



UCA

Universidad
de Cádiz

Escuela Superior
de Ingeniería

TRABAJO DE FIN DE GRADO

GRADO EN INGENIERÍA INFORMÁTICA

ACELERACIÓN DE CÓDIGO C++ MODERNO USANDO HLS PARA FPGA EN EL CAMPO DE VISIÓN ARTIFICIAL

AUTOR: ANTONIO BORRERO FONCUBIERTA

Cádiz, Septiembre 2021



UCA

Universidad
de Cádiz

Escuela Superior
de Ingeniería

TRABAJO DE FIN DE GRADO

GRADO EN INGENIERÍA INFORMÁTICA

ACELERACIÓN DE CÓDIGO C++ MODERNO USANDO HLS PARA FPGA EN EL CAMPO DE VISIÓN ARTIFICIAL

DIRECTORA: MARÍA ÁNGELES CIFREDO CHACÓN
CODIRECTORA: MARÍA MERCEDES RODRÍGUEZ GARCÍA
AUTOR: ANTONIO BORRERO FONCUBIERTA

Cádiz, Septiembre 2021

DECLARACIÓN PERSONAL DE AUTORIA

Antonio Borrero Foncubierta con DNI , estudiante del Grado en Ingeniería Informática en la Escuela Superior de Ingeniería de la Universidad de Cádiz, como autor de este documento académico titulado «Aceleración de código C++ moderno usando HLS para FPGA en el campo de visión artificial» y presentado como Trabajo Final de Grado

DECLARO QUE

Es un trabajo original, que no copio ni utilizo parte de obra alguna sin mencionar de forma clara y precisa su origen tanto en el cuerpo del texto como su bibliografía y que no empleo datos de tercero sin la debida autorización, de acuerdo con la legislación vigente. Asimismo, declaro que soy plenamente consciente de que no respetar esta obligación podrá implicar la aplicación de sanciones académicas, sin perjuicio de otras actuaciones que pudieran iniciarse.

En Puerto Real, a 6 de Septiembre, 2021.

Fdo: Antonio Borrero Foncubierta

Agradecimientos

En primer lugar, me gustaría agradecer a mis tutoras Míriam y Mercedes por haber puesto su confianza en mí para el desarrollo de este proyecto, y por su invaluable ayuda y consejos.

A mis compañeros de trabajo por hacerme ver el potencial de la visión artificial y su alcance.

A mis compañeros de grado, con los que he vivido una experiencia inolvidable que siempre recordaré como los mejores años de mi vida.

Y a mi familia, por su constante apoyo y dedicación a mis estudios y mejora personal.

Resumen

Se documenta la creación de una librería de detección de movimiento en tiempo real en vídeo basada en C++ moderno sin dependencias externas, y su incremental adaptación a hardware FPGA Xilinx usando el paradigma High Level Synthesis (HLS). Se desarrollan los distintos filtros y algoritmos de procesamiento de imágenes tanto para CPU como para FPGA, y se contrastan las diferencias de diseño, eficiencia, velocidad y complejidad de éstos. Adicionalmente, se muestran las mejoras obtenidas en el diseño hardware sintetizado al aplicar diversas directivas HLS de optimización de uso de recursos y de rendimiento.

Palabras claves: HLS, FPGA, procesamiento de imágenes

Índice general

I Prolegómeno	9
1. Introducción	10
1.1. Motivación	11
1.2. Estado del arte	11
1.2.1. Plataformas de procesamiento	11
1.2.1.1. CPU	12
1.2.1.2. GPU	12
1.2.1.3. FPGA	13
1.2.1.4. TPU	14
1.2.1.5. Selección de arquitectura de procesamiento	15
1.2.2. Desarrollo para FPGA	15
1.2.2.1. Lenguajes HDL	16
1.2.2.1.1. Historia	16
1.2.2.1.2. RTL	16
1.2.2.2. VHDL y Verilog	17
1.2.2.2.1. Núcleos IP	17
1.2.2.3. Paradigma HLS	18
1.2.2.3.1. Historia	18
1.2.2.3.2. Lenguajes	19
1.2.2.3.3. Aplicaciones	20
1.2.2.3.4. Restricciones	20
1.2.2.3.5. Simulación	21
1.2.2.4. Plataformas de desarrollo para soluciones FPGA . .	22
1.2.2.5. Selección de metodología	22
1.2.3. Procesamiento de imágenes	23
1.2.3.1. Librerías	24
1.2.3.2. Aceleración	25
1.2.3.2.1. CPU independiente	25
1.2.3.2.2. CPU y GPU	27

1.2.3.2.3. FPGA independiente	27
1.2.3.2.4. CPU y FPGA	28
1.2.3.2.5. CPU, GPU y FPGA	29
1.2.4. Codificación C++ moderna	29
1.3. Objetivos y alcance del proyecto	30
1.4. Glosario de términos	31
1.5. Estructura del documento	31
2. Planificación	33
2.1. Metodología de desarrollo	33
2.2. Planificación del proyecto	33
2.3. Organización	35
2.4. Costes	35
2.4.1. Humanos	35
2.4.2. Materiales	37
2.4.3. Total	37
2.5. Riesgos	38
II Desarrollo	40
3. Planteamiento y Análisis	41
3.1. Descripción de la solución	41
3.2. Actores	41
3.3. Requisitos	42
3.3.1. Funcionales	42
3.3.1.1. Descripción de requisitos funcionales CPU	43
3.3.1.1.1. RF0 - Crear detector de movimiento	44
3.3.1.1.2. RF1 - Destruir detector de movimiento	44
3.3.1.1.3. RF2 - Añadir fotograma	45
3.3.1.1.4. RF3 - Obtener movimiento	46
3.3.1.2. Descripción de requisitos funcionales FPGA	47
3.3.1.2.1. RF0 - Procesar fotograma	47
3.3.2. No funcionales	47
3.3.2.1. RNF0 - Rendimiento	47
3.3.2.2. RNF1 - Uso de recursos	48
3.3.2.3. RNF2 - Robustez	48
3.3.2.4. RNF3 - Fiabilidad	48
3.3.2.5. RNF4 - Usabilidad	48
3.3.2.6. RNF5 - Mantenibilidad	49
3.3.2.7. RNF6 - Portabilidad	49

4. Diseño del sistema	50
4.1. Arquitectura física	50
4.1.1. Raspberry Pi 4B	50
4.1.2. Cámara Raspberry Pi Full HD	52
4.1.3. Chip FPGA	52
4.2. Arquitectura lógica	54
4.2.1. Lógica CPU independiente	54
4.2.2. Lógica FPGA independiente	55
5. Implementación	56
5.1. Librería base	56
5.1.1. Teoría de detección de movimiento	56
5.1.1.1. Transformación a escala de grises	57
5.1.1.2. Reducción de resolución	57
5.1.1.3. Desenfoque Gaussiano	58
5.1.1.4. Sustracción de fondo	60
5.1.1.5. Umbralización	61
5.1.1.6. Dilatación	63
5.1.1.7. Detección de componentes conexas	64
5.1.2. Implementación de la librería	65
5.1.2.1. Estructura y herramientas de automatización	65
5.1.2.1.1. Make	66
5.1.2.1.2. CMake	66
5.1.2.1.3. Git	66
5.1.2.1.4. Doxygen	67
5.1.2.1.5. Estructura de la librería	67
5.1.2.2. Interfaz pública	69
5.1.2.2.1. Clase imagen	69
5.1.2.2.2. Clase detector de movimiento	70
5.1.2.2.3. Pipeline principal	71
5.1.2.2.4. Transformación a escala de grises	72
5.1.2.3. Algoritmos de procesamiento de imágenes	72
5.1.2.3.1. Reducción de resolución	72
5.1.2.3.2. Desenfoque Gaussiano	74
5.1.2.3.3. Procesado de la referencia	75
5.1.2.3.4. Umbralización	75
5.1.2.3.5. Dilatación	76
5.1.2.3.6. Detección de contornos	76
5.1.2.4. Diseño de la librería	77
5.1.2.5. Optimizaciones	77
5.2. Adaptación a FPGA HLS	78

5.2.1.	Teoría HLS	78
5.2.1.1.	Métricas de velocidad	78
5.2.1.2.	Pragma Pipeline	80
5.2.1.3.	Pragma Dataflow	81
5.2.1.4.	Pragma Unroll	82
5.2.1.5.	Pragma Array Partition	82
5.2.1.6.	Pragma Dependence	83
5.2.1.7.	Pragma Interface	83
5.2.1.8.	Pragma Loop Tripcount	84
5.2.2.	Proceso de adaptación	84
5.2.2.1.	Eliminación de llamadas al sistema y código no usado	84
5.2.2.2.	Eliminación de incompatibilidades	85
5.2.2.3.	Síntesis inicial	86
5.2.2.4.	Streaming de imágenes	86
5.2.2.4.1.	Rediseño de la reducción de resolución	87
5.2.2.4.2.	Rediseño del desenfoque Gaussiano	88
5.2.2.4.3.	Rediseño de la interpolación, sustracción de fondo y umbralización	88
5.2.2.4.4.	Rediseño de la dilatación	88
5.2.2.4.5.	Rediseño de la detección de contornos	89
5.2.2.5.	Mejoras de eficiencia	89
5.2.2.6.	Simulación RTL y optimizaciones finales	92
6.	Pruebas	95
6.1.	Estrategia	95
6.2.	Entorno de pruebas	96
6.3.	Niveles de prueba	96
6.3.1.	Pruebas unitarias	96
6.3.2.	Pruebas de integración	96
6.3.3.	Pruebas del sistema	96
6.4.	Corrección de errores	98
7.	Resultados	100
7.1.	Resultados CPU	100
7.2.	Resultados FPGA	101
III	Epílogo	103
7.3.	Conclusiones	104
7.3.1.	Objetivos	104
7.3.2.	Aprendizaje	104

7.3.3. Mejoras del proyecto	105
7.3.4. Trabajo futuro	106
IV Anexos	107
7.4. Manual de usuario	108
7.4.1. Introducción	108
7.4.2. Requisitos software	108
7.4.2.1. Librerías para CPU	108
7.4.2.2. Librerías para FPGA	108
7.4.3. Requisitos hardware	108
7.4.4. Archivos	109
7.4.5. Instrucciones de prueba	110
7.4.5.1. Librerías CPU	110
7.4.5.2. Programas Piloto CPU	111
7.4.5.3. Librerías HLS	112
7.5. Apéndice	113
7.5.1. Código librería CPU base	113
7.5.1.1. CMakeLists.txt	113
7.5.1.2. motion_detector.hpp	115
7.5.1.3. motion_detector.cpp	123
7.5.1.4. image_utils.hpp	129
7.5.1.5. image_utils.ipp	136
7.5.1.6. contour_detector.hpp	147
7.5.1.7. contour_detector.cpp	148
7.5.2.1. CMakeLists.txt	154
7.5.2.2. main.cpp	155
7.5.2. Código programa piloto Save_to_disk	154
7.5.3.1. main.cpp	160
7.5.3.2. motion_detector.hpp	163
7.5.3.3. motion_detector.cpp	164
7.5.3.4. image_utils.hpp	165
7.5.3.5. image_utils.cpp	166
7.5.3.6. contour_detector.hpp	172
7.5.3.7. contour_detector.cpp	173

Índice de figuras

1.1.	Estructura interna de una CPU	13
1.2.	Estructura interna de una GPU	14
1.3.	Diferencias en la sintaxis de VHDL y Verilog.	17
1.4.	Xilinx Vitis e Intel OneAPI	23
1.5.	Representación RGB y HSV de un píxel de color.	24
1.6.	Pipelining de instrucciones en CPU	26
2.1.	Planificación del proyecto.	36
3.1.	Requisitos funcionales del sistema.	43
4.1.	Una Raspberry Pi 4B y el logo representativo de la marca.	51
4.2.	La placa de desarrollo objetivo y su chip FPGA integrado.	53
4.3.	Arquitectura lógica CPU	54
5.1.	Transformación de RGB a escala de grises.	57
5.2.	Reducción de resolución de una imagen.	58
5.3.	Aplicación de desenfoque Gaussiano.	59
5.4.	Métodos de manejo de bordes para filtros convolucionales.	59
5.5.	Sustracción de fondo.	61
5.6.	Aplicación de umbral doble e histéresis.	62
5.7.	Las operaciones morfológicas de dilatación y erosión.	63
5.8.	Aplicación de un algoritmo de componentes conexas.	65
5.9.	Logo CMake.	67
5.10.	Ejemplo documentación Doxygen.	68
5.11.	Modelo asíncrono de detección de movimiento.	71
5.12.	Pipeline completo de detección de movimiento.	73
5.13.	Algoritmo de downsampling.	74
5.14.	Operaciones por ciclo en FPGA.	79
5.15.	El pragma Pipeline en HLS.	80
5.16.	El pragma Dataflow en HLS.	81

5.17. Uso de recursos en la primera síntesis de la librería.	87
5.18. Uso de recursos tras usar streams.	89
5.19. Rendimiento base de la implementación HLS.	90
5.20. Rendimiento base conocido de la implementación HLS.	90
5.21. Rendimiento con Pipeline y Dataflow de la implementación HLS. . .	91
5.22. Rendimiento optimizado de la implementación HLS.	92
5.23. El efecto del uso de precisión arbitraria.	94
6.1. Pruebas unitarias de la librería.	97
6.2. Pruebas de uso de memoria usando Valgrind.	97
6.3. Test detección movimiento.	98
6.4. Errores encontrados.	99
7.1. Configuración inicial de un proyecto HLS usando la interfaz gráfica. .	113

Índice de tablas

2.1.	Coste de recursos humanos.	37
2.2.	Coste de recursos materiales.	37
2.3.	Coste total del proyecto.	38
4.1.	Características hardware de una Raspberry Pi 4B 8GB.	51
4.2.	Características del chip FPGA XC7Z020-1CLG400C.	53
7.1.	Métricas de rendimiento en CPU de sobremesa.	101
7.2.	Métricas de rendimiento en Raspberry Pi 4B.	102
7.3.	Métricas de uso de recursos finales en FPGA.	102

Parte I

Prolegómeno

Capítulo 1

Introducción

Cada día en el mundo digital en el que vivimos crecen más los requerimientos de procesamiento de las aplicaciones informáticas que nos rodean, y por ende, los requerimientos hardware de los sistemas que las soportan.

A medida que las CPUs dejaron de ser capaces de procesar datos masivos en tiempos adecuados como sería el procesamiento de gráficos o el entrenamiento y ejecución de redes neuronales, se hizo más prominente la necesidad de descargar estas tareas a aceleradores hardware externos como son las GPUs (Graphics Processing Unit), las FPGAs (Field Programmable Gate Array) o las TPUs (Tensor Processing Unit).

Cuando se necesita ejecutar una tarea altamente paralelizable genérica, como puede ser el procesamiento de gráficos 3D de un videojuego o cálculos masivos de tensores, usar una GPU es la opción seleccionada por defecto debido a su sencillez. Como prueba de esto es la extensiva adopción de GPUs en la gran mayoría de ordenadores de sobremesa, workstations, clústeres de procesamiento o mining rigs para criptomonedas.

Pero también es un caso de uso común el procesamiento de datos en sistemas empotrados, con unos requerimientos de eficiencia y latencia en los que cada nanosegundo o vatio adicional importan, como puede ser en fábricas robotizadas, aplicaciones militares y aeroespaciales, vehículos automáticos o el trading de alta frecuencia. Estos campos, entre muchos otros, comparten la necesidad de tener hardware especializado en tareas muy concretas que deben ser realizadas con alta confiabilidad y velocidad, unos requisitos que son a menudo inalcanzables por una CPU o GPU[1][2]. Es en estos casos de uso en el que brillan las FPGAs, y es por ello por lo que se estudiarán en mayor profundidad en este trabajo.

1.1 Motivación

Si bien es cierto que las FPGAs son usadas en una gran variedad de escenarios, sigue siendo un campo al que raramente se adentran los ingenieros informáticos y suele ser relevado a otras especialidades más cercanas a la electrónica.

La principal razón de este fenómeno es la necesidad de conocimientos electrónicos para crear diseños efectivos, ya que el comportamiento de hardware es generalmente muy distinto al comportamiento de software. Los lenguajes de definición de hardware (HDL) más utilizados, Verilog y VHDL, se pueden ver como el equivalente al lenguaje máquina usado en programación para CPUs, al definir en gran detalle la lógica que se desea obtener en la FPGA y los componentes electrónicos que se usarán para ello (LUTs, Flip Flops, RAM de bloque, registros, etc...), que es un nivel al que raramente trabaja un desarrollador de software.

En recientes años ha ganado tracción un paradigma de programación de FPGAs que habilita el uso de lenguajes de alto nivel como puede ser C, C++ o Python, facilitando la entrada desarrolladores de software en el mundo de las FPGAs, además de ayudar a diseñadores digitales de un gran abanico de niveles de experiencia a prototipar ideas de una forma rápida y elegante. A esta tecnología se le denomina High Level Synthesis, comúnmente conocido por su acrónimo HLS.

No sólo HLS es una herramienta de prototipado y una puerta de entrada para nuevos desarrolladores, sino que también es una gran herramienta para expertos que buscan una forma de acelerar la producción de sistemas complejos de gran escala, ya que tanto Verilog como VHDL requieren tiempos de desarrollo muy largos debido al bajo nivel del código, las dificultades de testeo y el gran porcentaje de tiempo muerto usado al sintetizar el código.[3]

Este trabajo busca dar a conocer esta tecnología a ingenieros informáticos mediante la adaptación de código de procesamiento de vídeo a FPGA, siendo este originalmente destinado para su ejecución en CPU. Se irá paso por paso, usando el paradigma HLS, sustituyendo los algoritmos y estructuras por otras más adecuadas para su ejecución en hardware digital.

1.2 Estado del arte

1.2.1 Plataformas de procesamiento

Cuando se embarca en el desarrollo de un nuevo producto, servicio o tecnología es clave saber qué plataforma de procesamiento será la base de este, o más comúnmen-

te, qué combinación de estas se utilizará. En la actualidad se dispone de cuatro principales opciones: CPU, GPU, FPGA y TPU.[4]

1.2.1.1 CPU

Una unidad central de procesamiento, más comúnmente conocida por su acrónimo CPU, es la plataforma de procesamiento más antigua de las cuatro mencionadas, siendo el núcleo lógico de todos los ordenadores de trabajo y personales en la actualidad.[5]

Las CPUs han llegado a ver un uso tan extendido gracias a su flexibilidad y facilidad de programación, caracterizadas por ser completamente reconfigurables por software y por ser capaces de procesar y ejecutar instrucciones genéricas, siendo esencial en una enorme variedad de campos y aplicaciones.[5][4]

Si bien es cierto que la capacidad de procesamiento de una CPU es elevada gracias a los distintos métodos de paralelismo, cachés y muchas otras optimizaciones, su naturaleza genérica impone restricciones en su arquitectura. Estas restricciones, principalmente, se traducen a un reducido número de núcleos de procesamiento muy complejos como se observa en la Figura 1.1, limitando su capacidad de procesamiento de datos masivos, como puede ser el procesamiento de gráficos o cálculo de tensores para aprendizaje profundo.[4]

1.2.1.2 GPU

Una unidad de procesamiento gráfico (GPU), como su nombre indica, es un tipo de procesador especializado en el procesamiento de gráficos y, junto con la CPU, está presente en la gran mayoría de los ordenadores de trabajo y personales del mundo.

Las GPUs son programables por software y consiguen procesar gráficos a altas velocidades gracias a su elevado número de núcleos de procesamiento, que son mucho más simples que los que se encuentran en CPUs, como se puede observar en la Figura 1.2.

Una GPU, al tener núcleos con un reducido número de instrucciones y funcionalidades, es mucho menos flexible que una CPU y requiere de lenguajes de programación especializados como CUDA o OpenGL.

Es un patrón muy común tener una CPU como procesador principal que se encargue de procesar lógica compleja y variada, mientras que una GPU se encarga de las

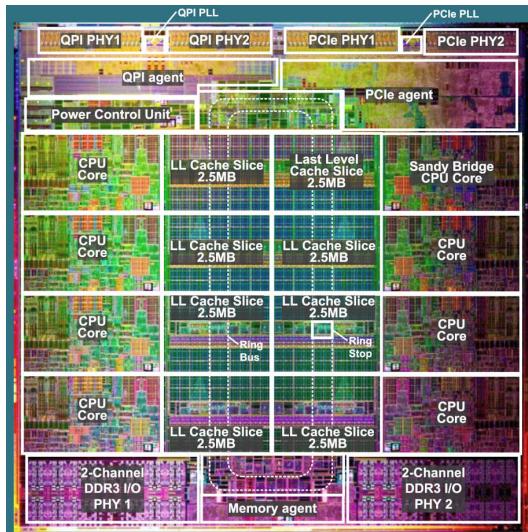


Figura 1.1: Arquitectura interna de una CPU en la que se puede ver la complejidad en el reducido número de núcleos. Intel Sandy Bridge, 2011.

Origen: Imagen obtenida de *CPU Shack* (<https://www.cpushack.com/2018/03/24/making-multicore-a-slice-of-sandy/sb-eplayout/>).

operaciones sobre datos masivos. Este patrón es tan común que en la actualidad la gran mayoría de CPUs tiene una GPU integrada dentro de ellas, aunque no son tan potentes como las placas auxiliares conectadas a los puertos de expansión del computador, comúnmente conocidas como *tarjetas gráficas*.

1.2.1.3 FPGA

Una matriz de puertas lógicas programables en campo, FPGA, es un chip que incluye distintos recursos hardware genéricos para realizar una tarea concreta y es programado usando lenguajes de descripción de hardware (HDL).

Una FPGA permite definir un circuito digital con el comportamiento deseado, siendo el diseño final completamente especializado a la tarea definida y muy eficiente en el uso de recursos disponibles.

En cierta forma, se puede ver a las FPGAs como chips capaces de emular circuitos integrados de aplicación específica (ASIC), pero son más lentas, usan hardware más simple y tienen un mayor consumo eléctrico que estos. Es decir, si se configura una FPGA para emular la arquitectura de una GPU, esta será notablemente inferior al circuito real, y es por este motivo por lo que la ventaja de una FPGA no es su velocidad bruta, sino su habilidad de especializarse lo suficiente al caso de uso como para superar a soluciones implementadas con arquitecturas más genéricas.[6]

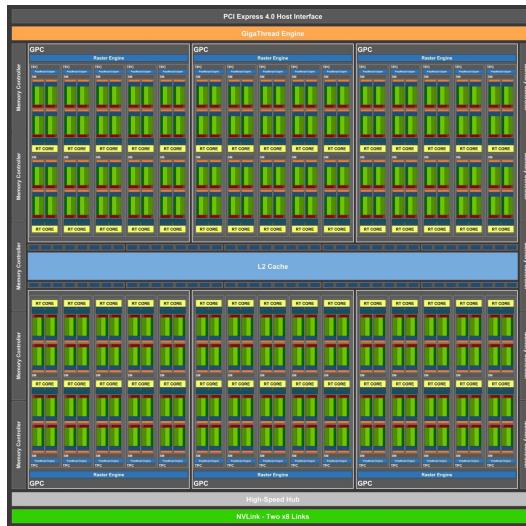


Figura 1.2: Arquitectura interna de una GPU en la que se puede ver el gran número de núcleos y bloques de procesamiento. Nvidia Ampere, 2020.

Origen: Imagen obtenida de Tech Centurion (<https://www.techcenturion.com/nvidia-ampere>).

Si bien es cierto que las FPGAs pueden superar a otras arquitecturas gracias a su capacidad de reconfiguración y especialización, esto también es su mayor desventaja, ya que existe una necesidad de desarrollar diseños hardware altamente especializados para el caso de uso, con las restricciones que ello impone.[7]

Las FPGAs, a diferencia de las CPUs y GPUs, raramente se encuentran en ordenadores personales, pero cada vez tienen mayor presencia en clústeres de procesamiento, aplicaciones industriales y militares, o para prototipado de nuevos ASIC, los cuales tienen un tiempo y coste de desarrollo a menudo prohibitivo para muchas empresas.[6]

1.2.1.4 TPU

Una unidad de procesamiento de tensores, TPU, es un procesador relativamente reciente desarrollado por Google para la aceleración de cálculo de tensores, es decir, de matrices y vectores ya que estos son la base del aprendizaje profundo y la inteligencia artificial moderna.

Fue el reciente éxito y la extensa adopción de la inteligencia artificial alrededor del mundo el promotor del desarrollo de esta nueva gama de ASICs, que pueden ser encontrados comúnmente en clústeres y dispositivos de borde que deban ejecutar y/o entrenar redes neuronales de forma eficiente y rápida.[8]

De una forma muy similar a las GPUs, los TPUs se suelen encontrar como aceleradores externos controlados por una CPU, aunque no suelen estar presentes en ordenadores personales en la actualidad.

1.2.1.5 Selección de arquitectura de procesamiento

Seleccionar las tecnologías que soportarán la aplicación o servicio que se desea desarrollar es una de las decisiones más importantes de un proyecto, que a menudo no tiene una respuesta clara e inmediata. Según las necesidades, presupuesto y objetivos de cada proyecto puede variar la arquitectura seleccionada, o la combinación de estas.

En el campo del procesado de imágenes existen dos grandes pilares, las GPUs y las FPGAs. En el caso de que nuestra aplicación deba funcionar en un ordenador utilitario, la opción por defecto es el uso de GPU debido a su extensiva adopción y flexibilidad, pero en cambio, si la aplicación está destinada a funcionar en sistemas embebidos o clústeres, será necesario un estudio en mayor profundidad de ambas opciones.

Las GPUs ofrecen una mayor facilidad de implementación de software y mucha flexibilidad, mientras que las FPGAs son mucho más complejas de programar y disponen de una reducida comunidad de desarrolladores en comparación.[7][6]

Si bien es cierto que las FPGAs ofrecen una notable mejora de rendimiento y uso de recursos al operar sobre secuencias de procesamiento de imágenes complejas sobre otras soluciones basadas en GPU, el tiempo y esfuerzo de desarrollo es notablemente mayor, haciendo que su aplicación no sea viable dependiendo del caso de uso.[9]

Es por estas razones que las FPGAs son una opción favorable en el caso de que se necesite máximo rendimiento, mínima latencia y un reducido consumo eléctrico, mientras que las GPUs son la mejor opción si se necesitan buenos resultados en un tiempo de desarrollo corto, siendo posible compensar el reducido rendimiento con mayor replicación (contratación) de hardware.

1.2.2 Desarrollo para FPGA

El desarrollo de diseños digitales para FPGA es un proceso largo y complejo y es a menudo visto como el principal motivo por el cual las FPGAs no se han convertido aún en una solución general de computación, sino que queda reservada para determinados campos y necesidades.[6]

En la actualidad existen dos metodologías principales de desarrollo para FPGAs, los lenguajes de definición de hardware, HDL, y el paradigma de síntesis de alto nivel, HLS.

1.2.2.1 Lenguajes HDL

Los lenguajes HDL son lenguajes especializados en la precisa descripción del comportamiento y estructura de circuitos electrónicos, permitiendo su análisis y simulación.

HDL es parecido a lenguajes de programación como C, conteniendo expresiones, estructuras de control y órdenes, pero adicionalmente incluyendo una noción del tiempo y paralelismo que debe ser controlada en todo momento.

Los lenguajes HDL no sólo son usados para la programación de FPGAs, sino que son una parte elemental del desarrollo de ASICs, microprocesadores y otros circuitos digitales.

1.2.2.1.1 Historia

El primer lenguaje HDL apareció a finales de la década de 1960, introduciendo el concepto de diseño RTL de una forma primitiva. Pero fue alrededor de 10 años más tarde, con el aumento de popularidad de los dispositivos lógicos programables (PLD), familia a la que pertenecen las FPGAs y sus predecesores, y con el avance del diseño de circuitos integrados VLSI, que aparecieron los lenguajes HDL más emblemáticos en la actualidad: Verilog y VHDL.

Estos lenguajes HDL estaban basados en el lenguaje de programación ADA y permitían diseñar a un nivel de abstracción mucho más elevado, habilitando a ingenieros trabajar en proyectos de mayor envergadura.[10]

El paso final a establecer estos dos lenguajes HDL en la industria fue su estandarización por la IEEE y su capacidad para sintetizar el código RTL generados en ellos a circuitos digitales manufacturables, que aunque eran menos eficientes que circuitos diseñados a mano, reducían los tiempos de desarrollo enormemente.

1.2.2.1.2 RTL

El diseño RTL, register-transfer level, es un nivel de abstracción de diseño digital superior al diseño estructural e inferior al diseño algorítmico (HLS) basado en el control de señales digitales entre registros hardware y las operaciones lógicas aplicadas

VHDL:	Verilog:
<pre> process ({S0,S1},A,B,C,D) begin case (S0,S1), is when "00" => Y <= A; when "01" => Y <= B; when "10" => Y <= C; when "11" => Y <= D; when others => Y <= A; end case; end process; </pre>	<pre> always @({S0,S1}, A, B, C, D) case ({S0,S1}) 2'b00: Y = A; 2'b01: Y = B; 2'b10: Y = C; 2'b11: Y = D; endcase </pre>

Figura 1.3: Diferencias en la sintaxis de VHDL y Verilog.

Diferencias en la sintaxis de VHDL y Verilog. Se puede ver la mayor similitud de Verilog a lenguajes de programación modernos como C y su mayor brevedad.

a estas. Es usado por HDLs como VHDL y Verilog para crear representaciones de alto nivel de circuitos, desde las cuales se pueden generar representaciones de nivel inferior hasta llegar al nivel de cableado y componentes hardware.

La representación RTL permite definir tanto lógica combinacional (puertas, multiplexores, etc..) y lógica secuencial (flip-flops, registros, etc...), por lo que se pueden implementar una gran variedad de circuitos y diseños, tan simples como un controlador de un botón y LED, o tan complejos como serían las CPUs modernas.

1.2.2.2 VHDL y Verilog

Los dos gigantes de los lenguajes HDL, Verilog y VHDL, tienen a grandes rasgos la misma funcionalidad, siendo la elección de uno u otro definida principalmente por preferencia del equipo de desarrollo.[11]

VHDL es un lenguaje fuertemente tipado, y es más verboso por diseño que Verilog, haciendo que pueda ser más lento desarrollar en el, pero facilitando a su vez el depurado de código.

Verilog es un lenguaje más compacto y natural, y si bien la estructura y lógica del código generado es similar o idéntica, la sintaxis usada es completamente distinta como se puede ver en la Figura 1.3.

1.2.2.2.1 Núcleos IP

Un concepto central del diseño digital son los núcleo de propiedad intelectual, conocidos por su acrónimo IP, que son paquetes que contienen un diseño con una

determinada funcionalidad y pueden ser utilizados en una gran variedad de proyectos.

Estos núcleos IP suelen ser comercializados por empresas y desarrolladores, pero también pueden ser encontrados de forma gratuita en internet, siendo popular la página Web OpenCores entre otras.

Antes de diseñar un nuevo módulo RTL es conveniente revisar la existencia de núcleos IP que contengan la lógica que deseamos implementar, ya que pueden ahorrar cientos de horas de desarrollo siempre que este dentro del presupuesto del proyecto.

1.2.2.3 Paradigma HLS

Aunque HDL elevó considerablemente el nivel de abstracción usado en el diseño digital, los diseños RTL siguen siendo muy detallados y laboriosos, aumentando el tiempo necesario para crear diseños complejos notablemente, y por ende, aumentando el coste del proyecto. Es por esto por lo que surge High Level Synthesis (HLS), que permite definir lógica en lenguajes de alto nivel como puede ser C, C++ o Python de forma que sea directamente sintetizable a RTL, descargando la creación de máquinas de estado finito, sincronización y muchas otras tareas complejas al sintetizador máquina.

Programar en lenguajes de alto nivel permite comprobar la lógica del programa de forma mucho más frecuente, ya que la compilación y ejecución de código de prueba en CPU es generalmente un proceso mucho más rápido que sintetizar el código a un diseño RTL y simular este con herramientas tradicionales.

1.2.2.3.1 Historia

High Level Synthesis es un proceso de diseño que ha sido investigado y desarrollado desde los finales de la década de 1970 hasta hoy, comenzando con primitivos modelos de automatización de creación de máquinas de estado finito, en las que cada estado consiste de una porción de la lógica a ejecutar.

Los lenguajes utilizados para definir la lógica del circuito eran Verilog y VHDL, pero no eran suficientemente aptos para definir comportamientos a alto nivel. Fue a finales de 1990 cuando se introdujo Cynthesizer, que usaba SystemC como lenguaje de entrada, apareciendo el concepto de HLS tal como lo conocemos hoy.

HLS ha pasado por un proceso de incertidumbre similar al desarrollo de compiladores de lenguajes de alto nivel para CPUs: Se ha dudado el potencial y viabilidad

de la tecnología para sustituir los métodos tradicionales de diseño digital, ya que de forma equivalente a la obtención de resultados altamente optimizados usando lenguajes máquina como puede ser x86 sobre los obtenidos con lenguajes a un mayor nivel de abstracción como son C o C++, los diseños RTL obtenidos por desarrolladores experimentados usando Verilog o VHDL son en su gran mayoría superiores en velocidad y eficiencia a los generados por herramientas HLS en la actualidad.[12]

Si bien en la actualidad los compiladores de lenguajes de alto nivel para CPU han eclipsado el uso del código ensamblador por más del 95%[13], este no es el caso para el diseño digital en FPGAs, quedándose HLS reservado principalmente para tareas concretas en los que se conoce su efectividad de antemano o para prototipado donde no se necesitan resultados optimizados.[12]

Aunque la tendencia a usar HDL como el método por defecto para crear diseños RTL para FPGAs prevalece, a medida que se investiga y se hacen nuevos avances, HLS está ganando popularidad para el desarrollo de determinados bloques RTL optimizados para producción y para la exploración de nuevas soluciones a problemas previamente inexplorados debido a los extensos tiempos de desarrollo del diseño RTL.[12]

Uno de los mayores elementos que actualmente está fomentando el avance del paradigma HLS es la competitividad comercial entre distintas compañías, como son Xilinx e Intel. Estos gigantes tecnológicos, en su constante batalla por una cuota de mercado dominante, han hecho grandes avances en tecnologías FPGA tanto en aspectos hardware como software, habilitando plataformas como Vitis por Xilinx y OneAPI por Intel, que unifican soluciones CPU, GPU, FPGA e incluso TPU para obtener los beneficios de cada una de ellas en un mismo proyecto.

En la actualidad HLS es especialmente efectivo en problemas de visión artificial y el procesamiento de imágenes debido a la gran cantidad de librerías disponibles y las optimizaciones que nos ofrecen los sintetizadores modernos, siendo capaces de aplicar procesos complejos a vídeos de alta resolución en tiempo real con un bajo uso de recursos hardware.

1.2.2.3.2 Lenguajes

Desde el lanzamiento de Cynthetizer usando SystemC han aparecido una gran variedad de sintetizadores HLS tanto académicos como comerciales. Estos sintetizadores toman como lenguaje de entrada uno o varios lenguajes, como puede ser SystemC, C, C++, MATLAB, Java, C#, Python y muchos otros.

Si bien existe una gran variedad de sintetizadores y lenguajes de entrada, los más

populares en la actualidad son *Vivado HLS* por Xilinx, que usa C, C++ o SystemC como entrada y genera código VHDL o Verilog como salida, e *Intel High Level Synthesis Compiler* por Intel FPGA, que usa C o C++ para generar código Verilog.

1.2.2.3.3 Aplicaciones

La calidad de los resultados generados por HLS en la actualidad depende del tipo de aplicación y los algoritmos que se estén codificando, siendo algunos de los campos en los que sobresale los siguientes[14]:

- **Media digital:** Para dar soporte a la gran cantidad de métodos de codificación y transmisión de imágenes, vídeos o audio, sobre todo en dispositivos móviles como son los teléfonos, tablets y portátiles, es necesario una elevada capacidad de procesamiento mientras se mantiene el uso de batería a un mínimo. Es en estas situaciones en las que HLS es apto para generar soluciones de buena calidad en un tiempo de desarrollo corto.
- **Seguridad:** Operaciones costosas y complejas como es el hashing o encriptación tienen demasiada carga en procesadores no especializados. HLS hace posible crear algoritmos de seguridad eficientes de forma rápida.
- **Redes:** El control de redes y comunicaciones requiere de tiempos de respuesta muy cortos y el procesado de cantidades de información muy grandes a altas velocidades, es en estos casos en los que las FPGA son necesarias, y HLS es capaz de acelerar la implementación de módulos controladores.

1.2.2.3.4 Restricciones

Aunque HLS se programa en lenguajes de alto nivel, estos tienen diversas restricciones para hacer posible su síntesis[15]. Se definen a continuación las restricciones impuestas en C++, el lenguaje utilizado en este proyecto:

- **Llamadas al sistema:** No se puede depender de funcionalidades ofrecidas por el sistema operativo, como es el acceso a ficheros o al reloj del sistema, el lanzamiento de hilos o la clonación de procesos. Si es necesario hacer alguna de estas tareas, las deberá realizar el sistema que controle el circuito que se está diseñando y le envíe la información necesaria, o que el circuito implemente su propio sistema al que hacer llamadas.
- **Memoria dinámica:** No se puede utilizar memoria dinámica por lo que siempre hay que definir la memoria usada en tiempo de compilación, permitiendo al

sintetizador crear las memorias físicas pertinentes. Esta restricción se extiende al uso de la librería estándar de plantillas de C++, la *STL*, ya que la gran mayoría de funcionalidades y estructuras ofrecidas por esta librería dependen de memoria dinámica, así que en su gran mayoría no es sintetizable y su uso es desaconsejado.

- **Punteros:** Existen diversas limitaciones respecto al manejo de punteros:
 - No se pueden utilizar punteros genéricos (`void*`).
 - Un vector de punteros es sintetizable siempre que estos apunten a escalares o vectores de escalares, pero no a otros punteros.
 - No se permiten punteros a funciones, y por ende no se pueden pasar funciones como parámetros a otras funciones.
- **Recursividad:** Las funciones recursivas no son sintetizables, haciendo más complejo el diseño de árboles, grafos y otras estructuras comunes.
- **Funciones a acelerar:** Solo es posible acelerar una función usando HLS en un diseño, por lo que si es necesario ejecutar múltiples funciones, estas deben ser agregadas bajo una misma función *Top-Level* o padre, o ser separadas en diferentes diseños.

No solo existen restricciones software al programar para FPGA, sino que también es necesario tener en cuenta el diseño RTL que será inferido por las líneas de código que se escriben, ya que si no se tiene en cuenta las distintas limitaciones del chip FPGA objetivo, se obtendrá un diseño demasiado complejo o lento para ser usado en este.

Un claro ejemplo es estas restricciones es la reducida memoria de bloques (BRAM) disponible en las FPGAs, que en muchos casos no es suficientemente grande como para almacenar una sola imagen de alta definición al completo, mientras que en ordenadores personales se dispone de gigabytes de memoria para soportar la lógica deseada.

1.2.2.3.5 Simulación

La simulación es una parte clave del desarrollo para FPGA, y HLS hace este proceso más simple permitiendo al desarrollador comprobar la lógica de su programa con diferentes programas de prueba y datos de entrada de forma rápida e intuitiva, de la misma forma que se comprobaría el funcionamiento de un programa diseñado para su ejecución en CPU.

Una vez la lógica del programa ha sido comprobada, se procede sintetizar el código y simular el código RTL para solucionar problemas específicos de la implementación FPGA. Para esta tarea se usan simuladores como Vivado XSim (el usado en este proyecto), Riviera o Xcelium, entre muchos otras opciones.

Estos simuladores nos permiten comprobar si el diseño RTL funcionaría en un chip FPGA, usando un programa de testeo que maneja la entrada y salida de datos con el módulo RTL siendo comprobado.

1.2.2.4 Plataformas de desarrollo para soluciones FPGA

Actualmente existen dos grandes plataformas de desarrollo para soluciones FPGA: Xilinx VITIS e Intel OneAPI, cuyos logos representativos se pueden ver en la Figura 1.4.

Xilinx VITIS unifica una gran variedad de elementos software para desarrollo de aplicaciones, como son compiladores, depuradores, analizadores, librerías de aceleración, entornos de desarrollo para dominios específicos como puede ser IA, redes, finanzas, seguridad, visión artificial y muchos otros.

VITIS permite a desarrolladores acelerar sus aplicaciones con distintos elementos hardware como son FPGAs, tarjetas de aceleración Alveo, GPUs o TPUs ya sea en la nube o en físico.

Intel OneAPI, como su nombre indica, hace especial hincapié en la unificación de distintas plataformas de procesamiento como son CPUs, GPUs, FPGAs y TPUs entre otras en una sola API o entorno de desarrollo con una gran variedad de librerías y extensiones.

Ambas plataformas facilitan el uso conjunto de núcleos IP, HLS y HDL en proyectos de gran escala, haciendo el uso de FPGAs mucho más fácil y atractivo, y fomentando su uso en una gran variedad de campos y ciencias.

1.2.2.5 Selección de metodología

Cuando se decide usar FPGAs para un nuevo proyecto, no es cuestión de usar exclusivamente HDL o HLS para su desarrollo, sino de seleccionar para qué aplicaciones usar uno y otro.

En secciones anteriores se han enumerado diversos casos de uso en los que HLS es una buena opción a utilizar, y por otra parte existen casos de uso para los cuales



(a) Logo Xilinx Vitis



(b) Intel OneAPI

Figura 1.4: Xilinx Vitis e Intel OneAPI

Origen: *Imágenes obtenidas de las páginas oficiales de Xilinx Vitis e Intel OneAPI.*

HLS no es óptimo, siendo mejor crear el diseño RTL a mano para obtener los mejores resultados posibles.

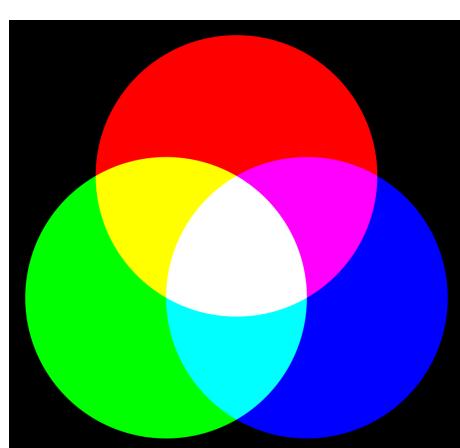
Es generalmente una buena regla seguir los siguiente pasos cuando se deba implementar un nuevo módulo en el sistema:

1. Comprobar si existen núcleos IP que implementen la lógica que deseamos de forma optimizada y precisa.
2. En caso de que no exista un núcleo IP que cumpla los requisitos deseados o sea demasiado caro para el presupuesto del proyecto, estudiar y comprobar la viabilidad de crear esta lógica usando HLS, obteniendo a su vez un prototipo implementado en un lenguaje de alto nivel.
3. En caso de que los resultados usando HLS no sean favorables, se utiliza el prototipo de alto nivel como referencia para crear un diseño RTL a mano, obteniendo un diseño optimizado al caso de uso.

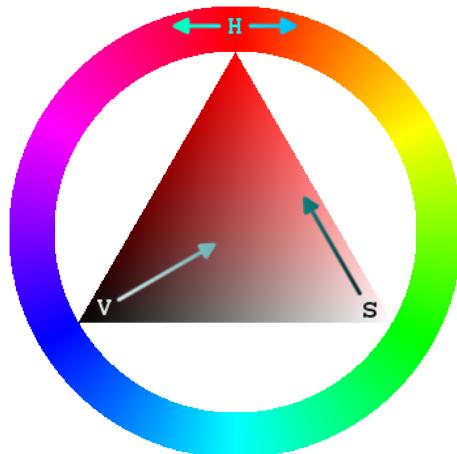
1.2.3 Procesamiento de imágenes

El procesamiento de imágenes es el conjunto de técnicas que se aplican a imágenes digitales para modificar sus propiedades o calidad, o para facilitar la búsqueda y extracción de información.

Una imagen es un conjunto de píxeles organizados en forma de matriz, teniendo un número definido de filas y columnas. Al número de filas y columnas de una imagen se le denomina *resolución*, por ejemplo, una imagen con resolución *Full HD* tendría 1920 columnas y 1080 filas, para un total de 2073600 píxeles.



(a) Modelo aditivo de colores primarios
RGB



(b) La rueda de matices HSV

Figura 1.5: Representación RGB y HSV de un píxel de color.

Origen: Imágenes obtenidas de las páginas principales para RGB y HSV en Wikipedia.

Cada píxel en una imagen puede venir definido de muchas maneras distintas dependiendo de la escala de colores y la representación usada: Una imagen en escala de grises suele tener un sólo valor por píxel que representa el nivel de luminiscencia o luma, mientras que una imagen de color suele tener 3 valores por píxel, cada uno representando el valor de uno de los colores primarios rojo, verde y azul (representación RGB), o representando el matiz, saturación y valor (representación HSV), entre algunas otras posibilidades. Se muestran los principios en los que se basan los modelos RGB y HSV en la Figura 1.5.

Un vídeo es una secuencia de fotogramas reproducidos a una determinada frecuencia denominada *fotogramas por segundo* o FPS. Cada fotograma de un vídeo es una imagen, por lo que si tenemos que procesar un vídeo de color Full HD a 30 FPS, sería necesario procesar más de 185 millones de valores por segundo, y es para el procesado de estas cantidades masivas de datos para lo que se necesitan GPUs y FPGAs.

1.2.3.1 Librerías

Existe una gran variedad de algoritmos y técnicas de procesamiento de imágenes que han sido implementados con distintos niveles de eficiencia en múltiples librerías de procesamiento de gráficos.

Una de las librerías de procesamiento de gráficos más conocidas en el mundo es

OpenCV, que es una librería libre, multiplataforma y con una gran comunidad de desarrolladores y contribuidores que han trabajado durante años en mejorar tanto el código como la documentación de esta.

OpenCV ofrece una gran cantidad de filtros, algoritmos y soluciones IA para una extensa variedad de campos relacionados con la visión artificial, estando disponible para su uso con C++, Python, Java, Matlab, Octave y Javascript.

Principalmente, OpenCV ha sido desarrollado para usar la CPU del computador y la GPU como acelerador, pero también existen adaptaciones que llevan esta librería a FPGA y es fácilmente accesible usando VITIS HLS.

Si bien es cierto que esta librería es de gran calidad, no se usará en este proyecto ya que abstrae al desarrollador de los detalles de implementación y el funcionamiento de los algoritmos, que es lo que se busca explorar como uno de los objetivos principales.

1.2.3.2 Aceleración

Procesar millones de píxeles por segundo es un tarea complicada y que debe optimizarse y acelerarse lo máximo posible para lograr buenos resultados, siendo aún más notable cuando es necesario ejecutar una secuencia ordenada de algoritmos de procesamiento de imágenes en lo que se conoce como un *pipeline* de procesamiento. En esta sección se verán brevemente algunas de las distintas soluciones de aceleración hardware para procesado de imágenes o visión máquina usadas en la actualidad, yendo de menor a mayor complejidad.

1.2.3.2.1 CPU independiente

Una CPU por sí sola tiene diversos métodos de aceleración de código, algunos pasivos como puede ser el pipelining a nivel de instrucción mostrado en la Figura 1.6[16] o el uso de cachés, que son manejados automáticamente por el chip y que el desarrollador de software debe de conocer para que su código los aproveche al máximo, por ejemplo, sabiendo el funcionamiento de una caché de un computador se puede deducir que un programa que accede a píxeles almacenados secuencialmente en memoria es mucho más rápido que uno que hace accesos aleatorios a píxeles dispersos, ya que se aprovecha el principio de localidad de datos correctamente.

Por otro lado existen una serie de métodos de aceleración que deben ser usados de forma activa por el desarrollador, siendo dos de estos métodos los siguientes:



Figura 1.6: Aceleración obtenida con pipelining de instrucciones en un procesador CPU RISC simple. Se observa como una instrucción se divide en etapas de forma que múltiples instrucciones estén en ejecución al mismo tiempo.

Origen: *Imagen editada de los recursos didácticos de Intel (<https://techdecoded.intel.io/resources/understanding-the-instruction-pipeline/>)*

- **Multithreading:** El procesamiento de imágenes y vídeo puede ser paralelizado usando hilos o procesos de múltiples formas, siendo algunas de ellas:

- Uso de un modelo **maestro-esclavo** en el que un maestro consuma fotogramas de entrada y ordene a un esclavo su procesamiento. Eventualmente, cuando el esclavo termine su tarea asignada, el maestro retomará los datos procesados y los devolverá o guardará según sea necesario. Este método suele ser implementado usando *thread pools* para una mayor eficiencia.
- Uso de un modelo de **pipeline heterogéneo**, de forma que haya uno o varios hilos dedicados a una sola tarea y que cuando un hilo termine de procesar un paso de procesamiento releve el fotograma a un hilo encargado de procesar el siguiente paso del pipeline. Este método viene con el inconveniente de necesitar balancear el número de hilos asignados a cada paso y la comunicación entre ellos.
- Uso de modelos híbridos o especializados.

- **Operaciones vectoriales:** Cuando se operan datos en masa, como puede ser para procesamiento de imágenes, criptografía, o redes neuronales, es conveniente considerar el uso de operaciones vectoriales como puede ser AVX[17], que dispone de registros de 512 bits, en los que es posible guardar tantos valores como sea posible (por ejemplo, 32 valores o píxeles de 16 bits) y aplicar la misma operación a todos a la vez, potencialmente acelerando la ejecución en grandes magnitudes.

Usando estos métodos es posible obtener resultados en tiempos aceptables para procesamiento de imágenes, pero el uso de la CPU será muy elevado y con ello la temperatura y consumo eléctrico.[18]

1.2.3.2.2 CPU y GPU

Cuando se dispone de una GPU auxiliar es posible descargar las secciones de código de cálculo intenso a la GPU mientras la CPU se encarga del control de flujo del programa, o incluso encontrar un equilibrio de cálculos entre ambas para aprovechar el potencial de ambos chips al máximo.

Una GPU es ideal para procesamiento de imágenes debido a su arquitectura que lleva los dos aspectos de aceleración previamente mencionados en la CPU a una escala mucho mayor: Se basa en un conjunto de bloques mínimos denominados *Streaming Multiprocessors* (SM) o *Compute Units* (CU), cada uno con un alto número de núcleos muy simples (procesadores escalares) y un planificador, entre algunos otros componentes de procesado de gráficos. A cada SM se le asigna un conjunto de hilos denominado *warp*¹, dando cada uno de estos hilos exactamente el mismo código a ejecutar. El planificador de la SM asigna un hilo a cada núcleo de forma que todos ejecuten la misma instrucción al mismo tiempo en datos distintos, efectivamente logrando un funcionamiento SIMD (Single Instruction, Multiple Data), es decir, se logra un efecto muy similar a las operaciones vectoriales de una CPU pero a mayor escala y flexibilidad.[19][20]

Estos bloques SM se replican un gran número de veces dentro del chip, permitiendo a una GPU procesar una gran cantidad de warps en paralelo, y con ello, permitiendo el procesado de una enorme cantidad de operaciones simples manteniendo a su vez un consumo eléctrico y temperatura moderada.[18]

1.2.3.2.3 FPGA independiente

Una FPGA, al igual que una CPU, puede operar de forma completamente independiente, implementando tanto la lógica de control de flujo como el cálculo masivo de píxeles.

Una FPGA es inherentemente paralela, es decir, de la misma forma que en un circuito electrónico la electricidad fluye en paralelo por distintos cables, en una FPGA se ejecutan múltiples bloques lógicos al mismo tiempo. Esta poderosa propiedad permite lo siguiente:

- **Pipeline de instrucciones:** Nótese la diferencia con los pipelines *a nivel de instrucción* en CPUs, ya que en vez de crear un pipeline con las distintas etapas de una instrucción, es posible crear uno que use instrucciones al completo, por

¹Término usado en CUDA, otras plataformas o APIs de programación de GPUs pueden diferir en la terminología usada y la planificación de ejecución de código en SMs.

lo que sería posible que múltiples o potencialmente todas las instrucciones en una función o bucle se estén ejecutando en paralelo para distintos datos (en este caso píxeles) siempre que no existan dependencias entre estos.[21]

- **Pipeline de funciones:** De una forma similar al pipeline de instrucciones, es posible ejecutar múltiples funciones en paralelo para distintos datos, a esto se le conoce como *Dataflow* y permite que las distintas etapas de un pipeline de procesamiento de imágenes estén siempre siendo procesadas en paralelo.[21]
- **Acceso a memoria masivo:** Las memorias en computadores corrientes suelen tener un número limitado de puertos de acceso, pudiendo ser un cuello de botella[22]. Las BRAM en una FPGA tienen el mismo problema, pero a diferencia de un computador normal, un diseñador puede crear más bloques de RAM que trabajen en paralelo, aumentando así el número máximo de accesos que se pueden hacer a memoria en un sólo ciclo de reloj.[21]
- **Desenrollado de bucles:** Es una técnica muy común para programas en CPU reducir el número de iteraciones de un bucle para ahorrar comparaciones y saltos. Una FPGA también utiliza esta técnica pero de una forma mucho más efectiva ya que es capaz de ejecutar múltiples iteraciones de un bucle en un mismo ciclo, potencialmente todas siempre que exista suficiente hardware en el chip para implementar la lógica necesaria.[21]

Existe una gran cantidad de otras técnicas que aprovechan la naturaleza paralela de una FPGA, siendo este uno de los pilares a tener en cuenta cuando se desarrolla para esta tecnología.

1.2.3.2.4 CPU y FPGA

Una FPGA tiene un número limitado de componentes hardware a utilizar por el desarrollador. Es por este motivo por lo que es conveniente implementar en ellas sólo la lógica crítica que requiera de mayor aceleración y eficiencia, mientras que se implementa el resto de la lógica en un computador con CPU que, potencialmente, es el maestro de la FPGA.

Un caso de uso común es cuando se desea utilizar un sistema operativo junto con la FPGA: Si bien es cierto que parte de una FPGA puede configurarse exactamente como una CPU, esto consume una gran cantidad de recursos que podrían ser dedicados a otra sección de código de mayor importancia. Es en estos casos en los que se utiliza un modelo dual CPU-FPGA.

Este caso de uso es tan común y poderoso que diversas empresas han creado placas que incorporan una CPU y una FPGA en un mismo chip, ambos conectados por

buses de alta velocidad. La gama de estos productos comercializados por Xilinx se conoce por *Zynq*.

1.2.3.2.5 CPU, GPU y FPGA

Como extensión del previo modelo de aceleración, es posible agregar una GPU al conjunto de forma que cada chip pueda dedicarse a las tareas óptimas para las que fueron diseñados.

Este modelo unificado de distintas plataformas de procesamiento está siendo promovido por Xilinx e Intel con sus plataformas de desarrollo, que facilitan la comunicación entre chips gracias a las distintas librerías e interfaces ofrecidas a los desarrolladores.

1.2.4 Codificación C++ moderna

A medida que C++ ha evolucionado como lenguaje ha aparecido una gran separación entre lo que se conoce como *C++ clásico* y *C++ moderno*, siendo las diferencias de estilo y patrones entre estos dos tan grande que incluso podrían ser considerados lenguajes diferentes.[23]

C++ moderno ha sido acuñado empezando por la versión C++11 y ha ido evolucionando con las subsecuentes versiones C++14 o C++17 hasta la actualidad.

Programar usando C++ moderno implica seguir una filosofía de codificación y, sobre todo, un uso abundante y efectivo de la librería estándar de plantillas (STL), que ofrece una miríada de funcionalidades y estructuras de datos que, en su mayor parte, eliminan la necesidad de que el desarrollador maneje su propia memoria dinámica y por ende se reduzca el riesgo de fugas de memoria a un mínimo. A su vez, C++ moderno facilita el manejo de conceptos previamente complejos de abordar como es la concurrencia, manejo de errores y uso de ficheros, entre otros.[24]

Por otro lado, C++ clásico se puede definir de forma informal como ‘Programación C pero con clases’, esto se debe a que cuando se lanzó C++ en la década de los 80 las funcionalidades ofrecidas por la librería estándar eran muy reducidas y una gran cantidad de los desarrolladores de C migraron a este nuevo lenguaje usándolo simplemente como una extensión del lenguaje con el que eran familiares.[25]

En la actualidad conviven ambos paradigmas, aplicados en distintos campos:

- **C++ moderno** es utilizado en aplicaciones principalmente orientadas a su

ejecución en sistemas modernos. Permite una codificación rápida, fácil de mantener y altamente legible.

- **C++ clásico** es utilizado en aplicaciones empotradas que requieren niveles de optimización muy elevados o no tienen soporte para las nuevas funcionalidades ofrecidas por C++11 y posteriores. La programación para FPGA HLS en C++ tiene un estilo clásico, ya que la gran mayoría de funcionalidades ofrecidas por la STL no son soportadas por limitaciones del paradigma y arquitectura.[15]

1.3 Objetivos y alcance del proyecto

Con este Trabajo de Fin de Grado (TFG) se busca implementar usando C++ moderno una librería de detección de movimiento en vídeo en tiempo real para CPU con pruebas de unidad que verifiquen su correcto funcionamiento y documentar paso a paso el proceso llevado a cabo para adaptarla a FPGA usando HLS. Se evita en todas las fases del desarrollo el uso de librerías de terceros, siendo la tarea principal del proyecto la correcta implementación y modificación de los algoritmos para permitir un estudio en profundidad de las diferencias fundamentales que existen entre programas diseñados para CPU y aquellos diseñados para FPGA.

Se pueden enumerar los siguientes objetivos principales:

- **Objetivo 1:** Codificación en C++ de una librería software para detección de movimiento en vídeo Full HD en tiempo real. El código será multihilo y deberá capturar las tendencias y patrones modernos de codificación, usando la librería estándar (STL) siempre que sea posible y buscando una buena calidad de código. No se dependerá de librerías externas para facilitar la implementación de los algoritmos necesarios, siendo necesario implementarlos siguiendo la especificación de estos.
- **Objetivo 2:** Documentación detallada del proceso de adaptación del código C++ a un diseño RTL optimizado mediante el uso del paradigma HLS. Se analizan los cambios realizados a la lógica y estructura de los algoritmos desarrollados para lograr pasar de un programa diseñado para su ejecución en CPU a uno especializado en su síntesis a un circuito digital.
- **Objetivo 3:** Simulación del diseño RTL para determinados chips FPGA que validen la corrección del diseño.
- **Objetivo 4:** Comparativa de rendimiento y uso de recursos hardware, y análisis de ventajas y desventajas del uso de ambas tecnologías.

1.4 Glosario de términos

- **FPGA:** Field Programmable Gate Array. Circuito integrado basado en bloques lógicos configurables (CLB), conectados entre si con interconexiones programables.
- **ASIC:** Application Specific Integrated Circuit. Circuito integrado no reconfigurable con una aplicación específica que realiza con gran eficacia.
- **HLS:** High Level Synthesis. Proceso automatizado de diseño que interpreta una descripción algorítmica y crea un diseño RTL con el comportamiento descrito.
- **HDL:** Hardware Description Language. Lenguaje de programación especializado en la descripción del comportamiento y estructura de circuitos electrónicos con un nivel de abstracción RTL.
- **VHDL:** VHSIC Hardware Description Language. HDL estandarizado por la IEEE (Institute of Electrical and Electronics Engineers) para el diseño de sistemas digitales con distintos niveles de abstracción.
- **Verilog:** HDL estandarizado por la IEEE para el diseño y verificación de sistemas digitales con un nivel RTL de abstracción.
- **RTL:** Register-transfer level. Nivel de abstracción de diseño digital superior al diseño estructural e inferior al diseño algorítmico (HLS) basado en el control de señales digitales entre registros hardware y las operaciones lógicas aplicadas a estas.
- **Módulo IP:** Módulo de propiedad intelectual, potencialmente comercializado, que contiene un diseño digital con una funcionalidad determinada.

1.5 Estructura del documento

En esta primera parte se presenta el proyecto con los motivos que han impulsado su desarrollo junto con una visión del contexto actual de la tecnología HLS en FPGA, el procesamiento de imágenes, métodos de aceleración hardware y las filosofías de codificación C++ actuales. Además, se enumeran los objetivos del proyecto y términos comúnmente utilizados en el campo del diseño digital para FPGA. Para finalizar, se expone la planificación y presupuesto del proyecto.

En la segunda parte se aborda el procedimiento de análisis y desarrollo de la librería software para CPU codificada en C++ y su progresiva adaptación a FPGA usando

el paradigma HLS.

En la tercera parte se documentan las conclusiones obtenidas, el conocimiento obtenido durante la creación del proyecto y futuras tareas y mejoras.

Por último, en la cuarta parte se puede encontrar el código fuente de la librería en C++ desarrollada, tanto para CPU como FPGA.

Capítulo 2

Planificación

2.1 Metodología de desarrollo

El proyecto ha sido planificado siguiendo la metodología SCRUM, que nos permite segmentar las numerosas tareas a realizar en segmentos denominados iteraciones o *sprints*. Cada iteración del proyecto tiene una temática y un marco de tiempo asignado para su finalización. Al principio de una iteración se planea la distribución de tareas a lo largo del tiempo asignado y al final se revisa el trabajo realizado y se valida.

2.2 Planificación del proyecto

El proyecto se ha desarrollado en las siguientes iteraciones:

1. **Estudio de tecnologías y algoritmos:** Se investiga el campo de procesamiento de imágenes y algoritmos de utilidad. Además, se investiga la tecnología HLS para FPGAs, sus limitaciones y posibles alternativas de aceleración hardware.
2. **Prototipado del pipeline de procesamiento de vídeo:** Se implementan diversos pipelines posibles para detección de movimiento y se selecciona el que de mejores resultados, teniendo en cuenta tanto la eficiencia como la calidad de resultados.
3. **Creación de la librería de detección de movimiento en C++:** Se codifica en C++ moderno una librería de detección de movimiento en tiempo real en vídeo, con pruebas de unidad que la validen. Se optimiza para que se

use todos los núcleos disponibles en la CPU y se eviten cálculos complejos o redundantes.

4. **Adaptación de la librería a FPGA HLS:** Se estudian los cambios a realizar y el orden a seguir. Progresivamente se sustituyen los algoritmos originales por versiones adaptadas a hardware FPGA, teniendo en cuenta las diferentes restricciones impuestas. Se optimiza el código aplicando directivas únicas de HLS.
5. **Documentación del proyecto:** Se recopila los resultados obtenidos, código y conocimiento adquirido en un documento final y se revisa por los coordinadores.

En cada iteración se han realizado las siguientes tareas:

1. Estudio de tecnologías y algoritmos:
 - **C++ moderno:** Estudio de patrones y tendencias actuales en la industria para programación en C++.
 - **Procesamiento de imágenes:** Estudio de algoritmos y pipelines de procesamiento de imágenes y vídeo.
 - **FPGA HLS:** Estudio de la tecnología HLS para FPGA, realización de talleres de aprendizaje ofrecidos por Xilinx y estudio de su contexto histórico y actual.
2. Prototipado de pipeline de procesamiento de vídeo:
 - **Algoritmos:** Codificación de diversos algoritmos de procesamiento de imágenes en Python.
 - **Pipeline:** Encadenamiento de algoritmos en pipelines, observando los resultados obtenidos, eficiencia y su potencial paralelismo.
 - **Métricas:** Compilación de métricas y resultados de los algoritmos usados.
3. Creación de la librería de detección de movimiento en C++:
 - **Codificación:** Codificación en C++ del pipeline seleccionado usando técnicas multihilo.
 - **Tests:** Creación de pruebas de unidad y programas de prueba de la librería desarrollada.
 - **Métricas:** Compilación de métricas del pipeline implementado.
4. Adaptación de la librería a FPGA HLS:

- **Código base:** Rediseño de la librería original para adaptarse a las limitaciones impuestas por HLS.
- **Código optimizado:** Optimización del código HLS obtenido para reducir la latencia del pipeline y el uso de recursos hardware.
- **Simulación RTL:** Simular el diseño RTL obtenido para un chip FPGA, verificando su funcionamiento.
- **Métricas:** Compilación de métricas del diseño obtenido.

5. Documentación del proyecto:

- **Documento TFG:** Creación del documento final del TFG, donde se recopila el trabajo e investigación realizada.
- **Revisión:** Revisión del documento y mejoras.

Para la planificación de las iteraciones se ha utilizado software de creación de diagramas Gantt y gestión de proyectos. El diagrama Gantt del proyecto se puede ver en la Figura 2.1.

2.3 Organización

- **Autor - Borrero Foncubierta, Antonio:** Autor del Trabajo de Fin de Grado.
- **Directora - Cifredo Chacón, María Ángeles:** Coordinación del proyecto y aportación de ideas y conocimiento de la tecnología FPGA HLS.
- **Codirectora - Rodríguez García, Mercedes:** Aportación de ideas y revisión del proyecto.

2.4 Costes

Estudio de los costes de los recursos usados para el desarrollo del proyecto, teniendo en cuenta tanto los recursos humanos usados como los materiales.

2.4.1 Humanos

Se han calculado los costes humanos teniendo en cuenta jornadas de trabajo de 4 horas, de lunes a viernes. Se estima el salario de un desarrollador full-stack a €20

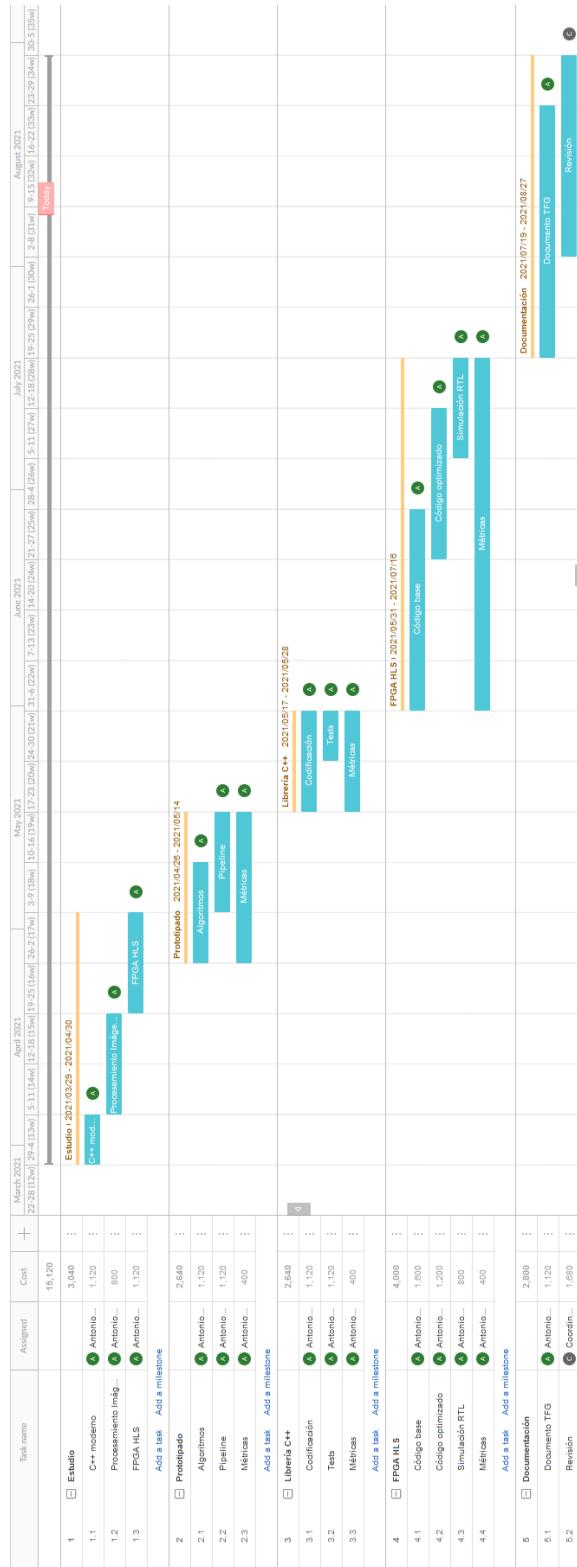


Figura 2.1: Diagrama de Gantt que comprende la planificación temporal del proyecto.

Categoría	Coste por hora (€)	Horas totales	Coste total (€)
Desarrollador	20	420	8.400
Gestor de proyectos Pi	30	80	2.400

Tabla 2.1: Coste de recursos humanos.

Material	Precio unitario (€)	Unidades	Precio Total (€)
Ordenador de desarrollo	1.500	1	1.500
Monitor Full HD	120	1	120
Cable HDMI	6	1	6
Periféricos	30	1	30
Escritorio	145	1	145
Silla oficina	80	1	80
Raspberry Pi 4B 4GB	100	1	100
Camara Raspberry Pi	9	1	9
Webcam Full HD	20	1	20

Tabla 2.2: Coste de recursos materiales.

brutos por hora, y se estima el salario de un gestor de proyectos a €30 brutos por hora, en base a sueldos medios en España encontrados en [glassdoor.es](https://www.glassdoor.es) en 2021.

El proyecto se ha realizado a lo largo de 22 semanas y el coste total humano es de **€10.800**, como se muestra en la Tabla 2.1.

2.4.2 Materiales

Se listan los recursos físicos usados para el desarrollo del proyecto en la Tabla 2.2. El precio del hardware ha sido ajustado para permitir un desarrollo rápido que minimice el tiempo muerto al compilar y sintetizar código HLS.

2.4.3 Total

La suma total de costes del proyecto se puede encontrar en el Tabla 2.3

Recursos	Total (€)
Humanos	10.800
Materiales	2.010
Totales	12.810

Tabla 2.3: Coste total del proyecto.

2.5 Riesgos

Durante el desarrollo del proyecto es posible que surjan problemas o complicaciones. En esta sección se listan los riesgos que se han tenido en cuenta durante el desarrollo del proyecto, las medidas preventivas que se han tomado, el impacto que tendrían si se manifestasen y su solución.

- **Falta de tiempo de desarrollo.**
 - **Descripción:** El desarrollo del proyecto lleva más tiempo de lo esperado, y no se puede completar para la fecha de presentación de este.
 - **Probabilidad:** Media. Se trabaja con tecnologías y conceptos nuevos para el equipo de desarrollo.
 - **Impacto:** Alto. El proyecto no estaría listo para ser presentado.
 - **Prevención:** Seguir la metodología SCRUM, planificando y revisando cada una de las iteraciones, y adaptando las iteraciones siguientes de forma acorde.
 - **Solución:** Posponer la presentación del proyecto a la siguiente convocatoria disponible.
- **No se consigue detectar movimiento en vídeo.**
 - **Descripción:** No se logra crear una librería que detecte movimiento en vídeo o su rendimiento es muy desfavorable.
 - **Probabilidad:** Baja. La detección de movimiento es un campo extensamente investigado.
 - **Impacto:** Alto. No se podría portar a HLS la librería.
 - **Prevención:** Estudio de algoritmos y pipelines para detección de movimiento y uso de referencias.
 - **Solución:** Seleccionar una tarea de visión máquina distinta a portar a FPGA HLS.

- **No se consigue portar el algoritmo a HLS.**
 - **Descripción:** La librería creada no se puede portar a HLS debido a sus limitaciones o falta de experiencia por el equipo de desarrollo.
 - **Probabilidad:** Baja. HLS es especialmente apto para tareas de visión artificial.
 - **Impacto:** Alto. No se podría finalizar el proyecto.
 - **Prevención:** Estudio de HLS y realización de talleres para aprender formas de circunventar sus limitaciones.
 - **Solución:** Seleccionar una tarea de visión máquina distinta a portar a FPGA HLS.

Parte II

Desarrollo

Capítulo 3

Planteamiento y Análisis

En este capítulo se profundizará en el planteamiento del desarrollo del proyecto y el análisis realizado.

3.1 Descripción de la solución

La solución empleada para lograr detección de movimiento en vídeo busca la simplicidad y corrección de código, mostrando a su vez la filosofía y los distintos patrones presentes en C++ moderno, dejando en segundo plano la obtención de máxima eficiencia y el uso de algoritmos y técnicas complejas que puedan dificultar el entendimiento de la librería desarrollada.

La entrada de vídeo se captura por una cámara Full HD con conexión USB y es controlada por un programa piloto. Este programa piloto hará llamadas a la librería desarrollada con los fotogramas capturados y recibirá como resultado los distintos contornos de movimiento detectados en ellos.

Se desarrollan 2 programas pilotos diferentes, uno que comience a grabar a disco el vídeo cuando se detecte movimiento y otro que muestre la entrada de vídeo en todo momento y se muestre en pantalla los contornos de movimiento detectados.

3.2 Actores

Sólo existe un tipo de actor en el sistema, los **programas piloto**. Al tratarse de una librería con una API únicamente interactuable mediante código, se consideran las aplicaciones que interactúen con la librería como los actores.

3.3 Requisitos

Se enumeran los requisitos que debe cumplir la librería creada para ser considerada como completa. Se enumerarán los requisitos de la librería codificada para CPU y los requisitos del núcleo IP para FPGA por separado, ya que sus características de funcionamiento difieren en gran medida dadas las diferencias de arquitectura.

Se listarán los requisitos funcionales usando la nomenclatura RF[N], siendo [N] un número que identifica de forma única al requisito funcional, comprendido en 0 y el número total de requisitos funcionales. De igual forma, se listarán los requisitos no funcionales usando la nomenclatura RNF[N].

3.3.1 Funcionales

Los requisitos funcionales describen una determinada funcionalidad del sistema que se describe. A continuación se enumeran los requisitos funcionales de la librería para CPU:

- **RF0 - Crear detector de movimiento:** El programa piloto crea un nuevo detector de movimiento con una resolución determinada que procesará imágenes de forma asíncrona.
- **RF1 - Destruir detector de movimiento:** El programa piloto destruye un detector de movimiento, terminando el procesado de cualquier imagen en curso.
- **RF2 - Añadir fotograma:** El programa piloto añade un fotograma en escala de grises de 16 bits a un detector de movimiento para ser procesado, si la cola de procesado está llena, se bloquea el piloto hasta que se añada el fotograma o se lanza una excepción según el modo de bloqueo seleccionado.
- **RF3 - Obtener movimiento:** El programa piloto solicita los resultados del fotograma más antiguo añadido a un detector de movimiento, recibiendo los contornos de movimiento detectados si ya han sido procesados, en caso contrario se bloquea el piloto hasta que se procese el fotograma o se lanza una excepción según el modo de bloqueo seleccionado.

Por otro lado, los requisitos funcionales de la librería para FPGA son los siguientes:

- **RF0 - Procesar fotograma:** El programa piloto envía un fotograma en escala de grises de 16 bits con resolución Full HD y recibe los resultados del movimiento detectado.

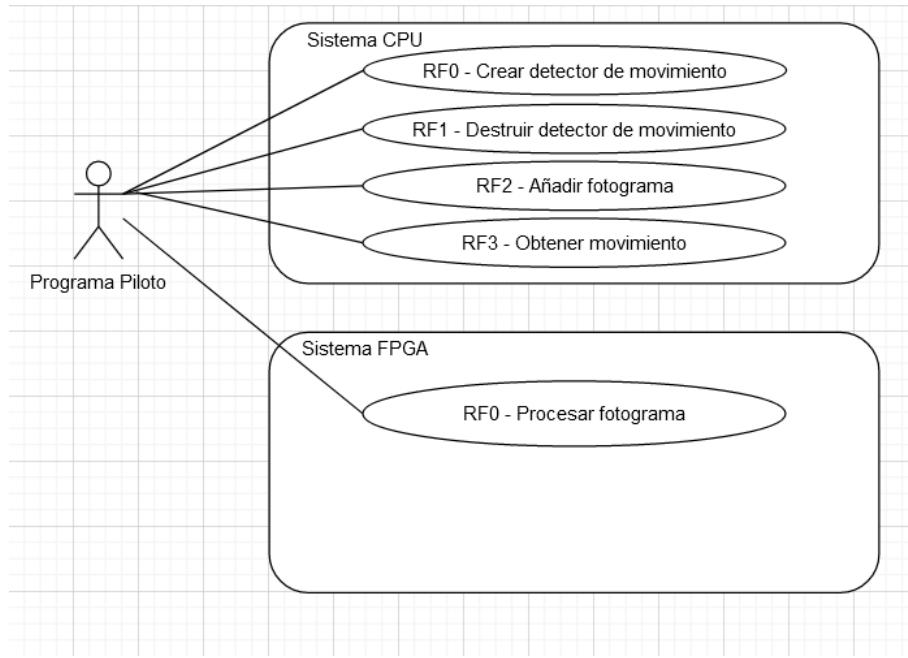


Figura 3.1: Requisitos funcionales del sistema.

Como se puede observar, la librería para CPU busca un funcionamiento asíncrono mientras que la librería para FPGA requiere de un diseño síncrono y bloqueante, por lo que los requisitos funcionales difieren considerablemente. Los distintos requisitos funcionales se pueden encontrar en la Figura 3.1.

3.3.1.1 Descripción de requisitos funcionales CPU

En esta sección se describe en profundidad cada uno de los requisitos funcionales en detalle. Cada requisito funcional describe un escenario principal con una secuencia numerada de interacciones entre el sistema y el actor, en el que cada interacción puede resultar en un escenario alternativo si se cumplen determinadas condiciones.

Dado un paso de un escenario principal descrito con un numeral simple (1, 2, 3, etc...), este podría bifurcar en uno o más escenarios alternativos, cada uno de estos identificados únicamente usando una letra, y esta a su vez es acompañada de un numeral que especifica el paso dentro del escenario alternativo (por ejemplo, 2.a1, 2.b1, 3.a1, etc...). A su vez, un escenario alternativo puede bifurcar recursivamente en otros escenarios alternativos, acompañándose de otra pareja letra-número (2.a1.a1, 2.b1.a1, etc...).

3.3.1.1.1 RF0 - Crear detector de movimiento

- **Descripción:** Se crea una entidad ‘detector de movimiento’ con una resolución determinada, un número de hilos de procesamiento a usar, el tamaño de cola de imágenes, el factor de *downsampling* y el ratio de interpolación de la referencia.
- **Actores:** Programa piloto.
- **Precondiciones:** Sin precondiciones.
- **Postcondiciones:** Se crea una entidad detectora de movimiento manejable por el programa piloto.
- **Escenario:**
 1. El programa piloto llama al constructor de detectores de movimiento.
 2. El sistema comprueba los datos enviados.
 3. El sistema crea un detector de movimiento con las características definidas.
- **Escenarios alternativos:**
 - 2.a1 El sistema detecta que los datos enviados son inválidos.
 - 2.a2 El sistema lanza una excepción.

3.3.1.1.2 RF1 - Destruir detector de movimiento

- **Descripción:** Se destruye una entidad ‘detector de movimiento’ que ha sido creada usando RF0. Cualquier procesado que esté realizando en el momento de destrucción será detenido y todos las imágenes en la cola serán destruidas.
- **Actores:** Programa piloto.
- **Precondiciones:** Existe un detector de movimiento.
- **Postcondiciones:** El detector de movimiento es destruido y por ende no puede seguir siendo usado.
- **Escenario:**
 1. El programa piloto llama al destructor de detectores de movimiento.
 2. El sistema detiene el detector de movimiento e inmediatamente lo destruye.

3.3.1.1.3 RF2 - Añadir fotograma

- **Descripción:** El programa piloto envía una imagen en escala de grises de 16 bits a un detector de movimiento. Si la cola de imágenes está llena y se ha seleccionado el modo bloqueante, el programa piloto se detendrá hasta que se complete el procesamiento de la imagen más antigua añadida, en caso de que no se haya seleccionado el modo bloqueante, se lanzará una excepción.
- **Actores:** Programa piloto.
- **Precondiciones:** Existe un detector de movimiento y la imagen a añadir ha sido correctamente inicializada como una imagen de 16 bits con una resolución igual a la del detector de movimiento.
- **Postcondiciones:** La imagen es eventualmente añadida a la cola de imágenes o se lanza una excepción.
- **Escenario:**
 1. El programa piloto solicita añadir una imagen al detector de movimiento en modo bloqueante.
 2. El sistema comprueba si la cola de imágenes está llena y determina que no lo está.
 3. El sistema añade la imagen a la cola.
- **Escenarios alternativos:**
 - 1.a1 El programa piloto solicita añadir una imagen al detector de movimiento en modo no bloqueante.
 - 1.a2 El sistema comprueba si la cola de imágenes está llena y determina que no lo está.

Vuelta a paso 3 del escenario principal.
 - 1.a2.a1 El sistema comprueba si la cola de imágenes está llena y determina que lo está.
 - 1.a2.a2 El sistema lanza una excepción.
- 2.a1 El sistema comprueba si la cola de imágenes está llena y determina que lo está.

2.a2 El sistema se bloquea hasta que se procesa la imagen mas antigua añadida.

Vuelta al paso 3 del escenario principal.

3.3.1.1.4 RF3 - Obtener movimiento

- **Descripción:** El programa piloto solicita los datos de movimiento detectados en la imagen más antigua añadida. Si la imagen más antigua no ha sido procesada aún y se ha seleccionado el modo bloqueante, el programa piloto se detendrá hasta que se complete el procesamiento de esta, en caso de que no se haya seleccionado el modo bloqueante, se lanzará una excepción.
 - **Actores:** Programa piloto.
 - **Precondiciones:** Existe un detector de movimiento.
 - **Postcondiciones:** Eventualmente se obtienen los resultados de movimiento de la imagen más antigua y esta se elimina del detector de movimiento, o se lanza una excepción.
 - **Escenario:**
 1. El programa piloto solicita datos de movimiento en modo bloqueante.
 2. El sistema comprueba si la imagen más antigua ha sido procesada y determina que lo está.
 3. El sistema informa los resultados y la imagen usada al programa piloto.
 - **Escenarios alternativos:**
 - 1.a1 El programa piloto solicita datos de movimiento en modo no bloqueante.
 - 1.a2 El sistema comprueba si la imagen más antigua ha sido procesada y determina que lo está.

Vuelta a paso 3 del escenario principal.
- 1.a2.a1 El sistema comprueba si la imagen más antigua ha sido procesada y determina que no lo está.
- 1.a2.a2 El sistema lanza una excepción.

- 2.a1 El sistema comprueba si la imagen más antigua ha sido procesada y determina que no lo está.
- 2.a2 El sistema se bloquea hasta que sea procesada.
Vuelta al paso 3 del escenario principal.

3.3.1.2 Descripción de requisitos funcionales FPGA

3.3.1.2.1 RF0 - Procesar fotograma

- **Descripción:** Se envía una imagen Full HD en escala de grises de 16 bits y se detecta movimiento en ella teniendo en cuenta previas imágenes enviadas.
- **Actores:** Programa piloto.
- **Precondiciones:** La imagen enviada está representada como un stream de píxeles de 16 bits en escala de grises con resolución Full HD.
- **Postcondiciones:** Se informa del movimiento detectado y la referencia es actualizada,
- **Escenario:**
 1. El programa piloto solicita detectar movimiento en una imagen.
 2. El sistema detecta movimiento en la imagen, actualiza la referencia y devuelve el resultado detectado.

3.3.2 No funcionales

Los requisitos no funcionales son aquellos requisitos abstractos que dan calidad al proyecto:

3.3.2.1 RNF0 - Rendimiento

Se debe alcanzar tanto con la versión de CPU como FPGA el procesado de vídeo Full HD para detección de movimiento en tiempo real como mínimo a 30 fotogramas por segundo, para alcanzar estas métricas es necesario que el código use algoritmos apropiados y sea suficientemente optimizado.

3.3.2.2 RNF1 - Uso de recursos

Se busca reducir el uso de recursos hardware utilizados para lograr el objetivo del proyecto:

- En CPU se busca lograr la detección de movimiento en tiempo real usando una CPU utilitaria, de forma que el código pueda funcionar correctamente en una gran variedad de dispositivos.
- En FPGA se busca que el uso de recursos hardware usados por el diseño resultantes sea bajo de forma que se pueda exportar como núcleo IP a una gran variedad de chips.

3.3.2.3 RNF2 - Robustez

Es necesario que la librería, tanto para CPU como para FPGA, esté libre de errores internos no controlados que puedan afectar a los programas piloto que la utilicen.

Una librería defectuosa potencialmente puede resultar en su desuso y abandono, por lo que es importante controlar los posibles errores de procesamiento que puedan surgir.

3.3.2.4 RNF3 - Fiabilidad

Se espera que se detecte movimiento de forma consistente y fiable en la secuencia de fotogramas enviados a la librería.

Se maximiza, dentro de la posibilidad de los simples algoritmos utilizados, la fiabilidad tanto de clasificación movimiento/no movimiento como de localización del movimiento en caso de que lo haya.

3.3.2.5 RNF4 - Usabilidad

La librería debe presentar sus funcionalidades a través de una API documentada y de fácil uso para desarrolladores.

Se minimiza el número de funciones y opciones de forma que su integración sea fácil e intuitiva.

3.3.2.6 RNF5 - Mantenibilidad

Se comenta el código de la librería para CPU y se genera documentación Doxygen de esta. Se busca un código limpio y que siga los patrones y filosofía de C++ moderno para facilitar su entendimiento por otros desarrolladores.

En el código para FPGA se seguirán los patrones de diseño indicados en la documentación de HLS por Xilinx, y se usarán las directivas ofrecidas por el paradigma de forma apropiada para un fácil entendimiento.

3.3.2.7 RNF6 - Portabilidad

Se busca crear una librería para CPU compatible con una gran gama de plataformas y sistemas operativos. A su vez, se busca crear un núcleo IP para FPGA apto para una gran gama de chips.

Capítulo 4

Diseño del sistema

En este capítulo se definen los componentes que forman la arquitectura física y lógica del sistema, así como los conceptos clave que la soportan.

4.1 Arquitectura física

Se describen los elementos físicos usados o simulados para el desarrollo del proyecto y sus características.

4.1.1 Raspberry Pi 4B

Raspberry Pi es una serie de ordenadores de una sola placa de bajo coste. Estos ordenadores se comenzaron a desarrollar en el Reino Unido por la Raspberry Pi Foundation y han ido evolucionando desde 2012 hasta la actualidad gracias a su enorme éxito y adopción en una gran cantidad de proyectos como plataformas de prototipado, testeo e incluso producción, siendo usados en clústeres de bajo coste o dispositivos de borde.[26]

En este proyecto se ha utilizado el modelo 4B de 8 gigabytes de RAM para pruebas de la librería destinada CPU con el objetivo de compararlo con los resultados obtenidos en ordenadores de sobremesa utilitarios, de forma que las métricas obtenidas sean fácilmente comparables y replicables.

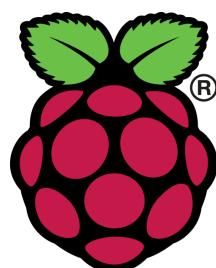
Las especificaciones hardware de la placa se pueden encontrar en la Tabla 4.1 y el dispositivo utilizado se puede ver en la Figura 4.1.

Tipo	Detalles
Procesamiento	1x Quad core 64-bit ARM-Cortex A72 1.5GHz 1x H.265 (HEVC) decodificador hardware (hasta 4Kp60) 1x VideoCore VI 3D Graphics
Memoria	1x 8 Gigabyte LPDDR4 RAM
Energía	5V 3A DC vía conector USB-C
USB y red	1x Módulo inalámbrico 2.4 GHz IEEE 802.11ac 1x Módulo inalámbrico 5.0 GHz IEEE 802.11ac 1x Gigabit Ethernet PoE 2x Puerto USB 3.0 2x Puerto USB 2.0
Audio y vídeo	1x Puerto display 2-lane MIPI DSI 1x Puerto cámara 2-lane MIPI CSI 1x Puerto compuesto 4-pole audio y vídeo 2x Puerto micro HDMI
E/S de uso genérico (GPIO)	Hasta 6x UART Hasta 6x I2C Hasta 6x SPI 1x Interfaz SDIO 1x DPI 1x PCM Hasta 2x Canal PWM Hasta 3x Salida GPCLK
Almacenamiento	Slot tarjeta SD

Tabla 4.1: Características hardware de una Raspberry Pi 4B 8GB.



(a) Raspberry Pi 4B usada en el proyecto



(b) Logo Rasoberry Pi

Figura 4.1: Una Raspberry Pi 4B y el logo representativo de la marca.

4.1.2 Cámara Raspberry Pi Full HD

La fundación Raspberry Pi comercializa una serie de periféricos destinados a su uso con placas Raspberry Pi, entre ellos el módulo cámara 2.

Este módulo de cámara de bajo coste ofrece un sensor Sony IMX219 de 8 megapíxeles para capturar tanto imágenes estáticas como vídeos a distintas frecuencias y resoluciones. Se distingue este módulo de cámara 2 de su antecesor por un sensor de mayor calidad y resolución.

Las funcionalidades de la cámara son las siguientes:

- Captura de vídeo Full HD ($1920 \times 1080p$) a 30 FPS.
- Captura de vídeo Half HD ($1280 \times 720p$) a 60 FPS.
- Captura de vídeo VGA ($640 \times 480p$) a 60 FPS.
- Captura de imágenes Full HD.

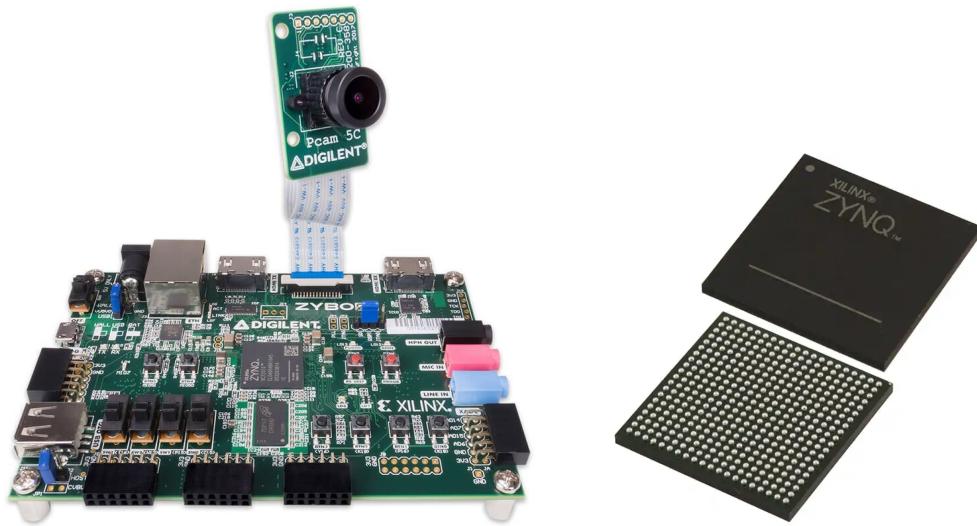
La cámara se conecta a la placa Raspberry Pi mediante el puerto CSI y es compatible con todas las versiones de Raspberry Pi que tengan este conector.

4.1.3 Chip FPGA

Para simular un diseño RTL es necesario seleccionar un chip FPGA objetivo. En este proyecto se usará como objetivo el chip **XC7Z020-1CLG400C** de la gama Xilinx Zynq que tiene un bajo coste comparable con una Raspberry Pi 4B, alrededor de €110 en el distribuidor de productos electrónicos Digikey.

Este chip se ha seleccionado por venir incluido en la placa de desarrollo Digilent ZYBO ZYNQ-7020 mostrada en la Figura 4.2, pensada para el prototipado de aplicaciones con soporte de cámaras Digilent y otros componentes. Disponible para su uso por alumnos interesados en la Universidad de Cádiz junto con la cámara Pcam 5C de Digilent, que permite grabar vídeo Full HD a 30 FPS de la misma forma que un módulo de cámara Raspberry, aunque con más funcionalidades integradas y capacidad de ser controlada directamente por el chip FPGA.

Las características del chip FPGA se muestran en la Tabla 4.2. Estos recursos indicados en la tabla son de vital importancia para el desarrollo de núcleos IP para dicho chip, ya que si el diseño obtenido requiere de más de algunos de los recursos listados, el diseño no podrá ser usado y se deberá optimizar el diseño o usar un chip más grande.



(a) La placa de desarrollo Digilent ZYBO ZYNQ-7020 con cámara.

(b) Chip FPGA XC7Z020-1CLG400C.

Figura 4.2: La placa de desarrollo objetivo y su chip FPGA integrado.

Tipo	Detalles
Series	Artix-7 FPGA
Celdas lógicas	85K
Look Up Tables (LUTs)	53200
Flip-Flops (FFs)	106400
DSP Slices	220

Tabla 4.2: Características del chip FPGA XC7Z020-1CLG400C.

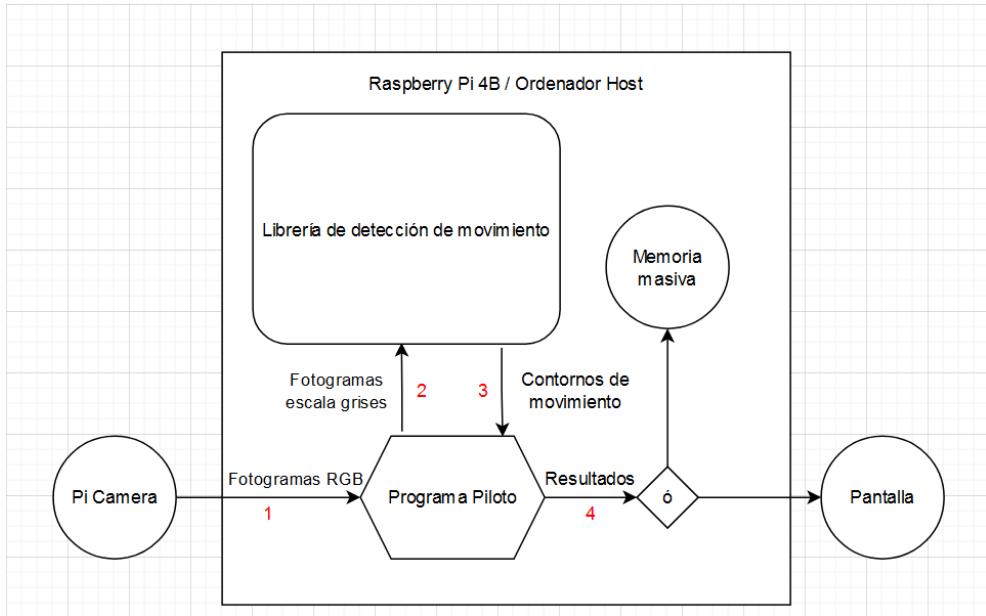


Figura 4.3: Arquitectura lógica de procesado de datos para librería CPU.

4.2 Arquitectura lógica

En esta sección se describe el modelo lógico seguido para el flujo y procesado de datos.

4.2.1 Lógica CPU independiente

En la Figura 4.3 se describe la arquitectura lógica de la librería para CPU independiente, que combina un ejemplo de un posible programa piloto interactuando con la librería desarrollada.

Un programa piloto obtiene fotogramas, en este caso del módulo cámara Raspberry Pi, lo transforma a escala de grises y los envía a la librería de detección de movimiento de forma que se procesa de forma asíncrona. En un paso posterior se solicitan los resultados de movimiento y el programa piloto decide qué hacer con estos, siendo dos posibilidades comunes guardarlos en disco o mostrarlos por pantalla.

4.2.2 Lógica FPGA independiente

La arquitectura lógica para FPGA es similar, pero en vez de realizar el procesado de datos de la librería en la CPU, se realizaría en la FPGA incorporada en el chip, o en una FPGA externa en el caso de que no se utilice un chip de la gama Zynq.

Debido a que este proyecto únicamente busca una simulación exitosa del programa, el procesado se releva al simulador RTL Vivado Xsim en vez de un chip FPGA real, que se comportará de la misma forma excepto por un tiempo de procesado mucho más lento, al tener que simular paso por paso las miles de señales involucradas en el diseño de forma secuencial, en vez de forma inherentemente paralela como lo haría una FPGA real.

Capítulo 5

Implementación

El desarrollo del proyecto pasa por distintas fases de implementación claramente divididas y ordenadas:

1. Creación de la librería de detección de movimiento básica.
2. Adaptación de la librería a FPGA usando HLS.
3. Optimización de ambos códigos y comparación.

5.1 Librería base

En esta sección se describe la teoría que soporta la detección de movimiento en vídeo y su implementación en una librería base codificada en C++ moderno.

5.1.1 Teoría de detección de movimiento

Para la detección de movimiento en vídeo se requieren múltiples algoritmos y filtros de procesamiento de imágenes aplicados secuencialmente para transformar la representación inicial, un vídeo de color RGB Full HD, a una representación de contornos binarios en los que sea posible distinguir movimiento.

A continuación se definirán los distintos algoritmos necesarios para detectar movimiento en vídeo de una forma simple, aunque en la actualidad existe una gran cantidad de métodos más avanzados y depurados.



Figura 5.1: Imagen RGB transformada a escala de grises usando conversión directa y conversión balanceada.

5.1.1.1 Transformación a escala de grises

Para la detección de movimiento el color es de poca utilidad, siendo el elemento de mayor interés los distintos niveles de luz y la variación de estos entre fotogramas. Es por esto por lo que podemos simplificar cada píxel RGB en el vídeo a escala de grises, reduciendo a su vez el espacio de memoria usado por la imagen siendo procesada.

Teniendo los 3 valores rojo, verde y azul, cada uno ocupando 8 bits, debemos de obtener una tonalidad de gris de 16 bits que lo represente. Una primera idea para el cálculo de este representante sería considerar cada color de igual importancia, tal como se muestra en la Ecuación 5.1.

$$Y' = \frac{R' + G' + B'}{3} \quad (5.1)$$

Pero esto no tiene en cuenta los pesos de luminosidad de los distintos colores[27], por lo que deberemos usar la Ecuación 5.2 en su lugar, que correctamente representa el peso de cada color en lo que se conoce como *Luma*, o Y' .

$$Y' = 0,299R' + 0,587G' + 0,114B' \quad (5.2)$$

La diferencia entre una transformación a escala de grises usando una conversión directa y una conversión con balance de pesos se puede observar en la Figura 5.1

5.1.1.2 Reducción de resolución

Procesar una imagen bruta píxel a píxel es una tarea computacionalmente cara debido a la gran cantidad de información contenida en ella, pero dependiendo del

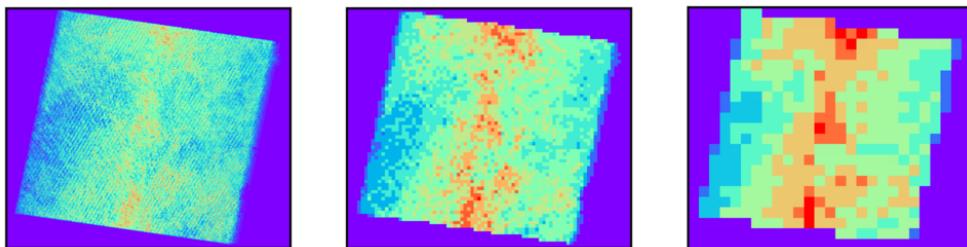


Figura 5.2: Reducción de resolución progresiva de una imagen, preservando los rasgos de interés.

Origen: Imagen obtenida del blog J.Gomez Danz (<https://jgomezdans.github.io/downsampling-with-gdal-in-python.html>)

caso de uso no se necesita información detallada de cada píxel, por lo que es un esfuerzo perdido que podría haber sido dedicado a otras operaciones.

Cuando sólo se necesita información a grandes rasgos en una imagen, es posible reducir la resolución de esta en distintos factores. Por ejemplo, reducir una imagen en factor 2 reduciría el número de columnas y filas a la mitad, siendo el valor de cada píxel resultante la media de los 4 que le corresponden en la imagen original. A este proceso se le conoce comúnmente en el campo de la visión artificial como *downsampling* y se puede ver un ejemplo de su uso en la Figura 5.2.

De forma equivalente, es posible aumentar la resolución de una imagen (*upsampling*), aunque esto no daría información adicional a un algoritmo de visión máquina ya que todos los nuevos píxeles creados son predicciones sintéticas.

5.1.1.3 Desenfoque Gaussiano

Cuando se analiza un vídeo o imagen, suele ser necesario eliminar la mayor cantidad de ruido posible y reducir el detalle de aquellos elementos demasiado pequeños como para ser relevantes. Una forma de conseguir estos dos efectos al mismo tiempo es el desenfoque Gaussiano, como se puede ver en la Figura 5.3.

El desenfoque Gaussiano es un filtro convolucional basado en una matriz de pesos que aproxima la función matemática Gaussiana. Para calcular el valor de un píxel en la imagen desenfocada resultante es necesario leer el valor de todos los píxeles vecinos afectados por dicha matriz, por lo que cuanto más grande sea la matriz, el desenfoque será de mayor calidad, pero también usará mayor potencia computacional.

Dado una longitud de lado y una desviación estándar σ se puede obtener una matriz, denominada *kernel*, que aproxime la función Gaussiana con mayor o menor detalle. Un ejemplo de un kernel G para desenfoque Gaussiano usando $\sigma = 1$ y una longitud



Figura 5.3: Aplicación de un filtro Gaussiano para la reducción de detalles no deseados.

Origen: Imagen editada de la página principal de desenfoque Gaussiano en Wikipedia.

K L I J K L I J O P M N O P M N	A A A B C D D D A A A B C D D D A A A B C D D D E E E F G H E F	F B E F G H C G E A A B C D D H B A A B C D D C F E E F G H H G	Q Q Q Q Q Q Q Q Q Q Q Q Q Q Q Q Q Q Q A B C D Q Q Q Q E F G H Q Q J I I J K L L K N M M N O P P P O I M M N O P P P L J N I J K L O K	?	?	?	?	?	?	?
C D A B C D A B G H E F G H E F	K L I J K L I J O P M N O P M N	M M M N O P P P P M M M N O P P P P	Q Q Q I J K L Q Q Q Q M N O P P Q Q Q Q Q Q Q Q Q Q Q Q Q Q Q Q Q Q	?	?	?	?	?	?	?
	(a) Repeat.	(b) Clamp.	(c) Mirror.	(d) Constant.	(e) Undefined.					

Figura 5.4: Métodos de manejo de bordes para filtros convolucionales.

Origen: Imagen obtenida del trabajo "HIPAcc: A Domain-Specific Language and Compiler for Image Processing" por Richard Membarth et al.

de lado 5 se puede observar en la Ecuación 5.3.

$$G = \frac{1}{273} \begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix} \quad (5.3)$$

Al aplicar esta matriz G a una imagen en escala de grises de 6 columnas y 8 filas, se obtiene el resultado mostrado en la Ecuación 5.4. Los bordes de la imagen se tratan utilizando un manejo de bordes *clamp*, definido en la Figura 5.4.

$$\begin{pmatrix}
 23 & 32 & 0 & 0 & 0 \\
 39 & 0 & 0 & 0 & 0 \\
 13 & 23 & 0 & 0 & 0 \\
 20 & 40 & 58 & 63 & 50 \\
 30 & 60 & 74 & 90 & 80
 \end{pmatrix} *
 \begin{pmatrix}
 1 & 4 & 7 & 4 & 1 \\
 4 & 16 & 26 & 16 & 4 \\
 7 & 26 & 41 & 26 & 7 \\
 4 & 16 & 26 & 16 & 4 \\
 1 & 4 & 7 & 4 & 1
 \end{pmatrix} =
 \begin{pmatrix}
 23 & 17 & 7 & 1 & 0 & 0 \\
 23 & 16 & 8 & 4 & 3 & 3 \\
 22 & 22 & 20 & 19 & 16 & 14 \\
 26 & 35 & 43 & 46 & 43 & 39 \\
 27 & 42 & 56 & 63 & 62 & 57 \\
 17 & 31 & 45 & 52 & 52 & 48 \\
 6 & 13 & 19 & 23 & 23 & 21 \\
 0 & 2 & 3 & 4 & 4 & 4
 \end{pmatrix}
 \quad (5.4)$$

Dado un kernel 5x5 es necesario realizar 25 multiplicaciones, 25 sumas y 1 división para obtener un sólo píxel desenfocado, si se repite este proceso para toda una imagen completa el consumo de recursos será demasiado elevado para su uso práctico. Es por ello por lo que es conveniente usar la propiedad de *kernels separables*, que permite separar un kernel de desenfoque Gaussiano en 2 vectores equivalentes de longitud igual al lado de este, reduciendo el número de operaciones a menos de la mitad.[28]

5.1.1.4 Sustracción de fondo

La sustracción de fondo es una técnica utilizada para la detección de cambios en una imagen dada respecto a otra imagen de referencia denominada *fondo*. Esta técnica es frecuentemente utilizada para la detección de movimiento en vídeo, donde se mantiene un fotograma guardado en memoria con el que se comparan todos los subsecuentes fotogramas capturados, como se muestra en la Figura 5.5.

La sustracción de fondo en sí es una operación simple ya que sólo hay que calcular la diferencia absoluta entre cada píxel de la imagen entrante y la de referencia. Las complicaciones surgen cuando se aplica esta técnica a vídeos, en los que es necesario actualizar la imagen de referencia de forma periódica.

Si se utiliza el primer fotograma de un vídeo como referencia y este no es actualizado más, es posible que, dependiendo del caso de uso, se den los siguientes problemas:

- **Cambios de luz en la escena:** Ya sea por el ciclo natural día/noche o por luz artificial, una vez cambie el nivel de luminiscencia de la escena lo suficiente, una gran cantidad de píxeles serán detectados como diferencias respecto al fotograma de referencia.

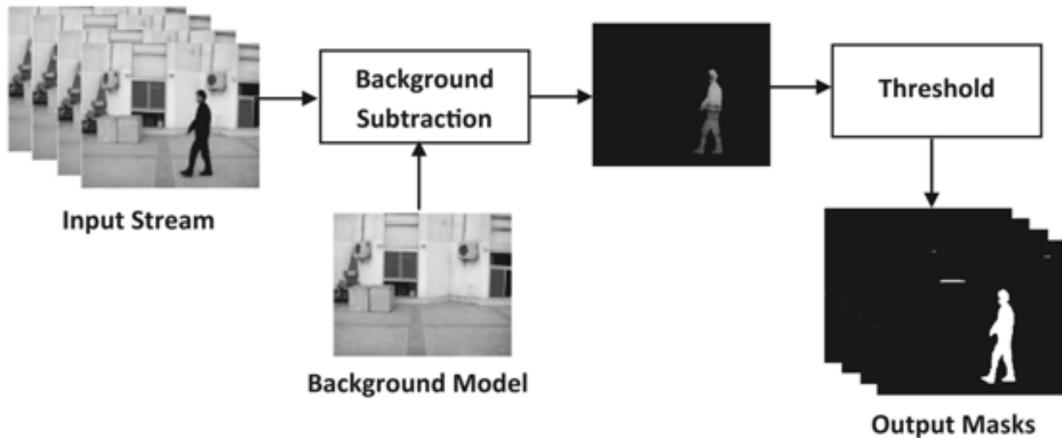


Figura 5.5: Sustracción de fondo.

Sustracción de fondo en vídeo y una posterior umbralización del resultado.

Origen: Imagen obtenida del trabajo "Moving Object Detection Using Background Subtraction" por Soharab Hossain Shaikh et al.

- **Cambios físicos en la escena:** Cuando ocurre un cambio en la escena, por ejemplo, se coloca un objeto que no está presente en el fotograma de referencia, este será detectado como una diferencia con la referencia de forma permanente hasta que el objeto se retire.

Para evitar estos problemas es conveniente implementar un algoritmo de interpolación de fondo que modifique levemente la imagen de referencia con cada fotograma obtenido. Esto soluciona los problemas anteriormente mencionados, pero no soluciona los artefactos visuales denominados *fantasmas*, que ocurren cuando un objeto es eliminado de la escena y, a pesar de que ya no está ahí, su contorno sigue siendo detectado por un tiempo hasta que se ajuste la referencia. Este problema ha sido investigado en gran profundidad y existen diversas soluciones con mayor o menor éxito, pero en este proyecto no se tratarán con el objetivo de mantener un código simple.

5.1.1.5 Umbralización

La umbralización de una imagen en escala de grises consiste en simplificarla a una imagen binaria, en la que un píxel sólo puede ser blanco o negro. Esta técnica es de gran utilidad para el análisis de imágenes ya que hace más fácil detectar donde empieza y acaban los contornos de interés.

Existen diversos métodos para umbralizar una imagen, siendo el más simple de ellos la selección de un valor umbral y colapsando todos los píxeles con un valor luma

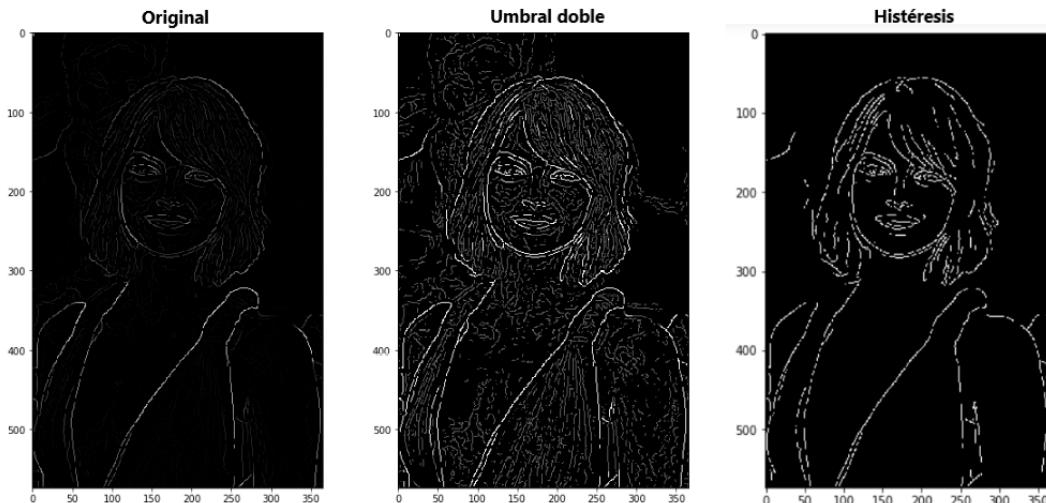


Figura 5.6: Aplicación de umbral doble e histéresis a una imagen en escala de grises.

Origen: Imagen editada del blog Towards Data Science (<https://towardsdatascience.com/canny-edge-detection-step-by-step-in-python-computer-vision-b49c3a2d8123>).

por debajo de este umbral a 0, y todos los demás a 1. El beneficio de este método es su simplicidad y velocidad, pero los resultados obtenidos pueden ser subóptimos debido a la dificultad de seleccionar un umbral apropiado que se adapte a los distintos colores y cambios de luminosidad en la escena.[29]

Existen diversos métodos para ajustar el umbral de forma automática, pero en este proyecto se usará un método de doble umbral más simple.[30]

Se seleccionan 2 umbrales, uno superior y otro inferior. Cualquier píxel con un luma superior al umbral superior se marcará como un píxel fuerte (1) mientras que cualquier otro píxel por encima del umbral inferior se marcará como un píxel débil (2), marcándose todos los demás píxeles restantes como 0 (eliminado).

Como paso adicional se aplica una técnica de histéresis que analizará todos los píxeles débiles y los colapsará a 1 o 0 dependiendo de su relevancia. En este proyecto se considera que un píxel débil se debe colapsar a 1 (fuerte) si está conectado a otro píxel fuerte ya sea directamente como un vecino de este o indirectamente por un camino de píxeles débiles. En la Figura 5.6 se puede ver una aplicación de este algoritmo.

Usar un umbral doble junto con histéresis permite obtener resultados mejores para distintos niveles de luz que un sólo umbral, pero requiere de mayor potencia computacional.

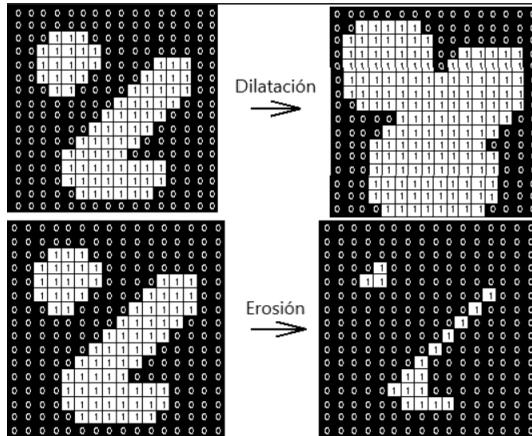


Figura 5.7: Las operaciones morfológicas de dilatación y erosión.

Origen: Imagen editada de <https://www.cs.auckland.ac.nz/courses/compsci773s1c/lectures/ImageProcessing-html/topic4.htm>.

5.1.1.6 Dilatación

La dilatación de una imagen binaria es una operación morfológica que consiste en aumentar el tamaño de las componentes conexas de píxeles con valor 1 (denominados *1-píxeles*), de forma que se tapen agujeros, se suavicen bordes con ruido, y que múltiples componentes conexas pequeñas se unan en una sola componente conexa de mayor tamaño.

Esta operación es de gran utilidad para la detección de movimiento ya que puede ayudar a conectar 2 o más componentes de medio tamaño en una de gran tamaño, haciendo más fácil su detección como ‘movimiento’ en el vídeo.

También existe la operación morfológica inversa, la erosión, que reduce el tamaño de las componentes conexas, pero esta no es de gran utilidad en el caso de uso de este proyecto. Se puede ver los efectos de la erosión y dilatación en la Figura 5.7.

La dilatación se puede implementar como un filtro convolucional: Se visita cada píxel de una imagen y si el píxel tiene valor 0 (denominado *0-píxel*), se miran sus vecinos en busca de un 1-píxel y si se encuentra al menos uno, el píxel siendo evaluado también se pone a valor 1, en cualquier otro caso, se mantiene en 0.

De una forma similar al desenfoque Gaussiano esta operación se puede dividir en 2 vectores equivalentes para reducir el número de comprobaciones necesarias.

5.1.1.7 Detección de componentes conexas

El último algoritmo usado para la detección de movimiento es la detección de componentes conexas para imágenes binarias, o la detección de contornos que daría el mismo resultado en este caso. Una vez se ha obtenido una imagen binaria es necesario localizar los distintos conjuntos de 1-píxeles para evaluar su tamaño y clasificarlos como movimiento o no.

Un algoritmo de detección de componente conexas consiste en iterar sobre todos los píxeles de la imagen y asignar una etiqueta a cada 1-píxel. La forma en la que se asigna una etiqueta u otra depende del algoritmo utilizado, pero generalmente se busca que dados 2 1-píxeles cualquiera interconectados, estos tengan la misma etiqueta asignada.

Debido a que es necesario iterar la imagen en un orden secuencial, usualmente desde la esquina superior izquierda hasta la esquina opuesta usando un escaneo *ráster*, es posible que se asignen 2 o más etiquetas distintas a una misma componente conexa, un claro ejemplo de esto sería una componente dispuesta en forma de U, en la que cada mitad de la U tendría una etiqueta distinta.

Para solventar esto se puede utilizar una clásica herramienta en el campo de teoría de grafos, una estructura de datos de componentes conexas. Cada componente conexa en esta estructura de datos guarda una etiqueta identificativa y una referencia a otra componente conexa que la representa, cada una representándose a sí misma inicialmente. Una vez se detecta mientras se escanea la imagen que dos etiquetas están conectadas entre sí, es posible conectarlas usando esta estructura de datos de forma eficiente haciendo que ambas componente conexas tengan el mismo representante. A su vez, mientras se actualizan las componentes conexas se hace un seguimiento del área que estas ocupan.

Una vez se ha escaneado la imagen al completo y se conocen los representantes y áreas de todas las componentes conexas es posible determinar si estas son movimiento válido o no.

Cabe notar que para cada píxel sólo es necesario comprobar el píxel a la izquierda y arriba de este, no el vecindario completo, reduciendo el número de comprobaciones a la mitad. Siguiendo este modelo se obtienen resultados como el mostrado en la Figura 5.8.

El otro método, la detección de contornos, se basa en el seguimiento de los bordes de cada componente conexa. Una vez se encuentra una componente conexa que no se ha encontrado hasta ahora, se sigue el borde de esta hasta completar un área cerrada o alcanzar el borde de la imagen. Este método da el mismo resultado que

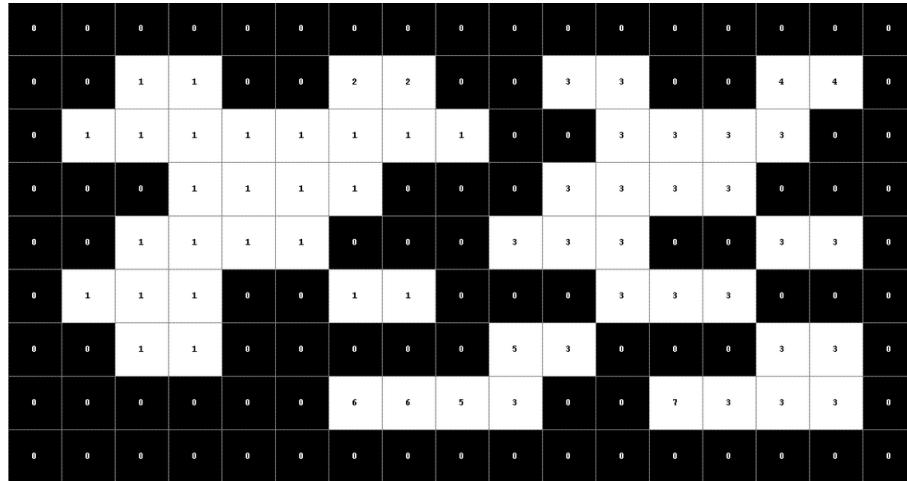


Figura 5.8: 1-píxeles con sus correspondientes etiquetas tras aplicar detección de componentes conexas. Obsérvese que las etiquetas 3, 4, 5, 6 y 7 pertenecen a la misma componente conexa.

Origen: *Imagen obtenida de la página principal de etiquetado de componentes conexas de Wikipedia.*

la detección de las componentes conexas al completo ya que sólo es de interés el tamaño del contorno y su posición.

El seguimiento de contornos es un método simple y apto para CPU, mientras que la detección de componentes conexas es más apto para FPGA. Se profundizará en este concepto en secciones posteriores.

5.1.2 Implementación de la librería

En esta sección analiza la estructura de la librería y las técnicas utilizadas para su implementación. Cabe notar que la librería está diseñada y comentada completamente en inglés, por lo que en cada sección se nombrarán los elementos que la forman tanto en español como inglés si procede.

5.1.2.1 Estructura y herramientas de automatización

Una de las claves para un proyecto de calidad es tener una buena organización de directorios y ficheros, además de hacer uso de herramientas de construcción automatizadas para lograr buena mantenibilidad y maximizar la productividad del equipo de desarrollo.[31]

5.1.2.1.1 Make

Make es una herramienta lanzada en 1976 de gestión de dependencias para código fuente de proyectos, de forma que dirige su generación y compilación de forma automática en un fichero ejecutable.

Make es una herramienta muy potente usada en una gran cantidad de proyectos a ser compilados en Unix o Linux que interpreta unos ficheros denominados *makefile*, los cuales contienen instrucciones detalladas para construir el proyecto.

Si bien Make es de gran potencial y flexibilidad, en la actualidad su sintaxis y lógica puede parecer excesivamente complicada y confusa para la gran mayoría de desarrolladores[32]. Es por este motivo por lo que aparecieron distintos sistemas de construcción que automatizan la generación de ficheros makefile de forma que el desarrollador solo tenga que definir una serie de simples reglas que definan la lógica de construcción deseada.

5.1.2.1.2 CMake

CMake es la herramienta de generación de ficheros makefile usada en este proyecto por ser multiplataforma, de código abierto y ampliamente adoptada en la industria de desarrollo en C++. El logo representativo del software y usado en los ficheros *cmake* se puede ver en la Figura 5.9.

CMake permite reducir lo que serían cientos de directivas Make en una serie de instrucciones de alto nivel en un fichero denominado *CMakeLists.txt*, simplificando el proceso de creación y actualización del código fuente y, por ende, mejorando la calidad del código.

CMake, de una forma similar al propio lenguaje C++, ha existido por un largo periodo de tiempo y ha visto una gran cantidad de actualizaciones y mejoras. Cada vez que se ha lanzado una nueva versión de CMake se ha necesitado mantener la retrocompatibilidad con patrones y directivas deprecadas en versiones anteriores, resultando en una separación entre lo que se conoce como CMake moderno y CMake clásico, siendo el primero el utilizado en este proyecto.[33]

5.1.2.1.3 Git

Git es una herramienta de control de versiones diseñada por Linus Tolvalds ampliamente adoptada en una gran variedad de proyectos alrededor del mundo.



Figura 5.9: El logo de la herramienta de construcción CMake.

Origen: *Imagen obtenida de la página oficial de CMake (<https://cmake.org/>).*

Git es una herramienta altamente eficiente y robusta que facilita la mantenibilidad del código fuente generado ya que permite a equipos de desarrollo colaborar de forma fácil y efectiva, y a versionar el código en *repositorios* de forma que se tiene una historia de todos los cambios realizados.

En este proyecto se usa esta herramienta junto con la plataforma GitHub, que permite guardar repositorios en la nube de forma gratuita, ya sea de forma pública o privada.

5.1.2.1.4 Doxygen

Doxygen es el estándar de facto para la generación de documentación a partir de código fuente C++ anotado, aunque también soporta otros lenguajes como C, PHP, C, Java, Python, etc...

Doxygen permite a los desarrolladores estandarizar el estilo de comentado de código a la vez que se habilita la generación de documentación con un formato claro y estructurado.

El formato en el que se genera la documentación es configurable y permite ser generada para una gran cantidad de plataformas y formatos, siendo algunas de las más notables el formato Web y el formato LATEX.

Un ejemplo de documentación en formato Web tomado de este mismo proyecto se puede observar en la Figura 5.10.

5.1.2.1.5 Estructura de la librería

Aunque existen distintas nociones de organización de proyectos y de estilo de codificación en C++, no existe un estándar establecido, por lo que cada proyecto es

The screenshot shows a detailed class reference for the `Motion_detector` class. It includes sections for `Classes` (containing `Contour` and `Detection`) and `Public Member Functions` (listing constructors, assignment operators, and member functions like `get_width()`, `get_height()`, and `get_total()`). The interface is a standard Doxygen-style navigation bar at the top.

Figura 5.10: Parte de la documentación web generada por Doxygen para la clase detectora de movimiento del proyecto.

libre de seguir la organización que considere oportuna siempre que quede claramente documentada y accesible a cualquier contribuidor.[34]

Es común que nuevos proyectos decidan utilizar la organización de otras librerías y proyectos populares para reducir el esfuerzo inicial de documentación y planificación. En el caso de esta librería, se usará el modelo de las librerías C++ POCO, cuyas normas de estilo se pueden encontrar públicamente en la web oficial del proyecto.[35]

La estructura de la librería es la siguiente:

- (D) **build** - *Ficheros generados por CMake.*
- (D) **doc** - *Documentación Doxygen.*
- (D) **include** - *Cabeceras públicas de la librería.*
 - (F) `motion_detector.hpp`
- (D) **src** - *Cabeceras privadas y código fuente.*
 - (F) `motion_detector.cpp`
 - (F) `image_utils.hpp`
 - (F) `image_utils.ipp`
 - (F) `contour_detector.hpp`

- (F) contour_detector.cpp
- (D) **test** - *Pruebas de unidad.*
 - (F) test_main.cpp
 - (F) test_utils.hpp
 - (F) test_motion_detector.hpp
 - (F) test_motion_detector.cpp
 - (F) test_image_utils.hpp
 - (F) test_image_utils.cpp
 - (F) test_contour_detector.hpp
 - (F) test_contour_detector.cpp
- (F) CMakeLists.txt

5.1.2.2 Interfaz pública

Un factor que determina la calidad de una librería es la forma en la que los usuarios interactúan con ella, es decir, su interfaz pública.

En esta sección se explican los componentes que componen esta interfaz, su diseño y detalles de implementación.

5.1.2.2.1 Clase imagen

Una imagen puede tener una gran variedad de tipos de píxeles de distintos tamaños y estructuras, es por esto por lo que es conveniente centralizar todos los posibles tipos que se puedan necesitar en una misma clase genérica.

La clase plantilla std::vector proveniente de la STL de C++ permite a un usuario crear una colección ordenada de elementos de cualquier tipo genérico, por lo que es posible almacenar en ella una colección de cualquier tipo de píxel. El único problema es que esta estructura es de una sola dimensión, mientras que una imagen tiene 2, altura y anchura.

Para solventar este problema se crea la clase *Image*, que es una clase plantilla que contiene en su núcleo un std::vector como estructura de almacenamiento de datos y que es capaz de simular 2 dimensiones al contener información sobre el número de

columnas y filas de la imagen representada por este vector. Este modelo permite al usuario utilizar los datos contenidos como si se tratase de una estructura bidimensional, es decir, permite acceder a un píxel seleccionando una fila y, dentro de esta, una columna.

5.1.2.2.2 Clase detector de movimiento

La clase *Motion_detector* contiene una serie de métodos que permiten lograr los requisitos funcionales descritos en la Sección 3.3.1.1, es decir, permite crear un objeto de esta clase, destruirlo, y utilizarlo para solicitar detección de movimiento en imágenes en escala de grises de 16 bits (Objetos Image con tipo de píxel *unsigned short*).

Esta clase busca un funcionamiento asíncrono de forma que se ejecute en paralelo al programa piloto, permitiendo introducir una nueva imagen a ser procesada en el detector de movimiento y, en algún momento en el futuro, recoger los resultados sin tener que esperar de forma activa.

Este diseño asíncrono se logra usando colas de fotogramas y técnicas multihilo, de forma que hay un número variable de esclavos esperando a que se añada un nuevo fotograma a la cola de entrada para apropiarse de él y procesarlo. Una vez uno de estos hilos termina de procesar el fotograma que se ha apropiado lo marcará como procesado, y si se da el caso de que este fotograma también es el más antiguo, lo moverá a la cola de salida de forma que sea accesible al programa piloto. Cuando el programa piloto solicite obtener resultados se mirará esta cola de salida, y en caso de que haya algún resultado en ella, se retornará y se eliminará de esta, respetándose siempre el orden de los fotogramas y sin saltarse ninguno.

Este modelo asíncrono se basa en un modelo paralelo maestro-esclavo, en el que el programa piloto es el maestro y permite aprovechar los distintos núcleos de procesamientos disponibles en una CPU de una forma sencilla y efectiva. Un diagrama de este modelo paralelo se puede ver en la figura 5.11.

Cabe notar que cuando se tienen varios hilos accediendo a un recurso compartido, como son en este caso las colas de entrada y salida, es necesario implementar mecanismos de sincronización que eviten comportamientos erróneos e inesperados, como pueden ser los *std::mutex* o *std::condition_variable* entre otros.

Los distintos parámetros como son el número de hilos, los tamaños de colas, la resolución de los fotogramas y otros específicos a los algoritmos de detección de movimiento internos son especificados en el constructor de la clase, por lo que es completamente configurable por el usuario de la librería.

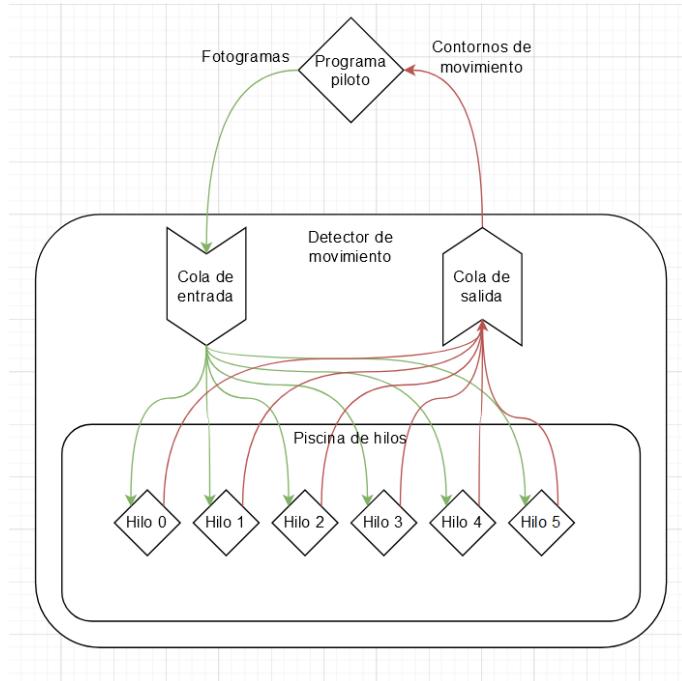


Figura 5.11: El modelo asincrónico paralelo usado por la clase Motion_detector.

5.1.2.2.3 Pipeline principal

Cada hilo esclavo del detector de movimiento debe transformar un fotograma a un conjunto de contornos que representan el movimiento detectado en él. Para lograr esta transformación de datos es necesario ir procesando las imágenes paso a paso, cambiando la imagen de entrada en otras de mayor utilidad hasta que se puedan distinguir componentes conexas claras, es decir, es necesario crear un *pipeline* de procesamiento de imágenes.

Una vez se han seleccionado los algoritmos a utilizar y se conoce el orden en el que hay que aplicarlos es posible crear el pipeline que se encargará de secuenciar los procesos y controlar el flujo de imágenes.

Para lograr detección de movimiento de una forma simple se ha seleccionado el siguiente pipeline, asumiendo una entrada en escala de grises:

1. Reducción de resolución o downsampling.
2. Desenfoque Gaussiano.
3. Se comprueba la imagen de referencia guardada en memoria.
 - Si no existe una imagen de referencia, se guarda el fotograma actual como

referencia y se termina la ejecución de pipeline, no retornando ningún contorno de movimiento.

- Si existe una imagen de referencia, se interpola el fotograma actual con la referencia, actualizándola.

4. Sustracción de fondo.
5. Umbralización.
6. Dilatación.
7. Detección de componentes conexas.

Como paso final al completar el pipeline se comprobará cada una de las componentes conexas o contornos detectados y se determinará, según su tamaño, si son suficientemente relevantes para ser informadas como *movimiento* detectado en la cola de salida. Este paso se hace para evitar falsos positivos al informar del ‘movimiento’ conformado por una cantidad muy pequeña de píxeles, que podría haber sido causado por ruido en el vídeo, vibraciones u otros fenómenos de poca importancia.

5.1.2.2.4 Transformación a escala de grises

Como utilidad adicional, se ha añadido a la interfaz pública la transformación de imágenes RGB a imágenes en escala de grises de 16 bits.

Esta función no se incluye en el pipeline de procesamiento ya que es posible que un usuario desee utilizar la librería con una cámara monocromo, en vez de una a color.

5.1.2.3 Algoritmos de procesamiento de imágenes

En esta sección se analizará la implementación de los algoritmos usados en el pipeline de detección de movimiento diseñado, con los que se logran los resultados mostrados en la Figura 5.12.

5.1.2.3.1 Reducción de resolución

La función *downsample* toma una imagen de entrada con una resolución $A \times B$ determinada junto a un factor de reducción F y produce una imagen transformada de resolución reducida $\lceil \frac{A}{F} \rceil \times \lceil \frac{B}{F} \rceil$.

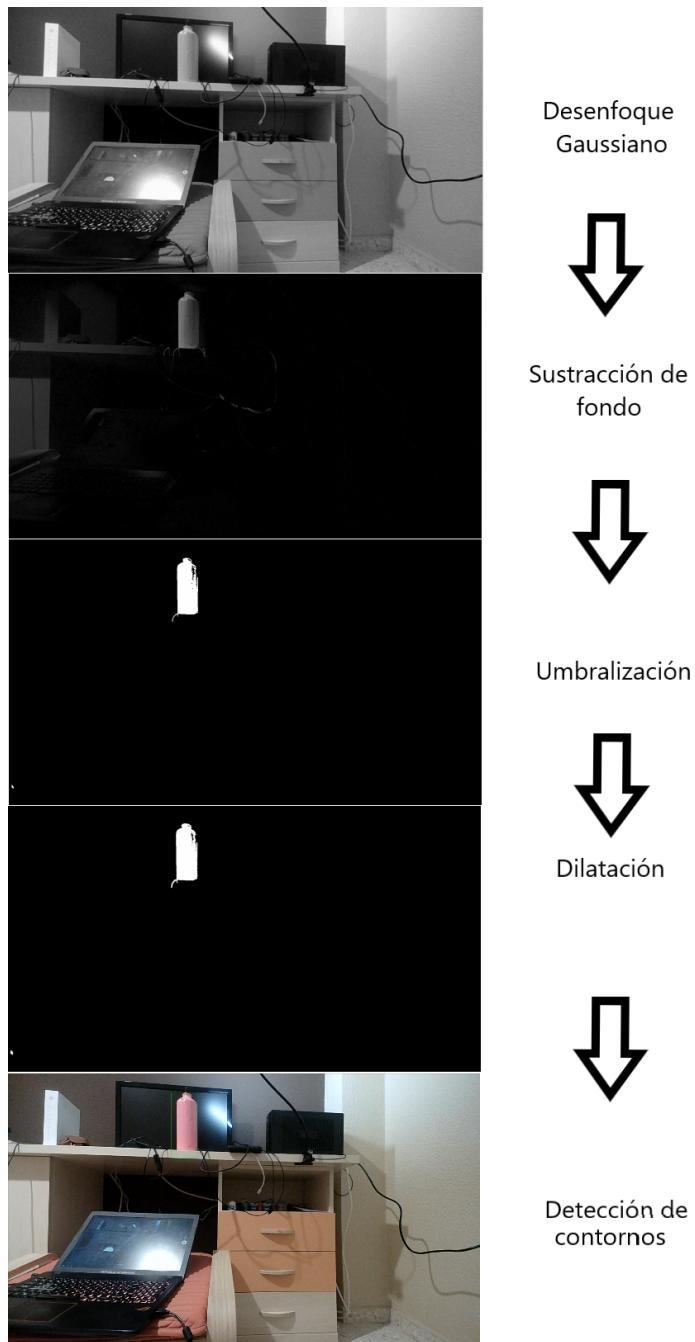


Figura 5.12: El pipeline completo de detección de movimiento paso a paso. Se detecta como movimiento la cantimplora rosa, que ha sido colocada en la posición mostrada segundos antes.

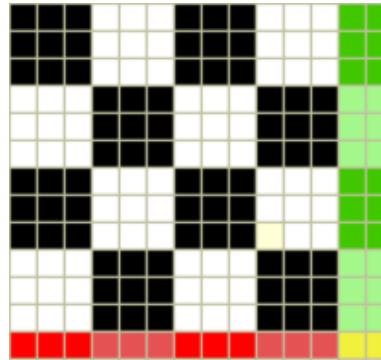


Figura 5.13: Al reducir una imagen con resolución 14x13 en factor 3 ($F=3$), aparecen regiones irregulares en los bordes de la imagen que debe ser tratados por separado, marcados en rojo, verde y amarillo. La parte regular se indica con los cuadrados blancos y negros.

La imagen original se segmenta en una cuadrícula de agrupaciones de píxeles cuadradas de lado F , de forma que cada píxel de la imagen resultante tenga una agrupación que lo represente. Cada píxel de la imagen resultante se calcula, por lo tanto, rea- lizando la media de la luminiscencia de los píxeles que conforman su agrupación representante.

Cuando el número de filas o columnas de la imagen no es un múltiplo de F , aparecen agrupaciones de píxeles incompletas que deben tratarse de forma especial. En este proyecto se ha optado por tratar estos casos especiales como si se agrupaciones completas se tratasesen, aunque también sería posible ignorar estos píxeles por sim- plicidad. Se pueden ver un ejemplo de las distintas áreas conflictivas que aparecen al procesar imágenes de resolución arbitraria en la Figura 5.13.

5.1.2.3.2 Desenfoque Gaussiano

La función *gaussian_blur_filter* se implementa mediante convolución usando dos vec- tores de longitud 5, logrando así el mismo efecto que produciría un kernel cuadrado 5×5 como se indica en la Sección 5.1.1.3.

Con el objeto de reducir la cantidad de código duplicado se crearán dos funciones genéricas, una de convolución 1D vertical (*vline_convolution*) y otra de convolución 1D horizontal (*hline_convolution*), que permitirán recibir una imagen genérica de entrada y un kernel 1D de longitud variable que debe ser aplicado a esta para producir una imagen de salida. El kernel es representado por un *std::function* que implementa la lógica deseada, siendo en este caso el desenfoque de los píxeles de una región determinada.

Estas funciones genéricas aseguran una aplicación correcta del kernel especificado

incluso en los bordes de la imagen, por lo que la implementación de la función *gaussian_blur_filter* consiste en simplemente ejecutar una convolución horizontal seguida de una convolución vertical a la imagen de entrada.

5.1.2.3.3 Procesado de la referencia

Una vez se tiene una imagen desenfocada se deberá comparar con la imagen de referencia para ver las diferencias que existen y, como paso adicional, se deberá modificar la referencia para mantenerla actualizada.

El primer paso, implementado en la función *image_subtraction*, se logra de forma simple iterando sobre ambas imágenes obteniendo la diferencia absoluta entre cada píxel.

El segundo paso, implementado en la función *image_interpolation*, es mas complejo ya que es necesario aplicar la Ecuación 5.5, en la que P es un píxel de una imagen dada y R es el ratio de interpolación, entre 0 y 1.

$$P_{\text{resultado}} = P_{\text{referencia}} + R * (P_{\text{actual}} - P_{\text{referencia}}) \quad (5.5)$$

5.1.2.3.4 Umbralización

La umbralización se compone de dos pasos, el colapso de píxeles en los tres estados eliminado (*Culled*), débil (*Weak*) y fuerte (*Strong*), y la histéresis, que elimina todos los píxeles débiles transformándolos a uno de los otros dos estados.

El primer paso se resuelve de forma iterativa en la función *double_threshold* comprobando a que categoría pertenece cada píxel mediante comparaciones con los umbrales dados, mientras que el segundo paso, la función *hysteresis*, requiere de estructuras de datos especiales.

Deducir si un píxel débil esta conectado directa o indirectamente a un píxel fuerte como se indica en la Sección 5.1.1.5 es demasiado complejo de realizar usando métodos iterativos. Es por esto por lo que se utilizará la estructura pila *std::stack* de la STL, de forma que primeramente se itera la imagen buscando todos los píxeles fuertes mientras se almacenan sus posiciones en esta.

Tras iterar la imagen al completo, se irán sacando píxeles de la pila y se comprobarán sus vecinos de forma que si hay algún píxel débil entre estos, se marcarán como fuertes y se guardarán sus posiciones en la pila de nuevo. Se repite este proceso en bucle hasta que la pila quede vacía, mientras se asegura que no se visita un mismo

píxel múltiples veces manteniendo en paralelo una imagen de booleanos en la que se marca si un píxel ya ha sido visitado o no.

La imagen de salida comienza siendo una imagen vacía con todos sus píxeles a 0 (eliminados), a la que se va escribiendo píxeles fuertes según se sacan de la pila. Se opta por este método en vez de modificar la imagen de entrada ya que es posible que haya píxeles débiles que no estén conectados a ningún píxel fuerte, por lo que podrían quedarse como residuo al no ser visitados.

Una vez la pila está vacía, la imagen de salida solo tendrá píxeles 1 y 0, por lo que el proceso estará completado. Si bien es cierto que esta implementación es simple, su orden de complejidad no es fácilmente acotable ya que depende del número de píxeles fuertes y débiles en la imagen dada.

5.1.2.3.5 Dilatación

La dilatación, al igual que el desenfoque Gaussiano, es un proceso convolucional divisible en dos vectores 1D, por lo que la función *dilation* simplemente llama a las funciones de convolución vertical y horizontal con un kernel que comprueba si existe al menos un píxel con valor 1 en el vecindario del píxel siendo procesado, en cuyo caso el píxel se pone a valor 1.

5.1.2.3.6 Detección de contornos

La detección de contornos se implementa como indica el estudio ‘*Topological Structural Analysis of Digitized Binary Images by Border Following*’ por Satoshi Suzuki en la función *contour-detection*.

Se itera sobre la imagen en busca de píxeles con valor 1, los cuales se consideran como potenciales contornos de interés. Cuando se encuentra un nuevo píxel con valor 1 se para de iterar y se comienza a seguir el contorno encontrado hasta que se hace un ciclo completo, cambiando todos los píxeles visitados por una etiqueta numérica con valor mayor que 1 que identifica de forma única al contorno.

Una vez de termina de iterar sobre la imagen al completo, todos los contornos habrán sido seguidos y guardados con su correspondiente etiqueta, sabiéndose su posición, área y otros datos topológicos.

Como paso adicional, se filtran todos los contornos que sean demasiado pequeños, para evitar informar sobre movimiento no relevante al usuario de la librería. El área mínima que debe tener un contorno depende de la resolución de la imagen original.

5.1.2.4 Diseño de la librería

La librería busca alta mantenibilidad y reusabilidad de código, por lo que se implementarán todos los algoritmos de procesamiento de imágenes de una forma genérica usando plantillas.

Cada función de procesamiento aceptará imágenes de cualquier tipo de píxel compatible según se define en su especificación Doxygen, de forma que se pueda aplicar la misma lógica a, por ejemplo, una imagen en escala de grises con píxeles de 8 bits y a otra de 16 bits sin necesitar cambiar el algoritmo.

Esta generalidad permite hacer modificaciones al código de una forma fácil y habilita su reutilización en una gran variedad de proyectos de visión artificial.

5.1.2.5 Optimizaciones

Una vez se tiene una librería base, es posible hacer optimizaciones sobre esta que reduzcan la carga computacional requerida.

Para la versión optimizada de la librería (*version fast*) se harán los siguientes cambios:

- **Minimización de operaciones de coma flotante:** Al tratar píxeles representados como enteros de 16 bits es conveniente evitar la transformación de estos valores a coma flotante para su cálculo y su posterior conversión de vuelta a entero. Si bien es cierto que esto da mayor precisión a los resultados, en el procesado de imágenes suele ser de mayor interés aumentar el rendimiento y reducir la complejidad de las operaciones. Un ejemplo de esta optimización se da con el uso de un kernel de enteros para el desenfoque Gaussiano.
- **Reducción de bucles:** Cada iteración de un bucle requiere aumentar contadores y hacer saltos, por lo que es preferible reducir el número de bucles en el programa. Por ejemplo, las funciones *image_subtraction* e *image_interpolation* pueden ser unidas en una sola función.
- **Reducción de generalidad:** Usar funciones genéricas es una gran forma de mantener código mantenable y limpio, pero en el caso de que sólo se utilicen un número reducido de veces puede ser beneficioso usar funciones especializadas en su lugar que aprovechen al máximo las características de cada problema. Un ejemplo de esto es el uso de las funciones de convolución genéricas por la función *dilation*, que podrían ser especializadas para sólo comprobar los vecinos de un píxel si este tiene el valor 0.

- **Reducción de condicionales en bucles:** La ejecución condicional en bucles puede bloquear optimizaciones como son el desenrollado de bucles y predicción de saltos, por lo que evitarla suele dar beneficios de rendimiento y calidad de código.
- **Reducción de detalles calculados:** Si una pieza de información obtenida no es de interés, es posible estudiar formas de evitar obtenerla reduciendo así la carga computacional del algoritmo. Un ejemplo de detalles innecesarios se da en la función de detección de contornos, en la que se analiza la topología de los contornos pero nunca se utiliza.

5.2 Adaptación a FPGA HLS

La librería base diseñada para CPU será portada a FPGA HLS para su aceleración y reducción de consumo eléctrico. En esta sección se verá la teoría en la que se basa la codificación en HLS y su uso para adaptar la librería siendo desarrollada.

5.2.1 Teoría HLS

HLS permite transformar lógica descrita en lenguajes secuenciales con un alto nivel de abstracción como C++ en diseños RTL hardware, que tienen de forma implícita un fuerte concepto del tiempo y paralelismo en su diseño.

Para permitir lograr esta conversión de una forma eficaz, HLS ofrece al desarrollador una serie de herramientas en forma de directivas del preprocesador con las que se pueden modificar secciones de código u objetos específicos con funcionalidades de aceleración y síntesis. Estas directivas se les conoce como **pragmas** y en esta sección se verán las más relevantes.

5.2.1.1 Métricas de velocidad

Antes de ver los distintos pragmas de aceleración ofrecidos por HLS es conveniente entender las métricas comúnmente usadas para cuantificar la eficacia de un diseño HLS:

- **Latencia:** Es el número de ciclos necesarios para que una función devuelva los resultados calculados desde el momento que se inició su ejecución.

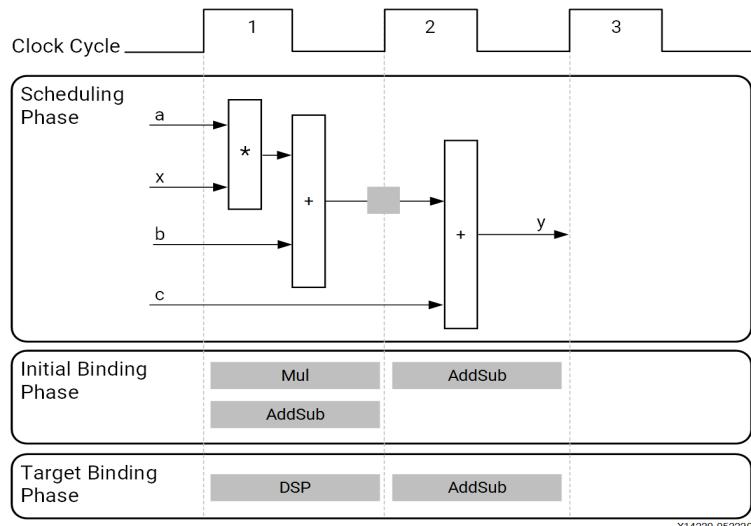


Figura 5.14: Un mismo ciclo de FPGA puede procesar dos o más operaciones con dependencias entre ellas, dependiendo de la longitud del ciclo y la velocidad de la FPGA. En el ciclo 1 se procesa de forma completa una multiplicación y una suma.

Origen: *Introducción a HLS por Xilinx* (https://www.xilinx.com/html_docs/xilinx2021_1/vitis_doc/introductionvitislhs.html)

- **Intervalo de iniciación (II):** Es el número de ciclos que tarda la función en ser capaz de volver a procesar datos desde el momento que se inició su ejecución. Esta métrica es, por lo general, 1 ciclo más larga que la latencia.
- **Latencia de bucle:** Al igual que las funciones, se puede medir la latencia de un bucle, es decir, el tiempo que tarda el bucle en ejecutar todas sus iteraciones.
- **II de bucle:** El número de ciclos que tarda en ejecutarse una iteración de un bucle y el tiempo hasta que este bucle sea capaz de ejecutar la siguiente iteración.

Nótese que un ciclo en una FPGA es mucho más poderoso que en una CPU ya que gracias a su naturaleza altamente paralela y especializada es posible ejecutar múltiples operaciones con dependencias entre ellas en un mismo ciclo de reloj, o incluso cientos de operaciones paralelas en el caso de que no haya dependencias. Esta propiedad permite el procesado de bucles a un ciclo por iteración, mientras que en una CPU se pueden llegar a necesitar incluso decenas de ciclos de reloj por iteración. Un ejemplo simple de esta propiedad se puede observar en la Figura 5.14.

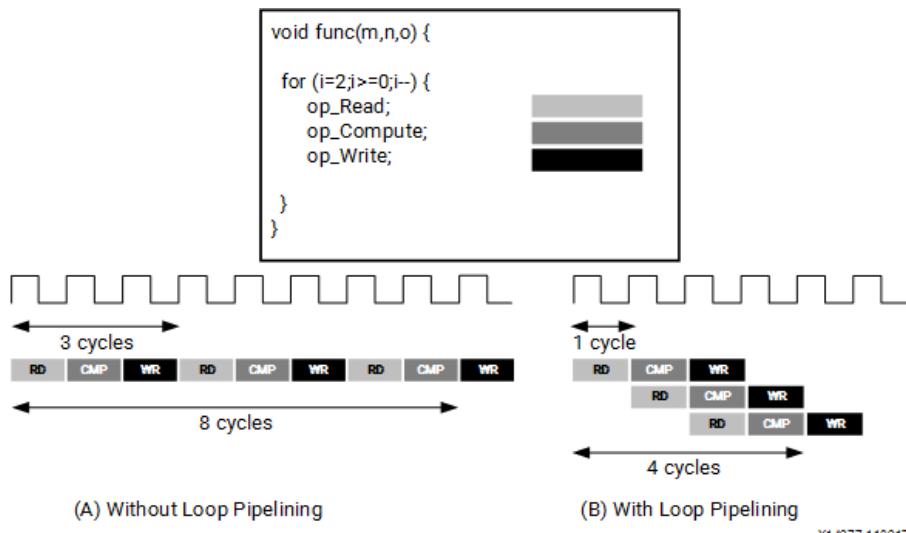


Figura 5.15: Un bucle con una lectura, una instrucción de computación y otra de escritura del resultado. La versión a la que se ha aplicado el pragma Pipeline tiene un II de bucle de 1 y su latencia final se reduce a la mitad.

Origen: *Introducción a HLS por Xilinx* (https://www.xilinx.com/html_docs/xilinx2021_1/vitis_doc/introductionvitis_hls.html)

5.2.1.2 Pragma Pipeline

Permite reducir el II de una función o bucle como se puede ver en la Figura 5.15 al habilitar la ejecución paralela de múltiples instrucciones. Esta funcionalidad opera de una forma parecida al pipeline a nivel de instrucción de una CPU, solo que más poderoso ya que se trata de un pipeline de instrucciones al completo, no de las etapas que las componen.

El pragma Pipeline es una directiva clave en el diseño HLS ya que permite reducir el II de bucles y funciones a 1, de forma que su ejecución esté altamente optimizada y únicamente limitada por la velocidad de reloj del propio chip. Cuando se trata con imágenes, reducir el II a 1 se traduce a la capacidad del diseño para procesar uno o varios píxel al completo por cada ciclo de reloj.

Lograr un II 1 es uno de los grandes retos del diseño HLS ya que no siempre se pueden ejecutar todas las instrucciones necesarias en un mismo ciclo de reloj, requiriendo rediseñar la lógica del código o usar un chip FPGA distinto según las necesidades del proyecto.

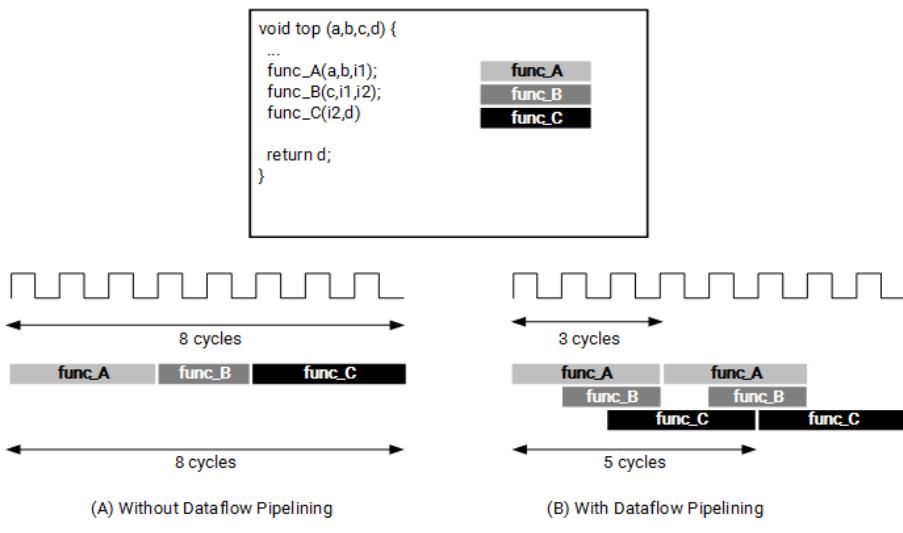


Figura 5.16: Una secuencia de llamadas a funciones con dependencias entre ellas. La versión en la que se ha aplicado el pragma Dataflow permite la ejecución paralela de estas sin esperar a que cada una termine de ejecutarse al completo.

Origen: *Introducción a HLS por Xilinx* (https://www.xilinx.com/html_docs/xilinx2021_1/vitis_doc/introductionvitis_hls.html)

5.2.1.3 Pragma Dataflow

La síntesis HLS suele crear un módulo lógico aislado por cada función que se define en el código, y estos módulos son posteriormente conectados usando buses, memorias y otros componentes intermedios. Esto quiere decir que cuando se está procesando datos en uno de estos módulos los demás módulos pueden estar procesando otros datos en paralelo en vez de quedarse inactivos.

El pragma Dataflow permite definir secciones de código en el que solo residen llamadas a funciones de forma que, al sintetizarse, estas puedan comenzar su funcionamiento antes de que las funciones de las que dependen terminen de procesar el conjunto de datos al completo, logrando así un funcionamiento paralelo y altamente optimizado, como se ve en la figura 5.16.

En el campo del procesamiento de imágenes, el pragma Dataflow permite que una misma imagen se este procesando en varias funciones al mismo tiempo, cada una trabajando en píxeles distintos. Esto evita la necesidad de esperar a que cada función procese la imagen por completo para comenzar con el siguiente paso de procesamiento.

5.2.1.4 Pragma Unroll

Al igual que en los programas para CPU, la lógica de control de bucles requiere de cálculos adicionales que a menudo pueden ser reducidos usando la técnica de desenrollado de bucles.

El desenrollado de bucles hace que se ejecuten múltiples iteraciones de un bucle en una misma iteración, de forma que se reduce el número de iteraciones necesarias para completar el bucle a cambio de aumentar la complejidad de estas. Si bien en una CPU esta mejora simplemente reduce el número de comparaciones y saltos usados, una FPGA aprovecha esta técnica mucho más permitiendo la ejecución paralela de todas las iteraciones desenrolladas.

Por ejemplo, dado un bucle en una FPGA que realiza 1024 iteraciones, es posible desenrollarlo en iteraciones de 16 pasos (factor 16) de forma que sólo se necesite hacer 64 iteraciones en las que se procesan 16 pasos en paralelo cada vez, siempre que estos pasos no dependan unos de otros. En el campo de la visión artificial esto es, potencialmente, equivalente a procesar 16 píxeles por cada iteración del bucle, siempre que sea posible acceder a todos estos píxeles en un mismo ciclo.

Cabe notar que desenrollar un bucle en una FPGA implica un aumento de los recursos hardware usados, por lo que hay que buscar un balance entre la cantidad de pasos desenrollados y el consumo de recursos hardware (LUTs, DSPs y FPs), que vendrá determinado por la aceleración que se necesite y el área disponible en el chip FPGA siendo utilizado.

5.2.1.5 Pragma Array Partition

Los bloques de memoria tienen un número limitado de puertos de acceso, normalmente 2, que se usan cada vez que se lee o escribe un dato a la memoria. Esta cantidad reducida de puertos limita el nivel de paralelismo que es capaz de alcanzar un programa ya que no se podrían hacer suficientes accesos a memoria paralelos para satisfacer la demanda de datos de un bucle desenrollado.

Para solucionar este problema, el pragma Array Partition permite dividir una memoria en un conjunto de memorias más pequeñas, de forma que se aumente el número de puertos disponibles para acceder a los datos guardados en estas.

Por ejemplo, dado un bucle desenrollado con factor 16 que debe leer un dato de una memoria de entrada, procesarlo y guardarla en una memoria de salida, alcanzar un II de 1 requiere hacer 16 accesos a cada memoria por ciclo de reloj. Para satisfacer la demanda de este bucle, se deben de disponer 16 memorias en total cada una con

2 puertos de acceso, es decir, se debe usar un pragma Array Partition con factor 8 en la memoria de entrada y en la memoria de salida.

5.2.1.6 Pragma Dependence

Las dependencias entre datos son uno de los principales motivos por los que no se puede desenrollar o siquiera alcanzar un II de 1 en muchos bucles y funciones. A menudo estas dependencias son reales y pueden no tener una solución clara, pero también es posible que estas dependencias sean falsas ya que el sintetizador HLS ha sido demasiado conservador como para obviarlas.

Algunas dependencias reales son:

- **Dependencia *loop-independent*:** Acceder a un mismo elemento en una memoria múltiples veces, por ejemplo, lectura y escritura a una misma posición de memoria.
- **Dependencia *loop-carried*:** Acceder a un mismo elemento en distintas iteraciones del bucle, limitando el desenrollado de este.

En el caso de que se detecte que una dependencia reportada por el sintetizador sea falsa, por ejemplo, se reporta un potencial doble acceso en una misma iteración a posición de memoria cuando es imposible dada la lógica abstracta del programa, el desarrollador puede usar el pragma Dependence para ignorarla, mejorando el rendimiento del programa. Antes de ignorar una dependencia es necesario asegurarse de que esta es falsa, ya que en caso contrario se obtendría un diseño RTL erróneo.

5.2.1.7 Pragma Interface

Cada módulo lógico generado por HLS tiene una serie de entradas, salidas y vías de control que lo gobiernan. La configuración de estas vías junto a los protocolos usados son definidos usando el pragma Interface.

El desarrollador solo necesita definir las interfaces de la función *Top-Level* del diseño, ya que HLS es capaz de deducir las interfaces necesarias para todos los demás módulos del diseño según la lógica usada y los datos a transmitir.

5.2.1.8 Pragma Loop Tripcount

HLS es capaz de calcular el número de ciclos necesarios para ejecutar un programa en un chip FPGA, siempre que se conozca el número de iteraciones que realiza cada bucle.

El pragma Loop Tripcount permite definir un número mínimo, medio y máximo de iteraciones que podría realizar un bucle, de forma que la herramienta HLS pueda hacer estimaciones útiles del rendimiento del código. Este pragma es puramente informativo y no influye en el diseño RTL final, pero su uso es altamente recomendado.

5.2.2 Proceso de adaptación

Cuando se desarrolla código C++ con High Level Synthesis, debido a las limitaciones descritas en la Sección 1.2.2.3.4, se debe usar un estilo C++ clásico que difiere en gran medida del estilo moderno, y es por esto, junto a las restricciones hardware que impone una FPGA, que se deberá rediseñar la librería en gran medida.

Se usará el IDE ofrecido por Vitis HLS para hacer esta adaptación, el cual integra una gran variedad de herramientas entre compiladores, sintetizadores, simuladores y librerías a usar.

Cada una de las secciones presentadas a continuación tiene una copia del código resultante tras las modificaciones especificadas en el repositorio del proyecto, de forma que se puede ver paso a paso como cambia el código original.

5.2.2.1 Eliminación de llamadas al sistema y código no usado

Ya que es necesario rediseñar una gran parte de la librería, es conveniente analizar primero qué funciones eliminar para reducir este esfuerzo. En la librería se tenían algunas funciones que aplicaban filtros de mediana, detección de bordes Canny y Sobel, de escalado de resolución y convolución genérica cuadrada que servían como herramientas para futuras utilidades o cambios. Todas estas funciones se eliminarán para agilizar el proceso de adaptación.

Una vez se tiene sólo el código necesario, se analizan las llamadas al sistema que se realizan dentro de la librería, pudiéndose ignorar las llamadas usadas por el programa piloto ya que se ejecutará en la CPU del chip Zynq y no en la FPGA que no las soporta.

La única llamada al sistema presente en la librería es el lanzamiento de hilos en la clase *Motion_detector* para lograr una ejecución paralela y asíncrona. Para eliminar el uso de hilos es necesario rediseñar la clase para que siga un modelo síncrono.

Tras estos cambios se obtiene una clase muy reducida con sólo un constructor y una función de detección de movimiento síncrona, en la que ya no es necesario el uso de colas y mecanismo de sincronización de hilos al ser estrictamente bloqueante.

5.2.2.2 Eliminación de incompatibilidades

Una vez se eliminan las llamadas al sistema es necesario eliminar el resto de incompatibilidades como son la STL, los punteros genéricos y los punteros a funciones.

La clase *Image*, basada en la estructura dinámica `std::vector` de la STL, debe ser rediseñada para usar memoria no dinámica como requiere HLS para poder sintetizar. Hacer este cambio de forma inmediata es problemático ya que en el caso de que se quiera simular el código usando un compilador de C++ utilitario, la memoria usada por la imagen será guardada en la pila del sistema y se producirá un desbordamiento de pila. Es decir, sería posible sintetizar el programa pero no se podría probar usando el compilador C++ de pruebas, por lo que es necesario utilizar la variable del preprocesador `_SYNTHESIS_` declarada por Vitis HLS cuando se sintetiza, de forma que se puede comprobar su existencia para saber si debemos declarar la memoria de forma dinámica o no de forma automática como se muestra a continuación:

```

1 #ifndef _SYNTHESIS_
2     uint16_t *data_ = new uint16_t[1024]; // Memoria dinamica para simulacion
3 #else
4     uint16_t data_[1024] = {};// Memoria estatica para sintesis
5 #endif

```

Este cambio a su vez hace que se deba especificar la resolución del vídeo en tiempo de compilación para poder definir el tamaño de la memoria a usar, por lo que hay que eliminar la flexibilidad que tenía la librería original y forzar que sólo se puedan procesar vídeos Full HD.

Al igual que se ha hecho con la clase *Image*, es necesario eliminar todas las estructuras de la STL de los algoritmos de procesamiento de imágenes. Cabe notar que cuando se eliminan todas las estructuras de la STL, la función de histéresis deja de ser eficiente en uso de memoria, por lo que se sustituirá el umbral doble por un umbral simple. Además, se eliminará la generalidad de tipos de píxeles que ofrecía la librería original para obtener un diseño específico y eficiente en la funciones de esta.

Por otra parte, cuando se programa en C++ de bajo nivel, especialmente para hardware, es recomendable utilizar tipos de datos que especifiquen su tamaño de

forma explícita, por ejemplo, usar `uint8_t` en vez de `unsigned char`, o `int64_t` en vez de `long long int` para evitar ambigüedad.

Debido a que el paso de funciones por parámetro no está permitido en HLS, es necesario cambiar todos los algoritmos de convolución para que usen implementaciones específicas del kernel a aplicar en vez de convoluciones genéricas. Esto, a su vez, aumenta la eficiencia del código ya que se puede aprovechar las características de cada proceso con mayor facilidad.

5.2.2.3 Síntesis inicial

Una vez se tiene un código compatible con HLS es necesario estructurarlo de la forma recomendada por Vitis, es decir, se debe crear un *testbench* que compruebe su funcionamiento y reunir el código a acelerar en una misma carpeta *src*. A partir de este punto ya no es necesario usar CMake, sino que se releva la tarea de construir el código a la herramienta de compilación y síntesis de Vitis HLS.

HLS requiere definir una función *Top-Level* a acelerar, pero sólo permite seleccionar una función específica, no una clase al completo, por lo que se deberá sustituir la clase *Motion_detector* por una función independiente que almacene todos los datos relevantes, incluyendo la imagen de referencia, en la memoria global del programa.

Una vez se tiene la estructura del proyecto requerida, se procede a crear un nuevo proyecto en Vitis HLS definiendo un fichero TCL que indica los ficheros de código fuente, el fichero testbench, la configuración a utilizar y la función Top-Level. Un elemento de importancia a configurar es la selección del chip FPGA que se va a utilizar, en este caso, el *XC7Z020ICLG400C -1 Speed*.

Una vez se sintetiza el código obtenemos las primeras métricas de utilización de recursos hardware en el chip, como se muestra en la Figura 5.17. Estas métricas son favorables respecto al uso de DSPs, FFs y LUTs, pero el uso de BRAM es demasiado alto y no es soportado por el chip. Este uso excesivo de memoria se da al guardar las imágenes intermedias en memoria estática, por lo que se deberá encontrar una forma de evitar estos búferes intermedios.

5.2.2.4 Streaming de imágenes

Para reducir la cantidad de memoria utilizada es necesario reducir la cantidad de imágenes que se guardan en memoria. HLS ofrece la librería de *streams*, los cuales funcionan como colas FIFO de profundidad limitada, por defecto 2, que permiten transmitir imágenes píxel a píxel, o en paquetes de píxeles.

Name	BRAM_18K	DSP	FF	LUT	URAM
DSP	-	1	-	-	-
Expression	-	-	0	4682	-
FIFO	-	-	-	-	-
Instance	512	30	6081	15475	-
Memory	828	-	0	0	-
Multiplexer	-	-	-	2040	-
Register	-	-	4009	480	-
Total	1340	31	10090	22677	0
Available	280	220	106400	53200	0
Utilization (%)	478	14	9	42	0

Figura 5.17: Las métricas de utilización de recursos en el chip FPGA en la primera síntesis. Se puede ver como se utiliza un 478 % de la memoria disponible en el chip.

Ya que no está soportado en HLS hacer un *hls::stream* un miembro de una clase, deberemos eliminar la clase *Image* y sustituir cada una de sus apariciones por un stream que soporte el mismo tipo de píxel que la imagen que se estaba usando en su lugar.

Usar streams viene con el inconveniente de que su acceso es estrictamente secuencial, por lo que se deberá procesar cada imagen píxel a píxel sin ser posible leer sus vecinos de forma arbitraria. Esto fuerza un rediseño de la gran mayoría de algoritmos de procesamiento de imágenes siendo utilizados.

5.2.2.4.1 Rediseño de la reducción de resolución

Para reducir la resolución de una imagen por un factor F dado es necesario saber el valor medio de una región de píxeles $F \times F$, es decir, se debe conocer la suma de los valores de los píxeles que la componen, pero no es posible acceder a una región al completo de forma directa ya que no se dispone de acceso aleatorio a la imagen.

Dado que se conoce la resolución del vídeo de entrada, se selecciona el factor 4 para la reducción de resolución implementada, evitando así regiones marginales que necesitan de lógica adicional para procesar (Las regiones rojas, verdes y amarillas en la Figura 5.13).

Para procesar este algoritmo se necesita un búfer en el que se almacenan las sumas parciales de los píxeles que se van recibiendo por el stream de entrada, de forma que cuando se han recibido 4 filas completas se conocerá el valor de cada píxel en una de las filas de la imagen de salida.

Este búfer será un vector de tamaño igual a la anchura de la imagen de salida del algoritmo, por lo que se ha logrado reducir la cantidad de memoria usada de 2

imágenes completas, una Full HD y otra reducida a 480×270 p, a una sola fila de 270 elementos.

5.2.2.4.2 Rediseño del desenfoque Gaussiano

El desenfoque Gaussiano se compone de dos algoritmos principales, la convolución vertical y la horizontal.

La convolución vertical usada en el proyecto toma un vector Gaussiano de longitud 5, por lo que requiere acceder a los 2 píxeles arriba del píxel actual y a los 2 píxeles por debajo de este. Ya que no es posible acceder a estos directamente, deberemos en su lugar almacenar los valores que se van recibiendo en un búfer temporal.

El búfer usado será de 5 filas de longitud 270, de forma que se van completando las filas según se reciben los píxeles que las componen. Una vez se conoce el valor de 5 filas, es posible calcular el píxel desenfocado correspondiente, y este puede ser escrito al stream de salida.

Por otro lado, la convolución horizontal requiere de un pequeño búfer de 5 elementos, ya que sólo es necesario acceder a los píxeles a la izquierda y derecha del píxel actual.

Usando este método es posible, sin perjudicar el rendimiento del algoritmo, almacenar solamente 5 filas de un fotograma y 5 píxeles en vez de 480 filas completas como se hacía en el algoritmo original.

5.2.2.4.3 Rediseño de la interpolación, sustracción de fondo y umbralización

Los algoritmos de interpolación, sustracción de fondo y umbralización procesan la imágenes píxel a píxel sin acceder a los vecinos de este, por lo que es posible simplemente sustituir las imágenes de entrada y salida por streams, sin cambiar la lógica implementada.

Por otra parte, es imposible reducir el espacio de memoria usado por la imagen de referencia ya que esta debe ser integralmente persistente en memoria.

5.2.2.4.4 Rediseño de la dilatación

La dilatación sigue un proceso casi idéntico al desenfoque Gaussiano, ya que usa una convolución vertical y horizontal de longitud 3 en vez de 5, por lo que usaría 3 filas de 270 píxeles y 3 píxeles adicionales en memoria.

Name	BRAM_18K	DSP	FF	LUT	URAM
DSP	-	3	-	-	-
Expression	-	-	0	1599	-
FIFO	-	-	693	476	-
Instance	12	11	6425	8211	-
Memory	129	-	279	3	-
Multiplexer	-	-	-	845	-
Register	-	-	1691	288	-
Total	141	14	9088	11422	0
Available	280	220	106400	53200	0
Utilization (%)	50	6	8	21	0

Figura 5.18: Las métricas de utilización de recursos en el chip FPGA en tras usar hls::stream.

5.2.2.4.5 Rediseño de la detección de contornos

La detección de contornos es uno de los algoritmos más problemáticos de adaptar ya que su funcionamiento se basa en el seguimiento de los bordes de cada componente conexa encontrada, dependiendo de un acceso aleatorio a píxeles en cualquier punto de la imagen.

En vez de adaptar este algoritmo, es más sencillo cambiar el algoritmo completamente por un algoritmo de detección de componentes conexas de una sola pasada, como se describe en la Sección 5.1.1.7, el cual depende de una estructura de árboles en vez de un búfer de píxeles.

5.2.2.5 Mejoras de eficiencia

Tras adaptar todos los algoritmos para que usen hls::stream en vez de imágenes completas se obtiene el uso de recursos mostrado en la Figura 5.18, pero actualmente no es posible saber la latencia e II del diseño ya que HLS no es capaz de deducir el número de iteraciones que tardarán los bucles usados. Se puede ver en la Figura 5.19 como faltan múltiples campos por definir.

Para permitir a HLS calcular la distintas métricas de rendimiento es necesario usar el pragma Loop Tripcount en aquellos bucles que no tengan cotas definidas en tiempo de compilación.

Los únicos bucles que no están acotados son aquellos usados para mantener la estructura de componentes conexas en la detección de contornos, ya que no se puede saber en tiempo de compilación la altura de los distintos árboles que se van a inferir internamente por esta estructura, o la cantidad de contornos que se van a encontrar

Modules && Loops	Issue Type	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	Ff	LUT	URAM
detect_motion	II Violation	-	-	-	-	-	-	no	141	14	9088	11478	0
gaussian_blur_vline	II Violation	-	391258	3.913e6	-	391258	-	no	4	3	1962	2673	0
connected_components	II Violation	-	-	-	-	-	-	no	6	1	1321	2415	0
dilation	-	260234	2.602e6	-	260234	-	-	no	2	0	371	892	0
VITIS_LOOP_145_1	II Violation	-	2080080	2.080e7	1926	-	1080	no	-	-	-	-	-
VITIS_LOOP_99_1	-	148230	1.482e6	549	-	270	-	no	-	-	-	-	-
VITIS_LOOP_179_1_VITIS_LOOP_181_2	-	129619	1.296e6	21	1	129600	yes	-	-	-	-	-	-
VITIS_LOOP_193_1_VITIS_LOOP_195_2	-	129602	1.296e6	4	1	129600	yes	-	-	-	-	-	-
VITIS_LOOP_204_1	-	129601	1.296e6	3	1	129600	yes	-	-	-	-	-	-
VITIS_LOOP_26_1_VITIS_LOOP_28_2	-	129600	1.296e6	2	1	129600	yes	-	-	-	-	-	-

Figura 5.19: Las métricas de rendimiento en el chip FPGA en tras usar hls::stream. No se muestra la latencia e II ya que HLS no puede deducir el número de iteraciones de uno o varios bucles.

Modules && Loops	Issue Type	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	Ff	LUT	URAM
detect_motion	II Violation	-	21286654	2.130e8	-	21286655	-	no	141	14	9088	11478	0
gaussian_blur_vline	II Violation	-	391258	3.913e6	-	391258	-	no	4	3	1962	2673	0
connected_components	II Violation	-	18018018	1.800e8	-	18018018	-	no	6	1	1321	2415	0
dilation	-	260234	2.602e6	-	260234	-	-	no	2	0	371	892	0
VITIS_LOOP_145_1	II Violation	-	2080080	2.080e7	1926	-	1080	no	-	-	-	-	-
VITIS_LOOP_99_1	-	148230	1.482e6	549	-	270	-	no	-	-	-	-	-
VITIS_LOOP_179_1_VITIS_LOOP_181_2	-	129619	1.296e6	21	1	129600	yes	-	-	-	-	-	-
VITIS_LOOP_193_1_VITIS_LOOP_195_2	-	129602	1.296e6	4	1	129600	yes	-	-	-	-	-	-
VITIS_LOOP_204_1	-	129601	1.296e6	3	1	129600	yes	-	-	-	-	-	-
VITIS_LOOP_26_1_VITIS_LOOP_28_2	-	129600	1.296e6	2	1	129600	yes	-	-	-	-	-	-

Figura 5.20: Las métricas estimadas tras usar el pragma Loop Tripcount. Se muestra el peor caso posible de ejecución en el que se detectan todos los contornos posibles, 1023.

en total, por lo que es necesario estimar los números de iteraciones mínimo, medio y máximo. Una vez se aplica este pragma, se obtienen las métricas mostradas en la Figura 5.20.

La latencia máxima actual es de 21286656 ciclos o 0.213 segundos, es decir, que en un peor caso sólo se obtendrán 5 fotogramas por segundo, siendo demasiado lento para los requisitos del proyecto. Es por este motivo por el que hay que comenzar a usar los pragmas Pipeline y Dataflow para mejorar el rendimiento del programa.

En el programa se tiene una gran cantidad de bucles anidados, por lo que es necesario utilizar el pragma Pipeline de forma inteligente ya que este desenrollará todos los bucles en su alcance, consumiendo una cantidad de recursos extremadamente elevada. Para obtener un II de 1 sin inducir un desenrollado de bucle masivo basta con aplicar el pragma al bucle interior del conjunto anidado y no al bucle externo.

Por otra parte, el pragma Dataflow requiere de una sección de llamadas a funciones sin otras instrucciones que puedan interferir, por lo que se usará en la función Top-Level que simplemente llama a los distintos algoritmos del pipeline en orden.

Una vez se aplica el pragma Dataflow a la función por completo, HLS informará de un problema: Dataflow exige que cualquier función dentro de su área de influencia se pueda ejecutar en paralelo sin conflictos de datos, pero el código contiene una función que modifica la referencia guardara en memoria mientras que otra que la

Modules & Loops	Issue Type	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSR	FF	LUT	URAM
# detect_motion	-	18929082	1.890e8	-	18929076	-	-	dataflow	141	14	9456	11808	0
↳ gaussian_blur	-	18929075	1.890e8	-	18929075	-	-	no	140	14	9116	11240	0
↳ gaussian_blur_filter_vline	II Violation	-	391258	3.913e6	-	391259	-	dataflow	4	8	5694	5210	0
↳ VITIS_LOOP_51_1	-	960	9.600e3	3	2	480	yes	-	-	-	-	-	-
↳ VITIS_LOOP_61_2	-	480	4.800e3	2	1	480	yes	-	-	-	-	-	-
↳ VITIS_LOOP_68_3_VITIS_LOOP_71_4	II Violation	-	385944	3.859e6	28	3	128640	yes	-	-	-	-	-
↳ VITIS_LOOP_84_6	II Violation	-	1925	1.925e4	10	4	480	yes	-	-	-	-	-
↳ VITIS_LOOP_93_8	II Violation	-	1925	1.925e4	10	4	480	yes	-	-	-	-	-
↳ gaussian_blur_filter_hline	-	148231	1.482e6	-	148231	-	-	no	0	5	3096	2276	0
↳ connected_components	II Violation	-	18018018	1.800e8	-	18018018	-	no	6	1	1306	2321	0
↳ p_anonymous_namespace_merge	-	39	390.000	-	39	-	-	no	0	0	423	698	0
↳ VITIS_LOOP_98_1	-	480	4.800e3	1	1	480	yes	-	-	-	-	-	-
↳ VITIS_LOOP_104_2	-	18015480	1.800e8	66724	-	270	no	-	-	-	-	-	-
↳ VITIS_LOOP_61_1	II Violation	-	2052	2.052e4	9	2	1023	yes	-	-	-	-	-
↳ t_pipe_interpolate_reference	-	129621	1.296e6	-	129621	-	-	no	0	5	852	1471	0
↳ dilation	-	130960	1.310e6	-	130952	-	-	dataflow	2	0	482	966	0
↳ reference_subtraction	-	129604	1.296e6	-	129604	-	-	no	0	0	166	325	0
↳ set_reference	-	129602	1.296e6	-	129602	-	-	no	0	0	67	215	0
↳ single_threshold	-	129603	1.296e6	-	129603	-	-	no	0	0	26	125	0
↳ downsample	II Violation	-	2593351	2.593e7	-	2593351	-	no	1	0	239	470	0
↳ VITIS_LOOP_160_1	II Violation	-	2593350	2.593e7	9605	-	270	no	-	-	-	-	-

Figura 5.21: Las métricas estimadas tras usar los pragmas Pipeline y Dataflow. Se puede ver que hay una gran variedad de violaciones de II en varios algoritmos.

lee en paralelo de forma no controlada, por lo que el pragma no se puede aplicar correctamente.

Para solucionar este problema se opta primeramente por crear dos zonas Dataflow distintas, una al principio del pipeline que engloba la reducción de resolución y el desenfoque Gaussiano, y otro al final del pipeline englobando la umbralización, dilatación y detección de componentes conexas, quedando así las funciones de sustracción de fondo y mantenimiento de este fuera del Dataflow y eliminando el conflicto.

Tras aplicar estos pragmas se obtienen las métricas de rendimiento mostradas en la Figura 5.21. Se da una reducción de 250 milisegundos, que sigue sin ser suficiente para alcanzar una ejecución en tiempo real fluida.

El motivo por el que el rendimiento sigue siendo bajo es por las distintas violaciones de II en el programa. Una violación de II ocurre cuando HLS intenta lograr un II objetivo pero fracasa (II de 1 en este caso) debido a distintas dependencias y problemas.

Es necesario analizar una por una cada una de estas violaciones usando la información ofrecida por el sintetizador. Las distintas soluciones son las siguientes:

- **Desenfoque Gaussiano:** No se disponen de suficientes puertos en la BRAM que implementa el búfer auxiliar para satisfacer la demanda del bucle principal. Se soluciona aplicando el pragma Array Partition para aumentar el número de puertos.
- **Reducción de resolución:** Se necesita hacer demasiados accesos al stream de entrada para generar un sólo píxel de salida, por lo que hay un cuello de botella.

connected_components	II&Timing Violation	-0.34	7519339	7.519e7	-	7519339	-	no	6	1	1272	2074	0
p_anonymous_namespace_merge		-	39	390.000	-	39	-	no	0	0	423	698	0
VITIS_LOOP_99_1		-	480	4.800e3	1	1	480	yes	-	-	-	-	-
VITIS_LOOP_104_2_VITIS_LOOP_106_3	Timing Violation	-	7516801	7.517e7	58	-	129600	no	-	-	-	-	-
VITIS_LOOP_40_1		-	4	40.000	2	-	2	no	-	-	-	-	-
VITIS_LOOP_61_1	II Violation	-	2052	2.052e4	9	2	1023	yes	-	-	-	-	-

Figura 5.22: Las métricas estimadas tras resolver las violaciones de II.

Se soluciona introduciendo paquetes de 4 píxeles en el stream de entrada, en vez de píxel a píxel.

- **Componentes conexas:** Se comprueban los vecinos de cada píxel usando una conectividad de 8 vías, que es demasiado compleja de realizar dado el número de comprobaciones necesarias. Se soluciona usando una conectividad de 4 vías, reduciendo el II a 2, que es aceptable y se marcará como objetivo para evitar que siga siendo reportado como un problema.

Tras estos cambios se obtiene una latencia de unos 80 milisegundos en el peor caso (12.4 FPS), como se muestra en la Figura 5.22. Por otra parte, en el mejor de los casos, cuando no hay movimiento alguno en la imagen, se obtienen 190 fotogramas por segundo. Teniendo en cuenta que el peor caso es extremadamente raro, siendo el número medio de contornos detectados cuando hay movimiento en una escena natural alrededor de 30, se puede considerar que la librería ya cumple con el requisito de rendimiento impuesto en el proyecto al alcanzar un procesado de más de 100 FPS en condiciones normales.

El último problema a solucionar es la violación de tiempo reportada. Una violación de tiempo ocurre cuando hay un ciclo en el que se está usando un poco más del tiempo de ejecución del disponible, por lo que su ejecución puede resultar en errores inesperados al sintetizarse en hardware, por ejemplo, cuando aumenta la temperatura del dispositivo. HLS da por defecto un margen conservativo al tiempo de ejecución de cada ciclo, de forma que esté asegurado su funcionamiento incluso en condiciones de temperatura desfavorables [3]. Dado que la violación de tiempo es de 0.34 nanosegundos por encima del límite, esta puede ser ignorada sin problemas ya que es un porcentaje muy pequeño del margen de 2.7 nanosegundos que se le ha dado al diseño. Una vez se reduce el margen de tiempo a 2.3 nanosegundos, la violación de tiempo desaparece.

5.2.2.6 Simulación RTL y optimizaciones finales

El último problema de prioridad a resolver es el tamaño de los hls::streams usados. Un stream debe tener una profundidad suficiente para almacenar los datos que se van introduciendo en él mientras un proceso consumidor lo vacía, es decir, si se

introducen datos y se consumen a exactamente la misma velocidad, el stream puede tener una profundidad de 1 sin problemas, pero en el caso de que se introduzcan elementos en ráfaga y se consuman periódicamente, el stream deberá tener una profundidad suficiente para almacenar una ráfaga completa.

El problema en el código desarrollado es que los streams que comunican las funciones de sustracción de fondo e interpolación deben tener un tamaño suficiente como para almacenar una imagen completa, ya que no están dentro de la zona Dataflow como se indicó en la sección anterior.

En el caso de que se ignore la necesidad de streams de profundidad suficiente, las simulaciones RTL se congelarán indefinidamente ya que no se podrán insertar todos los píxeles necesarios al stream y, en el caso de que no se ignore y se use la profundidad correcta requerida, los streams serán demasiado grandes como para ser viables, ya que consumirían toda la BRAM del chip.

Para solucionar este problema es necesario modificar el código de forma que todo el pipeline de procesamiento esté englobado por una única zona Dataflow. Esto se consigue fusionando las funciones de sustracción e interpolación de fondo en una misma función, de forma que desaparece la dependencia que existía entre estas dos y, por consecuencia, se puedan configurar todos los streams usados a la profundidad ideal 1.

Por otro lado, HLS ofrece la librería de precisión arbitraria (*Arbitrary Precision*) que permite el uso de tipos de datos exactamente tan precisos y grandes como sean necesarios, en vez de estar restringidos a 8, 16, 32 y 64 bits como aquellos comúnmente usados en una CPU. Un ejemplo de su uso es la reducción del tamaño de los píxeles booleanos que sólo necesitan almacenar o un 0 o un 1, pasando de ocupar 8 bits (`uint8_t`) a 1 bit (`ap_uint<1>`), de forma que se usa 8 veces menos memoria y 8 veces menos líneas de buses para operar con este tipo de imágenes.

Una vez se aplican estos tipos de datos a todos los datos en el código, incluyendo índices, píxeles y búferes, se obtienen las métricas de uso de recursos mostradas en la Figura 5.23, en la que se ve una reducción del uso de cientos LUTs y FFs.

Una vez se han realizado estos cambios, se tiene un módulo IP sintetizable y simulable en Vivado XSim, que permite la detección de movimiento en un vídeo Full HD en tiempo real de forma fluida y usando una cantidad muy reducida de elementos hardware del chip.

Cabe notar que dado el bajo uso de FFs, LUTs y otros componentes, sería posible aumentar aún más la velocidad del diseño creando streams de filas de píxeles completas en vez de píxeles únicos. Una fila completa puede ser procesada de forma altamente paralela usando el pragma `Unroll`, por lo que el rendimiento aumentaría

Name	BRAM_18K	DSP	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	12	-
FIFO	-	-	99	68	-
Instance	148	14	10752	12078	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	18	-
Register	-	-	2	-	-
Total	148	14	10853	12176	0
Available	280	220	106400	53200	0
Utilization (%)	52	6	10	22	0

(a) Uso de recursos antes de aplicar
precisión arbitraria.

Name	BRAM_18K	DSP	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	2	-
FIFO	-	-	495	340	-
Instance	146	19	9462	10993	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	-	-
Register	-	-	-	-	-
Total	146	19	9957	11335	0
Available	280	220	106400	53200	0
Utilization (%)	52	8	9	21	0

(b) Uso de recursos tras aplicar
precisión arbitraria.

Figura 5.23: El efecto del uso de tipos de datos de precisión arbitraria.

notablemente, pero no se ha implementado en este proyecto ya que no se necesita aún más velocidad.

Capítulo 6

Pruebas

En esta sección se explican las pruebas llevadas a cabo para la comprobación del funcionamiento del sistema, según se indica en los requisitos funcionales y no funcionales de este.

6.1 Estrategia

El desarrollo de las pruebas del sistema se ha llevado a cabo en las siguientes fases:

1. **Pruebas de unidad e integración de la librería para CPU:** Se comprueba el funcionamiento de los algoritmos utilizados y su corrección mediante pruebas de unidad e integración.
2. **Pruebas de uso correcto de memoria:** Se verifica que toda la memoria usada al detectar movimiento es manejada adecuadamente y liberada cuando deja de ser necesaria usando Valgrind.
3. **Pruebas de funcionamiento de la librería para CPU:** Se valida el funcionamiento de la librería mediante su uso con vídeos reales capturados con cámaras Full HD.
4. **Pruebas de funcionamiento del módulo IP para FPGA:** Se comprueba el funcionamiento del módulo IP creado usando HLS haciendo uso de la funcionalidad *testbench* que ofrece Vitis.

6.2 Entorno de pruebas

Las pruebas se han realizado usando vídeos capturados por cámaras Full HD, entre ellas diversas webcams y una Raspberry Camera en distintos entornos y lugares.

En estos vídeos la cámara es completamente estática y se enfoca a un área de una habitación iluminada y sin movimiento. Posteriormente se inicia movimiento de forma esporádica colocando y retirando objetos y andando a través de la escena.

6.3 Niveles de prueba

6.3.1 Pruebas unitarias

Se comprueba el funcionamiento de cada algoritmo utilizado con distintas entradas y salidas, que hacen especial hincapié en los casos borde ambiguos, verificando que no hayan entradas que causen un comportamiento errático.

Estas pruebas, codificadas en C++ y activables de forma opcional al compilar la librería, aseguran la corrección del código componente a componente pero no verifican la calidad de los resultados obtenidos por la librería.

En la Figura 6.1 se pueden ver los resultados de pasar las pruebas de unidad a la librería.

6.3.2 Pruebas de integración

Se unifican todas las funciones y algoritmos codificados en un solo pipeline y se verifica su funcionamiento con una serie de fotogramas sintéticos.

Estos fotogramas sintéticos son creados a mano y se comprueba que se reporta movimiento en los momentos esperados, sin errores o comportamientos no deseados.

Por último, se valida que este pipeline no tiene fugas de memoria usando Valgrind, como se muestra en la figura 6.2.

6.3.3 Pruebas del sistema

Se comprueba el funcionamiento de la librería en CPU como se muestra en la figura 6.3 mediante el uso de vídeos Full HD y un programa piloto que guarda a disco

```

Testing module image_utils...
[PASS] : vline_convolution
[PASS] : hline_convolution
[PASS] : gaussian_blur_filter
[PASS] : image_subtraction
[PASS] : double_threshold
[PASS] : single_threshold
[PASS] : hysteresis
[PASS] : image_interpolation
[PASS] : dilation
[PASS] : downsample
Finished tests for module image_utils.

Testing module motion_detector...
[PASS] : Image constructor
[PASS] : Image operator=
[PASS] : Image operator[]
[PASS] : Image getter and setter
[PASS] : Motion_detector constructor
[PASS] : Motion_detector getter and setter
[PASS] : Motion_detector detect_motion
[PASS] : uchar to BW
Finished tests for module motion_detector.

Testing module contour_detector...
[PASS] : contour_detection
Finished tests for module contour_detector.

Finished all module tests.

```

Figura 6.1: Resultados de las pruebas de unidad de la librería.

```

==4266==
==4266== HEAP SUMMARY:
==4266==   in use at exit: 206,050 bytes in 2,247 blocks
==4266==   total heap usage: 1,662,385 allocs, 1,660,138 frees, 23,905,017,746 bytes allocated
==4266==
==4266== LEAK SUMMARY:
==4266==   definitely lost: 61,440 bytes in 1,280 blocks
==4266==   indirectly lost: 0 bytes in 0 blocks
==4266==   possibly lost: 1,352 bytes in 18 blocks
==4266==   still reachable: 143,258 bytes in 949 blocks
==4266==           of which reachable via heuristic:
==4266==               newarray          : 1,536 bytes in 16 blocks
==4266==               suppressed: 0 bytes in 0 blocks
==4266== Rerun with --leak-check=full to see details of leaked memory
==4266==
==4266== For lists of detected and suppressed errors, rerun with: -s
==4266== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)

```

Figura 6.2: Pruebas de uso de memoria usando Valgrind.

```

Got results for 9824. | Processing time: 97 milliseconds.
Got results for 9856. | Processing time: 121 milliseconds.
Got results for 9888. | Processing time: 104 milliseconds.
Got results for 9920. | Processing time: 116 milliseconds.
Got results for 9952. | Processing time: 96 milliseconds.
Got results for 9984. | Processing time: 111 milliseconds.
Got results for 10016. | Processing time: 96 milliseconds.
Got results for 10064. | Processing time: 106 milliseconds.
Got results for 10096. | Processing time: 117 milliseconds.
Got results for 10128. | Processing time: 94 milliseconds.
Got results for 10160. | Processing time: 111 milliseconds.
Motion detected. Recording to "./.../data/out/motion_0.mp4"
[REC] Got results for 10192. | Processing time: 113 milliseconds.
[REC] Got results for 10224. | Processing time: 130 milliseconds.
[REC] Got results for 10256. | Processing time: 119 milliseconds.
[REC] Got results for 10288. | Processing time: 124 milliseconds.
[REC] Got results for 10320. | Processing time: 110 milliseconds.
[REC] Got results for 10352. | Processing time: 104 milliseconds.
[REC] Got results for 10399. | Processing time: 112 milliseconds.
[REC] Got results for 10432. | Processing time: 93 milliseconds.
[REC] Got results for 10464. | Processing time: 100 milliseconds.
[REC] Got results for 10496. | Processing time: 100 milliseconds.
[REC] Got results for 10528. | Processing time: 106 milliseconds.
[REC] Got results for 10560. | Processing time: 94 milliseconds.
[REC] Got results for 10592. | Processing time: 96 milliseconds.
[REC] Got results for 10624. | Processing time: 119 milliseconds.

```

(a) Momento en el que se detecta movimiento en el vídeo de entrada.



(b) La escena en la que se produce movimiento con cuadros que marcan los contornos detectados.

Figura 6.3: Prueba de detección de movimiento en un vídeo Full HD.

segmentos del vídeo en los que se encuentra movimiento, con anotaciones de color verde en los lugares en los que se ha detectado el movimiento.

Quedando verificada la calidad de los resultados de la librería para CPU, se verifica el funcionamiento del módulo IP para FPGA haciendo uso de la funcionalidad *testbench* que ofrece Vitis.

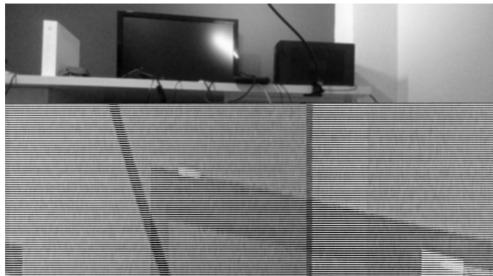
Los testbench son una herramienta ampliamente utilizada en el desarrollo para FPGA, ya sea usando HLS o HDL, que permite al desarrollador definir un programa que controla el funcionamiento del módulo IP, verificando sus entradas y salidas siguiendo una serie de reglas establecidas.

En este proyecto se adapta uno de los programas piloto a un fichero testbench, de forma que se prueba el módulo IP creado usando un fichero .mp4 guardado en disco en el que se conoce cuando y donde se debe detectar movimiento.

6.4 Corrección de errores

En todas las fases del desarrollo se han ido solucionando una gran variedad de errores detectados con las pruebas anteriormente listadas. Algunos de estos son:

- **Algoritmos con errores:** Al implementarse a mano cada uno de los algoritmos usados, aparecen problemas que deben ser detectados usando pruebas de unidad y solucionados uno a uno.
- **Problemas de memoria:** Se solucionan diversos errores de fugas de memoria detectados usando Valgrind, y se eliminan diversos problemas de desborda-



(a) Corrupción de la pila al guardar imágenes demasiado grandes en ella.

```
////////////////////////////////////////////////////////////////
// Inter-Transaction Progress: Completed Transaction / Total Transaction
// Intra-Transaction Progress: Measured Latency / Latency Estimation * 100%
//
// RTL Simulation : "Inter-Transaction Progress" ["Intra-Transaction Progress"] @ "Simulation Time"
////////////////////////////////////////////////////////////////
// RTL Simulation : 0 / 15 [0.00%] @ "125000"
// RTL Simulation : 0 / 15 [142.46%] @ "100000145000"
```

(b) La simulación RTL congelada durante millones de ciclos de reloj.

Figura 6.4: Algunos errores encontrados durante las pruebas del código desarrollado.

miento de pila durante la adaptación a HLS que surgían al usar imágenes de alta resolución que requieren usar memoria dinámica dado su tamaño.

- **Problemas de calidad de resultados:** Los resultados iniciales obtenidos eran erráticos y de baja calidad. Se corrigieron ajustando los distintos valores como son los umbrales y factor de interpolación para lograr mejores resultados.
- **Problemas de velocidad en CPU:** Los algoritmos usados no tenían un rendimiento suficientemente alto para procesar vídeo de forma fluida, evitando que la librería cumpla los distintos requisitos establecidos, por lo que se modificaron con diversas optimizaciones y mejoras.
- **Problemas de uso de recursos y rendimiento en FPGA:** Al portar el diseño a FPGA de forma directa se obtiene un diseño con un uso excesivo de recursos y con un rendimiento desfavorable, que debe ser modificado para alcanzar los objetivos del proyecto.
- **Bloqueos en la simulación RTL:** La simulación RTL que verifica el funcionamiento del núcleo IP desarrollado se bloquea y tiene comportamientos erráticos si el código es incorrecto. Se debe estudiar el código para detectar las áreas conflictivas y rediseñarlas, obteniendo una lógica más adecuada para su síntesis a hardware.

En la Figura 6.4 se pueden ver algunos de los errores mencionados.

Capítulo 7

Resultados

En esta sección se presentan los resultados obtenidos con el sistema desarrollado, realizando a su vez un análisis de las ventajas de usar hardware dedicado sobre software puro.

7.1 Resultados CPU

La librería se prueba con distintos niveles de paralelismo (número de hilos concurrentes) y con distintos factores de reducción de resolución, todos aplicados al mismo vídeo Full HD. Estas pruebas de rendimiento se realizan tanto en una Raspberry Pi 4B que tiene un procesador Cortex-A72 de 4 núcleos físicos y lógicos, como en un ordenador portátil con una CPU Intel Core i7-7700HQ de 4 núcleos físicos y 8 lógicos.

En la Tabla 7.1 se muestran las métricas de rendimiento obtenidas en el procesador portátil tanto para la librería base como para la librería optimizada con las mejoras indicadas en la Sección 5.1.2.5. Se puede observar como es posible obtener una ejecución en tiempo real usando la librería optimizada haciendo uso de todos los procesadores lógicos del sistema.

Si bien se ha logrado detectar movimiento en tiempo real de forma fluida a 30 FPS, la CPU del computador alcanza temperaturas extremadamente elevadas y tiene un uso cercano al 100 % de su capacidad, por lo que no se considera una solución sostenible para procesar vídeo durante horas.

En una Raspberry Pi 4B, que tiene un sistema de refrigeración y una capacidad de procesamiento inferior al portátil, se obtienen las métricas mostradas en la Tabla

Hilos	1	4	8	16
Reducción				
Original	0.8 FPS 18 % CPU	2.2 FPS 58 % CPU	2.8 FPS 83 % CPU	2.9 FPS 93 % CPU
4	10.4 FPS 25 % CPU	22.9 FPS 60 % CPU	23.4 FPS 70 % CPU	24.8 FPS 93 % CPU

(a) Métricas de rendimiento base.

Hilos	1	4	8	16
Reducción				
Original	1.3 FPS 16 % CPU	4.4 FPS 54 % CPU	5.4 FPS 75 % CPU	5.2 FPS 74 % CPU
4	15.8 FPS 27 % CPU	25.7 FPS 39 % CPU	30.1 FPS 89 % CPU	29.9 FPS 93 % CPU

(b) Métricas de rendimiento optimizadas.

Tabla 7.1: Rendimiento en FPS en CPU Intel Core i7-7700HQ para la librería base y la librería optimizada.

7.2, donde se puede observar que en ninguna de las configuraciones se logra obtener 30 FPS, siendo además imposible aumentar el factor de reducción de resolución ya que los resultados comienzan a ser erráticos.

Cabe notar que cuando la Raspberry Pi ejecuta la librería optimizada no se alcanza nunca un uso completo de la CPU. El motivo de este problema es el cuello de botella que supone acceder al vídeo guardado en la memoria masiva, que es magnitudes más lenta que el disco SSD utilizado en el portátil, siendo imposible conseguir fotogramas a procesar lo suficientemente rápido como para satisfacer la capacidad de procesamiento de la librería.

7.2 Resultados FPGA

El módulo IP optimizado para FPGA utiliza los recursos mostrados en la Tabla 7.3 y tiene una latencia completa de 77.8 milisegundos en el peor de los casos (12.9 FPS) y 5.2 milisegundos en el mejor de ellos (192 FPS). El mejor de los casos se da cuando no hay movimiento alguno en el vídeo de entrada (0 componentes conexas), mientras que el peor caso se da cuando hay un movimiento extremadamente complejo en la escena con 1023 componentes conexas a analizar. De forma empírica se ha determinado que cuando se detecta movimiento en entornos controlados, como puede ser andar por delante de la cámara o mover objetos, la media de componentes

Hilos Reducción \	1	4	8	16
Original	0.1 FPS 27 % CPU	0.2 FPS 96 % CPU	0.2 FPS 97 % CPU	0.2 FPS 99 % CPU
4	4.1 FPS 40 % CPU	5.8 FPS 92 % CPU	6.2 FPS 92 % CPU	5.7 FPS 93 % CPU

(a) Métricas de rendimiento base.

Hilos Reducción \	1	4	8	16
Original	0.7 FPS 27 % CPU	2.7 FPS 96 % CPU	2.8 FPS 98 % CPU	2.7 FPS 99 % CPU
4	7.9 FPS 57 % CPU	9.6 FPS 70 % CPU	9.7 FPS 70 % CPU	9.6 FPS 69 % CPU

(b) Métricas de rendimiento optimizadas.

Tabla 7.2: Rendimiento en FPS en CPU ARM Cortex-A72 para la librería base y la librería optimizada.

Uso	BRAM 18K	DSP	FF	LUT
Cantidad	146	19	9957	11335
Porcentaje	52 %	8 %	9 %	21 %

Tabla 7.3: Métricas de uso de recursos finales para el módulo IP creado.

conexas detectadas está alrededor de 30, por lo que se determina que este núcleo IP satisface el requerimiento de detección de movimiento en tiempo real de forma fluida.

Por último, cabe notar que el diseño obtenido tiene una frecuencia máxima de 133 Mhz, por lo que una amplia cantidad de chips FPGA, incluyendo la FPGA objetivo XC7Z020-1CLG400C que tiene una frecuencia máxima de 667MHz, pueden usarlo a máximo rendimiento.

Parte III

Epílogo

7.3 Conclusiones

En este último capítulo del TFG se exponen las ideas finales del proyecto, la forma en la que se han logrado los distintos objetivos propuestos y algunos objetivos futuros a llevar a cabo.

7.3.1 Objetivos

Se han logrado los objetivos especificados en la Sección 1.3:

- Se ha satisfecho el objetivo 1 del proyecto codificando una librería en C++ que detecta movimiento en vídeo Full HD en tiempo real para CPU. Si bien es cierto que no se ha logrado cumplir este objetivo para una Raspberry Pi 4B dada su baja potencia computacional y cuellos de botella, se ha cumplido para procesadores de sobremesa actuales.
- Se han creado dos versiones de la librería para CPU, una librería base altamente mantenible, genérica y reutilizable, y otra especializada al caso de uso para lograr obtener el máximo rendimiento posible. Habitualmente basta con codificar un programa genérico de alta calidad ya que la CPU no se va a someter a tanta carga, pero en este caso se está utilizando para procesar cantidades masivas de datos, por lo que ha sido necesario hacer esta separación de versiones para satisfacer el objetivo 1 del proyecto.
- Se ha creado una documentación detallada de los procesos implementados en la librería y su adaptación a HLS, logrando el objetivo 2 del proyecto.
- Se ha simulado el diseño RTL para el chip FPGA Zynq XC7Z020-1CLG400C y se ha verificado su funcionamiento, satisfaciendo el objetivo 3.
- Se han analizado los casos de uso y las ventajas e inconvenientes de usar una CPU y una FPGA, además de la realización de una comparativa del rendimiento de cada una de las soluciones desarrolladas en el proyecto, lográndose el objetivo 4.

7.3.2 Aprendizaje

Durante el desarrollo de este proyecto se han desarrollado las siguientes habilidades y capacidades:

7.3. CONCLUSIONES

- Se ha aprendido el funcionamiento y las bases matemáticas de los distintos algoritmos de visión artificial y de procesamiento de imágenes, así como diferentes métodos de mejorar su rendimiento y uso de recursos.
- Se ha aprendido las bases de la programación en C++ moderno, su filosofía y diversos patrones comunes encontrados, además de las metodologías actuales de documentación con Doxygen y construcción de proyectos con CMake.
- Se han estudiado distintos métodos de aceleración hardware usando FPGAs, GPUs y TPUs, además de posibles combinaciones de estos, junto con las plataformas Xilinx Vitis e Intel OneAPI para el uso de todas estas de forma simple y efectiva.
- Se ha conocido la metodología de desarrollo HLS para FPGA, que hace posible transformar código en C++ a un diseño RTL de forma rápida, además de su historia, bases y alcance.
- Se ha visto el gran rendimiento que ofrece usar hardware dedicado sobre software en CPU, además de las diversas dificultades y retos que conlleva su desarrollo y optimización.
- Se ha aprendido a gestionar el desarrollo de un proyecto usando una metodología SCRUM y a trabajar de forma autónoma, buscando información en distintas fuentes para lograr la realización de los objetivos marcados.

Este proyecto ha sido una gran herramienta para expandir mis habilidades en el mundo de la electrónica y las FPGAs, y ha sido una gran motivación para seguir aprendiendo sobre visión artificial, óptica e inteligencia artificial aplicada a este campo.

7.3.3 Mejoras del proyecto

Las librerías desarrolladas requieren de buena iluminación para funcionar y, a menudo, su comportamiento no es óptimo dada la simplicidad del pipeline utilizado. Es posible mejorar este pipeline utilizando algoritmos más sofisticados que a su vez mejoren el rendimiento del programa.

El núcleo IP creado usa streams de píxeles individuales de forma que en cada ciclo de reloj sólo se puede procesar un solo píxel de una imagen, si bien con un uso de recursos hardware muy bajo. Es posible mejorar los algoritmos para que procesen filas completas de píxeles en vez de píxeles individuales, siendo posible procesar decenas de estos por ciclo de reloj siempre que el chip FPGA sea suficientemente grande para soportar el tamaño del diseño RTL generado.

7.3. CONCLUSIONES

Por último, el núcleo IP generado solo ha sido simulado y verificado usando Vivado Xsim, por lo que una mejora al proyecto sería realizar una integración de este módulo IP en una FPGA real para comprobar de forma final su funcionamiento correcto y la calidad de los resultados.

7.3.4 Trabajo futuro

En este proyecto se explora el uso de una CPU independiente frente a una FPGA en el campo de la visión artificial, pero no se hace una implementación en GPU por lo que no es posible determinar qué acelerador sería el que ofrece mayor rendimiento en el caso de uso dado.

Como trabajo futuro es posible hacer una implementación de la librería para GPU y realizar una comparativa de las 3 tecnología disponibles, además de la posibilidad de su uso conjunto para alcanzar aún más rendimiento.

Parte IV

Anexos

7.4 Manual de usuario

7.4.1 Introducción

En esta sección se explica cómo instalar, probar y modificar las distintas versiones de la librería, tanto para CPU como para FPGA.

7.4.2 Requisitos software

7.4.2.1 Librerías para CPU

Para la instalación y uso de las librerías para CPU se requiere de un sistema con una distribución de GNU/Linux como sistema operativo, siendo necesario Raspberry Pi OS (anteriormente conocido como Raspbian) para su uso en Raspberry Pi usando una Pi Camera.

Es técnicamente posible usar las librerías en sistemas Windows, pero la instalación automática del código no está soportada y su instalación manual no se contempla en este manual.

7.4.2.2 Librerías para FPGA

Se requiere de la suite Vitis HLS 2020.2 o superior instalada en un sistema Windows o GNU/Linux. Para la instalación de esta suite se deberá acceder al enlace <https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/vitis.html> en el que se encuentran las diversas versiones del software ordenadas en orden cronológico.

7.4.3 Requisitos hardware

Se listan las herramientas o componentes hardware necesarios mínimos para la instalación y uso de las librerías.

- Herramientas para las librerías para CPU.
 - Computador utilitario o Raspberry Pi 4B.
 - Cámara USB Full HD o Raspberry Camera 2.
- Herramientas para las librerías para FPGA.

- Computador utilitario.

7.4.4 Archivos

El repositorio del proyecto tiene la estructura mostrada a continuación, en la que se indican los directorios con el prefijo (D) y los ficheros con el prefijo (F):

- (D) **cpu** - *Librerías C++ para CPU y programas piloto de ejemplo.*
 - (D) **libraries** - *Librerías instalables de detección de movimiento.*
 - (D) **base** - *Librería base que busca una alta calidad de código.*
 - (D) **fast** - *Librería especializada que busca un alto rendimiento.*
 - (D) **pilot_programs** - *Programas que hacen uso de las librerías en distintas formas*
 - (D) **save_to_disk** - *Guarda segmentos de vídeo cuando se detecta movimiento.*
 - (D) **show_on_screen** - *Muestra el movimiento en pantalla constantemente.*
- (D) **hls** - *Librerías para FPGA basadas en C++ HLS.*
 - (D) **final** - *Librería HLS optimizada en velocidad y área.*
 - (D) **steps_followed** - *Librerías de adaptación parcial a HLS y no finales.*
 - (D) **0_no_system_calls_and_unused_code**
 - (D) **1_fix_incompatibilities**
 - (D) **2_synthetizing**
 - (D) **3_streaming_images**
 - (D) **4_performance**
 - (D) **5 rtl_simulation**
- (D) **example_results** - *Vídeo de entrada de ejemplo y ejemplos de algunos resultados obtenidos.*
- (F) **LICENSE** - *Licencia aplicada sobre el software distribuido.*
- (F) **README.md** - *Fichero Mark-Down con una breve descripción del proyecto para GitHub.*

7.4.5 Instrucciones de prueba

7.4.5.1 Librerías CPU

Se asume un sistema GNU/Linux con una distribución Raspberry OS, en la que el usuario es *md* y el nombre del dispositivo es *pi*.

La configuración e instalación de paquetes iniciales se hace con los siguientes comandos, siendo algunos de estos necesarios para compilar los programas piloto descritos más adelante:

```
1 md@pi:~ $ sudo apt update -y && sudo apt upgrade -y
2 md@pi:~ $ sudo apt install -y build-essential cmake pkg-config libgtk-3-dev
    ↳ libavcodec-dev libavformat-dev libswscale-dev libv4l-dev libxvidcore-dev
    ↳ libx264-dev libjpeg-dev libpng-dev libtiff-dev gfortran openexr libatlas-base
    ↳ -dev python3-dev python3-numpy libtbb2 libtbb-dev libdc1394-22-dev libavutil-
    ↳ dev libavresample-dev libSDL1.2-dev libjack-jackd2-dev libmp3lame-dev
    ↳ libopencore-amrnb-dev libopencore-amrwb-dev libtheora-dev libva-dev libvdpau-
    ↳ dev libvorbis-dev libX11-dev libXfixes-dev texi2html zlib1g-dev libvpx-dev
    ↳ libgtk2.0-dev libqt4-dev libqt4-opengl-dev libtiff5-dev libjasper-dev
    ↳ libpng12-dev v4l-utils x264 yasm
3 md@pi:~ $ sudo apt install -y clang --install-suggests
```

Se asume que el repositorio de código está localizado en el directorio home del usuario *md* bajo en nombre *motdet*, siendo su ruta absoluta */home/md/motdet*.

Se procede a compilar e instalar la versión base de la librería para CPU con tests de unidad. En caso de que no se quiera compilar los tests, se debe cambiar el parámetro ‘true’ a ‘false’ en el comando *cmake* que aparece a continuación:

```
1 md@pi:~ $ cd ~/motdet/cpu/libraries/base
2 md@pi:~/motdet/cpu/libraries/base $ mkdir build && cd build
3 md@pi:~/motdet/cpu/libraries/base $ export CXX=/usr/bin/clang++
4 md@pi:~/motdet/cpu/libraries/base/build $ cmake .. -DBUILD_TEST=true
5 md@pi:~/motdet/cpu/libraries/base/build $ sudo make install -j4
```

Para instalar la librería *fast* en su lugar, que es la recomendada para obtener resultados con mejor rendimiento, simplemente hay que repetir los pasos accediendo primeramente al directorio *fast* en vez del *base*.

Una vez la orden *make install* sea satisfactoria, la librería estará instalada en el sistema como un paquete software y puede ser usada en cualquier programa que lo solicite. Para comprobar su funcionamiento e instalación correcta, se lanzan los siguientes comandos:

```
1 md@pi:~/motdet/cpu/libraries/base/build $ pkg-config --modversion motion_detector
2 md@pi:~/motdet/cpu/libraries/base/build $ ./test_exec
```

7.4. MANUAL DE USUARIO

Instalar una nueva librería con el mismo nombre de paquete sustituirá a la antigua instalada sin conflictos, por lo que es posible probar ambas versiones del código.

7.4.5.2 Programas Piloto CPU

Ambos programas pilotos hacen uso de OpenCV para el control de cámaras o lectura de ficheros .mp4, por lo que se instalará con los siguientes comandos:

```
1 md@pi:~ $ mkdir ~/opencv && cd ~/opencv
2 md@pi:~/opencv $ sudo apt install -y git
3 md@pi:~/opencv $ git clone https://github.com/opencv/opencv.git
4 md@pi:~/opencv $ git clone https://github.com/opencv/opencv_contrib.git
5 md@pi:~/opencv $ cd opencv && mkdir build && cd build
6 md@pi:~/opencv/opencv/build $ cmake .. -D CMAKE_BUILD_TYPE=RELEASE -D
    ↪ CMAKE_INSTALL_PREFIX=/usr/local -D INSTALL_C_EXAMPLES=OFF -D
    ↪ INSTALL_PYTHON_EXAMPLES=OFF -D OPENCV_GENERATE_PKGCONFIG=ON -D
    ↪ OPENCV_EXTRA_MODULES_PATH=~/opencv/opencv_contrib/modules -D BUILD_EXAMPLES=
    ↪ OFF ..
7 md@pi:~/opencv/opencv/build $ sudo make install -j4
```

Una vez se ha instalado OpenCV, es posible compilar cualquiera de los programas piloto de ejemplo. En este caso se compila el programa ‘save_to_disk’, asumiendo que la librería de detección de movimiento ha sido instalada como se indica en la sección anterior:

```
1 md@pi:~ $ cd ~/motdet/cpu/pilot_programs/save_to_disk
2 md@pi:~/motdet/cpu/pilot_programs/save_to_disk $ mkdir build && cd build
3 md@pi:~/motdet/cpu/pilot_programs/save_to_disk $ export CXX=/usr/bin/clang++
4 md@pi:~/motdet/cpu/pilot_programs/save_to_disk/build $ cmake ..
5 md@pi:~/motdet/cpu/pilot_programs/save_to_disk/build $ make -j4
```

Tras estos comandos, se podrán ver las opciones del programa lanzando el comando:

```
1 md@pi:~/motdet/cpu/pilot_programs/save_to_disk/build $ ./motion_detector_driver
```

El programa no ejecutará su función principal ya que no se le han enviado los parámetros mínimos necesarios para su funcionamiento, siendo estos:

- **input:** El origen de los fotogramas, si se selecciona la palabra ‘camera’, el programa accederá a la cámara de sistema, en caso contrario, se intentará acceder a un archivo .mp4 en la ruta especificada.
- **output:** Ruta a un directorio en el que guardar segmentos de vídeo en los que se han detectado movimiento.
- **threads:** Parámetro opcional de selección del número de hilos a usar por la librería.

- **downscale factor:** Parámetro opcional que reduce la resolución de vídeo original por un factor, aumenta la velocidad pero reduce la fidelidad.
- **display stats:** Parámetro opcional booleano (1 o 0) que indica si se debe imprimir en pantalla información de los fotogramas procesados.

En el caso de que se quiera usar el vídeo de ejemplo incluido en el repositorio de código usando todos los núcleos de la Raspberry Pi, se podría usar el siguiente comando:

```
1 md@pi:~/motdet/cpu/pilot_programs/save_to_disk/build $ ./motion_detector_driver ~/  
→ motdet/example_results/in_test_motion.mp4 ~/motdet/example_results 4 4 1
```

En el caso de que se quiera usar una Raspberry Camera, se deberá primero habilitar en las opciones con el comando *sudo raspi-config*, y navegando a la pestaña *Interface options/Camera*, en la que se deberá pulsar Enter para activarla. Una webcam USB no necesita configuración adicional.

El programa piloto *show_on_screen* tiene las mismas opciones excepto ‘output’, ya que por defecto muestra todos los fotogramas por pantalla y no los guarda a disco.

7.4.5.3 Librerías HLS

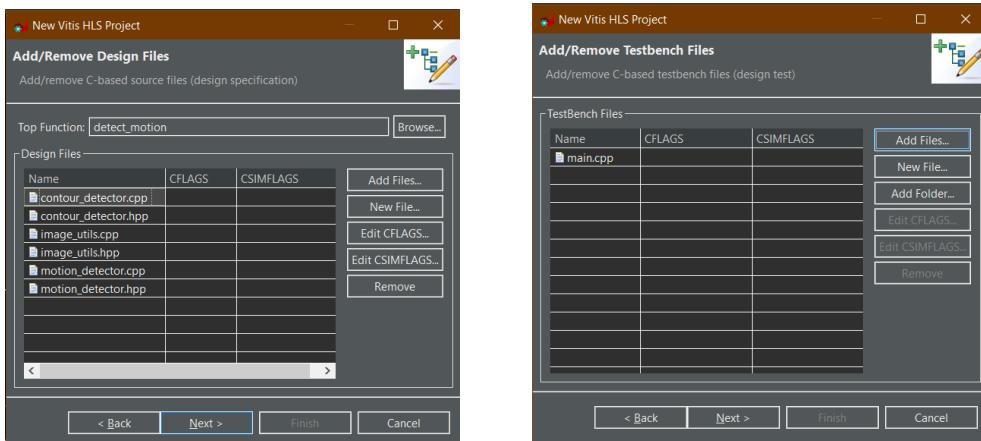
Una vez Vitis HLS está instalado en el sistema a utilizar, se puede optar por usar su interfaz gráfica para sintetizar y modificar la librería del repositorio, o la versión de línea de comandos.

La versión con interfaz gráfica requiere la creación de un nuevo proyecto vacío en el que se insertan los archivos de código fuente cuando se solicite, así como el fichero testbench a usar, como se muestra en la Figura 7.1. Una vez se seleccionan todos los ficheros, será necesario seleccionar el chip FPGA objetivo para el que se va a sintetizar el código, siendo escogido el chip xc7z020clg400-1.

Una vez el proyecto está creado, basta con pulsar el botón de síntesis para producir un diseño RTL con el código HLS introducido. En el caso que se quiera probar su funcionamiento usando el testbench incluido, será necesario vincular OpenCV en el proyecto accediendo a la pestaña *Project/Simulation*, donde se deberá incluir las siguientes banderas dentro del formulario *Linker Flags*, las cuales asumen que se ha instalado OpenCV en la ruta *E:/OpenCV*:

```
1 -LE:/OpenCV4/x64/mingw/lib -lopencv_videoio452 -lopencv_video452 -  
→ lopencv_imgproc452 -lopencv_imgcodecs452 -lopencv_core452
```

7.5. APÉNDICE



(a) Selección de los ficheros de código fuente y la función Top-Level a acelerar.

(b) Selección del fichero testbench.

Figura 7.1: Configuración inicial de un proyecto HLS usando la interfaz gráfica.

Por otra parte, se ofrecen ficheros .tcl en cada directorio HLS del repositorio que automáticamente vinculará los ficheros necesarios y ejecutará la síntesis del código. Para usar estos ficheros, basta con lanzar el siguiente comando en una consola:

```
1 vitis_hls -f tcl_script.tcl
```

Nótese que estos ficheros .tcl intentarán vincular OpenCV en el directorio especificado internamente, por lo que deberán ser editados para apuntar a la ruta de instalación de OpenCV en el sistema.

7.5 Apéndice

En este apéndice se muestra el código de la librería para CPU base y el código HLS optimizado. Para acceder a todo el código creado, incluyendo la librería de CPU optimizada, los pasos intermedios de adaptación a HLS, los otros programas piloto, ficheros de configuración y documentación Doxygen, se deberá consultar el repositorio original del proyecto.

7.5.1 Código librería CPU base

7.5.1.1 CMakeLists.txt

7.5. APÉNDICE

```
1 cmake_minimum_required(VERSION 3.9.0)
2 project(motion_detector VERSION 1.0.0 DESCRIPTION "No dependency motion detector
   ↪ library in C++")
3
4 set(DEFAULT_BUILD_TYPE "Release")
5
6 set(SOURCE_FILES src/motion-detector.cpp src/contour-detector.cpp)
7
8 add_library(${PROJECT_NAME} SHARED ${SOURCE_FILES})
9
10 # Set library version
11 set_target_properties(${PROJECT_NAME} PROPERTIES VERSION ${PROJECT_VERSION})
12
13 # Set version of the generated so files (For example: libmotion-detector.so.1.0.0.)
14 set_target_properties(${PROJECT_NAME} PROPERTIES SOVERSION 1)
15
16 # Avoid having to include with relative paths
17 target_include_directories(${PROJECT_NAME}
18   PUBLIC
19     $<INSTALL_INTERFACE:include>
20     $<BUILD_INTERFACE:${CMAKE_CURRENT_SOURCE_DIR}/include>
21   PRIVATE
22     ${CMAKE_CURRENT_SOURCE_DIR}/src
23 )
24
25 # Set the file with the public API for the library
26 set_target_properties(${PROJECT_NAME} PROPERTIES PUBLIC_HEADER include/
   ↪ motion-detector.hpp)
27
28 # Set compiler flags. Tell it to treat warnings as errors, pedantic and c++11
29 target_compile_options(${PROJECT_NAME} PRIVATE -Werror -pedantic -Wno-narrowing)
30 target_compile_features(${PROJECT_NAME} PRIVATE cxx_std_17)
31
32 target_link_libraries (${PROJECT_NAME} LINK_PUBLIC pthread)
33
34 if(BUILD_TEST)
35   set(TEST_FILES test/test_motion_detector.cpp test/test_contour_detector.cpp
      ↪ test/test_image_utils.cpp)
36   set(CMAKE_RUNTIME_OUTPUT_DIRECTORY ${CMAKE_BINARY_DIR})
37
38 # Compile the test sources while linking to the main library
39 add_library(test_lib ${TEST_FILES})
40 target_link_libraries(test_lib LINK_PUBLIC ${PROJECT_NAME})
41
42 target_include_directories(test_lib
43   PUBLIC ${CMAKE_CURRENT_SOURCE_DIR}/include>
44   PRIVATE ${CMAKE_CURRENT_SOURCE_DIR}/src
45 )
46 target_compile_options(test_lib PRIVATE -Werror -pedantic)
47 target_compile_features(test_lib PRIVATE cxx_std_17)
48
49 # Compile the main test executable while linking to the test library
50 add_executable(test_exec test/test_main.cpp)
51 target_link_libraries(test_exec LINK_PUBLIC test_lib)
52
53 target_include_directories(test_exec
54   PUBLIC ${CMAKE_CURRENT_SOURCE_DIR}/include>
55   PRIVATE ${CMAKE_CURRENT_SOURCE_DIR}/src
```

7.5. APÉNDICE

```
56      )
57      target_compile_options(test_exec PRIVATE -Werror -pedantic)
58      target_compile_features(test_exec PRIVATE cxx_std_17)
59  endif()
60
61 if (UNIX)
62   # To install a library in linux it is required to install both the .so files in
63   # Once we do this, the library can be used (shared) by any other project run on
64   # To make this even more convenient to use, we will also create a package so
65   # you can do -lmotion_detector when using g++
66
67   # Define GNU standard installation directories
68   include(GNUInstallDirs)
69
70   # Install libs and includes
71   install(TARGETS ${PROJECT_NAME} LIBRARY DESTINATION ${CMAKE_INSTALL_LIBDIR}
72           # PUBLIC_HEADER DESTINATION ${CMAKE_INSTALL_INCLUDEDIR})
73
74   # Create package so that it can included easily by other projects. This
75   # requires a file named "motion_detector.pc.in" to exist
76   # All the package info will be stored in there.
77   configure_file(motion_detector.pc.in motion_detector.pc @ONLY)
78   install(FILES ${CMAKE_BINARY_DIR}/motion_detector.pc DESTINATION ${CMAKE_INSTALL_DATAROOTDIR}/pkgconfig)
79 else()
80   message(STATUS ">>> Not UNIX/Linux OS detected, installing motion_detector
81           # automatically is not contemplated for this OS, please install the
82           # generated files as needed.")
83 endif()
```

7.5.1.2 motion_detector.hpp

```
1 #ifndef _MOTDET_MOTION_DETECTOR_HPP_
2 #define _MOTDET_MOTION_DETECTOR_HPP_
3
4 #include <iostream>
5 #include <cstddef>
6 #include <vector>
7 #include <array>
8 #include <deque>
9 #include <fstream>
10 #include <mutex>
11 #include <condition_variable>
12 #include <thread>
13
14 namespace motdet
15 {
16
17   // Class definitions
18
19   /**
20    * @brief Represents an image (or video frame) as a flat structure.
21    * @tparam T Type of pixel in the image. Cannot be bool, use unsigned char to
22    # store boolean values.
```

7.5. APÉNDICE

```
22     */
23     template <typename T> class Image
24     {
25     public:
26         /**
27          * @brief Construct a new Image object with the given width and height.
28          *        → Will fill all the pixels with the default value fill_value.
29          * @param width Length of each row. >0.
30          * @param height Row count. >0.
31          * @param fill_value Value to use as default. Use {} for default
32          *        → initializer.
33          */
34     Image(const std::size_t width, const std::size_t height, const T fill_value
35           → ):
36         w_(width),
37         h_(height),
38         total_(width*height)
39     {
40         if(w_ == 0) throw std::invalid_argument("ERROR Constructor: width must
41             → be >0.");
42         if(h_ == 0) throw std::invalid_argument("ERROR Constructor: height must
43             → be >0.");
44         data_.resize(total_, fill_value);
45     };
46
47         /**
48          * @brief Construct a new Image object from a given data vector. Each
49          *        → element of the vector represents a pixel.
50          * @param init_data Vector to copy
51          * @param width Length of each row in the inputted image. >0.
52          */
53     Image(const std::vector<T> &init_data, const std::size_t width):
54         w_(width),
55         h_(init_data.size()),
56         total_(init_data.size())
57     {
58         if(w_ == 0) throw std::invalid_argument("ERROR Constructor: width must
59             → be >0.");
60         h_ /= width;
61         if(w_*h_ != init_data.size()) throw std::invalid_argument("ERROR
62             → Constructor: Invalid width for this vector length.");
63         data_ = init_data;
64     };
65
66     Image() = default;
67     Image(const Image &other) = default;
68     Image(Image &&other) = default;
69
70     // Operator Overload
71
72     Image& operator=(const Image &other) = default;
73     Image& operator=(Image &&other) = default;
74
75         /**
76          * @brief Returns an immutable T element located at idx.
77          * @param idx Index to access.
78          * @return Element located at idx.
79          */
80
```

7.5. APÉNDICE

```
72     inline const T& operator [](const std::size_t idx) const { return data_[idx
    ↪ ]; };
73
74     /**
75      * @brief Returns a mutable T element located at idx.
76      * @param idx Index to access.
77      * @return Element located at idx.
78      */
79     inline T& operator [](const std::size_t idx) { return data_[idx]; };
80
81     // Getters and Setters
82
83     /**
84      * @brief Get the width
85      * @return std::size_t
86      */
87     inline std::size_t get_width() const { return w_; };
88
89     /**
90      * @brief Get the height
91      * @return std::size_t
92      */
93     inline std::size_t get_height() const { return h_; };
94
95     /**
96      * @brief Get the total pixels
97      * @return std::size_t
98      */
99     inline std::size_t get_total() const { return total_; };
100
101    /**
102     * @brief Get the internal data vector.
103     * @return const std::vector<T>&
104     */
105    inline const std::vector<T>& get_data() const { return data_; }
106
107    /**
108     * @brief Set a new value for the internal data vector.
109     * @param data Data vector, must be of the same type as the current one.
110     * @param width Length of each row in the image.
111     * @throw invalid_argument if width == 0 or the given width is not
    ↪ compatible with the given data vector.
112     */
113    inline void set_data(const std::vector<T>& data, const std::size_t width)
114    {
115        if(width == 0) throw std::invalid_argument("ERROR Constructor: width
    ↪ must be >0.");
116        std::size_t height = data.size()/width;
117        if(width*height != data.size()) throw std::invalid_argument("ERROR
    ↪ Constructor: Invalid width for this vector length.");
118
119        w_ = width;
120        h_ = height;
121        total_ = data.size();
122        data_ = data;
123    }
124
125    /**
```

7.5. APÉNDICE

```
126     * @brief Set a new value for the internal data vector. But now for rvalues
127     *      ↪ !
128     * @param data Data vector, must be of the same type as the current one.
129     * @param width Length of each row in the image.
130     * @throw invalid_argument if width == 0 or the given width is not
131     *      ↪ compatible with the given data vector.
132     */
133     inline void set_data(std::vector<T>&& data, const std::size_t width)
134     {
135         if(width == 0) throw std::invalid_argument("ERROR Constructor: width
136             ↪ must be >0.");
137         std::size_t height = data.size()/width;
138         if(width*height != data.size()) throw std::invalid_argument("ERROR
139             ↪ Constructor: Invalid width for this vector length.");
140         w_ = width;
141         h_ = height;
142         total_ = data.size();
143         data_ = std::move(data);
144     }
145
146     private:
147         std::size_t w_, h_, total_;
148         std::vector<T> data_;
149     };
150
151 /**
152 * @brief Represents a bounding box on an image.
153 */
154 struct Contour
155 {
156     Contour(std::size_t bb_tlx, std::size_t bb_tly, std::size_t bb_brx, std
157             ::size_t bb_bry):
158         bb_tlx(bb_tlx),
159         bb_tly(bb_tly),
160         bb_brx(bb_brx),
161         bb_bry(bb_bry)
162     {}
163
164 /**
165 * @brief Represents a bounding box on an image with additional topological
166 *      ↪ information.
167 */
168 struct Extended_contour
169 {
170     int id;                      /**< Unique id of the contour within a
171                                ↪ Contours class
172     int parent;                  /**< id of parent contour, 0 means top-level
173                                ↪ contour (no parent)
174     bool is_hole;                /**< A contour can either surround a hole (0-
175                                ↪ pixels) or be an outline, surrounding 1-pixels */
176     std::size_t n_pixels;        /**< Number of border pixels that compose
177                                ↪ this Contour
178 }
```

7.5. APÉNDICE

```
|173     std::size_t bb_tl_x, bb_tl_y; /*< Top left point of the bounding box of
|174         ↪ the Contour                                     */
|174     std::size_t bb_br_x, bb_br_y; /*< Bottom right point of the bounding box
|174         ↪ of the Contour                                     */
|175 };
|176
|177 /**
|178 * @brief Container for all the relevant info to return as a result when a
|178 *        ↪ frame is checked for movement.
|179 * @details Also contains the data_keep container that points at whatever data
|179 *        ↪ was sent as extra metadata when enqueueing.
|180 */
|181 struct Detection
|182 {
|183     unsigned long long timestamp;           /*< The timestamp of the video
|183     ↪ the motion was detected from */
|184     bool has_detections;                  /*< True if motion has been
|184     ↪ detected                           */
|185     std::vector<Contour> detection_contours; /*< Contour of the detected
|185     ↪ movements                          */
|186     unsigned long long processing_time;    /*< Time it took the frame to be
|186     ↪ processed                         */
|187
|188     std::shared_ptr<void> data_keep;       /*< Will point at NULL if no data_keep
|188     ↪ was sent when enqueueing          */
|189 };
|190
|191 /**
|192 * @brief Detects motion in a given grayscale frame, comparing against previous
|192 *        ↪ frames.
|193 */
|194 class Motion_detector{
|195 public:
|196
|197 /**
|198 * @brief Constructor. Uses fps to set the rate at which the reference
|198 *        ↪ image is adjusted.
|199 * @details Creates a threaded motion detector object.
|200 * It uses a reference image internally to compare to and this reference is
|200 *        ↪ slowly interpolated with new frames to adapt to scenario changes.
|201 * If the update span is too high, precision loss might make the reference
|201 *        ↪ not update, 5 seconds is a good update span.
|202 * @param threads Number of threads to use for frame processing. Min 1.
|202 *        ↪ Reference updating is not 100% deterministic with >1 threads.
|203 * @param queue_size Amount of frames enqueued (waiting or processing). Any
|203 *        ↪ less than "threads" will cripple concurrency.
|204 * Recommended values is threads*2.
|205 * @param downsample_factor Reduce the size of the image for faster
|205 *        ↪ processing. 1 will not downsample. Must be >0.
|206 * @param frame_update_ratio Ratio at which the reference is updated.
|206 *        ↪ Closer to 0 is slower.
|207 * Calculate using the following formula: 1/(fps*seconds). The default used
|207 *        ↪ is fps = 30 and seconds = 5.
|208 * @throw invalid_argument if threads == 0, queue_size == 0,
|208 *        ↪ downsample_factor == 0, width < 10 or height < 10
|209 */
|210 Motion_detector(const std::size_t width, const std::size_t height, const
|210     ↪ std::size_t threads = 1, const std::size_t queue_size = 2, const
|210     ↪ unsigned int downsample_factor = 1, const float frame_update_ratio =
|210     ↪ 0.0067);
```

7.5. APÉNDICE

```
|211 Motion_detector() = delete;
|212 Motion_detector(const Motion_detector &other) = delete;
|213 Motion_detector(Motion_detector &&other) = delete;
|214
|215 ~Motion_detector();
|216
|217 // Operator Overload
|218
|219 Motion_detector& operator=(const Motion_detector &other) = delete;
|220 Motion_detector& operator=(Motion_detector &&other) = delete;
|221
|222 // Getters and Setters
|223
|224 /**
|225 * @brief Get the width
|226 * @return std::size_t
|227 */
|228 inline std::size_t get_width() const { return w_; };
|229
|230 /**
|231 * @brief Get the height
|232 * @return std::size_t
|233 */
|234 inline std::size_t get_height() const { return h_; };
|235
|236 /**
|237 * @brief Get the total pixels
|238 * @return std::size_t
|239 */
|240 inline std::size_t get_total() const { return total_; };
|241
|242 /**
|243 * @brief Get the maximum amount of tasks that can be queued.
|244 * @return std::size_t
|245 */
|246 inline std::size_t get_max_task_queue_size() const { return queue_size_; };
|247
|248 /**
|249 * @brief Get the total amount of tasks stored in the queue, includes both
|250 * frames not processed and those currently being processed.
|251 * @return std::size_t
|252 */
|253 inline std::size_t get_task_queue_size() const { return task_queue_.size();
|254 * };
|255
|256 /**
|257 * @brief Get the total amount of completed tasks. Note a motion detector
|258 * stores an infinite amount of completed tasks, make sure to collect
|259 * them.
|260 * @return std::size_t
|261 */
|262 inline std::size_t get_result_queue_size() const { return result_queue_.
|263 * size(); };
|264
|265 // General Methods
|266
|267 /**
|268 * @brief Will enqueue a frame to be processed by the image detector.
```

7.5. APÉNDICE

```
265     * @param in Grayscale image to be processed wrapped in a smart pointer.  
266         ↪ Transfers ownership.  
267     * @param timestamp_millis Time in milliseconds of the frame being sent in.  
268     * @param blocking If true, will wait for queue to not be full, if false,  
269         ↪ will throw if queue is full.  
270     * @param data_keep Extra info to keep as "metadata" of the inputted frame,  
271         ↪ is returned as is upon result extraction.  
272     * An example usage would be to store the original RGB data of the frame  
273         ↪ here, so that it can be saved to disk later  
274     * if motion is detected.  
275     * @exception runtime_error if the queue is full and blocking is set to  
276         ↪ false. The input frame is lost forever.  
277     * @exception invalid_argument if the timestamp is older than one of the  
278         ↪ already enqueued frames.  
279     * @exception invalid_argument if the new frame has a different resolution  
280         ↪ from the one set in the constructor or is NULL.  
281     */  
282     void enqueue_frame(std::unique_ptr<Image<unsigned short>> in, unsigned long  
283         ↪ long timestamp_millis, bool blocking, std::shared_ptr<void>  
284         ↪ data_keep = {});  
285  
286     /**  
287     * @brief Gets the contours detected in the oldest frame submitted to the  
288         ↪ motion detector.  
289     * @details Will only return successfully if the oldest frame submitted is  
290         ↪ finished, regardless of the completion state of other frames.  
291     * @return detection struct with the detected motion.  
292     * @exception runtime_error if the queue is empty and blocking is set to  
293         ↪ false.  
294     */  
295     Detection get_detection(bool blocking);  
296  
297     /**  
298     * @brief Returns whether the oldest frame submitted has been processed.  
299         ↪ Thread safe method.  
300     * @return true if the oldest contours detected can be extracted safely  
301         ↪ with a non blocking get.  
302     * @return false if the contours are not yet ready to be returned.  
303     */  
304     bool is_frame_ready() const { std::unique_lock<std::mutex> locker(  
305         ↪ results_mutex_); return result_queue_.size() > 0; }  
306  
307 private:  
308  
309     std::size_t w_, h_, total_, downsampled_w_, downsampled_h_;  
310     std::size_t queue_size_;  
311  
312     float frame_update_ratio_;  
313     unsigned int min_cont_area_, downsample_factor_;  
314  
315     bool has_reference_ = false;  
316     Image<unsigned short> reference_;  
317  
318     unsigned long long last_ref_update_time_, last_submitted_time_;  
319  
320     /**  
321     * @brief Container for a motion detection job that is either pending for  
322         ↪ processing, is being processed or is done but not yet submitted.  
323     */
```

7.5. APÉNDICE

```
308     struct Motdet_task_
309     {
310         enum class task_state : unsigned char { waiting, processing, done };
311
312         unsigned long long timestamp, processing_time = 0;
313         task_state state = task_state::waiting;
314
315         std::unique_ptr<Image<unsigned short>> image;
316         std::vector<Contour> result.Contours;
317
318         std::shared_ptr<void> data_Keep;
319     };
320
321     std::size_t threads_;
322     mutable std::mutex reference_mutex_, tasks_mutex_, results_mutex_;
323     std::condition_variable tasks_full_cond_;           /*< Threads waiting for the
324     → task queue to not be empty. */
325     std::condition_variable results_empty_cond_;        /*< Threads waiting for the
326     → oldest frame to be finished. */
327     std::condition_variable noprocessable_frame_; /*< Threads waiting for a
328     → processable frame to be in the tasks queue. */
329
330     std::vector<std::thread> workers_container_;
331     bool keep_workers_alive_ = true;
332
333     void detect_motion_(std::size_t thread_id); /*< Executed by the worker
334     → threads on loop. */
335
336     /**
337      * @brief Checks if there is at least one task in the queue that can be
338      → processed. Does not lock mutex, so do it before calling the method.
339      * @return true if there is a task that can be processed.
340      * @return false if there are no tasks or all are either being processed or
341      → finished.
342      */
343     inline bool processable_frame_check_() noexcept
344     {
345         for(const Motdet_task_ &task : task_queue_) if(task.state ==
346             Motdet_task_::task_state::waiting) return true;
347         return false;
348     }
349
350     // Types
351
352     typedef std::array<unsigned char, 3> rgb_pixel;
353
354     /**
355      * @brief Turns a color image to black and white (grayscale).
356      * @details https://en.wikipedia.org/wiki/Luma\_\(video\) adapted to 16b
357      * @param in RGB image that will be converted. Must be completely initialized.
358      * @param out 16b grayscale image that will be outputted.
```

7.5. APÉNDICE

```
358     */
359     void rgb_to_bw( const Image<rgb_pixel> &in , Image<unsigned short> &out );
360
361     /**
362      * @brief Turns an rgb uchar array to a black and white image (grayscale).
363      * @details https://en.wikipedia.org/wiki/Luma\_\(video\) adapted to 16b
364      * @param in uchar C array that will be converted. Must be of length n_pix*3.
365      * @param n_pix Number of elements present in array "in". An R-G-B triplet in
366      *   "in" counts as 1 element.
367      * @param out 16b grayscale image that will be outputted.
368      */
369     void uchar_to_bw( const unsigned char *in , const std::size_t n_pix , Image<
370     *   > &out );
371
372 } // namespace motdet
373
374 #endif // _MOTDET_MOTION_DETECTOR_HPP_
```

7.5.1.3 motion_detector.cpp

```
1 #include "motion_detector.hpp"
2
3 #include <iostream>
4
5 #include <chrono>
6 #include <stdexcept>
7 #include <cmath>
8
9 #include "image_utils.hpp"
10 #include "contour_detector.hpp"
11
12 namespace motdet
13 {
14     // Motion_detector implementation
15
16     Motion_detector::Motion_detector( const std::size_t width , const std::size_t
17     *   height , const std::size_t threads , const std::size_t queue_size , const
18     *   unsigned int downsample_factor , const float frame_update_ratio ):
19     w_(width),
20     h_(height),
21     total_(width*height),
22     threads_(threads),
23     queue_size_(queue_size),
24     downsample_factor_(downsample_factor),
25     frame_update_ratio_(frame_update_ratio),
26     min_cont_area_(total_*0.002+5),
27     last_submitted_time_(0)
28     {
29         if (threads == 0)           throw std::invalid_argument("ERROR Constructor:
30         *   threads must be at least 1.");
31         if (queue_size == 0)        throw std::invalid_argument("ERROR Constructor:
32         *   queue_size must be at least 1.");
33         if (downsample_factor == 0) throw std::invalid_argument("ERROR Constructor:
34         *   downsample_factor must be at least 1.");
35         if (width < 10)            throw std::invalid_argument("ERROR Constructor:
36         *   width must be at least 10.");
37     }
```

7.5. APÉNDICE

```
31     if (height < 10)      throw std::invalid_argument("ERROR Constructor:  
32         ↪ height must be at least 10.");  
33  
34     // The size requirements for the downsampled image are given by the  
35     // → function in image_utils.  
36     downsampled_w_ = std::ceil((float)w_ / downsample_factor);  
37     downsampled_h_ = std::ceil((float)h_ / downsample_factor);  
38  
39     reference_ = Image<unsigned short>(downsampled_w_, downsampled_h_, {});  
40  
41     // Create all the motion detector slaves.  
42     for(std::size_t i = 0; i < threads; ++i) workers_container_.push_back(std::  
43         ↪ thread(&Motion_detector::detect_motion_, this, i));  
44 }  
45  
46 Motion_detector::~Motion_detector()  
47 {  
48     // The destructor will wait for all threads to die before destroying itself  
49     // → . Leaving a thread unhandled causes error unless it is a daemon.  
50     keep_workers_alive_ = false;  
51     noprocessableframe_.notify_all();  
52     for(std::thread &t : workers_container_) t.join();  
53 }  
54  
55 void Motion_detector::enqueue_frame(std::unique_ptr<Image<unsigned short>> in,  
56     ↪ unsigned long long timestamp_millis, bool blocking, std::shared_ptr<void>  
57     ↪ data_keep)  
58 {  
59     if(in.get() == NULL) throw std::invalid_argument("ERROR Enqueue: The input  
60         ↪ image is NULL.");  
61     if(in->get_total() != total_ || in->get_width() != w_) throw std::  
62         ↪ invalid_argument("ERROR Enqueue: Wrong resolution.");  
63  
64     // Lock mutex for checks  
65     std::unique_lock<std::mutex> locker(tasks_mutex_);  
66     if(timestamp_millis < last_submitted_time_) throw std::invalid_argument("ERROR  
67         ↪ Enqueue: Submitted timestamps must be chronologically ordered.");  
68     last_submitted_time_ = timestamp_millis;  
69  
70     if(blocking)  
71     {  
72         // Blocking mode, sleep until queue is not full.  
73         tasks_full_cond_.wait(locker, [this](){ return task_queue_.size() <  
74             ↪ queue_size_; });  
75     }  
76     else  
77     {  
78         // Non-blocking, check if the queue is full and if it is, throw  
79         // → exception  
80         if(task_queue_.size() < queue_size_) throw std::runtime_error("ERROR  
81             ↪ Enqueue: Queue is full.");  
82     }  
83     // If reached this point, there is a spot available in the queue and the  
84     // → input is valid  
85  
86     Motdet_task_ new_task;  
87     new_task.timestamp = timestamp_millis;  
88 }
```

7.5. APÉNDICE

```
75     new_task.image = std::move(in); // Need to move smart pointer with move to
76     // represent ownership transfer
77     new_task.data_keep = data_keep; // Store the extra metadata but nothing
78     // will be done with it.
79
80     task_queue_.push_back(std::move(new_task));
81     locker.unlock(); // Release the lock and notify one of the worker threads
82     // that there is a new available job to process.
83     noprocessableframe_.notify_one();
84 }
85
86 Detection Motion_detector::get_detection(bool blocking)
87 {
88     // Lock mutex for checks
89     std::unique_lock<std::mutex> locker(results_mutex_);
90
91     if(blocking)
92     {
93         // Blocking mode, sleep until the oldest frame is ready for output.
94         results_empty_cond_.wait(locker, [this](){ return result_queue_.size()
95             // frame is not ready
96             > 0; });
97     }
98     else
99     {
100        // Non-blocking, throw exception if the queue is empty or the oldest
101        // frame is not ready
102        if(result_queue_.size() == 0) throw std::runtime_error("ERROR Enqueue:
103            // No results ready.");
104    }
105    // If reached here, it means there is a valid result to return
106
107    auto result = result_queue_.front();
108    result_queue_.pop_front();
109
110    return result;
111 }
112
113 void Motion_detector::detect_motion_(std::size_t thread_id)
114 {
115     while(keep_workers_alive_)
116     {
117         // Grab a new task to process. It will need to grab the mutex to do so.
118
119         std::unique_lock<std::mutex> tasks_locker(tasks_mutex_);
120         noprocessableframe_.wait(tasks_locker, [this](){ return
121             // processable_frame_check_() || !keep_workers_alive_; });
122         if(!keep_workers_alive_) break;
123
124         // If the thread reached this point, there is at least 1 task that can
125         // be processed in the queue and it got "permission" to process it.
126
127         std::deque<Motdet_task_>::iterator to_process = task_queue_.begin();
128         while (to_process != task_queue_.end() && to_process->state !=
129             // Motdet_task_::task_state::waiting) to_process++;
130         if(to_process == task_queue_.end()) throw std::runtime_error("ERROR
131             // detect_motion_: No processable frame found but expected one.");
132
133         // Successfully got the frame, now mark it so that other threads do not
134         // start processing it as well.
```

7.5. APÉNDICE

```
123     to_process->state = Motdet_task_::task_state::processing;
124     tasks_locker.unlock();
125
126     // It is assured by program logic that this frame will not be edited by
127     // → another thread now. Begin processing.
128     auto processing_time_start = std::chrono::high_resolution_clock::now();
129
130     // Get the input image and downsample it, if needed.
131     Image<unsigned short> in(std::move(*to_process->image.get()));
132     Image<unsigned short> downsampled_in(downscaled_w_, downsampled_h_, 0)
133     → ;
134
135     if(downsampling_factor_ > 1) imgutil::downsample(in, downsampled_in,
136     → downsample_factor_);
137     else downsampled_in = std::move(in);
138
139     // If the motion detector has a reference frame, compare with it to
140     // → check for motion. If not, make a new reference.
141     std::unique_lock<std::mutex> reference_locker(reference_mutex_);
142     if(has_reference_)
143     {
144         reference_locker.unlock(); // Release the reference mutex since we
145         // → will not edit the reference for a while.
146
147         // Define all the container images for intermediate processing
148         // → steps.
149         Image<unsigned short> blur_image(downscaled_w_, downsampled_h_, 0)
150         → , sub_image(downscaled_w_, downsampled_h_, 0), new_ref_image
151         → (downsampled_w_, downsampled_h_, 0);
152         Image<unsigned char> thr_image(downscaled_w_, downsampled_h_, 0),
153         → cnt_image(downscaled_w_, downsampled_h_, 0);
154         Image<int> dil_image(downscaled_w_, downsampled_h_, 0);
155
156         // Blur the image to remove any noise that can result in false
157         // → positives.
158         if(!keep_workers_alive_) break;
159         imgutil::gaussian_blur_filter<unsigned short, unsigned short>(
160             → downsampled_in, blur_image);
161
162         // Subtract the blurred frame with the reference image. Leaving
163         // → only the changes between frames.
164         if(!keep_workers_alive_) break;
165         reference_locker.lock();
166         imgutil::image_subtraction<unsigned short, unsigned short>(
167             → blur_image, reference_, sub_image);
168         reference_locker.unlock();
169
170         // Interpolate the blurred image and the reference frame to obtain
171         // → a new reference.
172         // Interpolation is done so that the reference can adapt to
173         // → changing environment.
174         if(!keep_workers_alive_) break;
175         reference_locker.lock();
176         imgutil::image_interpolation<unsigned short, unsigned short>(
177             → reference_, blur_image, new_ref_image, frame_update_ratio_);
178         reference_ = std::move(new_ref_image);
179         reference_locker.unlock();
180
181         // Threshold the image so that any value below a certain number is
182         // → ignored.
```

7.5. APÉNDICE

```
166 // Using double threshold along with hysteresis for better results
167 //   → over single threshold.
168 if (!keep_workers_alive_) break;
169 imgutil::double_threshold<unsigned short, unsigned char>(sub_image,
170 //   → thr_image, 5000, 22500);
171 imgutil::hysteresis<unsigned char, unsigned char>(thr_image,
172 //   → cnt_image);
173
174 // Dilate the image so that the contours are better defined and
175 //   → with less holes.
176 if (!keep_workers_alive_) break;
177 imgutil::dilation<unsigned char, int>(cnt_image, dil_image);
178
179 // Detect contours in the image. Any contour detected here is "
180 //   → movement".
181 if (!keep_workers_alive_) break;
182 std::vector<Extended_contour> unfiltered_contours;;
183 imgutil::contour_detection(dil_image, unfiltered_contours);
184
185 // Go over the detected contorus and discard any contour that is
186 //   → too small to be relevant.
187 // Also scale the bounding box of the contour back to the original
188 //   → size before downscaling.
189
190 if (!keep_workers_alive_) break;
191 for(Extended_contour &cont : unfiltered_contours)
192 {
193     unsigned int cont_area = (cont.bb_tl_x - cont.bb_br_x) * (cont.
194 //   → bb_tl_y - cont.bb_br_y) * downsample_factor_;
195
196     if(cont_area > min_cont_area_)
197     {
198         to_process->result.Conts.push_back({
199             cont.bb_tl_x * downsample_factor_,
200             cont.bb_tl_y * downsample_factor_,
201             cont.bb_br_x * downsample_factor_,
202             cont.bb_br_y * downsample_factor_
203         });
204     }
205 }
206 else
207 {
208     // There is no reference, create a new one with the current input
209     //   → frame.
210
211     if (!keep_workers_alive_) break;
212     imgutil::gaussian_blur_filter<unsigned short, unsigned short>(
213 //   → downsampled_in, reference_);
214
215     has_reference_ = true;
216     reference_locker.unlock();
217 }
218 // Processing has ended here, the only thing missing is submitting the
219 //   → result.
220 to_process->state = Motdet_task_::task_state::done;
221
222 // Record the time it took the frame to be processed.
223 if (!keep_workers_alive_) break;
```

7.5. APÉNDICE

```
214     auto processing_time_end = std::chrono::high_resolution_clock::now();
215     to_process->processing_time = std::chrono::duration_cast<std::chrono::
216         → milliseconds>(processing_time_end - processing_time_start).count
217         → ();
218
219     // Now that this frame is finished, check from oldest to newest the
220     // → state of the different tasks.
221     // Submit the tasks to the result queue until a task with a state
222     // → different from finished is found.
223     // This assures that the results are outputted in chronological order,
224     // → not processing order.
225     // Note it might cause a thread to not submit any results, since the
226     // → frame it just processed it too new.
227
228     // Lock both the tasks queue and the results queue.
229     tasks_locker.lock();
230     std::unique_lock<std::mutex> results_locker(results_mutex_);
231
232     std::deque<Motdet_task_>::iterator to_submit = task_queue_.begin();
233     while (to_submit != task_queue_.end() && to_submit->state ==
234         → Motdet_task_::task_state::done)
235     {
236         // For each finished task, create a new Detection struct and submit
237         // → it to the results.
238         Detection det;
239         det.timestamp = to_submit->timestamp;
240         det.processing_time = to_submit->processing_time;
241         det.detection_contours = std::move(to_submit->result_consts);
242         det.has_detections = det.detection_contours.size() > 0;
243         det.data_keep = to_submit->data_keep;
244
245         result_queue_.push_back(det);
246         to_submit = task_queue_.erase(to_submit); // This updates the
247             → iterator to the next element automatically.
248     }
249
250     results_locker.unlock();
251     tasks_locker.unlock();
252     tasks_full_cond_.notify_all(); // Notifying all because we might have
253         → submitted more than 1 frame.
254     results_empty_cond_.notify_all();
255 }
256
257 // Other functions implementation
258
259 void rgb_to_bw(const Image<rgb_pixel> &in, Image<unsigned short> &out)
260 {
261     for (std::size_t i = 0; i < in.get_total(); ++i)
262     {
263         rgb_pixel pixel = in[i];
264         out[i] = pixel[0] * 76.245 + pixel[1] * 149.685 + pixel[2] * 29.07;
265     }
266 }
267
268 void uchar_to_bw(const unsigned char *in, const std::size_t n_pix, Image<
269     → unsigned short> &out)
```

7.5. APÉNDICE

```
262     {
263         for (std::size_t i = 0; i < n_pix; ++i)
264         {
265             std::size_t mapped = i * 3;
266             out[i] = in[mapped] * 76.245 + in[mapped+1] * 149.685 + in[mapped+2] *
267             ↪ 29.07;
268         }
269     }
270 } // namespace motdet
```

7.5.1.4 image_utils.hpp

```
1 #ifndef __MOTDET_IMAGE_UTILS_HPP__
2 #define __MOTDET_IMAGE_UTILS_HPP__
3
4 #include "motion_detector.hpp"
5
6 #include <iostream>
7 #include <cstddef>      // std::size_t
8 #include <array>        // std::array
9 #include <functional>   // std::function
10 #include <cmath>         // std::atan2 std::abs
11 #include <stack>        // std::stack
12
13 /*
14  * This a template headers file. The implementation of the functions has been put
15  * → in image_utils.hpp so that it resembled the structure
16  * of a.hpp/cpp pair, but in the end an ipp file is simply a file that is appended
17  * → to this file, very different from a.cpp.
18
19  Most of the functions in this module allow templated pixel types for the image
19  → inputs and outputs.
20  This is done to allow easy conversions from different image types without
20  → needing to completely reallocate the image data.
21  For example if you have an uchar image and need an int image for another
21  → function (i.e. contour detection)
22  you template the function to input uchar and output int and it will be done.
22  These templates require the type to be a BASIC TYPE, but if you make a custom
22  → type behave like a basic type, it will work too.
23 */
24
25 namespace motdet
26 {
27     namespace imgutil
28     {
29         namespace detail
30         {
31             inline float pi_ = 3.1416; /*< The constant PI. */
31             inline unsigned char n_pixel_neighbor_ = 8; /*< Amount of pixels that
31             → surround a given pixel. In this case, 8. */
32
33             inline std::array<float, 7> gaussian_kernel_5_( { 0.06136, 0.24477,
33             → 0.38775, 0.24477, 0.06136 } );
34 }
```

7.5. APÉNDICE

```
35 // 3x3 Horizontal sobel edge detection kernel.
36 inline std::array<signed char, 9> sobel_h_kernel_3x3_()
37     {1, 0, -1,
38      2, 0, -2,
39      1, 0, -1
40  };
41
42 // 3x3 Vertical sobel edge detection kernel.
43 inline std::array<signed char, 9> sobel_v_kernel_3x3_()
44     {1, 2, 1,
45      0, 0, 0,
46      -1, -2, -1
47  };
48
49 /**
50 * @brief Image kernel function. Find the median of 9 pixels (3x3
51 *       → kernel).
52 * @tparam IN_T must be a basic type of any bit length.
53 * @tparam OUT_T must be a basic type of same or greater bit length
54 *       → than IN_T.
55 * @param p Set of pixels of type IN_T to process.
56 * @return The pixel from these 9 pixels that represents the median.
57 */
58 template <typename IN_T, typename OUT_T>
59 OUT_T kernel_op_median_3x3_(std::array<IN_T, 9> &p);
60
61 /**
62 * @brief Image kernel function. Get the Gaussian blur value of a line
63 *       → kernel of length 5.
64 * @tparam IN_T must be a basic type of any bit length. At max 64b.
65 * @tparam OUT_T must be a basic type of same or greater bit length
66 *       → than IN_T.
67 * @param p Set of pixels of type IN_T to process.
68 * @return The value of the final blurred pixel.
69 */
70 template <typename IN_T, typename OUT_T>
71 OUT_T kernel_op_gaussian_5_(const std::array<IN_T, 5> &p);
72
73 /**
74 * @brief Image kernel function. Detect horizontal edge value of a 3x3
75 *       → kernel. The result can be negative.
76 * @tparam IN_T must be a basic type of any bit length.
77 * @tparam OUT_T must be a signed basic type. Minimum 2 bits longer
78 *       → than IN_T.
79 * @param p Set of pixels of type IN_T to process.
80 * @return The value of the horizontal edge.
81 */
82 template <typename IN_T, typename OUT_T>
83 OUT_T kernel_op_sobel_h_3x3_(const std::array<IN_T, 9> &p);
84
85 /**
86 * @brief Image kernel function. Detect vertical edge value of a 3x3
87 *       → kernel. The result can be negative.
88 * @tparam IN_T must be a basic type of any bit length.
89 * @tparam OUT_T must be a signed basic type. Minimum 2 bits longer
90 *       → than IN_T.
91 * @param p Set of pixels of type IN_T to process.
92 * @return The value of the vertical edge.
93 */
```

7.5. APÉNDICE

```
86 template <typename IN_T, typename OUT_T>
87 OUT_T kernel_op_sobel_v_3x3_(const std::array<IN_T, 9> &p);
88
89 /**
90 * @brief Image kernel function. Returns 1 if any 1-pixel is in the
91 *        ↪ kernel, else return 0.
92 * @tparam IN_T must be a basic type of at least 1 bit (+1 for signed).
93 * @tparam OUT_T must be a basic type of at least 1 bit (+1 for signed)
94 *        ↪ .
95 * @tparam SIZE_K is the size of the kernel to use for dilation. Must
96 *        ↪ be powers of odd numbers: 1, 9, 25, 49.
97 * @param p Set of pixels of type IN_T to process.
98 * @return The value of the dilated pixel.
99 */
100 template <typename IN_T, typename OUT_T, std::size_t SIZE_K>
101 OUT_T kernel_op_dilation_(const std::array<IN_T, SIZE_K> &p);
102
103 /**
104 * @brief Image kernel function. Returns 0 if a 0-pixel is in the
105 *        ↪ kernel, else return 1.
106 * @tparam IN_T must be a basic type of at least 1 bit (+1 for signed).
107 * @tparam OUT_T must be a basic type of at least 1 bit (+1 for signed)
108 *        ↪ .
109 * @tparam SIZE_K is the size of the kernel to use for dilation. Must
110 *        ↪ be powers of odd numbers: 1, 9, 25, 49.
111 * @param p Set of pixels of type IN_T to process.
112 * @return The value of the eroded pixel.
113 */
114 template <typename IN_T, typename OUT_T, std::size_t SIZE_K>
115 OUT_T kernel_op_erosion_(const std::array<IN_T, SIZE_K> &p);
116
117 /**
118 * @brief Fast square root approximation by Jim Ulery.
119 * @details http://www.azillionmonkeys.com/qed/sqrroot.html
120 * @param val long integer to square root.
121 * @return The approximation of the square root, no decimals.
122 */
123 inline unsigned long fast_sqrt_(unsigned long val);
124
125 /**
126 * @brief Swaps the value of a and b. Used by pixel_sort().
127 * @tparam IN_T Type of the parameters to swap. Must be a basic type.
128 * @param a Parameter 1 to be swapped.
129 * @param b Parameter 2 to be swapped.
130 */
131 template <typename IN_T>
132 inline void pixel_swap_(IN_T &a, IN_T &b) { IN_T tmp(std::move(a)); a =
133 *        ↪ b; b = tmp; }
134
135 /**
136 * @brief Swaps the value of a and b if a is bigger than b. Used by
137 *        ↪ kernel_median_3x3().
138 * @tparam IN_T Type of the parameters to sort. Must be a basic type.
139 * @param a Parameter 1 to be compared and possibly swapped.
140 * @param b Parameter 2 to be compared and possibly swapped.
141 */
142 template <typename IN_T>
143 inline void pixel_sort_(IN_T &a, IN_T &b) { if (a > b) pixel_swap_<IN_T
144 *        ↪ >(a, b); }
```

7.5. APÉNDICE

```
136
137  /**
138   * @brief Apply a generic square kernel to an image.
139   * The kernel is applied by a function that takes all the pixels in and
140   * → array and outputs the pixel result.
141   * @tparam IN_T must be a basic type of any bit length.
142   * @tparam OUT_T must be a basic type of equal or greater length than
143   * → IN_T.
144   * @tparam K_SIZE is the size of the kernel to extract around each
145   * → pixel. Must be powers of odd numbers: 1, 9, 25, 49...
146   * @param in Image to convolute.
147   * @param out Image convoluted with the given kernel operator.
148   * @param kernel_operator Function that receives the values of all the
149   * → kernel elements and outputs the pixel value.
150   */
151   template<typename IN_T, typename OUT_T, std::size_t K_SIZE>
152   void square_convolution(const Image<IN_T> &in, Image<OUT_T> &out, const
153   → std::function<OUT_T(std::array<IN_T, K_SIZE>&)> kernel_operator)
154   → ;
155
156 /**
157  * @brief Apply a vertical line kernel to an image.
158  * The kernel is applied by a function that takes all the pixels in and
159  * → array and outputs the pixel result.
160  * @tparam IN_T must be a basic type of any bit length.
161  * @tparam OUT_T must be a basic type of equal or greater length than
162  * → IN_T.
163  * @tparam K_SIZE is the size of the kernel to extract. Must be odd
164  * → numbers: 1, 3, 5, 7...
165  * @param in Image to convolute.
166  * @param out Image convoluted with the given kernel operator.
167  * @param kernel_operator Function that receives the values of all the
168  * → kernel elements and outputs the pixel value.
169  */
170   template<typename IN_T, typename OUT_T, std::size_t K_SIZE>
171   void vline_convolution(const Image<IN_T> &in, Image<OUT_T> &out, const
172   → std::function<OUT_T(std::array<IN_T, K_SIZE>&)> kernel_operator);
173
174 /**
175  * @brief Apply a horizontal line kernel to an image.
176  * The kernel is applied by a function that takes all the pixels in and
177  * → array and outputs the pixel result.
178  * @tparam IN_T must be a basic type of any bit length.
179  * @tparam OUT_T must be a basic type of equal or greater length than
180  * → IN_T.
181  * @tparam K_SIZE is the size of the kernel to extract. Must be odd
182  * → numbers: 1, 3, 5, 7...
183  * @param in Image to convolute.
184  * @param out Image convoluted with the given kernel operator.
185  * @param kernel_operator Function that receives the values of all the
186  * → kernel elements and outputs the pixel value.
187  */
188   template<typename IN_T, typename OUT_T, std::size_t K_SIZE>
189   void hline_convolution(const Image<IN_T> &in, Image<OUT_T> &out, const
190   → std::function<OUT_T(std::array<IN_T, K_SIZE>&)> kernel_operator);
191
192 } // namespace detail
193
194 /**
195
196
197
198
```

7.5. APÉNDICE

```
179 * @brief Apply a 5x5 blurring filter to an image using a traditional
180 *        ↪ kernel. Slow but simple.
181 * @tparam IN_T Type of the pixels to be blurred. Must be a basic type of
182 *        ↪ max 64b long.
183 * @tparam OUT_T Type of the blurred pixels. Basic type of equal or greater
184 *        ↪ bit length than IN_T.
185 * @param in Grayscale image to blur.
186 * @param out Grayscale blurred image.
187 */
188 template <typename IN_T, typename OUT_T>
189 void gaussian.blur_filter(const Image<IN_T> &in, Image<OUT_T> &out);
190
191 /**
192 * @brief Apply a 3x3 median filter to an image. Useful for salt&pepper
193 *        ↪ noise.
194 * @tparam IN_T must be a basic type of any length.
195 * @tparam OUT_T must be a basic type of equal or greater length than IN_T.
196 * @param in Grayscale image to process.
197 * @param out Grayscale image with noise removed.
198 */
199 template <typename IN_T, typename OUT_T>
200 void median.filter(const Image<IN_T> &in, Image<OUT_T> &out);
201
202 /**
203 * @brief Gets the absolute difference between 2 images (always positive).
204 *        ↪ Useful for motion detection.
205 * @tparam IN_T must be a basic type of any length.
206 * @tparam OUT_T must be a basic type of equal or greater length than IN_T.
207 * @param in1 Grayscale image 1 to subtract.
208 * @param in2 Grayscale image 2 to subtract.
209 * @param out Subtracted grayscale image, values are always positive.
210 */
211 template <typename IN_T, typename OUT_T>
212 void image_subtraction(const Image<IN_T> &in1, const Image<IN_T> &in2,
213                      ↪ Image<OUT_T> &out);
214
215 /**
216 * @brief Edge detection that produces a strictly binary image. 0 for no
217 *        ↪ edge, 1 for edge.
218 * @details https://towardsdatascience.com/canny-edge-detection-step-by-step-in-python-computer-vision-b49c3a2d8123
219 * @tparam OUT_T must be a basic type of at least 1 bit.
220 * @param in 8b grayscale image to process.
221 * @param out Binary image of the edges. 1 = edge, 0 = no edge.
222 * @param low_threshold Edges with strength below this value are ignored.
223 * @param high_threshold Edges above this strength value are assured to
224 *        ↪ appear in the final result. Strong edges.
225 */
226 template <typename OUT_T>
227 void canny.edge_detection_8b(const Image<unsigned char> &in, Image<OUT_T> &
228                             ↪ out, const unsigned char low_threshold, const unsigned char
229                             ↪ high_threshold);
230
231 /**
232 * @brief Detect edges with magnitude (strength) and gradient (direction)
233 *        ↪ in a grayscale pictu[U+EFFD] [U+FFFD]
234 * @tparam OUT_T must be a basic type of equal or greater length than IN_T.
235 * @param in 8b grayscale image to process.
```

7.5. APÉNDICE

```
226 * @param out_magnitude Grayscale image where a higer value means stronger
227   ↪ edge.
228 * @param out_gradient Grayscale image with the direction of the edges. 0
229   ↪ is 0 deg, 255 is 360 deg, it prioritizes memory efficiency to
230   ↪ precision.
231 */
232 template<typename OUT_T>
233 void sobel_edge_detection_8b(const Image<unsigned char> &in, Image<OUT_T> &
234   ↪ out_magnitude, Image<unsigned char> &out_gradient);
235
236 /**
237 * @brief Reduces thickness of sobel edges in an image by removing non-
238   ↪ essential points, picked out by edge direction analysis.
239 * @tparam IN_T must be a basic type of any bit length.
240 * @tparam OUT_T must be a basic type of equal or greater length than IN_T.
241 * @param in_magnitude Edges grayscale image, like the ones outputted by
242   ↪ sobel_edge_detection().
243 * @param in_gradient Grayscale image where 0 means 0 degrees and 255 means
244   ↪ 360 degree direction.
245 * @param out Grayscale image with reduces edge thickness.
246 */
247 template<typename IN_T, typename OUT_T>
248 void non_max_suppression(const Image<IN_T> &in_magnitude, const Image<
249   ↪ unsigned char> &in_gradient, Image<OUT_T> &out);
250
251 /**
252 * @brief Collapses all the values in a grayscale image to the states
253   ↪ Culled 0, Strong 1 and Weak 2 depending on 2 thresholds.
254 * @tparam IN_T must be a basic type of any bit length.
255 * @tparam OUT_T must be a basic type of at least 2 bits (+1 for signed).
256 * @param in Image to collapse.
257 * @param out Image of the collapsed states. A value that is between the 2
258   ↪ thresholds is set to Weak.
259 * @param low_threshold Any value below this threshold is transformed to
260   ↪ Culled.
261 * @param high_threshold Any value equal or above this threshold is
262   ↪ transformed to Strong. REQ: high_threshold > low_threshold.
263 */
264 template <typename IN_T, typename OUT_T>
265 void double_threshold(const Image<IN_T> &in, Image<OUT_T> &out, const IN_T
266   ↪ low_threshold, const IN_T high_threshold);
267
268 /**
269 * @brief Collapses all the values in a grayscale image to the states
270   ↪ Culled 0 and Strong 1 depending on a threshold.
271 * @tparam IN_T must be a basic type of any bit length.
272 * @tparam OUT_T must be a basic type of at least 1 bit (+1 for signed).
273 * @param in Image to collapse.
274 * @param out Image of the collapsed states.
275 * @param threshold Any value below this threshold is set to Culled, the
276   ↪ rest are Strong.
277 */
278 template <typename IN_T, typename OUT_T>
279 void single_threshold(const Image<IN_T> &in, Image<OUT_T> &out, const IN_T
280   ↪ threshold);
281
282 /**
283 * @brief Takes the output of a double threshold function and turns Weak
284   ↪ pixel into either Strong or Culled.
```

7.5. APÉNDICE

```
|269 * @tparam IN_T must be a basic type of any bit length.  
|270 * @tparam OUT_T must be a basic type of at least 1 bit (+1 for signed).  
|271 * @details Turns a Weak pixel into Strong if connected directly or  
|272     ↪ indirectly to another Strong pixel, else culls it.  
|273 * @param in Images with 3 possible values: Culled 0, Strong 1, Weak 2.  
|274 * @param out Image with 2 possible values: Culled 0, Strong 1.  
|275 */  
|276 template <typename IN_T, typename OUT_T>  
|277 void hysteresis(const Image<IN_T> &in, Image<OUT_T> &out);  
|278  
|279 /**  
|280 * @brief Creates an intermediate image between 2 given images. If ratio is  
|281     ↪ 1 it will be equivalent to "to", and 0 will be equivalent to "from"  
|282     ↪ ".  
|283 * @tparam IN_T must be a basic type of any bit length.  
|284 * @tparam OUT_T must be a basic type of equal or greater length than IN_T.  
|285 * @param from Image that has more relevance the closer "ratio" is to 0.  
|286 * @param to Image that has more relevance the closer "ratio" is to 1.  
|287 * @param out Image with interpolated pixels.  
|288 * @param ratio Selector for which input image has more relevance. [0-1]  
|289 */  
|290 template <typename IN_T, typename OUT_T>  
|291 void image_interpolation(const Image<IN_T> &from, const Image<IN_T> &to,  
|292     ↪ Image<OUT_T> &out, const float ratio);  
|293  
|294 /**  
|295 * @brief Takes a binary image (0 or 1) and dilates the 1-pixels. It  
|296     ↪ interprets any value above 1 as 1. Below 0 is 0.  
|297 * @tparam IN_T must be a basic type of at least 1 bit (+1 for signed).  
|298 * @tparam OUT_T must be a basic type of at least 1 bit (+1 for signed).  
|299 * @param in Binary image to process.  
|300 * @param out Dilated binary image.  
|301 */  
|302 template <typename IN_T, typename OUT_T>  
|303 void dilation(const Image<IN_T> &in, Image<OUT_T> &out);  
|304  
|305 /**  
|306 * @brief Takes a binary image (0 or 1) and erodes the 1-pixels. It  
|307     ↪ interprets any value above 1 as 1. Below 0 is 0.  
|308 * @tparam IN_T must be a basic type of at least 1 bit (+1 for signed).  
|309 * @tparam OUT_T must be a basic type of at least 1 bit (+1 for signed).  
|310 * @param in Binary image to process.  
|311 * @param out Eroded binary image.  
|312 */  
|313 template <typename IN_T, typename OUT_T>  
|314 void erosion(const Image<IN_T> &in, Image<OUT_T> &out);  
|315  
|316 /**  
|317 * @brief Turn an image that only contains the values 0 and 1 to an image  
|318     ↪ that only contains the values 0 and max_val.  
|319 * @tparam IN_T must be a basic type of any bit length.  
|320 * @tparam OUT_T must be a basic type of any bit length.  
|321 * @param in Image that only contains the values 0 and 1. A value below 0  
|322     ↪ is considered 0 and above 1 is considered 1.  
|323 * @param out Image with all the 1s in the input image turned to max_val.  
|324 * @param max_val The value that will represent 1 in the output image.  
|325 */  
|326 template<typename IN_T, typename OUT_T>
```

7.5. APÉNDICE

```
319     void reescale_pix_length(const Image<IN_T> &in, Image<OUT_T> &out, IN_T
320                             ↪ in_max_val, OUT_T out_max_val);
321
322     /**
323      * @brief Resizes to a lower resolution by a given factor. Ignores floating
324      *       → point precision.
325      * @tparam IN_T must be a basic type of any bit length.
326      * @tparam OUT_T must be a basic type of equal or greater length than IN_T.
327      * @param in Image to resize, resolution must be at least "factor" in width
328      *       → and height.
329      * @param out Resized image. Resolution must be ceil(in.w/factor) by ceil(
330      *       → in.h/factor)
331      * @param factor Factor to resize the image, must be > 0.
332      */
333     template<typename IN_T, typename OUT_T>
334     void downsample(const Image<IN_T> &in, Image<OUT_T> &out, std::size_t
335                     ↪ factor);
336
337     /**
338      * @brief Resizes to a higher resolution by a given factor.
339      * @tparam IN_T must be a basic type of any bit length.
340      * @tparam OUT_T must be a basic type of equal or greater length than IN_T.
341      * @param in Image to resize, can have any resolution.
342      * @param out Resized image. Resolution must be in.w*factor by in.h*factor
343      * @param factor Factor to resize the image, must be > 0.
344      */
345     template<typename IN_T, typename OUT_T>
346     void upsample(const Image<IN_T> &in, Image<OUT_T> &out, std::size_t factor)
347                     ↪ ;
348
349 // Include the file with the actual definitions for the headers we have
350 // declared above.
351 #include "image_utils.ipp"
352
353 } // namespace imgutil
354 } // namespace motdet
355
356#endif // __MOTDET_IMAGE_UTILS_HPP__
```

7.5.1.5 image_utils.ipp

```
1 namespace detail
2 {
3     template <typename IN_T, typename OUT_T>
4     OUT_T kernel_op_median_3x3_(std::array<IN_T, 9> &p)
5     {
6         // This is the fastest known algorithm without making assumptions about the
6         //       → input.
7         // "Borrowed" from http://ndevilla.free.fr/median/median.pdf
8
9         pixel_sort_<IN_T>(p[1], p[2]); pixel_sort_<IN_T>(p[4], p[5]); pixel_sort_<
10        ↪ IN_T>(p[7], p[8]);
11         pixel_sort_<IN_T>(p[0], p[1]); pixel_sort_<IN_T>(p[3], p[4]); pixel_sort_<
11        ↪ IN_T>(p[6], p[7]);
12         pixel_sort_<IN_T>(p[1], p[2]); pixel_sort_<IN_T>(p[4], p[5]); pixel_sort_<
12        ↪ IN_T>(p[7], p[8]);
```

7.5. APÉNDICE

```
12     pixel_sort_<IN_T>(p[0], p[3]); pixel_sort_<IN_T>(p[5], p[8]); pixel_sort_<
13         ↪ IN_T>(p[4], p[7]);
14     pixel_sort_<IN_T>(p[3], p[6]); pixel_sort_<IN_T>(p[1], p[4]); pixel_sort_<
15         ↪ IN_T>(p[2], p[5]);
16     pixel_sort_<IN_T>(p[4], p[7]); pixel_sort_<IN_T>(p[4], p[2]); pixel_sort_<
17         ↪ IN_T>(p[6], p[4]);
18     pixel_sort_<IN_T>(p[4], p[2]);
19
20     return(p[4]);
21 }
22
23 template <typename IN_T, typename OUT_T>
24 OUT_T kernel_op_gaussian_5_(const std::array<IN_T, 5> &p)
25 {
26     double total = 0;
27     for(std::size_t i = 0; i < 5; ++i) total += p[i] * gaussian_kernel_5_[i];
28     return total;
29 }
30
31 template <typename IN_T, typename OUT_T>
32 OUT_T kernel_op_sobel_h_3x3_(const std::array<IN_T, 9> &p)
33 {
34     OUT_T total = 0;
35     for(std::size_t i = 0; i < 9; ++i) total += p[i] * sobel_h_kernel_3x3_[i];
36     return total;
37 }
38
39 template <typename IN_T, typename OUT_T>
40 OUT_T kernel_op_sobel_v_3x3_(const std::array<IN_T, 9> &p)
41 {
42     OUT_T total = 0;
43     for(std::size_t i = 0; i < 9; ++i) total += p[i] * sobel_v_kernel_3x3_[i];
44 }
45
46 template <typename IN_T, typename OUT_T, std::size_t SIZE_K>
47 OUT_T kernel_op_dilation_(const std::array<IN_T, SIZE_K> &p)
48 {
49     for(const IN_T &i: p)
50     {
51         if (i >= 1) return 1;
52     }
53     return 0;
54 }
55
56 template <typename IN_T, typename OUT_T, std::size_t SIZE_K>
57 OUT_T kernel_op_erosion_(const std::array<IN_T, SIZE_K> &p)
58 {
59     for(const IN_T &i: p)
60     {
61         if (i <= 0) return 0;
62     }
63     return 1;
64 }
65
66 inline unsigned long fast_sqrt_(unsigned long val)
67 {
68     unsigned long temp, g = 0, b = 0x8000, bshft = 15;
```

7.5. APÉNDICE

```
68
69     do
70     {
71         if (val >= (temp = (((g << 1) + b)<<bshft--)))
72         {
73             g += b;
74             val -= temp;
75         }
76     }
77     while (b >>= 1);
78
79     return g;
80 }
81
82 template<typename IN_T, typename OUT_T, std::size_t K_SIZE>
83 void square_convolution(const Image<IN_T> &in, Image<OUT_T> &out, const std::
84     ↪ function<OUT.T(std::array<IN_T, K_SIZE>&)> kernel_operator)
85 {
86     std::size_t current_pos;
87     std::size_t height = in.get_height(), width = in.get_width();
88     int kernel_side, kernel_radius;
89
90     kernel_side = detail::fast_sqrt_(K_SIZE); // Since a kernel is strictly
91     ↪ square, the square root is the side length.
92     if(kernel_side == 0 || kernel_side % 2 == 0) throw std::invalid_argument("
93     ↪ Kernel is of invalid size.");
94
95     kernel_radius = kernel_side >> 1;
96
97     std::array<IN_T, K_SIZE> extracted_kernel;
98
99     for(std::size_t i = 0; i < height; ++i)
100    {
101        for(std::size_t j = 0; j < width; ++j)
102        {
103            current_pos = i * width + j;
104
105            for(int ki = -kernel_radius; ki <= kernel_radius; ++ki)
106            {
107                for(int kj = -kernel_radius; kj <= kernel_radius; ++kj)
108                {
109                    int real_ki = i + ki;
110                    int real_kj = j + kj;
111
112                    if(real_ki < 0) real_ki = 0;
113                    else if (real_ki >= height) real_ki = height-1;
114
115                    if(real_kj < 0) real_kj = 0;
116                    else if (real_kj >= width) real_kj = width-1;
117
118                    // Fill in the kernel that will be passed to the function
119                    ↪ with the values extracted from the image.
120                    extracted_kernel[(ki + kernel_radius)*kernel_size + kj +
121                                     ↪ kernel_radius] = in[real_ki*width + real_kj];
122                }
123            }
124        }
125    }
126
127    // The kernel has been fully extracted, time to execute kernel
128    ↪ operation.
129    out[current_pos] = kernel_operator(extracted_kernel);
130 }
```

7.5. APÉNDICE

```
121         }
122     }
123 }
124
125 template<typename IN_T, typename OUT_T, std::size_t K_SIZE>
126 void vline_convolution(
127 {
128     std::size_t current_pos;
129     std::size_t height = in.get_height(), width = in.get_width();
130     int kernel_radius;
131
132     if(K_SIZE % 2 == 0) throw std::invalid_argument("Kernel is of invalid size.
133     ");
134
135     kernel_radius = K_SIZE >> 1;
136
137     std::array<IN_T, K_SIZE> extracted_kernel;
138
139     for(std::size_t i = 0; i < height; ++i)
140     {
141         for(std::size_t j = 0; j < width; ++j)
142         {
143             current_pos = i * width + j;
144
145             for(int ki = -kernel_radius; ki <= kernel_radius; ++ki)
146             {
147                 int real_ki = i + ki;
148
149                 if(real_ki < 0) real_ki = 0;
150                 else if (real_ki >= height) real_ki = height-1;
151
152                 // Fill in the kernel that will be passed to the function with
153                 // the values extracted from the image.
154                 extracted_kernel[ki + kernel_radius] = in[real_ki*width + j];
155             }
156             // The kernel has been fully extracted, time to execute kernel
157             // operation.
158             out[current_pos] = kernel_operator(extracted_kernel);
159         }
160     }
161
162     template<typename IN_T, typename OUT_T, std::size_t K_SIZE>
163     void hline_convolution(
164     {
165         std::size_t current_pos;
166         std::size_t height = in.get_height(), width = in.get_width();
167         int kernel_radius;
168
169         if(K_SIZE % 2 == 0) throw std::invalid_argument("Kernel is of invalid size.
170         ");
171
172         kernel_radius = K_SIZE >> 1;
173
174         std::array<IN_T, K_SIZE> extracted_kernel;
175
176         for(std::size_t i = 0; i < height; ++i)
```

7.5. APÉNDICE

```
174     {
175         for(std::size_t j = 0; j < width; ++j)
176     {
177         current_pos = i * width + j;
178
179         for(int kj = -kernel_radius; kj <= kernel_radius; ++kj)
180         {
181             int real_kj = j + kj;
182
183             if(real_kj < 0) real_kj = 0;
184             else if (real_kj >= width) real_kj = width-1;
185
186             // Fill in the kernel that will be passed to the function with
187             // → the values extracted from the image.
188             extracted_kernel[kj + kernel_radius] = in[i*width + real_kj];
189         }
190         // The kernel has been fully extracted , time to execute kernel
191         // → operation .
192         out[current_pos] = kernel_operator(extracted_kernel);
193     }
194 }
195 } // namespace detail
196
197 template <typename IN_T, typename OUT_T>
198 void gaussian_blur_filter(const Image<IN_T> &in, Image<OUT_T> &out)
199 {
200     std::size_t height = in.get_height(), width = in.get_width();
201     Image<float> half_blurred(width, height, {});
202
203     // An NxN gaussian blur can be decomposed into 2 1-dimensional kernels , N
204     // → vertical and N horizontal.
205     // Using float as the intermediate pixel type to avoid detail loss between
206     // → steps .
207
208     detail::vline_convolution<IN_T, float, 5>(in, half_blurred, detail::
209     // → kernel_op_gaussian_5_-<IN_T, float>);
210     detail::hline_convolution<float, OUT_T, 5>(half_blurred, out, detail::
211     // → kernel_op_gaussian_5_-<float, IN_T>);
212 }
213
214 template <typename IN_T, typename OUT_T>
215 void median_filter(const Image<IN_T> &in, Image<OUT_T> &out)
216 {
217     detail::square_convolution<IN_T, OUT_T, 9>(in, out, detail::
218     // → kernel_op_median_3x3_-<IN_T, OUT_T>);
219 }
220
221 template <typename IN_T, typename OUT_T>
222 void image_subtraction(const Image<IN_T> &in1, const Image<IN_T> &in2, Image<OUT_T>
223     // → &out)
224 {
225     std::size_t total = in1.get_total();
226
227     for(std::size_t i = 0; i < total; ++i)
228     {
229         if(in1[i] > in2[i]) out[i] = in1[i] - in2[i];
230         else out[i] = in2[i] - in1[i];
231     }
232 }
```

7.5. APÉNDICE

```
|225     }
|226 }
|227
|228 template <typename OUT_T>
|229 void canny_edge_detection_8b(const Image<unsigned char> &in, Image<OUT_T> &out,
|230   ↪ const unsigned char low_threshold, const unsigned char high_threshold)
|231 {
|232     std::size_t height = in.get_height(), width = in.get_width();
|233
|234     Image<unsigned char> gra_image(width, height, {}), thresh_image(width, height,
|235       ↪ {}), mag_image(width, height, {}), sup_image(width, height, {});
|236
|237     sobel_edge_detection_8b<unsigned char>           (in, mag_image, gra_image
|238       ↪ );
|239     non_max_suppression<unsigned char, unsigned char> (    mag_image, gra_image,
|240       ↪ sup_image
|241     );
|242     double_threshold<unsigned char, unsigned char>    (
|243       ↪ sup_image, thresh_image,      low_threshold, high_threshold);
|244     hysteresis<unsigned char, OUT_T>                  (
|245       ↪
|246       ↪ thresh_image, out
|247     );
|248 }
|249
|250 template<typename OUT_T>
|251 void sobel_edge_detection_8b(const Image<unsigned char> &in, Image<OUT_T> &
|252   ↪ out_magnitude, Image<unsigned char> &out_gradient)
|253 {
|254     short aux_mag; // input is 8b and we need a signed type of 10b. Closest
|255       ↪ candidate is short.
|256     double aux_gra;
|257
|258     std::size_t height = in.get_height(), width = in.get_width(), total = in.
|259       ↪ get_total();
|260     Image<short> sobel_h(width, height, {}), sobel_v(width, height, {});
|261
|262     detail::square_convolution<unsigned char, short, 9>(in, sobel_h, detail::
|263       ↪ kernel_op_sobel_h_3x3_-<unsigned char, short>);
|264     detail::square_convolution<unsigned char, short, 9>(in, sobel_v, detail::
|265       ↪ kernel_op_sobel_v_3x3_-<unsigned char, short>);
|266
|267     for(std::size_t i = 0; i < total; ++i)
|268     {
|269         aux_mag = detail::fast_sqrt_(sobel_h[i] * sobel_h[i] + sobel_v[i] * sobel_v
|270           ↪ [i]);
|271
|272         // Intensity of the edge from 0 (B) to 255 (W)
|273         out_magnitude[i] = aux_mag > 255 ? 255 : aux_mag;
|274
|275         // Angle of the edge in radians. Returns the angle of radians from origin
|276           ↪ (0, 0) to point (sobel_h[i], sobel_v[i])
|277         aux_gra = atan2(sobel_h[i], sobel_v[i]);
|278
|279         // Convert rad to deg: degrees = radians * 180 * PI
|280         // To map from 360 deg range to 255 deg, multiply by 0.71: degrees =
|281           ↪ radians * 127.8 * PI
|282         // Fuse that number with PI and the final instruction is: degrees = radians
|283           ↪ * 401.45
|284         out_gradient[i] = aux_gra * 401.45;
|285     }
```

7.5. APÉNDICE

```
|268 }
269
270 template<typename IN_T, typename OUT_T>
271 void non_max_suppression(const Image<IN_T> &in_magnitude, const Image<unsigned char
272   ↪ > &in_gradient, Image<OUT_T> &out)
273 {
274   IN_T q, r;
275   bool valid_pixel;
276   unsigned short deg;
277   std::size_t height = in_magnitude.get_height(), width = in_magnitude.get_width
278   ↪ (), current_pos;
279
280   for(std::size_t i = 1; i < height-1; ++i)
281   {
282     for(std::size_t j = 1; j < width-1; ++j)
283     {
284       current_pos = i * width + j;
285
286       deg = in_gradient[current_pos] * 1.4; // Map the degrees from 255 back
287       ↪ to 360. This is an approximation, but it's good enough.
288       valid_pixel = false;
289
290       if((0 <= deg && deg < 22.5) || (157.5 <= deg && deg <= 180)){ // Angle
291         ↪ 0 (-)
292         q = in_magnitude[current_pos+1];
293         r = in_magnitude[current_pos-1];
294       }
295       else if(22.5 <= deg && deg < 67.5){                                // Angle 45 (/)
296         q = in_magnitude[current_pos+width-1];
297         r = in_magnitude[current_pos-width+1];
298       }
299       else if(67.5 <= deg && deg < 112.5){                               // Angle 90 (|)
300         q = in_magnitude[current_pos+width];
301         r = in_magnitude[current_pos-width];
302       }
303       else if(112.5 <= deg && deg < 157.5){                           // Angle 135 (\})
304         q = in_magnitude[current_pos-width-1];
305         r = in_magnitude[current_pos+width+1];
306       }
307     }
308   }
309
310 template <typename IN_T, typename OUT_T>
311 void double_threshold(const Image<IN_T> &in, Image<OUT_T> &out, const IN_T
312   ↪ low_threshold, const IN_T high_threshold)
313 {
314   std::size_t total = in.get_total();
315   IN_T val;
316
317   for(int i = 0; i < total; ++i)
318   {
319     val = in[i];
320     if      (val >= high_threshold) out[i] = 1; // Strong
321     else if(val < low_threshold)    out[i] = 0; // Culled
```

7.5. APÉNDICE

```

321         else                                out[i] = 2; // Weak
322     }
323 }
324
325 template <typename IN_T, typename OUT_T>
326 void single_threshold(const Image<IN_T> &in, Image<OUT_T> &out, const IN_T
327   ↪ threshold)
328 {
329     std::size_t total = in.get_total();
330     IN_T val;
331
332     for(int i = 0; i < total; ++i)
333     {
334         val = in[i];
335         if (val >= threshold) out[i] = 1; // Strong
336         else                  out[i] = 0; // Culled
337     }
338
339 template <typename IN_T, typename OUT_T>
340 void hysteresis(const Image<IN_T> &in, Image<OUT_T> &out)
341 {
342     std::size_t stack_top = 0, current_pos, k_pos;
343     std::stack<std::size_t> strong_pixel_stack;
344     std::size_t height = in.get_height(), width = in.get_width();
345
346     Image<unsigned char> visited_map(width, height, 0); // We need a way to check
347       ↪ if a pixel has already been processed.
348
349     // First iterate over image to set borders to 0 and collect all the strong
350       ↪ edges into a stack.
351     for(std::size_t i = 0; i < height; ++i)
352     {
353         for(std::size_t j = 0; j < width; ++j)
354         {
355             current_pos = i * width + j;
356
357             // If edge of image, set to Culled and mark as visited.
358             if(j == 0 || j == width-1 || i == 0 || i == height-1)
359             {
360                 out[current_pos] = 0;
361                 visited_map[current_pos] = 1;
362             }
363             // If current position is a strong edge, add to the stack to check
364               ↪ later
365             else if(in[current_pos] == 1) strong_pixel_stack.push(current_pos);
366         }
367     }
368
369     // Once all the strong edges are collected, analyze them for neighboring weak
370       ↪ edges that can be set as strong.
371     while(!strong_pixel_stack.empty())
372     {
373         current_pos = strong_pixel_stack.top();
374         strong_pixel_stack.pop();
375
376         out[current_pos] = 1;
377
378         for(signed char ki = -1; ki < 2; ++ki)
379         {
380             for(signed char kj = -1; kj < 2; ++kj)
381             {
382                 if(ki == 0 && kj == 0) continue;
383
384                 int pos = current_pos + (kj * width) + ki;
385
386                 if(visited_map[pos] == 1) continue;
387
388                 if(out[pos] == 0) out[pos] = 1;
389                 else if(out[pos] == 1) visited_map[pos] = 1;
390
391                 if(out[pos] == 1) strong_pixel_stack.push(pos);
392             }
393         }
394     }
395 }
```

7.5. APÉNDICE

```

375     {
376         for(signed char kj = -1; kj < 2; ++kj)
377     {
378         k_pos = current_pos + width*ki + kj;
379         if(!visited_map[k_pos] && in[k_pos] == 2) strong_pixel_stack.push(
380             ↪ k_pos);
381         visited_map[k_pos] = true;
382     }
383 }
384 }

385 template <typename IN_T, typename OUT_T>
386 void image_interpolation(const Image<IN_T> &from, const Image<IN_T> &to, Image<
387     ↪ OUT_T> &out, const float ratio)
388 {
389     std::size_t total = out.get_total();
390
391     for(std::size_t i = 0; i < total; ++i)
392     {
393         out[i] = from[i] + ratio * (to[i] - from[i]); // Simplified from equation:
394             ↪ from[i]*(1-ratio) + to[i]*ratio
395     }
396 }

397 template <typename IN_T, typename OUT_T>
398 void dilation(const Image<IN_T> &in, Image<OUT_T> &out)
399 {
400     std::size_t height = in.get_height(), width = in.get_width();
401     Image<IN_T> half_dilated(width, height, {});
402
403     detail::vline_convolution<IN_T, IN_T, 3>(in, half_dilated, detail::
404         ↪ kernel_op_dilation_<IN_T, IN_T, 3>);
405     detail::hline_convolution<IN_T, OUT_T, 3>(half_dilated, out, detail::
406         ↪ kernel_op_dilation_<IN_T, OUT_T, 3>);
407 }

408 template <typename IN_T, typename OUT_T>
409 void erosion(const Image<IN_T> &in, Image<OUT_T> &out)
410 {
411     std::size_t height = in.get_height(), width = in.get_width();
412     Image<IN_T> half_eroded(width, height, {});
413
414     detail::vline_convolution<IN_T, IN_T, 3>(in, half_eroded, detail::
415         ↪ kernel_op_erosion_<IN_T, IN_T, 3>);
416     detail::hline_convolution<IN_T, OUT_T, 3>(half_eroded, out, detail::
417         ↪ kernel_op_erosion_<IN_T, OUT_T, 3>);
418 }

419 template<typename IN_T, typename OUT_T>
420 void reescale_pix_length(const Image<IN_T> &in, Image<OUT_T> &out, IN_T in_max_val,
421     ↪ OUT_T out_max_val)
422 {
423     std::size_t total = out.get_total();
424
425     double slope = (double)out_max_val/(double)in_max_val;
426
427     for(std::size_t i = 0; i < total; ++i)

```

7.5. APÉNDICE

```
426     { out[i] = ((float)in[i])*slope; }
```

```
427 }
```

```
428
```

```
429
```

```
430 template<typename IN_T, typename OUT_T>
```

```
431 void downsample(const Image<IN_T> &in, Image<OUT_T> &out, std::size_t factor)
```

```
432 {
```

```
433     std::size_t in_height = in.get_height(), in_width = in.get_width();
```

```
434     std::size_t out_height = out.get_height(), out_width = out.get_width();
```

```
435
```

```
436     std::size_t excess_height = in_height%factor, excess_width = in_width%factor;
```

```
437     std::size_t iter_height = out_height, iter_width = out_width;
```

```
438
```

```
439     if(excess_height > 0) —iter_height;
```

```
440     if(excess_width > 0) —iter_width;
```

```
441
```

```
442     // The image will be analyzed with sampler boxes of size factor*factor.
```

```
443     // The image is divided into 4 sections.
```

```
444     // 1.— Area that fits within the sampler box matrix
```

```
445     // 2.— Excess area on the side (w) that does not fit in the sampler matrix.
```

```
446     // 3.— Excess area at the bottom (h) that does not fit in the sampler matrix.
```

```
447     // 4.— Excess corner at the bottom right of the image.
```

```
448
```

```
449     unsigned int sampler_divisor = factor*factor;
```

```
450     unsigned int sampler_divisor_excessw = factor*excess_width;
```

```
451     unsigned int sampler_divisor_excessh = factor*excess_height;
```

```
452     unsigned int sampler_divisor_excesswh = excess_width*excess_height;
```

```
453     long long sampler_accumulator;
```

```
454
```

```
455     for(std::size_t i = 0; i < iter_height; ++i)
```

```
456     {
```

```
457         std::size_t sampler_i = i*factor;
```

```
458         for(std::size_t j = 0; j < iter_width; ++j)
```

```
459         {
```

```
460             std::size_t sampler_j = j*factor;
```

```
461             std::size_t sampler_pos = sampler_i*in_width + sampler_j;
```

```
462             sampler_accumulator = 0;
```

```
463
```

```
464             for(std::size_t box_i = 0; box_i < factor; ++box_i)
```

```
465             {
```

```
466                 std::size_t box_i_dis = box_i*in_width;
```

```
467                 for(std::size_t box_j = 0; box_j < factor; ++box_j)
```

```
468                 {
```

```
469                     sampler_accumulator += in[sampler_pos + box_i_dis + box_j];
```

```
470                 }
```

```
471             }
```

```
472             out[i*out_width + j] = sampler_accumulator / sampler_divisor;
```

```
473         }
```

```
474
```

```
475         if(excess_width)
```

```
476         {
```

```
477             // Now process the excess width for the corresponding sampler row
```

```
478
```

```
479             std::size_t sampler_pos = (sampler_i+1)*in_width - excess_width;
```

```
480             sampler_accumulator = 0;
```

```
481
```

```
482             for(std::size_t box_i = 0; box_i < factor; ++box_i)
```

```
483             {
```

```
484                 std::size_t box_i_dis = box_i*in_width;
```

7.5. APÉNDICE

```
485         for (std::size_t box_j = 0; box_j < excess_width; ++box_j)
486         {
487             sampler_accumulator += in[sampler_pos + box_i_dis + box_j];
488         }
489     }
490     out[(i+1)*out_width - 1] = sampler_accumulator /
491     // sampler_divisor_excessw;
492 }
493
494 if(excess_height)
495 {
496     // Now process the excess height of the image, the remaining excess bottom
497
498     std::size_t sampler_i = (in_height-excess_width+1)*in_width;
499     for(std::size_t j = 0; j < iter_width; ++j)
500     {
501         std::size_t sampler_pos = sampler_i + j*factor;
502         sampler_accumulator = 0;
503
504         for(std::size_t box_i = 0; box_i < excess_height; ++box_i)
505         {
506             std::size_t box_i_dis = box_i*in_width;
507             for(std::size_t box_j = 0; box_j < factor; ++box_j)
508             {
509                 sampler_accumulator += in[sampler_pos + box_i_dis + box_j];
510             }
511         }
512         out[(out_height-1)*out_width + j] = sampler_accumulator /
513         // sampler_divisor_excessh;
514     }
515
516     if(excess_width)
517     {
518         // Process the last excess corner at the bottom right
519         std::size_t sampler_pos = ((in_height-excess_width+1)*in_width) +
520         // in_width-excess_width;
521         sampler_accumulator = 0;
522
523         for(std::size_t box_i = 0; box_i < excess_height; ++box_i)
524         {
525             std::size_t box_i_dis = box_i*in_width;
526             for(std::size_t box_j = 0; box_j < excess_width; ++box_j)
527             {
528                 sampler_accumulator += in[sampler_pos + box_i_dis + box_j];
529             }
530         }
531         out[(out_height-1)*out_width + out_width - 1] = sampler_accumulator /
532         // sampler_divisor_excesswh;
533     }
534 }
535
536 template<typename IN_T, typename OUT_T>
537 void upsample(const Image<IN_T> &in, Image<OUT_T> &out, std::size_t factor)
538 {
539     std::size_t in_height = in.get_height(), in_width = in.get_width();
```

7.5. APÉNDICE

```
540     std::size_t out_height = out.get_height(), out_width = out.get_width();
541
542     for (std::size_t i = 0; i < in_height; ++i)
543     {
544         std::size_t scaled_i = i * factor;
545         for (std::size_t j = 0; j < in_width; ++j)
546         {
547             std::size_t scaled_pos = scaled_i * out_width + j * factor;
548             IN_T val = in[i * in_width + j];
549             for (std::size_t box_i = 0; box_i < factor; ++box_i)
550             {
551                 std::size_t box_i_dis = box_i * out_width;
552                 for (std::size_t box_j = 0; box_j < factor; ++box_j)
553                 {
554                     out[scaled_pos + box_i_dis + box_j] = val;
555                 }
556             }
557         }
558     }
559 }
```

7.5.1.6 contour_detector.hpp

```
1 #ifndef _MOTDET_CONTOUR_DETECTOR_HPP_
2 #define _MOTDET_CONTOUR_DETECTOR_HPP_
3
4 #include "motion_detector.hpp"
5
6 namespace motdet
7 {
8     namespace imgutil
9     {
10
11     /**
12      * @brief Detects contours in a binary image. The input image will be
13      *        modified with the found contour IDs.
14      * @details
15      * Based on Topological Structural Analysis of Digitized Binary Images by
16      *        Border Following, by Suzuki, S. and Abe, K. 1985.
17      * The resulting image from this function can be complex to understand, we
18      *        recommend reading the original paper to make use of it.
19      * @param in Binary integer image. All values must be either 0 or 1 upon
20      *        input. The output image is as defined in the paper.
21      * @param conts The found contours along with its hierarchy and
22      *        topological information.
23      * @param trim_borders if the input image has any value other than 0 in the
24      *        outermost borders, it must be trimmed to apply this algorithm to it
25      *        .
26      * * Deactivate for better speed but make sure the input matrix has 0-pixel
27      *        borders, else the function can hang in an infinite loop or segfault.
28      */
29     void contour_detection(Image<int> &in, std::vector<Extended_contour> &conts
30                           , bool trim_borders = true);
31
32 } // namespace imgutil
```

7.5. APÉNDICE

```
24 } // namespace motdet
25
26 #endif // _MOTDET_CONTOUR_DETECTOR_HPP_
```

7.5.1.7 contour_detector.cpp

```
1 #include "contour_detector.hpp"
2
3 namespace motdet
4 {
5     namespace imgutil
6     {
7         /**
8          * @brief Using a pixel x,y and a neighbour id (0 to 7), gets the neighbour
9          *        ↪ x,y position.
10         * @param i y position of the center pixel.
11         * @param j x position of the center pixel.
12         * @param id id of the neighbour to get.
13         * @param out_i y position of the neighbour.
14         * @param out_j x position of the neighbour.
15         * @return true A valid id was given and the neighbour has been set.
16         * @return false The id was invalid and out_i and out_j are invalid.
17         */
18     bool neighbor_id_to_index_(const std::size_t i, const std::size_t j, const
19                               ↪ unsigned char id, std::size_t &out_i, std::size_t &out_j);
20
21     /**
22      * @brief Using a pixel x0,y0 and another with position x,y, gets the
23      *        ↪ neighbour id of the other pixel, if valid.
24      * @param i0 y position of the center pixel.
25      * @param j0 x position of the center pixel.
26      * @param i y position of the other pixel to check.
27      * @param j x position of the other pixel to check.
28      * @return signed char id of the neighbour that was checked. -1 if it was
29      *        ↪ not a neighbour.
30      */
31     signed char neighbor_index_to_id_(const std::size_t i0, const std::size_t
32                                     ↪ j0, const std::size_t i, const std::size_t j);
33
34     /**
35      * @brief Find the first 0-pixel in the neighbourhood of a pixel x0,y0.
36      *        ↪ Check neighbours rotating counterclockwise.
37      * @param in Image to analyse.
38      * @param i0 y position of the center pixel.
39      * @param j0 x position of the center pixel.
40      * @param i y position of the first neighbour to check.
41      * @param j x position of the first neighbour to check.
42      * @param offset neighbours to skip before actually checking. Regardless of
43      *        ↪ this value, all neighbours will be checked.
44      * @param out_i y position of the found pixel.
45      * @param out_j x position of the found pixel.
46      * @return true a valid pixel was found, and out_i, out_j is valid.
47      * @return false a valid pixel was not found, and out_i, out_j is not valid
48      *        ↪ .
49      */
50 }
```

7.5. APÉNDICE

```
42     bool ccw_not0_(const Image<int> &in, const std::size_t i0, const std::
43                     size_t j0, const std::size_t i, const std::size_t j, const unsigned
44                     char offset, std::size_t &out_i, std::size_t &out_j);
45
46     /**
47      * @brief Find the first 0-pixel in the neighbourhood of a pixel x0,y0.
48      *        ↪ Check neighbours rotating clockwise.
49      * @param in Image to analyse.
50      * @param i0 y position of the center pixel.
51      * @param j0 x position of the center pixel.
52      * @param i y position of the first neighbour to check.
53      * @param j x position of the first neighbour to check.
54      * @param offset neighbours to skip before actually checking. Regardless of
55      *        ↪ this value, all neighbours will be checked.
56      * @param out_i y position of the found pixel.
57      * @param out_j x position of the found pixel.
58      * @return true a valid pixel was found, and out_i, out_j is valid.
59      * @return false a valid pixel was not found, and out_i, out_j is not valid
60      *        ↪ .
61     */
62     bool cw_not0_(const Image<int> &in, const int i0, const int j0, const int i
63                     , const int j, const unsigned char offset, std::size_t &out_i, std::
64                     size_t &out_j);
65
66     /**
67      * @brief Follow a contour border while updating the Contour object.
68      *        ↪ Modifies the input image with the ID for the contour that is being
69      *        ↪ followed.
70      * @param in Contour image that will be read and updated as we follow the
71      *        ↪ contour.
72      * @param width Length of a row in the image.
73      * @param found_contour contour object that will be filled in with the
74      *        ↪ found border.
75      * @param i y position of the pixel that is beign analyzed.
76      * @param j x position of the pixel that is beign analyzed.
77      * @param i2 y position of the first neighbour to check.
78      * @param j2 x position of the first neighbour to check.
79      * @param nbd the id of the current border.
80      * @param lnbdr the id of the prebious encoutnered border.
81      */
82     void follow_border(Image<int> &in, std::size_t width, Extended_contour &
83                         found_contour, const std::size_t i, const std::size_t j, std::size_t
84                         i2, std::size_t j2, const int nbd, unsigned int &lnbdr);
85
86     bool neighbor_id_to_index_(const std::size_t i, const std::size_t j, const
87                               unsigned char id, std::size_t &out_i, std::size_t &out_j)
88     {
89         switch(id){
90             case 0: out_i = i ; out_j = j+1; return true;
91             case 1: out_i = i-1; out_j = j+1; return true;
92             case 2: out_i = i-1; out_j = j ; return true;
93             case 3: out_i = i-1; out_j = j-1; return true;
94             case 4: out_i = i ; out_j = j-1; return true;
95             case 5: out_i = i+1; out_j = j-1; return true;
96             case 6: out_i = i+1; out_j = j ; return true;
97             case 7: out_i = i+1; out_j = j+1; return true;
98             default: return false;
```

7.5. APÉNDICE

```
87         }
88     }
89
90     signed char neighbor_index_to_id_(const std::size_t i0, const std::size_t
91     ↪ j0, const std::size_t i, const std::size_t j)
92 {
93     signed char di = i - i0;
94     signed char dj = j - j0;
95
96     if (di == 0 && dj == 1) return 0;
97     if (di == -1 && dj == 1) return 1;
98     if (di == -1 && dj == 0) return 2;
99     if (di == -1 && dj == -1) return 3;
100    if (di == 0 && dj == -1) return 4;
101    if (di == 1 && dj == -1) return 5;
102    if (di == 1 && dj == 0) return 6;
103    if (di == 1 && dj == 1) return 7;
104
105    return -1;
106}
107
108 bool ccw_not0_(const Image<int> &in, const std::size_t i0, const std::size_t
109   ↪ j0, const std::size_t i, const std::size_t j, const unsigned
110   ↪ char offset, std::size_t &out_i, std::size_t &out_j)
111 {
112     signed char id = neighbor_index_to_id_(i0, j0, i, j);
113     std::size_t width = in.get_width();
114
115     for (std::size_t k = 0; k < 8; ++k)
116     {
117         int kk = (k + id + offset) % 8;
118         neighbor_id_to_index_(i0, j0, kk, out_i, out_j);
119
120         if (in[out_i * width + out_j] != 0) return true;
121     }
122     return false;
123 }
124
125 bool cw_not0_(const Image<int> &in, const int i0, const int j0, const int i
126   ↪ , const int j, const unsigned char offset, std::size_t &out_i, std::size_t
127   ↪ &out_j)
128 {
129     signed char id = neighbor_index_to_id_(i0, j0, i, j);
130     std::size_t width = in.get_width();
131
132     for (std::size_t k = 0; k < 8; ++k)
133     {
134         int kk = (-k + id - offset) % 8;
135         neighbor_id_to_index_(i0, j0, kk, out_i, out_j);
136
137         if (in[out_i * width + out_j] != 0) return true;
138     }
139     return false;
140 }
141
142 void contour_detection(Image<int> &conts_image, std::vector<
143   ↪ Extended_contour> &conts, bool trim_borders)
144 {
145 }
```

7.5. APÉNDICE

```
140 // Topological Structural Analysis of Digitized Binary Images by Border
141 //   ↪ Following.
142 // By Suzuki, S. and Abe, K. 1985
143 // Referece , by Lingdong Huang: https://github.com/LingDong-/PContour/
144 //   ↪ blob/master/src/pcontour/PContour.java
145
146 std::size_t height = conts_image.get_height(), width = conts_image.
147 //   ↪ get_width();
148
149 // Set the borders to zero. Required by the algorithm to avoid infinite
150 //   ↪ loops and indexing errors.
151 if(trim_borders)
152 {
153     for(std::size_t i = 0; i < height; ++i) conts_image[i*width] =
154 //   ↪ conts_image[(i+1)*width-1] = 0;
155     for(std::size_t i = 1; i < width-1; ++i) conts_image[i] =
156 //   ↪ conts_image[(height-1)*width+i] = 0;
157 }
158
159 int nbd = 1;           // Current contour
160 unsigned int lnbld = 1; // Last found contour
161
162 // NOTE: The great majority of the comments here are excerpts from the
163 //   ↪ original paper.
164
165 // Scan the picture with a TV raster and perform the following steps
166 //   ↪ for each pixel such that fij ≠ 0.
167 // Every time we begin to scan a new row of the picture, reset LNBD to
168 //   ↪ 1
169
170 for(std::size_t i = 1; i < height-1; ++i)
171 {
172     lnbld = 1;
173
174     for(std::size_t j = 1; j < width-1; ++j)
175     {
176         std::size_t i2 = 0, j2 = 0;
177
178         // If the pixel is not a contour edge, nothing to do.
179         if (conts_image[i*width + j] == 0) continue;
180
181         // (a) If fij = 1 and fi, j-1 = 0, then decide that the pixel (
182         //   ↪ i, j) is the border following
183         // starting point of an outer border, increment NBD, and (i2,
184         //   ↪ j2) <- (i, j - 1).
185         // (b) Else if fij ≥ 1 and fi, j+1 = 0, then decide that the
186         //   ↪ pixel (i, j) is the border following starting point of a
187         // hole border, increment NBD, (i2, j2) <- (i, j + 1), and LNBD
188         //   ↪ + fij in case fij > 1.
189         // (c) Otherwise, go to (4).
190
191         std::size_t curr_idx = i*width + j;
192         if (conts_image[curr_idx] == 1 && conts_image[curr_idx - 1] ==
193             //   ↪ 0)
194         {
195             ++nbd;
196             i2 = i;
197             j2 = j-1;
198         }
199 }
```

7.5. APÉNDICE

```
185     else if (conts_image[curr_idx] >= 1 && conts_image[curr_idx +
186         ↪ 1] == 0)
187     {
188         ++nbd;
189         i2 = i;
190         j2 = j+1;
191         if (conts_image[curr_idx] > 1) lnbda = conts_image[curr_idx
192             ↪ ];
193     }
194     else
195     {
196         // (4) If fij != 1, then LNBD <- |fij| and resume the
197         ↪ raster scan from pixel (i,j+1).
198         // The algorithm terminates when the scan reaches the lower
199         ↪ right corner of the picture
200
201         if (conts_image[curr_idx] != 1) lnbda = std::abs(conts_image
202             ↪ [curr_idx]);
203         continue;
204     }
205
206     // (2) Depending on the types of the newly found border
207     // and the border with the sequential number LNBD
208     // (i.e., the last border met on the current row),
209     // decide the parent of the current border.
210
211     // Lets start by filling in all the initial data of the contour
212     Extended_contour found_contour;
213     found_contour.id = nbd;
214     found_contour.is_hole = (j2 == j+1);
215     found_contour.n_pixels = 1;
216     found_contour.bb_br_x = j2;
217     found_contour.bb_br_y = i2;
218     found_contour.bb_tl_x = j2;
219     found_contour.bb_tl_y = i2;
220
221     // Now let's find it's parent.
222     // Let's assume there is no previous contour by default since
223     ↪ lnbda could not point at anything.
224
225     int prev_pos = -1;
226     for(std::size_t i = 0; i < conts.size(); ++i)
227     {
228         if(conts[i].id == lnbda) // Found contour corresponding to
229             ↪ lnbda
230         {
231             prev_pos = i;
232             break;
233         }
234
235         // If there is no previous contour, the current contour has no
236         ↪ parent.
237         if(prev_pos < 0) found_contour.parent = 0;
238         else if(conts[prev_pos].is_hole)
239         {
240             if(found_contour.is_hole) found_contour.parent = conts[
241                 ↪ prev_pos].parent;
242             else found_contour.parent = lnbda;
```

7.5. APÉNDICE

```
|235         }
|236     else
|237     {
|238         if(found_contour.is_hole) found_contour.parent = lnbdb;
|239         else found_contour.parent = conts[prev_pos].parent;
|240     }
|241
|242     // Follow the contour while updating it's data.
|243     follow_border(conts_image, width, found_contour, i, j, i2, j2,
|244                     ↪ nbd, lnbdb);
|245     conts.push_back(std::move(found_contour));
|246 }
|247 }
|248
|249 void follow_border(Image<int> &in, std::size_t width, ExtendedContour &
|250                     ↪ found_contour, const std::size_t i, const std::size_t j, std::size_t
|251                     ↪ i2, std::size_t j2, const int nbd, unsigned int &lnbdb)
|252 {
|253     // (3) From the starting point (i, j), follow the detected border:
|254     // this is done by the following substeps (3.1) through (3.5).
|255
|256     // (3.1) Starting from (i2, j2), look around clockwise the pixels
|257     // in the neighborhood of (i, j) and find a nonzero pixel.
|258     // Let (i1, j1) be the first found nonzero pixel. If no nonzero
|259     // pixel is found, assign -NBD to fij and go to (4).
|260
|261     std::size_t curr_idx = i*width + j;
|262     std::size_t i1 = 0, j1 = 0;
|263     if(!cw_not0_(in, i, j, i2, j2, 0, i1, j1))
|264     {
|265         in[curr_idx] = -nbd;
|266
|267         // Did not find a valid neighbor. Going to (4).
|268         // NOTE: We are not actually going to (4), just replicating step
|269         // ↪ (4) here again.
|270
|271         if(in[curr_idx] != 1) lnbdb = std::abs(in[curr_idx]);
|272         return;
|273     }
|274
|275     // (3.2) (i2, j2) ← (i1, j1) and (i3, j3) ← (i, j).
|276     i2 = i1;
|277     j2 = j1;
|278     std::size_t i3 = i;
|279     std::size_t j3 = j;
|280
|281     // (3.3) Starting from the next element of the pixel (i2, j2)
|282     // in the counterclockwise order, examine counterclockwise
|283     // the pixels in the neighborhood of the current pixel (i3, j3)
|284     // to find a nonzero pixel and let the first one be (i4, j4).
|285
|286     while(true)
|287     {
|288         std::size_t i4 = 0, j4 = 0;
|289         bool found_pair4 = ccw_not0_(in, i3, j3, i2, j2, 1, i4, j4);
```

7.5. APÉNDICE

```
|290
|291     {
|292         if(found_contour.bb_br_x <= j4) found_contour.bb_br_x = j4;
|293         else if(found_contour.bb_tl_x >= j4) found_contour.bb_tl_x = j4
|294             ↪ ;
|295         if(found_contour.bb_br_y <= i4) found_contour.bb_br_y = i4;
|296         else if(found_contour.bb_tl_y >= i4) found_contour.bb_tl_y = i4
|297             ↪ ;
|298     }
|299     found_contour.n_pixels++;
|300
|301     // (a) If the pixel (i3 , j3 + 1) is a O-pixel examined in the
|302     //      substep (3.3) then fi3 , j3 <- -NBD.
|303     // (b) If the pixel (i3 , j3 + 1) is not a O-pixel examined
|304     //      in the substep (3.3) and fi3 ,j3 = 1, then fi3 ,j3 <- NBD.
|305     // (c) Otherwise , do not change fi3 , j3.
|306
|307     std::size_t curr_idx3 = i3*width + j3;
|308     if (in[curr_idx3 + 1] == 0) in[curr_idx3] = -nbd;
|309     else if(in[curr_idx3] == 1) in[curr_idx3] = nbd;
|310
|311     // (3.5) If (i4 , j4) = (i , j) and (i3 , j3) = (i1 , j1)
|312     // (coming back to the starting point), then go to (4);
|313     // otherwise , (i2 , j2) + (i3 , j3),(i3 , j3) + (i4 , j4) , and go back
|314     //      ↪ to (3.3)
|315
|316     if(found_pair4)
|317     {
|318         if (i4 == i && j4 == j && i3 == i1 && j3 == j1)
|319         {
|320             // Followed the border completely , going to (4).
|321
|322             if (in[curr_idx] != 1) lnbda = std::abs(in[curr_idx]);
|323             return;
|324         }
|325         else
|326         {
|327             i2 = i3;
|328             j2 = j3;
|329             i3 = i4;
|330             j3 = j4;
|331         }
|332     }
|333 } // namespace imgutil
|334 } // namespace motdet
```

7.5.2 Código programa piloto Save_to_disk

7.5.2.1 CMakeLists.txt

```
1 cmake_minimum_required(VERSION 3.9.0)
2 project(motion_detector_driver VERSION 1.0.0 DESCRIPTION "Driver program for motion
   ↪ detector library")
```

7.5. APÉNDICE

```
3 set(DEFAULT_BUILD_TYPE "Release")
4
5 # Find the OpenCV package in the system
6 find_package(OpenCV REQUIRED)
7
8 # Add the main.cpp to the executable file
9 add_executable(${PROJECT_NAME} main.cpp)
10
11 # Link OpenCV to the executable
12 target_link_libraries(${PROJECT_NAME} ${OpenCV_LIBS})
13
14 # Link motion_detector library to the executable
15 target_link_libraries(${PROJECT_NAME} motion_detector)
16
17 # Link the filesystem library
18 target_link_libraries(${PROJECT_NAME} stdc++fs)
19
20 # Set compiler flags. Tell it to treat warnings as errors , pedantic and c++11
21 target_compile_options(${PROJECT_NAME} PRIVATE -Werror -pedantic)
22 target_compile_features(${PROJECT_NAME} PRIVATE cxx_std_17)
```

7.5.2.2 main.cpp

```
1 #include <iostream>
2 #include <cstddef>
3 #include <filesystem>
4 #include <cstring>
5 #include <vector>
6 #include <chrono>
7 #include <exception>
8
9 #include <motion_detector.hpp>
10 #include <opencv2/core.hpp> // OpenCV is used to capture frames to feed into
11 // → the motion detector
12 #include <opencv2/videoio.hpp>
13 #include <opencv2/imgproc.hpp>
14
15 namespace fs = std::filesystem; // Rename namespaces for convenience.
16 namespace md = motdet;
17
18 using std::chrono::high_resolution_clock;
19 using std::chrono::duration_cast;
20 using std::chrono::duration;
21 using std::chrono::milliseconds;
22
23 int main(int argc, char** argv)
24 {
25     if(argc < 3)
26     {
27         std::cerr <<
28             "ERROR: Missing parameters" << std::endl <<
29             "Specify the following parameters: " << std::endl <<
30             " - input: Either 'camera' which will capture from connected USB cam or a
31             // → path to a .mp4" << std::endl <<
```

7.5. APÉNDICE

```
30     " - output: Folder to store captured video in. Will create .mp4 files with
31     ↪ date when motion is detected. NOTE: All files are deleted in the
32     ↪ directory." << std::endl <<
33     " [OPTIONAL] - threads : Amount of threads, by default, 1." << std::endl <<
34     " [OPTIONAL] - downscale factor : Resolution reductor for input video, 4 is
35     ↪ fastest while still working. 1 is native resolution." << std::endl
36     ↪ <<
37     " [OPTIONAL] - display stats : Print timing stats. 1 for true, 0 for false"
38     ↪ << std::endl;
39
40     return 0;
41 }
42
43     std::string in_source;
44     std::string out_dir;
45     unsigned int threads = 1;
46     unsigned int reduction_factor = 1;
47     bool display_stats = false;
48
49     // Get input source
50     in_source = argv[1];
51     if(in_source != "camera")
52     {
53         if(!fs::exists(in_source)) throw std::invalid_argument("The selected input
54         ↪ video file is not accessible or does not exist.");
55     }
56
57     // Get output directory
58     out_dir = argv[2];
59     fs::path out_recordings_dir(out_dir);
60     if(!fs::exists(out_recordings_dir)) throw std::invalid_argument("The selected
61     ↪ output directory is not accessible or does not exist.");
62     fs::remove_all(out_recordings_dir / "*");    // Clear the output directory.
63
64     // Get threads
65     try { if(argc >= 4) threads = std::abs(std::stoi(argv[3])); }
66     catch(const std::exception &e) { throw std::invalid_argument("The number of
67     ↪ threads must be a number."); }
68     if(threads == 0) throw std::invalid_argument("The number of threads must be at
69     ↪ least 1");
70
71     // Get reduction factor
72     try { if(argc >= 5) reduction_factor = std::abs(std::stoi(argv[4])); }
73     catch(const std::exception &e) { throw std::invalid_argument("The reduction
74     ↪ factor must be a number."); }
75     if(reduction_factor == 0) throw std::invalid_argument("The reduction factor
76     ↪ must be at least 1");
77
78     // Get display stats
79     if(argc >= 6) display_stats = std::stoi(argv[5]) == 1 ? true : false;
80
81     // At this point, all input parameters have been validated and stored.
82
83     cv::VideoCapture cap;
84
85     // Open either a connected camera or an input video.
86     if(in_source == "camera")
87     {
88         int device_id = 0;                      // 0 = open default camera
```

7.5. APÉNDICE

```
78     int api_id = cv::CAP_ANY;           // 0 = autodetect default API
79     cap.open(device_id, api_id);
80     std::cout << "Using camera input..." << std::endl;
81 }
82 else
83 {
84     cap.open(in_source);
85     std::cout << "Using .mp4 input..." << std::endl;
86 }
87 if (!cap.isOpened()) throw std::runtime_error("Failed opening input.");
88
89 // When motion is detected a new .mp4 video will be created in the output
//   ↪ directory.
90 // In order to do this, create a .mp4 writer from OpenCV.
91
92 cv::VideoWriter writer;
93 int codec = cv::VideoWriter::fourcc('m', 'p', '4', 'v'); // Saving to .mp4
94 unsigned char fps = cap.get(cv::CAP_PROP_FPS);
95 std::size_t width = cap.get(cv::CAP_PROP_FRAME_WIDTH);
96 std::size_t height = cap.get(cv::CAP_PROP_FRAME_HEIGHT);
97 cv::Size stream_res = cv::Size(width, height);
98
99 // Create the Motion detector object from the library.
100 // Specify an input queue 2 times the amount of threads, so that threads are
//   ↪ always busy working.
101 md::Motion_detector motion_detector(width, height, threads, threads*2,
//   ↪ reduction_factor);
102
103 // We will record for a few extra seconds after motion is no longer detected to
//   ↪ avoid videos turning off and on intermittently.
104 unsigned long long millis_keep_recording = 3000;
105 unsigned long long millis_last_movement = 0;
106 unsigned long long millis_prev_frame = 0, millis_frame = 0;
107 unsigned int recordings_counter = 0; // We will create a separate video for
//   ↪ each time movement is detected, use a counter to sequence videos.
108 bool recording = false, input_video_ended = false, first_iteration = true;
109
110 if(display_stats) auto t_video0 = high_resolution_clock::now();
111
112 // Start grabbing frames and checking for motion. Store the captured frame in a
//   ↪ OpenCV Matrix (Mat).
113
114 std::vector<motdet::Contour> contours; // Storage for detected movement across
//   ↪ frames.
115
116 auto video_processing_start = high_resolution_clock::now(); // Chrono the time
//   ↪ it takes to process the input source.
117 while(true)
118 {
119     // Only keep grabbing new frames as long as the input source has frames. A
//       ↪ camera will not run out, but a video will.
120     if(!input_video_ended)
121     {
122         // Create a container for the new frame that we are going to read, and
//           ↪ fill it from the input source.
123         std::shared_ptr<cv::Mat> frame = std::make_unique<cv::Mat>();
124         cap.read(*frame.get());
125
126         if(first_iteration)
```

7.5. APÉNDICE

```
127     {
128         millis_prev_frame = millis_frame = 0;
129         first_iteration = false;
130     }
131     else
132     {
133         millis_prev_frame = millis_frame;
134         millis_frame = cap.get(cv::CAP_PROP_POS_MSEC);
135     }
136
137     if(frame->empty() || millis_frame < millis_prev_frame)
138     {
139         std::cout << "Reached end of video, waiting for all frames to be
140             processed ..." << std::endl;
141         input_video_ended = true;
142     }
143     else
144     {
145         // Turn the captured frame from RGB to grayscale. First create a
146             // new grayscale image wrapped in a std::unique_ptr
147         auto grayscale_input_frame = std::make_unique<md::Image<unsigned
148             // short>>(width, height, 0);
149
150         // Get the pointer to the input RGB data, and make the conversion
151             // from the rgb to grayscale.
152         unsigned char *rgb_data = (unsigned char *)frame->data;
153         md::uchar_to_bw(rgb_data, frame->total(), *grayscale_input_frame.
154             // get());
155
156         // Enqueue this new grayscale image to the motion detector,
157             // transferring the ownership of the Image away.
158         // We also sent the shared ptr with the frame we read from source.
159         // The reason for this is because we want to get it back when
160             // polling for results later so that we can
161             // use it to record a video if motion is detected.
162         motion_detector.enqueue_frame(std::move(grayscale_input_frame),
163             // millis_frame, true, frame);
164     }
165     }
166     else
167     {
168         // Once the video is ended, we need to keep looping until the queued
169             // tasks are done.
170         if(motion_detector.get_task_queue_size() == 0 && motion_detector.
171             // get_result_queue_size() == 0) break;
172     }
173
174     // Now poll for results and if there are, check whether we need to start
175         // recording or not.
176     // If we simply read a frame from the source, enqueue it, and wait for the
177         // frame to be done to know if there
178         // is movement to start recording or not we will not be taking advantage of
179             // the threads in the motion detector.
180     // Instead we can push a new frame each iteration and try to poll in non-
181         // blocking mode, if there is no results
182         // we simply continue to the next iteration, filling up the queue and
183             // takign advantage of the concurrency.
184
185     md::Detection detected_result;
```

7.5. APÉNDICE

```
171     bool result_available = true;
172
173     try{ detected_result = motion_detector.get_detection(false); } // Non-
174         ↪ blocking result request
175     catch(const std::runtime_error &e){ result_available = false; };
176
177     if(result_available)
178     {
179         // Process the detected movement contours. Start or stop recording
180         ↪ accordingly.
181
182         if(recording) std::cout << "[REC] ";
183         std::cout << "Got results for " << detected_result.timestamp << ".";
184         if(display_stats) std::cout << " | Processing time: " <<
185             ↪ detected_result.processing_time << " milliseconds." << std::endl;
186         else std::cout << std::endl;
187
188         if(detected_result.has_detections)
189         {
190             if(!recording)
191             {
192                 // First movement detected in a while, start recording
193
194                 fs::path file("motion_" + std::to_string(recordings_counter) +
195                     ↪ ".mp4");
196                 fs::path rec_path = out_recordings_dir / file;
197
198                 std::cout << "Motion detected. Recording to " << rec_path <<
199                     ↪ std::endl;
200                 writer.open(rec_path, codec, fps, stream_res, true); // Color
201                     ↪ video is assumed.
202             }
203
204             millis_last_movement = detected_result.timestamp;
205             contours = detected_result.detection_contours;
206             recording = true;
207         }
208         else
209         {
210             if(recording && millis_last_movement + millis_keep_recording <
211                 ↪ detected_result.timestamp)
212             {
213                 // No movement detected in feed after a while, closing
214                 ↪ recording.
215
216                 std::cout << "No motion detected. Closing recording " <<
217                     ↪ recordings_counter << std::endl;
218                 ++recordings_counter;
219
220                 contours.clear();
221                 recording = false;
222                 writer.release();
223             }
224         }
225
226         if(recording)
227         {
228             // Remember how we sent the raw frame as a shared_ptr when
229             ↪ enqueueing? Recover it now.
```

7.5. APÉNDICE

```
220     cv::Mat* recovered_frame = (cv::Mat*)detected_result.data_keep.get
221     ↪ ();
222
223     // Draw the detected motion onto the recovered frame as rectangles.
224     for(md::Contour &cont : contours)
225     {
226         // Draw the detected movement on screen
227
228         cv::Point pt1(cont.bb_tl_x, cont.bb_tl_y);
229         cv::Point pt2(cont.bb_br_x, cont.bb_br_y);
230         cv::rectangle(*recovered_frame, pt1, pt2, cv::Scalar(0, 255, 0)
231           ↪ );
232     }
233
234     // Save the edited frame into the video we are recording.
235     writer.write(*recovered_frame);
236   }
237
238   if(display_stats)
239   {
240     auto video_processing_end = high_resolution_clock::now();
241     std::cout << "Processed video in " << duration_cast<milliseconds>(
242       ↪ video_processing_end - video_processing_start).count() << " millis " <<
243       ↪ std::endl;
244   }
245   else
246   {
247     std::cout << "Finished processing video, exiting..." << std::endl;
248   }
249   return 0;
250 }
```

7.5.3 Código librería HLS final

7.5.3.1 main.cpp

```
1 #include <iostream>
2 #include <cstddef>
3 #include <cstring>
4 #include <vector>
5 #include <chrono>
6 #include <exception>
7
8 #include "motion_detector.hpp"
9 #include <opencv2/core.hpp>      // OpenCV is used to capture frames to feed into
10    ↪ the motion detector
11 #include <opencv2/videoio.hpp>
12 #include <opencv2/imgproc.hpp>
13 int main(int argc, char** argv)
14 {
```

7.5. APÉNDICE

```
15 // We will follow the same dynamic as in previous test main functions,
16 // but here we wont record the result, just inform it is recording so that we
17 // → know its working.
18 std::string in_source = "~/.motdet/example_results/in_test_motion.mp4";
19 std::string out_recordings_dir = "~/.motdet/example_results/";
20 cv::VideoCapture cap;
21 cap.open(in_source);
22
23 if (!cap.isOpened()) throw std::runtime_error("Failed opening input.");
24
25 // We will record for a few extra seconds after motion is no longer detected to
26 // → avoid videos turning off and on intermittently.
27 unsigned long long millis_keep_recording = 3000;
28 unsigned long long millis_last_movement = 0;
29 unsigned long long millis_prev_frame = 0, millis_frame = 0;
30 unsigned int recordings_counter = 0; // We will create a separate video for
31 // → each time movement is detected, use a counter to sequence videos.
32 bool recording = false, first_iteration = true;
33
34 cv::VideoWriter writer;
35 int codec = cv::VideoWriter::fourcc('m', 'p', '4', 'v'); // Saving to .mp4
36 unsigned char fps = cap.get(cv::CAP_PROP_FPS);
37 std::size_t width = cap.get(cv::CAP_PROP_FRAME_WIDTH);
38 std::size_t height = cap.get(cv::CAP_PROP_FRAME_HEIGHT);
39 cv::Size stream_res = cv::Size(width, height);
40
41 std::vector<motdet::Contour> detected.Conts;
42
43 // Start grabbing frames and checking for motion. Store the captured frame is a
44 // → OpenCV Matrix (Mat).
45 while(true)
46 {
47     // Create a container for the new frame that we are going to read, and fill
48     // → it from the input source.
49     cv::Mat frame;
50     cap.read(frame);
51
52     if(first_iteration)
53     {
54         millis_prev_frame = millis_frame = 0;
55         first_iteration = false;
56     }
57     else
58     {
59         millis_prev_frame = millis_frame;
60         millis_frame = cap.get(cv::CAP_PROP_POS_MSEC);
61     }
62
63     if(frame.empty() || millis_frame < millis_prev_frame) break; // Check for
64     // → the end of the input video.
65
66     if(millis_frame < 9500 || millis_frame > 20000) continue;
67
68     // Video did not end, get the data from the frame.
69
70     unsigned char *rgb_data = (unsigned char *)frame.data;
```

7.5. APÉNDICE

```
67     hls::stream<motdet::Packed_pix, MOTDET_STREAM_DEPTH> &image_in = *(new hls
68         ↪ ::stream<motdet::Packed_pix, MOTDET_STREAMDEPTH>());
69     hls::stream<motdet::Streamed_contour, MOTDET_STREAM_DEPTH> &conts_out = *(  

70         ↪ new hls::stream<motdet::Streamed_contour, MOTDET_STREAMDEPTH>("br/>
71             ↪ out_main"));
72
73     // Turn the RGB image to grayscale before sending to FPGA
74     for(uint32_t i = 0; i < ORIGINAL_TOTAL; i += MOTDET_REDUCTION_FACTOR)
75     {
76         motdet::Packed_pix packed;
77         for(uint8_t k = 0; k < MOTDET_REDUCTION_FACTOR; ++k)
78         {
79             uint32_t mapped = (i+k)*3;
80             packed.pix[k] = rgb_data[mapped] * 76.245 + rgb_data[mapped+1] *  

81                 ↪ 149.685 + rgb_data[mapped+2] * 29.07;
82         }
83         image_in.write(packed);
84     }
85
86     detect_motion(image_in, conts_out); // Submit frame to FPGA for processing.
87     motdet::Streamed_contour cont = conts_out.read();
88
89     // Process the detected movement contours. Start or stop recording
90     ↪ accordingly.
91
92     if(recording) std::cout << "[REC] ";
93     std::cout << "Got results for " << millis_frame << "." << std::endl;
94
95     if(!cont.stream_end)
96     {
97         if(!recording)
98         {
99             // First movement detected in a while, start recording
100            std::string file = "motion_" + std::to_string(recordings_counter) +  

101                ↪ ".mp4";
102            std::string rec_path = out_recordings_dir + file;
103            writer.open(rec_path, codec, fps, stream_res, true);
104
105            std::cout << "Motion detected. Recording..." << std::endl;
106        }
107        millis_last_movement = millis_frame;
108        recording = true;
109
110        detected_conts.clear();
111        while(!cont.stream_end){
112            detected_conts.push_back(std::move(cont.contour));
113            cont = conts_out.read();
114        }
115    }
116    else
117    {
118        if(recording && millis_last_movement + millis_keep_recording <  

119            ↪ millis_frame)
120        {
121            // No movement detected in feed after a while, closing recording.
122            std::cout << "No motion detected. Closing recording " <<  

123                ↪ recordings_counter << std::endl;
124    }
```

7.5. APÉNDICE

```
118         ++recordings_counter;
119         recording = false;
120         writer.release();
121     }
122 }
123
124 if (recording)
125 {
126     // Draw the detected motion onto the recovered frame as rectangles.
127     for(motdet::Contour &cont : detected_conts)
128     {
129         // Draw the detected movement on screen
130
131         cv::Point pt1(cont.bb_tl_x, cont.bb_tl_y);
132         cv::Point pt2(cont.bb_br_x, cont.bb_br_y);
133         cv::rectangle(frame, pt1, pt2, cv::Scalar(0, 255, 0));
134     }
135
136     // Save the edited frame into the video we are recording.
137     writer.write(frame);
138 }
139
140 return 0;
141
142 }
```

7.5.3.2 motion_detector.hpp

```
1 #ifndef _MOTDET_MOTION_DETECTOR_HPP_
2 #define _MOTDET_MOTION_DETECTOR_HPP_
3
4 #include <cstdint>
5 #include "hls_math.h"
6 #include "hls_stream.h"
7 #include "ap_int.h"
8
9 #define MOTDET_STREAM_DEPTH 1
10
11 #define ORIGINAL_WIDTH 1920 // (11b) 32
12 #define ORIGINAL_HEIGHT 1080 // (11b) 32
13 #define ORIGINAL_TOTAL 2073600 // (21b) 1024
14
15 // How to calculate: motdet_width = ceil( original_width/reduction_factor )
16 #define MOTDET_WIDTH 480 // (9b) 8
17 #define MOTDET_HEIGHT 270 // (9b) 8
18 #define MOTDET_TOTAL 129600 // (17b) 64
19
20 #define MOTDET_MAX_CONTOURS 1023
21 #define MOTDET_REDUCTION_FACTOR 4
22
23 namespace motdet
24 {
25     const float motdet_frame_update_ratio = 0.0067;
26     const ap_uint<15> motdet_threshold = 23500;
```

7.5. APÉNDICE

```
27     const ap_uint<12> motdet_min_cont_area = MOTDET_TOTAL*0.004+5;
28
29     // Stream depths
30
31     const ap_uint<1> stream_depth = 1;
32
33     struct Contour
34     {
35         ap_uint<11> bb_tl_x , bb_tl_y; /*< Top left point of the bounding box of
36                                         → the Contour */
37         ap_uint<11> bb_br_x , bb_br_y; /*< Bottom right point of the bounding box
38                                         → of the Contour */
39     };
40
41     struct Streamed_contour
42     {
43         Contour contour;
44         bool stream_end;
45     };
46
47     struct Contour_package
48     {
49         Contour contours [MOTDET.MAX_CONTOURS];
50         ap_uint<10> contour_count = 0;
51     };
52
53     struct Packed_pix
54     {
55         ap_uint<16> pix [MOTDET.REDUCTION_FACTOR];
56     };
57
58 } // namespace motdet
59 /**
60 * @brief Detects motion in a sequence of grayscale frames. HLS TOP FUNCTION.
61 * @param in grayscale streamed image where each pixel is represented by a 16b
62 *        → unsigned integer. The higher, the more intense white.
63 * @param out Set of contours that have been detected as movement.
64 */
65 void detect_motion( hls :: stream<motdet::Packed_pix , MOTDET_STREAM_DEPTH> &in , hls ::
66                     → stream<motdet::Streamed_contour , MOTDET_STREAM_DEPTH> &out );
67
68 #endif // _MOTDET_MOTION_DETECTOR_HPP_
```

7.5.3.3 motion_detector.cpp

```
1 #include "motion_detector.hpp"
2
3 #include "image_utils.hpp"
4 #include "contour_detector.hpp"
5
6 void detect_motion( hls :: stream<motdet::Packed_pix , MOTDET_STREAM_DEPTH> &in , hls ::
6                     → stream<motdet::Streamed_contour , MOTDET_STREAM_DEPTH> &out )
7 {
8 #pragma HLS INTERFACE ap_fifo port=in
```

7.5. APÉNDICE

```
9 #pragma HLS INTERFACE ap_fifo port=out
10 #pragma HLS DATAFLOW
11 using namespace motdet;
12
13 hls::stream<ap_uint<16>, MOTDET_STREAM_DEPTH> downsampled("downsampled");
14 hls::stream<ap_uint<16>, MOTDET_STREAM_DEPTH> blurred("blurred");
15 hls::stream<ap_uint<16>, MOTDET_STREAM_DEPTH> subbed("subbed");
16 hls::stream<ap_uint<1>, MOTDET_STREAM_DEPTH> thresholded("thresholded");
17 hls::stream<ap_uint<1>, MOTDET_STREAM_DEPTH> dilated("dilated");
18
19 // Downsample the grayscale image.
20 imgutil::downsample(in, downsampled);
21
22 imgutil::gaussian_blur(downsampled, blurred);
23
24 // Interpolate the blurred image and the reference frame to obtain a new
25 // → reference.
26 // Also subtract the blurred frame with the reference image. Leaving only the
27 // → changes between frames.
28 imgutil::apply_reference(blurred, subbed);
29
30 // Threshold the image so that any value below a certain number is ignored.
31 imgutil::single_threshold(subbed, thresholded);
32
33 // Dilate the image so that the contours are better defined and with less holes
34 // → .
35 imgutil::dilation(thresholded, dilated);
36 // Detect contours in the image. Any contour detected here is "movement".
37 imgutil::connected_components(dilated, out);
38 }
```

7.5.3.4 image_utils.hpp

```
1 #ifndef __MOTDET_IMAGE_UTILS_HPP__
2 #define __MOTDET_IMAGE_UTILS_HPP__
3
4 #include "motion_detector.hpp"
5 #include "hls_stream.h"
6 #include "ap_int.h"
7
8 #include <cstdint>
9
10 namespace motdet
11 {
12     namespace imgutil
13     {
14         /**
15          * @brief Apply a 5x5 blurring filter to an image using a split kernel.
16          * @param in Grayscale streamed image to blur.
17          * @param out Grayscale streamed blurred image.
18          */
19         void gaussian_blur(hls::stream<ap_uint<16>, MOTDET_STREAM_DEPTH> &in, hls::
20                           //→ stream<ap_uint<16>, MOTDET_STREAM_DEPTH> &out);
```

7.5. APÉNDICE

```
21 /**
22 * @brief Resizes to a lower resolution by a given factor. Ignores floating
23 *       ↪ point precision.
24 * @param in Streamed image to resize, will use the sizes original_width
25 *       ↪ and original_height, with packed pixels.
26 * @param out Resized image.
27 */
28 void downsample(hls::stream<motdet::Packed_pix, MOTDET_STREAM_DEPTH> &in,
29                 ↪ hls::stream<ap_uint<16>, MOTDET_STREAM_DEPTH> &out);
30 /**
31 * @brief Gets the absolute difference between an image and the reference
32 *       ↪ image, while updating it.
33 * @param in Grayscale streamed image to subtract.
34 * @param out Subtracted grayscale image, values are always positive.
35 */
36 void apply_reference(hls::stream<ap_uint<16>, MOTDET_STREAM_DEPTH> &in, hls
37                 ↪ ::stream<ap_uint<16>, MOTDET_STREAM_DEPTH> &out);
38 /**
39 * @brief Collapses all the values in a grayscale image to the states
40 *       ↪ Culled 0 and Strong 1 depending on a threshold.
41 * @param in Streamed image to collapse. Used motdet_threshold.
42 * @param out Streamed image of the collapsed states.
43 */
44 void single_threshold(hls::stream<ap_uint<16>, MOTDET_STREAM_DEPTH> &in,
45                      ↪ hls::stream<ap_uint<1>, MOTDET_STREAM_DEPTH> &out);
46 /**
47 * @brief Takes a binary image (0 or 1) and dilates the 1-pixels.
48 * @param in Streamed binary image to process.
49 * @param out Streamed dilated binary image.
50 */
51 void dilation(hls::stream<ap_uint<1>, MOTDET_STREAM_DEPTH> &in, hls::stream
52                 ↪ <ap_uint<1>, MOTDET_STREAM_DEPTH> &out);
53 } // namespace imgutil
54 } // namespace motdet
55 #endif // __MOTDET_IMAGE_UTILS_HPP__
```

7.5.3.5 image_utils.cpp

```
1 #include "image_utils.hpp"
2
3 #include <iostream>
4
5 namespace motdet
6 {
7     namespace imgutil
8     {
9         namespace // Anonymous namespace
10        {
11             ap_uint<16> motdet_reference[MOTDET_HEIGHT][MOTDET_WIDTH];
12             ap_uint<7> gaussian_kernel[5] = { 16, 62, 99, 62, 16 }; // Total 255
13         }
14     }
15 }
```

7.5. APÉNDICE

```
13         bool has_reference = false;
14     } // Anonymous namespace
15
16     void gaussian_blur_filter_vline(hls::stream<ap_uint<16>,
17                                     ↪ MOTDET_STREAM_DEPTH> &in, hls::stream<ap_uint<16>,
18                                     ↪ MOTDET_STREAM_DEPTH> &out)
19     {
20 #ifndef __SYNTHESIS__
21         ap_uint<16>** buffer = new ap_uint<16>*[5];
22         for(ap_uint<3> i = 0; i < 5; ++i) buffer[i] = new ap_uint<16>[
23             ↪ MOTDET_WIDTH];
24 #else
25         ap_uint<16> buffer[5][MOTDET_WIDTH];
26 #pragma HLS ARRAY_PARTITION variable=buffer dim=1 complete
27 #endif
28
29         ap_uint<3> buffer_ptr = 0;
30
31         // The first row we read will need to fill out the 2 extra pixels
32         // → outside the border of the image.
33         for(ap_uint<9> j = 0; j < MOTDET_WIDTH; ++j)
34         {
35 #pragma HLS PIPELINE
36             ap_uint<16> curr_val = in.read();
37             buffer[3][j] = curr_val;
38             buffer[4][j] = curr_val;
39             buffer[0][j] = curr_val;
40
41         // The second row just needs to be written to the buffer, but we cannot
42         // → still output results because we only have 4 rows out of 5.
43         for(ap_uint<9> j = 0; j < MOTDET_WIDTH; ++j){
44 #pragma HLS PIPELINE
45             buffer[1][j] = in.read();
46
47         // Now iterate over the rest of the rows, now each row we get, we can
48         // → output results.
49         for(ap_uint<9> i = 2; i < MOTDET_HEIGHT; ++i)
50         {
51             buffer_ptr = i%5;
52             for(ap_uint<9> j = 0; j < MOTDET_WIDTH; ++j)
53             {
54 #pragma HLS PIPELINE
55                 buffer[buffer_ptr][j] = in.read();
56
57                 ap_uint<24> res = 0;
58                 for(ap_uint<3> k = 0; k < 5; ++k) res += buffer[(buffer_ptr+k
59                     // → +1)%5][j] * gaussian_kernel[k];
60                 out.write(res/255);
61             }
62
63             // Now we have iterated the whole image, but we only have outputted
64             // → MOTDET_HEIGHT-2 rows.
65             // Output those last 2 rows now extending the pixels at the border of
66             // → the image.
67             for(ap_uint<9> j = 0; j < MOTDET_WIDTH; ++j){
```

7.5. APÉNDICE

```
63 #pragma HLS PIPELINE
64     buffer[buffer_ptr][j] = buffer[(buffer_ptr-1)%5][j];
65
66     ap_uint<24> res = 0;
67     for(ap_uint<3> k = 0; k < 5; ++k) res += buffer[(buffer_ptr+k+1)
68         ↪ %5][j] * gaussian_kernel[k];
69     out.write(res/255);
70 }
71
71     for(ap_uint<9> j = 0; j < MOTDET_WIDTH; ++j){
72 #pragma HLS PIPELINE
73     buffer[(buffer_ptr+1)%5][j] = buffer[buffer_ptr][j];
74
75     ap_uint<24> res = 0;
76     for(ap_uint<3> k = 0; k < 5; ++k) res += buffer[(buffer_ptr+k+2)
77         ↪ %5][j] * gaussian_kernel[k];
78     out.write(res/255);
79 }
79
80 #ifndef __SYNTHESIS__
81     for(ap_uint<3> i = 0; i < 5; ++i) delete[] buffer[i];
82     delete[] buffer;
83 #endif
84 }
85
86 void gaussian_blur_filter_hline(hls::stream<ap_uint<16>,
87     ↪ MOTDET_STREAM_DEPTH> &in, hls::stream<ap_uint<16>,
88     ↪ MOTDET_STREAM_DEPTH> &out)
89 {
90     ap_uint<16> buffer[5];
91     ap_uint<24> res = 0;
92
93     for(ap_uint<9> i = 0; i < MOTDET_HEIGHT; ++i)
94     {
95         ap_uint<16> init_val = in.read();
96
97         buffer[3] = init_val;
98         buffer[4] = init_val;
99         buffer[0] = init_val;
100        buffer[1] = in.read();
101        buffer[2] = in.read();
102
103    #pragma HLS PIPELINE
104        res = 0;
105        for(ap_uint<3> k = 0; k < 5; ++k) res += buffer[(j+k)%5] *
106            ↪ gaussian_kernel[k];
107        out.write(res/255);
108
109        buffer[j%5] = in.read();
110    }
111
112        res = 0;
113        for(ap_uint<3> k = 0; k < 5; ++k)
114    {
114 #pragma HLS PIPELINE
115        res += buffer[(MOTDET_WIDTH+k)%5] * gaussian_kernel[k];
116    }
```

7.5. APÉNDICE

```
117         out.write(res/255);
118
119         buffer[MOTDET.WIDTH%5] = buffer[(MOTDET.WIDTH-1)%5];
120         res = 0;
121         for(ap_uint<3> k = 0; k < 5; ++k)
122         {
123 #pragma HLS PIPELINE
124             res += buffer[(MOTDET.WIDTH+k+1)%5] * gaussian_kernel[k];
125         }
126         out.write(res/255);
127
128         buffer[(MOTDET.WIDTH+1)%5] = buffer[MOTDET.WIDTH%5];
129         res = 0;
130         for(ap_uint<3> k = 0; k < 5; ++k)
131         {
132 #pragma HLS PIPELINE
133             res += buffer[(MOTDET.WIDTH+k+2)%5] * gaussian_kernel[k];
134         }
135         out.write(res/255);
136     }
137 }
138
139 void gaussian_blur(hls::stream<ap_uint<16>, MOTDET_STREAMDEPTH> &in, hls::
140     ↪ stream<ap_uint<16>, MOTDET_STREAMDEPTH> &out)
141 {
142 #pragma HLS DATAFLOW
143     hls::stream<ap_uint<16>, MOTDET_STREAMDEPTH> half_blurred("half-
144     ↪ blurred");
145
146 // An NxN gaussian blur can be decomposed into 2 1-dimensional kernels,
147 // ↪ N vertical and N horizontal.
148 gaussian.blur.filter_vline(in, half_blurred);
149 gaussian.blur.filter_hline(half_blurred, out);
150
151 void downsample(hls::stream<motdet::Packed_pix, MOTDET_STREAMDEPTH> &in,
152     ↪ hls::stream<ap_uint<16>, MOTDET_STREAMDEPTH> &out)
153 {
154 #ifndef __SYNTHESIS__
155     ap_uint<20> *buffer = new ap_uint<20>[MOTDET.WIDTH];
156 #else
157     ap_uint<20> buffer[MOTDET.WIDTH];
158 #endif
159
160     ap_uint<5> squared_red_factor = MOTDET_REDUCTION_FACTOR*
161     ↪ MOTDET_REDUCTION_FACTOR;
162
163     for(ap_uint<9> j = 0; j < MOTDET.WIDTH; ++j) buffer[j] = 0;
164
165     for(ap_uint<11> i = 0; i < ORIGINAL_HEIGHT; ++i)
166     {
167 // Keep filling the buffer
168         for(ap_uint<9> motdet_j = 0; motdet_j < MOTDET.WIDTH; ++motdet_j)
169         {
170 #pragma HLS PIPELINE
171         motdet::Packed_pix packed = in.read();
172         ap_uint<18> total = 0;
```

7.5. APÉNDICE

```
170     for(ap_uint<3> k = 0; k < MOTDET.REDUCTION_FACTOR; ++k) total
171         ↪ += packed.pix[k];
172     buffer[motdet_j] += total;
173 }
174 // Processed the last line into the buffer , output.
175 if(!((i+1) % MOTDET.REDUCTION_FACTOR))
176 {
177     // Line that completes the buffer , start outputting
178     for(ap_uint<9> motdet_j = 0; motdet_j < MOTDET.WIDTH; ++
179         ↪ motdet_j)
180     {
181 #pragma HLS PIPELINE
182         out.write(buffer[motdet_j]/squared_red_factor);
183         buffer[motdet_j] = 0;
184     }
185 }
186
187 #ifndef __SYNTHESIS__
188     delete[] buffer;
189 #endif
190 }
191
192 void apply_reference(hls::stream<ap_uint<16>, MOTDET.STREAM_DEPTH> &in, hls
193     ↪ ::stream<ap_uint<16>, MOTDET.STREAM_DEPTH> &out)
194 {
195     float update_ratio = has_reference ? motdet_frame_update_ratio : 1.0;
196     has_reference = true;
197
198     for(ap_uint<9> i = 0; i < MOTDET.HEIGHT; ++i)
199     {
200         for(ap_uint<9> j = 0; j < MOTDET.WIDTH; ++j)
201     #pragma HLS PIPELINE
202         ap_uint<16> from_val = motdet_reference[i][j];
203         ap_uint<16> to_val = in.read();
204         ap_uint<16> new_val = from_val + update_ratio * (to_val -
205             ↪ from_val); // Simplified from equation: from*(1-ratio) +
206             ↪ to*ratio
207         motdet_reference[i][j] = new_val;
208
209         out.write(hls::abs(to_val - new_val));
210     }
211 }
212
213 void single_threshold(hls::stream<ap_uint<16>, MOTDET.STREAM_DEPTH> &in,
214     ↪ hls::stream<ap_uint<1>, MOTDET.STREAM_DEPTH> &out)
215 {
216     for(ap_uint<17> i = 0; i < MOTDET.TOTAL; ++i){
217 #pragma HLS PIPELINE
218         out.write(in.read() > motdet_threshold ? 1 : 0);
219     }
220
221 void dilation_vline(hls::stream<ap_uint<1>, MOTDET.STREAM_DEPTH> &in, hls::
222     ↪ stream<ap_uint<1>, MOTDET.STREAM_DEPTH> &out)
223 }
```

7.5. APÉNDICE

```
|222 #ifndef __SYNTHESIS__
|223     ap_uint<1>** buffer = new ap_uint<1>*[2];
|224     for(ap_uint<2> i = 0; i < 2; ++i) buffer[i] = new ap_uint<1>[
|225         MOTDET_WIDTH];
|226     ap_uint<1> buffer[2][MOTDET_WIDTH];
|227 #endif
|228     ap_uint<3> buffer_ptr;
|229
|230     // The first row we read will need to fill out the pixels outside the
|231     // border of the image as if they were 0.
|232     for(ap_uint<9> j = 0; j < MOTDET_WIDTH; ++j){
|233 #pragma HLS PIPELINE
|234         buffer[1][j] = 0;
|235     }
|236
|237     // The second row just needs to be written to the buffer , but we cannot
|238     // still output results because we only have 2 rows out of 3.
|239     for(ap_uint<9> j = 0; j < MOTDET_WIDTH; ++j)
|240     {
|241 #pragma HLS PIPELINE
|242         buffer[0][j] = in.read();
|243     }
|244
|245     // Now iterate over the rest of the rows, now each row we get , we can
|246     // output results.
|247     for(ap_uint<9> i = 1; i < MOTDET_HEIGHT; ++i)
|248     {
|249 #pragma HLS PIPELINE
|250         ap_uint<1> val_read = in.read();
|251
|252         if(val_read || buffer[0][j] || buffer[1][j]) out.write(1);
|253         else out.write(0);
|254
|255         buffer[buffer_ptr][j] = val_read;
|256     }
|257
|258     // Now we have iterated the whole image, but we only have outputted
|259     // MOTDET_HEIGHT-1 rows.
|260     // Output those last row extending the image with zeros at the bottom.
|261     for(ap_uint<9> j = 0; j < MOTDET_WIDTH; ++j){
|262 #pragma HLS PIPELINE
|263         if(buffer[0][j] || buffer[1][j]) out.write(1);
|264         else out.write(0);
|265     }
|266
|267 #ifndef __SYNTHESIS__
|268     for(ap_uint<2> i = 0; i < 2; ++i) delete[] buffer[i];
|269     delete[] buffer;
|270 #endif
|271 }
|272
|273 void dilation_hline(hls::stream<ap_uint<1>, MOTDET_STREAMDEPTH> &in, hls::
|274     stream<ap_uint<1>, MOTDET_STREAMDEPTH> &out)
```

7.5. APÉNDICE

```
|275 ap_uint<1> buffer [2];
276
277     for (ap_uint<9> i = 0; i < MOTDET.HEIGHT; ++i)
278     {
279         buffer [0] = 0;
280         buffer [1] = in.read();
281
282         for (ap_uint<9> j = 1; j < MOTDET.WIDTH; ++j)
283         {
284 #pragma HLS PIPELINE
285             ap_uint<1> val_read = in.read();
286
287             if (val_read || buffer [0] || buffer [1]) out.write(1);
288             else out.write(0);
289
290             buffer [j % 2] = val_read;
291         }
292
293         if (buffer [0] || buffer [1]) out.write(1);
294         else out.write(0);
295     }
296 }
297
298     void dilation(hls::stream<ap_uint<1>, MOTDET_STREAM_DEPTH> &in, hls::stream
299     ↪ <ap_uint<1>, MOTDET_STREAM_DEPTH> &out)
300 {
301 #pragma HLS DATAFLOW
302     hls::stream<ap_uint<1>, MOTDET_STREAM_DEPTH> half_dilated;
303
304     // An NxN dilation can be decomposed into 2 1-dimensional kernels , N
305     ↪ vertical and N horizontal.
306     dilation_vline(in, half_dilated);
307     dilation_hline(half_dilated, out);
308 }
309 } // namespace imgutil
309 } // namespace motdet
```

7.5.3.6 contour_detector.hpp

```
1 #ifndef _MOTDET_CONTOUR_DETECTOR_HPP_
2 #define _MOTDET_CONTOUR_DETECTOR_HPP_
3
4 #include <cstdint>
5 #include "hls_stream.h"
6 #include "ap_int.h"
7
8 #include "motion_detector.hpp"
9
10 namespace motdet
11 {
12     namespace imgutil
13     {
14
15         /**
```

7.5. APÉNDICE

```

16     * @brief Detect connected components (contours) in a streamed binary image
17     * @param in Binary image. All values must be either 0 or 1 upon input. It
18     * @param conts The found contour bounding boxes without hierarchy.
19     */
20     void connected_components(hls::stream<ap_uint<1>, MOTDET_STREAM_DEPTH> &in,
21     *                           hls::stream<motdet::Streamed_contour, MOTDET_STREAM_DEPTH> &out);
22 } // namespace imgutil
23 } // namespace motdet
24
25 #endif // _MOTDET_CONTOUR_DETECTOR_HPP_

```

7.5.3.7 contour_detector.cpp

```

1 #include "contour_detector.hpp"
2
3 namespace motdet
4 {
5     namespace imgutil
6     {
7         namespace
8         {
9             class Disjoint_contour_connector
10            {
11                public:
12                    Disjoint_contour_connector() {}
13
14                    uint16_t add_cont(ap_uint<9> point_x, ap_uint<9> point_y)
15                    {
16                        if (conts.contour_count >= MOTDET_MAX_CONTOURS) return 0x3FF;
17
18                        ap_uint<10> new_tag = conts.contour_count++;
19                        conts.contours[new_tag].bb_br_x = point_x;
20                        conts.contours[new_tag].bb_tl_x = point_x;
21                        conts.contours[new_tag].bb_br_y = point_y;
22                        conts.contours[new_tag].bb_tl_y = point_y;
23                        parents[new_tag] = 0x3FF;
24                        return new_tag;
25                    }
26
27                    void update_cont(ap_uint<10> tag, ap_uint<9> point_x, ap_uint<9>
28                                     ↪ point_y)
29                    {
30                        ap_uint<10> repr = get_repr(tag);
31                        if (point_x < conts.contours[repr].bb_tl_x) conts.contours[repr
32                                         ↪ ].bb_tl_x = point_x;
33                        else if (point_x > conts.contours[repr].bb_br_x) conts.contours[repr
34                                         ↪ ].bb_br_x = point_x;
35                        if (point_y < conts.contours[repr].bb_tl_y) conts.contours[repr
36                                         ↪ ].bb_tl_y = point_y;
37                        else if (point_y > conts.contours[repr].bb_br_y) conts.contours[repr
38                                         ↪ ].bb_br_y = point_y;
39                    }
40

```

7.5. APÉNDICE

```
35     uint16_t get_repr(ap_uint<10> c)
36     {
37         ap_uint<10> og_c = c;
38         ap_uint<10> parent = parents[c];
39         while(parent != 0x3FF)
40         {
41             #pragma HLS LOOP_TRIPCOUNT avg=1 max=2 min=0
42             c = parents[c];
43             parent = parents[c];
44         }
45         if(og_c != c) parents[og_c] = c;
46         return c;
47     }
48
49     void merge(ap_uint<10> c0, ap_uint<10> c1)
50     {
51         ap_uint<10> repr_c0 = get_repr(c0);
52         ap_uint<10> repr_c1 = get_repr(c1);
53         if(repr_c0 != repr_c1) parents[repr_c0] = repr_c1;
54         update_cont(repr_c1, conts.contours[repr_c0].bb_br_x, conts.
55             ↪ contours[repr_c0].bb_br_y);
56         update_cont(repr_c1, conts.contours[repr_c0].bb_tl_x, conts.
57             ↪ contours[repr_c0].bb_tl_y);
58     }
59
60     void get_merged_conts(hls::stream<Streamed_contour,
61             ↪ MOTDET_STREAM_DEPTH> &out)
62     {
63         Streamed_contour streamed_cont;
64
65         #pragma HLS LOOP_TRIPCOUNT avg=30 max=1023 min=0
66         #pragma HLS PIPELINE II=2
67         if(parents[k] == 0x3FF)
68         {
69             ap_uint<21> area = hls::abs((conts.contours[k].bb_tl_x
70                 ↪ - conts.contours[k].bb_br_x) * (conts.contours[k].
71                 ↪ .bb_tl_y - conts.contours[k].bb_br_y));
72             if(area >= motdet_min_cont_area)
73             {
74                 streamed_cont.contour = conts.contours[k];
75                 streamed_cont.contour.bb_tl_x *=
76                     ↪ MOTDET_REDUCTION_FACTOR;
77                 streamed_cont.contour.bb_tl_y *=
78                     ↪ MOTDET_REDUCTION_FACTOR;
79                 streamed_cont.contour.bb_br_x *=
80                     ↪ MOTDET_REDUCTION_FACTOR;
81                 streamed_cont.contour.bb_br_y *=
82                     ↪ MOTDET_REDUCTION_FACTOR;
83                 streamed_cont.stream_end = false;
84                 out.write(streamed_cont);
85             }
86         }
87     }
88
89     Streamed_contour streamed_cont;
90     if(streamed_cont.stream_end)
91     {
92         out.write(streamed_cont);
93     }
94 }
```

7.5. APÉNDICE

```
85
86     private :
87         Contour_package  conts;
88         ap_uint<10> parents [MOTDET_MAX_CONTOURS];
89     };
90
91 }
92
93 void connected_components(hls::stream<ap_uint<1>,> MOTDET_STREAM_DEPTH &in ,
94                           hls::stream<motdet::Streamed_contour ,> MOTDET_STREAM_DEPTH &out)
95 {
96
97 // Find the contours in the image, we are using a one pass algorithm so in
98 // some cases it is not possible to know if 2 pixels belong to the same
99 // object.
100 // This why we tag them as different contours and then "merge" them when a
101 // pixel that connects both is discovered.
102 // In essence, we have a graph of blobs with a set of connections (mergers)
103 // , so we can use well known graph theory algorithms here.
104 // Use a disjoint-set data structures, based on a list of trees with
105 // representative nodes.
106
107 #ifndef __SYNTHESIS__
108     ap_uint<10> *buffer = new ap_uint<10>[MOTDET_WIDTH];
109     Disjoint_contour_connector &dcc = *(new Disjoint_contour_connector());
110 #else
111     ap_uint<10> buffer [MOTDET_WIDTH];
112     Disjoint_contour_connector dcc;
113 #endif
114
115 // 0x3FF will be the tag representing a null tag.
116 for(ap_uint<9> k = 0; k < MOTDET_WIDTH; ++k) buffer [k] = 0x3FF;
117
118 ap_uint<10> prev = 0x3FF, top_prev , tag;
119
120 #pragma HLS PIPELINE
121         tag = 0x3FF;
122
123         ap_uint<1> read_val = in.read();
124         if(read_val)
125         {
126             if(prev != 0x3FF){
127                 // Previous pix to the left exists
128                 tag = prev;
129
130                 top_prev = buffer [j];
131                 if(top_prev != 0x3FF)
132                 {
133                     // And top pix also exists , check for merge.
134                     if(top_prev != tag) dcc.merge(top_prev , tag);
135                 }
136             }
137         }
```

7.5. APÉNDICE

```
138     top_prev = buffer[j];
139     if (top_prev != 0x3FF)
140     {
141         // Only the top pix exists.
142         tag = top_prev;
143     }
144     else
145     {
146         // Neither pix exists, this is a new contour.
147         tag = dcc.add_cont(i, j);
148     }
149 }
150 }
151
152     prev = tag;
153     buffer[j] = tag;
154     if (tag != 0x3FF) dcc.update_cont(tag, j, i);
155 }
156 }
157
158     dcc.get_merged_conts(out);
159
160 #ifndef __SYNTHESIS__
161     delete [] buffer;
162     delete &dcc;
163 #endif
164 }
165
166 } // namespace imgutil
167 } // namespace motdet
```

Bibliografía

- [1] “¿Qué es un FPGA?” Disponible en <https://www.xilinx.com/products/silicon-devices/fpga/what-is-an-fpga.html> (Última visita: 2021/08/21), 2021.
- [2] “Trading de alta velocidad con FPGA.” Disponible en <https://www.velvetech.com/blog/fpga-in-high-frequency-trading/> (Última visita: 2021/08/21), 2021.
- [3] Xilinx, “Introducción oficial a HLS.” Disponible en https://www.xilinx.com/support/documentation/sw_manuals/ug998-vivado-intro-fpga-design-hls.pdf (Última visita: 2021/08/21), 2019.
- [4] “CPU, GPU, FPGA y TPU para aprendizaje máquina.” Disponible en <https://inaccel.com/cpu-gpu-fpga-or-tpu-which-one-to-choose-for-my-machine-learning-training/> (Última visita: 2021/08/21), 2018.
- [5] “Historia de las CPUs.” Disponible en https://en.wikipedia.org/wiki/History_of_general-purpose CPUs (Última visita: 2021/08/21), 2021.
- [6] “Usos de FPGAs en el pasado, presente y futuro.” Disponible en <https://cacm.acm.org/magazines/2020/10/247594-the-history-status-and-future-of-fpgas/fulltext> (Última visita: 2021/08/21), 2020.
- [7] “Comparando soluciones GPU y FPGA para procesamiento de imágenes.” Disponible en <https://www.automate.org/blogs/embedded-vision-image-processing-fpgas-vs-gpus> (Última visita: 2021/08/21), 2019.
- [8] “Chips de inteligencia artificial, TPUs.” Disponible en <https://www.electronicdesign.com/industrial-automation/article/21804901/cpus-gpus-and-now-ai-chips> (Última visita: 2021/08/21), 2017.
- [9] Struyf, Lars De Beugher, Stijn Hoon, Dong Van UytSEL, Dong Hoon Kanters, Frans Goedemé, Toon, “The battle of the giants: A case study of GPU vs FPGA optimisation for real-time image processing.” 2014.

- [10] Barbacci, Mario R.; Grout, Steve; Lindstrom, Gary; Maloney, Michael Patrick, “Ada as a hardware description language : An initial report.” 1984.
- [11] “VHDL vs Verilog, ¿Cuál es mejor?.” Disponible en <https://blog.digilentinc.com/battle-over-the-fpga-vhdl-vs-verilog-who-is-the-true-champ/> (Última visita: 2021/08/21), 2021.
- [12] “La evolución de High Level Synthesis.” Disponible en <https://semiengineering.com/the-evolution-of-high-level-synthesis/> (Última visita: 2021/08/21), 2020.
- [13] “Ranking de lenguajes de programación.” Disponible en <https://staticstimes.com/tech/top-computer-languages.php> (Última visita: 2021/08/21), 2021.
- [14] “Casos de uso aptos para HLS.” Disponible en <https://sem wiki.com/eda/2627-what-applications-implement-best-with-high-level-synthesis/> (Última visita: 2021/08/21), 2013.
- [15] “Documentación oficial de VITIS HLS.” Disponible en https://www.xilinx.com/html_docs/xilinx2021_1/vitis_doc (Última visita: 2021/08/21), 2021.
- [16] “Paralelismo a nivel de instrucción.” Disponible en https://en.wikipedia.org/wiki/Instruction_pipelining (Última visita: 2021/08/21), 2021.
- [17] “Uso de AVX para la aceleración de operaciones en CPU.” Disponible en <https://hardzone.es/reportajes/que-es/instrucciones-avx-procesador/> (Última visita: 2021/08/21), 2020.
- [18] M. Qasaimeh, K. Denolf, J. Lo, K. Vissers, J. Zambreno and P. H. Jones, “Comparing Energy Efficiency of CPU, GPU and FPGA Implementations for Vision Kernels.” 2019.
- [19] “El funcionamiento de una GPU.” Disponible en <https://www.extremetech.com/gaming/269335-how-graphics-cards-work> (Última visita: 2021/08/21), 2021.
- [20] “Funcionamiento de GPUs Nvidia y el uso de warps.” Disponible en <https://nyu-cds.github.io/python-gpu/02-cuda/> (Última visita: 2021/08/21), 2017.
- [21] “Optimización de código en FPGA usando HLS.” Disponible en https://www.xilinx.com/html_docs/xilinx2021_1/vitis_doc/

- vitis_hls_optimization_techniques.html (Última visita: 2021/08/21), 2021.
- [22] “RAM de puerto doble.” Disponible en <https://www.maximintegrated.com/en/design/technical-documents/app-notes/6/62.html> (Última visita: 2021/08/21), 2001.
- [23] “¿Qué es C++ moderno?” Disponible en <https://www.modernescpp.com/index.php/what-is-modern-c> (Última visita: 2021/08/21), 2017.
- [24] “Guía de programación en C++ moderno.” Disponible en <https://github.com/isocpp/CppCoreGuidelines/blob/master/CppCoreGuidelines.md> (Última visita: 2021/08/21), 2021.
- [25] “Pasando de C a C++. C++ moderno.” Disponible en <https://docs.microsoft.com/en-us/cpp/cpp/welcome-back-to-cpp-modern-cpp?view=msvc-160> (Última visita: 2021/08/21), 2020.
- [26] “El gran éxito de Raspberry Pi.” Disponible en <https://www.raspberrypi.org/blog/ten-millionth-raspberry-pi-new-kit/> (Última visita: 2021/08/21), 2016.
- [27] “Luma y luminiscencia relativa.” Disponible en [https://en.wikipedia.org/wiki/Luma_\(video\)](https://en.wikipedia.org/wiki/Luma_(video)) (Última visita: 2021/08/21), 2021.
- [28] “Separación de kernels 2D en 2 kernels 1D.” Disponible en <https://bartwronski.com/2020/02/03/separate-your-filters-svd-and-low-rank-approximation-of-image-filters/> (Última visita: 2021/08/21), 2020.
- [29] “Umbralización de imágenes.” Disponible en [https://en.wikipedia.org/wiki/Thresholding_\(image_processing\)](https://en.wikipedia.org/wiki/Thresholding_(image_processing)) (Última visita: 2021/08/21), 2021.
- [30] “Detección de bordes Canny.” Disponible en <https://towardsdatascience.com/canny-edge-detection-step-by-step-in-python-computer-vision-b49c3a2d8123> (Última visita: 2021/08/21), 2019.
- [31] “La importancia de la organización de ficheros en proyectos.” Disponible en <https://www.gamedev.net/articles/programming/general-and-gameplay-programming/organizing-code-files-in-c-and-c-r1798/> (Última visita: 2021/08/21), 2002.
- [32] “La complejidad de Make.” Disponible en <https://nibblestew.blogspot.com/2017/12/a-simple-makefile-is-unicorn.html> (Última visita: 2021/08/21), 2017.

- [33] “CMake moderno.” Disponible en <https://cliutils.gitlab.io/modern-cmake/> (Última visita: 2021/08/21), 2017.
- [34] “Estándares de codificación de C++.” Disponible en <https://isocpp.org/wiki/faq/coding-standards#coding-std-wars> (Última visita: 2021/08/21), 2021.
- [35] “Normas de estilo de las librerías C++ POCO.” Disponible en <https://www.appinf.com/download/CppCodingStyleGuide.pdf> (Última visita: 2021/08/21), 2014.

GNU Free Documentation License

Version 1.3, 3 November 2008

Copyright © 2000, 2001, 2002, 2007, 2008 Free Software Foundation, Inc.
<https://fsf.org/>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document “free” in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of “copyleft”, which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The “Document”, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as “you”. You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A “Modified Version” of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A “Secondary Section” is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document’s overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The “Invariant Sections” are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The “Cover Texts” are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A “Transparent” copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not “Transparent” is called “Opaque”.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The “Title Page” means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, “Title Page” means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

The “publisher” means any person or entity that distributes copies of the Document to the public.

A section “Entitled XYZ” means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as “Acknowledgements”, “Dedications”, “Endorsements”, or “History”.) To “Preserve the Title” of such a section when you modify the Document means that it remains a section “Entitled XYZ” according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document,

thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- A.** Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- B.** List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
- C.** State on the Title page the name of the publisher of the Modified Version, as the publisher.
- D.** Preserve all the copyright notices of the Document.
- E.** Add an appropriate copyright notice for your modifications adjacent to the other copyright notices. **F.** Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below. **G.** Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- H.** Include an unaltered copy of this License.
- I.** Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- J.** Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- K.** For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled “Endorsements”. Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled “Endorsements” or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers. If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version’s license notice. These titles must be distinct from any other section titles.

You may add a section Entitled “Endorsements”, provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled “History” in the various original documents, forming one section Entitled “History”; likewise combine any sections Entitled “Acknowledgements”, and any sections Entitled “Dedications”. You must delete all sections Entitled “Endorsements”.

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an “aggregate” if the copyright resulting from the compilation is not used to limit the legal rights of the compilation’s users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document’s Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled “Acknowledgements”, “Dedications”, or “History”, the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See <https://www.gnu.org/licenses/>.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License “or any later version” applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy’s public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

“Massive Multiauthor Collaboration Site” (or “MMC Site”) means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A “Massive Multiauthor Collaboration” (or “MMC”) contained in the site means any set of copyrightable works thus published on the MMC site.

“CC-BY-SA” means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

“Incorporate” means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is “eligible for relicensing” if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

ADDENDUM: How to use this License for your documents To use this License in a document you have written, include a copy of the License in the document and put

the following copyright and license notices just after the title page:

Copyright (C) YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled “GNU Free Documentation License”. If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the “with . . . Texts.” line with this:

with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST. If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.