[7] (dispara). La variable de estado se restaura a un valor más pequeño después de disparar.

Se puede resumir el funcionamiento de la neurona en los siguientes pasos:

- Cada neurona tiene un valor que equivale al potencial eléctrico de las neuronas biológicas en un momento dado. En ausencia de estímulo, la membrana posee un potencial de reposo. Cada spike de entrada de las neuronas conectadas aumenta o disminuye su potencial de membrana, depende de su modelo matemático.

- *"Cada neurona presináptica dispara a su propio ritmo y los spikes son enviados hacia adelante por la sinapsis correspondiente. La intensidad del spike traducido a neurona post sináptica depende de la fuerza de la sinapsis de conexión. Ahora, debido a los spikes de entrada, el potencial de membrana de la neurona postsináptica aumenta"* [30]. Cuando el potencial cruza un valor de umbral, la neurona envía un único impulso a cada neurona descendente (post sináptica) conectada a la primera (presináptica), y la neurona entra en un período refractario en el que no se permiten nuevas entradas y el potencial permanece constante. El valor de la neurona cae inmediatamente por debajo de su media, pero el valor volverá gradualmente a su media con el tiempo [25]. Se podría monitorizar cuáles de las neuronas presinápticas ayudaron a disparar. *Esto podría hacerse observando qué neuronas presinápticas enviaron spikes antes de que se disparara la neurona postsináptica. De*

[7] El umbral se podría considerar un hiperparámetro. No obstante, en el aprendizaje no supervisado, es muy difícil entrenar una red donde los patrones tienen una cantidad variable de activaciones. Los patrones con activaciones más altas tienden a ganar en el aprendizaje competitivo y, por lo tanto, eclipsan a otros. Por lo tanto, se han introducido métodos de normalización para llevarlos a todos al mismo nivel. El umbral de cada patrón se calcula en función del número de activaciones que contiene. Cuanto mayor sea el número de activaciones, mayor será el valor del umbral.

esta manera, ayudaron en el spike post sináptico al aumentar el potencial de membrana y, por lo tanto, se fortalece la sinapsis correspondiente" [30].

Para saber más sobre el modelo LIF consulte [25]. La ecuación diferencial para el potencial de membrana en el modelo LIF es (2.1); haciéndose una analogía eléctrica con la membrana celular [31] [32].

$$I_{mA}(t) - \frac{V_m(t)}{R_m} = C_m \frac{dV_m(t)}{dt} \qquad (2.1)$$

Debido a que estas neuronas sólo reciben como entrada spikes, se han construido esquemas de codificación para interpretar estas secuencias de spikes como un número de impulsos y el tiempo en el que éstos se introducen. Los métodos para codificar la información usando trenes de spikes[8] tienen en cuenta la frecuencia del spike o el intervalo del spike [33]. Se podría establecer un modelo de red neuronal basado en el tiempo de generación de spikes.

Las SNN están diseñadas para hacer cálculos neuronales, hay que dotar de significado a los impulsos neuronales. El codificador hace la conversión de información a spikes de neuronas artificiales. Sirve como interfaz entre los datos numéricos (del mundo físico o simulaciones digitales) y los SNN. Hay diversas formas de realizar la codificación de información, podría ser: Codificación binaria, Códigos completamente temporales, Codificación de latencia, Codificación de tasa, codificación de posición[9], etc [2] [34]. Por ejemplo, en la binaria, las neuronas individuales se representan como unidades binarias que sólo pueden aceptar los valores de encendido o apagado. Una neurona está activa o inactiva en un intervalo de tiempo específico, disparando uno o más impulsos a lo largo de ese marco temporal.

Un concepto a tener en cuenta es el de campo receptivo [34] [35]. Es un área en la que la estimulación conduce a la respuesta de una neurona sensorial particular. La Figura 15 exhibe este proceso. Cuando es centrado, la iluminación del centro del campo receptor "*provoca un incremento en la tasa de disparo de potenciales de acción*" [36]. No hay si se ilumina solo la periferia. Cuando es descentrado, se comporta de forma opuesta.

Pongamos de ejemplo el caso de una SNN donde la entrada es una imagen [37]; el campo receptivo de una neurona sensorial es la parte de la imagen que aumenta su potencial de membrana. Para realizar un campo receptivo centrado, se utiliza una ventana deslizante cuyas celdas se pueden ponderar según la Distancia de Manhattan[10], desde el centro de la ventana. Los campos de las diferentes neuronas se superponen. La capa de neuronas de entrada tiene que ser alimentada con el estímulo provocado por su campo receptivo. El estímulo calculado a partir de la ventana deslizante es un valor analógico y debe convertirse en un tren de spikes para que la neurona pueda entenderlo, como ya se mencionó, hay que usar un codificador. El tipo de codificación adoptado en este caso podría ser la codificación de velocidad. Sugiere que la información es transportada por la tasa de disparo de la neurona. Por lo tanto, el tren de spikes generado tiene una frecuencia proporcional al potencial de membrana correspondiente [37]. "*El método de codificación de campo receptivo es muy similar a la operación de convolución utilizada en redes neuronales no-spiking*" [34].

[8] Los trenes de pulsos se refieren a que los impulsos de las neuronas llegan cada ciertos tiempos para ser procesadas. Una serie de impulsos es lo que se conoce como tren de impulsos o tren de spikes.
[9] Campos receptivos gaussianos (GRF)
[10] Geometría del taxista. La distancia entre dos puntos es la suma de las diferencias absolutas de sus coordenadas.

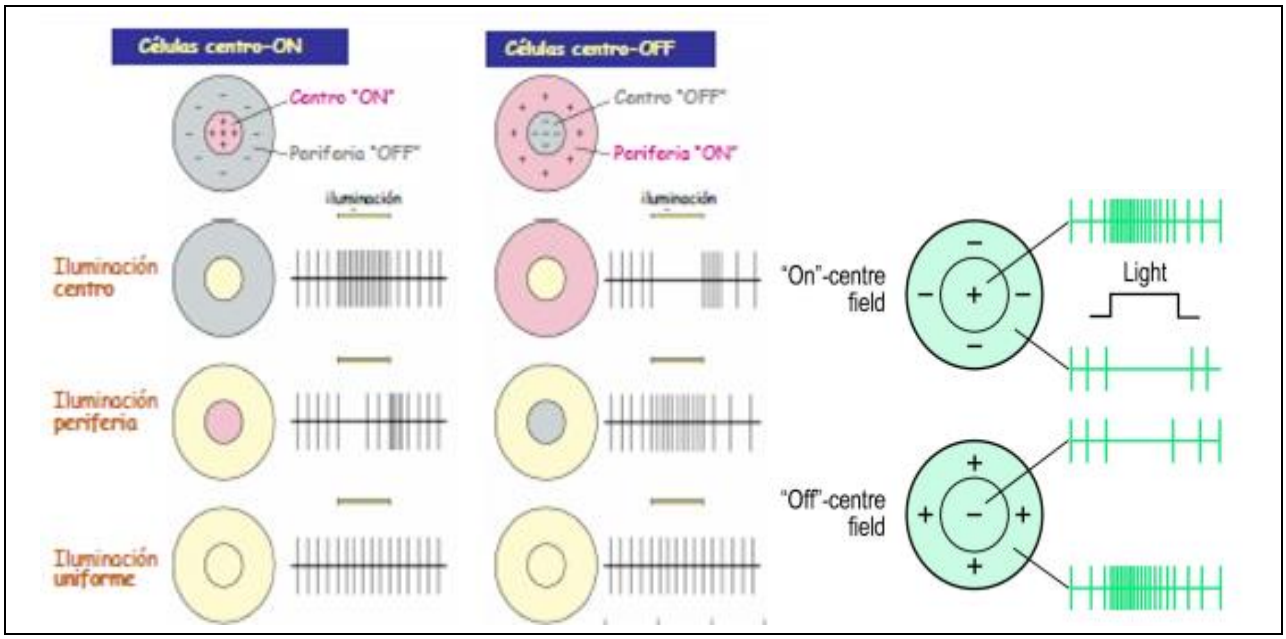


Figura 15. Campo receptivo neuronal descentrado y centrado y trenes de pulsos correspondientes. Fuente: [36] [34].

Los trenes de pulsos ofrecen una mayor capacidad para procesar datos espacio-temporales (datos sensoriales del mundo real) [31]. El aspecto espacial se refiere al hecho de que las SNN consideran el espacio, al conectar las neuronas solo a las neuronas cercanas, para que procesen los bloques de entrada por separado (similar a la CNN[11] utilizando un filtro). El aspecto temporal se refiere al hecho de que los trenes de pulsos se producen a lo largo del tiempo; consideran el tiempo codificando la información como trenes de pulsos para no perder información en una codificación binaria. En otras palabras, lo que se pierde usando codificación binaria, se gana en la información temporal de los pulsos [31]. "*Esto evita la complejidad adicional de una red neuronal recurrente (RNN[12]). Resulta que las neuronas de impulso son unidades computacionales más poderosas que las neuronas artificiales tradicionales*" [38].

Las neuronas spiking (de impulso) y las sinapsis de enlace se describen mediante pesos escalares configurables en la arquitectura SNN (Figura 12). La etapa inicial en la construcción de una SNN, como ya se mencionó, consiste en codificar los datos de entrada analógicos en los trenes de impulsos; utilizando técnicas basada en la velocidad, codificación temporal o codificación de la población. Un dato que mencionar es que las SNN están simplificadas respecto a las biológicas, se hace la suposición de que tienen una dinámica de umbral pura.

El aprendizaje se logra alterando los pesos sinápticos de valores escalares (Figura 14). En este tipo de redes permite aplicar reglas de aprendizaje basadas en el término de plasticidad dependiente del tiempo de los picos (STDP[13]). El peso que conecta una neurona pre y postsináptica se altera en función de sus tiempos de impulso relativos en intervalos de tiempo de decenas de milisegundos. El ajuste del peso se basa en información que es tanto local a la sinapsis como local en el tiempo [25]. Se puede aplicar aprendizaje supervisado o no supervisado.

[11] Redes neuronales convolucionales (CNN) o ConvNets. Para tener una idea inicial de lo que son redes neuronales convolucionales se recomienda ver: [125]. También se explican un poco en el apartado **¡Error! No se encuentra el origen de la referencia.**.

[12] "*Una red neuronal recurrente (RNN) es una clase de redes neuronales artificiales donde las conexiones entre nodos forman un gráfico dirigido a lo largo de una secuencia temporal. Esto le permite exhibir un comportamiento dinámico temporal*" [136].

[13] Las SNN se basa en la Plasticidad dependiente del tiempo de pico (STDP). Esto es un proceso biológico utilizado por el cerebro para modificar sus conexiones neuronales (sinapsis). Dado la eficiencia de aprendizaje del cerebro esta regla se incorporó en las ANN. Es "*un proceso que fortalece un peso sináptico si la neurona postsináptica se activa poco después de que se active la neurona presináptica, y lo debilita si la neurona postsináptica se activa más tarde*" [25]. El moldeado de pesos se basa en las siguientes dos reglas: a) Cualquier sinapsis que contribuya a la activación de una neurona postsináptica debe fortalecerse, es decir, debe aumentar su valor. b) Las sinapsis que no contribuyen a la activación de una neurona postsináptica deben disminuirse, es decir, su valor debe disminuir.

- No supervisado*: "Los datos se entregan sin etiqueta y la red no recibe comentarios sobre su rendimiento. Detectar y reaccionar a las correlaciones estadísticas en los datos es una actividad común. El aprendizaje Hebbiano[14] y sus generalizaciones punzantes, como STDP, son un buen ejemplo de esto"* [25].

- Supervisado: "*Los datos (la entrada) van acompañados de etiquetas (los objetivos), y el propósito del dispositivo de aprendizaje es correlacionar (clases de) entradas con las salidas objetivo (un mapeo o regresión entre entradas y salidas). Se calcula una señal de error entre el objetivo y la salida real y se utiliza para actualizar los pesos de la red*" [25].

"*El principal problema que plantea actualmente el uso práctico de las SNN es el del entrenamiento*" [31]. Aunque se dispone de métodos de aprendizaje biológico no supervisado (como el Hebbiano y el STDP), no se conocen aún métodos de entrenamiento supervisado suficientemente eficaces para las SNN. La activación basada en spikes de SNN no es diferenciable, lo que dificulta el desarrollo de métodos de entrenamiento basados en descenso de gradiente[15] para realizar la propagación inversa de errores; se perdería la información temporal precisa en los trenes de spikes [31].

Otra cuestión es que la simulación de las SNN en hardware normal tiene alto coste computacional, pues requiere simular ecuaciones diferenciales. No obstante, el hardware neuromórfico[16] pretende resolver este inconveniente simulando las neuronas mediante un hardware que puede aprovechar la naturaleza discreta y dispersa del comportamiento neuronal [31].

Se podría decir que hay dos tipos principales de capas en las SNN: capa convolucional frente a capa fully-connected (totalmente conectada).

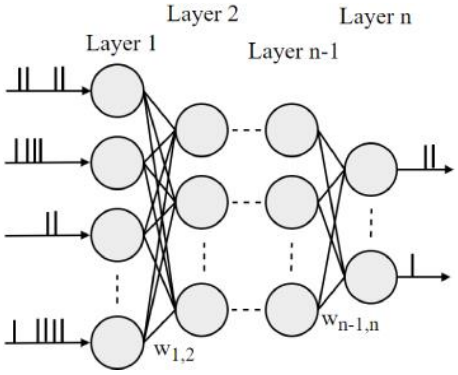


Figura 16. Fully connected SNN. Fuente: [27].

"Las neuronas de una capa totalmente conectada tienen conexiones completas con todas las activaciones de la capa anterior, como se ve en las redes neuronales normales. Por lo tanto, sus activaciones pueden calcularse con una multiplicación matricial seguida de una compensación de sesgo[17]" [39]. Como se ve en Figura 16, cada neurona de la primera capa está conectada a todas las neuronas de la segunda capa a través de la sinapsis.

A continuación, se explicará en qué consisten las redes convolucionales y cómo se les aplicarían a los sensores de visión basados en eventos.

---

[14] "*Aprendizaje hebbiano por lo general se refiere a algún tipo de abstracción matemática del principio original propuesto por Hebb*" [139].

[15] Gradient descent es un método general de minimización para cualquier función [128]. Se busca el mínimo global mirando alrededor cual tendría potencial más bajo, si comparamos con campos potenciales, encontrando el camino hasta el más bajo.

[16] Por ejemplo, el TrueNorth de IBM.

[17] Se le denomina bias o sesgo. "*La red neuronal está compuesta por neuronas de entrada que corresponden a los datos de una observación, y que se atribuye un peso a cada entrada. Esta pareja entrada/peso permite realizar la fase de propagación usando una función de activación*". Si se quiere forzar el valor de la predicción para algunos valores de entrada, se puede hacer con la ayuda de lo que se denomina un sesgo. "*Si a partir de los mismos valores de entrada, queremos que el valor de la predicción sea diferente y que se active la neurona*" [134].

2.2.1 Capa convolucional

La capa convolucional "*es el bloque central de una red convolucional que hace la mayor parte del trabajo computacional*" [39]. Las redes neuronales convolucionales son muy similares a las redes ordinarias, están constituidas por neuronas que poseen pesos y sesgos aprendibles. No obstante, asumen explícitamente que las entradas son imágenes. Tienen neuronas dispuestas en 3 dimensiones: anchura, altura y profundidad[18]. Las neuronas de una capa sólo estarán conectadas a una pequeña región de la capa anterior, en lugar de todas las neuronas de forma totalmente conectada. "*Puede interpretarse como una salida de una neurona que mira sólo una pequeña región en la entrada y comparte parámetros con todas las neuronas a la izquierda y a la derecha espacialmente*". "*La extensión espacial de esta conectividad es un hiperparámetro[19], es el llamado campo receptivo de la neurona*" (sería el tamaño del filtro) [39]. Este concepto ya se mencionó con anterioridad.

Al final de la arquitectura se reduce imagen completa a un único vector de puntuaciones de clase de la dimensión de profundidad [39], ilustrándose en Figura 17. Cada capa "*transforma el volumen de entrada 3D en un volumen de salida 3D de activaciones neuronales*"; mediante una función diferenciable que puede o no tener parámetros. Además, cada capa puede tener o no hiperparámetros adicionales [39].

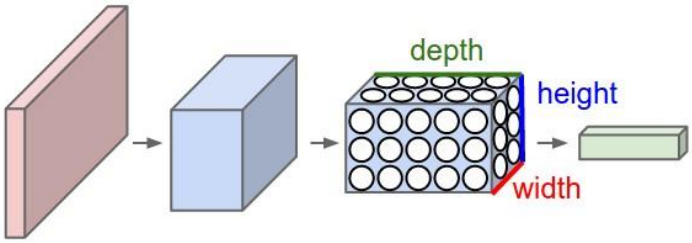


Figura 17. La red neuronal convolucional organiza sus neuronas en tres dimensiones. Fuente: [39].

Hay varios tipos de capas para construir las redes neuronales. Los 3 tipos principales de capas: Capa convolucional, capa de agrupación (Pooling) y capa totalmente conectada (Fully connected).

Un ejemplo de las capas que pueden conformar una CNN se muestra en la Figura 18. Se transforma "*la imagen original capa por capa desde los valores originales de los píxeles hasta las puntuaciones finales de las clases*" [39]. Algunas capas contienen parámetros y otras no. "*Las capas CONV/FC realizan transformaciones que son función no sólo de las activaciones en el volumen de entrada, sino también de los parámetros (los pesos y sesgos de las neuronas). Por otro lado, las capas RELU/POOL implementarán una función fija. Los parámetros de las capas CONV/FC se entrenan con descenso de gradiente*" [39], así las puntuaciones de clase sean consistentes con las etiquetas del conjunto de entrenamiento

[18] La profundidad se refiere a la tercera dimensión de un volumen de activación.

[19] "*Permiten controlar el proceso de entrenamiento de un modelo*" [135]. "*Los hiperparámetros de un modelo son los valores de las configuraciones utilizadas durante el proceso de entrenamiento. Son valores que generalmente se no se obtienen de los datos*". No hay que confundir parámetro con hiperparámetro. "*En los modelos de aprendizaje automático, los parámetros son las variables que se estiman durante el proceso de entrenamiento con los conjuntos de datos*". "*Se fijan los valores de los hiperparámetros para que con estos se obtengan los parámetros*" [140].

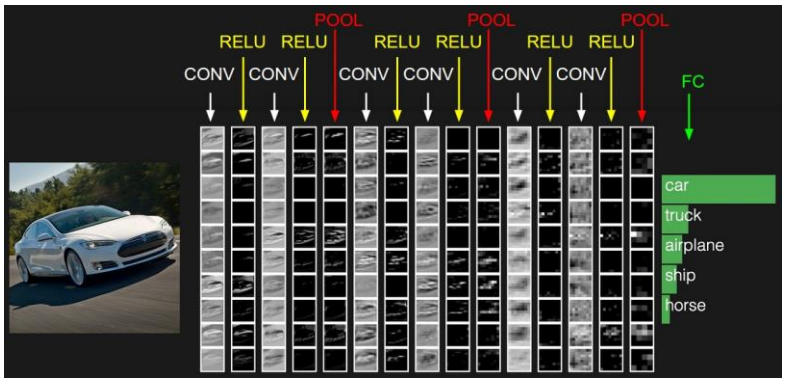


Figura 18. Una CNN. En la imagen se ha representado como una columna cada volumen de activaciones a lo largo de la ruta de procesamiento. Al ser difícil visualizar los volúmenes en 3D, están colocados los cortes de cada volumen en filas. Fuente: [39].

Centrándose en la capa convolucional (Figura 19), los parámetros consisten en un conjunto de filtros aprendibles que son pequeños en anchura y altura, pero se extiende por toda la profundidad del volumen de entrada (una imagen en color sería profundidad 3). Se hace convolución con cada filtro (o kernel) a lo largo de la anchura y la altura del volumen de entrada. Se producirá "*un mapa de activación bidimensional que da las respuestas de ese filtro en cada posición espacial*". "*La red aprenderá filtros que se activan cuando ven algún tipo de característica visual como un borde de alguna orientación o una mancha de algún color en la primera capa, o eventualmente patrones enteros tipo panal o rueda en capas superiores de la red*" [39]. En cada capa convolucional habrá un conjunto de filtros. Cada uno de ellos producirá un mapa de activación bidimensional independiente. El apilamiento de estos mapas determina el volumen de salida. Con respecto al tamaño del volumen de salida, los tres hiperparámetros que lo controlan son la profundidad, el paso y el relleno cero.

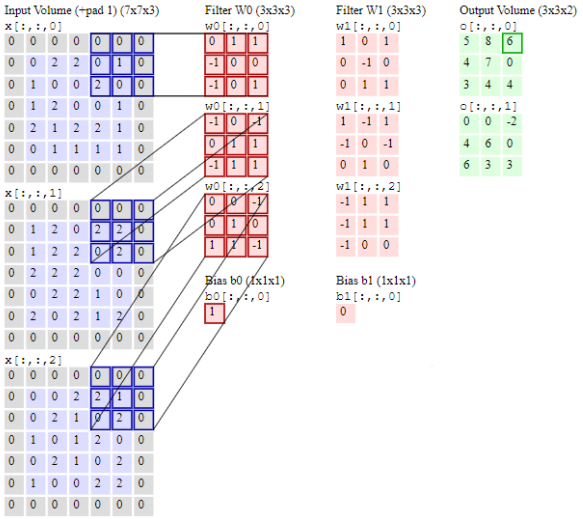


Figura 19. Ejemplo de capa de convolución. Fuente: [39].

2.2.1.1 Red convolucional con cámara de eventos

Una vez entendido el concepto de lo que consiste una red convolucional, ahora hay que entender cómo se utilizaría para el caso de las cámaras de eventos. Para ello se recomienda leer el artículo [40], el cual presenta un Módulo de Convolución Dirigida por Eventos (ConvModule) para computar convoluciones 2D en tales flujos de eventos. El módulo de convolución tiene capacidad multi-núcleo, es decir, selecciona el kernel (plantilla) de convolución dependiendo del origen del evento. Ahora, se hará un breve resumen de los aspectos más principales del artículo.

Los flujos de eventos pueden alimentar los "módulos de extracción de características basados en eventos". *"Una primera capa extraería características de bajo nivel, como bordes cortos orientados a diferentes escalas*

y ángulos, una segunda capa las combinaría en formas más complejas, y las capas siguientes seguirían combinando características más simples en otras más complejas y especializadas, hasta reconocer objetos específicos" [40]. Se procesa el flujo de eventos procedentes de los sensores evento por evento, con pequeños retrasos de procesamiento, preservando la resolución temporal de los eventos del sensor. Los flujos de eventos de entrada y salida tienen retrasos de procesamiento por evento insignificantes, los dos flujos son prácticamente simultáneos.

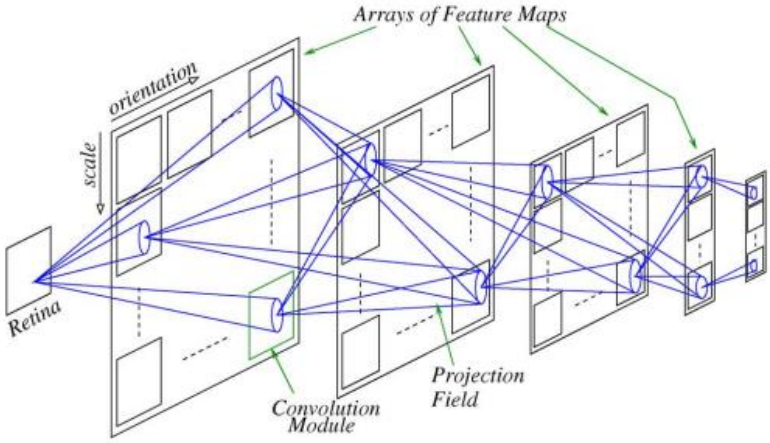


Figura 20. La estructura feed-forward multicapa de una arquitectura red convolucional típica. Fuente: [40].

En la Figura 20 se ve la estructura feed-forward multicapa de una arquitectura red convolucional típica. Cada capa contiene un conjunto de "mapas de características" (Feature Maps (FM)). Cada FM en la primera capa, después del sensor de visión, recibe solamente la entrada del sensor y computa una única convolución del núcleo (kernel). Sin embargo, los FM a partir de la segunda capa reciben más de un flujo visual, y para cada uno tienen que utilizar un kernel diferente para computar y acumular las convoluciones. El ConvModule del artículo tiene capacidad de multi-núcleo y puede computar y acumular diferentes convoluciones de núcleo en paralelo para múltiples flujos de entrada simultáneos.

La Figura 21 muestra el funcionamiento de un Event-Driven ConvModule (chip). Cuando se recibe un evento de coordenadas (x, y) desde un sensor de visión, "*se envía una contribución a un "Campo Receptivo" de píxeles en el ConvModule dirigido por eventos de destino*". Cuando un píxel detecta un determinado nivel de contraste, solicita acceso al bus AER (Address Event Representation) para enviar su coordenada utilizando un handshaking asíncrono. El ConvModule recibe el evento "*y lo envía a un "campo de proyección" de píxeles. Cada píxel del ConvModule acumula las contribuciones de los eventos entrantes, hasta alcanzar un umbral, en cuyo caso enviará un nuevo evento a través del puerto AER de salida del ConvModule*". "*Si* (x_i, y_i) *es la coordenada del evento entrante y* (x_c, y_c) *es un píxel dentro del Campo de Proyección de este evento, la contribución del evento de entrada a los píxeles del campo de proyección se pondera por un factor que depende de sus posiciones espaciales relativas*", $w = w(x_c - x_i, y_c - y_i)$; esta función define el kernel de convolución. Se programa con un kernel de Gabor 2D.

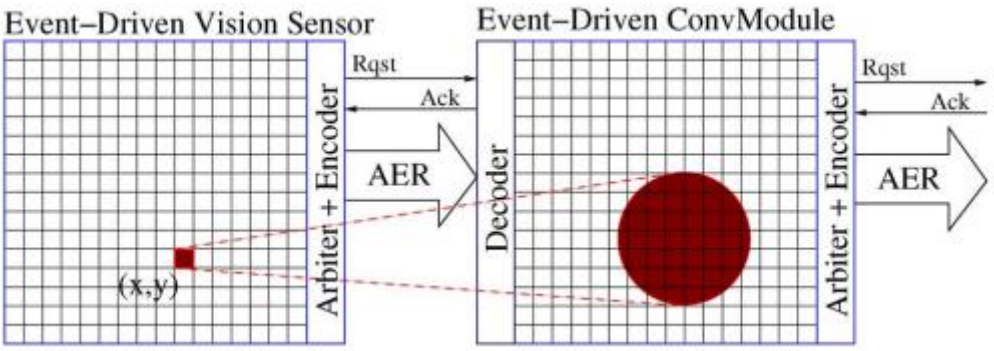


Figura 21. Cuando se recibe un evento de coordenadas (x, y) desde un sensor de visión (a la izquierda imagen), "*se envía una contribución a un "Campo Receptivo" de píxeles en el ConvModule dirigido por eventos de destino*" (a la derecha imagen está el procesador). Fuente: [40].

En una red convolucional jerárquica de reconocimiento de objetos, el reconocimiento puede lograrse tan pronto como el sensor proporcione suficientes eventos significativos correlacionados en el espacio-tiempo. Esto es lo llamado propiedad de "pseudo-simultaneidad" del procesamiento de convolución dirigido por eventos. Un requisito es que el sensor proporcione una representación dispersa de la realidad observada para no saturar la tasa de eventos máxima de los enlaces AER. Cada ConvModule reduce la tasa de eventos de la entrada a la salida; por ello la tasa de eventos más alta se encuentra en la salida del sensor.

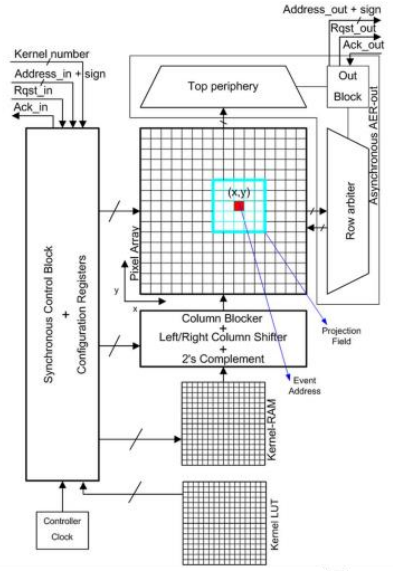


Figura 22. La arquitectura del ConvModule. Fuente: [40].

La arquitectura del ConvModule (Figura 22) se resume en lo siguiente: "*Los píxeles de la "matriz de píxeles" mantienen su estado de forma continua y dinámica. Cuando el módulo recibe un evento de entrada , el kernel (que es una matriz 2D almacenada en la "Kernel-RAM") se añade/sustrae al "Campo de Proyección" de los píxeles alrededor de la "Dirección del Evento". El signo del evento de entrada determina si el kernel se suma o se resta. Independientemente del flujo de eventos de entrada, todos los píxeles "sufren" una fuga de tasa constante que llevará su estado a un nivel de reposo. Cuando un píxel alcanza un umbral positivo (negativo), se restablece su nivel de reposo, y se envía un evento de salida con signo positivo (negativo) a través del puerto de salida del AER con la coordenada del píxel. De esta manera, los ConvModules son excelentes extractores de características espacio-temporales, ya que si se reciben suficientes eventos de entrada que representen la característica espacial del kernel cerca en el tiempo (para evitar el efecto de la fuga), se producen eventos de salida que representan la ubicación de estas características"* [40].

Otro artículo relacionado que se recomienda leer es [41], describe un chip de convolución para sistemas de detección y procesamiento de visión basados en eventos. "*Un procesador de eventos de convolución bidimensional de 32x32 píxeles cuyo kernel puede tener una forma y un tamaño arbitrarios hasta 32x32*" [41]. En él participan los mismos autores que el artículo explicado antes.

2.2.2 Aplicaciones

La mejor manera de entender las SNN es viendo ejemplos. En el artículo [42] se ilustra un caso de estimación de profundidad, útil para robots. Se comentará su contenido brevemente.

La estimación de la profundidad es una importante tarea de visión por ordenador, útil en particular para la navegación en vehículos autónomos, o para la manipulación de objetos en robótica. En el artículo se propone resolverla utilizando StereoSpike en el artículo, un enfoque neuromórfico de extremo a extremo, que combina dos cámaras basadas en eventos y una red neuronal de Spiking Neural Network (SNN) con una arquitectura de codificación y decodificación codificador-decodificador ligeramente modificada. "*Más concretamente, se utiliza el conjunto de datos de la cámara de eventos estereoscópicos de múltiples vehículos (MVSEC). Proporciona una profundidad que se utilizó para entrenar a StereoSpike de forma supervisada, utilizando el método de descenso de gradiente sustituto. StereoSpike podría implementarse en chips neuromórficos, abriendo la puerta a sistemas integrados de bajo consumo y en tiempo real*" [42].

"*El procesamiento de la profundidad en los seres humanos está muy bien desarrollado y se basa en señales visuales monoculares y binoculares*". "*Consume muy poca energía, ya que el sistema visual codifica la información de la retina en forma de potenciales de acción, o spikes*". Las SNN "*son una buena opción para las DVS, ya que pueden aprovechar la escasez de sus resultados*". Además, las SNN mantienen el mismo nivel de plausibilidad biológica (level of biological plausibility) que las retinas de silicio [42].

En el documento se muestra que el problema de la estimación de la profundidad a partir de estos flujos de eventos neuromórficos puede tratarse como una tarea no temporal; reinician todas las neuronas a un potencial de membrana de cero en cada paso de tiempo. Si bien esta característica puede no aprovechar plenamente las capacidades de procesamiento temporal de las SNN, disminuye drásticamente la carga computacional y energética del modelo [42].

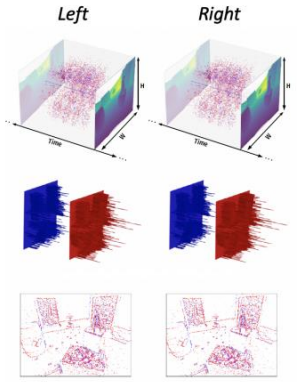


Figura 23. Representa los datos agrupando todos los spikes entrantes en cada píxel en una ventana de tiempo de 50 ms. En la imagen de arriba del todo, los fotogramas mostrados en el eje temporal son los mapas de profundidad reales, proporcionados por el LIDAR a 20 Hz. Fuente: [42].

Como se presenta en la Figura 23, representan los datos agrupando todos los spikes entrantes en cada píxel en una ventana de tiempo de 50 ms. Acumulando los spikes de cada polaridad en un canal diferente. Como hay dos polaridades, el tensor resultante tiene una forma de (2, Altura, Anchura) y contiene enteros positivos, correspondientes al número de spikes de cada polaridad que aparecían en cada posición de la escena durante la

ventana de tiempo. El tensor de entrada final se obtiene concatenando los de las cámaras izquierda y derecha, lo que da lugar a un volumen en forma de (4, Altura, Anchura) [42].

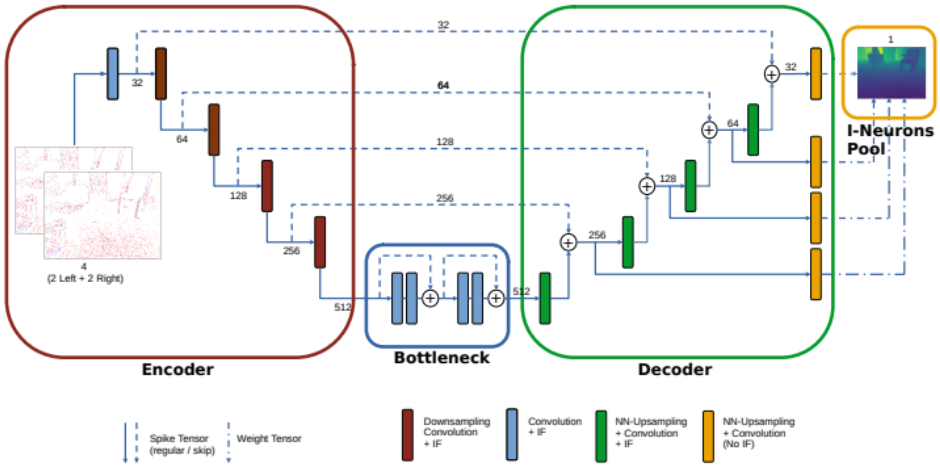


Figura 24. Arquitectura detallada de StereoSpike. Fuente: [42].

Con respecto a la arquitectura del StereoSpike (Figura 24), "*su codificador contiene una o dos ramas, cada una para una cámara DVS. La salida de ambas ramas se combina con una función de suma, y se procesa posteriormente a través de un bottleck que consiste en 2 bloques SEWResBlocks seguidos y con la función ADD connect. Como resultado, el tensor que sale de estas capas residuales está compuesto por enteros en el rango [0, 4]. Esta representación es muestreada progresivamente por las capas del decodificador y cuyas salidas se suman con los tensores de spikes del mismo nivel del codificador, dando lugar a valores enteros en el rango [0, 3]. Paralelamente, las sinapsis de predicción de diferentes escalas se proyectan directamente a las neuronas I de salida cuyos potenciales llevan la predicción final. Los números indican el tamaño de la dimensión del canal para cada volumen de spikes*".

El código Python del artículo se encuentra en [43]. PyTorch y SpikingJelly son las bibliotecas principales de desarrollo en ese proyecto. PyTorch es una herramienta para el aprendizaje profundo y la diferenciación automática, mientras que Spikingjelly es un marco de trabajo de código abierto para las SNN, basado en PyTorch. También podría resultar de utilidad el uso de SpykeTorch; un simulador en Python de redes neuronales convolucionales de spike del ecosistema PyTorch. "*Fue desarrollado inicialmente para trabajar con SNNs*" [44].

En resumidas palabras, el procedimiento de entrenamiento adoptado en la SNN del artículo es:

1. Inicializar todos los potenciales (incluyendo la salida) a cero,

2. pasar hacia delante una muestra o un lote de ellas,

3. calcular la pérdida y retropropagarla, (para entender esto, se puede pensar que actúa como un control feed fordward[20] que compensa el error)

4. actualizar los pesos y

[20] "*El término prealimentacion (Feed-forward) describe un tipo de sistema que reacciona a los cambios en su entorno, normalmente para mantener algún estado concreto del sistema. Un sistema que exhibe este tipo de comportamiento responde a las alteraciones de manera predefinida, en contraste con los sistemas retroalimentados.*
Se necesitan muchos prerrequisitos para implementar un sistema feed-forward: las alteraciones deben poder medirse, sus efectos en la salida del sistema deben ser conocidos y el tiempo durante el que las alteraciones afectan a la salida debe ser mayor que el del sistema en sí. Si estas condiciones se cumplen, el sistema feed-forward debe ser afinado para que sea extremadamente efectivo" [129]. Para encontrar información de las Feedforward neural network mire [137]

5. empezar de nuevo con una nueva muestra.

También puede resultar de interés el artículo [45] donde implementaron un algoritmo basado en eventos en una FPGA[21] para reconstruir mapas de profundidad de baja resolución en un conjunto de datos de pequeño tamaño. Mientras que en el artículo [46] se propuso una SNN para el procesamiento de la profundidad a partir del desenfoque (DFD). En [34] "*analiza las Redes Neuronales Spiking desde la perspectiva de Aprendizaje Automático, donde la plausibilidad biológica no es el objetivo principal, pero la capacidad de crear algoritmos de inteligencia artificial basados en SNN es uno de los objetivos principales, junto con su viabilidad de implementación de hardware*". Haciéndose hincapié en su en su implementación hardware dirigidas a FPGA. En él se propone una nueva arquitectura de neuronas de tipo LIF. STDP es el algoritmo de aprendizaje más popular para las SNN, derivado de los fenómenos biológicos. Rara vez se usan implementaciones de hardware digital de la STDP, dado que el algoritmo usa causalidad de sincronización hacia atrás, supone un empleo significativo de recursos de hardware. Ahí usa una versión mejorada. También abarca la implementación FPGA de algoritmos de codificación visual. Se describe la codificación de campos receptivos visuales tipo Gabor[22]. Para saber más sobre las funciones Gabor puede consultar [47].

Las SNN tienen la ventaja que se pueden usar para estimular sistemas nerviosos de seres biológicos, aparte de poder usarse, en principio, en las mismas aplicaciones que las ANN. Muchos investigadores están probando su utilización en drones, un ejemplo de su uso en drones pequeños está en [48]. En [49] se realiza segmentación de movimiento del mundo real utilizando la cámara DVS basada en eventos como entrada, el tema está bastante relacionado con este proyecto. Se puede encontrar más información sobre las SNN en [50] [51] [52] [53] [54] [55] [56] [57] [58] [30] [59] [27] [28].

2.2.3 Resumen

Los eventos producidos serían procesados por una red neuronal. Las SNN trabajan con eventos discretos que ocurren en momentos definidos. Toma un conjunto de impulsos como entrada y produce un conjunto de impulsos como salida. Las neuronas transmiten información solo cuando un potencial de membrana alcanza un valor llamado umbral de disparo. Cuando sucede, la neurona se dispara y genera un evento de salida que viaja a otras neuronas vecinas, las cuales aumentan o disminuyen sus potenciales. Los pulsos de entrada provocan que el potencial de membrana se incremente durante un período de tiempo, luego disminuirá gradualmente. Después de disparar entra en un período refractario en el que no se permiten nuevas entradas.

Existen esquemas de codificación para interpretar estas secuencias de pulsos como un número de impulsos y el tiempo en el que éstos se introducen. Se hace la conversión de información a pulsos de neuronas artificiales.

Los trenes de pulsos ofrecen una mayor capacidad para procesar datos espacio-temporales (datos sensoriales del mundo real). Las SNN consideran el espacio, al conectar las neuronas solo a las neuronas cercanas, para que procesen los bloques de entrada por separado (similar a las CNN utilizando un filtro). El aspecto temporal se refiere al hecho de que los trenes de pulsos se producen a lo largo del tiempo; consideran el tiempo codificando la información como trenes de pulsos. Las neuronas spiking y las sinapsis de enlace se describen mediante pesos escalares configurables. El aprendizaje se logra alterando los pesos sinápticos de valores escalares. Este tipo de redes permite aplicar reglas de aprendizaje basadas en el término de plasticidad dependiente del tiempo de los picos (STDP). STDP es el algoritmo de aprendizaje más popular para las SNN, derivado de los fenómenos biológicos. El peso que conecta una neurona pre y postsináptica se altera en función de sus tiempos de impulso relativos en intervalos de tiempo de decenas de milisegundos. El ajuste del peso se basa en información que es tanto local a la sinapsis como local en el tiempo. Cualquier sinapsis que contribuya a la activación de una neurona

[21] Field-Programmable Gate Arrays. "*Los dispositivos FPGA son elegidos debido a sus excelentes capacidades de cálculo paralelo masivo, bajo consumo de energía, baja latencia y versatilidad*" [133].

[22] "*Los campos específicos de orientación de Gabor son importantes en el procesamiento de imágenes, ya que son fenómenos bien estudiados observados en la corteza visual de mamíferos y se desempeñan bien en el procesamiento de imágenes y en las tareas de codificación de spikes*" [133].

postsináptica debe aumentar su valor. Las sinapsis que no contribuyen a la activación de una neurona postsináptica deben disminuirse.

Un concepto a tener en cuenta es el de campo receptivo. Siendo una imagen la entrada de la SNN, el campo receptivo de una neurona sensorial es la parte de la imagen que aumenta su potencial de membrana. Para realizar un campo receptivo centrado, se utiliza una ventana deslizante cuyas celdas se pueden ponderar desde el centro de la ventana. La capa de neuronas de entrada tiene que ser alimentada con el estímulo provocado por su campo receptivo. El estímulo calculado a partir de la ventana deslizante es un valor analógico y debe convertirse en un tren de pulsos.

Hay dos tipos principales de capas en las SNN: capa convolucional y capa totalmente conectada. La red convolucional asume que las entradas son imágenes. Tienen neuronas dispuestas en 3 dimensiones: anchura, altura y profundidad. Las neuronas de una capa sólo estarán conectadas a una pequeña región de la capa anterior, en lugar de todas las neuronas de forma totalmente conectada. La extensión espacial es el campo receptivo de la neurona. Al final de la arquitectura se reduce imagen completa a un único vector de puntuaciones de clase de la dimensión de profundidad. Los parámetros consisten en un conjunto de filtros aprendibles que son pequeños en anchura y altura, pero se extiende por toda la profundidad del volumen de entrada. Se hace convolución con cada filtro (o kernel) a lo largo de la anchura y la altura del volumen de entrada. *La red aprenderá filtros que se activan cuando ven algún tipo de característica visual como un borde de alguna orientación".* En cada capa convolucional habrá un conjunto de filtros. Cada uno de ellos producirá un mapa de activación bidimensional independiente.

Las capas de convolución se recomienda usarlas en el caso de las cámaras de eventos. Hay diferentes formas de realizar la SNN, tanto en hardware como en software. Una manera es la siguiente: Se selecciona el kernel (plantilla) de convolución dependiendo del origen del evento. *"Una primera capa extraería características de bajo nivel, como bordes cortos orientados a diferentes escalas y ángulos, una segunda capa las combinaría en formas más complejas, y las capas siguientes seguirían combinando características más simples en otras más complejas y especializadas, hasta reconocer objetos específicos*". Se usaría una estructura feed-forward multicapa. Cada capa contiene un conjunto de "mapas de características" (Feature Maps (FM)). Cada FM en la primera capa, después del sensor de visión, recibe solamente la entrada del sensor y computa una única convolución del núcleo (kernel). Sin embargo, los FM a partir de la segunda capa reciben más de un flujo visual, y para cada uno tienen que utilizar un kernel diferente para computar y acumular las convoluciones.

Al usar un Módulo de Convolución Dirigida por Eventos (ConvModule), cuando se recibe un evento de coordenadas (x, y) desde un sensor de visión, "*se envía una contribución a un "Campo Receptivo" de píxeles en el ConvModule".* "*Cada píxel del ConvModule acumula las contribuciones de los eventos entrantes, hasta alcanzar un umbral, en cuyo caso enviará un nuevo evento*" a través del puerto de salida. "*Si* (x_i, y_i) *es la coordenada del evento entrante y* (x_c, y_c) *es un píxel dentro del Campo de Proyección de este evento, la contribución del evento de entrada a los píxeles del campo de proyección se pondera por un factor que depende de sus posiciones espaciales relativas*", $w = w(x_c - x_i, y_c - y_i)$; esta función define el kernel de convolución. En una red convolucional jerárquica de reconocimiento de objetos, el reconocimiento puede lograrse tan pronto como el sensor proporcione suficientes eventos significativos correlacionados en el espacio-tiempo. "*Los ConvModules son excelentes extractores de características espacio-temporales, ya que si se reciben suficientes eventos de entrada que representen la característica espacial del kernel cerca en el tiempo, se producen eventos de salida que representan la ubicación de estas características".*

2.3 Foveación

La visión humana (Figura 25) usa un mecanismo de foveación para maximizar la resolución espacial en la zona donde se enfoca la vista (zona foveal); mientras que mantiene una baja resolución en las zonas de visión

periférica, como se muestra en la Figura 26. Sólo una fracción muy pequeña de nuestro campo de visión tiene alta resolución. Así se reduce la cantidad de información generada por la retina manteniendo la capacidad de reconocimiento visual. La región foveal del ojo es el centro de la alta agudeza visual y la visión del color, y esta región puede utilizarse para optimizaciones técnicas [60]. El proyecto consiste en una serie de técnicas para ahorrar procesamiento en píxeles que están fuera de la región denominada foveal. Se quiere aplicar un mecanismo de foveación a las retinas artificiales.

"*La visión central es la zona de alta definición, de color y de resolución de rasgos, lo que se denomina "reconocimiento de la esencia", mientras que la visión periférica es menos perceptiva del color y es mucho más sensible a la detección del movimiento, es decir, a los cambios temporales en la intensidad de los "píxeles". También resulta que la información que el cerebro interpola de cada uno de estos campos es diferente, por lo que puede utilizarse para afinar la escena percibida*" [61]. En la Figura 27 se ilustra que la máxima resolución se percibe en el ángulo de visión de 30º.

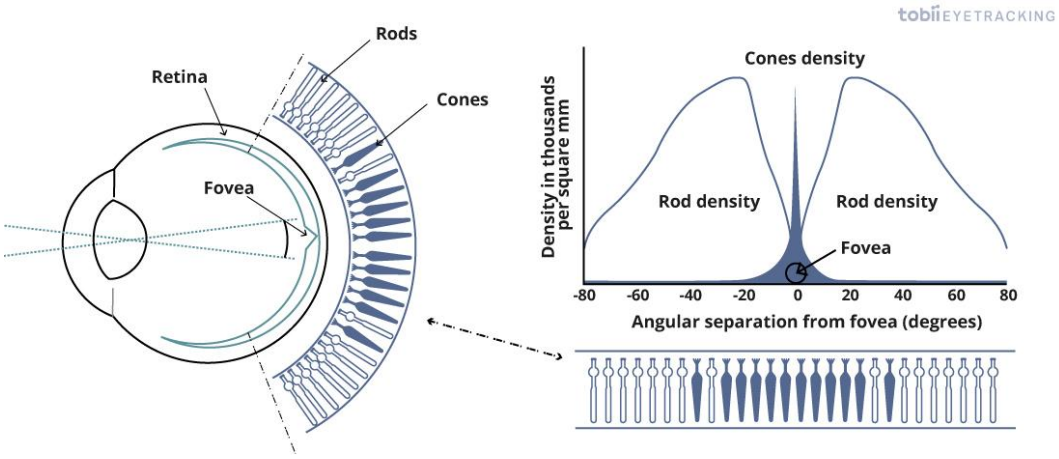


Figura 25. "*La foveación aprovecha que sólo una fracción muy pequeña de nuestro campo de visión tiene alta resolución*". Fuente: [62].

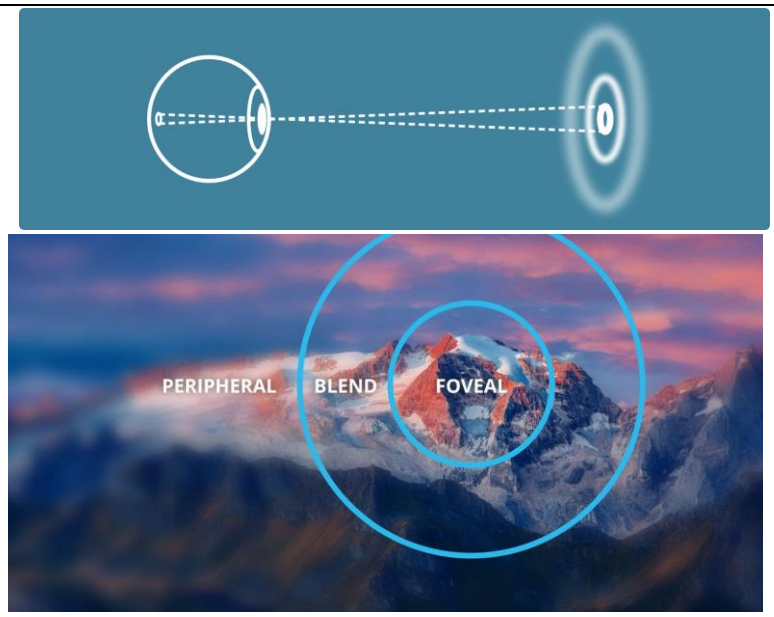


Figura 26. Ejemplo de en qué consiste la foveación. Fuente: [60] [63].

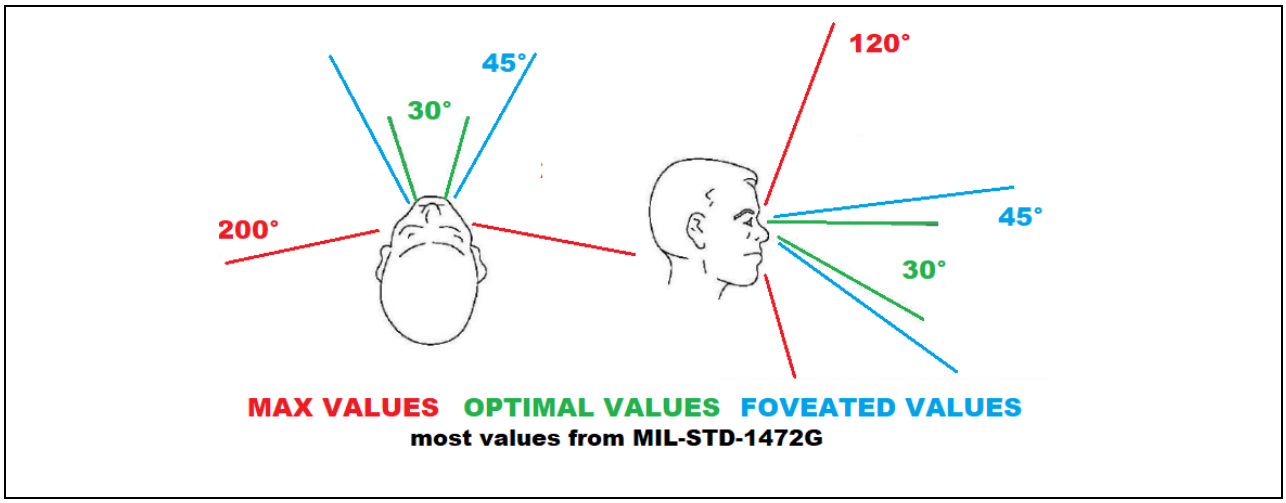


Figura 27. Ángulos de visión. Fuente: [61].

2.3.1 Aplicaciones

Teniendo en cuenta el mecanismo de foveación, han sido realizados diversos proyectos para reducir coste computacional, una representación más rápida, menor consumo de energía, mejorar gráficos o mejorar el transporte ahorrando ancho de banda [62]. Se puede tener en cuenta en tecnología que use el seguimiento ocular, por ejemplo, para la realidad virtual. En [64] se encuentra un ejemplo sobre el rendimiento foveado (FR), es decir, la técnica de realidad virtual (RV) que adapta la calidad de la imagen a la fijación del ojo del usuario. En el artículo [65] se propone un modelo neuronal de extremo a extremo para la visión foveal-periférica, inspirado en el mapeo retino-cortical en humanos.

2.4 Conclusiones

Las cámaras de eventos permiten capturar el movimiento con mucha más resolución temporal que una cámara estándar. Capturan los cambios de luminosidad que se producen delante de la cámara, cuando un fotosensor de la retina detecta un cambio de luz superior al umbral, pues compara con los valores de intensidad anteriores, se produce un evento. En un evento está codificado el tiempo en que se produjo, las coordenadas de posición en la retina (x,y) y el signo de cambio de luminosidad. Cuando no hay cambios de luz no se producen eventos, haciendo que disminuya la cantidad de datos y, por tanto, el consumo. Permitiendo obtener una retina que imita a la del ojo humano.

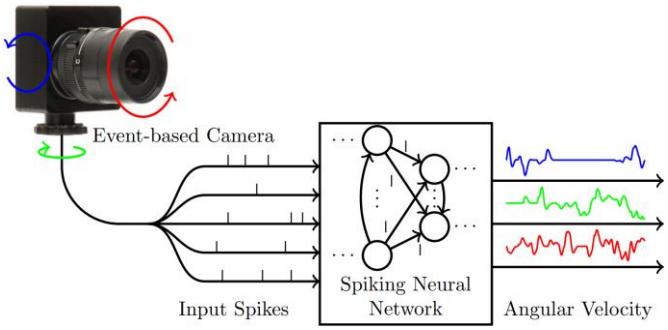


Figura 28. "*Las redes neuronales de spikes (SNN) son redes bioinspiradas que procesan la información transmitida como spikes temporales en lugar de valores numéricos. Un ejemplo de sensor que proporciona este tipo de datos es la cámara de eventos*". Fuente: [66].

Las SNN "*pretenden tender un puente entre la neurociencia y el aprendizaje automático*" [31]. Son redes neuronales artificiales que intentan imitar el comportamiento de las biológicas. Procesan la información

transmitida con spikes, que son eventos discretos que tienen lugar en puntos del tiempo. La aparición de un spike está determinado por ecuaciones diferenciales que representan procesos biológicos, el más importante es el potencial de membrana de la neurona. Las neuronas transmiten información solo cuando un potencial de membrana alcanza un valor llamado umbral de disparo, posteriormente se restablece el potencial. El modelo más común para esto es el modelo LIF. Cada neurona genera un evento de salida que viaja a otras neuronas vecinas, las cuales aumentan o disminuyen sus potenciales. Los pulsos de entrada provocan que el potencial de membrana se incremente durante un período de tiempo, después disminuirá gradualmente. Después de disparar entra en un período refractario en el que no se permiten nuevas entradas. El aprendizaje se logra alterando los pesos sinápticos. El ajuste del peso se basa en información que es tanto local a la sinapsis como local en el tiempo. Cualquier sinapsis que contribuya a la activación de una neurona postsináptica debe aumentar su valor. Las sinapsis que no contribuyen a la activación de una neurona postsináptica deben disminuirse.

Son lo más recomendable para trabajar con cámaras de eventos, pues el sensor de visión proporciona spikes (Figura 28). Sólo produce un evento cuando un píxel informa de un cambio de brillo significativo. Del mismo modo, la neurona con spikes de una SNN sólo produce un spike cuando se produce un número significativo de spikes en un corto período de tiempo. Debido a su modelo computacional basado en spikes, las SNN pueden procesar la salida de sensores asíncronos basados en eventos sin ningún tipo de preprocesamiento y con una potencia extremadamente baja, a diferencia de las redes neuronales artificiales estándar [66]. Se recomienda ver [67].

Hay diferentes formas de realizar la SNN, tanto en hardware como en software. Suelen usarse capas convolucionales. Las neuronas de una capa sólo estarán conectadas a una pequeña región de la capa anterior. La extensión espacial es el campo receptivo de la neurona. Al final de la arquitectura, se reduce imagen completa a un único vector de puntuaciones de clase. Los parámetros consisten en un conjunto de filtros aprendibles. Se hace convolución con cada filtro a lo largo de la anchura y la altura del volumen de entrada. Aprenderá "*filtros que se activan cuando ven algún tipo de característica*". En cada capa convolucional habrá un conjunto de filtros.

La principal dificultad que tienen estas redes es a la hora de realizar el entrenamiento, pues la activación basada en pulsos de SNN no es diferenciable. No obstante, en el artículo [42] explica cómo utilizarlas con sensores de visión basados en eventos. Por otra parte, en los artículos [40] y [41] se explican cómo hacer una red convolucional para DVS, haciéndolo en chip. Los eventos "*pueden ser procesados sobre la marcha por chips de convolución basados en eventos, proporcionando a su salida un flujo continuo de eventos que representa la versión filtrada en 2D del flujo de entrada*" [41]. Hay diferentes formas de implementar un SNN tanto en software como en hardware.

En la cámara de eventos puede resultar de interés imitar la foveación que realiza el ojo humano, es decir, aumentar la resolución en la zona de interés y disminuir el resto. La modificación dinámica de la resolución por zonas. El ojo humano no tiene una alta resolución en toda su área de fotosensores, tiene una muy alta resolución en la zona central y muy baja resolución en la periferia. Mediante nuestro enfoque, cuando miramos un punto, la zona cercana al punto es donde está la mayor concentración de fotosensores, la visión periférica es más turbia. Se trata de reproducir ese mecanismo de foveación en las retinas artificiales, incluyendo la capacidad de la retina de combinar pixeles o no combinarlos, tener la resolución total o reducirla. Para fovear una retina artificial se combinaría el valor de varios pixeles; hacer clusters de píxeles. Los eventos se generan sobre la suma (el conjunto), no sobre los pixeles individuales. Con el fin de reducir la cantidad de información innecesaria que se procesa.

Se pueden encontrar varios proyectos recientes sobre las cámaras de eventos y los algoritmos que se usan. Por ejemplo, en [68] se muestra un proyecto que proporciona un pipeline de vídeo que utiliza sensores basados en eventos para capturar el proceso de visión. Puede ejecutarse en PC y con la placa Xilinx Pynq-Z2. Por otra parte, puede resultar de interés consultar [69] [13] [70] [11].

3 PROCEDIMIENTO REALIZADO

"Los errores por usar datos inadecuados son mucho menores que los que se cometen por no usar ningún tipo de datos."

Charles Babbage

Este proyecto trata sobre el estudio de la aplicación de mecanismo de foveación sobre los sistemas de visión artificial basada en eventos. En concreto, se ha modelado a nivel de software distintas estrategias para controlar de forma dinámica la resolución de un sensor de visión, usando la máxima resolución posible en una o varias regiones de interés (identificadas a partir de distintos métodos de detección y seguimiento de objetos) y reduciendo la resolución en la periferia.

La finalidad última será diseñar una retina de eventos capaz de modificar su resolución por zonas de forma dinámica, y para ello se tiene que estudiar este mecanismo a nivel de simulación. Con el fin de hacerse una idea de la resolución que se debe aplicar y de otros parámetros cuando se implemente en hardware; obtener información previa al diseño. En la Figura 29 se ilustran los pasos realizados en este proyecto: Se parte de una grabación de eventos; se convierten a fotogramas; luego se interpolan las intensidades de esos fotogramas para aumentar la resolución temporal; se seleccionan las zonas de interés; se le aplica la foveación combinando pixeles en la periferia de los objetos, es decir, se hace cambio de resolución en la zona que no interesa; y finalmente se generan eventos con las imágenes modificadas.

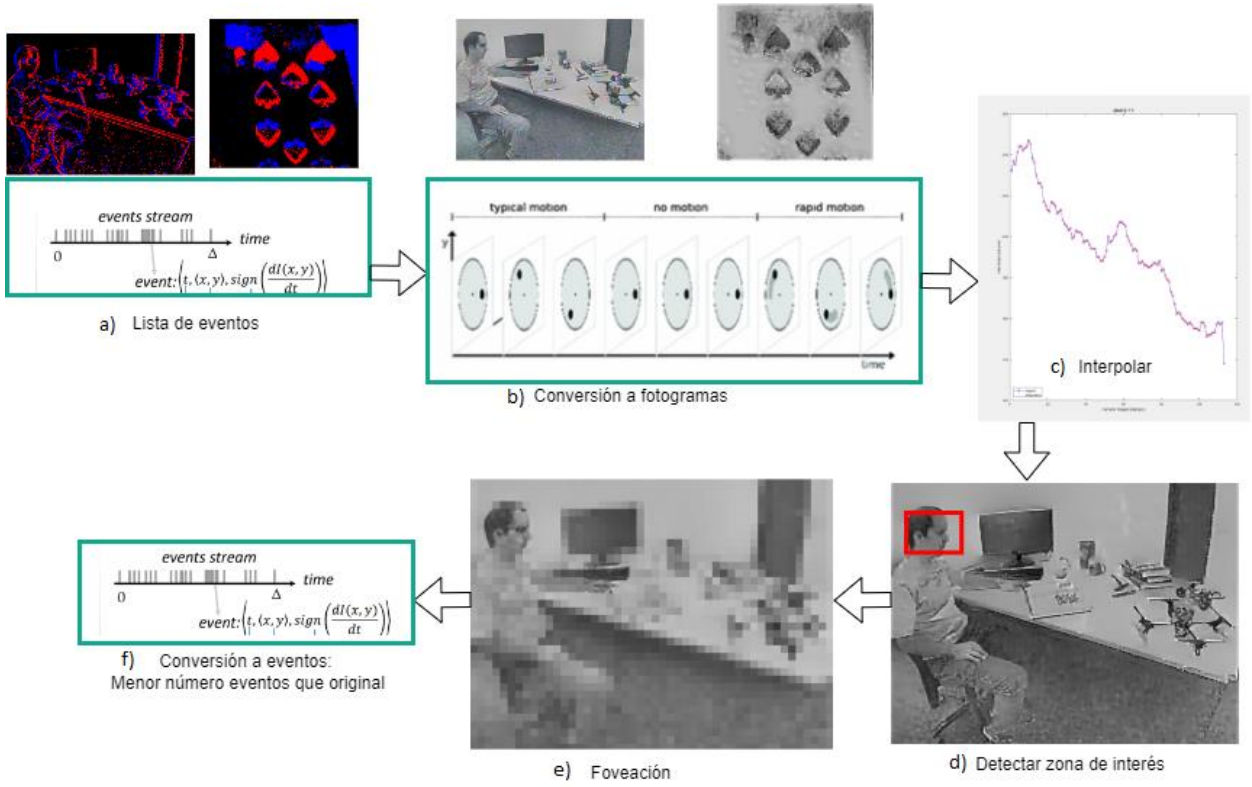


Figura 29. Objetivo del trabajo. a) Se parte de una grabación de eventos, b) se convierten a fotogramas, c) luego se interpolan las intensidades de esos fotogramas para aumentar la resolución temporal, d) se seleccionan las zonas de interés, e) se le aplica la foveación combinando pixeles en la periferia de los objetos, y f) finalmente se generan eventos con las imágenes modificadas

Como aclaración, después de la detección de la zona de interés, no se descarta la información del resto de la imagen, porque se quiere conservar toda la imagen (con más resolución en zona deseada). La zona de interés puede ir cambiando (se puede ir ampliando o reduciendo), siempre hay que mantener una visión general, aunque se focalice en una parte. Se quiere imitar el funcionamiento del ojo.

Uno podría pensar cuál es el sentido de usar los eventos si se están convirtiendo a fotogramas, aunque estrictamente no es convertir a fotogramas. Las cámaras de eventos no incluyen la foveación originalmente, y se querría incluir este mecanismo cuando los eventos ya estén generados. Hay que reconstruir la iluminancia con la que se crearon los eventos (acercándose a la realidad). Con la información de iluminancia se emula el comportamiento de la retina incluyendo el mecanismo de foveación.

En resumen, la retina lo que hará será generar eventos a partir de la intensidad lumínica, y para poder simular esta conversión de intensidad lumínica a eventos se necesita disponer de algún tipo de intensidad lumínica que se pueda convertir. Se necesita recuperar la información que "vería" la retina para poder generar los eventos correspondientes. Por ello se usa una herramienta que transforma las grabaciones de eventos en sucesiones de fotogramas que se corresponden con la intensidad lumínica original de la grabación de eventos (hace el camino inverso), pero lógicamente muestreada en el tiempo. Con la información conseguida, se va a tratar de interpolar en el tiempo para aproximarse todo lo posible a una reproducción de la intensidad lumínica real (que sería continua en el tiempo). A partir de ahí se emula la conversión que realiza la retina desde intensidad lumínica a eventos, pero con la peculiaridad de haber jugado antes con la combinación de intensidad lumínica entre píxeles, para así poder modificar la resolución espacial; de forma que se produzca el efecto de foveación que se desee según la zona de interés.

3.1 Obtención de archivo de eventos

En la Figura 30 se muestran los dos tipos de grabaciones que se utilizan, pasando una baraja de cartas (‘dynamic_6dofcartas.txt’) y moviendo la cámara en un despacho donde hay un hombre (‘dynamic_6dof.txt’). Los eventos no se comprueban cada cierto tiempo (como ocurre con fotogramas en las cámaras estándar), sólo cuando se produzcan. Estos se graban en una matriz cuando se crean, para luego representarla o procesarla. Como resultado, hay un total de 57132658 eventos en el archivo ‘dynamic_6dof.txt’ y 644672 eventos en ‘dynamic_6dofcartas.txt’; por tanto, la grabación de las cartas tiene 56487986 eventos menos.

Figura 30. Se trabajó con dos grabaciones de ventos: movimiento de cartas (izquierda) y una de un hombre en un despacho (derecha) en la que se mueve la cámara. Fuente: [71].

El programa usado para guardar grabaciones con la cámara se denomina jAER (Figura 31), “*un marco de trabajo de código abierto basado en Java que se inició alrededor de 2005 para dar soporte a los sensores de eventos*”, desarrollado por el grupo de Sensores del Instituto de Neuroinformática de Zurich. Se puede encontrar una guía de usuario y tutoriales en [72] [73] [74]. Posee muchas funciones a parte de grabar y visualizar lo que capta la cámara, por ejemplo, se le puede indicar que haga un seguimiento de objetos como se muestra en este video: [75]. En la Figura 33 se ilustra cómo es la interfaz del programa y el resultado de la grabación que se realizó con las cartas. Cuando no hay movimiento esta de color gris la ventana de representación. Los eventos se representan en negro o blanco según si cambia de claro a oscuro o de oscuro a claro, respectivamente; depende de si pasa a más o menos luminosidad cuando se mueve el objeto. En la Figura 32 se expone un ejemplo en el que se distingue claramente la forma del objeto; se mueve una mano delante de la cámara. Si quiere saber cómo instalar jAER y obtener los archivos para manipulación de eventos en Matlab y Python, consulte el Anexo A.

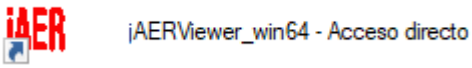


Figura 31. Icono del programa jAER.

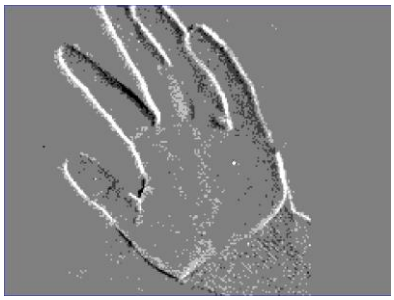


Figura 32. Ejemplo de resultado en jAER. Fuente: [72].

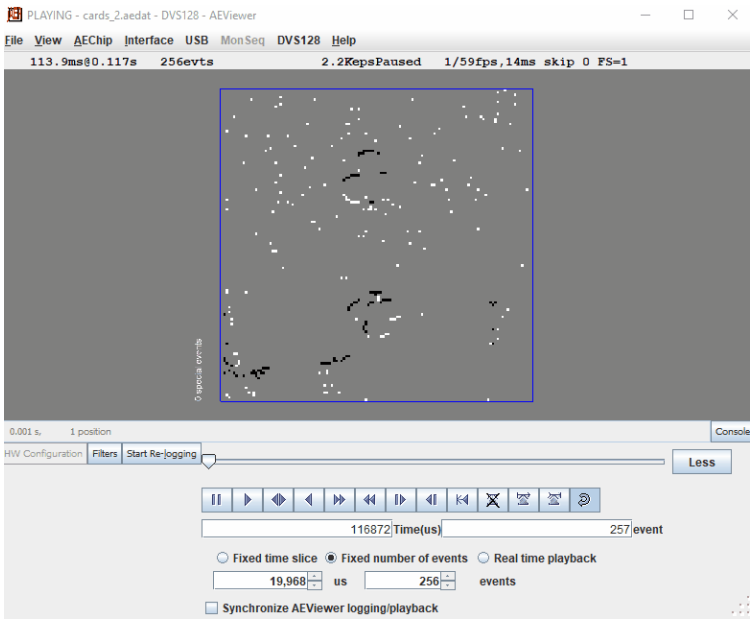


Figura 33. Interfaz del programa jAER, permite visualizar grabación de la cámara.

La grabación del hombre venía de ejemplo en un programa que se explicará más adelante; viniendo como archivo de texto (*.txt*). La grabación de las cartas se realizó en el IMSE a partir de del programa jAER, se usa con las cámaras de eventos que ellos fabrican (de 128x128 pixeles). También se puede usar en cámaras que están en el mercado, estando disponible una cámara de 1-Mega-píxel. Las grabaciones se pasan a Matlab para manipularlas, los resultados del procesamiento se pueden pasar a formato del programa jAER para su visualización.

El archivo resultante, que viene como una estructura con terminación '*.aedat'* (Figura 34), se convirtió a un archivo de texto. Esto se puede ver en el código que se encuentra en el Anexo B.1. Se realizó este cambio para que estuviese en el mismo formato que la grabación del hombre.

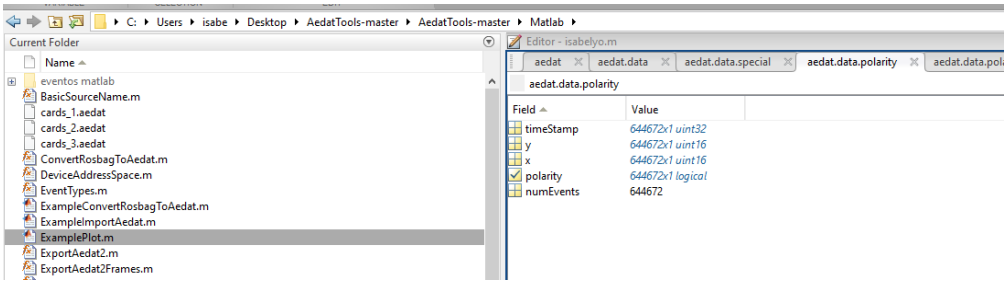


Figura 34. Dentro de la estructura *aedat* en Matlab, grabación de cartas *cards_3.aedat*.

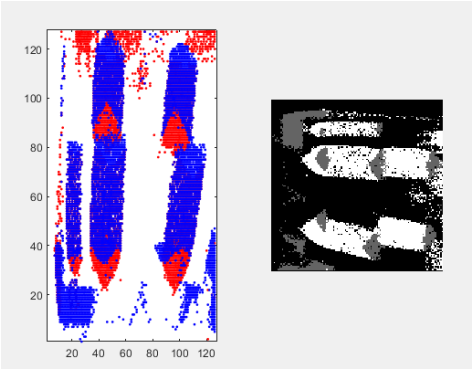


Figura 35. Visualizando eventos del movimiento de cartas, tanto la imagen de la izquierda (usando plot) como la de la derecha (usando imshow). Son reconstrucciones de fotogramas obtenidos sobre una grabación de eventos. Se utiliza un periodo de tiempo fijo en este caso.

Dado el conjunto de eventos almacenados en el archivo de texto (indicándose en la primera línea el tamaño del sensor), se realizó un programa para su representación en Matlab. Reproduce los eventos de la cámara como imágenes (frames[23]). En la Figura 35 se muestra la representación de los eventos como fotogramas. Se probó a representar usando *imshow* y usando *plot*. En el primer caso se representa los cambios de intensidad de signo positivo de color blanco y los cambios negativos en gris; cuando no hay eventos es negro. En el caso del *plot* son azul y rojo respectivamente. En este caso cuando no hay eventos es blanco. Nótese que en la Figura 35 la imagen aparece girada debido a la manera de representar con cada comando.

Una vez se ha conseguido representar los eventos; ahora lo que se quiere es reducir el número de eventos, reducir la resolución.

3.2 Convertir eventos en imágenes de escala de grises

La cámara crea eventos a partir de datos de luminancia, ahora se quiere el proceso inverso, obtener datos de luminancia a partir de eventos. Dado un conjunto de eventos de una grabación, se procedió a convertirlos en fotogramas. Para ello se utilizó un programa en Python que proporcionó la Universidad de Zurich, y que se puede consultar en [76] [77] [78] [79]. El código lee grabaciones de eventos y reconstruye una serie de fotogramas que a nosotros nos pueden servir para extraer la luminancia continua para cada píxel. Se ejecuta en un servidor de Ubuntu (Linux) desde un terminal. En principio se probó desde un ordenador con Windows 10, pero daba problemas. Es posible que la tarjeta gráfica no fuese compatible con el programa, entre otros posibles contratiempos. El código está pensado para usar Cuda, que es una plataforma para conseguir que instrucciones de un código se ejecuten por la GPU, especializada en cálculos en paralelo, pero esto sólo funciona con tarjetas Nvidia. Si la tarjeta gráfica no lo es, habría que hacer modificaciones en el código.

En la Figura 36 se muestra la arquitectura del código de Python usado; referente al artículo [79]. "*La entrada es un tensor de eventos con 5 segmentos temporales. La red consta de capas convolucionales (H, P), unidades recurrentes convolucionales (G1, G2) y bloques residuales (R1, R2). Cada capa utiliza la activación ReLU, excepto la capa final (P).*" "*Todas las capas utilizan convoluciones de 3 × 3 de una sola capa (sin muestreo descendente), excepto la capa final que es de 1 × 1. La unidad de cabeza (H) consiste en una convolución de 16 canales (y = w ∗x + b) con activación ReLU. Las unidades recurrentes convolucionales (G1, G2) consisten en una convolución de 16 canales con activación ReLU seguida de una unidad recurrente gated". "Los bloques residuales (R1, R2) utilizan convoluciones de 16 canales con activación ReLU y conexiones de salto*". *"La capa de predicción final (P) es una convolución de 1 × 1 canales. La salida es una imagen por tensor de eventos de entrada". "En escenarios difíciles, como los movimientos muy rápidos y la inicialización, FireNet[24] presenta*

[23] Frames es en inglés fotogramas
[24] Fast Image Reconstruction with an Event Camera [130].

defectos como manchas o una reconstrucción incompleta en lugares sin eventos". "El uso de una ventana de tiempo más pequeña o de un número fijo de eventos por tensor de entrada puede disminuir el emborronamiento para los movimientos rápidos" [79].

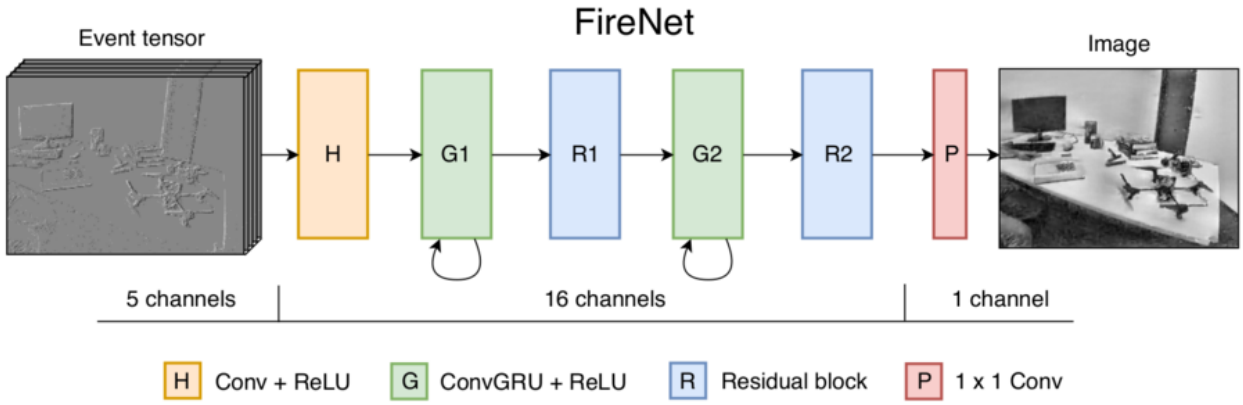


Figura 36. Arquitectura de FireNet. Fuente: [79].

Previamente a la ejecución del código se han tenido que hacer una serie de pasos. Primero se instala Anaconda. Se instalará en */home/isabel/anaconda3*, suponiendo que el usuario en Linux sea 'isabel'; se establecerá el usuario que corresponda.

```
chmod +x Anaconda3-2021.11-Linux-x86_64.sh
./Anaconda3-2021.11-Linux-x86_64.sh
```

Luego hay que agregar al *path* los ejecutables de anaconda:

```
PATH=$PATH:/home/isabel/anaconda3/bin
```

Crear el *environment* de Python:

```
conda create -n E2VID -c conda-forge python=3.8 opencv
conda activate E2VID
conda install pytorch torchvision cudatoolkit=10.0 -c pytorch
conda install pandas
```

Por alguna razón no se detectan bien las dependencias, así que se instalan algunas cosas de nuevo con *pip*:

```
pip install torch
pip install pandas
pip install scipy
```

Se monta la estructura de archivos como viene en la web y se ejecuta:

```
python run_reconstruction.py  -c firenet_1000.pth.tar -i
data/dynamic_6dof.zip --auto_hdr --display --show_events
```

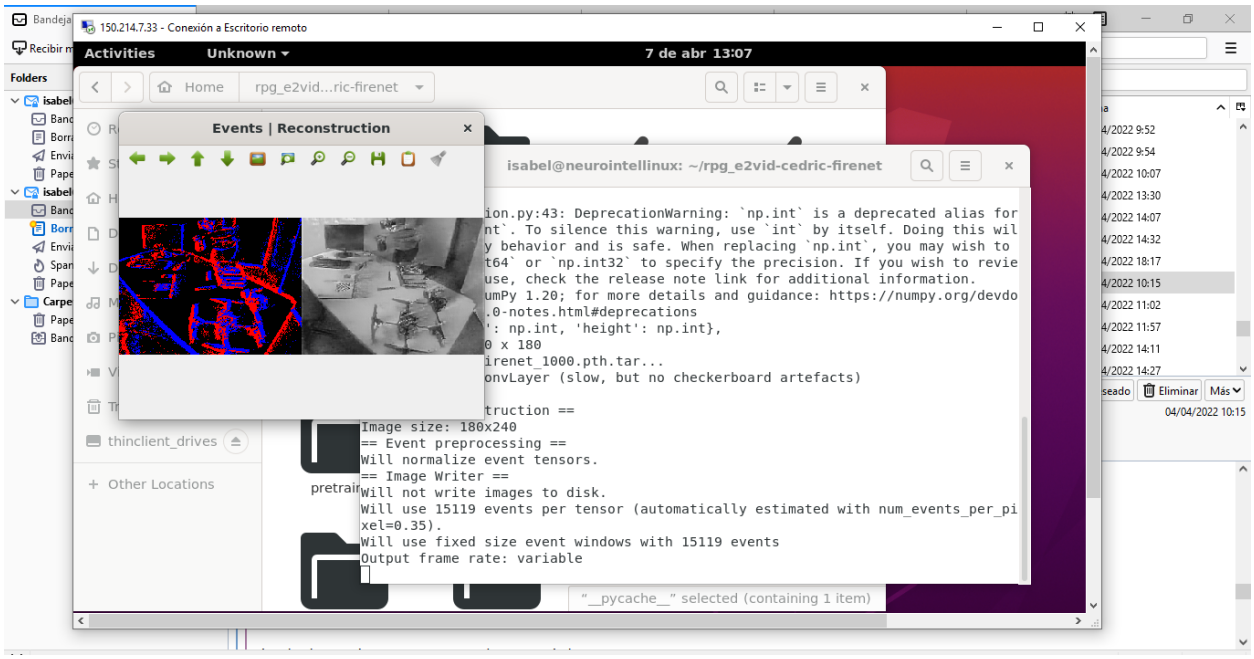


Figura 37. Muestra como resultado un video en pantalla de la reconstrucción.

Inicialmente se probó a reconstruir fotogramas para la grabación de eventos que ellos incluyen en su web (la del hombre en el despacho). En la Figura 37 se ve lo que muestra el programa al ejecutar. De lo que enseña por terminal, lo más importante sería:

```
Sensor size: 240 x 180
Loading model firenet_1000.pth.tar...
Using UpsampleConvLayer (slow, but no checkerboard artefacts)
Device: cuda:0
== Image reconstruction ==
Image size: 180x240
== Event preprocessing ==
Will normalize event tensors.
== Image Writer ==
Will not write images to disk.
Will use 15119 events per tensor (automatically estimated with num_events_per_pixel=0.35).
Will use fixed size event windows with 15119 events
Output frame rate: variable
```

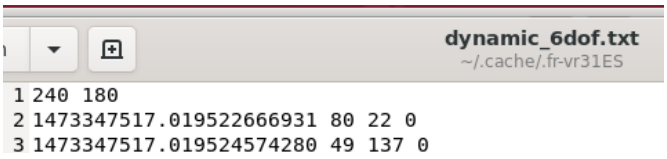


Figura 38. La primera fila del archivo *.txt* es el tamaño del sensor. En el caso de grabación del hombre es 240x180.

Una vez ejecutado el ejemplo, hay que mirar cómo son y dónde se encuentran los datos de entrada (grabación de eventos (Figura 38)) y los de salida (conversión a fotogramas (Figura 39)), para entender bien el formato de dichos datos. Una vez analizado, el siguiente paso será correr este código usando grabaciones de eventos nuestras (grabación de cartas), y para ello quizás haya que adaptar el formato.

De la manera en que se estaba ejecutando, con el comando *run* mostrado anteriormente, no se estaba guardando el resultado, solo lo mostraba. Para guardar los datos hay que indicarle una carpeta. Tener presente que hay que borrar los archivos obtenidos (o cambiarlos de carpeta) antes de volver a ejecutar, para que no se mezclen los fotogramas obtenidos con anterioridad con los nuevos. El comando sería:

```
python run_reconstruction.py  -c firenet_1000.pth.tar -i data/dynamic_6dof.zip --auto_hdr --display --show_events --output_folder /home/isabel/Documents
```

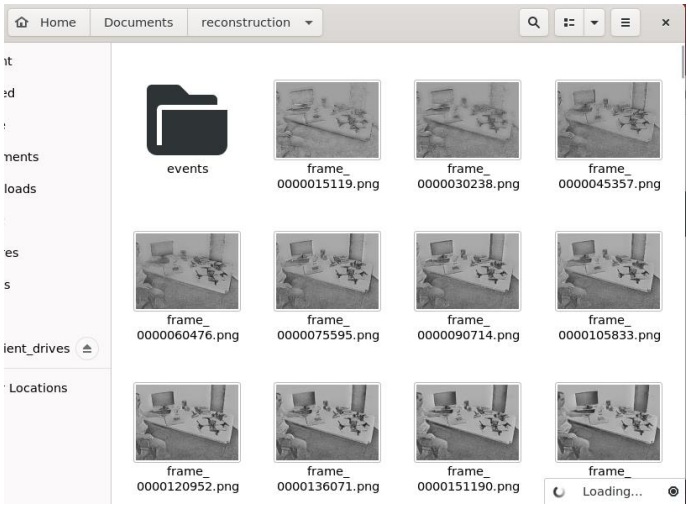


Figura 39. Resultado de salida.

Los eventos se encuentran en un archivo de texto (*.txt*), que se encuentra dentro de un archivo comprimido (*.zip*). En la carpeta *data* está el archivo con los eventos (Figura 40).

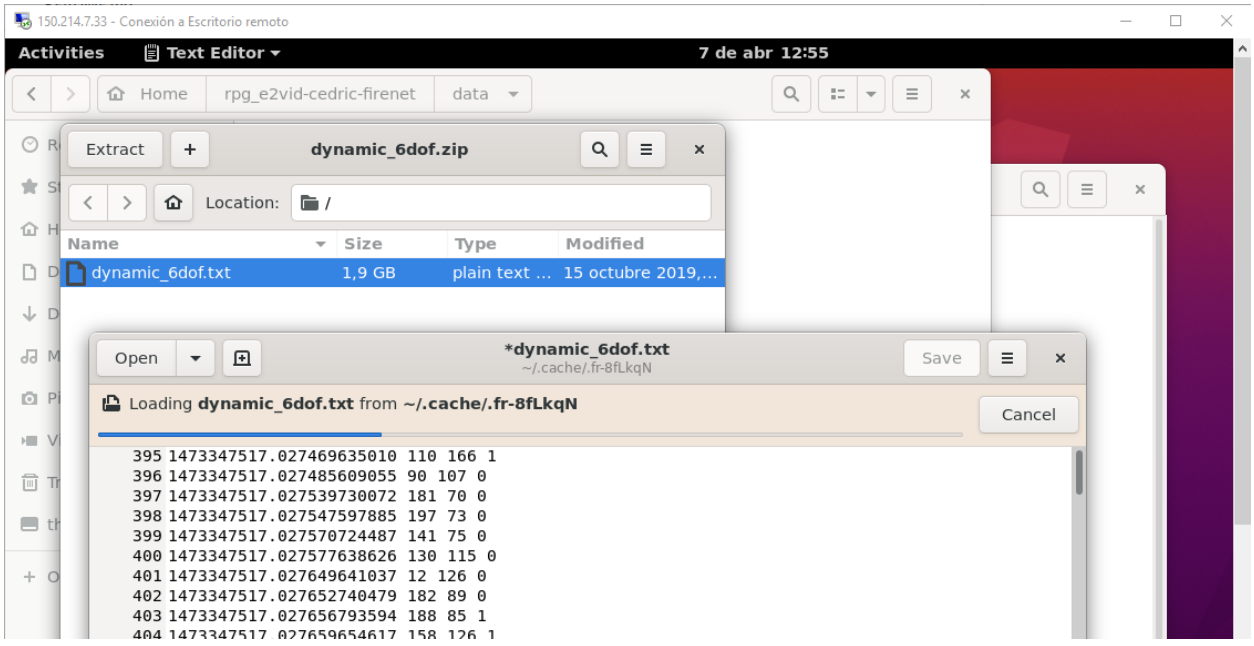


Figura 40. El archivo de eventos de la grabación del hombre (dynamic_6dof.zip).

El programa, a parte de los fotogramas, devuelve un *.txt* llamado *timestamps* (Figura 41, Figura 42) con el tiempo al que corresponde cada fotograma.

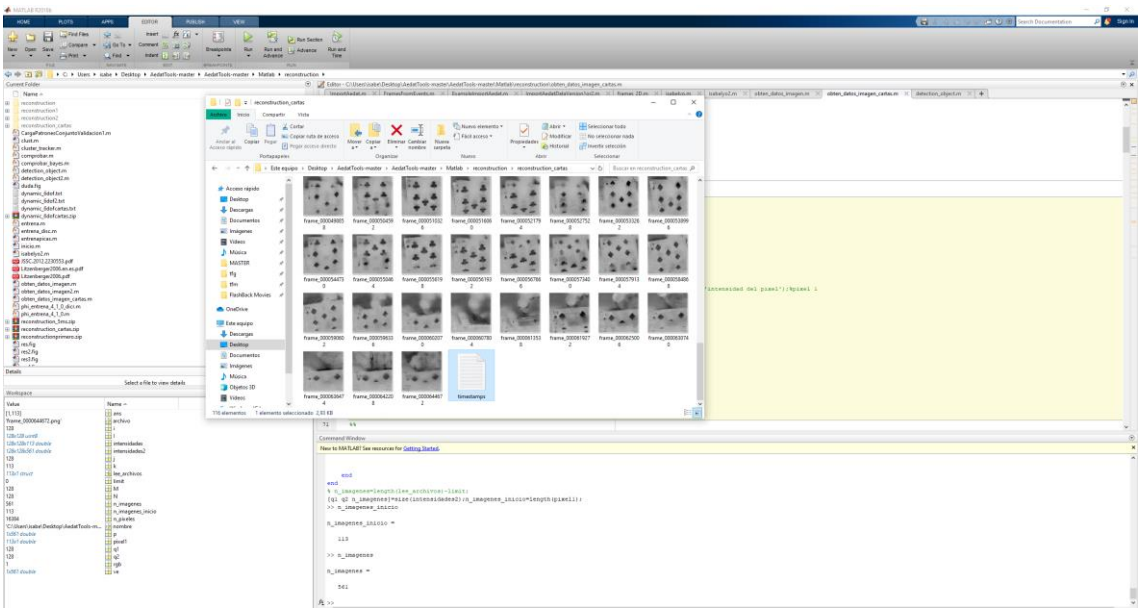


Figura 41. Archivo timestamps de los fotogramas reconstruidos (usando archivo dynamic_6dofcartas.zip).

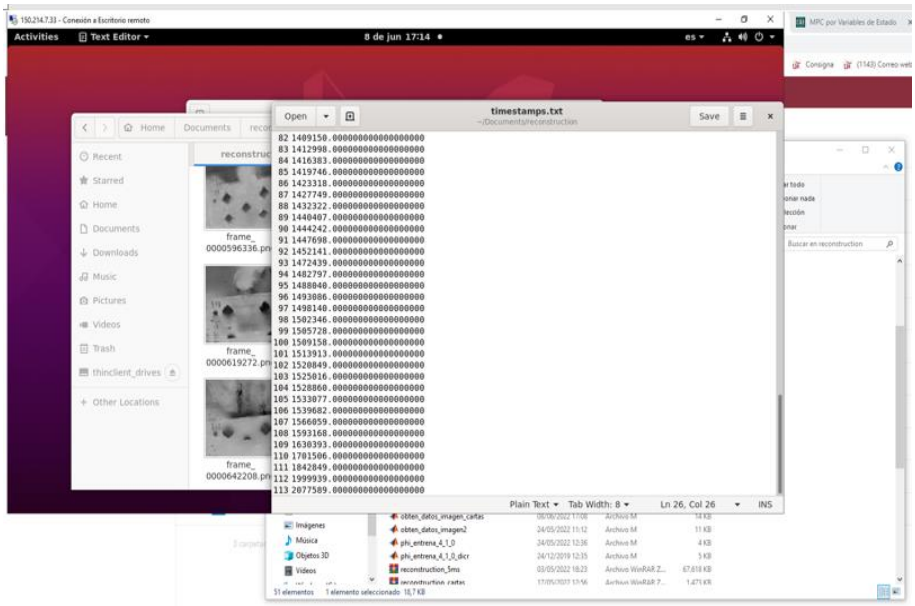


Figura 42. Archivo *timestamps* en el servidor Ubuntu de la grabación de cartas. Un total de 2 segundos aproximadamente de grabación, y pone que el ultimo evento fue en 2077589.

Se puede establecer una duración (periodo) fija entre cada fotograma o no. No es fija por defecto, en caso de que se le establezca que se quiere fija, cogerá 33ms de periodo; si no se le indica ningún número concreto.

Cuando se le ponía que el periodo entre cada par de fotograma fuese variable, se mostraba por terminal que había 15119 eventos por frame en la grabación del hombre. Se probó a hallar en Matlab el tiempo que transcurría entre cada dos frames en ese caso, obteniéndose lo que se muestra en la Figura 45.

La Figura 43 muestra lo que se obtiene por el terminal cuando se establece periodo fijo; no muestra cuántos eventos hay por imagen. Para establecer el tiempo de frame a 33ms:

```
python run_reconstruction.py  -c firenet_1000.pth.tar -i data/dynamic_6dof.zip --auto_hdr --display --show_events --output_folder /home/isabel/Documents --fixed_duration
```

Surgieron problemas respecto al tema de los tiempos. La grabación de las cartas se sabe que fue de 2 segundos y que los tiempos están en microsegundos, sin embargo, en la documentación del programa no viene especificado en qué unidad de tiempo deben estar los eventos. Se puede deducir a partir del *timestamps* la unidad de tiempo si se establece que el periodo son 33ms y se restan dos tiempos consecutivos. Cuando se le pone periodo fijo de 33ms a la de las cartas, no llega a funcionar bien, apareciendo las imágenes prácticamente en negro. Se comprobó restando tiempos consecutivos que no daba el valor de 0.033, como ocurría en la grabación del hombre (Figura 44). Se podría suponer que los tiempos de los eventos de la grabación del hombre están en segundos. Sin embargo, no parece lógico la gran cantidad de segundos que tendría la duración de la grabación, puede que la documentación del código presente una errata o no se está interpretando correctamente. Solo funciona bien la de las cartas si no se establece periodo fijo, ya que, diríamos que lo está considerando con un valor de unidad de tiempo distinto a los microsegundos. Hay que trabajar teniendo esto en cuenta. No obstante, no afecta mucho, ya que se han ajustado los parámetros de forma empírica para conseguir que las marcas temporales de los eventos finales (que se obtienen al final de todo el proceso) sean coherentes con los eventos originales.

En general, funciona mejor el programa cuando se crean las imágenes por número de eventos en lugar de por un periodo de tiempo fijo.

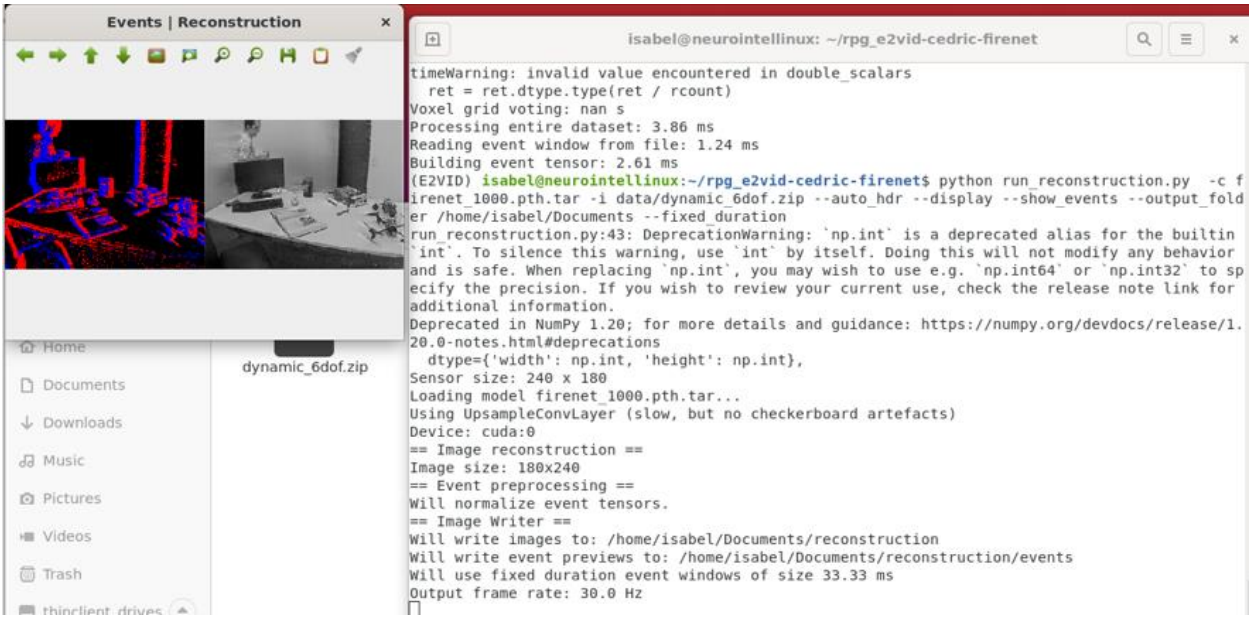


Figura 43. Cuando se fija el tiempo de periodo fijo, en terminal muestra que son 33.33ms por defecto. Cuando se pone fijo no muestra por terminal cuántos eventos hay por imagen.

```
>> 1473347517.052893638610839844-1473347517.086224794387817383

ans =

   -0.0333
```

Figura 44. Comprobación de que el periodo son 33ms dado el tiempo de cada fotograma en grabación del hombre.

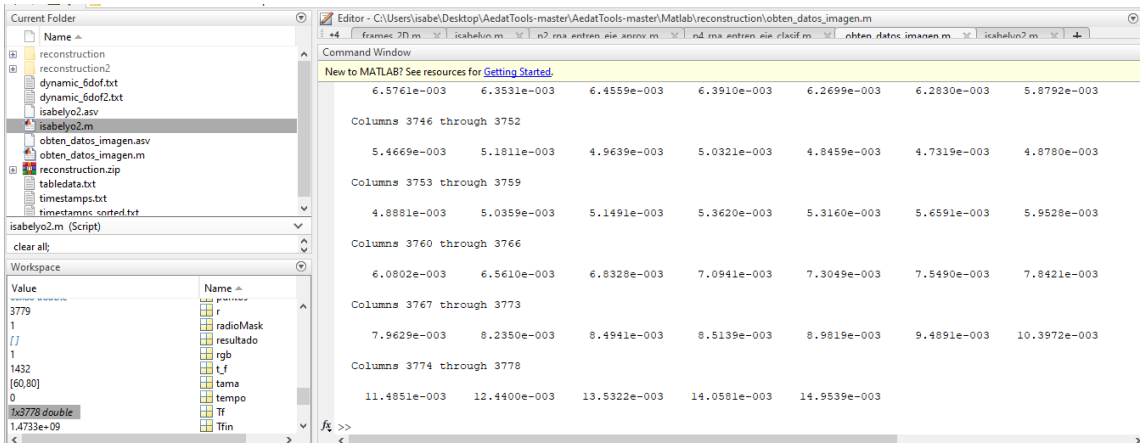


Figura 45. Cuando no se le ponía periodo fijo entre cada fotograma, se mostraba por terminal que había 15119 eventos por frame en la grabación del hombre. Se probó a hallar en Matlab el tiempo que transcurría entre cada dos frames cuando no es fijo.

En la web (en la que se encuentra el programa) se explican los diferentes parámetros del código que se pueden modificar (estableciéndolos al ejecutar en el terminal) y cuáles son sus valores por defecto. Los más destacables son:

--window_size/ -N(predeterminado: Ninguno) Número de eventos por ventana. Este es el parámetro que más influye en la calidad de reconstrucción de la imagen. Si se establece en Ninguno, este número se calculará automáticamente en función del tamaño del sensor, como N = ancho * alto * num_events_per_pixel. Ignorado si --fixed_duration está establecido.

--num_events_per_pixel(predeterminado: 0,35): Parámetro utilizado para estimar automáticamente el tamaño de la ventana en función del tamaño del sensor. Se eligió el valor de 0,35 para corresponder a ~ 15 000 eventos en un sensor de 240x180 como el DAVIS240C. El número de eventos por píxel es un parámetro que la herramienta usa para definir la duración de la ventana temporal. Tener en cuenta que cuando se convierte una lista de eventos a fotogramas, lo que hacemos es agrupar una determinada cantidad de eventos producidos durante un tiempo Tframe, y a partir del número de eventos que aparecen en este tiempo para cada píxel calculamos la escala de gris de dicho píxel. Así, si tomamos un Tframe demasiado corto, tendremos pocos eventos y el fotograma reconstruido mostrará una información incompleta (no podremos observar las formas completas de los objetos). Sin embargo, si tomamos un Tframe demasiado largo, tendremos demasiados eventos y el fotograma reconstruido mostrará cada objeto móvil en distintas posiciones a la vez, lo cual tampoco es deseable. Algo así es lo que se observa en las capturas que me muestras con distinto valor de Tframe en el apartado 4.2.

--fixed_duration(predeterminado: Falso) Si es Verdadero, utilizará ventanas de eventos con una duración fija (es decir, una velocidad de fotogramas de salida fija).

--window_duration/ -T(predeterminado: 33 ms) Duración de cada ventana de evento, en milisegundos. Es posible que su valor deba adaptarse a la dinámica de la escena. Ignorado si --fixed_duration no está establecido.

Parámetros de salida:

--output_folder: ruta de la carpeta de salida. Si no se establece, las reconstrucciones de imágenes no se guardarán en el disco.

--dataset_name: nombre del directorio de la carpeta de salida (predeterminado: 'reconstrucción').

3.3 Interpolación

Se quiere amentar la resolución temporal y reducir la espacial. Una posible forma de reducir el tiempo de frame en Matlab es interpolando; para aumentar la resolución temporal se crean nuevas imágenes interpolando todas. La interpolación consiste en deducir puntos entre dos puntos conocidos, de forma que se puedan unir aproximándolo a una función, como se muestra en la Figura 46.

En otras palabras, una vez se ha obtenido la sucesión de frames de Python, se trata de introducir nuevos frames intermedios mediante interpolación, permitiendo aumentar la resolución temporal. Matlab tiene funciones que lo hacen directamente. Simplemente, para cada píxel, hay que tomar la evolución de su nivel de iluminancia a partir de los fotogramas existentes, y tratar de introducir un mayor número de muestras. Se pueden aplicar diferentes tipos de interpolación (lineal, cubica, pchip, splines …). Por ejemplo, en la Figura 47 se ve una interpolación lineal y una por splines. En este caso se utilizó la lineal. Puede consultar más información sobre la interpolación en Matlab en [80] [81] [82].

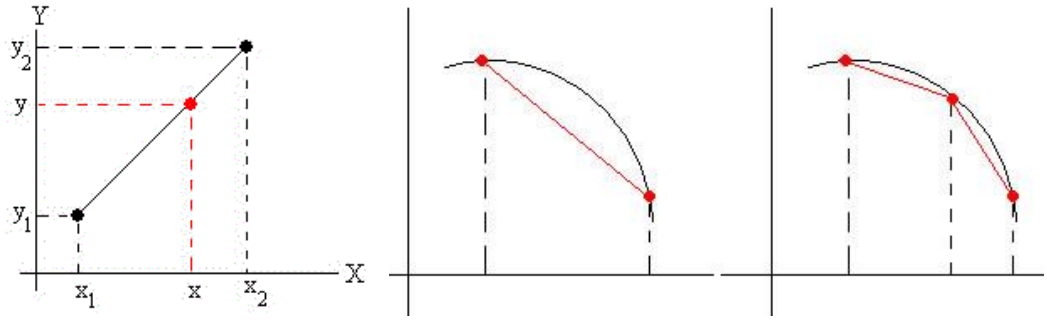


Figura 46. Se muestra la aproximación lineal (en rojo) de una función (en negro). Al usar este tipo de interpolación hay que añadir suficientes puntos intermedios para no tener una mala aproximación en algunos casos. Fuente: [83].

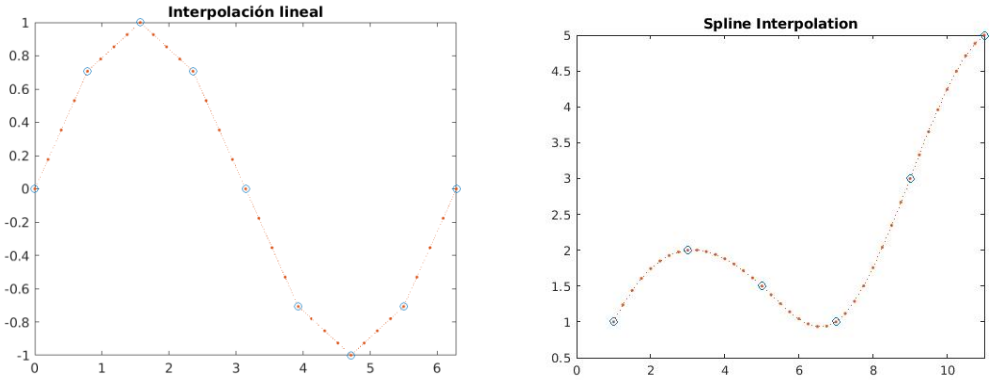


Figura 47. Hay diferentes maneras de interpolar. Por ejemplo, a la izquierda se ve una interpolación lineal y a la derecha una por splines. Fuente: [81].

No solo se hará interpolación de las intensidades de los pixeles, sino que también de los tiempos de frame que estaban almacenados.

3.4 Detección zona de interés

Inicialmente se probó a reducir la resolución de toda la imagen que se está estudiando en cada momento, pero al final el objetivo será mantener la resolución alta en una zona de la imagen (donde haya un objeto, por ejemplo) y reducir la resolución en el resto de la imagen. Para ello, se puede tratar de pensar en algún algoritmo que

detecte objetos en movimiento y nos proporcione sus coordenadas. Usando esas coordenadas, se puede decidir en cada momento qué área de la imagen es el área de interés y reducir la resolución en el resto de la imagen.

Para cada grabación se aplicó un método distinto de determinación de zona de interés. No obstante, ambos casos son procedimientos de detección aplicados a imágenes en escala de grises, no a eventos. Se podría haber trabajado en el ámbito de los eventos en lugar de las intensidades; lo cual resultaría de mayor interés de simular, ya que la retina trabaja con los eventos. Existen métodos para realizar la detección objetos con eventos. Esto es una línea de trabajo futura que se mencionará en el apartado 6.2.1.

El caso de detección de objetos en imágenes ha sido estudiado por muchos investigadores. En general, la mayoría se basa en el Machine Learning, puede encontrar bastante información sobre el concepto en el documento [84] y en sus referencias respectivas. Existe un gran abanico de posibilidades a la hora de realizar algoritmos de detección y seguimiento de objetos. Se recomiendan examinar las siguientes referencias: [85] [86] [87] [88].

3.4.1 Grabación cartas

En este caso se quería que aumentase la resolución de las figuras de las cartas (picas, rombos, corazones, tréboles). Para ello se determinó entre qué valores de intensidad de gris se consideraría zona de interés. Para esto se puede usar la herramienta *imtool* o *imshow* en Matlab.

Se binariza cada imagen, detectando los objetos con los umbrales especificados. Luego se le aplica un procesamiento morfológico para eliminar pixeles sueltos y rellenar los huecos dentro de los objetos. Una vez realizado esto se sacan características, concretamente se determina el número de objetos y el boundingbox[25] (Figura 48) de cada uno.

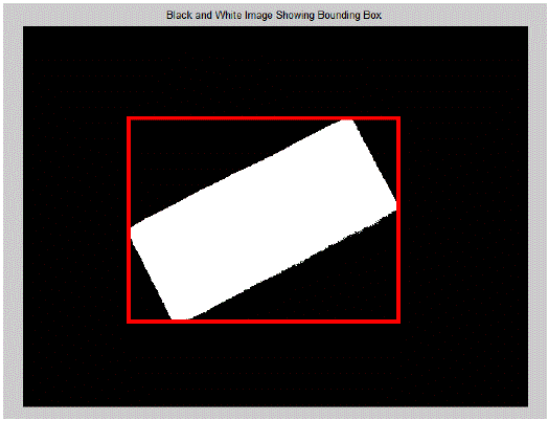


Figura 48. Ejemplo BoundingBox. Fuente: [89].

Si se detecta el objeto de interés, se redondea las coordenadas obtenidas (como se trabaja con coordenadas de píxel se necesitan valores enteros) y se establece que se mantenga la resolución alta en los límites marcados por el boundingbox. Siempre verificando que los limites no superen los bordes de la imagen, pues se ha redondeado los valores.

[25] El rectángulo más pequeño que envuelve a cada objeto.

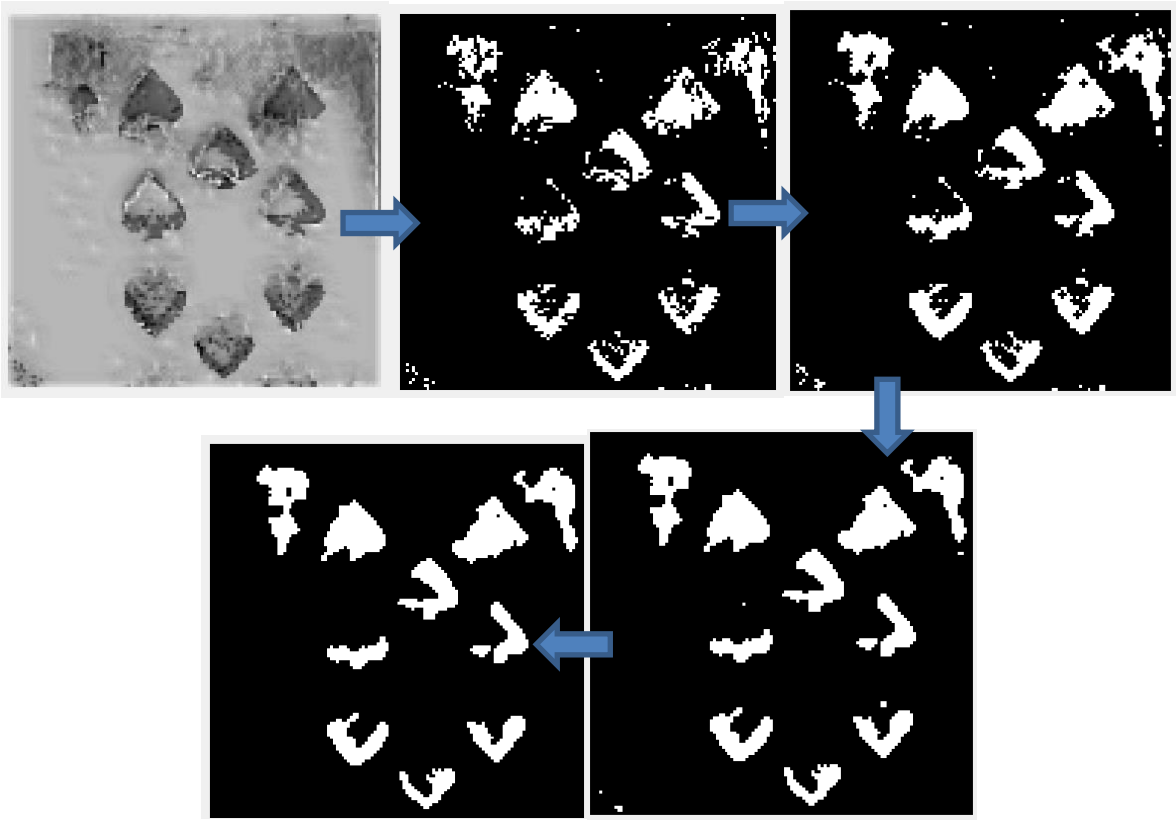


Figura 49. Proceso binarización y conversiones morfológicas (imopen, imclose …). La inicial es la imagen de arriba a la izquierda.

El principal inconveniente que se tuvo en el procesamiento fue la baja calidad de la imagen y que no fuera en color; esto dificulta la detección de objetos. Como se ve en Figura 49, aparte de la figura de la carta (las picas), también se consideraba objeto algunas sombras que había en los bordes. Por otro lado, los objetos conseguidos en la imagen binaria (las picas) aparecen recortados.

Si se quisiese detectar un objeto muy concretos, por ejemplo, que solo considere las picas y no los corazones, tréboles o rombos; se realizaría un proceso típico de aprendizaje automático (Machine Learning). Una vez obtenida la plantilla binaria, se obtienen características de los objetos. El procedimiento de binarización y obtención de características se realizaría con varias imágenes en las que solo aparezca el objeto de interés en varias posiciones y tamaños, serían las muestras. Una vez obtenidas, se puede realizar el entrenamiento de un clasificador con el conjunto de muestras de entrenamiento. Una vez realizado el entrenamiento, se prueba el clasificador en un conjunto de imágenes distinto, en el que pueden aparecer otros objetos. Se realizaría otra vez el proceso de obtención de características y se procedería con la clasificación. Aunque no se ha realizado en este caso, sería una línea futura. No obstante, no es el objetivo del proyecto y es fácil de encontrar ejemplos de clasificación de objetos en imágenes, ya que ha sido bastante estudiado en los procesos de percepción en robótica con cámaras estándar.

3.4.2 Grabación hombre

También se utiliza el boundingbox, pero se parte de una función que viene en Matlab instalada, la cual contiene un algoritmo de detección de ojo derecho de la cara. Siendo el rectángulo un poco más grande de los límites marcados por el objeto (Figura 50). El algoritmo a veces falla, detectando como ojo cosas que no lo son (Figura 51).

En resumen, se establecerá que aumente la resolución en el rostro del hombre en esta grabación.

Figura 50. Detecta ojo derecho.

Figura 51. Hay veces que detecta como ojo cosas que no lo son. Se muestra con un cuadrado el objeto que considera.

3.5 Cambio de resolución (foveación)

Se establece una reducción de la resolución en la imagen completa y se establece que se mantenga la resolución original en las zonas indicadas por los boundingbox, en cada imagen correspondiente. Para la reducción de resolución se realiza la media de las intensidades de un conjunto de pixeles (Figura 52) y luego se le atribuye el valor de la media a los pixeles con las que se calculó. En otras palabras, se recorre cada imagen convolucionándola con una plantilla (cuadrada) de la media. Se hace media de las intensidades de los pixeles que estén dentro de los límites de la plantilla; se asigna a esos pixeles el nuevo valor de intensidad calculado (el obtenido al hacer la media), estableciendo un conjunto de píxeles como si fuera uno solo; y se desplaza la plantilla de forma que no realice media con los píxeles que ya se procesaron, es decir, se desplace su centro 2*Radio_plantilla+1 hacia la derecha o hacia abajo, según corresponda. Cuando se desplaza la plantilla, se tiene presente que no se puede procesar pixeles fuera de la imagen. Por esta razón, quedará un borde a la derecha y abajo sin procesar, si no se hace nada más. Estos bordes se le aplica un procesamiento de la media cogiendo un numero de pixeles diferente al resto de casos, un tamaño de plantilla diferente a la principal. Por ejemplo, se recorre la primera fila de la imagen, el hueco que queda entre el borde de la imagen y el ultimo píxel procesado, se hace media de pixeles comprendidos entre estos límites de derecha a izquierda; pero de arriba abajo se tiene en cuenta el tamaño de la plantilla principal (si no hay ningún problema de bordes, de sobrepasar los límites). Sucede lo mismo en el borde de abajo, pero siendo arriba abajo lo que varía y de izquierda a derecha lo que permanece como en la plantilla. Cuando se encuentra en la esquina inferior derecha no se tiene en cuenta el tamaño de plantilla en ninguno de los dos casos, ni horizontal ni verticalmente.

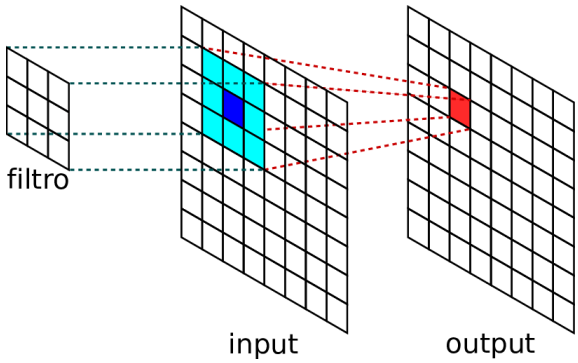


Figura 52. Cuando realizamos lo de la plantilla de la media, es un proceso de convolución de imágenes. Fuente: [90].

3.6 Convertir las imágenes modificadas a eventos

Una vez aplicado el proceso de foveación se procedió a convertir nuevamente las imágenes en eventos. Se irán recorriendo todas las imágenes. Consiste en ir comparando las intensidades de la imagen actual con las anteriores, viendo cuando supera un determinado umbral. Se tendrá una matriz (*Io_indice*) del tamaño de la imagen y cada componente de ésta indica con qué imagen (intensidad) hay que comparar cada píxel de la imagen actual. Inicialmente esta matriz estará rellena de unos, lo que indica que inicialmente se tienen que tomar como referencia las intensidades de la imagen primera. Cuando se coge una nueva imagen, en cada posición de píxel se hace el cálculo:

$$\frac{|Intensidad_{actual(i,j)} - Intensidad_{Ioindice(i,j)}|}{Intensidad_{Ioindice(i,j)}} \geq umbral \qquad (3.1)$$

Cuando se cumple la condición se considera que se produce un evento en esas coordenadas de píxel. La componente de la matriz *Io_indice* se actualiza en esa posición, indicando que para esas coordenadas se tiene que comparar con la imagen que acaba de provocar que se produzca evento (el resto de los componentes de las matriz puede que tengan asignado que tienen que comparar con una imagen distinta a esa). El umbral indica cuándo la diferencia de intensidad normalizada se considera evento. Se divide por el anterior para normalizar la medida; es relativa para que no dependa de si se está en zona muy luminosa u oscura. Hay que tener en cuenta que ese umbral sería en realidad corriente eléctrica. Se integra tensión que se compara con una referencia y cuando se produce el evento restaura su valor.

Cuando se produce un evento se comprueba si el cambio de luminosidad es de signo positivo o negativo (signo 1 o 0). Después se guarda en una línea de un archivo de texto, poniendo el tiempo al que corresponde, la coordenada de fila, la de columna y el signo. Con respecto al tiempo, se pone el que se guardó previamente con cada imagen (teniendo en cuanta que hay valores interpolados).

La primera línea del archivo de texto indicará el tamaño de la imagen, es decir, el tamaño del sensor. Una vez terminado el archivo se comprueba el resultado utilizando el código que se usó anteriormente para visualizar eventos en Matlab.

Para comprender mejor el proceso de obtención de eventos, es recomendable leer el artículo: [91]. En la sección II, donde explica cómo se genera un evento de salida cada vez que la luz detectada por un píxel experimenta un cambio relativo de θ_{ev} (la θ_{ev} representa el mínimo contraste temporal detectable), dado por el logaritmo neperiano de la corriente *Iph* en el instante *t2* entre la corriente *Iph* en el instante *t1*. Esto está ilustrado en la Figura 53 (el eje vertical representa el logaritmo de la corriente *Iph*). En realidad, como se ve en la expresión

(3.2), este cambio se puede aproximar por el cambio relativo Δ*Iph/Iph;* esta aproximación es la usada en este trabajo.

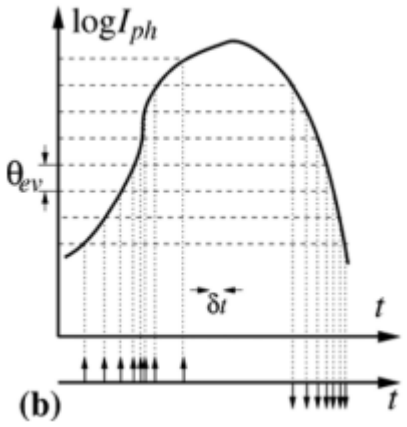


Figura 53. "*Generación de eventos asíncronos basados en datos*". Fuente: [91].

$$\theta_{ev} = \left| \ln\left(\frac{I_{ph}(t_2)}{I_{ph}(t_1)} \right) \right| = \left| \ln\left(1 + \frac{I_{ph}(t_2) - I_{ph}(t_1)}{I_{ph}(t_1)} \right) \right| \approx \left| \frac{\Delta I_{ph}}{I_{ph}} \right| \qquad (3.2)$$

¿Qué es lo quiere decir todo esto? Si para un píxel se toma un vector que representa el valor de luz capturado por ese píxel a lo largo del tiempo (fotograma a fotograma), eso se corresponde con la curva de la Figura 53. Entonces, si a partir del valor inicial de esa curva *I0*, se busca el instante en el que la curva alcanza un valor *(I0+*Δ*Iph)/I0*, en ese instante se generará un evento. Además, en ese instante (llamémoslo instante 1), uno se fija en el valor *I1*, y a partir de ahí se busca el instante en que la curva alcanza un valor *(I1+*Δ*Iph)/I1*. Y así sucesivamente.

La mejor manera de entenderlo es mostrando como ejemplo uno de los resultados obtenidos con las cartas, el cual se encuentra en apartado 4.5.

3.7 Conclusiones

La finalidad última será diseñar una retina de eventos capaz de modificar su resolución por zonas de forma dinámica, y para ello se tiene que estudiar este mecanismo a nivel de simulación. El proyecto consiste en realizar esa simulación, para determinar los resultados que se obtendrían según los parámetros establecidos. La retina lo que hará será generar eventos a partir de la intensidad lumínica, y para poder simular esta conversión de intensidad lumínica a eventos se necesita disponer de algún tipo de intensidad lumínica que se pueda convertir. Por ello se usa un programa que transforma las grabaciones de eventos en sucesiones de fotogramas que se corresponden con la intensidad lumínica original de la grabación de eventos, muestreada en el tiempo. A partir de esta información, se interpola en el tiempo para aproximarse a una reproducción de la intensidad lumínica real. A partir de ahí se emula la conversión que realiza la retina desde intensidad lumínica a eventos, pero con la peculiaridad de haber aplicado el mecanismo de foveación según la zona deseada, es decir, reduciendo la resolución en la parte que no es de interés.

En este capítulo se ha hecho una explicación de los diferentes pasos realizados para obtener el modelo de la retina foveada:

1. Se obtiene la grabación de eventos usando la cámara y jAER. Se van a utilizar dos grabaciones: una moviendo unas cartas y otra de un hombre en una oficina. Los eventos no se determinan cada cierto tiempo (como ocurre con los fotogramas), sólo cuando se produzcan. Dado el conjunto de eventos almacenados en el

archivo de texto se realizó un programa para su representación en Matlab; permitiendo comprobar si se distinguen los objetos de la grabación. Se muestra una representación gráfica de los cambios de luminosidad en determinados píxeles, mediante la acumulación de eventos en un intervalo de tiempo. Se está teniendo en cuenta la polaridad, poniendo en un color los eventos positivos y en otro los negativos. El código se encuentra en el Anexo B.1.

2. Se utiliza un programa en Python (proporcionado por la Universidad de Zurich). Lee grabaciones de eventos y reconstruye una serie de fotogramas que servirán para extraer la luminancia continua de cada píxel. Convertir eventos en imágenes en escala de grises; viéndose, aproximadamente, lo que se obtendría con una cámara estándar. En general, funciona mejor el programa cuando se crean las imágenes por número de eventos en lugar de por un periodo de tiempo fijo. El programa, a parte de los fotogramas, devuelve un *.txt* llamado *timestamps* con el tiempo de muestreo al que corresponde cada fotograma.

3. Se introducen nuevos fotogramas intermedios a partir de los fotogramas existentes. Se realiza la interpolación de las intensidades y marcas temporales de los fotogramas resultantes, permitiendo aumentar la resolución temporal. La interpolación consiste en deducir puntos entre dos puntos conocidos, de forma que se puedan unir aproximándolo a una función. El código se encuentra en el Anexo B.2.

4. Utilizando algoritmos que permiten la detección de objetos concretos; se determina la zona o zonas de interés. En otras palabras, se determina qué zonas mantendrán la resolución original y qué zonas se le reducirá. El principal inconveniente que se tuvo en el procesamiento fue la baja calidad de las imágenes reconstruidas y que no estuviesen en color. Además, estos tipos de algoritmos a veces fallan. El código se encuentra en el Anexo B.3.

5. Se realiza el mecanismo de foveación, es decir, se reduce resolución espacial en las zonas no deseadas, sin perder la capacidad de reconocimiento visual. Para este proceso, se recorre toda la imagen haciendo una convolución con una plantilla, la cual hace la media de intensidades de los píxeles según la región que abarque. Se tiene en cuenta que, si la plantilla sobresale de la imagen, se le aplicará una de tamaño distinto. Para que la imagen resultante sea del mismo tamaño que la original, se le aplicará el valor obtenido con la plantilla a toda la vecindad del píxel correspondiente. La vecindad viene determinada por el tamaño de la plantilla. Se están agrupando píxeles en la periferia como si fueran uno solo. Una vez realizado esto, se reestablecerá la resolución original en las zonas donde se determine que está el objeto de interés. El código se encuentra en el Anexo B.4.

6. Una vez obtenidas las imágenes con la foveación aplicada, se imita el proceso que realiza la cámara para determinar los eventos (tanto de polaridad positiva como negativa). Consiste en ir comparando las intensidades de la imagen en un instante actual con las anteriores, viendo cuando supera un determinado umbral. Se tiene una matriz (*Io_indice*), del tamaño de la imagen, en la que cada componente de ésta indica con qué imagen hay que comparar cada píxel de la imagen actual. El número de eventos dependerá en gran medida del umbral que se establezca. En cada posición de píxel se comprueba si se cumple la condición (3.1). Se divide por el anterior para normalizar la medida; es relativa para que no dependa de si se está en zona muy luminosa u oscura. Cuando se cumple la condición se genera un evento y se actualiza la matriz *Io_indice*. Con respecto al tiempo asignado al evento, se pone el que se guardó previamente con cada imagen (teniendo en cuanta que hay valores interpolados). El código se encuentra en el Anexo B.5.

4 RESULTADOS EXPERIMENTALES

"La vida, la naturaleza, la humanidad solo son bellas palabras cuando son transformadas por un cerebro creador."
Edmond Jaloux

Se mostrarán los resultados obtenidos en los pasos explicados en el capítulo anterior para modelar el mecanismo de foveación. En lo que respecta a la parte de obtención de los eventos iniciales y a la de detección de las zonas de interés, ya se mostraron anteriormente, porque complementaban la explicación teórica.

4.1 Resultados experimentales de representar los eventos iniciales (sin foveación)

Se expondrán los resultados al hacer la representación de las grabaciones de eventos en Matlab (Figura 35). Los eventos necesitan ser representados para corroborar el correcto funcionamiento del sensor de visión y para que nosotros podamos visualizar lo que sucede en la escena (los eventos son vectores de 4 componentes). Los resultados de la representación de eventos con la foveación aplicada se han incluido en el apartado 4.5.

Se va a realizar la representación con el periodo de tiempo fijo. El principal parámetro es el tiempo de fotograma (frame). Se determinó inicialmente un periodo entre frames y se ajustó con prueba y error. La unidad de tiempo depende de la unidad en que esté la de los eventos del archivo. El número de frames se determina redondeando la división del tiempo de la grabación entre el periodo de frame (TFrame).

Usando el archivo de las cartas sin foveación con Tframe = 10000, hay 208 fotogramas y se representan un total de 644672 (el número de eventos que hay en el archivo de texto). Como se ve en la Figura 58, se comprobó cuantos eventos se dan en cada fotograma. En la Figura 54 se muestra un histograma de las veces que se produce una cantidad de eventos en un fotograma, es decir, se representa el número de veces que muestra una determinada cantidad de eventos en un frame. Cuando se usa Tframe = 6000, hay 346 fotogramas y se muestran 644639 eventos en total (parece ser que hay eventos que no se han tenido en cuenta en este caso, deberían de salir siempre igual), véase Figura 55 y Figura 59. Si Tframe = 800 se visualiza muy lento, hay 2597 fotogramas y se muestran 644672 eventos en total, véase Figura 56 y Figura 60. Si Tframe = 100000 no se visualiza bien, hay 21 fotogramas y se muestran 644672 eventos en total, véase Figura 57 y Figura 61.

A medida que disminuya el valor de Tframe, la visualización será más lenta, ya que habrá mayor número de fotogramas. Además, en los histogramas y las gráficas se deduce que aumenta el número de eventos en un frame a medida que aumenta el periodo.

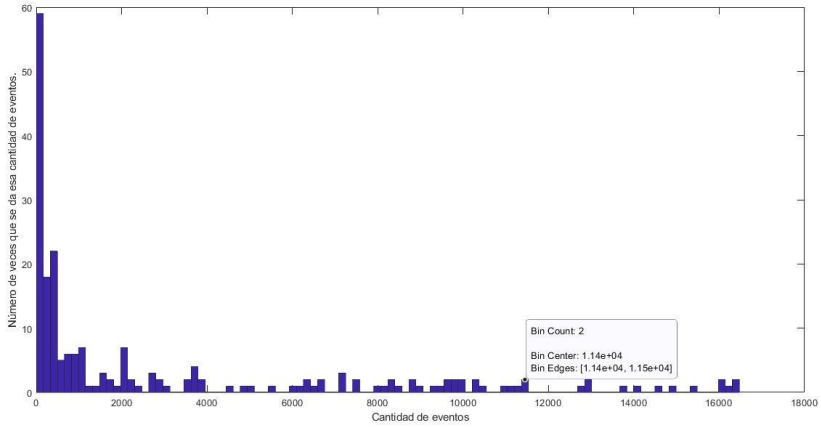


Figura 54. Número de veces que se da una cantidad de eventos en un fotograma con Tframe = 10000. En Matlab se puso **hist(numero_eventos_en_el_frame,100)** para su representación. Para saber el número de eventos total que hay, se pone numero**_total_eventos=sum(numero_eventos_en_el_frame(:))**.

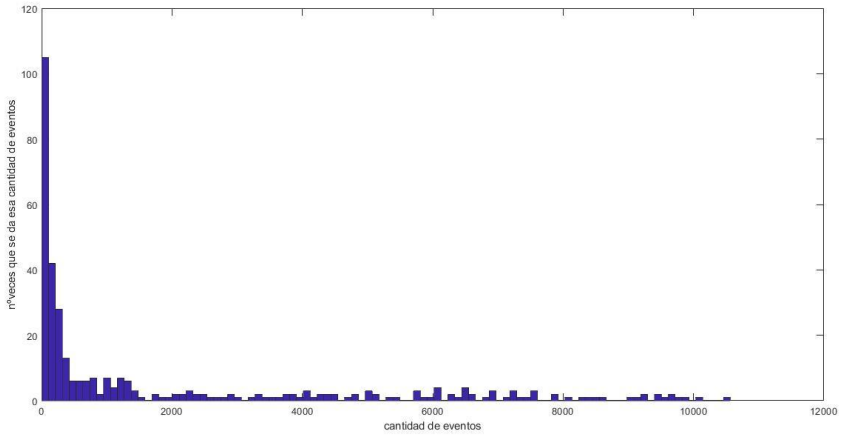


Figura 55. Histograma. Número de veces que se da una cantidad de eventos en un fotograma con Tframe = 6000.

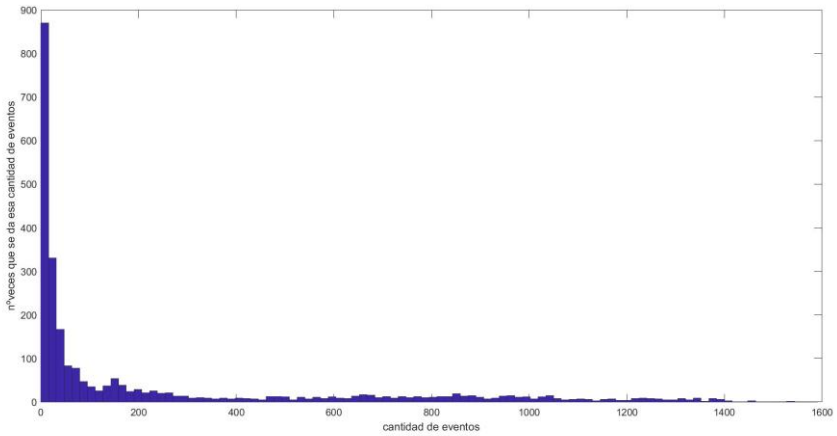


Figura 56. Histograma. Número de veces que se da una cantidad de eventos en un fotograma con Tframe = 800.

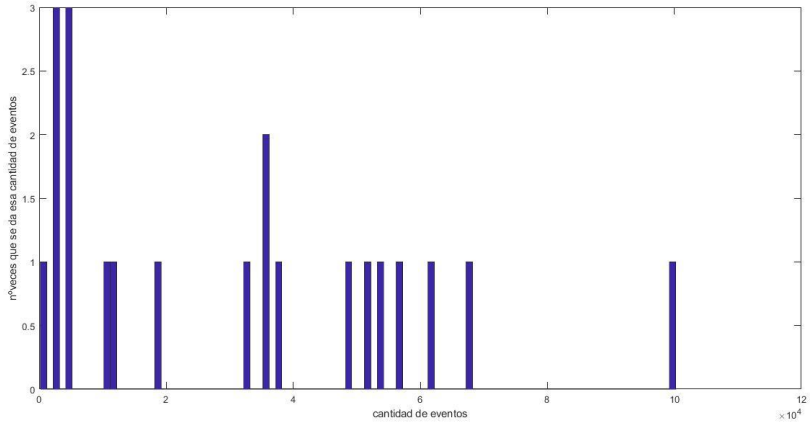


Figura 57. Número de veces que se da una cantidad de eventos en un fotograma con Tframe =100000.

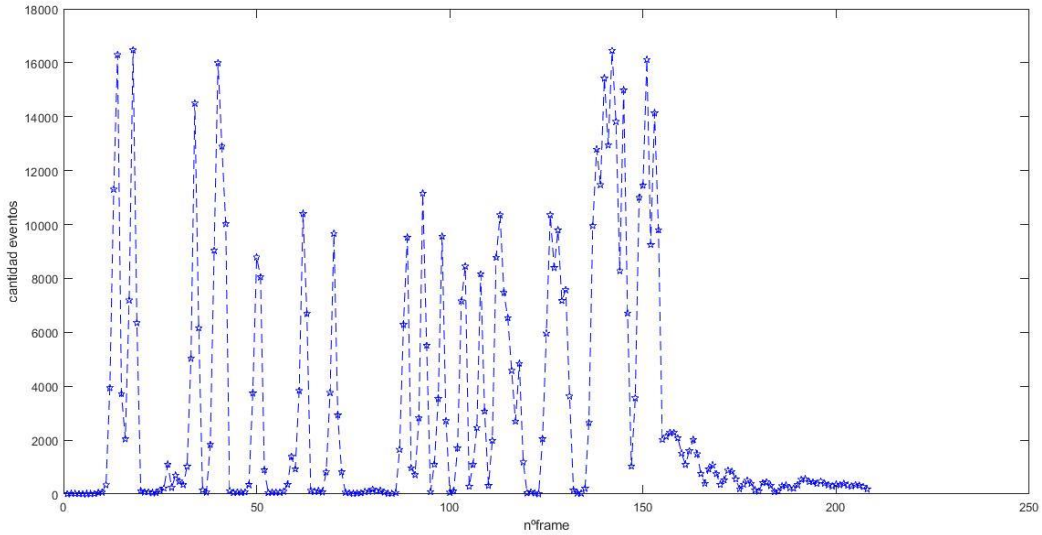


Figura 58. Número de eventos en cada fotograma con Tframe = 10000, grabación cartas. Se está indicando con estrellitas en la figura, la línea discontinua es para poder visualizar mejor los momentos en que aumenta o disminuye la cantidad. Para representar en Matlab se usa plot(1:1:nframes, numero_eventos_en_el_frame,'p--b');xlabel('nºframe');ylabel('cantidad eventos').

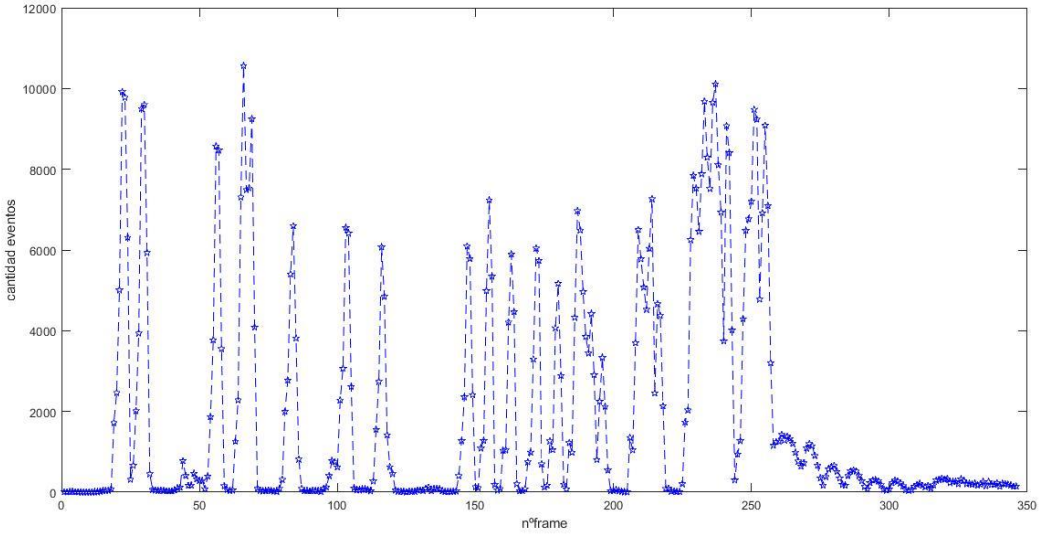


Figura 59. Número de eventos en cada fotograma con Tframe = 6000

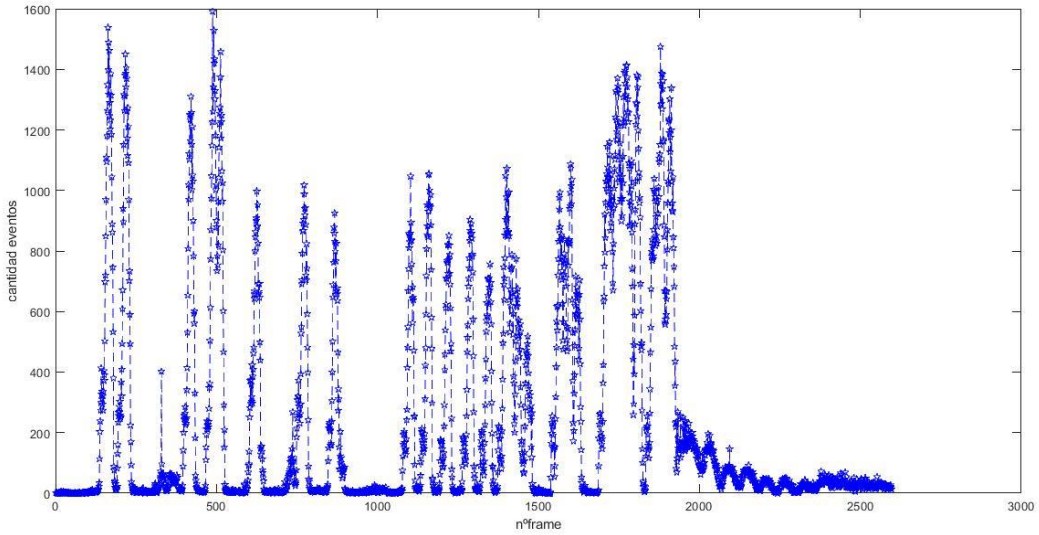


Figura 60. Número de eventos en cada fotograma con Tframe = 800.

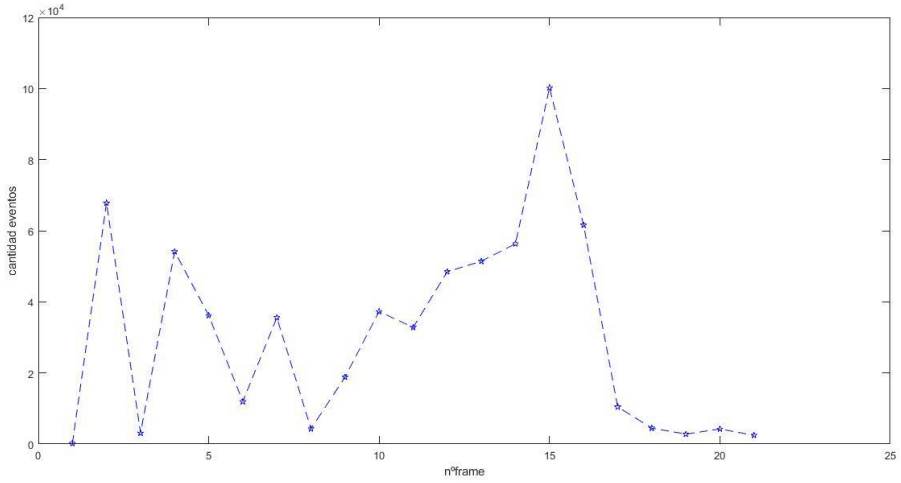


Figura 61. Número de eventos en cada fotograma con Tframe = 100000.

Con respecto la grabación del hombre, con Tframe = 100000000 se obtiene 15 frames y se muestran 57132658 eventos en total. No se visualiza prácticamente nada usando este periodo de tiempo fijo. Por otra parte, la grabación dura demasiado tiempo, son demasiados datos, al ordenador usado le cuesta mucho procesar tanta cantidad de información. Desde el paso 3.2 se trabaja con menos cantidad de eventos de esta grabación.

4.2 Resultados experimentales al convertir eventos en imágenes de escala de grises

Se hicieron diferentes pruebas cambiando los parámetros por defecto (usando el programa de Python). Por ejemplo, se probó con diferentes periodos fijos (variar tiempo de frame), repetir la conversión a eventos buscando el mínimo tiempo de frame posible. En la Figura 62 se muestra el resultado con periodo entre fotogramas de 1ms. Va más lento el video obtenido y se ven unos cuadraditos. Se distingue la imagen, pero no es tan buena calidad como con 33ms. En la Figura 63 se ve que con 10ms de periodo es aceptable. La Figura 64 muestra con 3ms de periodo, sigue siendo aceptable. En la Figura 65 se demuestra que con 0.1ms de periodo no se distingue nada. Por otra parte, como se ve en la Figura 66, con periodo de 500ms se ve mal. Si se establece un periodo de frame fijo, no debe ser ni pocos ni muchos milisegundos. En otras palabras, si se coge un valor menor a 3ms o mucho mayor a 33ms, la reconstrucción no es buena. El valor de 33ms (el de por defecto) parece el más aceptable.

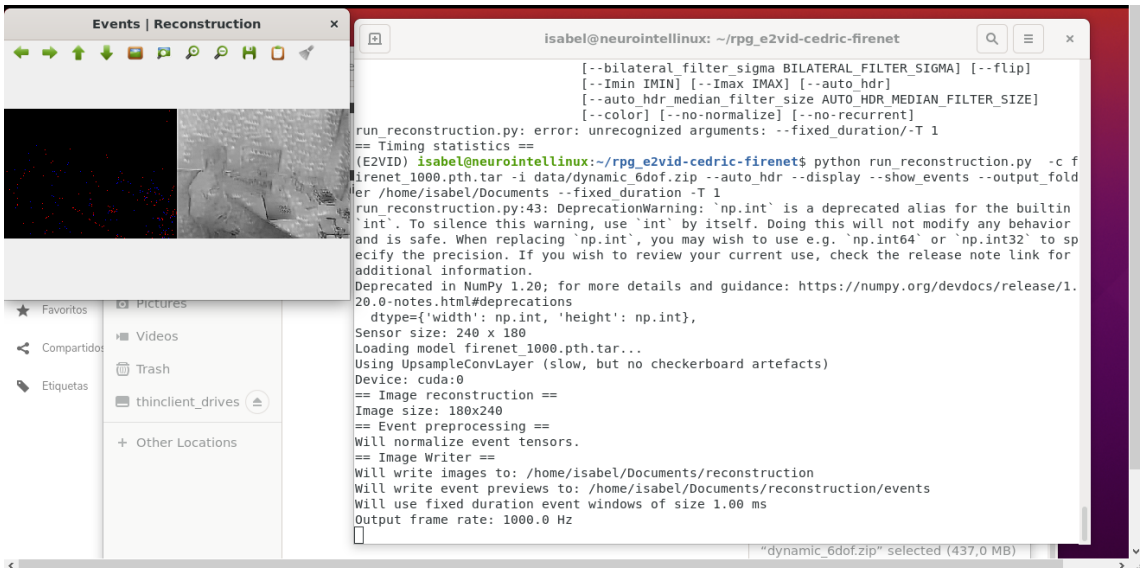


Figura 62. Se probó a cambiar el tiempo de duración de ventana a 1ms.

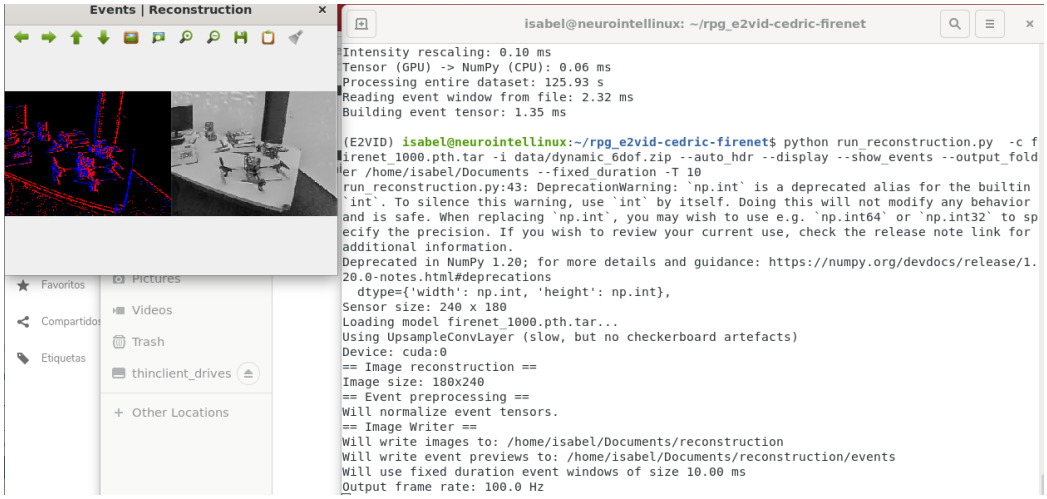


Figura 63. Se probó a cambiar el tiempo de duración de ventana a 10ms.

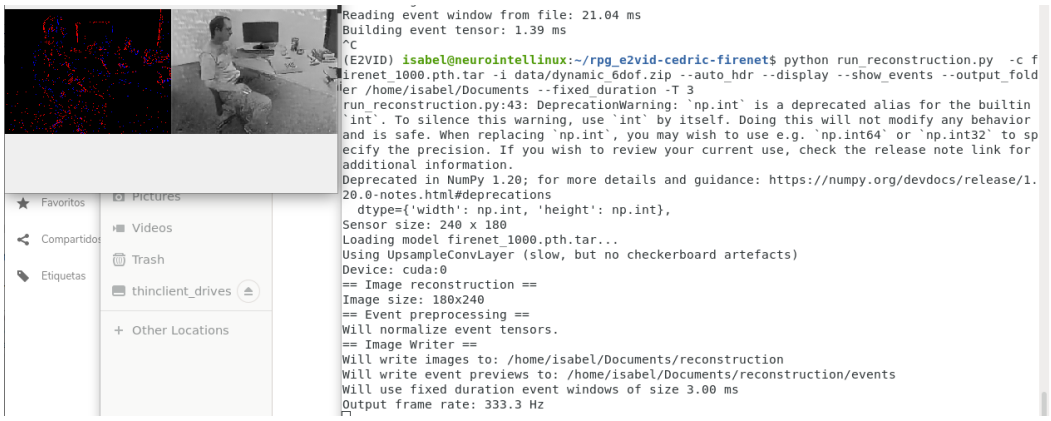


Figura 64. Con 3ms de periodo.

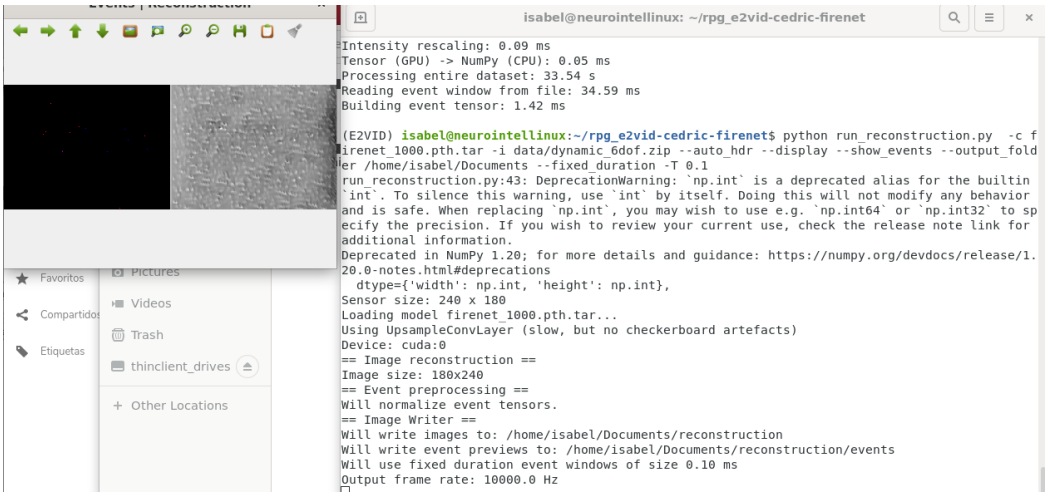


Figura 65. Con 0.1ms de periodo no se distingue nada.

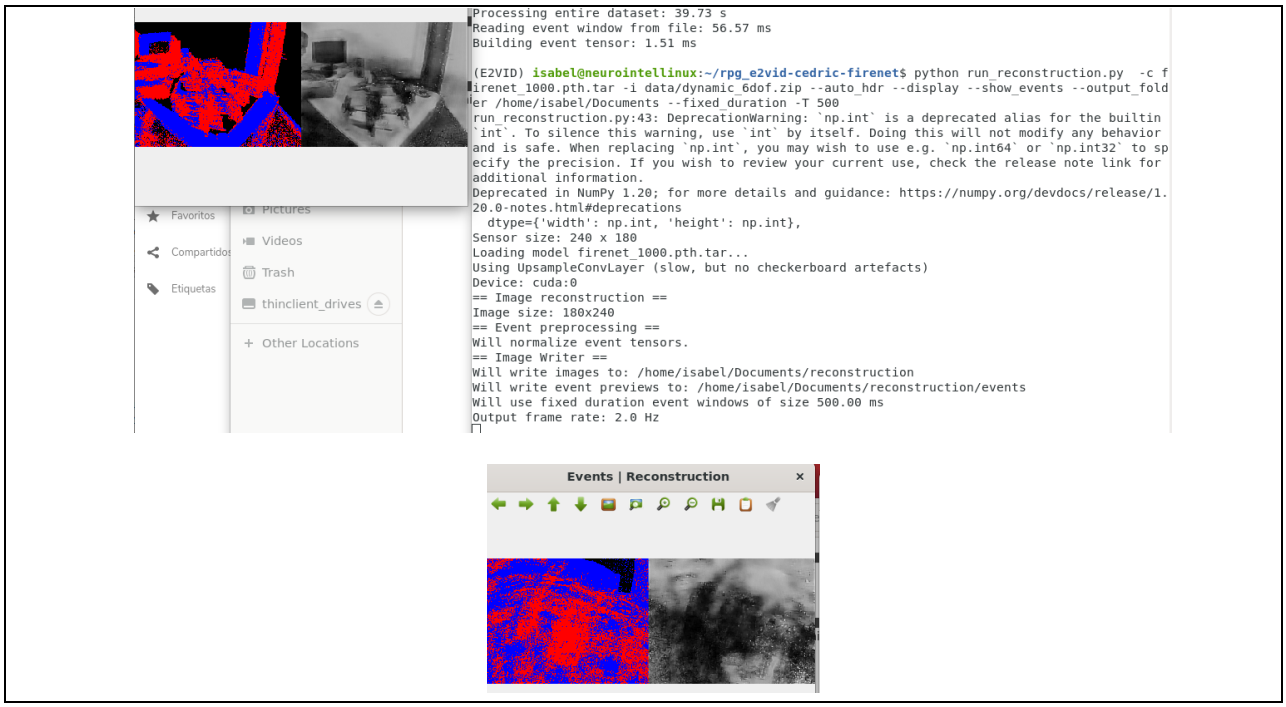


Figura 66. Con periodo de 500ms se ve mal.

Por último, se probó con la grabación de las cartas (Figura 67). El video que muestra es más rápido que con la grabación del hombre. Por otra parte, hay que tener en cuenta que el tamaño del sensor utilizado para cada grabación fue distinto (en caso de las cartas es 128x128 y el del hombre es 180x240) y que la escala y duración del tiempo varía. El programa funciona mejor cuando el periodo es variable, en lugar de fijo; pues espera a que haya un determinado número de eventos, haciendo que no sea tan determinante la duración de la grabación.

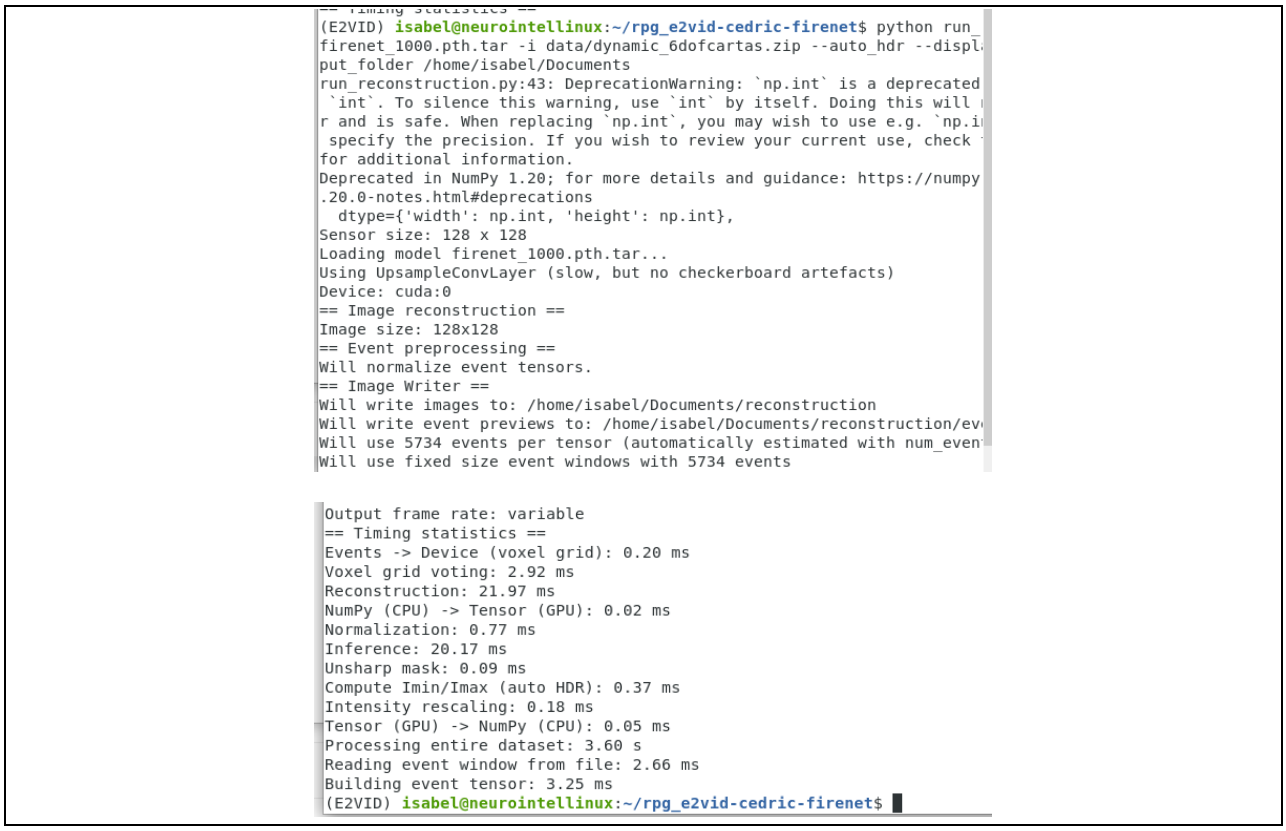


Figura 67. Resultado con grabación de cartas, con periodo variable.

4.3 Resultados experimentales de interpolación

Ahora se mostrarán gráficas representando la intensidad de pixeles concretos con respecto al tiempo, en el caso de las cartas. Los puntos rojos que aparecen en las gráficas son las intensidades interpoladas en el tiempo. Si se aumenta el número de puntos interpolados se obtendrán más imágenes, pero aumentará el coste computacional necesario. Hay que elegir un valor entero adecuado, ni demasiado grande, ni demasiado pequeño. En Matlab se realiza la interpolación haciendo `interp1(1:1:length(pixel1),pixel1,p,'linear')`; se indica los puntos intermedios a coger con `p=[1:0.2:length(pixel1)]` ; siendo el 0.2 el que indica el tamaño de paso para interpolar, apareciendo en ese caso 4 imágenes intermedias. Lo que se le está indicando es para que puntos se quiere obtener su valor estimado.

En la Figura 68 se muestra la evolución de la intensidad del pixel, con coordenadas [i=3,j=59], a lo largo del tiempo. Se decidió obtener 4 puntos mediante interpolación, es decir, se añaden 4 imágenes entre cada dos existentes. Permitiendo así mejorar la resolución temporal. Inicialmente se tenían 113 imágenes y ahora se tienen 561 imágenes. Al aumentar el número de imágenes, se producirán más eventos y mejorará la capacidad de reconocimiento visual en la zona de interés. En la Figura 69 se ve la evolución de la intensidad del pixel [i=1,j=1] cogiendo 4 imágenes intermedias. La imagen de la izquierda muestra la intensidad del píxel respecto al número de imagen correspondiente y la de la derecha se muestra la intensidad del píxel con respecto al momento de tiempo correspondiente de ese fotograma (microsegundos en grabación de cartas). En la Figura 70 se muestra la evolución en [i=66,j=66] cogiendo paso de interpolación de 0.5, es decir, una imagen entre cada dos originales. En ese caso, se obtienen al final 225 imágenes, inicialmente se tenían 113.

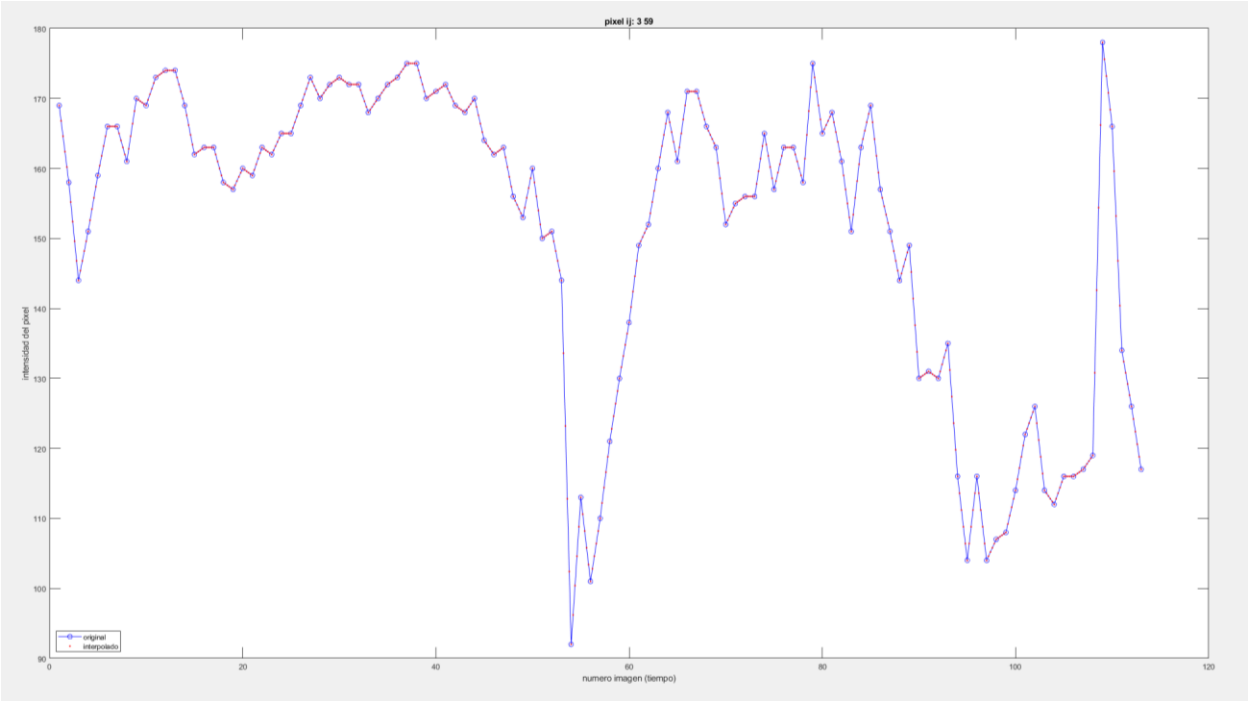


Figura 68. Evolución de la intensidad del pixel ij=[3,59] a lo largo del tiempo, en el caso de las cartas. Se decidió obtener 4 puntos mediante interpolación. Inicialmente se tenían 113 imágenes y ahora se tienen 561 imágenes.

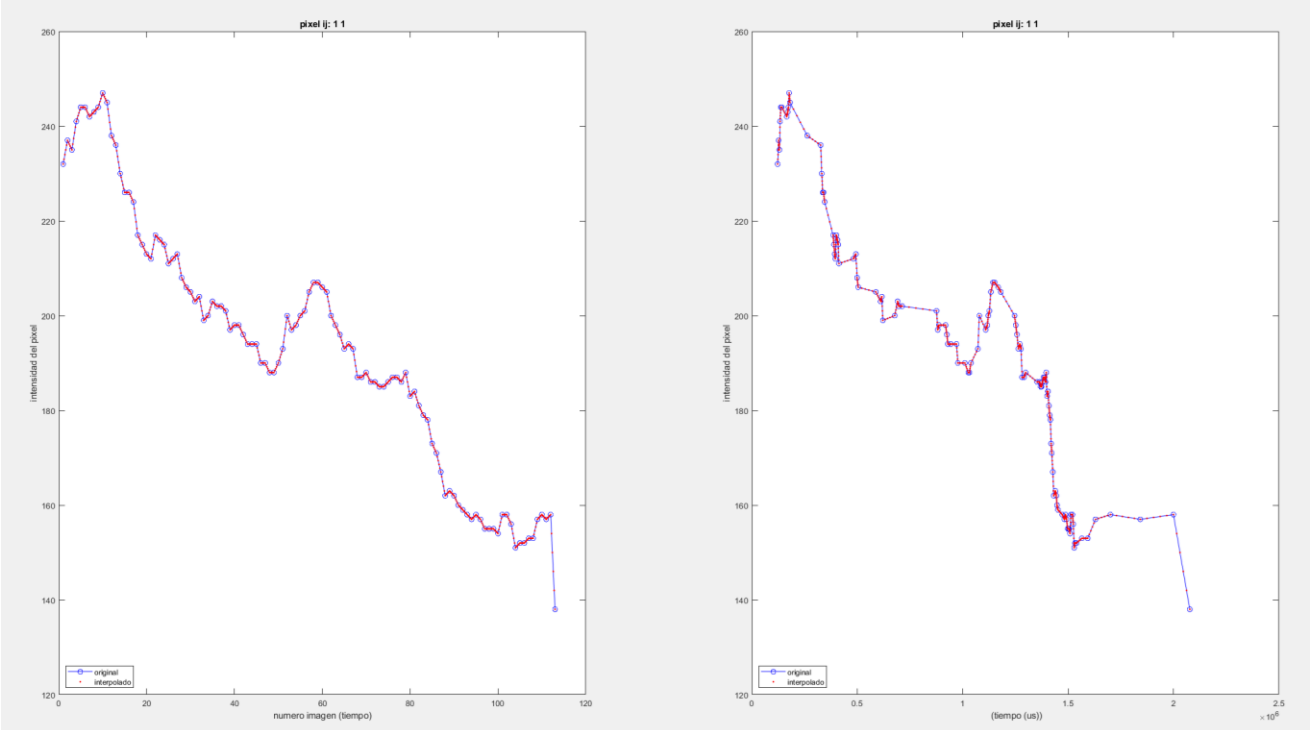


Figura 69. Evolución de la intensidad del pixel ij=[1,1] a lo largo del tiempo, en el caso de las cartas. Cogiendo 4 imágenes intermedias. A la izquierda se muestra la intensidad del píxel respecto al número de imagen correspondiente, a la derecha se muestra con respecto al momento de tiempo correspondiente de ese fotograma (se sabe que son microsegundos en grabación de cartas).

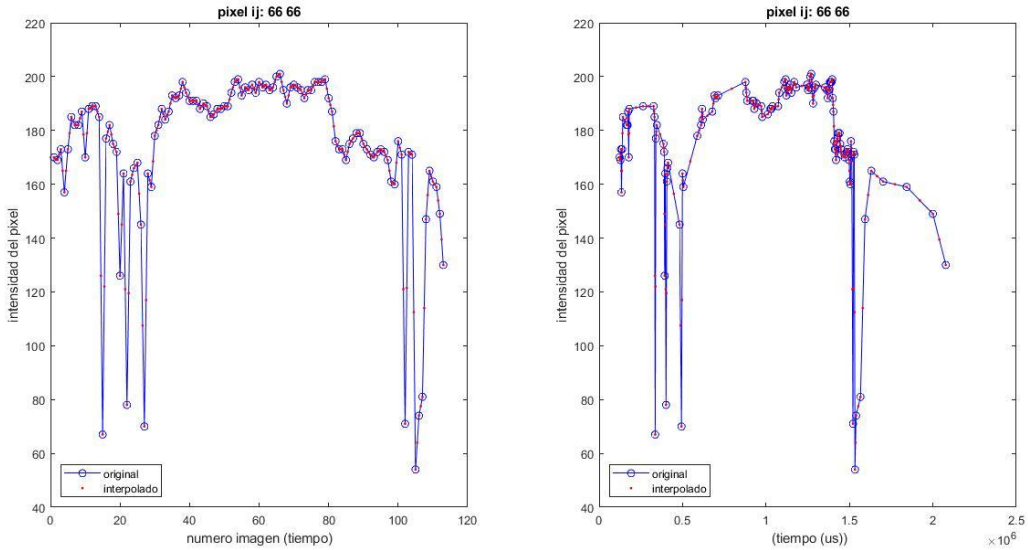


Figura 70. Evolución de la intensidad del pixel ij=[66,66] a lo largo del tiempo, en el caso de las cartas. Cogiendo paso de interpolación de 0.5, es decir, una imagen entre cada dos originales. Inicialmente se tenían 113 imágenes y ahora se tienen 225.

La Figura 71 muestra el caso de la grabación del hombre, en el píxel [i=61,j=44], interpolando 4 imágenes.

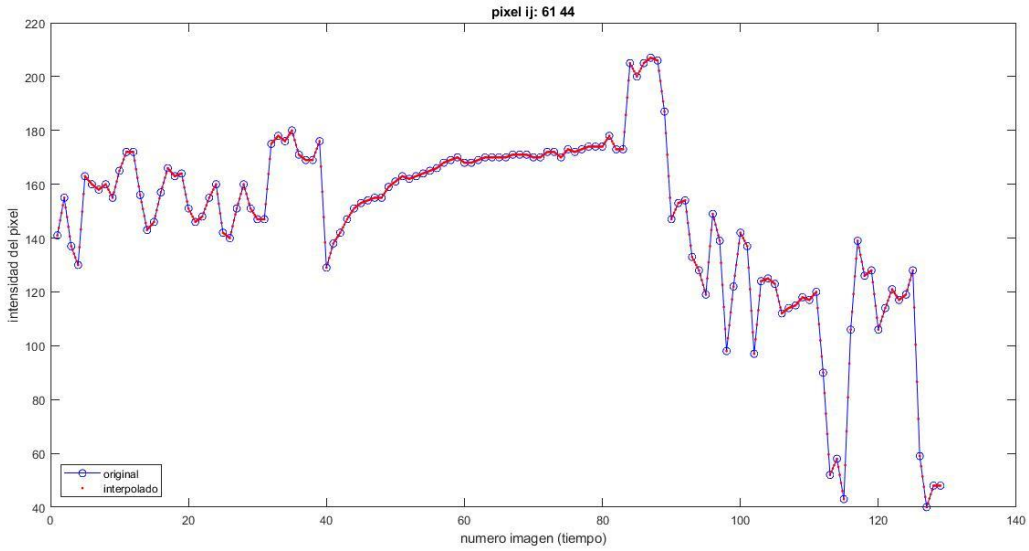


Figura 71. Grabación hombre, interpola 4 imágenes.

4.4 Resultados experimentales en proceso de cambio de resolución (foveación)

Como se aprecia en la Figura 72, se probaron con distintos tamaños de conjuntos de píxeles en la periferia, es decir, con diferentes radios de plantilla[26] para reducir la resolución ($tamaño_{cluster} = \text{nºPixeles}_{\text{plantilla}} = \left(2 * \text{radio}_{\text{plantilla}} + 1\right)^2$). Consiste en ver qué pasa si se disminuye o aumenta el número de pixeles a los que se agrupan como uno solo en la periferia (haciéndose media). Tiene que permitir reducir la cantidad de información sin perder la capacidad de reconocimiento visual del entorno del objeto.

[26] Se le puede llamar plantilla o máscara.

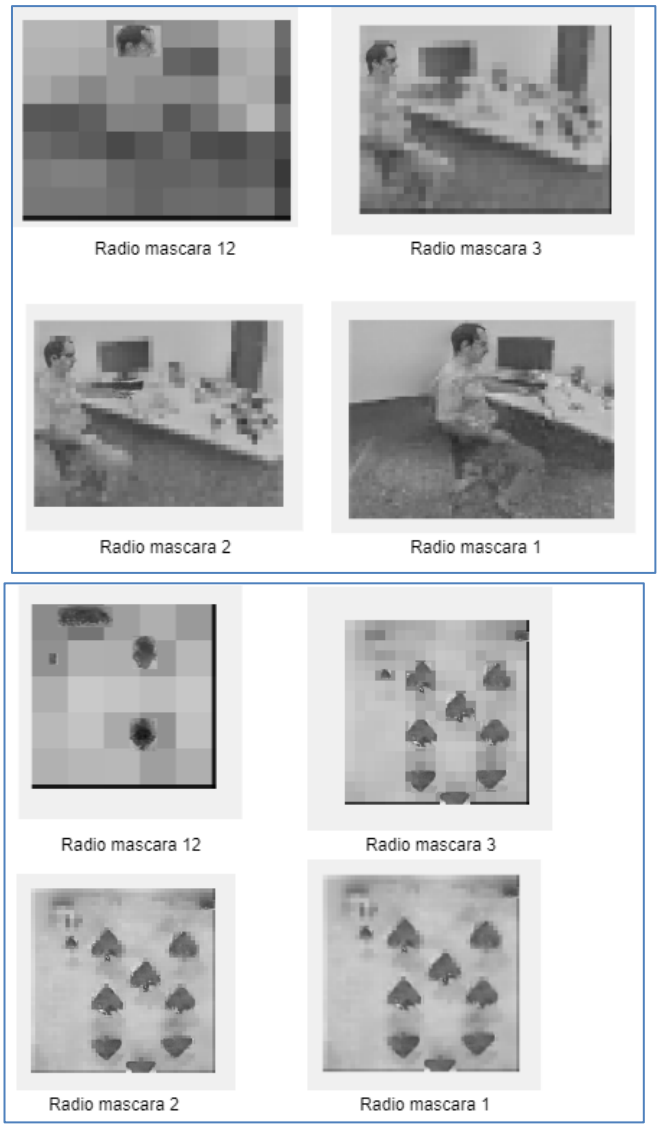


Figura 72. Probando con diferentes radios de la máscara para reducir resolución.

4.5 Resultados experimentales en conversión de imágenes modificadas (con foveación) a eventos

El procedimiento es el mismo en las cartas y en el del hombre, en ambos se probó con el mismo umbral inicialmente, pero no tiene por qué ser así. El umbral se ajusta probando, visualizando cuando se puede ver la cantidad de eventos necesarias para distinguir los elementos que se mueven. A continuación, se muestran los resultados obtenidos de las cartas.

dynamic_6dofcar: Bloc de notas

Archivo Edición Formato Ver Ayuda

```
128 128
123402.000000 7 126 1
123402.000000 7 127 1
123402.000000 8 126 1
123402.000000 8 127 1
123402.000000 14 33 1
123402.000000 15 33 1
123402.000000 16 33 1
123402.000000 17 33 1
123402.000000 18 33 1
123402.000000 18 115 1
123402.000000 18 117 1
123402.000000 18 118 1
123402.000000 18 126 0
123402.000000 18 127 0
123402.000000 19 97 1
123402.000000 19 98 1
123402.000000 19 126 0
123402.000000 19 127 0
123402.000000 20 50 1
123402.000000 20 94 1
123402.000000 20 97 1
123402.000000 20 98 1
123402.000000 20 99 1
123402.000000 20 126 0
123402.000000 20 127 0
123402.000000 21 21 1
123402.000000 21 22 1
123402.000000 21 25 1
123402.000000 21 97 1
123402.000000 21 98 1
123402.000000 21 99 1
```

dynamic_6dofcar: Bloc de notas

Archivo Edición Formato Ver Ayuda

```
2077589.000000 80 118 1
2077589.000000 81 112 0
2077589.000000 83 105 0
2077589.000000 84 111 0
2077589.000000 85 102 0
2077589.000000 85 103 0
2077589.000000 85 104 0
2077589.000000 85 106 0
2077589.000000 85 108 0
2077589.000000 85 111 0
2077589.000000 85 116 1
2077589.000000 86 108 0
2077589.000000 86 115 1
2077589.000000 87 34 1
2077589.000000 87 35 1
2077589.000000 88 45 0
2077589.000000 89 34 1
2077589.000000 94 48 0
2077589.000000 95 43 1
2077589.000000 96 75 1
2077589.000000 96 76 1
2077589.000000 96 79 1
2077589.000000 97 78 1
2077589.000000 98 74 1
2077589.000000 99 72 1
2077589.000000 99 74 0
2077589.000000 99 75 0
2077589.000000 99 76 0
2077589.000000 99 77 0
2077589.000000 99 78 0
```

Figura 73. Eventos de cartas después de foveación (no se muestran todos en la imagen, solo el principio y el final). El umbral usado fue de 0.5 y tamaño de cluster de píxeles de 25.

En la Figura 73 se muestra el archivo de texto obtenido, es decir, los eventos resultante a partir de imágenes foveadas). En el caso mostrado, el umbral usado fue de 0.5 y el tamaño de cluster de píxeles de 25 (la plantilla abarca 25 píxeles). No muestran todos los eventos en la imagen, ya que es un archivo con muchas líneas. Su estructura es como la del archivo texto original (antes de fovear). Mostrándose en cada línea: tiempo, fila del píxel (i), columna del píxel (j) y signo del cambio de luminosidad. Las coordenadas de píxel empiezan en [1,1] en Matlab. En la primera línea se muestra el tamaño del sensor.

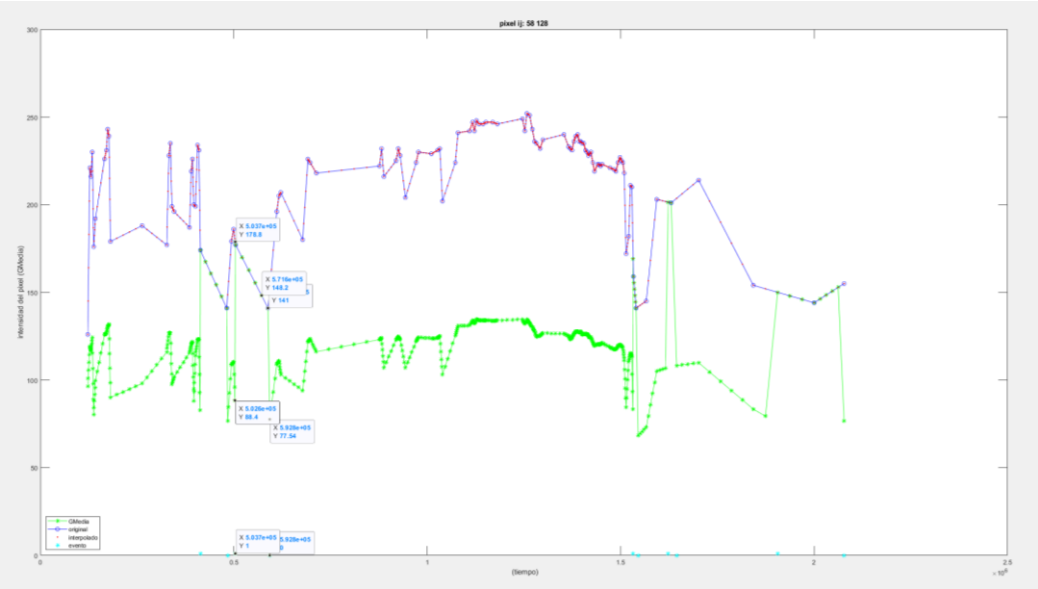


Figura 74. Calculando *Delta(I)/I* para cada par de eventos sucesivos y comparando con el umbral. Se usa la grabación de las cartas con un umbral de 0.4 y el radio de mascara es de 2. Con 4 imágenes intermedias en interpolación (paso interpolación de 0.2).

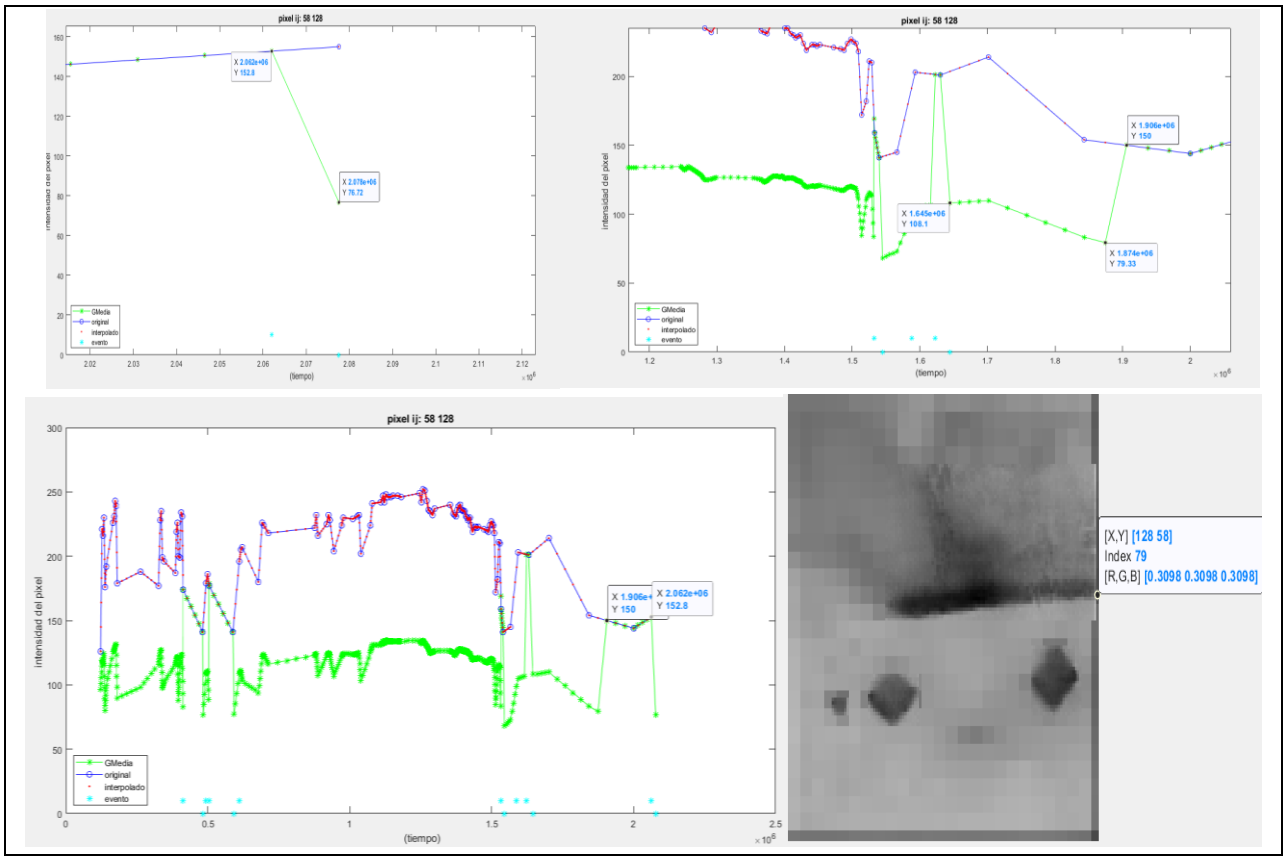


Figura 75. Intensidad de un píxel respecto al tiempo Resultado en el píxel i=58 j=128 (en borde imagen) con umbral de 0.4 y cluster de 25. Con 4 imágenes intermedias en interpolación (paso interpolación de 0.2). Las imágenes de arriba son haciendo zoom (acercamiento) en la de abajo.

En la Figura 74 se muestra la evolución de la intensidad de un píxel respecto al tiempo, comparando la imagen inicial con la modificada. Se usa un umbral de 0.4 y el radio de mascara es de 2. La línea verde (*GMedia*) es después de la foveación y la línea azul es la original. Las estrellas azules indican cuando se considera que se produce evento (siendo 0 cuando es signo 0 y siendo 10 cuando es signo 1). Se representa un píxel de la zona del borde de la imagen ([i=58,j=128]), la cual se le aplicó una media un poco distinta al resto, porque el tamaño de la máscara no podía hacer media de pixeles que no existen, como se explicó en capítulo 3. Cuando pasa un objeto de interés cerca, la subida de resolución hace que haya mayor cambio de intensidad. En esa figura se quiso comprobar que pasaba si, en lugar de hacer la comparación normal para determinar evento, se hallaba el evento calculando *Delta(I)/I* para cada par de eventos sucesivos y comparando con el umbral. En esa circunstancia, sólo se obtiene un evento cuando se produce un cambio brusco (entre dos instantes de tiempo consecutivos), pero no cuando hay un cambio paulatino. Lo correcto es partir del primer valor *I(0)*, y calcular primero (*I(1)-I(0))/I(0).* Si este valor es mayor que el umbral, se genera un evento. Si no, se calcula (*I(2)-I(0))/I(0),* y se compara con el umbral. Si sigue sin superar el umbral, se calcula (*I(3)-I(0))/I(0),* y así sucesivamente. Cuando se llegue al evento que supera el umbral, se empieza a tomar ese evento como referencia. Es decir, si se supera el umbral para *I(8),* a partir de ahí se calcula (*I(9)-I(8))/I(8),* y si no llega al umbral, *(I(10)-I(8))/I(8),* etc. En la Figura 75 se aprecia cómo se realiza correctamente. Se representa con las condiciones anteriores (en borde imagen, con umbral de 0.4 y cluster de 25). Se comprueba que funciona correctamente, (76.72-152.8) /152.8=0.49, es mayor de 0.4, se produce evento. En (150-108.1)/108.1=0.387, no es mayor de 0.4, ya no hay evento en ese punto. En Figura 74 salía 10 eventos en la evolución de intensidad de este píxel, cuando solo se buscaban cambios bruscos. Ahora hay 13 eventos y no tiene por qué coincidir que salgan los hallados en el otro caso. Hay más número de eventos, pero no todos están en el mismo sitio. Es normal que algunos eventos que salían antes ya no estén, ya que el nivel inicial es distinto.

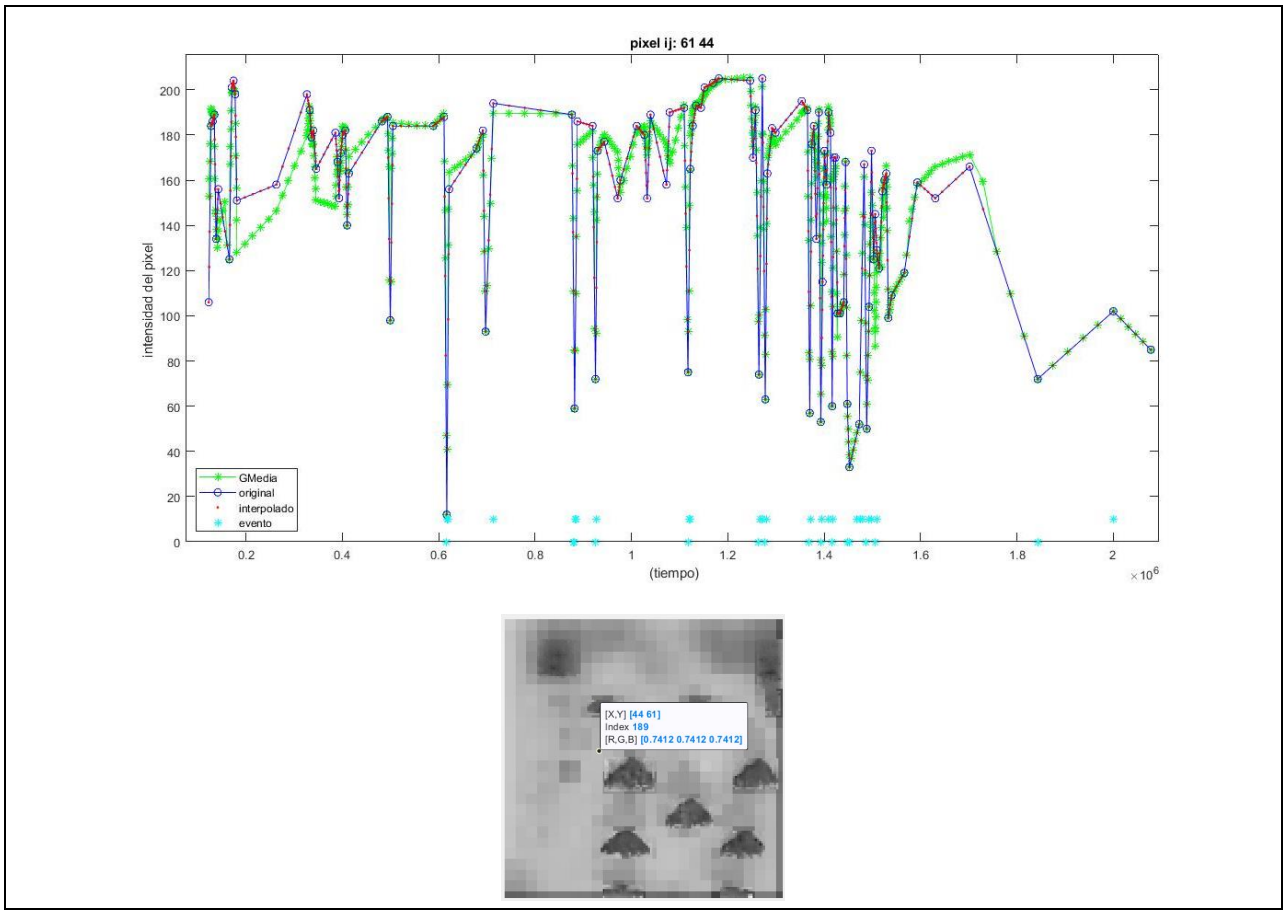


Figura 76. En la de arriba: intensidad de un píxel respecto al tiempo. En la de abajo: qué píxel se ha representado. El umbral usado fue de 0.5 y tamaño de cluster de píxeles de 25. Con 4 imágenes intermedias en interpolación (paso interpolación de 0.2). Los verde (*GMedia*) es después de la foveación.

En Figura 76 se muestra la evolución de la intensidad en el píxel [i=61,j=44]. Viéndose como se mantienen los valores originales del píxel cuando pasa el objeto de interés. En la imagen de abajo se muestra qué píxel es el que se ha usado en el estudio. El umbral usado fue de 0.5, tamaño de cluster de píxeles de 25 y con 4 imágenes intermedias en interpolación (paso interpolación de 0.2). En la Figura 77, la Figura 78 y la Figura 79 se representa la misma gráfica, en ese mismo píxel, pero cambiando parámetros como umbral, tamaño de cluster de píxeles en la periferia y número de imágenes interpoladas.

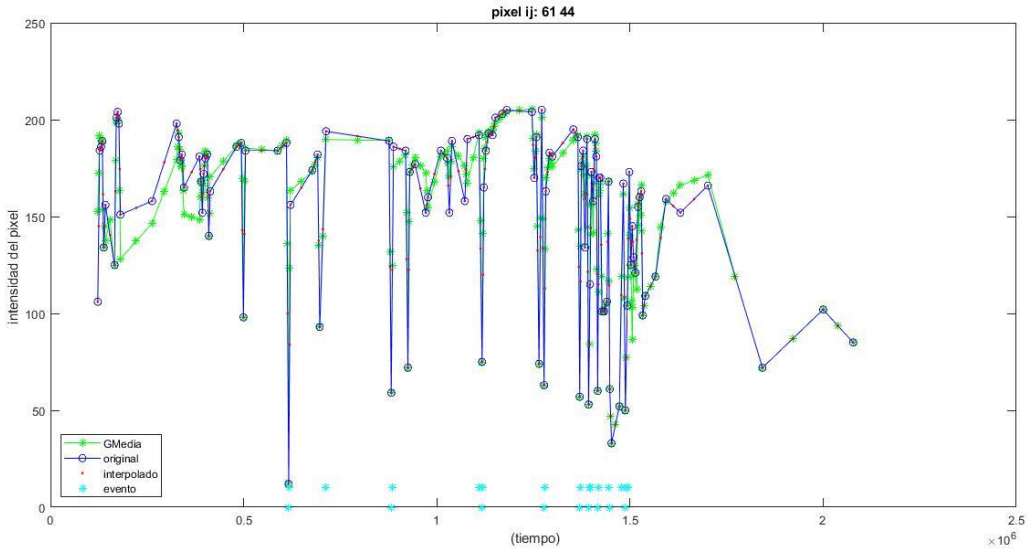


Figura 77. Resultado con umbral de 0.5 y tamaño de cluster de píxeles de 25. Con 1 imagen intermedia en interpolación (paso interpolación de 0.5).

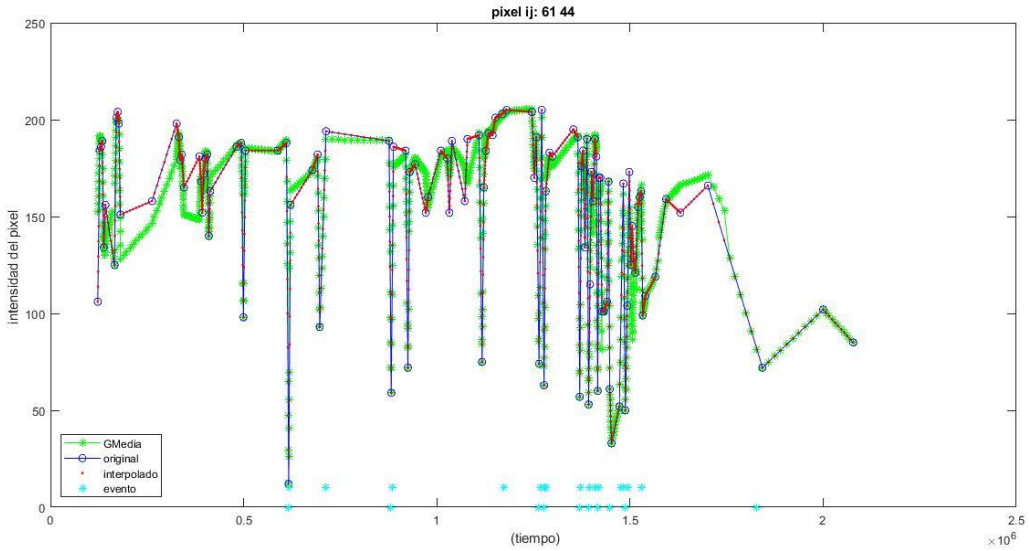


Figura 78. Resultado con umbral de 0.5 y tamaño de cluster de píxeles de 25. Con 9 imágenes intermedias en interpolación (paso interpolación de 0.1).

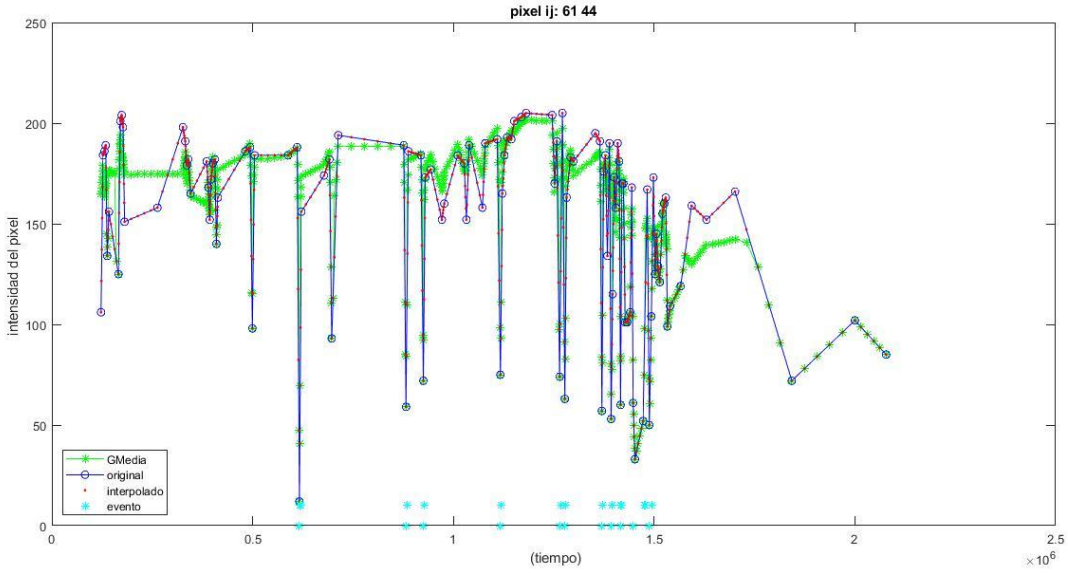


Figura 79. Resultado con umbral de 0.5 y tamaño de cluster de píxeles de 625. Con 4 imágenes intermedias en interpolación (paso interpolación de 0.2).

Después de obtener los eventos modificados, como ya se mencionó, se procedió a representarlos para ver que había funcionado. En la Figura 80, la Figura 81 y la Figura 83 se representa el número de eventos en cada frame en distintas condiciones (variando parámetros). Esto depende del periodo de frame establecido y del número de eventos total que haya (lo cual depende de los parámetros elegidos anteriormente). Se compara el archivo de eventos original (rojo) con el de después de la foveación (azul), viéndose cómo ha disminuido el número de eventos después de la foveación. En la Figura 82 se ve el resultado de representación con Tframe = 10000 (umbral de 0.5, paso de interpolación 0.2 y tamaño de cluster de píxeles de 25). Por otra parte, en la Figura 84 se muestra la representación de eventos en caso de usar periodo variable, con 800 eventos en cada fotograma (umbral de 0.5, paso de interpolación 0.2 y tamaño de cluster de píxeles de 25).

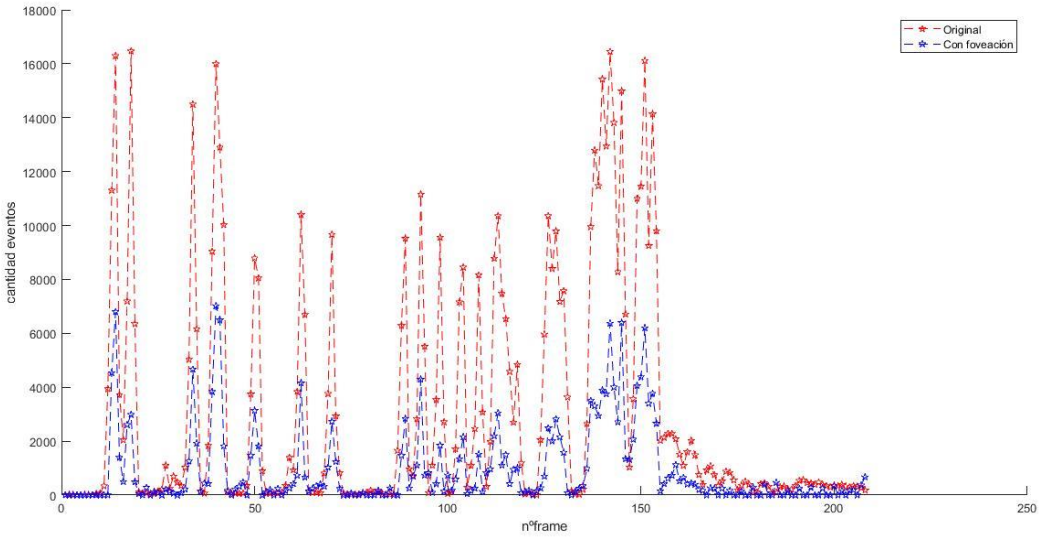


Figura 80. Número de eventos en cada frame, con Tframe = 10000 (208 frames). Resultado con umbral de 0.5 y tamaño de cluster de píxeles de 625. Con 4 imágenes intermedias en interpolación. Hay 195311 eventos después de fovear.

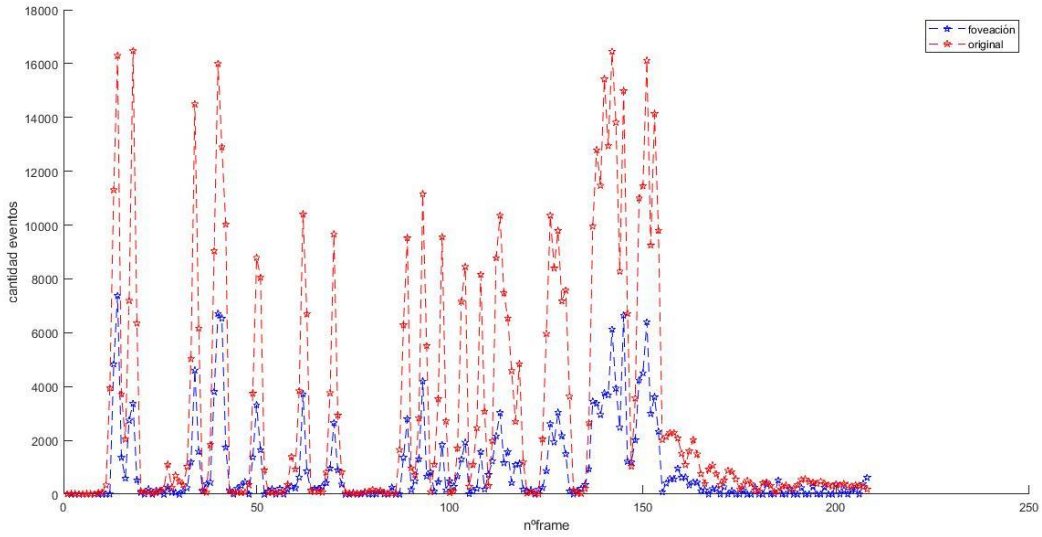


Figura 81. Se compara el original (rojo) con el de después de la foveación (azul), Número de eventos en cada frame, con Tframe = 10000 (208 frames). Resultado con umbral de 0.5 y tamaño de cluster de píxeles de 25. Con 4 imágenes intermedias en interpolación.

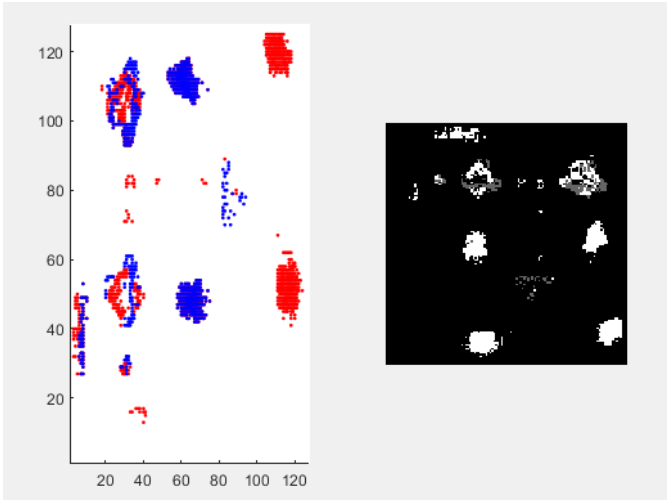


Figura 82. Representación de eventos. Tframe = 10000 (208 frames). Resultado con umbral de 0.5 y tamaño de cluster de píxeles de 25. Con 4 imágenes intermedias en interpolación

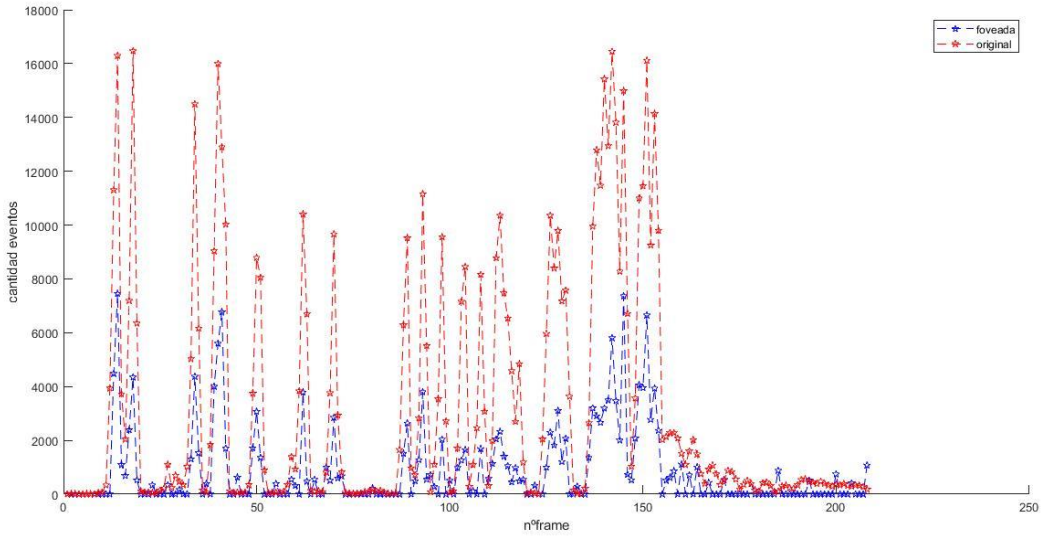


Figura 83. Resultado con umbral de 0.5, tamaño de cluster de píxeles de 25 y con 1 imagen intermedia en interpolación. Con Tframe = 10000 son 208 frames.

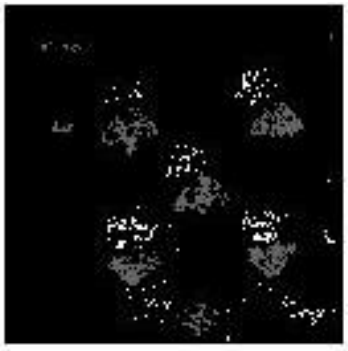


Figura 84. Representación eventos. Caso de la grabación cartas con foveación aplicada (umbral de 0.5, paso de interpolación 0.2 y tamaño de cluster de píxeles de 25). Con 800 eventos en cada fotograma durante representación.

Con respecto a la grabación del hombre, se mostrarán los resultados seguidamente. El archivo de eventos obtenidos después de foveación se encuentra en la Figura 85 (no se muestran todos en la ilustración, solo el principio y el final). En ese archivo, el umbral usado fue de 0.5, tamaño de conjunto de píxeles de 25 y paso de interpolación de 0.2.

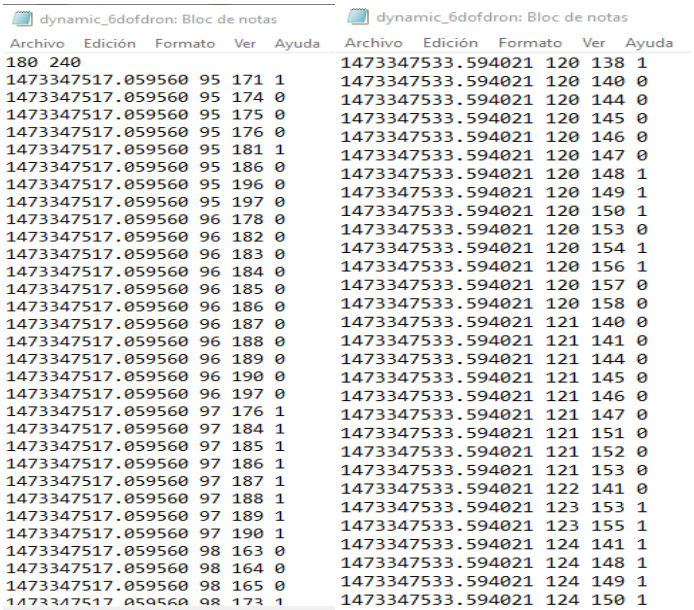


Figura 85. Eventos grabación del hombre después de foveación (no se muestran todos en la ilustración, solo el principio y el final). El umbral usado fue de 0.5 y tamaño de cluster de píxeles de 25.

En la Figura 86 se ilustra la evolución intensidad del píxel [i=61,j=44] en grabación del hombre, con umbral de 0.5 y cluster de píxeles de 25. La zona de interés pasa por ese píxel, por ello hay un tramo que coincide casi perfecto la línea original con la modificada. Hay que decir que antes de interpolación se tenían 129 imágenes y después había 641 imágenes (cogiendo 4 imágenes intermedias). En la Figura 87 está la misma gráfica, pero con un mayor tamaño de cluster de píxeles. En la Figura 88 se muestran los resultados en el borde de la imagen [i=180 j=230] (con mismos parámetros que Figura 86), nunca pasa la zona de interés por él. Después de fovear, se aprecian menos cambios bruscos de intensidad.

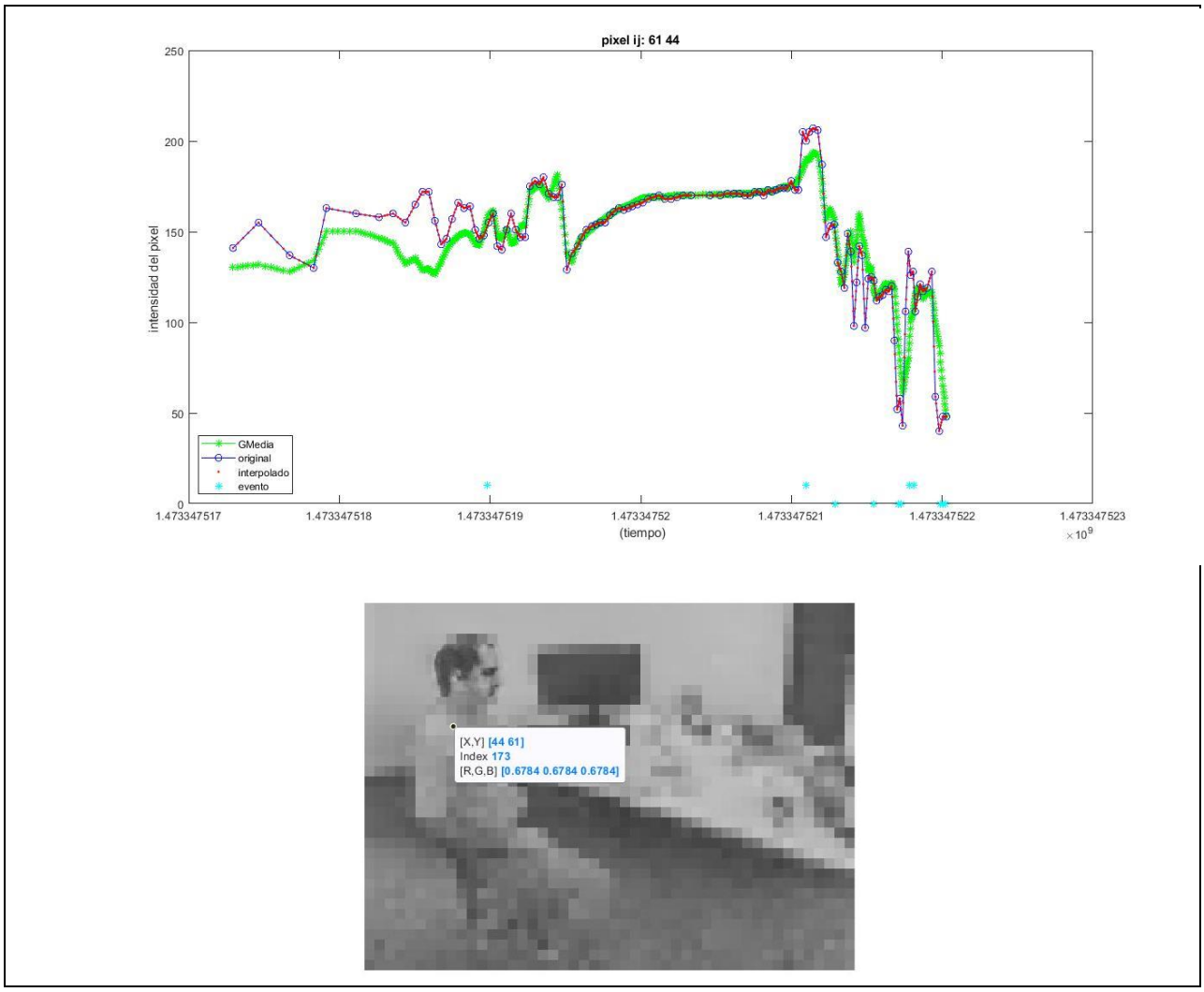


Figura 86. Evolución intensidad de un píxel (en [i=61,j=44]) en grabación del hombre (arriba). La imagen de abajo indica la posición del píxel que se está estudiando (se muestra imagen número 157, después de foveación e interpolación). Tamaño de cluster de píxeles de 25, umbral de 0.5 y paso interpolación de 0.2.

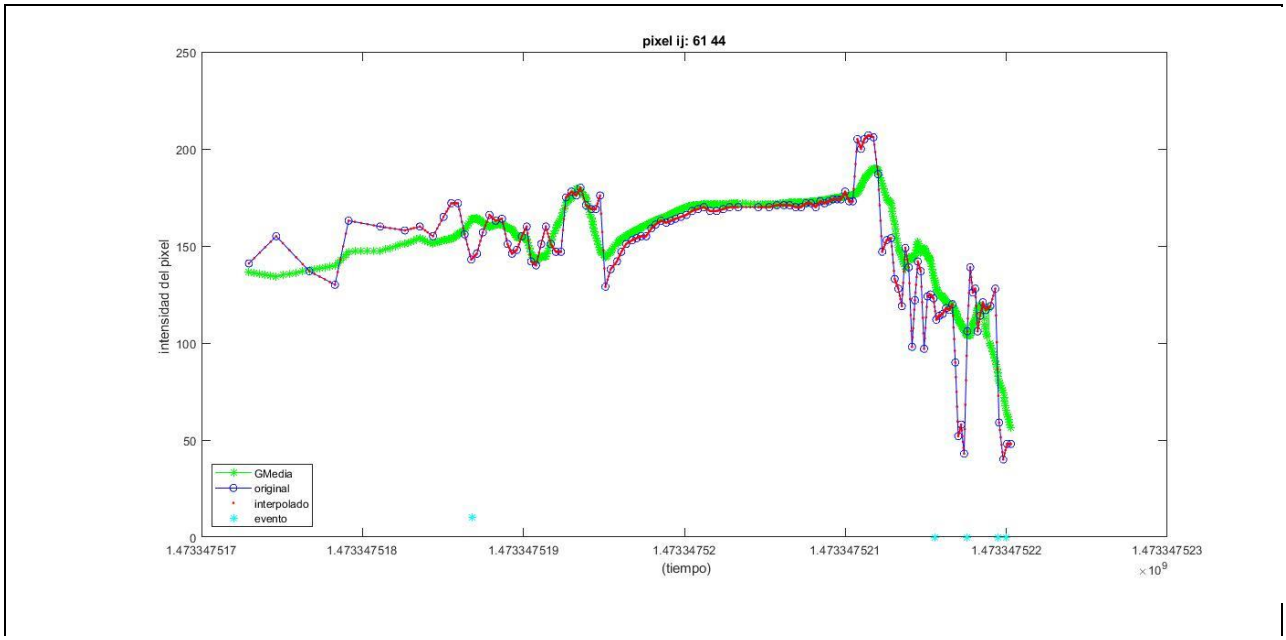


Figura 87. Evolución intensidad de un píxel en grabación del hombre en i=61, j=44. Tamaño de cluster de píxeles de 121 (radio de la máscara de 5), umbral de 0.5 y paso interpolación de 0.2.

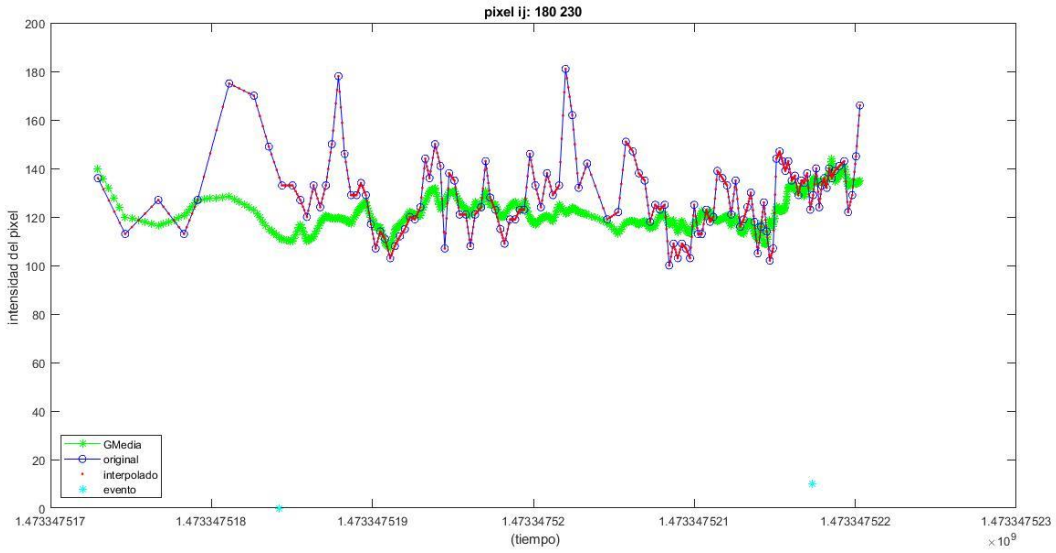


Figura 88. Evolución intensidad de un píxel en grabación del hombre en i=180 j=230. Tamaño de cluster de píxeles de 25, umbral de 0.5 y paso interpolación de 0.2.

La parte de representar los eventos obtenidos, después de las modificaciones, no funcionó bien con esta grabación (en el caso de usar periodo fijo). Se experimentó con varios Tframe, pero ninguno permitió representar correctamente. En la simulación con Tframe = 900900 (1636 frames), con umbral de 0.5 y tamaño de cluster de píxeles de 25, se obtenía 595262 eventos en el último frame y 1 evento en el primero, el resto 0.

Se probó a usar un periodo variable en la representación, esperando a que haya una determinada cantidad de eventos. Se consiguió visualizar de manera aceptable la grabación. En la Figura 89 se representa el caso de la grabación del hombre con foveación aplicada (umbral de 0.2, paso de interpolación 0.2 y tamaño de cluster de píxeles de 25), usando 10000 eventos en cada fotograma (Figura 90). Se distingue una mayor concentración de eventos en la zona del rostro. Si se usase 1000 eventos por fotograma, se obtendría lo mostrado en la Figura 91. En la Figura 92 se ve el caso con umbral de 0.2, tamaño de cluster de píxeles de 121 y paso de interpolación de 0.2. Se visualiza bastante peor que el caso anteriormente mencionado.

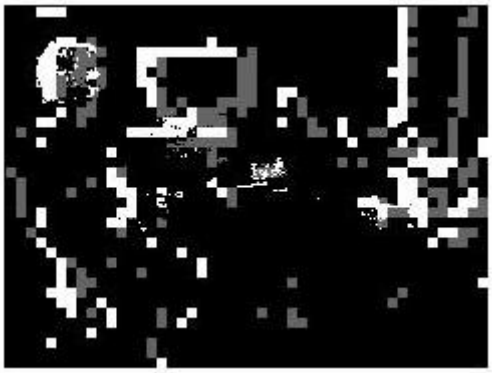


Figura 89. Representación eventos. Caso de la grabación del hombre con foveación aplicada (umbral de 0.2, paso de interpolación 0.2 y tamaño de cluster de píxeles de 25), usando 10000 eventos en cada fotograma.

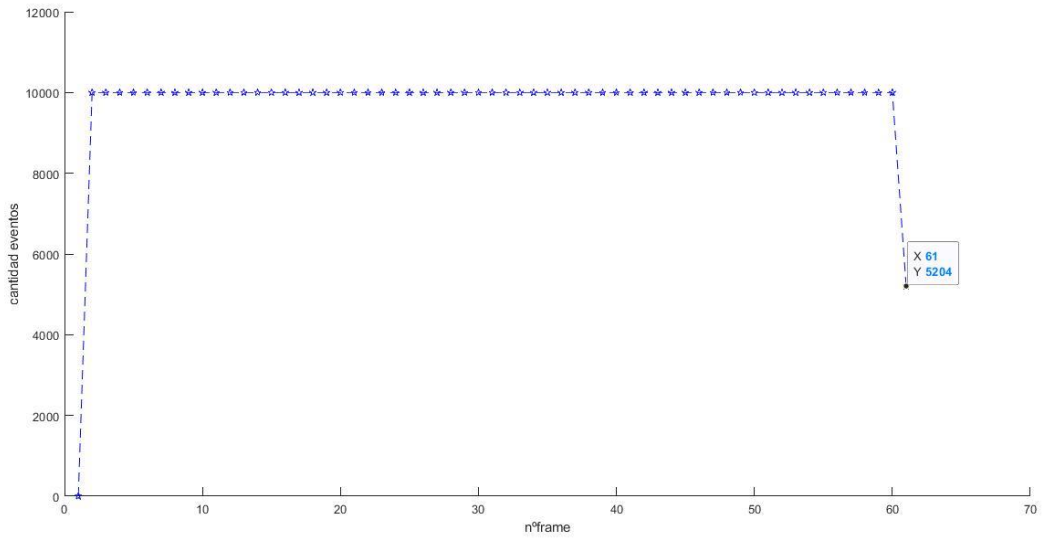


Figura 90. Número de eventos en cada fotograma. Caso de la grabación del hombre con foveación aplicada (umbral de 0.2, paso de interpolación 0.2 y tamaño de cluster de píxeles de 25), usando 10000 eventos en cada fotograma.

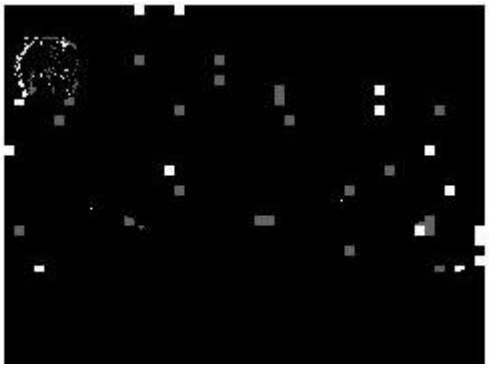


Figura 91. Representación eventos. Caso de la grabación del hombre con foveación aplicada (umbral de 0.2, paso de interpolación 0.2 y tamaño de cluster de píxeles de 25), usando 1000 eventos en cada fotograma.

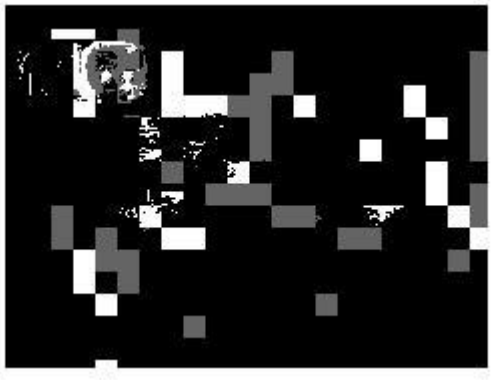


Figura 92. Representación eventos. Caso de la grabación del hombre con foveación aplicada (umbral de 0.2, paso de interpolación 0.2 y tamaño de cluster de píxeles de 121), usando 10000 eventos en cada fotograma.

4.6 Conclusiones

Se mostrará una tabla con resultados obtenidos en ambas grabaciones. En lo referente al paso de conversión de eventos a imágenes en escala de grises, se estableció el periodo entre fotogramas variable con las condiciones por defecto; pues era lo que daba mejor resultado, en lo respecta a reconocimiento visual y tiempo de procesado.

Grabación Cartas:

Número inicial de eventos	Número final de eventos	Umbral para (If-Io)/Io	Tamaño cluster pixeles	Paso Interpolación	Porcentaje de reducción eventos
644672	195311	0.5	625	0.2	69.70%
644672	195602	0.5	49	0.2	69.66%
644672	293221	0.4	25	0.2	54.52%
644672	181103	0.5	25	0.5	71.91%
644672	202247	0.5	25	0.1	68.63%
644672	193039	0.5	25	0.2	70.06%

Tabla 1. Resultados de foveación con cartas.

Inicialmente teníamos en las cartas 644672 eventos; después de la foveacion hay 193039 eventos, cuando el umbral es de 0.5 y tamaño de conjunto de pixeles en periferia es de 25 pixeles ($25 = \text{nPixelsMask} = (2 * \text{radioMask} + 1)^2$). El número de eventos obtenidos debe ser mayor que $\frac{\text{nºeventos}_{\text{inicial}}}{\text{nºpixeles}_{\text{cluster}}}$ y menor que nºeventos_inicial (644672/25=25787<193039<644672), es una manera de comprobar que funciona correctamente.

Se aprecia que si disminuye el tamaño paso de interpolación, es decir, aumenta el número de puntos (intensidades de píxel) intermedios; aumenta el número de eventos. Esto no es extraño, ya que, al haber aumentado el número de imágenes, existe más posibilidades de producirse evento.

Por otra parte, se percibe que si se disminuye el umbral (dejando el resto de los parámetros constante (tamaño cluster y paso de interpolación)) aumenta el número de eventos resultante; si aumenta el umbral, disminuyen.

Si se aumenta el tamaño del conjunto (cluster) de pixeles de la periferia (dejando el resto de los parámetros constante), el número de eventos disminuye, por tanto, el porcentaje de reducción de eventos $\left(porcentaje = \frac{|nº\ eventos_{incial} - nºeventos_{final}|}{nºeventos_{inicial}}\right)$ aumenta. Si disminuye el tamaño, aumentan los eventos.

El número de eventos se ha conseguido reducir con respecto al de sin foveación un 70%. Esto variará según cuanto bajemos la resolución en la periferia, el umbral que determina cuando hay evento, cuantas imágenes se consigan interpolando, la duración de la grabación y el periodo usado entre fotogramas (durante la conversión a escala de gris). Se desea que la cámara produzca menos eventos para que haya menos coste computacional, y, en consecuencia, menor consumo. No obstante, a mayor número de eventos, mejor visualización. Es deseable que haya el menor número de eventos posibles sin perder la capacidad del reconocimiento visual. En el del 70.06% se comprobó que todavía se podían distinguir las formas de las figuras de las cartas de manera más o menos aceptable (Figura 84, Figura 82).

Grabación Hombre:

Número inicial de eventos	Número final de eventos	Umbral para (If-Io)/Io	Tamaño cluster pixeles	Paso Interpolación	Porcentaje de reducción eventos
1950351	89340	0.5	25	0.2	95.42%
1950351	595263	0.2	25	0.2	69.47%
1950351	445973	0.2	121	0.2	77.13%

Tabla 2. Resultados de foveación con grabación del hombre.

Durante el paso de reconstrucción (conversión de eventos a imágenes de escala de grises, en apartado 3.2) se estableció que había unos 15119 eventos por ventana, es decir, por imagen. Como se usan 129 imágenes (no se cuentan las obtenidas después de interpolar), serían unos 1950351 eventos inicialmente.

En el primer caso de la tabla se aprecia que $\frac{\text{nº eventos}_{\text{inicial}}}{\text{nº pixeles}_{\text{cluster}}} = \frac{\mathbf{1950351}}{\mathbf{25}} = 78014 <$89340<1950351. En el resto de los casos también se cumple la condición que verifica el correcto funcionamiento del programa. El proceso es mucho más lento que con las cartas. Además, en el primer caso se aprecia que el porcentaje de reducción de los eventos es mucho mayor que en las cartas; dándose los mismos valores a los parámetros de umbral, tamaño de cluster e interpolación. Como se mencionó con anterioridad; se ve como al aumentar el tamaño de conjunto de píxeles (que se consideran como uno solo) en la periferia, disminuye el número de eventos.

En el caso del porcentaje de reducción del 69.47% se consigue un reconocimiento visual más o menos aceptable (Figura 89,Figura 91). La visualización de los eventos depende del método usado para su representación (depende del periodo usado).

5 EXPLICACIÓN CÓDIGO

"Inteligencia artificial, aprendizaje profundo, aprendizaje automático… te dediques a lo que te dediques, si no lo comprendes tienes que ponerte con ello y aprender qué es. Porque de lo contrario serás un dinosaurio dentro de 3 años."

Mark Cuban

Se explicarán los códigos realizados en MATLAB de manera resumida. Los archivos utilizados son **obten_datos_imagen.m, obten_datos_imagen_cartas.m, detection_object.m, obten_event_cart.m, repre.m, isabelyo3.m, isabelyo5.m, isabelyo.m, isabelyo4.m, ImportAedat.m, frames_2D.m.** Los códigos y las explicaciones menos relevantes se encuentran en el ANEXO B**.** Para la representación de los eventos usando el programa de la cámara (jAER) se partió de archivos que fueron proporcionados por el IMSE y de [92].

Adjunto a este trabajo estarán los fotogramas obtenidos de la reconstrucción, tanto de la grabación de las cartas (**reconstruction_cartas**) como del hombre (**reconstruction1**), sin tiempo de frame fijo. También los obtenidos con el hombre dado un periodo de 33.33ms (**reconstruction33hombre**).

Los programas principales (los cuales realizan la foveación a partir de eventos) son **obten_datos_imagen.m** para la grabación del hombre y **obten_datos_imagen_cartas.m** para la grabación de las cartas. En las la Figura 93 y en la Figura 94 se encuentran los diagramas de flujo de cada programa respectivamente.

Se hicieron unos códigos para representar los eventos:

- Para representar los eventos a partir del archivo creado a partir de la cámara (*.aedat*) (**isabelyo.m**)
- Para representar la grabación del hombre original o la de las cartas original en formato *.txt* (**isabelyo5.m**)
- Para los archivos texto que nosotros creamos con Matlab (después de foveación) (**isabelyo3.m**) usando periodo de tiempo fijo; y otro usando periodo variable (**isabelyo4.m**).

Hay que tener presente que el archivo de eventos que genera el programa (de la cámara) no empieza las coordenadas de píxel en [1,1] como en Matlab, sino en [0,0]. Esto hay que tenerlo en cuenta en el código. En la grabación del hombre también sucede esto.

En **isabelyo.m, isabelyo4.m, isabelyo5.m, isabelyo3.m, obten_datos_imagen_cartas.m** y **obten_datos_imagen.m**; hay que indicar la dirección donde están los archivos necesarios o guardarlos en la misma carpeta donde este el archivo '*.m*' que se va a ejecutar. Hay que especificar la carpeta donde se encuentra

el archivo original de eventos, la carpeta con los fotogramas de la reconstrucción o las funciones necesarias para su ejecución.

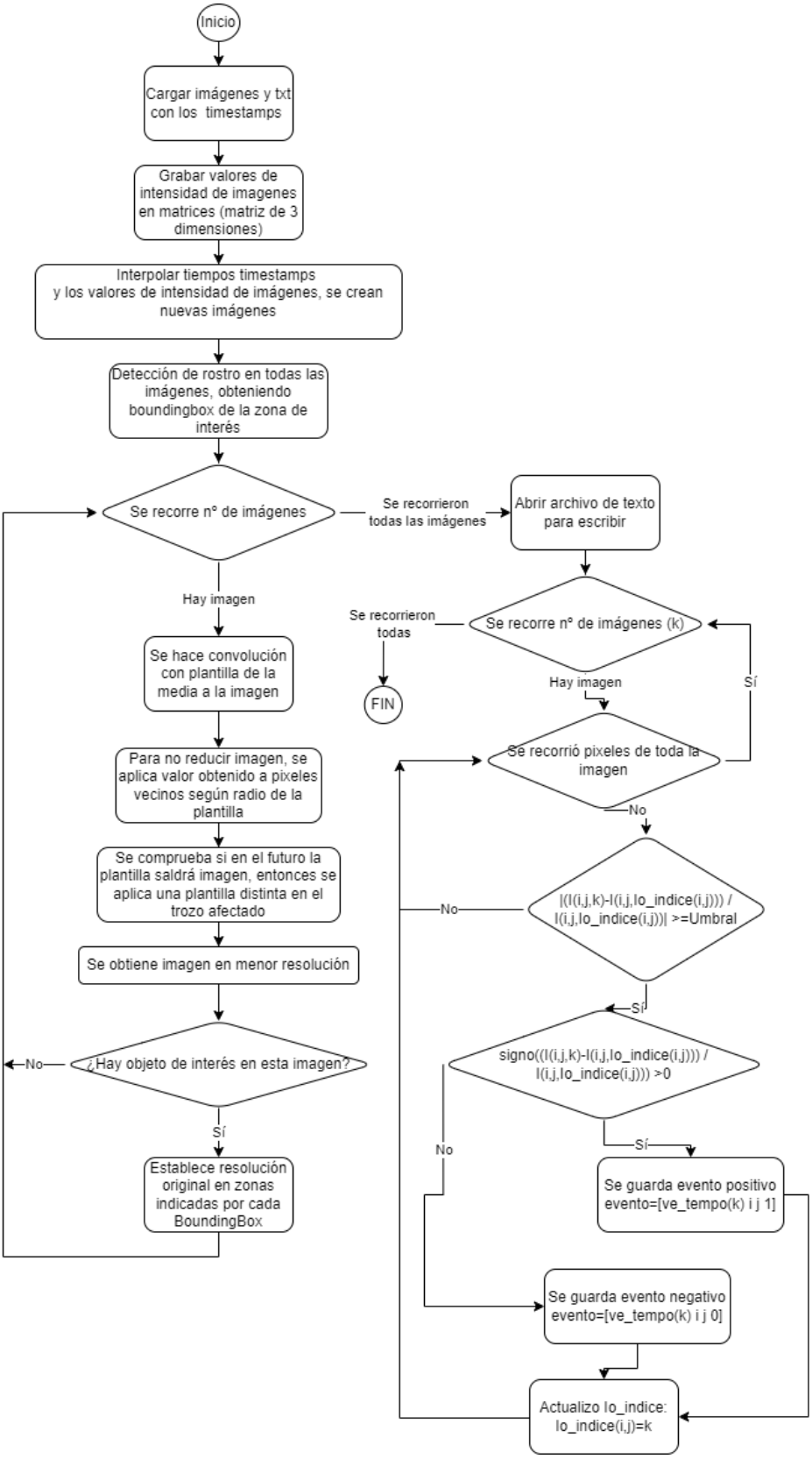


Figura 93. Diagrama de flujo de **obten_datos_imagen.m.** Realizado con draw.io.

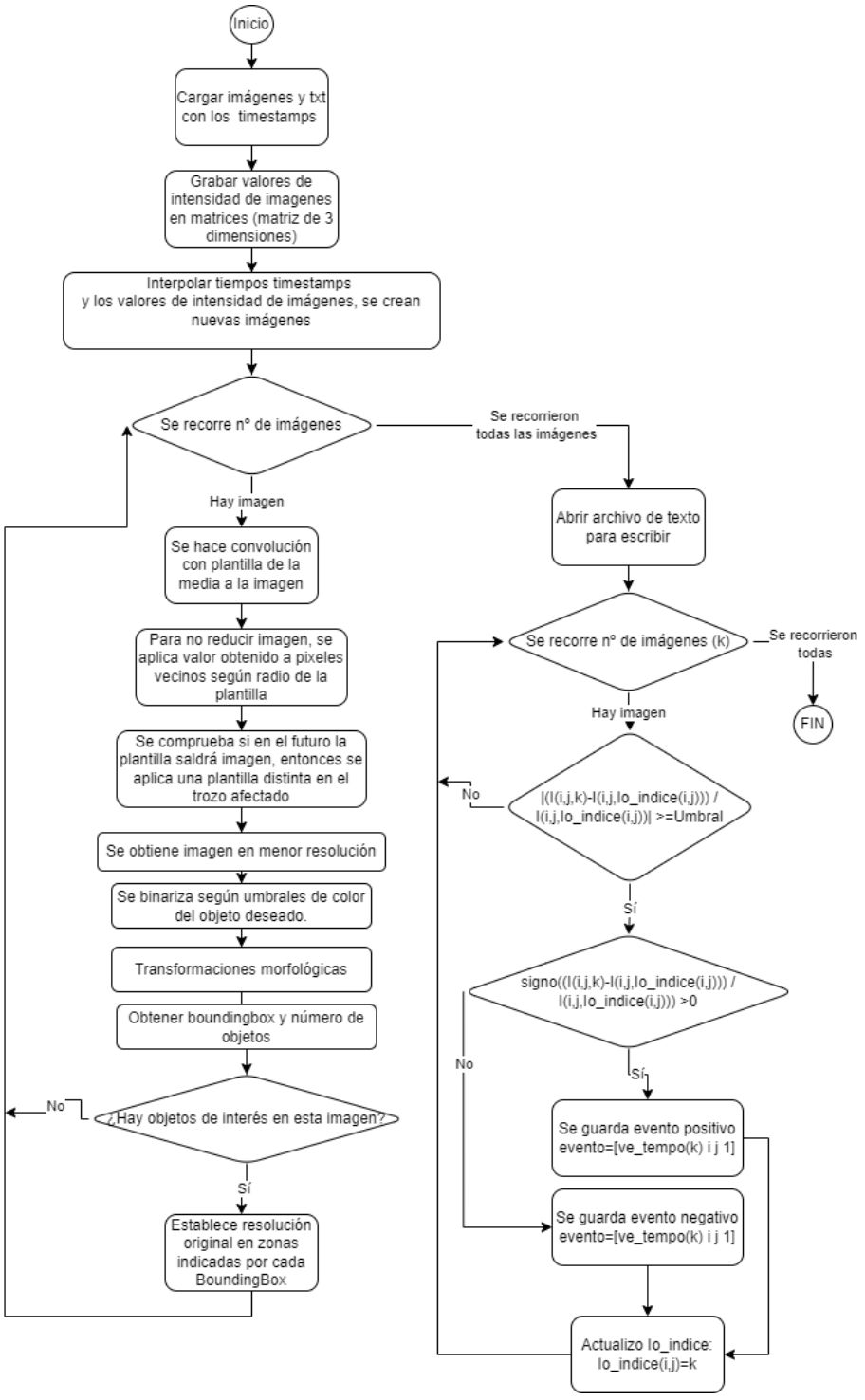


Figura 94. Diagrama de flujo de **obten_datos_imagen_cartas.m**. El recorrido de la imagen está de manera implícita. Realizado con draw.io.

Se ha realizado una tabla con el significado de las variables más importantes, usadas en **obten_datos_imagen_cartas.m** y **obten_datos_imagen.m**:

Variable	Significado
`archivof`	Dirección donde se encuentra el archivo de texto con los tiempos.
`tempo`	Vector con los tiempos originales en los que se creó el fotograma en escala de gris
`limit`	Por si se quieren usar menos imágenes de las obtenidas

I	Imagen leída
M	Número total filas de la imagen, la altura en pixeles
N	Número total columnas de la imagen, la anchura en pixeles
i , j	Índices de los bucles for al recorrer imágenes. La i para las filas y la j para las columnas.
intensidades	Matriz de 3 dimensiones que va almacenando las imágenes cargadas
pixel1	Vector con la intensidad de un píxel concreto a lo largo del tiempo (no interpolado).
p	Puntos a lo que se van a estimar durante interpolación
ve_tempo	Vector con tiempos interpolados. Los tiempos son los de muestreo de los fotogramas cuando se hizo reconstrucción.
ve	Vector con la intensidad de un píxel concreto a lo largo del tiempo (interpolado).
intensidades2	Matriz de 3 dimensiones que va almacenando las imágenes, incluyendo interpoladas
n_imagenes_inicio	Número de imágenes usadas antes de interpolar
n_imagenes	Número de imágenes que hay después de interpolar
radioMask	Radio de la máscara (plantilla) que se usa para recorrer imagen aplicando la media; haciendo convolución sin reducir tamaño de la imagen. Copiando el valor del píxel a los pixeles vecinos (según plantilla), después de haber aplicado media
nPixelsMask	Cuántos pixeles conforman la plantilla
k	Índice de número de fotograma a procesar
gMedia	Donde se guarda imágenes resultantes de después de cambio de resolución. Inicialmente se tiene la imagen en alta resolución, primero se guardan imágenes con reducción de resolución espacial completa. Luego se le cambia los valores con corresponden a zona de interés, poniendo sus valores de intensidad originales en esas zonas.
io , jo	Índices filas y columna de `gMedia2`
gMedia2	Guarda valor de intensidad obtenido de haber aplicado la plantilla de la media.
ii , jj	Índices filas y columna de `gMedia`. Para recorrer plantilla de la media.
id,jd	Índices filas y columna de `gMedia`. Para recorrer pixeles que corresponden a plantilla (la cual está centrada en un determinado píxel) dándoles el valor de la media, el guardado en `gMedia2`, según las coordenadas del centro de la plantilla.
inic	Se usa para la última fila de la imagen, es decir, a partir de la cual la plantilla se sale de la imagen. Es el valor de la última fila que se procesa antes de que se produzca el problema de que se sale plantilla.
h	Una de las imágenes a alta resolución de `intensidades2`
C	Centroides de objetos de una imagen
B , bbox	Boundingbox de objetos de una imagen
Cx	Coordenadas x de los centroides objetos

Cy	Coordenadas y de los centroides objetos
n_cuadrados	Número de Boundingboxes que aparecen en una imagen.
s	Índice para recorrer los Boundingboxes de una imagen.
[m n]	Tamaño de la imagen.
fin_M	Fila final del Boundingbox
ini_i	Fila inicial del Boundingbox
fin_N	Columna final del Boundingbox
ini_j	Columna inicial del Boundingbox
[a b c]	Tamaño de `gMedia`
Ay	Umbral que determina la aparición de evento
tama	Tamaño de imagen, por tanto, tamaño sensor
Io_indice	Matriz del tamaño de la imagen. Almacena el valor de intensidad con el que hay que comparar cada píxel para determinar si se produce evento o no. La Matriz se actualiza cada vez que se produce un evento.
event	Matriz en la que se almacenan los eventos de después de foveación (4 columnas y el número de filas es el número de eventos)
cont	Número de eventos, índice de la fila de `event`
n_eventos1	Número de eventos con signo positivo
n_eventos0	Número de eventos con signo negativo
n_eventos	Número de eventos total producidos
index	Vector con número de Boundingbox de todas las imágenes
ty	Índice de for para contar número de Boundingboxes procesados
facedetector	Lo que permite detectar ojo derecho de persona en una imagen.
bb	Boundingbox que se obtiene después de aplicar `facedetector` a la imagen

Tabla 3. Significado variables en códigos principales: **obten_datos_imagen_cartas.m** y **obten_datos_imagen.m**.

Las variables usadas en **isabelyo3.m, isabelyo.m** son las siguientes.

Variable	Significado
eventos	Matriz con eventos del archivo de texto
vector_tiempos	Primera columna de los eventos
x	Segunda columna de los eventos
y	Tercera columna de los eventos
M	Número filas sensor (imagen)
N	Número columnas sensor
Tframe	Periodo que determina el número de imágenes que se van a representar a partir de los eventos
nframes	Número de imágenes a representar
i	Índice de número de imagen en el for
puntos	Matriz que es una imagen donde vienen representada eventos. Para mostrar con imshow()
index1 , indey1	Coordenadas del evento de signo positivo, para representar con plot()
index0 , indey0	Coordenadas del evento de signo negativo, para representar con plot()
u	Índice de número eventos positivos
w	Índice de número eventos negativos

cont	Valor de tiempo del evento. Mientras que los valores de tiempo de los eventos sean menores que el periodo, se consideraran de la misma imagen `puntos`
numero_eventos_en_el_frame	Número de eventos que hay en cada imagen
frames	Matriz de 3 dimensiones que guarda todas las imágenes `puntos`
t_f	Índica cuando se considera que ha pasado un `Tframe`. El número de periodos transcurrido. Se actualiza así: `t_f=Tframe+i*Tframe`
archivo	Archivo de texto
aedat	Estructura con grabación .aedat de jAER

Tabla 4. Significado variables en códigos principales: **isabelyo3.m, isabelyo.m**

5.1 Conclusiones

En este capítulo se ha podido aclarar los códigos desarrollados en Matlab durante el proyecto. Principalmente se han requerido conocimientos de procesamiento de imágenes. También se requería tener conceptos claros sobre Machine Learning para la parte de detección de objetos.

Los códigos de Python de la Universidad de Zurich no han sido mostrados en este documento, pero se encuentran en [77].

6 CONCLUSIONES Y LÍNEAS FUTURAS

"Siempre he estado convencido de que la única forma de hacer que funcione la inteligencia artificial es hacer el cálculo de manera similar al cerebro humano. Ese es el objetivo que he estado persiguiendo. Estamos progresando, aunque todavía tenemos mucho que aprender sobre cómo funciona realmente el cerebro."

Geoffrey Hinton

Se procede a hacer un repaso de lo visto en el documento sacando algunas conclusiones al respecto. También se comentará brevemente posibles ampliaciones de este proyecto.

6.1 Conclusiones

A través de este proyecto se puede llegar a tener una idea de lo que consiste una cámara basada en eventos (DVS), una cámara que imita la retina humana. Al usar eventos, para trabajar con ellas requiere el uso de algoritmos distintos a los usados en cámaras tradicionales (por ejemplo, hacer uso de las redes neuronales basadas en eventos (SNN)). No obstante, presentan grandes ventajas respecto a las cámaras estándar, en lo referente a su bajo consumo y poca perdida de resolución temporal.

Las SNN son lo más recomendable para trabajar con cámaras de eventos, pues es un sensor que proporciona spikes. Sólo produce un evento cuando un píxel informa de un cambio de brillo significativo. Del mismo modo, la neurona de una SNN sólo produce un spike de salida cuando se produce un número significativo de spikes de entrada en un corto período de tiempo. Debido a su modelo computacional basado en spikes, las SNN pueden procesar la salida de sensores asíncronos basados en eventos sin ningún tipo de preprocesamiento y con una potencia extremadamente baja, a diferencia de las redes neuronales artificiales estándar [66].

El campo de las cámaras de eventos sigue actualmente en desarrollo y todavía hay que mejorar muchos aspectos para que su uso esté más extendido.

El ojo humano posee un mecanismo de foveación, lo cual le permite aumentar la resolución espacial en la zona donde se enfoca (donde se encuentra zona de interés) y reducir en la periferia. Esto permite reducir la cantidad de información necesaria. Las cámaras de eventos no incluyen la foveación originalmente. Puede resultar beneficioso realizar una cámara de eventos que incorpore este mecanismo, pues se reduciría la información innecesaria. Reduciría el coste computacional y, por tanto, reduciría aún más el consumo necesario. Se desea realizar un DVS que integre este mecanismo.

En lo referente al proyecto realizado, se ha modelado a nivel de software lo que realizaría la retina con foveación que se desea fabricar. Se realizó un programa que devuelve como resultado lo que debería dar la retina foveada; permitiendo tener una idea inicial de lo que se obtendría y así interpretar los resultados. Se puede ver cómo afectaría a la retina cambiar determinados parámetros, como el tamaño del cluster de pixeles en la periferia y el umbral. Permite emular lo que se 'vería' con la retina, como medida de control del proceso de fabricación y utilización de la retina.

Los códigos utilizados están desarrollados en Matlab y Python. Los pasos realizados son los siguientes:

1. Obtención de los eventos con la cámara real.
2. Se convierten a fotogramas en escala de grises, es decir, se reconstruye la iluminancia con la que se crearon los eventos. Con la información de luminancia se emulará el comportamiento de la retina incluyendo el mecanismo de foveación.
3. Se interpolan las intensidades de los pixeles de los fotogramas obtenidos, permitiendo crear más imágenes, para aumentar la resolución temporal.
4. Se identifican la zona o zonas donde se encuentran los objetos de interés mediante un algoritmo de detección.
5. Se realiza el cambio de resolución, estableciendo conjuntos de pixeles como uno solo en la periferia y permaneciendo en alta resolución la zona de interés.
6. Se convierte el nuevo conjunto de imágenes modificadas en eventos estableciendo un determinado umbral para su obtención.
7. Luego se utilizaría un programa para representar los eventos obtenidos y así verificar el correcto funcionamiento del modelo.

En lo referente a la determinación de las zonas donde va a tener resolución alta, se utilizó un algoritmo de detección tradicionales de cámaras estándar (trabaja con intensidades de los pixeles); sin embargo, existen algoritmos de detección basados en eventos que pueden resultar de gran utilidad para el objetivo del proyecto.

Durante las simulaciones del modelo, se ha podido verificar los siguientes hechos:

- El periodo utilizado durante la creación de fotogramas a partir de eventos es determinante en la simulación, es decir, cada cuanto tiempo se crea un fotograma a partir de los eventos disponibles desde que se creó el último fotograma. También hay que considerar si el periodo es fijo o variable. En caso de ser variable, espera a que haya una determinada cantidad de eventos para crear la imagen a partir de ellos. El número de imágenes variara según el caso. Cuando se convierte una lista de eventos a fotogramas, lo que se hace es agrupar una determinada cantidad de eventos producidos durante un periodo de tiempo, y a partir del número de eventos

que aparecen en este periodo para cada píxel se calcula la escala de gris de dicho píxel. Así, si se toma un periodo demasiado pequeño, se tendrán pocos eventos y el fotograma reconstruido mostrará una información incompleta (no se visualizan bien los objetos). Sin embargo, si se toma demasiado grande, habrá demasiados eventos y el fotograma reconstruido mostrará cada objeto móvil en distintas posiciones a la vez, lo cual no es deseable. Se verificó que se logra una mejor visualización cuando se establece el periodo variable, en lugar de fijo.

- Si disminuye el tamaño paso de interpolación, es decir, aumenta el número de puntos (intensidades de píxel) intermedios; puede aumentar el número de eventos. Al haber aumentado el número de imágenes, existe más posibilidades de producirse evento.

- Al variar el umbral que determina si se produce evento o no (cuando se crean eventos a partir de imágenes foveadas), se ve afectado el número de eventos. Si disminuye, aumenta el número de eventos producidos, y si aumenta, disminuyen los eventos.

- El tamaño del conjunto (cluster) de pixeles de la periferia afecta a la cantidad de eventos que se produzcan. Si aumenta el tamaño, el número de eventos disminuye (el porcentaje de reducción de eventos aumenta); y si disminuye el tamaño, aumenta.

- En el caso de la grabación de 2 segundos de las cartas, el número de eventos se ha conseguido reducir con respecto al de sin foveación un 70%. Este valor variará según cuanto bajemos la resolución en la periferia, el umbral establecido, la interpolación realizada y el periodo usado entre fotogramas. Estos valores dependen de la grabación que se use, así como de la duración de ésta. Se desea que la cámara produzca la menor cantidad de eventos posibles sin perder la capacidad del reconocimiento visual. En el del 70.06 % se comprobó que todavía se podían distinguir las formas de las figuras de las cartas de manera más o menos aceptable. En lo referente a la grabación del hombre, en del 69.47% seguía habiendo un reconocimiento visual aceptable.

Con respecto al cómo se incluirían los algoritmos realizados en una futura retina en silicio; se ha estado diseñando en el IMSE una versión de la retina que incluye una serie de conexiones entre píxeles mediante unos transistores que hacen de llaves. El objetivo es que, cuando una determinada llave esté activa, las corrientes producidas por los fotodiodos de dos o más píxeles diferentes se integren juntas, como si fueran un único píxel. Esto es bastante sencillo a nivel de la retina, de forma que lo complicado es cómo programar todas esas llaves desde fuera para controlar la resolución en cada zona de la retina, y ahí es donde entra este trabajo. En otras palabras, el algoritmo de foveación no se implementará en la retina, sino que la retina se conectará a un microcontrolador que será el encargado de activar y desactivar llaves para agrupar o desagrupar píxeles. Así que, básicamente, los algoritmos se implementarán en un microcontrolador que irá incluido en la PCB de la retina para comunicarse con ella.

6.2 Líneas futuras

Los próximos pasos del proyecto serían mejorar el algoritmo y hacer diferentes pruebas en software y hardware. Además, una vez realizado el SVD foveado, hay que probar diferentes maneras de controlarlo.

Una posible mejora del algoritmo sería realizar la detección de la zona de interés directamente sobre los eventos, en lugar de trabajar con valores de intensidad de píxel. Para ello existen algoritmos, algo complejos, que permiten hacer este seguimiento de objetos. En el apartado 6.2.1 se explica un poco más sobre esto.

Otra posible ampliación del trabajo sería utilizar la cámara conectándola a una placa System on Chip[27] (SoC), llamada PYNQ-Z2. Se realizaría la programación necesaria en esta placa, no teniendo que conectar la cámara

[27] "*Circuito integrado que incorpora gran parte de los componentes de un ordenador o cualquier otro sistema informático o electrónico*" [127]."*Significa literalmente «sistema en un chip». Su origen se remonta al auge de compañías como Nokia, cuando comenzaron el desarrollo de los primeros teléfonos móviles y era necesario que éstos fueran lo más pequeños y autónomos posible. Por ello, y siguiendo con la idea de miniaturizarlo todo en mente, la tendencia fue la*

al ordenador. En el 6.2.2 se explican los detalles sobre esta placa. Un paso adicional sería desarrollar una SNN que utilice los eventos que se obtengan.

La parte de implementación en hardware la llevó a cabo una compañera del IMSE, la cuál creo un primer modelo de la retina foveada (se ilustra en la Figura 95). En la misma PCB se incorporó un microcontrolador para el sensor, pero necesita de otro aparato externo para programarlo (el cual presentó problemas). Para controlar el sensor, se ha pensado en utilizar un microcontrolador externo (el modelo STM32f103 Nucleo-64 de la empresa STMicroelectronics). En la Figura 96 se muestra cómo se realiza la conexión al DVS.

Figura 95. A la derecha de la placa, se encuentra el sensor de visión basado en eventos con foveación aplicada.

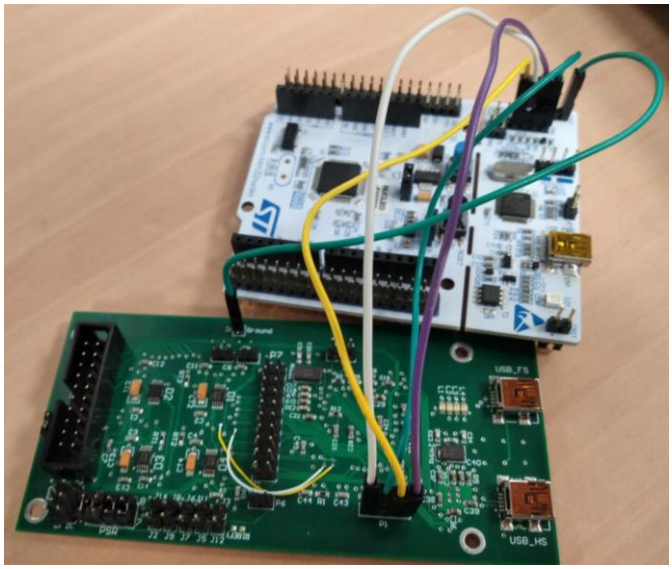


Figura 96. Conexión de sensor de visión con microcontrolador.

6.2.1 Algoritmo de detección de objetos basado en eventos

Puede interesar hacer la detección de objetos en el dominio de los eventos, de forma que nuestro algoritmo vaya haciéndolo todo de forma simultánea, es decir, conforme va generando eventos, va buscando objetos en ellos, y en cuanto encuentra un objeto en movimiento, modifica sobre la marcha la resolución de la generación de eventos.

de implementar un chip que tuviera tantas funciones integradas como fuera posible" [126].*"El nivel más avanzado de la tecnología en el campo de diseño de chips se ha logrado, mediante la implementación de sistemas embebidos basados en SoCs sobre Field Programmable Gate Arrays (FPGA)"* [99].

Esto forma parte de las líneas futuras del proyecto, una vez se pruebe con fotogramas, habría que introducir el algoritmo de *object tracking* a nivel de eventos (en lugar de fotogramas). Es algo que ya se sabe hacer y que se podría integrar con el propio algoritmo que seleccione la resolución de cada parte de las imágenes. Para comprender como se realizaría hay que leer el artículo [93]. También se recomienda ver [94] [95] , donde se muestra un artículo, un video y códigos del Grupo de Percepción y robótica de la Universidad de Maryland. Presenta un enfoque de realización del seguimiento de objetos con cámaras asíncronas. Enseñan una representación del flujo de eventos que les permite utilizar información sobre el componente dinámico (temporal) del flujo de eventos. "*La geometría 3D del flujo de eventos se aproxima con un modelo paramétrico para compensar el movimiento de la cámara (sin seguimiento de características ni cálculo explícito del flujo óptico), y luego se detectan los objetos en movimiento que no se ajustan al modelo en un proceso iterativo*" [94]. También puede resultar de interés consultar el artículo de cómo detectar objetos para ser esquivados por un dron: [96] [97]. En el ANEXO C se muestra un ejemplo de una función que hace *object tracking* en Matlab.

6.2.2 PYNQ-Z2

Una línea futura era utilizar la cámara de eventos junto con la placa PYNQ-Z2 (Figura 97). La placa de desarrollo se basa en Xilinx Zynq XC7Z020 System on Chip (SoC) (Figura 98). "*Zynq es un SoC basado en el procesador ARM Cortex-A9 de doble núcleo (sistema de procesamiento llamado o sistema de procesamiento PS), estructura FPGA integrada (llamada lógica programable o PL). El subsistema PS incluye muchos periféricos dedicados (controladores de memoria, USB, UART, IIC, SPI, etc.), y se pueden extender por IP*[28] *adicional de hardware en la anulación de PL*" [98].El PS y el PL se muestran de forma esquemática en la Figura 99. En general, los elementos que conforman un System on Chip son: procesador programable, memorias on chip, unidades de aceleración implementadas en hardware, interfaces con dispositivos periféricos … [99].

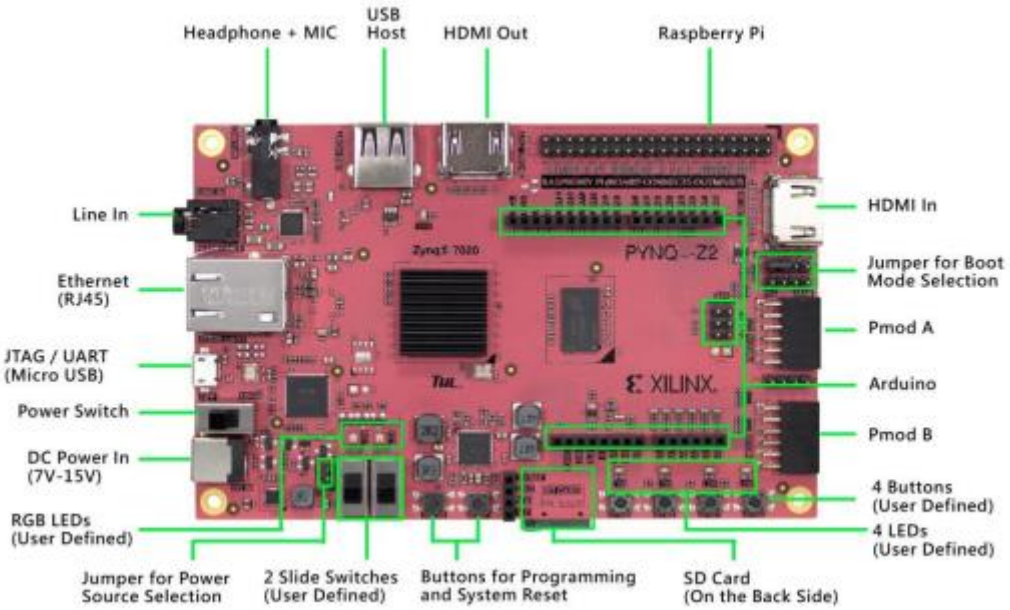


Figura 97. La placa TUL PYNQ ™ -Z2, basada en Xilinx Zynq SoC, está diseñada para admitir el marco PYNQ (Python Productivity for Zynq) y el desarrollo de sistemas integrados. Fuente: [100].

[28] *"El sistema de procesamiento IP es la interfaz de software del sistema de procesamiento Zynq"" El contenedor IP del sistema de procesamiento actúa como una conexión lógica entre el PS y el PL mientras lo ayuda a integrar IP personalizadas e integradas con el sistema de procesamiento utilizando el integrador IP Vivado"* [98].

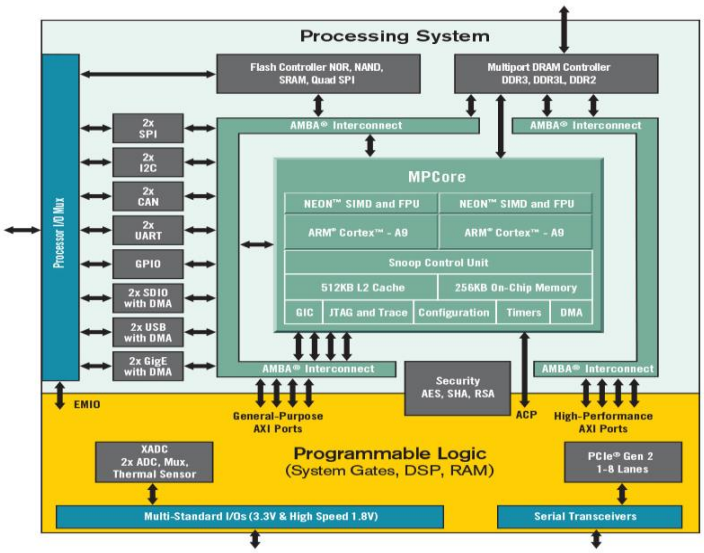


Figura 98. Esquema Zynq-7000 de Xilinx. Fuente: [101].

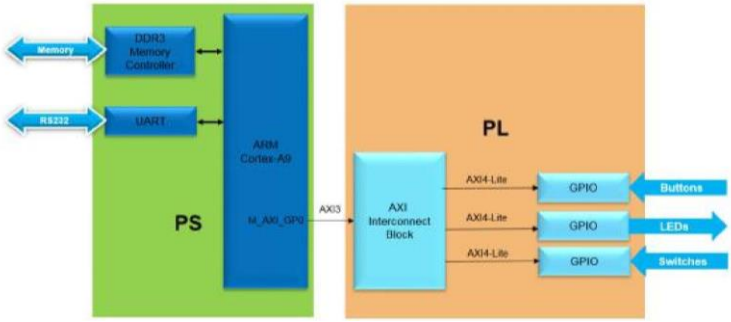


Figura 99. El Pynq-Z2 incluye "*el brazo Cortex-A9 PS y tres IP de GPIO estándar para conectar el LED, los botones y los interruptores de la placa*". Fuente: [98].

"*La PYNQ-Z2 está diseñada para soportar PYNQ, un marco de trabajo de código abierto que permite a los programadores integrados explorar las posibilidades de los SoCs Xilinx Zynq sin tener que diseñar circuitos lógicos de programación. Gracias a la lógica programable y al avanzado procesador Arm de Zynq, los diseñadores pueden crear potentes aplicaciones integradas. Los SoC pueden programarse en Python y el código puede desarrollarse y probarse directamente en el PYNQ-Z2. Los circuitos lógicos programables se importan como bibliotecas de hardware y se programan a través de APIs, básicamente de la misma manera que se importan y programan las bibliotecas de software"* [102]. "*Los diseñadores pueden aprovechar los beneficios de la lógica programable y los microprocesadores para construir sistemas electrónicos más capaces"* [103].

"La placa PYNQ-Z2 cuenta con conectividad Ethernet, entrada y salida HDMI, entrada MIC, salida de audio, una interfaz Arduino, una interfaz Raspberry Pi, 2 cabezales Pmod, un LED de usuario, un pulsador de usuario y un interruptor de usuario. La placa está diseñada para ser fácilmente extensible con Pmod, Arduino y periféricos, así como pines GPIO de propósito general" [102].

Se explica el procedimiento para utilizar esta placa en el ANEXO D**.** Se recomienda encarecidamente mirar el siguiente tutorial de la placa PYNQ-Z2: [104] [105] [106]. Al ejecutar el ejemplo, tener presente que "*el módulo*

pynq.gpio es un controlador para leer y escribir pines PS GPIO en una placa. Los pines PS GPIO no están conectados al PL" [107].

Para saber más sobre la tarjeta Pynq, se recomienda también leer la página web [103]. El datasheet se encuentra en [100]. Algunos videotutoriales que se recomienda consultar son: [108] [109] [110]. Otra documentación recomendada de PYNQ y Vivado sería: [111] [112] [113] [114] [115] [103].

REFERENCIAS

[1] «grlum.dpe.upc.edu,» [En línea]. Available: https://grlum.dpe.upc.edu/manual/fundamentosIluminacion-laVision.php. [Último acceso: 2022].

[2] V. Lendave, «A Tutorial on Spiking Neural Networks for Beginners,» 2021. [En línea]. Available: https://analyticsindiamag.com/a-tutorial-on-spiking-neural-networks-for-beginners/. [Último acceso: 2022].

[3] «Event camera,» Wikipedia, [En línea]. Available: https://en.wikipedia.org/wiki/Event_camera. [Último acceso: Marzo 2022].

[4] C. Gu, E. Learned-Miller, D. Sheldon, G. Gallego y P. Bideau, «The Spatio-Temporal Poisson Point Process:A Simple Model for the Alignment of Event Camera Data,» *ICCV*, 2021.

[5] G. Gallego, «guillermogallego,» [En línea]. Available: https://sites.google.com/view/guillermogallego/research/event-based-vision. [Último acceso: 2022].

[6] Unitectra, «Dynamic Vision Sensor (DVS),» 2016. [En línea]. Available: https://www.youtube.com/watch?v=0ZEM57DZJes. [Último acceso: 2022].

[7] G. Gallego, J. Lund, E. Mueggler, H. Rebecq, T. Delbruck y D. Scaramuzza, «Event-based, 6-DOF Camera Tracking from Photometric Depth Maps,» *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.

[8] D. Scaramuzza, «Tutorial on Event-based Cameras,» 2020. [En línea]. Available: https://rpg.ifi.uzh.ch/research_dvs.html.

[9] T. Delbruck, «Silicon retina technology,» 2017. [En línea]. Available: http://ims.unipv.it/CASWS2017/Slides/Delbruck.pdf. [Último acceso: 2022].

[10] T. Delbruck, «Event Cameras,» 2020. [En línea]. Available: https://drive.google.com/file/d/1w7easbLhcHh-BoMFQq1RRt5RAjGkfk7u/view. [Último acceso: 2022].

[11] Y. Zhang, Y. Zhao, H. Lv, Y. Feng, H. Liu y C. Han, «Adaptive Slicing Method of the Spatiotemporal Event Stream Obtained from a Dynamic Vision Sensor,» *Sensors*, vol. 22, nº 7, p. 2614, 2022.

[12] R. a. P. Group, «The Event-Camera Dataset and Simulator: Event-based Data for Pose Estimation, Visual Odometry, and SLAM,» 2017. [En línea]. Available: https://rpg.ifi.uzh.ch/davis_data.html. [Último acceso: 2022].

[13] T. Delbruck y M. Lang, «Robotic Goalie with 3ms Reaction Time at 4% CPU Load Using Event-Based Dynamic Vision Sensor,» *Frontiers in neuroscience*, vol. 7, p. 223, 2013.

[14] H. Kim, S. Leutenegger y A. J. Davison, «Real-Time 3D Reconstruction and 6-DoF Tracking with an Event Camera,» de *European Conference on Computer Vision*, Imperial College London, UK, 2016.

[15] G. Gallego, T. Delbruck, G. Orchard, C. Bartolozzi, B. Taba, A. Censi, S. Leutenegger, A. J. Davison, J. Conradt, K. Daniilidis y D. Scaramuzza, «Event-based Vision: A Survey,» *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020.

[16] «imago-technologies,» [En línea]. Available: https://imago-technologies.com/smart-event-based-camera/. [Último acceso: Marzo 2022].

[17] U. R. a. P. Group, «Event-based, 6-DOF Pose Tracking for High-Speed Maneuvers using a Dynamic Vision Sensor,» [En línea]. Available: https://www.youtube.com/watch?v=LauQ6LWTkxM&t=30s&ab_channel=UZHRoboticsandPerceptionGroup. [Último acceso: 04 Marzo 2022].

[18] Y. Zhou, G. Gallego y S. Shen, «Event-based Stereo Visual Odometry,» 2021.

[19] S. K. MOORE, «Event-Based Camera Chips Are Here, What's Next?,» IEEE, 12 Octubre 2021. [En línea]. Available: https://spectrum.ieee.org/event-based-camera-chips. [Último acceso: Marzo 2022].

[20] A. Rodnitzky, «Cámaras de eventos: ¿dónde están ahora?,» 30 Diciembre 2020. [En línea]. Available: https://medium.com/tangram-visions/event-cameras-where-are-they-now-293343754bfd. [Último acceso: Marzo 2022].

[21] U. R. a. P. Group, «Event Cameras: Opportunities and the Road Ahead (CVPR 2020),» [En línea]. Available: https://www.youtube.com/watch?v=6Sn9-M7qXLk. [Último acceso: Marzo 2022].

[22] Sony, «youtube: Sony | Event-based Vision Sensor (EVS) to detect only changes in moving subjects -Full ver.-,» [En línea]. Available: https://www.youtube.com/watch?v=6xOmo7Ikwzk. [Último acceso: 2022].

[23] Neuromorphic_Workshop_Telluride, «youtube: Tobi Delbruck - Event Camera Tutorial 2020 - v4 1,» [En línea]. Available: https://www.youtube.com/watch?v=D6rv6q9XyWU. [Último acceso: 2022].

[24] UZH_Robotics_and_Perception_Group, «youtube: Event-based Vision for Autonomous High-Speed Robotics,» [En línea]. Available: https://www.youtube.com/watch?v=VbvJGQFCvGs. [Último acceso: 2022].

[25] W. Gerstner, W. M. Kistler, R. Naud y L. Paninski, «neuronaldynamics,» Cambridge University , [En línea]. Available: https://neuronaldynamics.epfl.ch/online/Ch1.S3.html.

[26] J. L. Lobo, «Spiking Neural Networks in Stream Learning scenarios,» 2020. [En línea]. Available: https://towardsdatascience.com/spiking-neural-networks-in-stream-learning-scenarios-7bf2112b0c35. [Último acceso: 2022].

[27] F. Corradi, G. Adriaans y S. Stuijk, «Gyro: A Digital Spiking Neural Network Architecture for Multi-Sensory Data Analytics,» de *Drone Systems Engineering and Rapid Simulation and Performance Evaluation: Methods and Tools Proceedings*, Budapest, Hungary, 2021.

[28] G. Chen, H. Cao, C. Ye, Z. Zhang, X. Liu, X. Mo, Z. Qu, J. Conradt, F. Röhrbein y A. Knoll, «Multi-Cue Event Information Fusion for Pedestrian Detection With Neuromorphic Vision Sensors,» *Frontiers in Neurorobotics.*, vol. 13, 2019.

[29] R. V. W. Putra y M. Shafique, «FSpiNN: An Optimization Framework for Memory- and Energy-Efficient Spiking Neural Networks.,» *IEEE TRANS. ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS*, 2020.

[30] Shikhargupta, «github: Spiking-Neural-Network,» 2018. [En línea]. Available: https://github.com/Shikhargupta/Spiking-Neural-Network. [Último acceso: 2022].

[31] D. Soni, «Spiking Neural Networks, the Next Generation of Machine Learning,» 2018. [En línea]. Available: https://towardsdatascience.com/spiking-neural-networks-the-next-generation-of-machine-learning-84e167f4eb2b. [Último acceso: 2022].

[32] M. T. Rivera, «Modelos Neuronales Memresistivos,» INSTITUTO POTOSINO DE INVESTIGACIÓN CIENTÍFICA Y TECNOLÓGICA, 2016.

[33] «wikipedia: red neuronal de impulsos,» [En línea]. Available: https://es.wikipedia.org/wiki/Red_neuronal_de_impulsos. [Último acceso: 2022].

[34] T. Iakymchuk, «Spiking Neural Networks models,» Universitat de Valencia, Valencia, 2017.

[35] «hmong: Receptive_field,» [En línea]. Available: https://hmong.es/wiki/Receptive_field. [Último acceso: 2022].

[36] Xanky, «Fisiología del sistema visual,» 2016. [En línea]. Available: http://elbuenpresagio.blogspot.com/2016/03/. [Último acceso: 2022].

[37] Shikhargupta, «github: Spiking-Neural-Network,» 2016. [En línea]. Available: https://github.com/Shikhargupta/Spiking-Neural-Network/tree/master/receptive_field. [Último acceso: 2022].

[38] «hmong.es: Spiking_neuron,» [En línea]. Available: https://hmong.es/wiki/Spiking_neuron. [Último acceso: 2022].

[39] « CS231n: Convolutional Neural Networks for Visual Recognition,» [En línea]. Available: https://cs231n.github.io/convolutional-networks/#conv. [Último acceso: 2022].

[40] L. Camuñas-Mesa, C. Zamarreño-Ramos, A. Linares-Barranco, A. J. Acosta-Jiménez, T. Serrano-Gotarredona y B. Linares-Barranco, «An Event-Driven Multi-Kernel Convolution Processor Module for Event-Driven Vision Sensors,» *IEEE Journal of Solid-State Circuits*, vol. 47, nº 2, pp. 504-517, 2012.

[41] L. Camuñas-Mesa, A. Acosta-Jiménez, C. Zamarreño-Ramos, T. Serrano-Gotarredona y B. Linares-Barranco, «A 32 32 Pixel Convolution Processor Chip for Address Event Vision Sensors With 155 ns Event Latency and 20 Meps Throughput,» *IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS*, vol. 58, nº 4, pp. 777-790, 2011.

[42] U. Rancon, J. Cuadrado-Anibarro, B. R. Cottereau y T. Masquelier, «StereoSpike: Depth Learning with a Spiking Neural Network,» de *CerCo CNRS UMR 5549, Universite Toulouse III*, Toulouse, France, 2021.

[43] urancon, «github: StereoSpike,» 2021. [En línea]. Available: https://github.com/urancon/StereoSpike. [Último acceso: 2022].

[44] V. LYASHENKO, «Basic Guide to Spiking Neural Networks for Deep Learning,» [En línea]. Available: https://cnvrg.io/spiking-neural-networks/. [Último acceso: 2022].

[45] N. Risi, E. Calabrese y G. Indiveri, «Instantaneous Stereo Depth Estimation of Real-World Stimuli with a Neuromorphic Stereo-Vision Setup,» *Giacomo*, 2021.

[46] G. Haessig, X. Berthelon, S.-H. Ieng y R. Benosman, «A Spiking Neural Network Model of Depth from Defocus for Event-based Neuromorphic Vision,» *Scientific Reports*, vol. 9, p. 3744, 2019.

[47] A. T. GALÁN, «REPRESENTACION DE IMÁGENES MEDIANTE FUNCIONES DE GÁBOR. MODELADO DEL SISTEMA VISUAL Y ANALISIS DE TEXTURAS,» Universidad Complutense Madrid, Madrid, 1992.

[48] K. Balasundar, «allaboutrobotix,» 2020. [En línea]. Available: https://www.allaboutrobotix.com/spiking-neural-networks/. [Último acceso: 2022].

[49] C. M. Parameshwara, S. Li, C. Fermuller, N. J. Sanket, M. S. Evanusa y Y. Aloimonos, «SpikeMS: Deep Spiking Neural Network for Motion Segmentation,» de *IROS*, University of Maryland, College Park, 2021.

[50] M. Pfeiffer y T. Pfeil, «Deep Learning With Spiking Neurons: Opportunities and Challenges,» *Frontiers in Neuroscience*, 2018.

[51] WaterlooAI, «youtube: Spiking Neural Networks for More Efficient AI Algorithms,» [En línea]. Available: https://www.youtube.com/watch?v=PeW-TN3P1hk. [Último acceso: 2022].

[52] A. Tavanaei, M. Ghodrati, S. R. Kheradpisheh, T. Masquelier y A. Maida, «Deep Learning in Spiking Neural Networks,» 2019. [En línea]. Available: https://arxiv.org/pdf/1804.08150.pdf.

[53] Y. Kim y P. Panda, «Visual explanations from spiking neural networks using inter-spike intervals.,» *Sci Rep,* vol. 11, 2021.

[54] S. Dutta, C. Schafer, J. Gómez, K. Ni, S. Joshi y S. Datta, «Supervised Learning in All FeFET-Based Spiking Neural Network: Opportunities and Challenges.,» *Frontiers in Neuroscience,* vol. 14, p. 634, 2020.

[55] H. Hazan, D. J. Saunders, H. Khan, D. Patel, D. T. Sanghavi, H. T. Siegelmann y R. Kozma, «BindsNET: A Machine Learning-Oriented Spiking Neural Networks Library in Python,» *Frontiers in Neuroinformatics,* vol. 12, 2018.

[56] D. MÜLLER-KOMOROWSKA, «Spiking Neuronal Networks in Python,» 2021. [En línea]. Available: https://danielmuellerkomorowska.com/2021/02/22/spiking-neuronal-networks-in-python/. [Último acceso: 2022].

[57] M. Mozafari, M. Ganjtabesh, A. Nowzari-Dalini y T. Masquelier, «SpykeTorch: Efficient Simulation of Convolutional Spiking Neural Networks with at most one Spike per Neuron,» *Frontiers in Neuroscience,* 2019.

[58] «wikipedia: Spiking_neural_network,» [En línea]. Available: https://en.wikipedia.org/wiki/Spiking_neural_network. [Último acceso: 2022].

[59] uzh-rpg, «github: Event-based Vision Resources,» 2021. [En línea]. Available: https://github.com/uzh-rpg/event-based_vision_resources. [Último acceso: 2022].

[60] «tobii,» [En línea]. Available: https://vr.tobii.com/sdk/learn/foveation/. [Último acceso: 2022].

[61] Ron, «directx: A 20Megapixel VR display and Foveated rendering,» 2018. [En línea]. Available: http://directx.com/2018/01/20megapixel-foveated/. [Último acceso: 2022].

[62] «tobii: Principles of Foveation,» [En línea]. Available: https://vr.tobii.com/sdk/learn/foveation/principles/. [Último acceso: 2022].

[63] J. Felthan, «uploadvr,» 2019. [En línea]. Available: https://uploadvr.com/tobii-foveated-rendering/. [Último acceso: 2022].

[64] R. Radkowski y S. Raul, «Impact of Foveated Rendering on Procedural Task Training,» *Springer,* vol. 11574, p. 258–267, 2019.

[65] H. Lukanov, P. König y G. Pipa, «Biologically Inspired Deep Learning Model for Efficient Foveal-Peripheral Vision,» *Frontiers in Computational Neuroscience,* vol. 22, 2021.

[66] M. Gehrig, S. B. Shrestha, D. Mouritzen y D. Scaramuzza, «Event-Based Angular Velocity Regression with Spiking Networks,» de *IEEE International Conference on Robotics and Automation (ICRA)*, Paris, 2020.

[67] UZH_Robotics_and_Perception_Group, «youtube: Event-Based Angular Velocity Regression with Spiking Networks (ICRA20 Video Pitch),» [En línea]. Available: https://www.youtube.com/watch?v=cffwH41ReF4. [Último acceso: 2022].

[68] @GuillermoFdez98, «github: Video pipeline for event-based sensors,» 2021. [En línea]. Available: https://github.com/GuillermoFdez98/Video-pipeline-for-event-based-sensors. [Último acceso: 2022].

[69] S.-C. Liu y T. Delbruck, «sensors.ini.uzh.ch,» [En línea]. Available: http://sensors.ini.uzh.ch/home.html. [Último acceso: 2022].

[70] V. Koifman, «image-sensors-world,» 2017. [En línea]. Available: http://image-sensors-world.blogspot.com/2017/05/dynamic-vision-sensor-presenation.html. [Último acceso: 2022].

[71] «imse-cnm,» [En línea]. Available: http://www2.imse-cnm.csic.es/neuromorphs/. [Último acceso: 2022].

[72] I. d. N. Zurich, «User Guide - jAER,» [En línea]. Available: https://docs.google.com/document/d/1fb7VA8tdoxuYqZfrPfT46_wiT1isQZwTHgX8O22dJ0Q/edit. [Último acceso: 2022].

[73] T. Delbruck, «youtube,» 2021. [En línea]. Available: https://www.youtube.com/playlist?list=PLVtZ8f-q0U5hD9KOM4OZ1lixhwupj9uOm. [Último acceso: 2022].

[74] SensorsINI, «github,» 2021. [En línea]. Available: https://github.com/SensorsINI/jaer/. [Último acceso: 2022].

[75] T. Delbruck, «youtube: 4 tracking objects in jaer,» [En línea]. Available: https://www.youtube.com/watch?v=5I6haFXVuD8&list=PLVtZ8f-q0U5hD9KOM4OZ1lixhwupj9uOm&index=4. [Último acceso: 2022].

[76] H. Rebecq, R. Ranftl, V. Koltun y D. Scaramuzza, «Events-to-Video: Bringing Modern Computer Vision to Event Cameras,» de *IEEE Conf. Comput. Vis. Pattern Recog. (CVPR)*, 2019.

[77] H. Rebecq, R. Ranftl, V. Koltun y D. Scaramuzza, «github,» [En línea]. Available: https://github.com/cedric-scheerlinck/rpg_e2vid/tree/cedric/firenet. [Último acceso: 2022].

[78] H. Rebecq, R. Ranftl, V. Koltun y D. Scaramuzza, «High Speed and High Dynamic Range Video with an Event Camera,» *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE*, 2019.

[79] C. Scheerlinck, H. Rebecq, N. B. D. Gehrig, R. Mahony y D. Scaramuzza, «Fast Image Reconstruction with an Event Camera,» de *IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2020.

[80] R. D. Q. Laura, «Métodos Interpolación con MatLab,» [En línea]. Available: https://docplayer.es/9890582-Metodos-interpolacion-con-matlab-el-comando-interp1-el-comando-interp1-se-emplea-para-interpolar-una-serie-de-datos-el-formato-de-este-comando-es.html. [Último acceso: 2022].

[81] «Aproximación de funciones: Interpolación.,» [En línea]. Available: https://estadistica-dma.ulpgc.es/FCC/05-6-Interpolacion.html. [Último acceso: 2022].

[82] A. S. Ramos, «youtube: Interpolación en Matlab,» [En línea]. Available: https://www.youtube.com/watch?v=Ci4nZTqTTjw. [Último acceso: 2022].

[83] «sc.ehu: Interpolación con MATLAB. Extrapolación,» [En línea]. Available: http://www.sc.ehu.es/sbweb/fisica3/datos/ajuste/interpol_matlab.html. [Último acceso: 2022].

[84] I. Ortiz Ramírez, «Machine Learning para MI-BCI orientada al procesado de las señales EEG en tiempo real,» Universidad de Sevilla, Sevilla, 2021.

[85] D. P. Hidalgo, «Desarrollo de una aplicación para la detección y seguimiento de objetos. Aplicación a vehículos,» Universidad Politécnica de Valencia.

[86] M. S. N. RODRIGUEZ y M. A. S. FERNÁNDEZ, «SEGUIMIENTO EN TIEMPO REAL DE OBJETOS SOBRE SECUENCIA DE IMAGENES EMPLEANDO DISPOSITIVOS DE LÓGICA PROGRAMABLE,» UNIVERSIDAD TECNOLOGICA DE PEREIRA, PEREIRA, COLOMBIA, 2010.

[87] E. C. Laso, «Tracking automático de objetos en secuencias de imágenes usando Filtro de Partículas,» UNIVERSIDAD DE EXTREMADURA, Extremadura.

[88] R. Alvarez-Torrico, «tecbolivia: Aplicación de Seguimiento de Objetos con Drones: Usando OpenCV, MAVSDK y PX4,» [En línea]. Available: https://tecbolivia.com/index.php/articulos-y-tutoriales-microcontroladores/104-aplicacion-de-seguimiento-de-objetos-con-drones. [Último acceso: 2022].

[89] «rroji,» [En línea]. Available: https://www.rroij.com/articles-images/IJAREEIE-05-g008.html. [Último acceso: 2022].

[90] J. Redolfi, «Aplicación en agricultura de precisión de esquemas actuales de reconocimiento visual,» Universidad Nacional de Córdoba, Córdoba, 2018.

[91] T. Serrano-Gotarredona y B. Linares-Barranco, «A 128x28 1.5% Contrast Sensitivity 0.9% FPN 3 µs Latency 4 mW Asynchronous Frame-Free Dynamic Vision Sensor Using Transimpedance Preamplifiers,» *IEEE JOURNAL OF SOLID-STATE CIRCUITS*, vol. 48, nº 3, pp. 827-838, 2013.

[92] simbamford, «github,» 2019. [En línea]. Available: https://github.com/simbamford/AedatTools. [Último acceso: 2022].

[93] M. Litzenberger, D. B. C. Posch y A. Belbachir, «EMBEDDED VISION SYSTEM FOR REAL-TIME OBJECT TRACKING USING AN ASYNCHRONOUS TRANSIENT VISION SENSOR,» de *IEEE*, Teton National Park, WY, USA, 2006.

[94] A. Mitrokhin, C. Fermuller, C. Parameshwara y Y. Aloimonos, «Event-based Moving Object Detection and Tracking,» University of Maryland, College Park, 2020. [En línea]. Available: http://prg.cs.umd.edu/BetterFlow.html. [Último acceso: 2022].

[95] PRGUMD, «youtube: Event-based Moving Object Detection and Tracking (IROS 2018),» [En línea]. Available: https://www.youtube.com/watch?v=UCAJi0ZFaZ8. [Último acceso: 2022].

[96] H. L. S. W. Botao He, D. Wang, Z. Zhang, Q. Dong, C. Xu y F. Gao, «FAST-Dynamic-Vision: Detection and Tracking Dynamic Objects with Event and Depth Sensing,» 2021. [En línea]. Available: https://arxiv.org/abs/2103.05903. [Último acceso: 2022].

[97] F. Gao, «youtube: FAST-Dynamic-Vision: Detection and Tracking Dynamic Objects with Event and Depth Sensing,» 2021. [En línea]. Available: https://www.youtube.com/watch?v=QPpwppeE_x0. [Último acceso: 2022].

[98] «programmerclick: PYNQ-Z2 Primer conocimiento,» [En línea]. Available: https://programmerclick.com/article/77472976636/. [Último acceso: 2022].

[99] S. Julio Cadena, L. Gabriel Mollocana, H. Ortiz y V. Vanessa Vargas, «Diseño de Hardware y Software de Systems on Chip empleando tecnología Xilinx EDK,» *MASKAY*, vol. 2, nº 1, pp. 39-48, 2012.

[100] xilinx, «farnell,» [En línea]. Available: https://www.farnell.com/datasheets/2632678.pdf. [Último acceso: 2022].

[101] «xilinx,» [En línea]. Available: https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html. [Último acceso: 2022].

[102] DFRobot, «mouser,» [En línea]. Available: https://www.mouser.es/new/dfrobot/dfrobot-pynqz2-dev-board/. [Último acceso: 2022].

[103] Xilinx, «pynq.io,» [En línea]. Available: http://www.pynq.io/. [Último acceso: 2022].

[104] U. Farooq, «A PYNQ-Z2 Guide for Absolute Dummies — Part I: Fun with LEDs and Switches,» 2020. [En línea]. Available: https://blog.umer-farooq.com/a-pynq-z2-guide-for-absolute-dummies-part-i-fun-with-leds-and-switches-47dd76abf9a9. [Último acceso: 2022].

[105] U. Farooq, «A PYNQ-Z2 Guide for Absolute Dummies — Part II: Let's Burn some Verilog code,» 2020. [En línea]. Available: https://blog.umer-farooq.com/a-pynq-z2-guide-for-absolute-dummies-part-ii-using-verilog-and-vivado-to-burn-code-on-pynq-d856f79948b1. [Último acceso: 2022].

[106] U. Farooq, «A PYNQ-Z2 Guide for Absolute Dummies — Part III: Tick Tock, Using FPGA Clock,» 2021. [En línea]. Available: https://blog.umer-farooq.com/a-pynq-z2-guide-for-absolute-dummies-part-iii-tick-tock-using-fpga-clock-33a34ef3f51a. [Último acceso: 2022].

[107] Xilinx, «pynq.readthedocs.io,» 2021. [En línea]. Available: https://pynq.readthedocs.io/en/v2.7.0/pynq_package/pynq.gpio.html. [Último acceso: 2022].

[108] M. T. Limited, «youtube: Hello world tutorial vivado,» [En línea]. Available: https://www.youtube.com/watch?v=nUzq1KL6J-c. [Último acceso: 2022].

[109] J. Llanos, «youtube:Vivado Tutorial 1 Crear un proyecto,» 2019. [En línea]. Available: https://www.youtube.com/watch?v=YH9PR8ra384. [Último acceso: 2022].

[110] C. McCabe, «youtube:Set up the PYNQ-Z2 board from TUL to run PYNQ,» [En línea]. Available: https://www.youtube.com/watch?v=RiFbRf6gaK4&t=486s. [Último acceso: 2022].

[111] Xilinx, «pynq.readthedocs.io,» [En línea]. Available: https://pynq.readthedocs.io/en/v2.7.0/overlay_design_methodology/pspl_interface.html. [Último acceso: 2021].

[112] Xilinx, «pynq.readthedocs.io,» 2021. [En línea]. Available: https://pynq.readthedocs.io/en/v2.7.0/pynq_overlays/pynqz2.html. [Último acceso: 2022].

[113] cathalmccabe, «PYNQ: PYTHON PRODUCTIVITY,» 2021. [En línea]. Available: https://discuss.pynq.io/t/tutorial-using-a-new-hardware-design-with-pynq-axi-gpio/146. [Último acceso: 2022].

[114] cathalmccabe, «Tutorial: Creating a hardware design for PYNQ,» 2021. [En línea]. Available: https://discuss.pynq.io/t/tutorial-creating-a-hardware-design-for-pynq/145. [Último acceso: 2022].

[115] J. Cumps, «element14 : Add Pynq-Z2 board to Vivado,» 2021. [En línea]. Available: https://community.element14.com/technologies/fpga-group/b/blog/posts/add-pynq-z2-board-to-vivado. [Último acceso: 2022].

[116] «git,» [En línea]. Available: https://git-scm.com/download/win. [Último acceso: 2022].

[117] «oracle,» [En línea]. Available: https://www.oracle.com/java/technologies/downloads/#java8-windows. [Último acceso: 2022].

[118] «ant,» [En línea]. Available: https://ant.apache.org/bindownload.cgi. [Último acceso: 2022].

[119] «/maker-hub.georgefox.edu,» 2021. [En línea]. Available: https://maker-hub.georgefox.edu/wiki/Xilinx_Vivado. [Último acceso: 2022].

[120] Xilinx, «pynq.readthedocs.io,» 2021. [En línea]. Available: https://pynq.readthedocs.io/en/v2.7.0/pynq_package/pynq.mmio.html. [Último acceso: 2022].

[121] «tulembedded.com,» [En línea]. Available: https://www.tulembedded.com/FPGA/ProductsPYNQ-Z2.html. [Último acceso: 2022].

[122] «digilent,» [En línea]. Available: https://digilent.com/reference/programmable-logic/guides/installing-vivado-and-sdk. [Último acceso: 2022].

[123] C. McCabe, «youtube: PYNQ example of controlling IP using GPIO,» [En línea]. Available: https://www.youtube.com/watch?v=UBsCNPWudww. [Último acceso: 2022].

[124] J. LUCAS, «El Rango Dinámico Explicado de la Manera Más Sencilla,» dzoom, [En línea]. Available: https://www.dzoom.org.es/el-rango-dinamico-por-que-nuestra-camara-no-capta-lo-que-ven-nuestros-ojos-ahmf31-dia12/. [Último acceso: Marzo 2022].

[125] D. CSV, «youtube: ¡Redes Neuronales CONVOLUCIONALES! ¿Cómo funcionan?,» [En línea]. Available: https://www.youtube.com/watch?v=V8j1oENVz00&t=5s. [Último acceso: 2022].

[126] R. Alonso, «hardzone: En hardware se usa mucho el término SoC pero, ¿sabes qué es?,» 2020. [En línea]. Available: https://hardzone.es/reportajes/que-es/soc-caracteristicas-hardware/. [Último acceso: 2022].

[127] M. J. Bellido Díaz, «Introducción al Diseño de SoC (Systems On Chip),» 2017. [En línea]. Available: http://www.dte.us.es/docencia/master/micr/soc-basados-en-sistemas-abiertos-socbsa/temas/SoCIntro.

[128] V. M. Ruiz, «Descenso por gradiente (Gradient descent),» 2016. [En línea]. Available: https://turing.iimas.unam.mx/~ivanvladimir/posts/gradient_descent/. [Último acceso: 2022].

[129] «wikipedia: prealimentación,» [En línea]. Available: https://es.wikipedia.org/wiki/Prealimentaci%C3%B3n. [Último acceso: 2022].

[130] C. Scheerlinck, «cedricscheerlinck.com,» [En línea]. Available: https://www.cedricscheerlinck.com/firenet.

[131] UZH_Robotics_and_Perception_Group, «youtube: Rapid, Dynamic Obstacle Avoidance with an Event-based Camera,» [En línea]. Available: https://www.youtube.com/watch?v=sbJAi6SXOQw. [Último acceso: 2022].

[132] X. M. Posadas, «Anatomia del sistema nervioso y organos de los sentidos,» [En línea]. Available: https://sites.google.com/site/xmpanatomy/4-3-sinapsis-1. [Último acceso: 2022].

[133] T. Iakymchuk, «Spiking Neural Networks models targeted for implementation on Reconfigurable Hardware,» 2017. [En línea]. Available: https://roderic.uv.es/handle/10550/60934?show=full. [Último acceso: 2022].

[134] «ediciones-eni: El sesgo, una neurona particular,» [En línea]. Available: https://www.ediciones-eni.com/open/mediabook.aspx?idR=cb2c5aadd6799b3bffe4bc6930c2faee. [Último acceso: 2022].

[135] Azure, «Ajuste de hiperparámetros de un modelo (v2),» 2022. [En línea]. Available: https://docs.microsoft.com/es-es/azure/machine-learning/how-to-tune-hyperparameters. [Último acceso: 2022].

[136] «hmong.es: Recurrent_neural_networks,» [En línea]. Available: https://hmong.es/wiki/Recurrent_neural_networks. [Último acceso: 2022].

[137] «wikipedia: Feedforward_neural_network,» [En línea]. Available: https://en.wikipedia.org/wiki/Feedforward_neural_network. [Último acceso: 2022].

[138] D. J. Matich, «Redes Neuronales: Conceptos Básicos y Aplicaciones,» Universidad Tecnológica Nacional – Facultad Regional Rosario , 2001.

[139] «wikipedia: Teoría hebbiana,» [En línea]. Available: https://es.wikipedia.org/wiki/Teor%C3%ADa_hebbiana#:~:text=Hoy%20en%20d%C3%ADa%2C%20el%20t%C3%A9rmino,relaci%C3%B3n%20posible%20entre%20los%20nodos.. [Último acceso: 2022].

[140] D. Rodríguez, «analyticslane: ¿Cuál es la diferencia entre parámetro e hiperparámetro?,» 16 Diciembre 2019. [En línea]. Available: https://www.analyticslane.com/2019/12/16/cual-es-la-diferencia-entre-parametro-e-hiperparametro/. [Último acceso: 2022].

ANEXO A

A continuación, se explicará cómo hay que instalar el programa jAER en *Windows 10 64-bit*. Antes que nada, se necesita tener instalado *git* en el sistema, para instalar: [116]. También hay que tener *java 1.8 de 64 bits*, se puede descargar desde [117] (se necesita crear una cuenta de *Oracle*). Asegurarse de descargar la versión de *64 bits jdk-8u321-windows-x64.exe*. Cuando se hayas instalado, se debe verificar que haya una carpeta *jdk* y otra *jre* en *C:\Program Files\Java*. Luego se crea una carpeta donde quedará la instalación de jAER, en cualquier ubicación. En este ejemplo se llamará *'jaer-dist'* y desde la consola hay que ir a la ubicación de la carpeta y ejecutar:

```
cd jaer-dist

git init

git remote add origin https://github.com/SensorsINI/jaer.git

git fetch --depth=1

git checkout -t origin/master -f
```

Ahora se procede a compilar el jAER, y eso se hace con un programa llamado *'Ant'*. Se descarga en [118], el *zip* con la versión 1.10.12, y se descomprime en cualquier ubicación, puede ser *C:\Program Files*.

Luego hay que crear dos variables de entorno (Figura 100), ANT_HOME y JAVA_HOME, que apuntan a las ubicaciones de instalación de *Ant* y *Java*.

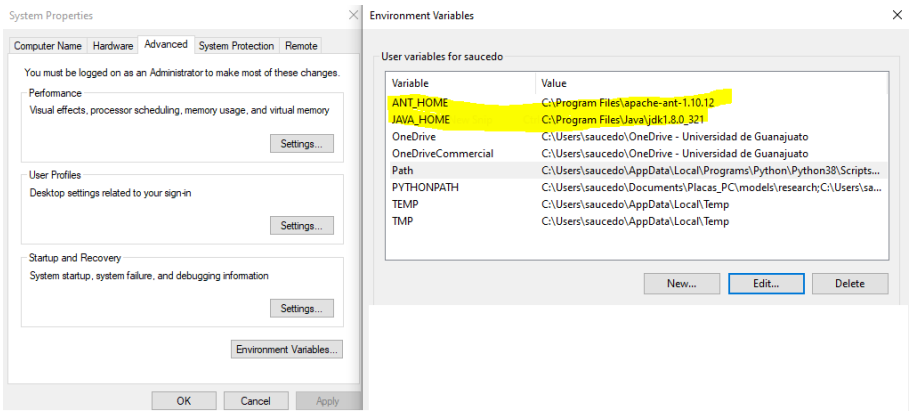


Figura 100. Variables de entorno.

También hay que agregar la carpeta 'bin' de Ant a la variable de entorno Path (Figura 101).

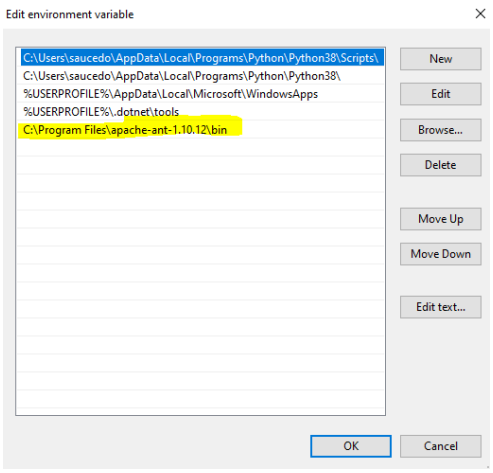


Figura 101. Agregar carpeta bin de Ant.

Ya quedaría sólo compilar jAER, esto se hace entrando a la carpeta de jAER y ejecutando Ant:

```
cd jaer-dist

ant
```

Una vez realizado, se puede utilizar el visor de eventos dentro de la carpeta jAER: *jaerViewer_win64.exe*. Por otro lado, el paquete de manipulación de eventos para Matlab y Python se encuentra en [92]. En Matlab sólo basta descargarlo y agregarlo al path (Figura 102).

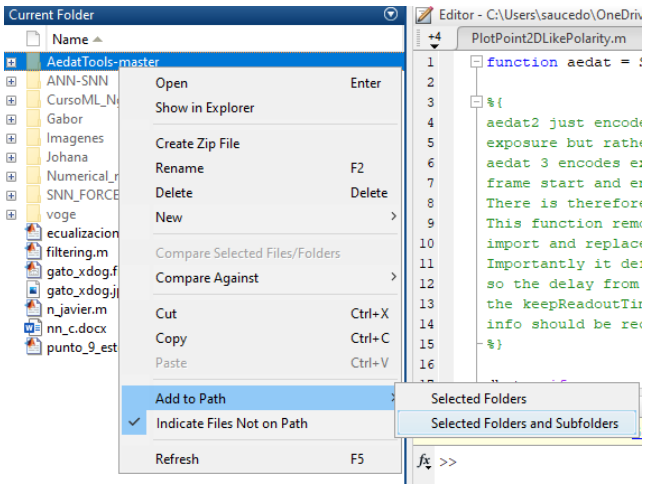


Figura 102. Para agregar un directorio al path de Matlab no hace falta modificar las variables de entorno, solo basta con hacer ésto (si se quiere agregar algún directorio permanentemente toca hacerlo en 'set path').

En estos archivos de Matlab se pueden realizar algunas funciones básicas como importar y exportar ficheros (ImportAedat, ExportAedat), recortar temporal o espacialmente (TrimSpace, TrimTime) convertir a 'frames' (FramesFromEvents) y generar gráficos (PlotPolarity, PlotPolarity3D). Además, en Matlab se pueden explorar los datos (como son estructuras en Matlab se puede hacer accediendo a sus atributos). También se puede probar a visualizar los *.aedat* generados en Matlab con jAER usando distintos parámetros de las funciones.

Para usar la cámara es necesario instalar drivers en el ordenador, tener presente que depende del sistema operativo que se posea, puede haber problemas (Figura 103). En Windows 10 había problemas con los drivers de la cámara, los bloquea. Hay que realizar una serie de pasos para poder instalar los drivers. En versiones más antiguas de Windows sí funcionan. Una vez se instalen (Figura 104), se ejecuta jAER y saldrá una ventana, en la que aparece lo que se mueva delante de la retina en tiempo real. A veces, hace falta quitar y poner la alimentación de la retina para que empiecen a visualizarse los datos.

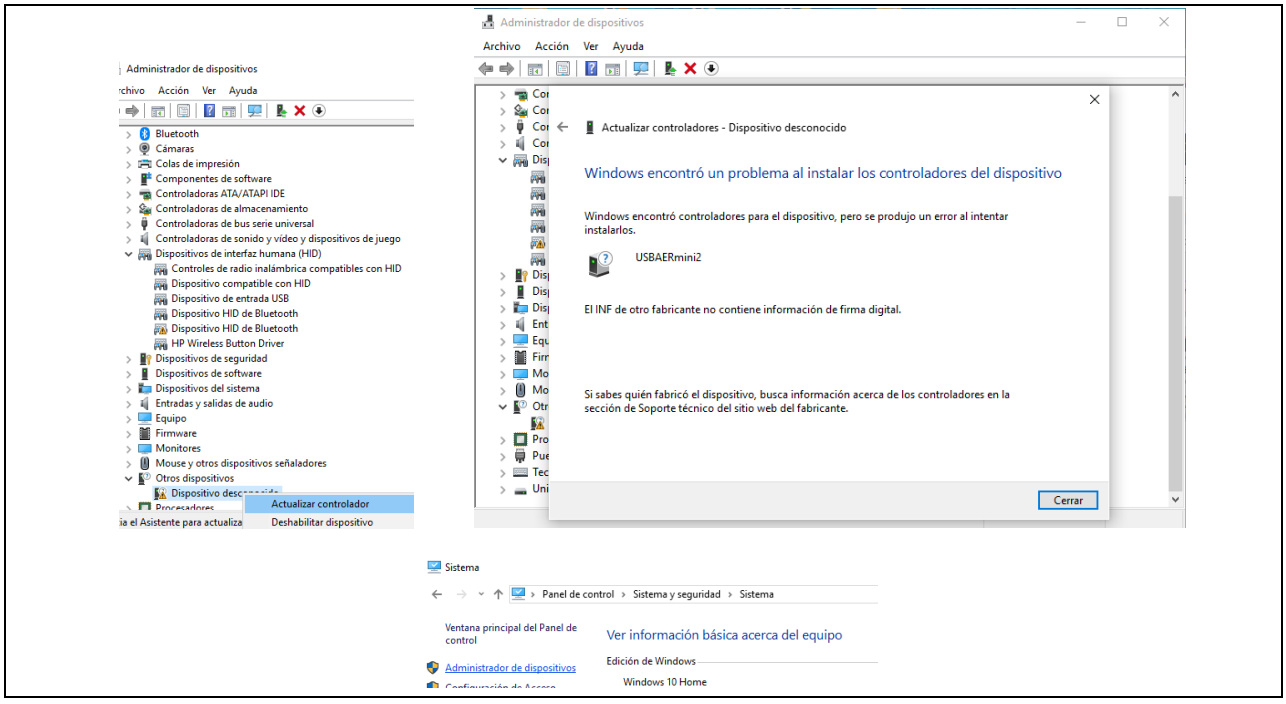


Figura 103. Para ver si está instalado el driver, hay que ir al Panel de Control y darle a Administrador de dispositivos.

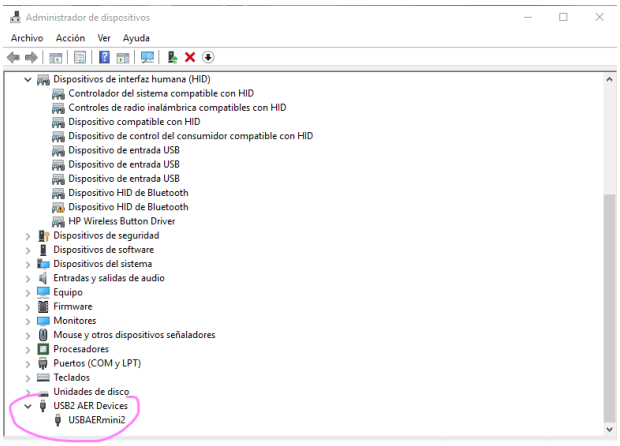
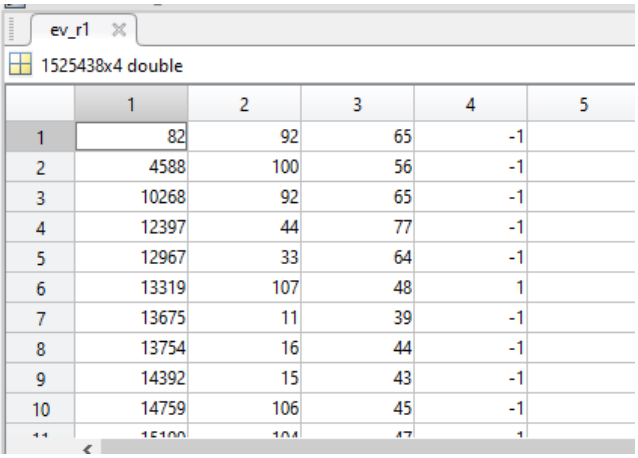


Figura 104. Driver instalado.

ANEXO B

En este anexo se muestran los códigos usados en Matlab.

El archivo **frames_2D.m** fue proporcionado por el tutor Luis Alejandro Camuñas como ejemplo sencillo de representación de eventos, a partir del archivo **events_fan_10.mat** (la grabación son dos lápices rotando conectados a las hélices de un ventilador), para entender el concepto de visión por eventos. Se ejecuta en primer lugar *load events_fan_10*, con lo cual se cargará una matriz que contiene una grabación obtenida con un sensor de visión por eventos. Si se observa esa matriz (Figura 105), se verá que cada fila almacena un evento, que incluye en la columna 1 una marca temporal (indica cuándo se generó ese evento), en la columna 2 la coordenada x del píxel que generó ese evento, en la columna 3 la coordenada y, y en la columna 4 la polaridad de ese evento (indica si el evento representa un incremento de luminosidad o un decremento). Para entender el significado de esos eventos, se puede ejecutar frames_2D(ev_r1(:,2),ev_r1(:,3),ev_r1(:,1)), con lo cual (tras pulsar una tecla para salir del estado de pausa inicial) se verá una reproducción visual de la grabación original (esta rutina genera unos fotogramas artificiales (o frames) y los va cambiando para generar sensación de movimiento). Se puede modificar la velocidad de reproducción entrando en el código de frames_2D y modificando el valor de Tframe.


ev_r1

1525438x4 double

	1	2	3	4	5
1	82	92	65	-1	
2	4588	100	56	-1	
3	10268	92	65	-1	
4	12397	44	77	-1	
5	12967	33	64	-1	
6	13319	107	48	1	
7	13675	11	39	-1	
8	13754	16	44	-1	
9	14392	15	43	-1	
10	14759	106	45	-1	

Figura 105. Archivo **events_fan_10.mat.**

Código 1. El código frames_2D.m

```
function frames_2D(x1,x2,t)

Tframe = 10000;     % 300000,200000,100000
Toverlap = Tframe/4;
nframes = ceil(max(t)/Tframe);


figure
pause
for i = 1 : nframes
    index = find((t>(i-1)*Tframe-Toverlap) & (t<=i*Tframe+Toverlap));
```

```
    plot(x1(index),x2(index),'.r'),axis([1 128 1 128])
    set(gca,'fontsize',15)
    grid on
    pause(Tframe/1e6)
    %     pause
end
```

El archivo **repre.m** se usó para representar la evolución de la intensidad de un píxel concreto respecto al tiempo. El código de **repre.m** se muestra a continuación, hay que ejecutarlo después de **obten_datos_imagen_cartas.m** o **obten_datos_imagen.m**.

Código 2. El código repre.m

```
i=61;j=44;
 % find((event(:,1)>1.9371*10^6))
 find((event(:,2)==i) & (event(:,3)==j))
 gtn=1;
 eventime=[];
 eventvalor=[];
 for gt=1:n_eventos
     posicioni=event(gt,2);
     posicionj=event(gt,3);
     poly=event(gt,4);
     if (posicioni==i) && (posicionj==j)
         eventime(gtn)=event(gt,1);
         eventvalor(gtn)=poly*10;
         gtn=gtn+1;
     end


 end
 %ten en cuenta que despues de interpolar se tienen más imagenes
 pixel12=gMedia(i,j,:);
 pixel12=pixel12(:);%convertir matriz en vector
 pixel1=intensidades(i,j,:);
 pixel1=pixel1(:);%convertir matriz en vector
 %         p=[1:0.2:length(pixel1)];
 ve=interp1(1:1:length(pixel1),pixel1,p,'linear');
 %
 figure(18);
  tempo2=tempo(1:length(pixel1));

 plot(ve_tempo,pixel12','g-*',tempo2,pixel1,'b-
o',ve_tempo,ve,'r.',eventime,eventvalor,'c*');xlabel(' (tiempo)');
ylabel('intensidad del pixel ');%pixel 1
 title(['pixel ij: ' num2str(i) ' ' num2str(j)])
 legend({'GMedia','original','interpolado','evento'},'Location','southwest')
```

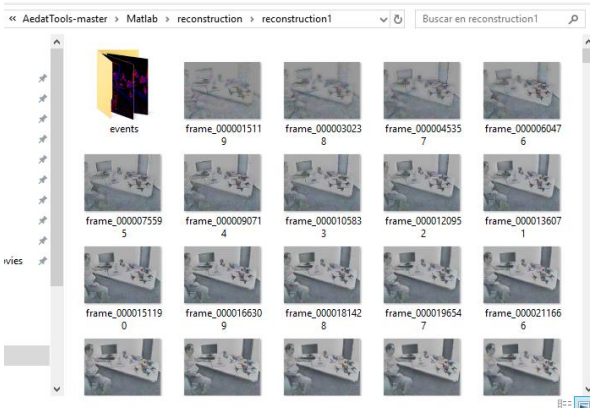


Figura 106. Lo que se encuentra en carpeta **reconstruction1.**

Respecto a cómo se cargan las imágenes y el archivo de los tiempos desde la carpeta (Figura 106), al principio del programa se hace lo siguiente.

Código 3. Trozo de código: Cargar imágenes y archivos.

```
 lee_archivos = dir('C:\Users\isabe\Desktop\AedatTools-master\AedatTools-
master\Matlab\reconstruction\reconstruction_cartas\*.png'); %el formato de
imagen puede ser modificado.

 archivo = lee_archivos(1).name; %Obtiene el nombre de los archivos
 nombre='C:\Users\isabe\Desktop\AedatTools-master\AedatTools-
master\Matlab\reconstruction\reconstruction_cartas\'; %Recore el diretorio
 I = imread(strcat(nombre,archivo));% lee la primera imagen
 [M N rgb]=size(I);
 n_pixeles=M*N;
 intensidades=[];
 limit=0;

 %% tiempos de cada foto
 archivof='C:\Users\isabe\Desktop\AedatTools-master\AedatTools-
master\Matlab\reconstruction\reconstruction_cartas\timestamps.txt';

 fid = fopen(archivof, 'r'); % opción rt para abrir en modo texto
 if fid==-1
     disp('error')
     return;
 end
 tempo=fscanf(fid,'%f');
 fclose(fid);
 % % k es numero de imagen
 for k = 1:length(lee_archivos)-limit %recorre número de archivos guardados
en el directorio
     archivo = lee_archivos(k).name; %Obtiene el nombre de los archivos
     I = imread(strcat(nombre,archivo));% lee la primera imagen
 . . .
```

B.1 Representar eventos

Código 4. Tamaño sensor: obtenido a partir de archivo original

```
M=max(x);
N=max(y);
```

Para representar los eventos del archivo de texto creado después de foveación (con periodo fijo) (**isabelyo3.m**).

Código 5. El código isabelyo3.m

```
close all;
%clear all;
addpath('C:\Users\isabe\Desktop\AedatTools-master\AedatTools-master\Matlab')
addpath('C:\Users\isabe\Desktop\AedatTools-master\AedatTools-
master\Matlab\reconstruction')
frames=[];

eventos=table2array(readtable('dynamic_6dofcar.txt'));%no coge la primera
fila que viene el tamaño del sensor

vector_tiempos=eventos(:,1);
x=eventos(:,2);
y=eventos(:,3);

M=max(x);
N=max(y);
tama=[M N];

Tframe = 10000;

cont=0;
k=1;

t_f=Tframe;

puntos=[];
u=1;
w=1;
nframes = ceil(max(vector_tiempos)/Tframe);
index1=[];indey1=[];index0=[];indey0=[];

for i = 1 : nframes
    puntos(:,:)=zeros(M,N);

    while cont<=t_f & k<=length(x) & k<=length(y)
        %         x(k)%se vio que hay x(k)=0, en matlab empieza en 1 indices
        %         y(k)
        if eventos(k,4)==1
            puntos(x(k),y(k))=255;
            index1(u)=x(k);
            indey1(u)=y(k);
            u=u+1;

        else
            if eventos(k,4)==0
                puntos(x(k),y(k))=100;
```

```
                index0(w)=x(k);
                indey0(w)=y(k);
                w=w+1;
            end
        end


        cont=vector_tiempos(k);
        k=k+1;

    end

    frames(:,:,i)=puntos;
    t_f=Tframe+i*Tframe;

    figure(1);
    clf;
    hold on;
    if(~isempty(index1) || ~isempty(indey1))
        plot(index1,indey1,'.r');
    end
    if(~isempty(index0) || ~isempty(indey0))
        plot(index0,indey0,'.b');
    end
    axis([1 M 1 N]);
    hold off;

    w=1;u=1;
    index1=[];indey1=[];index0=[];indey0=[];
    figure(2)
    imshow(uint8(frames(:,:,i)))
    %     figure(3);
    %     g1 = imresize(frames(:,:,i), 3, 'nearest');%
    %     imshow(g1)
    pause(Tframe/1e6)


end
```

Para representar los eventos a partir de la estructura que devuelve el programa para la cámara (*.aedat*), la cual será convertida a un *.txt* (para usarlo en la conversión a fotogramas de escala de grises), se utiliza lo siguiente (**isabelyo.m**).

Código 6. El código isabelyo.m

```
close all;
%clear all;
addpath('C:\Users\isabe\Desktop\AedatTools-master\AedatTools-master\Matlab')

% Create a structure with which to pass in the input parameters.
aedat = struct;

% Put the filename, including full path, in the 'file' field.
```

```
 %aedat.importParams.filePath = 'C:\project\example3.aedat'; % Windows
 aedat.importParams.filePath = 'C:\Users\isabe\Desktop\AedatTools-
master\AedatTools-master\Matlab\cards_3.aedat';
 %aedat.importParams.filePath = '/home/project/example3.aedat'; % Linux

 % Alternatively, make sure the file is already on the matlab path.
 %addpath('C:\project')
 addpath('C:\Users\isabe\Desktop\AedatTools-master\AedatTools-master\Matlab')
 aedat.importParams.filePath = 'cards_3.aedat';

 aedat = ImportAedat(aedat);
 % return
 frames=[];

 figure(1)

 vector_tiempos=aedat.data.polarity.timeStamp;
 x=aedat.data.polarity.x;
 y=aedat.data.polarity.y;
 M=max(x)+1;
 N=max(y)+1;
 %% convertir a txt
 tama=[M N];
 archivo=fopen('dynamic_6dofcartas.txt','w');
 fprintf(archivo,'%d %d\n',tama);
 for k = 1:length(vector_tiempos)
     fprintf(archivo,'%f %d %d
%d\n',vector_tiempos(k),x(k),y(k),aedat.data.polarity.polarity(k));

 end

 fclose(archivo);
 fprintf('Archivo txt\n');
 %%
 Tframe = 10000;
 cont=0;
 k=1;
 t_f=Tframe;
 puntos=[];
 numero_eventos_en_el_frame=[];
 u=1;
 w=1;
 nframes = ceil(max(vector_tiempos)/Tframe);

 index1=[];indey1=[];index0=[];indey0=[];
 for i = 1 : nframes
     puntos(:,:)=zeros(M,N);
     figure(1)
     while cont<=t_f & k<=length(x) & k<=length(y)
         %         x(k)%se vio que hay x(k)=0, en matlab emoieza en 1 indices
         %         y(k)
         if aedat.data.polarity.polarity(k)==1
             puntos(x(k)+1,y(k)+1)=255;
             index1(u)=x(k);
             indey1(u)=y(k);
             u=u+1;

         else
             if aedat.data.polarity.polarity(k)==0
                 puntos(x(k)+1,y(k)+1)=100;
                 index0(w)=x(k);
```

```
                indey0(w)=y(k);
                w=w+1;
            end
        end


        cont=vector_tiempos(k);
        k=k+1;

    end

    frames(:,:,i)=puntos;
    numero_eventos_en_el_frame(i)=u-1+w-1;
    t_f=Tframe+i*Tframe;
    clf;
    subplot(1,2,1)

    hold on;
    if(~isempty(index1) || ~isempty(indey1))
        plot(index1,indey1,'.r');
    end
    if(~isempty(index0) || ~isempty(indey0))
        plot(index0,indey0,'.b');
    end
    axis([1 M 1 N]);
    hold off;
    %     plot(index1,indey1,'.r',index0,indey0,'.b');axis([1 128 1 128]);

    w=1;u=1;
    index1=[];indey1=[];index0=[];indey0=[];
    subplot(1,2,2)
    imshow(uint8(frames(:,:,i)))
    %     figure(3);
    %     g1 = imresize(frames(:,:,i), 3, 'nearest');%
    %     imshow(g1)
    pause(Tframe/1e6)

end
```

El tiempo de frame usado fue de 10.000, pero podría haber sido otro valor. Esto se ajusta según la duración de la grabación que se use y la unidad de tiempo.

El archivo **isabelyo5.m** sigue un procedimiento semejante a los anteriores (es para visualizar eventos de la grabación del hombre sin foveación o la de las cartas sin foveación en formato *.txt*), por ello no parece que sea relevante mostrarlo. Es el archivo **isabelyo3.m** con la modificación de leer el archivo 'dynamic_6dof.txt' o 'dynamic_6dofcartas.txt' y tener en cuenta que, como en **isabelyo.m**, los índices 'x' e 'y' empiezan en [0,0].

El archivo **isabelyo4.m** es parecido al archivo **isabelyo3.m**, en lugar de esperar un determinado tiempo, se espera a que haya una determinada cantidad de eventos en cada fotograma.

Código 7. El código de la función isabelyo4.m

```
frames=[];

eventos=table2array(readtable('dynamic_6dofdron.txt'));%no coge la primera
fila que viene el tamaño del sensor

vector_tiempos=eventos(:,1);
x=eventos(:,2);
y=eventos(:,3);
M=max(x);
N=max(y);
%% txt
tama=[M N];
cont=0;
k=1;
puntos=[];
u=1;
w=1;
numero_eventos_en_el_frame=[];

 i=1;
while k <= length(x)& k<=length(y)
    i=i+1;
    puntos(:,:)=zeros(M,N);

    while cont<=10000 & k <= length(x)& k<=length(y)

        %         x(k)%se vio que hay x(k)=0, en matlab empieza en 1 indices
        %         y(k)
        if eventos(k,4)==1
            puntos(x(k),y(k))=255;
            u=u+1;
            cont=cont+1;

        else
            if eventos(k,4)==0
                puntos(x(k),y(k))=100;
                w=w+1;
                cont=cont+1;
            end
        end

        k=k+1;

    end
    cont=0;
    numero_eventos_en_el_frame(i)=u-1+w-1;
    frames(:,:,i)=puntos;

     figure(1);

    w=1;u=1;

    imshow(uint8(frames(:,:,i)))
     pause(0.1)


end
```

Para importar el archivo *.aedat* se usa **ImportAedat.m** (este archivo me lo proporcionó el IMSE, [92]).

Código 8. El código de la función ImportAedat.m

```
function aedat = ImportAedat(aedat)


dbstop if error %Set breakpoints for debugging

% If the input variable doesn't exist, create a dummy one.
if ~exist('aedat', 'var')
    aedat = struct;
end

% Open the file
if ~isfield(aedat, 'importParams') || ~isfield(aedat.importParams,
'filePath')
    [fileName path ~] = uigetfile('*.aedat','Select aedat file');
    if fileName==0
        disp('File to import not specified')
        return
    end
    aedat.importParams.filePath = [path fileName];
end

aedat.importParams.fileHandle = fopen(aedat.importParams.filePath, 'r');

if aedat.importParams.fileHandle == -1
    error('file not found')
end

% Process the headers if they haven't been processed already
% The 'info' field is created by the ImportAedatHeaders function
if ~isfield(aedat, 'info')
    aedat = ImportAedatHeaders(aedat);
end

% Process the data - different subfunctions handle fileFormat 2 vs 3
if aedat.info.fileFormat < 3
    aedat = ImportAedatDataVersion1or2(aedat);
else
    aedat = ImportAedatDataVersion3(aedat);
end

fclose(aedat.importParams.fileHandle);
aedat.importParams = rmfield(aedat.importParams, 'fileHandle');
```

B.2 Interpolación

Código 9. Trozo de código: Interpolación con cartas

```
%interpolación
intensidades2=[];
pixel1=intensidades(1,1,:);
```

```
pixel1=pixel1(:);
p=[1:0.2:length(pixel1)];
ve_tempo=interp1(1:1:length(tempo),tempo,p,'linear');
for  i=1:M
    for j=1:N
        pixel1=intensidades(i,j,:);
        pixel1=pixel1(:);%convertir matriz en vector
        ve=interp1(1:1:length(pixel1),pixel1,p,'linear');
        intensidades2(i,j,:)=ve;
    end
end
n_imagenes_inicio=length(pixel1)
[q1 q2 n_imagenes]=size(intensidades2);
```

Código 10. Trozo de código: Interpolación con hombre

```
%interpolación
intensidades2=[];
pixel1=intensidades(1,1,:);
pixel1=pixel1(:);
p=[1:0.2:length(pixel1)];
ve_tempo=interp1(1:1:length(pixel1),tempo(1:length(pixel1)),p,'linear');

for  i=1:M
    for j=1:N
        pixel1=intensidades(i,j,:);
        pixel1=pixel1(:);%convertir matriz en vector
        ve=interp1(1:1:length(pixel1),pixel1,p,'linear');
        %          ve=interp1(1:1:length(pixel1),pixel1,p,'pchip');%hermite
        %         v=interp1(1:1:length(pixel1),pixel1,p,'nearest');%no vale
        %          v=interp1(1:1:length(pixel1),pixel1,p,'cubico');
        intensidades2(i,j,:)=ve;
    end
end
% n_imagenes=length(lee_archivos)-limit;
n_imagenes_inicio=length(pixel1)
[q1 q2 n_imagenes]=size(intensidades2);
n_imagenes
```

B.3 Encontrar objeto de interés

Figura cartas

Se realizó la función **detection_object.m**, a partir de la imagen devuelve los centroides y boundingbox de los objetos.

Código 11.Función detection_object.m

```
function [centroides,BoundingBox]=detection_object(h)
%% binarizacion
[m n]=size(h);
im_salida=ones(m,n);
U_sup_r=105;
U_inf_r=0;
```

```
for i=1:m
    for j=1:n

        if (h(i,j)<=U_sup_r && h(i,j)>=U_inf_r)%

            im_salida(i,j)=1;
        else
            im_salida(i,j)=0;
        end
    end
end

%% modificar
se=strel('disk',1);
im_cerrada=imclose(im_salida,se);
im_cerrada2=medfilt2(im_cerrada);
bw=bwareaopen(im_cerrada2,20);

%% detección características
s=regionprops(bw,'centroid','BoundingBox');
centroides=cat(1,s.Centroid);
BoundingBox=cat(1,s.BoundingBox);
end
```

En el código principal se muestra:

Código 12. Trozo de código: Detección zona interés con cartas

```
%% encontrar objeto
    h=intensidades2(:,:,k);
    [m n]=size(h);
    %% deteccion objetos (sin identificar lo que son)
    Cx=[];
    Cy=[];
    bbox=[];
    n_cuadrados=0;
    [C,B]=detection_object(h);
    if(~isempty(C))% si ha detectado algún objeto
        C=round(C);
        Cx=[Cx;C(:,1)];
        Cy=[Cy;C(:,2)];
        bbox=[bbox;B];
        [n_cuadrados col]=size(bbox)
    end

    if n_cuadrados>0

        for s=1:n_cuadrados

            %[x,y,w,h]
            fin_M=round(bbox(s,2)+bbox(s,4));%%y+h
            ini_i=round(bbox(s,2));
            fin_N=round(bbox(s,1)+bbox(s,3));
            ini_j=round(bbox(s,1));
```

```
            if fin_M>m
                fin_M=m;
            end
            if fin_N>n
                fin_N=n;
            end

            %-----------------------------------------
            for i=ini_i:1:fin_M
                for j=ini_j :1:fin_N
                    gMedia(i,j,k)=intensidades2(i,j,k);%
                end
            end
            %-----------------------------------------
        end
    end

end
```

Rostro hombre:

Código 13. Trozo de código: Detección zona interés con hombre

```
%Deteccion cara
facedetector=vision.CascadeObjectDetector('RightEye');
k=1;
bbox=[];
index=[];
for r=1:n_imagenes
    h=uint8(intensidades2(:,:,r));
    bb=step(facedetector,h);
    [a b]=size(bb);
    if a==0
        bbox(k,:)=[0,0,0,0];%[x,y,w,h]
        k=k+1;

    else
        bbox(k:k+a-1,:)=bb;
        k=k+a;

    end
    index(r)=a;

end
```

B.4 Cambio de resolución (foveación)

Código 14. Trozo de código: Foveación con cartas

```
%una vez obtenidos datos de intensidades de cada imagen se obtienen la que
%es aplicando media de intensidades en cada una
gMedia = [];
gMedia2=[];
```

```
 resultado=[];
 radioMask = 2;%12; % 3, 5
 nPixelsMask = (2*radioMask+1)^2;
 s=0;
 for k = 1:n_imagenes %recorre número de imágenes guardadas en el directorio
     io=1;jo=1;
     gMedia(:,:,k)=intensidades2(:,:,k);

     inic=0;

     for ii=radioMask+1 :2*radioMask+1: M-radioMask
         jo=1;
         for jj=radioMask+1 :2*radioMask+1: N-radioMask

             gMedia2(io,jo,k) = 1/nPixelsMask * sum(sum(intensidades2(ii-
radioMask:ii+radioMask,jj-radioMask:jj+radioMask,k)));
             gMedia(ii,jj,k)=gMedia2(io,jo,k);
             for id=ii-radioMask:ii+radioMask
                 for jd=jj-radioMask:jj+radioMask
                     gMedia(id,jd,k)=gMedia2(io,jo,k);
                 end
             end
             jo=jo+1;

             %comprobar si va a haber problemas con tamaño de mascara en
columnas en futuro

             if(((jj+2*radioMask+1)>N-radioMask) ||
((jj+2*radioMask+1+radioMask)>N))
                 gMedia2(io,jo,k) = 1/nPixelsMask * sum(sum(intensidades2(ii-
radioMask:ii+radioMask,jj+radioMask+1:N,k)));
                 %
                 for id=ii-radioMask:ii+radioMask
                     for jd=jj+radioMask+1:N
                         gMedia(id,jd,k)=gMedia2(io,jo,k);
                     end
                 end
                 jj=N;
             end




         end
         io=io+1;

 %comprobar si va a haber problemas con tamaño de mascara en filas en futuro

         if(((ii+2*radioMask+1)>M-radioMask) ||
((ii+2*radioMask+1+radioMask)>M) )
             inic=ii;

             ii=M;
         end

     end
```

```
    if(inic~=0)%para la última fila
        ii=inic+1+radioMask;

        jo=1;
        for jj=radioMask+1 :2*radioMask+1: N-radioMask


            gMedia2(io,jo,k) = 1/nPixelsMask *
sum(sum(intensidades2(ii:M,jj-radioMask:jj+radioMask,k)));
            for id=ii:M
                for jd=jj-radioMask:jj+radioMask
                    gMedia(id,jd,k)=gMedia2(io,jo,k);
                end
            end

            jo=jo+1;

            if(((jj+2*radioMask+1)>N-radioMask) ||
((jj+2*radioMask+1+radioMask)>N))
                gMedia2(io,jo,k) = 1/nPixelsMask *
sum(sum(intensidades2(ii:M,jj+radioMask+1:N,k)));
                %
                for id=ii:M
                    for jd=jj+radioMask+1:N
                        gMedia(id,jd,k)=gMedia2(io,jo,k);
                    end
                end
                jj=N;
            end



        end
        io=io+1;


    end

    %% encontrar objeto
    h=intensidades2(:,:,k);
    [m n]=size(h);
    Cx=[];
    Cy=[];
    bbox=[];
    n_cuadrados=0;
    [C,B]=detection_object(h);
    if(~isempty(C))% si ha detectado algún objeto
        C=round(C);
        Cx=[Cx;C(:,1)];
        Cy=[Cy;C(:,2)];
        bbox=[bbox;B];
        [n_cuadrados col]=size(bbox);


    end


    if n_cuadrados>0
```

```
        for s=1:n_cuadrados

            %[x,y,w,h]
            fin_M=round(bbox(s,2)+bbox(s,4));%%y+h
            ini_i=round(bbox(s,2));
            fin_N=round(bbox(s,1)+bbox(s,3));
            ini_j=round(bbox(s,1));
            if fin_M>m
                fin_M=m;
            end
            if fin_N>n
                fin_N=n;
            end

            %--------------poner a resolución original la zona de interés
            %
            for i=ini_i:1:fin_M
                for j=ini_j :1:fin_N
                    gMedia(i,j,k)=intensidades2(i,j,k);%
                end
            end
            %-----------------------------------------------
        end
    end

end

[a b c]=size(gMedia);
```

Para representar el resultado:

Código 15. Trozo de código: Representar secuencia de imágenes resultantes

```
%% representacion

for k=1:c
    figure(1)
     hk=uint8(gMedia(:,:,k));

    imshow(hk)
%     pause(0.5)
%       hk2=imresize(hk,2);
%      figure(2)
%      imshow(hk2)


end
```

Código 16. Trozo de código: Foveación con hombre

```
 %una vez obtenidos datos de intensidades de cada imagen se obtiene la que
 %es aplicando media de intensidades en cada una
 gMedia = [];
 resultado=[];
 radioMask = 2; % 3, 5
 nPixelsMask = (2*radioMask+1)^2;
 s=0;
 for k = 1:n_imagenes %recorre número de imágenes guardadas en el directorio
     io=1;jo=1;
     gMedia(:,:,k)=intensidades2(:,:,k);
     inic=0;

     for ii=radioMask+1 :2*radioMask+1: M-radioMask
         jo=1;
         for jj=radioMask+1 :2*radioMask+1: N-radioMask

             gMedia2(io,jo,k) = 1/nPixelsMask * sum(sum(intensidades2(ii-
radioMask:ii+radioMask,jj-radioMask:jj+radioMask,k)));
             gMedia(ii,jj,k)=gMedia2(io,jo,k);
             for id=ii-radioMask:ii+radioMask
                 for jd=jj-radioMask:jj+radioMask
                     gMedia(id,jd,k)=gMedia2(io,jo,k);
                 end
             end
             jo=jo+1;

 %comprobar si va a haber problemas con tamaño de mascara en columnas en
futuro
             if(((jj+2*radioMask+1)>N-radioMask) ||
((jj+2*radioMask+1+radioMask)>N))
                 gMedia2(io,jo,k) = 1/nPixelsMask * sum(sum(intensidades2(ii-
radioMask:ii+radioMask,jj+radioMask+1:N,k)));

                 for id=ii-radioMask:ii+radioMask
                     for jd=jj+radioMask+1:N
                         gMedia(id,jd,k)=gMedia2(io,jo,k);
                     end
                 end
                 jj=N;
             end




         end
         io=io+1;

 %comprobar si va a haber problemas con tamaño de mascara en filas en futuro

         if(((ii+2*radioMask+1)>M-radioMask) ||
((ii+2*radioMask+1+radioMask)>M) )
             inic=ii;

             ii=M;
         end

     end



     if(inic~=0)%para la última fila
         ii=inic+1+radioMask;
```

```
        jo=1;
        for jj=radioMask+1 :2*radioMask+1: N-radioMask


            gMedia2(io,jo,k) = 1/nPixelsMask *
sum(sum(intensidades2(ii:M,jj-radioMask:jj+radioMask,k)));
            for id=ii:M
                for jd=jj-radioMask:jj+radioMask
                    gMedia(id,jd,k)=gMedia2(io,jo,k);
                end
            end

            jo=jo+1;

            if(((jj+2*radioMask+1)>N-radioMask) ||
((jj+2*radioMask+1+radioMask)>N))
                gMedia2(io,jo,k) = 1/nPixelsMask *
sum(sum(intensidades2(ii:M,jj+radioMask+1:N,k)));

                for id=ii:M
                    for jd=jj+radioMask+1:N
                        gMedia(id,jd,k)=gMedia2(io,jo,k);
                    end
                end
                jj=N;
            end



        end
        io=io+1;


    end
    %------ahora se determinan las zonas de interés

    n_cuadrados=index(k);
    if n_cuadrados>0

        for ty=1:n_cuadrados
            s=s+1;
            %[x,y,w,h]
            fin_M=bbox(s,2)+bbox(s,4);%%y+h
            ini_i=bbox(s,2);
            fin_N=bbox(s,1)+bbox(s,3);
            ini_j=bbox(s,1);
            ini_jo=ini_j;
            ini_io=ini_i;


            %-----------pongo a resolución original esa zona
            for i=ini_io:fin_M
                for j=ini_j:fin_N
                    gMedia(i,j,k)=intensidades2(i,j,k);
                end
```

```
            end
        end
    else
        s=s+1;

    end
end
```

B.5 Convertir conjunto de imágenes en eventos

El caso de grabación del hombre tiene el mismo código que las cartas (‘dynamic_6dofcar.txt’), pero se guarda resultado en ‘dynamic_6dofdron.txt’. La variable ‘Ay’ es el umbral que determina cuando se produce evento y se debe modificar su valor según el caso.

Código 17. Trozo de código: Obtener eventos a partir de imágenes

```
numero_pixeles_total =a*b*c;
Ay=0.5;% variación intensidad a partir de la cual se considera evento
cont=1;
event=[];
n_eventos1=0;
n_eventos0=0;

tama=[a b];

archivo=fopen('dynamic_6dofcar.txt','w');
fprintf(archivo,'%d %d\n',tama);%tamaño del sensor (tamaño matriz resultante
(gMedia))

Io_indice=ones(a,b);
for k = 2:c

    for i=1:a

        for j=1:b

            if abs((gMedia(i,j,k)-
gMedia(i,j,Io_indice(i,j)))/gMedia(i,j,Io_indice(i,j)))>=Ay

                if sign((gMedia(i,j,k)-
gMedia(i,j,Io_indice(i,j)))/gMedia(i,j,Io_indice(i,j)))>0

                    event(cont,:)=[ve_tempo(k) i j 1];
                    n_eventos1=n_eventos1+1;

                else
                    event(cont,:)=[ve_tempo(k) i j 0];
                    n_eventos0=n_eventos0+1;

                end
                fprintf(archivo,'%f %d %d %d\n',event(cont,:));
                cont=cont+1;

                Io_indice(i,j)=k;
```

```
            else
            end

        end
    end
end
fclose(archivo);
n_eventos=n_eventos1+n_eventos0
%pcr= (644672-n_eventos)/644672
```

ANEXO C

Se muestra la función que realizaría el algoritmo de detección de objetos basado en eventos, en Matlab. La dificultad que tiene es entender lo que son las variables de entrada y de salida.

El siguiente código fue proporcionado por el IMSE (referente al artículo [93]), en él se muestra un ejemplo de *object tracking*:

Código 18. Función de object tracking

```
function [ev_out,clusters_out] = cluster_tracker(ev_in,clusters_in)

% Format ev_in: 1 row, 4 columns
% [t, x, y, sign]

% Format ev_out: 1 row, 5 columns
% [t, x, y, sign, cl]

% Matrix clusters: nclusters rows (variable), 7 columns
% [xc, yc, tx, ty, nev, nmatch, tlast] = [x-coordinate of center, y-
coordinate of center,
% x-dimension of cluster size, y-dimension of cluster size, number of events
included in the cluster,
% row number of the corresponding cluster in the other retina,
% time of the last event associated to the cluster]

alpha_k = 4;  % Proportion of the cluster size where the new event can be
contained
alpha_c = 0.9;  % Parameter to smooth position change
alpha_r = 0.9;  % Parameter to smooth the size of the cluster
min_size = 3;  % Minimum size allowed for clusters

initial_size = 20;
% figure,axis([0 128 0 128])

% [nevents,~] = size(ev_in);
ev_out = zeros(1,5);
ev_out(:,1:4) = ev_in;
[ncluster_write,~] = size(clusters_in);
ncluster_write = ncluster_write + 1;

clustered = 0;
ncluster_read = 1;
time = ev_in(1); % time of new event
x = ev_in(2); % x-coordinate of new event
y = ev_in(3); % y-coordinate of new event
while (clustered == 0) & (ncluster_write-1 >= ncluster_read)
    xc = clusters_in(ncluster_read,1);  % x-coordinate of the center of the
cluster
    yc = clusters_in(ncluster_read,2);  % y-coordinate of the center of the
cluster
    tx = clusters_in(ncluster_read,3);  % x-dimension of the cluster size
    ty = clusters_in(ncluster_read,4);  % y-dimension of the cluster size
    nev = clusters_in(ncluster_read,5); % number of events in the cluster
    % We check if the new event in contained in this cluster
    if (x >= xc-(1+alpha_k)*tx/2) && (x <= xc+(1+alpha_k)*tx/2)...
            && (y >= yc-(1+alpha_k)*ty/2) && (y <= yc+(1+alpha_k)*ty/2)
```

```
            ev_out(5) = ncluster_read;
            clustered = 1;
            xc_new = xc*alpha_c + x*(1-alpha_c);    % Update cluster center
            yc_new = yc*alpha_c + y*(1-alpha_c);
            clusters_in(ncluster_read,1:2) = [xc_new,yc_new];
            tx_new = max(min_size,tx*alpha_r + 2*abs(xc-x)*(1-alpha_r));    %
Update cluster size
            ty_new = max(min_size,ty*alpha_r + 2*abs(yc-y)*(1-alpha_r));
            clusters_in(ncluster_read,3:4) = [tx_new,ty_new];
            clusters_in(ncluster_read,5) = nev + 1;
            clusters_in(ncluster_read,7) = time;
        end
        ncluster_read = ncluster_read + 1;
    end
    if clustered == 0
        clusters_in(ncluster_write,1:7) =
[x,y,initial_size,initial_size,1,0,time];
        %     clusters_in(ncluster_write,1:4) = [x,y,initial_size,initial_size];
        ev_out(5) = ncluster_write;
        %         ncluster_write = ncluster_write + 1;
        %         clustered = 1;
    end

    clusters_out = clusters_in;
```

ANEXO D

Los programas principalmente necesario para usar la placa PYNQ-Z2 son Jupyter, Vivado, Linux y Python (Figura 107). El Linux se instala en la placa Pynq, no en ordenador. También hay que tener un navegador web compatible para poder programar con Jupyter. La parte de Python se programa en Jupyter y la parte de VHDL en Vivado.

Figura 107. Software necesario. Fuentes: [103] [119].

Para que se conecte correctamente la placa al ordenador, con conexión directa, hay que configurar la dirección ip de Windows como:

```
IP asignada: 192.168.2.100

Submáscara: 255.255.255.0

Pasarela/Gateway: 192.168.2.99
```

Para conectarlo directo a la red había que enviar la MAC a los informáticos del IMSE. Para obtener la MAC hay que entrar con Putty (Figura 108) a la placa y escribir:

```
ip -a link
```

Ahí debería devolver la MAC de la interfaz eth0. Lo único que haría falta es saber la dirección de la placa, para ello, desde Putty:

```
ip address
```

Y ahí debería haber una dirección 150.214.x.x. Para conectarse, hay que usar esa dirección en un navegador más el término :9090, es decir, "150.214.x.x:9090". Ahí se pedirá contraseña: "xilinx". El nombre de usuario/contraseña del sistema en general es "xilinx"/"xilinx", puede ser útil desde Putty para ejecutar comandos o similar.

```
pynq login: xilinx

Password:
```

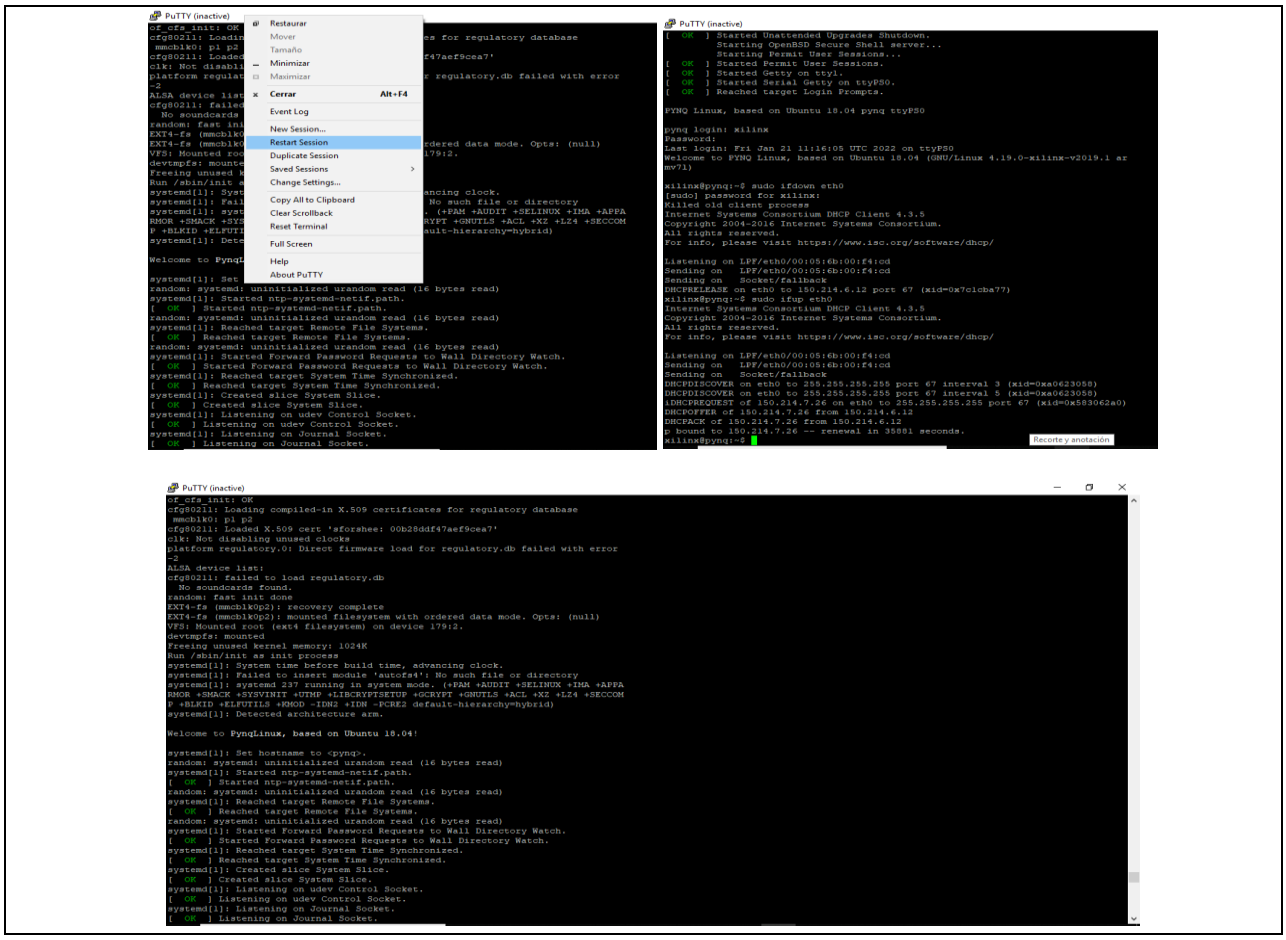


Figura 108. Se encontraron dificultades para conectar la placa. Para ello se usa Putty, pues se programa con la red de internet. Se tuvo que dar de alta la placa a la red del instituto de microelectrónica, comprobando su dirección de IP: xilinx@pynq:~$ ip a

Para desconectar de la red:

```
sudo ifdown eth0
```

Para conectar a la red (Figura 109):

```
sudo ifup eth0
```

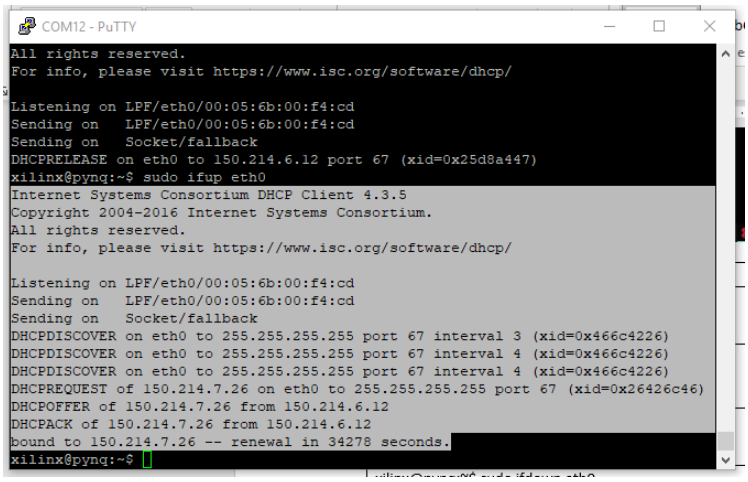


Figura 109. Resultado al conectar placa a la red.

Hubo al principio algunos problemas con la conexión por la versión de Ubuntu instalada en la placa y que no cogía correctamente la IP, pero se solucionó. Para programar la placa se usó Jupyter (es online) y Vivado. Para programar la parte de Python (Jupyter) (Figura 110, Figura 111) se puso en buscador de Chrome:

http://150.214.7.26:9090/notebooks/pynq_tutorial/Tutorial%20GPIO.ipynb

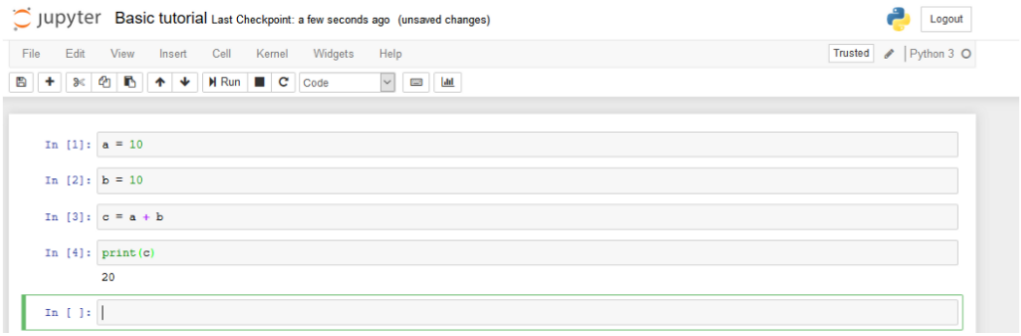


Figura 110. Ejemplo de uso de Jupyter: "*Intenta agregar cuatro celdas. Escriba el código como se muestra en la figura y presione Shift + Enter, lo que debería generar 20 después de la celda 4"* [104]. Fuente: [104].

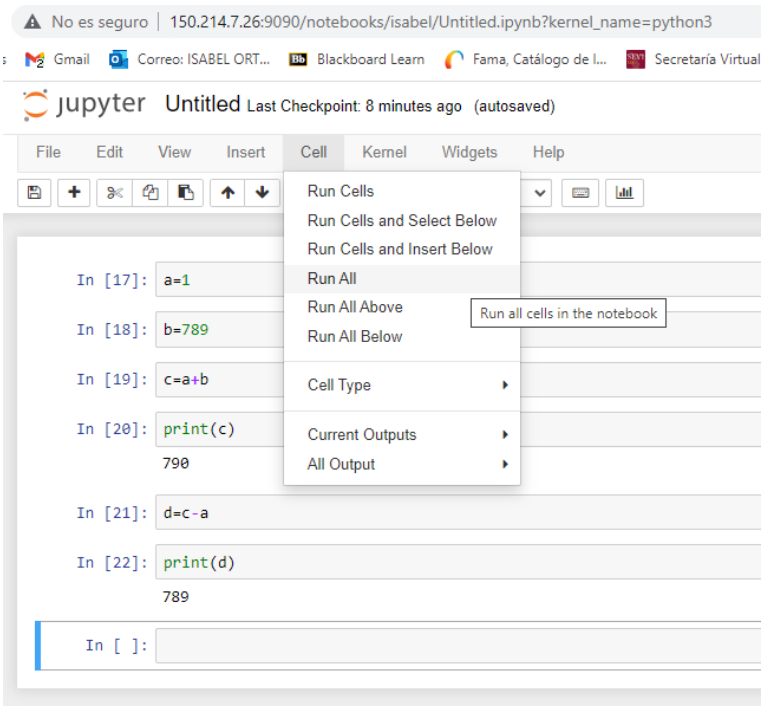


Figura 111. Se pueden ejecutar todas las celdas dándole a Run All.

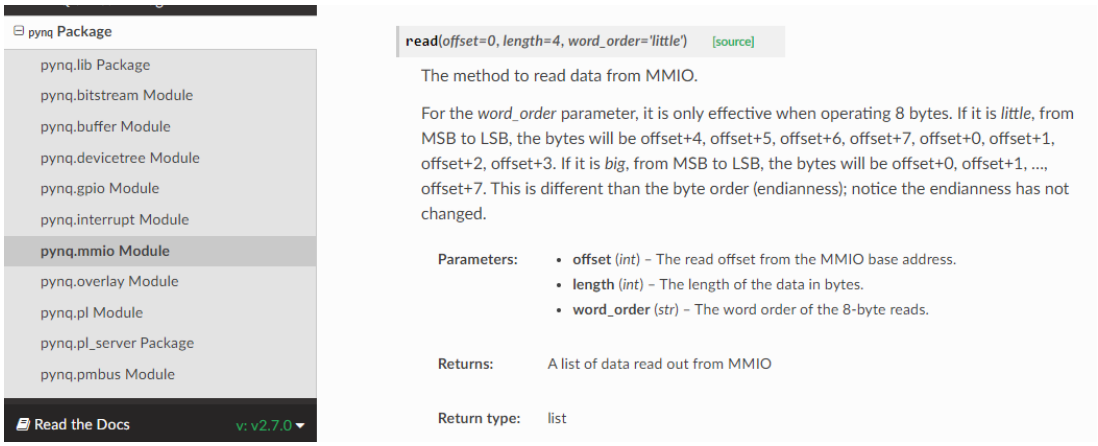


Figura 112. Para realizar ejemplos de tutoriales se fue buscando que era lo que realizaba cada función. Fuente: [120].

Para realizar ejemplos de tutoriales se fue buscando que era lo que realizaba cada función en [120]. Por ejemplo, la función **read(*offset=0, length=4, word_order='little'*)** es el método para leer datos de MMIO (Figura 112).

El Vivado utilizado fue la versión de 2020 (en principio se usó el de 2021, pero daba problemas en el ordenador usado). En él no aparece por defecto la Pynq Z2, para hacer uso de las placas hay que descargar un archivo (Figura 113) que viene en la página del fabricante [121], su contenido se descomprime en la carpeta *...\Xilinx\Vivado\20xx.x\data\boards\board_files/*. Se recomienda mirar el punto 3 del tutorial [122].

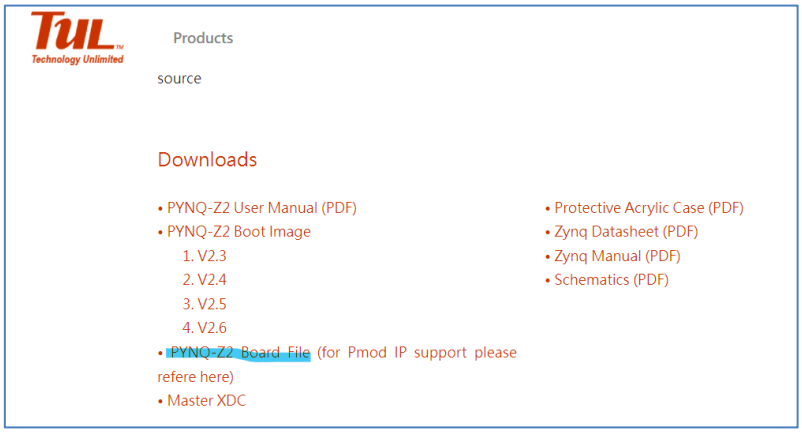


Figura 113. Descarga en página del fabricante. Fuente: [121].

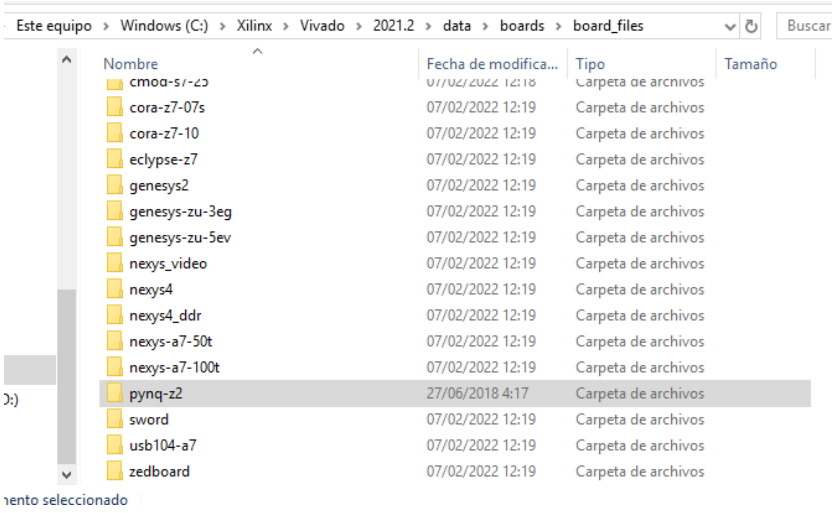


Figura 114. Ubicación donde colocar archivos de placa descargada.

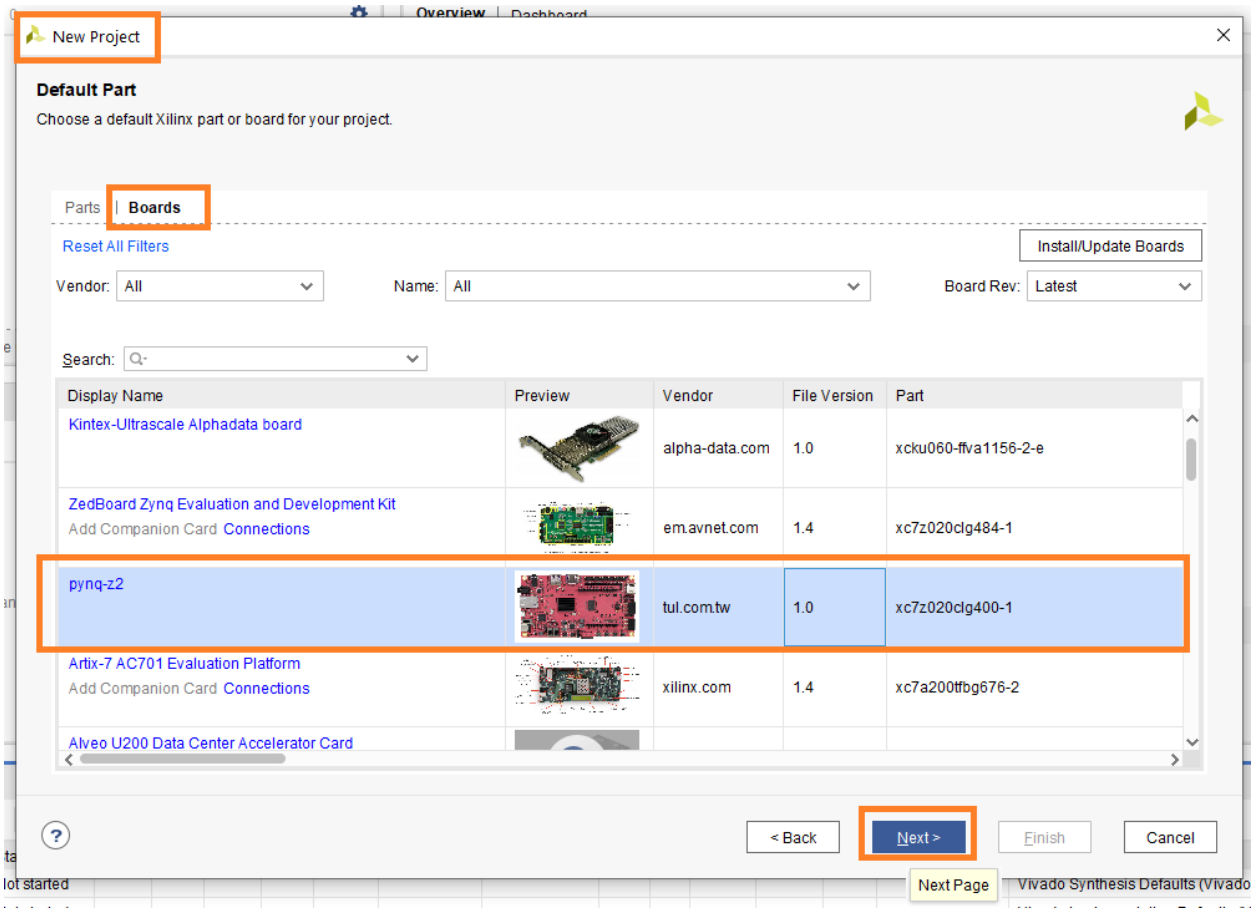


Figura 115. Elección de placa en Vivado. Fuente: [115]

Una vez copiada la carpeta con los datos de la tarjeta (Figura 114), se puede crear un proyecto en Vivado [113] [123], como se muestra en Figura 116.

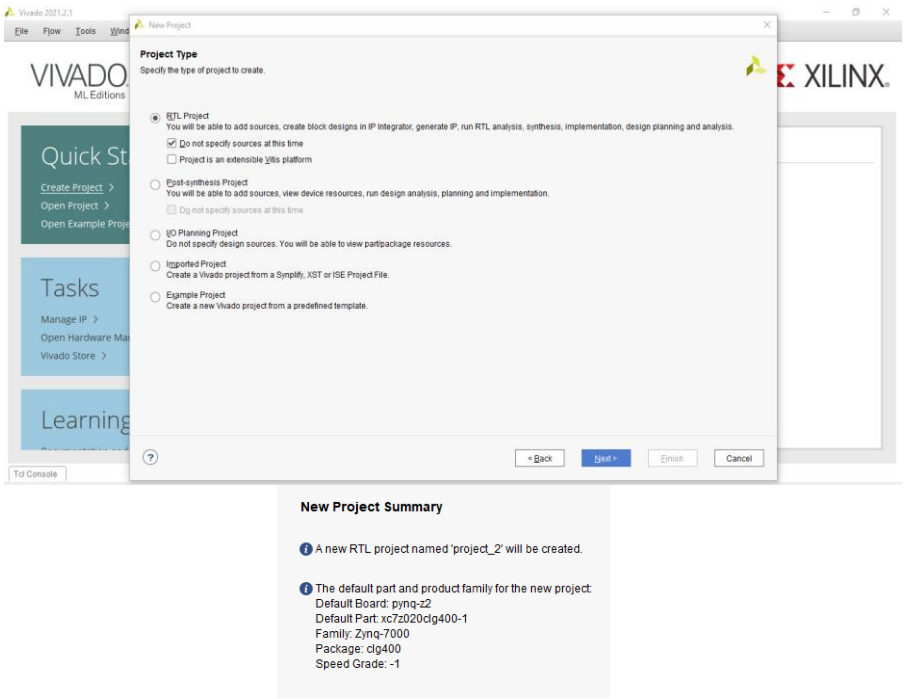


Figura 116. El RTL es VHDL a nivel de transferencia de registros. A la derecha se muestran los datos sobre el nuevo proyecto.

Una vez creado el proyecto, se crea el bloque de diseño (Figura 117). En la barra de menú del diagrama de bloques, se busca y selecciona ZYNQ7 Processing System (PS) (Figura 118), haciendo doble clic para agregarlo al diseño. Al agregar el bloque Zynq PS, debería aparecer un mensaje dando la opción de ejecutar la automatización del bloque (Figura 119), habrá que darle clic para ejecutar este proceso (Figura 120). Esta opción establece el preajuste de la placa en el Sistema de Procesamiento. Todas las propiedades actuales serán sobrescritas y esta acción no puede deshacerse. La automatización del bloque Zynq7 aplica el preajuste actual de la placa y genera conexiones externas para las interfaces FIXED_IO, Trigger y DDR. Al aplicar el preajuste de la placa se descartará la configuración IP existente; hay que desactivar esta casilla si desea conservar la configuración anterior. Una vez terminado esto, estará listo para empezar a crear el diseño (Figura 121).

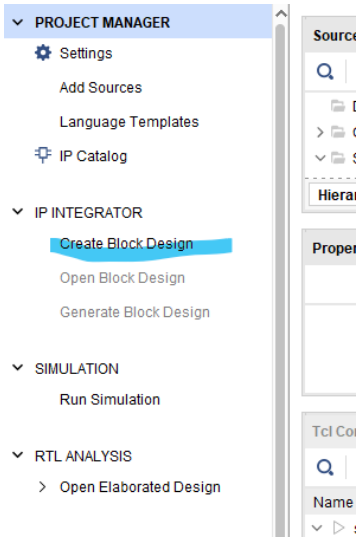


Figura 117. Crear bloque de diseño.

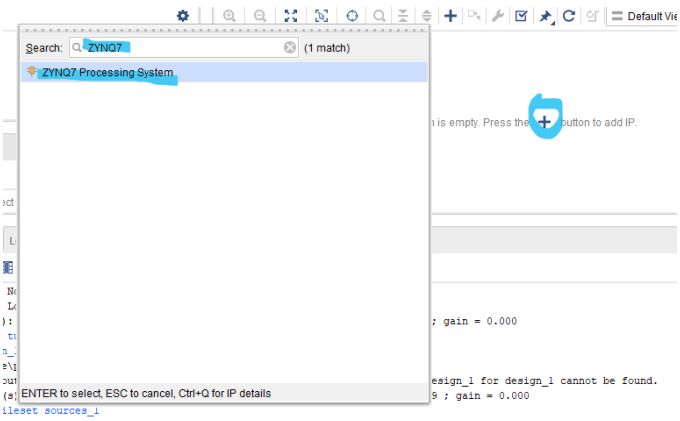


Figura 118. Anadir ZYNQ7 Processing System.

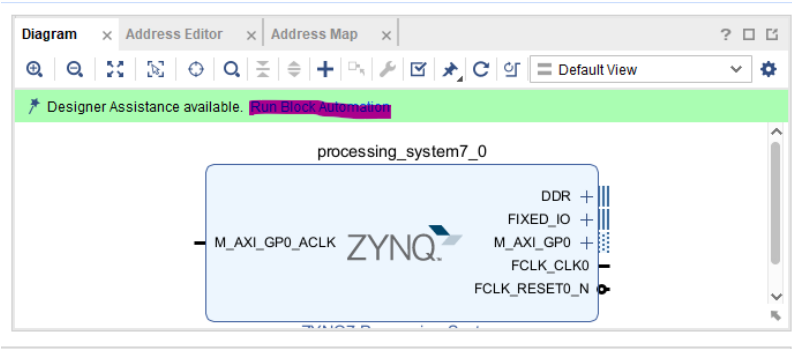


Figura 119. Ejecutar automatización del bloque. Esto aplica una configuración predeterminada para el PS de su placa.

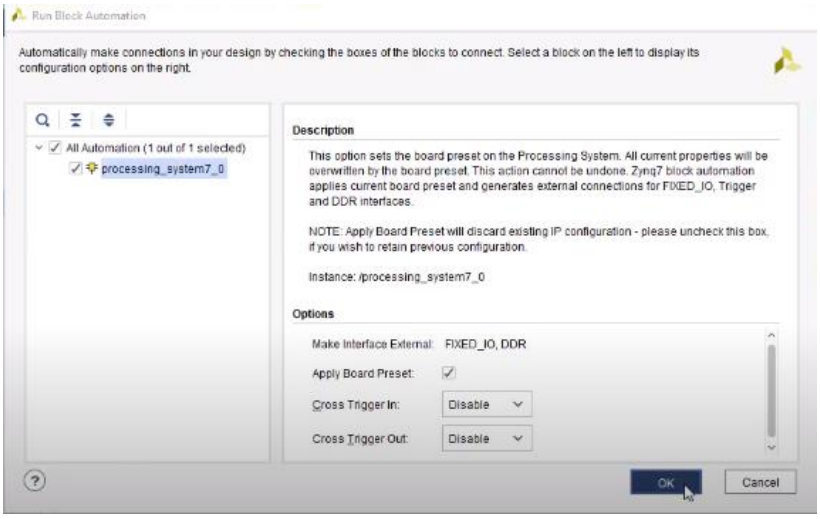


Figura 120. Automatización de bloque.

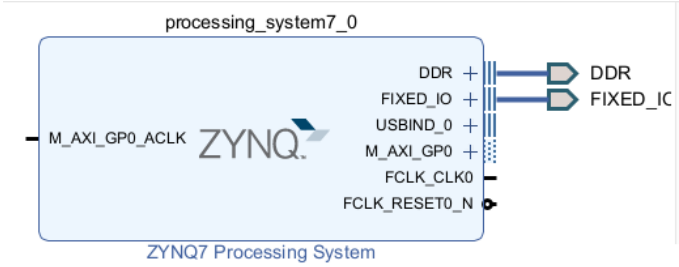


Figura 121. Placa preparada.

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

ÍNDICE DE CÓDIGOS

GLOSARIO

A

AER
Address Event Representation, 35, 36, 40
ANN
Redes neuronales artificiales, 27, 31, 39
APS
Sensor de píxeles activos, 23
AR
Realidad Aumentada, 26

B

BCI
Brain Computer Interface, 93

C

CNN
Redes neuronales convolucionales, 31, 34, 39
CONV
Capa convolucional, 33
ConvModule
Módulo de Convolución Dirigida por Eventos, 34, 35, 36, 40
ConvNets
Redes neuronales convolucionales, 31

D

DAVIS
Sensor de visión dinámica y de píxeles activos, 23, 24
DVS
Sensor de visión dinámica, 20, 21, 22, 23, 37, 38, 39, 43

F

FC
Fully connected, 33
FM
Feature Maps, 35, 40
FPGA
Field-Programmable Gate Arrays, 39, 96, 97

I

IMSE
Instituto de Microelectrónica de Sevilla, 47, 87, 122, 134, 136
IoT
Internet of Things, 26

L

LIDAR
Light Detection and Ranging o Laser Imaging Detection and Ranging, 37
LIF
Modelo fuga-integración-disparo, 29, 39, 43

M

MVSEC
cámara de eventos estereoscópicos de múltiples vehículos, 37

P

PC
Personal Computer, 43
POOL
Capa Pooling, 33

R

RNN
Red neuronal recurrente, 31
RV
Realidad Virtual, 42

S

SNN
Spiking Neural Network, 20, 27, 28, 30, 31, 32, 37, 38, 39, 42, 43, 93
spike
Evento o pulso, 43

STDP
Plasticidad dependiente del tiempo de los picos, 31, 32, 39
StereoSpike
Enfoque neuromórfico de extremo a extremo, que combina dos cámaras basadas en eventos y una red neuronal de Spiking Neural Network, 37, 38

V

VR
Realidad Virtual, 26